Universidad Autónoma de Madrid
Escuela politécnica superior

Máster en Bioinformática y Biología Computacional

Trabajo Fin de Máster

PLASMIDNET. SISTEMA DE INFORMACIÓN DE MÓDULOS FUNCIONALES DE PLÁSMIDOS

Autor: Fernando Freire Gómez
Tutor: David Abia Holgado
Ponente: Gonzalo Martínez Muñoz

Febrero 2020

PLASMIDNET. SISTEMA DE INFORMACIÓN DE MÓDULOS FUNCIONALES DE PLÁSMIDOS

Autor: Fernando Freire Gómez
Tutor: David Abia Holgado
Ponente: Gonzalo Martínez Muñoz

Escuela Politécnica Superior
Universidad Autónoma de Madrid
Febrero 2020

Resumen

Resumen

La publicación en web de los datos obtenidos en proyectos bioinformáticos o la exposición de sus algoritmos o procesos de forma que puedan ser utilizados con fuentes de datos alternativas suministradas por el usuario, requiere el diseño y construcción de sistemas capaces de lidiar con grandes volúmenes de datos, habillitados para configurar, distribuir, ejecutar y monitorear flujos de procesos distribuidos entre varios nodos, y conseguir presentar la información de forma interactiva, numérica y gráficamente, confiriendo significación a los datos y a las relaciones entre las diferentes entidades de información.

El sistema desarrollado afronta estos retos, teniendo en cuenta además las limitaciones de los entornos humanos de investigación, donde los recursos están sobre todo orientados a la investigación en sí misma. Por ello el sistema se ha construido bajo un estricto criterio de automatización máxima, desde el empaquetamiento y distribución de software hasta la generación de la documentación, eludiendo al máximo la instalación de productos software que conllevarían dificultades de administración y actualización, homogeneizando las pautas de diseño y construcción en todos los ámbitos de la aplicación (servidor, navegador), construyendo un conjunto de utilidades para facilitar las pruebas y utilizando los patrones de diseño de software mínimos y necesarios para acometer las diferentes tareas. El sistema se ha construido para soportar las necesidades de los usuarios de la base de datos de módulos funcionales de plásmidos, pero puede ser aplicado fácilmente a cualquier otro proyecto bioinformático.

Palabras Clave

Módulo funcional, plásmido, proteína, redes, superfamilia, PWA, nodejs, service worker, pipeline, docker, lean, literate programming

Abstract

The web publication of the data obtained by bioinformatics projects or the accessibility of their algorithms or processes so that they can be used with alternative data sources provided by the user, requires the design and construction of systems capable of dealing with large volumes of data.

Also these systems must be enabled to configure, distribute, execute and monitorice pipelines of batch processes distributed among several nodes, and should have the ability to draw the information interactively, numerically and graphically, conferring meaning to the data and to the relationships between the different information entities.

The system developed addresses these challenges, also taking into account the human resources limitations in research environments, where the staff deal primarily towards research itself. Therefore, the system has been built under a strict criteria of maximum automation, from the packaging and distribution of software to the generation of documentation, avoiding the installation of software products that would lead to complexities in administration and updating. To achieve this goal we homogenize to a maximum extent the guidelines of design and construction in all areas of the application (server, browser, pipeline scheduling), building a set of utilities to ease testing and using the minimum and necessary software design patterns to undertake the different tasks. The system has been built to support the user needs of the plasmid functional modules database, but it can be easily extended to any other bioinformatic project.

Key words

Functional module, plasmid, protein, networks, superfamily, PWA, nodejs, service worker, pipeline, docker, lean, literate programming

Agradecimientos

Puedo escribir estas líneas gracias a los avances científicos. Si no hubiera sido por aquel antibiótico suministrado a tiempo para curar esa neumonía, por aquellos anticoagulantes que eliminaron los trombos antes de que el daño se convirtiera en irreversible, si no fuera por tantas otras cosas pequeñas y grandes, yo ya no estaría en este mundo.

Por ello, después de que la ciencia me regalara tiempo en forma de supervivencia y después de que mi última empresa me obsequiara con tiempo para dedicarme a lo que más me gustara, me planteé colaborar con departamentos de investigación con el ánimo de aportar algo de mi experiencia profesional en entornos TIC, algo que ayudara en el trabajo diario de las personas dedicadas a la ardua tarea de interpretar la realidad y cambiarla.

Con el fin de lograrlo resultaba imprescindible recibir la formación necesaria para adentrarme en un área gobernada por otra semántica, por otras reglas y fundamentos ajenos en gran medida a los relacionados con mi experiencia previa.

El ser elegido para cursar este máster significó un enorme privilegio que no dudé en aprovechar. Después, cuando el CBM, por iniciativa de David Abia, me propuso construir un sistema web para publicar información funcional de plásmidos, tampoco titubeé en aceptar la encomienda.

Este trabajo es un intento de construir una primera versión de este sistema. He intentado utilizar las mejores estrategias de diseño y construcción que he ido aprendiendo durante mi vida profesional previa, soslayando los errores, desde una época que arranca para mí en los inicios de Internet y las comunicaciones de banda ancha, donde todo estaba por hacer, por inventar.

Pero este trabajo tiene también mucho de invención, de investigación en estrategias que jamás había utilizado ni he visto tan siquiera plantear.

Espero de verdad que con esta dedicación pueda retornar al menos una ínfima parte de lo que he recibido, más allá de que signifique el trámite final para culminar este máster de Bioinformática y Biología Computacional.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

1.1. Motivación del proyecto

El presente trabajo tiene por objetivo construir el sistema informático que dé soporte a la publicación web, desarrollo, pruebas y mantenimiento de la información de módulos funcionales de plásmidos.

Un módulo funcional es un concepto emergente que nos permite organizar de forma estructurada el conocimiento actual sobre la función de un determinado plásmido. De otra forma, la presencia en un plásmido de un determinado módulo sirve para caracterizar la funcionalidad del plásmido. La construcción de un módulo funcional se basa en el establecimiento de una serie de entidades agrupadoras intermedias: familias y superfamilias, en cuyo nivel más bajo encontramos las proteínas y en su nivel más alto los citados módulos.

Los procesos necesarios para la elaboración de esta información han sido desarrollados en el ámbito de [27], pero dentro del alcance de este proyecto se organizarán y automatizarán, actualizándolos cuando sea pertinente a las nuevas versiones de las herramientas utilizadas.

La construcción del sistema informático se regirá por varios principios básicos: homogeneidad, integración continua, mantenibilidad y tolerancia a fallos. Para conseguirlo abrazaremos algunos estándares de desarrollo (devops [21], lean/agile [37]) y otros criterios que propondremos. Y si existe un único concepto que pueda caracterizar nuestros objetivos es este: automatización.

Merece especial atención el concepto de homogeneidad. En un primer momento pensamos en referirnos a él como simplicidad, pero llegamos a la conclusión que este último término resultaría inevitablemente confuso. De hecho, tal vez, el objetivo de simplicidad pueda resultar en exceso ambicioso y hasta equívoco, porque lo que pudiera parecer simple en una fase del proyecto podría no serlo en otra. Cabría pensar que la homogeneidad no es más que uno de los ingredientes de la simplicidad, pero esto no es realmente así. Como veremos, la persecución de la homogeneidad nos llevará a tolerar algunas complejidades que nos podríamos haber ahorrado de otra manera. Por ejemplo, la utilización de un servidor *web* como *Apache*, nos habría ahorrado muchas líneas de código.

Al sistema a construir nos referiremos a él con el nombre **PlasmidNet**, como combinación de plásmido y red funcional y a la vez acercándose al término anglosajón Plasmid, que es nuestra entidad de estudio. También el término hace referencia a la simplicidad y compacidad

estructural de los plásmidos y a la simplicidad de los protocolos de comunicación empleados en su transmisión (simplicidad desde el punto de vista informacional, no biológica).

No obstante no es objetivo de **PlasmidNet** ligarse fuertemente al tipo de problema a solucionar. Deseamos definir unas pautas de implementación que puedan servir de molde para otras necesidades y proyectos similares. Todo ello sin perder la orientación *lean*, a grandes rasgos, minimalista. Cómo puede tal cosa realizarse, manteniendo a la vez generalidad y especificidad, será uno de los aspectos que desarrollaremos en el proyecto. También analizaremos la conveniencia de las estrategias utilizadas, si deberían ser ampliadas y desarrolladas con mayor profundidad o entendemos que hemos transitado por rutas que mejor deben ser recorridas mediante implementaciones más ortodoxas.

1.2. Objetivos y enfoque

Se diseñarán y desarrollarán los siguientes componentes de **PlasmidNet**, que es el sistema completo:

1. Orquestador distribuido de flujos de tareas para la obtención inicial y actualizaciones de la base de datos de módulos funcionales.
2. Orquestador distribuido de flujos de tareas para resolver en segundo plano asíncronamente las peticiones de los usuarios del portal que requieran una respuesta no inmediata.
3. Portal web, que consta de frontal *http(s)* que da acceso a capa de servicios interna (*http*) desarrollada en tecnología *REST-json*.
4. *Web app* HTML5 *responsive* capaz de ejecutarse sobre motores HTML5: navegadores, entornos *nodejs* e incluso *webviews* de *IOS* y *Android*, aunque no es objetivo inicial del proyecto la construcción de *APP* movilizadas como contenedores de la *web app*. Utilizarán como librerías fundamentales *jquery* y *bootstrap*. Probablemente optemos por una *web app* sustentada en una única página *Single Page App*.
5. Base de datos, hospedada probablemente en *SQLite3*, y si no fuera suficiente para cubrir los requerimientos de paralelismo, optaremos por *PostgreSQL*.

Cada elemento de servidor (servidor *web*, pasos de orquestación) será empaquetado, distribuido y monitorizado mediante tecnología *Docker*.

Para que estos componentes puedan cubrir los requerimientos especificados en la introducción, utilizaremos varios criterios de diseño troncales que dirigirán la implementación:

- **Frameworkless** El desarrollo de la *web app* no utilizará ninguna plataforma al uso (*Angular*, *Vue.js*, *React*, ...) , que se basan en la dinamización de páginas mediante la incrustación de código (*javascript*, *java*, *php*, ...) mezclado con *HTML* estático. Esta estrategia también se utiliza en tecnologías de servidor menos recientes, como *Java Server Pages (jsp)* o *Active Script Pages (asp)*. Renunciamos a este tipo de mezclas porque consideramos crucial que el diseño estático o maqueta inicial de la que parten este tipo de desarrollos no se modifique o adapte en ningún momento por el programador de la aplicación. También para permitir una mayor legibilidad del código HTML, legibilidad contra la que se atenta inevitablemente bajo las estrategias anteriores. La idoneidad de utilizar un *framework* de desarrollo suscita muchas dudas [30], aunque la legión de adeptos es mucho más significativa.

- **thinCMS** Los diferentes componentes *web* normalmente son construidos y almacenados sobre un sistema *CMS Content Management System*, de acuerdo a los estándares de facto de la industria actual. Estos sistemas suelen ser complejos e introducen una serie de servidumbres que consideramos es mejor evitar para garantizar la simplicidad y por tanto la mantenibilidad del sistema. Pero sí vamos a plantear un gestor de contenidos ad-hoc, muy sencillo, delgado (*thin*) y se apoyará o bien en un gestor de versiones, como los utilizados para Software (git, fossil, mercurial, ...), o en un modelo de versiones ad-hoc que albergaremos en una base de datos. También el nombre significa para nosotros *THis Is Not a CMS*. Buscamos garantizar estos puntos:

  1. Gestión de versiones de contenidos.
  2. Facilidad para replicar los contenidos entre todos los entornos (maqueta, pruebas, producción). Construiremos un subsistema de empaquetado y distribución.
  3. Mantenibilidad. No se precisan expertos, normalmente caros y escasos, en herramientas muy específicas de mercado (*CMS*). Aquí estamos utilizando productos más genéricos con más base de conocimiento en la comunidad de ingeniería de SW (*CVS*).
  4. Menor riesgo de obsolescencia del producto (*CMS*).

- **Código como contenido** Trataremos como contenido a cualquier componente de código y compartirá el ciclo de vida de las componentes digamos visuales (html, css, imágenes). Por supuesto javascript de navegador, pero también los programas de *backend* independientemente de su tecnología (*javascript*, *perl*, *python*, *tcsh*).

- **Versiones** El sistema debe permitir la coexistencia de versiones, de forma que podamos en algunos nodos ejecutar una versión de la *web app* y en otros otra versión. Muy relacionado e igualmente importante, será posible dirigir los clientes *web* a una nueva versión sin necesidad de reiniciar ningún servicio. El objetivo es facilitar las pruebas sobre cualquier entorno, también el productivo, aunque en este último caso trataremos preferentemente de posibilitar las pruebas en concurrencia con la operación normal de los usuarios.

- **Distribución** Intentaremos unificar los ámbitos de publicación de contenidos (propio de un *CMS*), distribución de software (asociado normalmente a un CVS) y replicación de datos (asociado a una BD [44]). Mostraremos que el problema es esencialmente el mismo para los tres ámbitos.

1.3. Metodología y plan de trabajo

Como primera idea nos propusimos abordar el proyecto mediante las fases convencionales del desarrollo de software: especificación de requisitos, diseño de alto nivel, diseño detallado, construcción, pruebas, implantación. De inmediato comprendimos que las restricciones temporales iban a requerir una aproximación más directa al problema.

Para la especificación de requisitos, ineludible, nos adentramos en las temáticas desarrolladas en [27] con la ayuda inestimable de la autora, a partir de la cual pudimos comprender qué tipo de elementos debían ser desarrollados y por tanto conseguimos habilitarnos para plantear los objetivos de este proyecto en la propuesta que presentamos hace unos meses. Pero, ¿y el resto de fases?. ¿Por qué no abrazar metodologías más recientes como *Agile* o *Extreme Programming*?

No es tan sencillo, para empezar este es un desarrollo abordado por una sola persona, apoyada por el tutor, pero en esencia no existe atisbo alguno de equipo humano que debe dividirse las tareas y coordinarse, amén de toda variedad de perfiles como el *scrum master*, *product owner*, totalmente fuera de lugar en un proyecto hombre orquesta o *full stack*.

Otro problema es que carecíamos de referencia sobre cómo hacer las cosas, en el sentido de que no existe en este entorno de trabajo ninguna metodología establecida ni arquitectura o lenguajes recomendados. Este aspecto nos otorgaba enorme libertad y nos retaba a investigar cómo construir el sistema desde cero, *from scratch*. Nos veíamos abocados a una fase de I+D+i, pero que no debía atascarnos, porque necesitamos un producto informático operativo, no sólo redactar documentación de alto nivel como esta o explorar el estado del arte o comparar alternativas. Por tanto nos encontrábamos ante un proyecto informático perfectamente adecuado al ámbito de un trabajo de fin de master. Obviamente nuestra relativamente dilatada experiencia profesional previa iluminaba nuestro camino, pero no podíamos confiar del todo en ella porque partía de entornos empresariales con más recursos de hardware y humanos, donde no importa que los equipos de trabajo los integren personas de todo tipo de competencias y que se desarrollen versiones de las aplicaciones para un amplio abanico de entornos: navegadores *web* variopintos, *IOS* y *Android*. Aunque sí podíamos confiar en todas aquellas carencias con las que hemos convivido y en todos los errores cometidos en el pasado. En un entorno con recursos tan limitados como este, ¿podríamos ser capaces de soslayar alguno de las dificultades vividas?. Veremos.

Debíamos movernos rápido en la toma de decisiones relacionadas con cada aspecto de desarrollo. Porque son muchas las opciones que están disponibles para solucionar cada detalle, cientos de lenguajes de programación, decenas de servidores web, diferentes entornos de desarrollo *IDE*, varios gestores de versiones (*git*, *mercurial*), etc...

¿Cómo desplazarnos a través de las listas interminables de posibilidades? Fue necesario introducir el concepto de **decisión guiada por restricciones** o ligaduras, que utilizaremos a lo largo de este trabajo. Una restricción es todo aspecto de implementación ineludible o justo lo contrario, una línea de trabajo que queremos evitar. Algo asimilable a un requisito pero de índole más comprometida, irrenunciable. Lo que hicimos fue un desplazamiento del problema, planteando primero las restricciones y sus prioridades y después eligiendo la opción que mejor se adapta a las mismas. Nos encontramos con que las restricciones en algunos casos podían también requerir una decisión, pero finalmente conseguíamos reducir adecuadamente la cardinalidad de los conjuntos de posibilidades.

Antes de marcar las restricciones, dividimos el problema en ámbitos, en aquellos subsistemas que deben ser construidos. En primera instancia distinguimos cinco ámbitos que representamos en la figura 1.1.

Y el primera decisión que debíamos afrontar es la arquitectural. Comenzando por el servidor *web*, después base de datos, aplicación cliente y orquestación.

Sin duda alguna, la necesidad de que la aplicación funcionara sobre plataforma *web* nos llevaba inevitablemente al uso de tecnologías *HTML5* formadas por componentes *javascript*, *css* y *html*. Es cierto que existen desarrollos incipientes que prometen la ejecución de casi cualquier lenguaje de programación en el navegador (*WebAssembly* [29]), pero todavía no pueden considerarse suficientemente maduros.

Por tanto, utilizar en el servidor cualquier otro lenguaje de programación distinto a javascript (*java*, *php*, ...) habría atentado contra el principio de homogeneidad.

La confrontación es incluso más seria de lo que pudiera parecer. El prefijo *java* del nombre del lenguaje javascript parece sugerir amplias similitudes entre éste y el lenguaje java, el preferido hoy en día para la programación en servidor. Nada más lejos de la realidad. El asincronismo en las operaciones de entrada y salida de Javascript pueden confundir al más experto programación de Java, totalmente síncrono, salvo que explícitamente se desarrolle en base a hilos.

El entorno **nodejs** que implementa el motor *javascript V8* de *Chrome* fuera del navegador nos ha servido para desarrollar en puro *javascript* la parte de servidor. Al menos todo lo relacionado

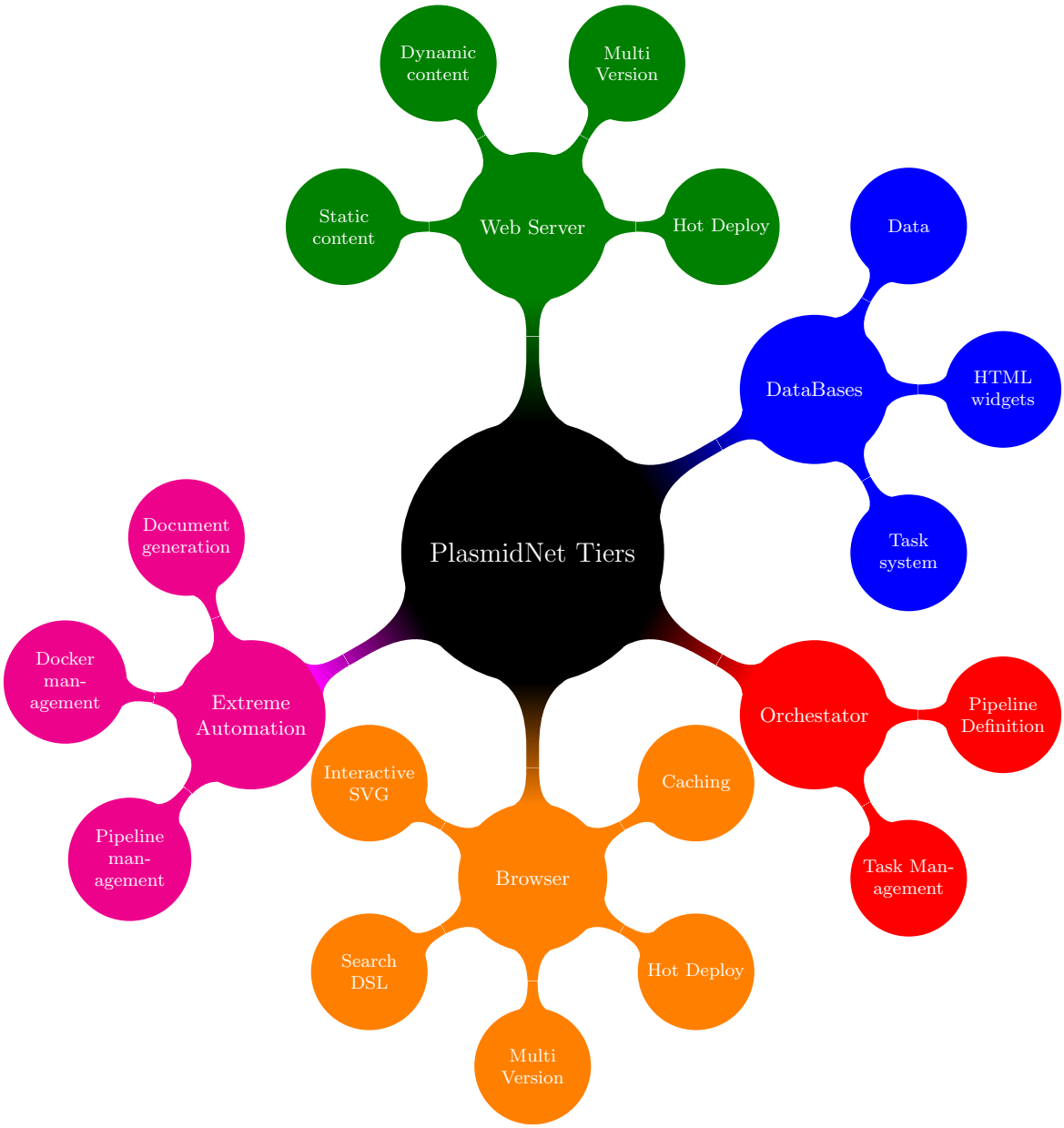

Figura 1.1: Mapa mental del sistema PlasmidNet

con el servidor *web* y la orquestación de tareas. Este planteamiento no sólo nos evita en el futuro la necesidad de contar con diversos perfiles de ingenieros de software, también nos evita introducir en el sistema dos elementos nuevos que hay que conocer, instalar, configurar y mantener: el servidor web, y el servidor de aplicaciones *J2EE* (imprescindible en el caso de *java*, aunque existen productos que unifican los dos tipos de servidores *web* y de aplicaciones en un único entorno). Como contrapartida, *nodejs* nos obliga a codificar muchos aspectos del servidor. Lo que en los productos del tipo mencionado es configuración, aquí es programación. Pero veremos que existen también ventajas asociadas a que esto sea precisamente así.

Iremos mostrando sobre la marcha algunas otras restricciones aplicadas como que **la maqueta *web* no se modifica** cuando se incorpora a la aplicación o **la documentación acompaña al código** o **el lenguaje de consulta de datos debe basarse en un lenguaje específico de dominio (DSL)** no en un formulario, **la aplicación debe funcionar sobre sistemas Linux CENTOS**, o que **nos aseguramos que las versiones de las librerías utilizadas son las**

que se han probado y no se descargan dinámica e incontroladamente desde los proveedores.

Somos conscientes de que algunas de las restricciones aplicadas son discutibles. Nosotros mismos haremos notar, donde sea necesario, los problemas que tenemos, hemos tenido o prevemos tener en la convivencia con la restricción discutida.

Otro aspecto que no queremos olvidar de mencionar aquí es el de la reorganización (*refactoring*) expresado por la restricción **la reorganización del código debe ser fácil de realizar**. Con fácil nos referimos a rápida, cómoda, incluso segura. Porque estamos convencidos desde el inicio del proyecto en que vamos a necesitar cambiar una y otra vez el código, simplificándolo o complicándolo, redistribuyéndolo en los diferentes módulos, borrando incluso fragmentos relativamente grandes e introduciendo otros posiblemente sustanciosos. Necesitamos reorganizar porque estamos investigando soluciones, en muchos casos inventando soluciones. Y aquí entroncamos directamente con el concepto de **integración continua**, muy relacionado con **devops** (ver glosario). Existe una amplia literatura que explica de una forma u otra estos conceptos pero nosotros nos quedamos con una de sus características comunes: **automatización extrema**, o en forma de una restricción: **cualquier tarea relacionada con la aplicación debe de estar automatizada**, desde la generación de los paquetes para la distribución de software en la aplicación *web*, hasta la generación de los diferentes formatos de salida de la documentación. Por automatizada queremos decir que involucrará el menor número de pasos manuales posible. Es una definición un poco laxa, pero en esencia nos planteamos no escatimar ningún esfuerzo en la automatización de tareas repetitivas. Esto puede parecer obvio, y emitir automáticamente una admiración del estilo ¡cómo va a ser de otra manera!. Pues sí, aunque parezca increíble es muy fácil cometer pecado capital de pereza y acostumbrarse a hacer las cosas una y otra vez manualmente porque consideramos que el coste de automatización no vale la pena, casi siempre erróneamente. Incluso en organizaciones bien establecidas, con procedimientos perfectamente definidos y adheridos a unas u otras normativas que impulsan la excelencia operativa, este aspecto termina siendo secundario y no se somete a la debida vigilancia y seguimiento dentro de los equipos de trabajo. Este es un proyecto individual, hecho que puede o no facilitar la consecución de este objetivo, pero sí tenemos claro que si pensamos que no es necesario, estamos equivocados.

Y no podemos dar por finalizado este punto sin referirnos a los conceptos **lean** que nos guiarán en todo este proceso. En [37], la autora repasa las líneas maestras básicas de esta filosofía de trabajo rectora de algunos sistemas productivos, no sólo los informáticos. De hecho nace en las plantas de producción automovilística de Toyota. Y digo "guiar" porque evidentemente no hemos aplicado la restricción básica de lean: **producir sólo lo que se demanda y cuando el cliente lo solicita**. Bueno, tal vez aquí es más difícil porque no está plenamente precisado el concepto de cliente y de producto a desarrollar. Casi nunca lo está por otra parte, pero este proyecto tiene mucho de invención e investigación. La mera adopción de lean es una adaptación particular basada sólo en los siguientes principios, que están íntimamente relacionados con lo que hemos comentado previamente:

1. No desarrollar nada que creamos no aporta valor al departamento. El valor lo entendemos como la consecución de una aplicación *web* capacitada para publicar y dar sentido a la información funcional de plásmidos, documentada, mantenible y desplegable, con los mínimos componentes que exijan instalación, configuración o mantenimientos específicos.

2. Creación de flujo: hacer que todos los procesos de la aplicación fluyan automáticamente, reduciendo el *time to market*, la disponibilidad de una nueva versión.

3. Mejora continua. En cuanto vemos que alguno de los pasos puede ser mejorado por ejemplo con una mayor automatización o una generalización que robustezca el caso particular,

abordarlo sin contemplaciones. La capacidad de rápida reorganización es una capacidad imprescindible para abordar esta necesidad.

Generalización. Este concepto merece que nos detengamos en él. Si bien *lean* nos impele a desarrollar estrictamente lo requerido, hemos observado muchas veces a lo largo de nuestra experiencia profesional, y en este proyecto en varias ocasiones, que el abordar de una forma más general y por tanto más funcional alguno de los requerimientos, ha conllevado una implementación más adecuada del caso particular, el requerido, dotándolo incluso de mayor simplicidad, y por tanto comprensibilidad y mantenibilidad. Digamos que es buena práctica someter a una implementación deliberadamente limitada, al siguiente interrogante ¿y si después quisiéramos generalizar?, ¿sería fácil?. A veces el simple planteamiento nos lleva a reorganizar la solución, sin llegar a generalizarla y por tanto siendo fieles todavía a una filosofía económica. Otras veces la generalización nos ha permitido precisamente encontrar una solución de un problema concreto que se nos resistía. Estos aspectos nos han sorprendido y los comentaremos en cada caso. Para no dejar esta disertación sin ejemplos, diremos que la generalización multi-idioma de la generación documental nos permitió eliminar funcionalidades innecesarias del sistema de las que no fuimos conscientes hasta que abordamos esta generalización.

En definitiva, resultó crucial afrontar el desarrollo de una forma más directa. No podíamos abordar primero un diseño de alto nivel porque no teníamos clara cuál sería la arquitectura final, ya que esta surgiría de nuestras investigaciones y verificaciones, reorganizando las veces que fueran necesarias y por tanto incumpliendo cualquier documento inicial que por tanto debía ser modificado, generando desperdicio, atentando contra la filosofía *lean*. Por tanto, el planteamiento fue el contrario: primero programamos y después documentamos. No es preocupante, pasa en los casos más ortodoxos. La alternativa: admitir documentación desactualizada desde el primer momento. En seguida nos dimos cuenta que estábamos siguiendo una metodología que sólo habíamos utilizado antaño en la construcción de prototipos, y que no dudamos en bautizar como **Programación Exploratoria**, o en inglés **Exploratory Programming**. ¿Existiría ya esta metodología?. Pues sí, resulta muy difícil ser original en estos tiempos [32]. Sin embargo quizá podamos orlarla en base a la experiencia adquirida durante el desarrollo del proyecto.

Para finalizar este punto y la introducción, estos son los conceptos clave en nuestra metodología y plan de trabajo, sin que el orden tenga significado:

1. Programación exploratoria.
2. Ajuste a las necesidades.
3. Automatización extrema.
4. Homogeneidad.

2
Navegador

2.1. Planteamiento

Nos enfrentábamos como siempre a la compleja decisión de escoger entre múltiples alternativas de implementación y en muchos niveles.

A nivel de **estructura del código** es práctica común utilizar un marco de trabajo (*framework*) reconocido como *Angular* [1] o emplear *javascript* 'nativo', o apoyado en alguna librería de interacción con el *DOM* como *jQuery* [31]. También es habitual recurrir a *requirejs* [12] para contar con un sistema de importación de módulos en el navegador tan estructurado como puede estarlo en el servidor.

Como mínimo iba a resultar imprescindible la interacción fácil com los objetos de la página (*DOM*), siendo *jQuery* la alternativa más extendida, pero el resto de productos ¿es verdaderamente necesario?.

Uno de los principales problemas históricos de el código *javascript* inmerso en el navegador estaba relacionado con la descarga. Si fraccionabas el código en muchos pequeños ficheros (módulos), la descarga podía ser más rápida (al ejecutarse en paralelo, bajo el hace tiempo ubicuo protocolo *HTTP 1.1*), pero cada incorporación de un nuevo módulo obliga a modificar también el *HTML* de la página para incluirlo. Si para evitar esto se concentra todo el código en unos pocos ficheros que contienen funciones de contenido similar, a medio o largo plazo los ficheros dejaban de ser modulares: su heterogeneidad iba creciendo.

En realidad la solución al problema se basa en la renuncia a albergar en el navegador la misma estructura de código de los repositorios de desarrollo, agregando los diferentes módulos por criterios de distribución, no por criterios funcionales. La implementación fue facilitada por la aparición de herramientas concatenadoras de código, *minifiers* (que eliminan los saltos de línea y los comentarios) y *uglifiers* que además alteran los nombres de variables y funciones ofuscándolo.

Decidimos que con estas herramientas era suficiente ya que en principio podíamos estructurar cómodamente en servidor y descargar incluso un único fichero javascript compactado. Además el entorno de tareas *gulpjs* [7] nos ofrecía complementos (*plugins*) para automatizar fácilmente estas tareas.

Solventado este problema realmente no necesitábamos complicarnos con *frameworks*. Hay que tener en cuenta que estos entornos facilitan cosas pero introducen servidumbres como aprender de inicio toda su semántica y procedimientos, a veces del mismo orden de dedicación que aprender *javascript*, y sobre todo estar muy ligado al versionado de este tipo de aplicaciones, que pueden requerir reorganizaciones de envergadura entre una versión y otra, es cierto que no siempre, pero ha ocurrido, por ejemplo en *Angular* [40]. Queremos también mantener un control absoluto de la implementación, sin limitaciones estilísticas impuestas por estos entornos.

El control nos ayudará a afrontar esta lista de criterios de implementación:

1. La visualización de la aplicación debe adaptarse a dispositivos móviles y a los diferentes tamaños de pantalla.
2. Debe funcionar sobre una única página (*SPA*) para evitarnos la gestión de la frecuentemente incontrolable navegación que provee el navegador. Por otra parte, ¿para qué queremos varias páginas, si estamos implementando una aplicación, no un documento?.
3. Debemos tener control total del caché en el navegador, fuente de históricos problemas con los usuarios, que dependiendo de su configuración pueden estar accediendo a una versión de la aplicación obsoleta o incluso incompatible con la versión de otras partes de la aplicación (otros programas *javascript* que hayan sido afectados de distinta forma por el sistema de caché.
4. Queremos que la aplicación sea operativa incluso sin conexión con el servidor, a partir de los datos almacenados en caché, y que por tanto, en dispositivos móviles pueda comportarse como una aplicación nativa.
5. No buscamos compatibilidad con todos los navegadores ni con todas las versiones históricas de los mismos. Vamos a utilizar versiones avanzadas de *javascript* (relativamente recientes) y modernas capacidades de los navegadores (*service workers*). Tomaremos de referencia el navegador más utilizado (*Chrome*) y revisaremos si también funciona en *Safari* y *Firefox*, o se adapta de forma sencilla a los mismos. No tiene sentido dentro de la filosofía *lean*, partiendo que los usuarios de la aplicación son usuarios avanzados que o bien ya cuentan con una versión moderna del navegador o tienen posibilidad de instalársela.
6. Carga dinámica de módulos *javascript* o de insertos *html* y *css*. Importante durante el desarrollo para evitar molestos reinicios del navegador e interesante igualmente en producción para no entorpecer el trabajo de los usuarios.
7. En dispositivos móviles, de funcionamiento indistinguible del de una *app* nativa.
8. Multiversión. Será posible que sobre el mismo entorno servidor un usuario opere con una versión y otro con otra. Esto facilitará las pruebas post-implantación, mediante la 'publicación' de la nueva versión de la aplicación para los desarrolladores mientras los clientes operan todavía sobre la antigua. Esto requiere también, como veremos, adaptaciones en el entorno de servidor.

No vamos a utilizar *frameworks* ni ningún sistema de plantillas con código javascript o similar inserto en el html (la mayoría de los *frameworks* de hecho incorporan uno propio). Utilizaremos un sistema de plantillas propio que definiremos en el ámbito del subsistema **thinCMS**. Las plantillas se construirán simplemente añadiendo unos tag ad-hoc semánticos cuya interpretación permitirá al código cliente del navegador fundir las plantillas con los datos.

Normalmente las plantillas incluyen referencias al modelo de datos y sentencias de control de flujo programático. El código *html* se vuelve en cierto modo ilegible y resulta imposible no modificar el *html* que se genera con todo el cuidado de los diseñadores cuando se construye la maqueta.

¿Estamos renunciando también al ubicuo modelo vista controlador (*MVC*)?. Rotundamente no, sobre todo porque la realidad es que es imposible sustraerse a él. Evidentemente contamos

siempre con una información (modelo) que debe ser adaptada para su presentación. Este formato está definido en la vista, que es aplicada al modelo por el controlador. La correspondecia entre modelos y vistas puede ser *n* a *n*. Lo que hemos hecho nosotros es que nuestras vistas, que denominaremos **artefactos injertables** (*widgets* en el código fuente), no guardan referencia alguna al modelo. Las hemos simplificado al máximo. A cambio el controlador incorpora toda la lógica para alimentar las vistas con los modelos, sin la ayuda de vistas inteligentes. En una frase, lo que en otras estrategias es una mezcla de configuración y programación, aquí es sólo programación.

Pero no todo lo que parece configuración en otras estrategias es configuración realmente. Y tiene un coste: la modificación de la maqueta y por tanto la pérdida de compatibilidad con la misma.

Esta es una vista *Angular*:

```
<h1>{{title}}</h1>
<h2>My favorite hero is: {{myHero}}</h2>
<p>Heroes:</p>
<ul>
  <li *ngFor="let hero of heroes">
    {{ hero }}
  </li>
</ul>
```

Y esto un fragmento de una página activa *jsp*.

```
...
<body>
  <h3>Choose an author:</h3>
  <form method="get">
    <input type="checkbox" name="author" value="Tan Ah Teck">Tan
    <input type="checkbox" name="author" value="Mohd Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <input type="submit" value="Query">
  </form>

  <%
  String[] authors = request.getParameterValues("author");
  if (authors != null) {
  %>
    <h3>You have selected author(s):</h3>
    <ul>
  <%
      for (int i = 0; i < authors.length; ++i) {
  %>
        <li><%= authors[i] %></li>
  <%
      }
  %>
    </ul>
    <a href="<%= request.getRequestURI() %>">BACK</a>
  <%
```

```
  }
  %>
</body>
...
```

El concepto es similar: *html* con código insertado cuya interpretación por el controlador genera nuevo código *html*. En el caso de *Angular* el controlador realiza la interpretación en el navegador, y en el caso de *jsp* la operación tiene lugar en el servidor.

Nosotros lo haremos en el navegador, pero como la tecnología es homogénea con el nodo servidor, el sistema está capacitado también para implementarlo en ambos nodos.

Estas ideas aportan indudablemente ventajas, pero ¿sirven para todas las situaciones?. En absoluto. Nosotros hemos utilizado dos librerías *javascript*, una para las presentaciones de tablas de datos y otra para la presentación de gráficos interactivos. Ninguna de las dos se ajustan a los modelos de plantillas de ninguno de los frameworks analizados. Esto es así porque en sí mismas estas librerías incluyen su propia estrategia de transformación de los datos para su presentación. Y si no se ajusta a una parte importante de la *web app*, ¿para qué utilizarlo?.

Porque, en definitiva, ¿en qué consiste una aplicación *web*?, o en general una aplicación.

Si nos situamos delante de cualquier aplicación convencional intentando el milagro de despojarnos de cualquier tecnicismo y conocimiento previo, observamos una pantalla donde se van dibujando cosas, datos en tablas, datos en gráficos, datos sueltos, imágenes sin datos, etc... A veces se dibujan unas cosas y a veces otras. Depende de eventos: de que pulsemos tal o cual tecla o verbalicemos un comando de voz, el propio arranque de la aplicación es un evento en sí mismo que encadena una serie de visualizaciones. O sea, la aplicación va presentando varios aspectos en pantalla, va pasando por diversos estados. En cada estado hay cosas que se muestra y cosas que no y cosas que pueden hacerse o cosas que ya no se pueden hacer porque ha desaparecido el botón. Observamos que ante un determinado evento, la aplicación activa o desactiva partes que se corresponden con zonas de la pantalla o a la pantalla entera. Es posible que cada una de esas partes aporte no sólo su visualización sino también su capacidades sensitivas para responder a determinados eventos y las funciones que debe ejecutar en cada caso. Las aporta o contiene suficientes indicaciones para que la aplicación las implemente.

Podemos entender que la aplicación está dotada de un cerebro, el controlador, que actúa como una máquina de estados y que a modo de director de orquesta va dando paso a unos u otros componentes, reprogramándolos si es necesario. A veces sólo necesitará ocultarlos o mostrarlos, otras veces necesitará crearlos y activar o programar su respuesta a eventos.

La vistas y los modelos subyacen a todo esto pero en realidad, en nuestra opinión, no reside en el modelo *MVC* el núcleo del funcionamiento de una aplicación, sino en esta sucesión de cambios de estado provocados por eventos que generan a su vez nuevos eventos y cambios de estado y nuevas visualizaciones. Y todo son datos, las vistas son datos, los modelos son datos. El controlador se alimenta de ellos.

Estos componentes son los artefactos injertables a los que hacíamos mención unos párrafos más arriba. En el manual de programación puede apreciarse que no tienen una correspondencia localizada en la implementación en el sentido de estar contenidos en clases o módulos ad hoc. Pertenecen a un plano conceptual que subyacerá en todo el código. Se puede estructurar de otra manera más convencional, pero nosotros nos hemos focalizado en el controlador.

2.2. Carga dinámica de módulos javascript

Desde hace pocos años está disponible el método **import** para cargar dinámicamente ficheros javascript desde el javascript de la página, el que se carga inicialmente a la vez que el código *html* y las hojas de estilos, el *javascript* convencional.

Existen ejemplos de uso en [5] y [19] y lo utilizan internamente utilidades como *webpack*.

Una utilización exhaustiva sería la siguiente:

1) Cargar estáticamente el *javascript* necesario de la página.
2) Cargar bajo demanda, en cuanto se vayan a utilizar, el resto de ficheros *javascript*.

Y en cuanto al almacenamiento en caché de la información existen diversas opciones:

1) Abandonar la gestión del caché en manos de los elementos de red http (navegadores, *proxys*), es decir, basado en expiraciones, o en las políticas definidas por el usuario en su navegador, fuera del control de la aplicación
2) Gestionarlo internamente mediante almacenamiento local en el navegador.

a) Comprobando periódicamente la existencia de nuevas versiones de los módulos, descargándolos si se detecta un cambio de versión.
b) Advirtiendo al *service worker* mediante mensajes tipo *push*.

Un aspecto crucial a tener en cuenta es que el módulo es un ente aislado: sólo son visibles las funciones y variables exportadas por él y sólo puede acceder a las funciones de otros módulos si se importan explícitamente. Tampoco tiene acceso a funciones del javascript declaradas en los estáticos. Parece por tanto que en cualquier estrategia vamos a necesitar una buena pila de código relacionado con importaciones y exportaciones. Los módulos sólo tienen acceso a los objetos definidos como variables globales a nivel de página (objeto *window*).

Nosotros hemos asumido una estrategia original, muy sencilla, facilitada por los siguientes aspectos y criterios:

1) La aplicación funciona en una sola página. Por tanto todas las variables definidas a nivel de ventana (objeto *window*) son accesibles desde toda la aplicación, también desde cualquier módulo dinámico.

2) No consideramos necesario dinamizar todo, sólo aquellas funciones o módulos complejos más susceptibles a cambiar durante el desarrollo. Normalmente los módulos más complejos técnicamente. Por ejemplo, los que construyen los gráficos interactivos o los paneles de datos.

3) No consideramos necesaria una verificación exhaustiva de la existencia de nuevas versiones (*polling*) ni la técnica de avisar a los navegadores mediante *push* desde el servidor.

Teniendo todo esto en cuenta hemos optado por la siguiente solución:

1) En la carga inicial de la página se carga todo el javascript completo, también el de los módulos. Esto permite que aunque en momentos posteriores falle la carga de módulos, la aplicación siga funcionando con la carga convencional estática. Esto no genera dos versiones de los módulos como pudiera pensarse, una la que se incorpora al aglomerado estático y otra la que se carga dinámicamente.

2) Las funciones de los módulos que se van a dinamizar se referenciarán en un objeto de ventana *window.dyn*. Esto ha de hacerse también para aquellas funciones de cualquier otro módulo o componente estático que vayan a ser utilizadas por los módulos.

3) El caché de módulos se maneja en el *service worker* al igual que el caché de estáticos.

4) Los módulos se cargan almacenando además su versión (*service worker*) que se comparará con la del servidor. Esta comparación se realizará en cada momento en que se necesiten descargar, pero en segundo plano. Si se detecta una versión disponible, se marca el módulo como descargable (*service worker*).

5) En el momento en que se necesita descargar, se verifica si el módulo está marcado para descarga y se procede, en segundo plano, a la misma. En la ejecución inicial del módulo se referencian las nuevas funciones en el objeto *dyn*.

6) Como "momento en que se necesita descargar" entendemos aquel lugar en el código a partir del cual se va a necesitar la funcionalidad prevista por el módulo. Es decir, lanzamos el intento de comprobación diferida y descarga diferida cuando vayamos a utilizar el código (aunque en realidad el nuevo código necesite otra interacción con la funcionalidad para ser ejecutado). Por otra parte la ubicación es a criterio del programador, no existe una razón especial para hacerlo en uno u otro momento. Con eso nos evitamos desarrollar un mecanismo de *polling* o *pushing*. Esta estrategia de diseño, del estilo "ya que paso por aquí, compruebo y descargo", la utilizamos en varios lugares de la aplicación y le hemos denominado *inpassing*.

2.2.1. Circuito

Veamos el flujo de la información:

En *plasmidnet.js* definimos el objeto *dyn* y la importación de esta forma:

```
var dyn = {}; // Dynamic functions or functions used by dynamic modules
...
function load_module(module) {
  ...
  import('/plas/service/module/' + module + '?mtime=' + Date.now())
  ...
}
```

No existe ninguna declaración de funciones ni ningún código prolijo al respecto, porque todo se hace en *plasmidnet_datatables.js* donde apuntamos desde *dyn* las a las funciones a exportar:

```
...
dyn.datatable = datatable;
dyn.get_data = get_data;
```

También aquí necesitamos apuntar a la función externa *loki_get_info* de *plasmidnet_loki.js*, así:

```
...
function get_data(search_service, query, bypass_cache) {
```

```
  let info_object = dyn.loki_get_info(search_service, query);
...
```

Y en *plasmidnet_loki.js* declaramos en *dyn* la función:

```
...
dyn.loki_get_info = loki_get_info;
...
```

El punto de comprobación y descarga en segundo plano no es otro que la función *generate_inner_widget_structure*, justo en el momento de necesitar el módulo para generar el contenido de los injertables. Como en la variable *search_service* nos llega el nombre de la función, esta se referencia simplemente como *dyn[search_service]*.

```
...
function generate_inner_widget_structure(pn_ref, new_seq_ref, query,
    ↪ search_service, data) {
  ...
  } else if (search_service === "force_network") {
    load_module('plasmidnet_graphs.js');
  } else if (search_service === "datatable") {
    load_module('plasmidnet_datatables.js');
  }
  dyn[search_service](data.data, pn_ref, new_seq_ref, query);
}
...
```

El flujo de registros de funciones en *dyn* es compatible totalmente con la versión inicial estática de todo el javascript. Lo que pasa cuando se carga dinámicamente un módulo es una vez compiladas y cargadas en memoria sus funciones, los punteros a las mismas son actualizados en el objeto dyn, donde sustituyen a los punteros de la anterior versión.

Y en teoría, como las versiones antiguas dejan de estar referenciadas, el recolector de basura del motor de javascript del navegador las termina eliminando. De hecho en los perfilados realizados hasta este momento, con todas las caché desactivadas no se aprecia ningún incremento de memoria no liberada posteriormente.

La elección entre caché o descarga se realiza en el *service worker* de la misma forma en que se gestionan los estáticos.

2.3. Artefactos injertables

Los injertables o *widgets* son trozos de código *html* que se obtienen por deconstrucción de la maqueta inicial de la *web app*.

Representan zonas concretas de la página que van a ser modificadas dinámicamente por el programa en respuesta a determinados eventos y utilizando la información relacionada en la base de datos.

Tipos

Existen tres tipos de artefactos injertables:

1) **Gancho** o estructural. Marcan el lugar de la página donde sus injertables asociados deben insertarse. No muestran ninguna información en el navegador.
2) **Molde** de clonado. Aportan la plantilla a partir de la cual se generarán los injertables asociados. No alteran tampoco la visualización de la aplicación.
3) **Semántico**. Son los injertables informativos, los que imponen la estructura de visualización a sus datos asociados.

Acciones

Los injertables se incluirán, modificarán o borrarán dependiendo del estado de la aplicación, estado que cambia en respuesta a determinados eventos. Las modificaciones de estos artefactos se realizan en tres pasos:

1) Recuperar desde la página, desde la caché de injertables o desde el servidor, el código *html* del injertable.
2) Alterar su *html* de acuerdo a los valores de sus datos asociados.
3) Insertar en el *DOM* el *html* producido. La inserción se efectúa habitualmente en el lugar indicado por un tipo de injertables que llamamos gancho(*hook*).

Etiquetas

Hemos definido un conjunto de etiquetas (*tags html*) cuyas funciones son:

1) Identificar los injertables, otorgándoles un número de referencia **pn-ref**.
2) Asociar el estado de la aplicación bajo el cual que los injertables se muestran al usuario **pn-state**.
3) Identificar las etiquetas del injertable que deben ser modificados en el proceso de clonado, mediante la concatenación de un sufijo incremental. Se posicionan en el *html* justo delante de la etiqueta que debe ser modificada.

Cuadro 2.1: Etiquetas de dinamización(I) utilizadas en los injertables.

etiqueta	significado
pn-ref	referencia
pn-state	estado
pn-hook	indicador del lugar de inserción
pn-multi	indicador de clon, puede ser un clon o el modelo inicial
pn-id	indica que la etiqueta *id* que le sucede debe incrementarse durante el clonado
pn-title	indica que el título ha de ser incrementado al clonar

Cuadro 2.2: Etiquetas de dinamización(II) utilizadas en los injertables.

etiqueta	significado
pn-cmod	indica que el valor de la etiqueta que le sucede debe ser incrementado al clonar
pn-action	nombre de la acción a realizar cuando se produce el evento onclick
pn-action-ref	referencia del injertable afectado por la acción
pn-transition	indica que el evento onclick del elemento originará un cambio de estado
pn-state-transition	identifica el nuevo estado
pn-option	identifica que el área del injertable se muestra opcionalmente

Eventos

1) Evento **onReady**, en la carga de la página. Sin acción del usuario.
2) Evento **onClick** disparado desde diversos injertables. Bajo acción del usuario.

Ambos tipos provocan cualquiera de las acciones descritas más arriba. Tenemos un espectro tan limitado porque gran parte de la complejidad de navegación es gestionada por los paquetes *javascript* **DataTables** y **d3js**.

Cambios de estado

Para cada estado las operaciones que se desencadenan se muestran en la figura 2.1

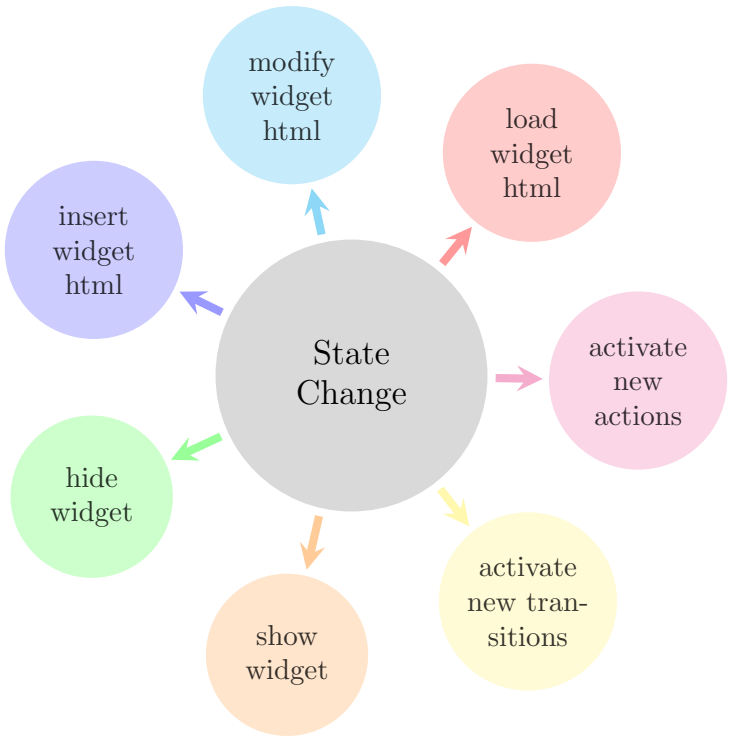

Figura 2.1: Cambio de estado

2.4. Base de datos local

Con el fin de albergar la base de datos local en el navegador hemos decidido utilizar la base de datos noSQL **LokiJS** [36].

Se trata de una base de datos en memoria dotada de persistencia a disco. La custodia de la información en el disco se produce a intervalos de tiempo regulares que se configuran en arranque.

Hemos utilizado esta base de datos para almacenar dos tipos de objetos, cada uno almacenado en una colección de *LokiJS*.

1) Datos procedentes de las consultas.
2) Artefactos injertables.

Esta base de datos nos permite implementar los procedimientos de caché de injertables y datos y por tanto hacer posible el modo de funcionamiento desconectado.

Implementa varios adaptadores de acceso a disco, uno de ellos , el más eficiente y recomendado por el autor *LokiIndexedAdapter* no funciona en la versión actual de *Firefox* (*v73*) y aunque no es nuestro objetivo garantizar la compatibilidad con todos los navegadores, hemos decidido renunciar a esta opción, porque por otra parte observamos un correcto rendimiento. Utilizaremos entonces el adaptador por defecto suministrado por *Lokijs*.

En la imagen 2.2 se muestran los datos tal y como se almacenan en local en *LokiJS* y tal y cómo se informan a la librería *DataTables*.

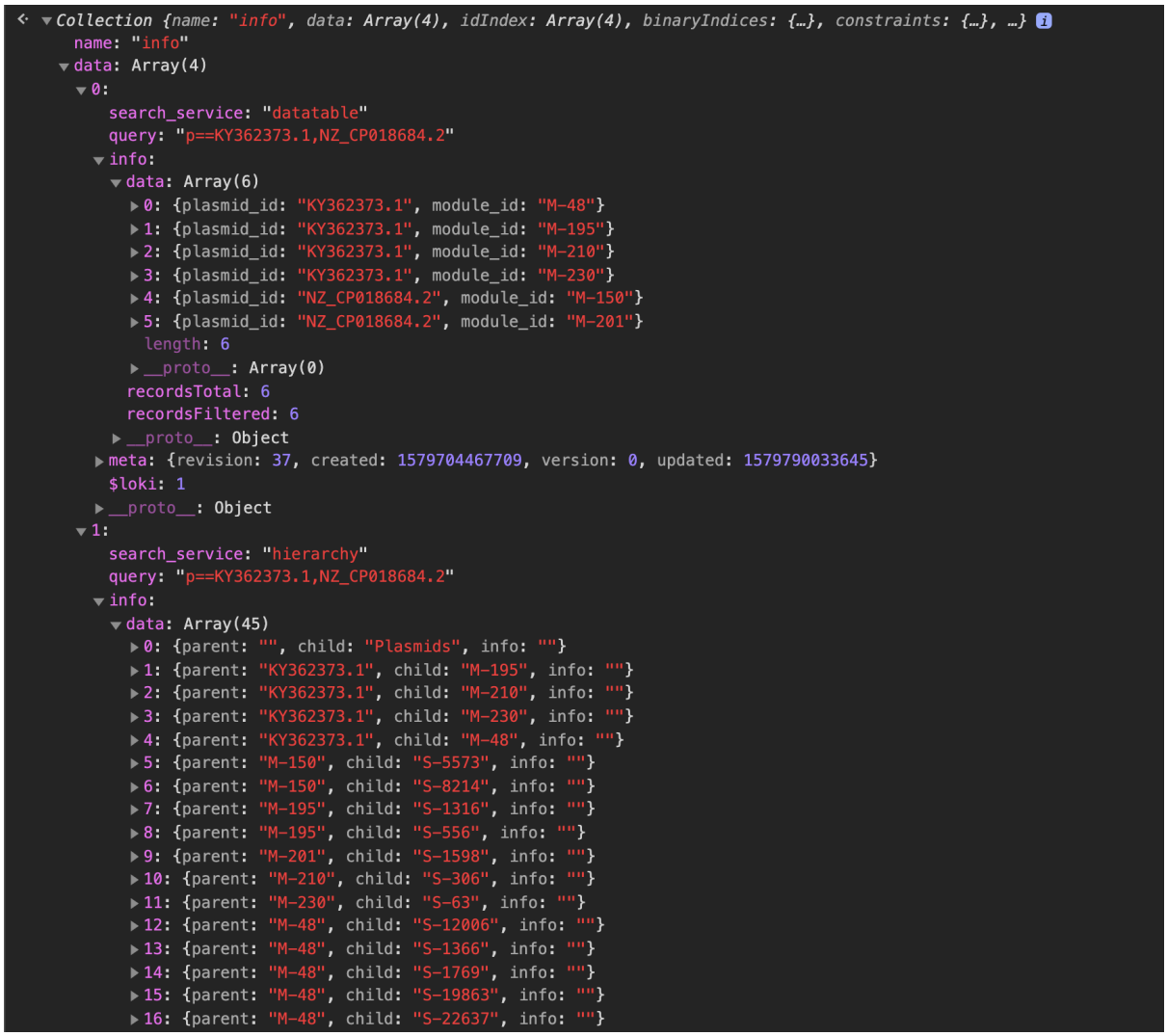

Figura 2.2: Datos almacenados en LokiJS

2.5. Tablas de datos

La presentación de hojas de datos la realizamos apoyándonos en el paquete *DataTables.js* [4] que aporta funcionalidades que requerirían bastantes diseños y desarrollos adicionales: paginado, ordenaciones, ocultación de columnas, exportaciones a *csv* y *pdf*, ... En la imagen 2.3 se muestra el aspecto de la tarjeta generada a partir de estos datos.

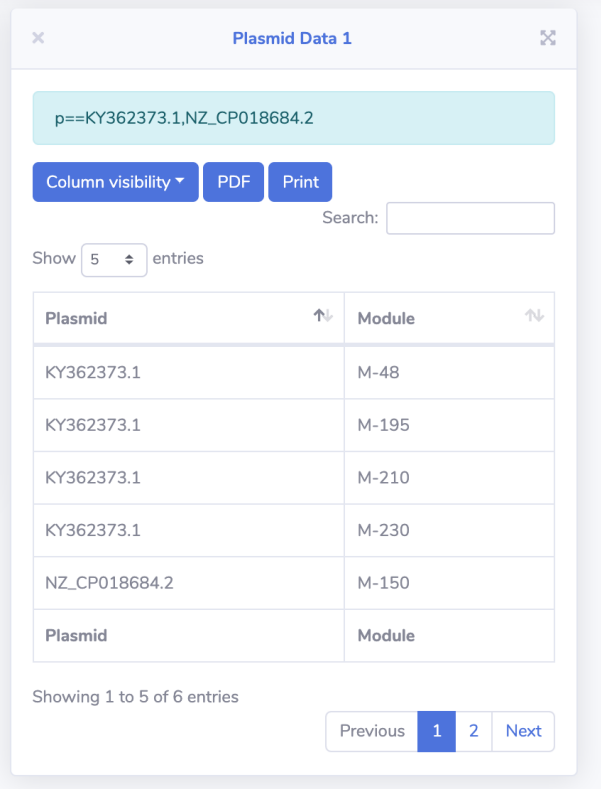

Figura 2.3: Datos presentados por DataTables

2.6. Gráficos vectoriales interactivos.

Decidimos utilizar la librería **d3js** [3] por su grado de madurez y la proliferación de ejemplos disponibles pero sobre todo:

1) Genera *html* dinámico a partir de los datos (similar a *jquery*).
2) Permite asignar funciones a eventos.
3) Es posible asociar datos a elementos *html5 svg* en escasas líneas de código.

Como contrapartida es tan compacto que no resulta excesivamente legible.

d3js no visualiza, sólo lo utilizamos para generar código *web-svg*.

El navegador realiza el renderizado a un gráfico escalable.

El gráfico puede ser interactivo y asignar cualquier función a cualquier evento en el navegador.

Lo consideramos crucial para explorar con comodidad información biológica.

En la imagen 2.4 se muestra una tarjeta donde hemos dibujado un gráfico jerárquico. La jerarquía presentada es la de módulos y plásmidos para dos plásmidos seleccionados.

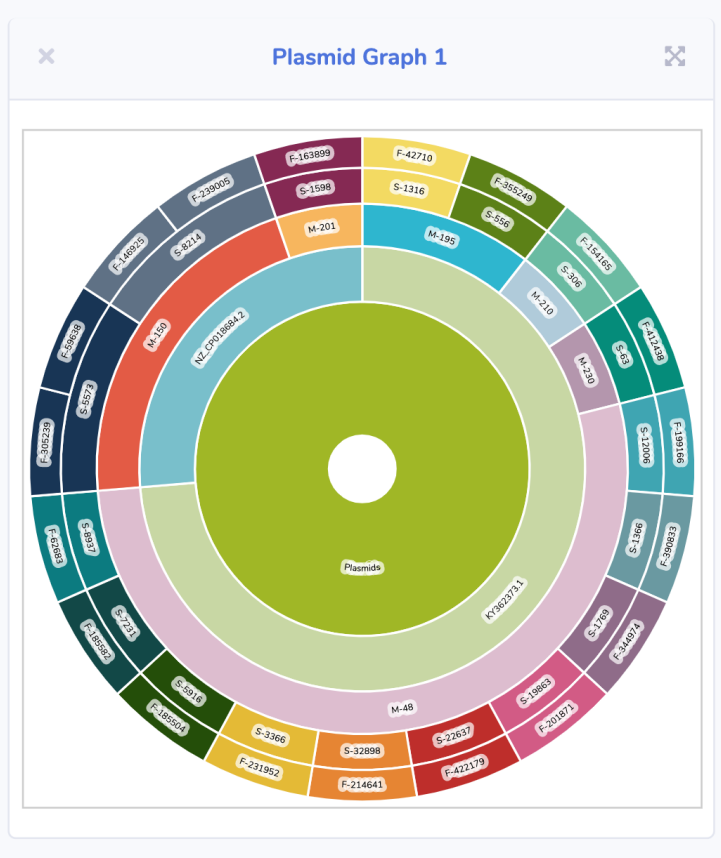

Figura 2.4: Gráfico jerárquico d3js

Si pulsamos en cualquiera de las áreas, el gráfico despliega los niveles inferiores.

También se puede retroceder.

La estructura de uno solo de los plásmidos se muestra en la figura 2.5

Los gráficos tipo *force network* ofrecen una representación interactiva de agrupamientos tal y como se muestra en la figura 2.6 Es posible programar componentes interactivos con los que modificar los parámetros del gráfico.

2.7. Subsistema de caché

Este sistema se encarga de gestionar los elementos que se almacenan localmente para evitar accesos recurrentes innecesarios al servidor. Este almacenamiento, llevado hasta su máxima expresión habilita la aplicación para funcionar también en modo desconectado.

Tenemos cuatro tipos de caché, dos gestionados en el *service worker* sobre el *application cache* del navegador y otros dos gestionados por la aplicación (librería *LokiJS*) sustentados en el *local storage* del navegador sobre bases de datos de objetos *javascript* (*noSQL*).

1) Caché de estáticos. Gestionada por el *service worker*.
2) Caché de artefactos injertables. Gestionada por la aplicación.
3) Caché de datos. Gestionada por la aplicación.
4) Caché de módulos javascript. Gestionada por el *service worker*.

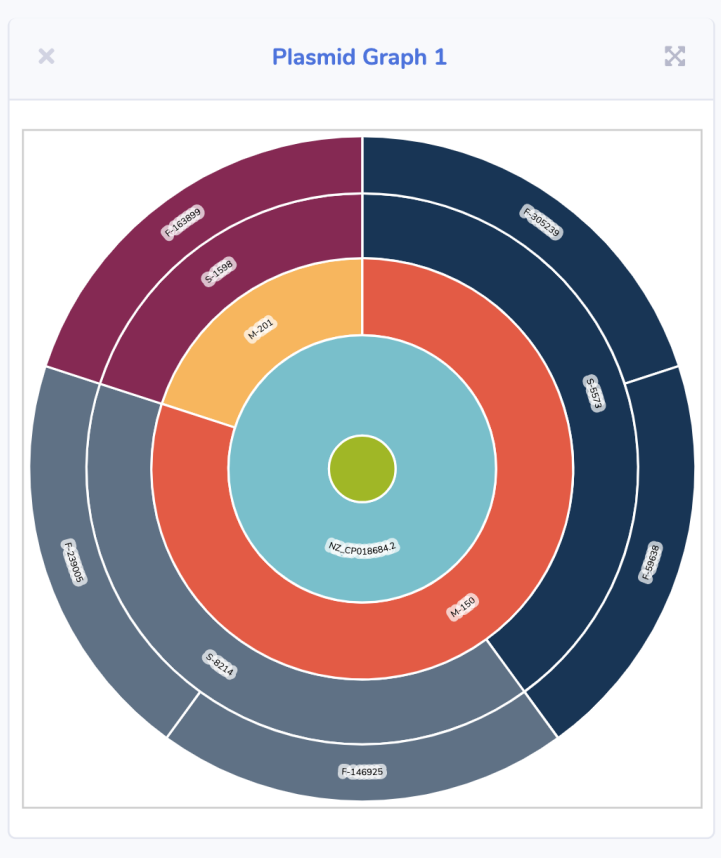

Figura 2.5: Gráfico jerárquico d3js drill down

2.7.1. Service Worker

El *service worker* es es la pieza básica del sistema de caché. Todas las llamadas que parten desde nuestra aplicación hacia la red pueden ser interceptadas escuchando el evento *fetch*. En ese punto tenemos acceso tanto al objeto *request* de la llamada como al objeto *response*. Es el lugar ideal para tomar decisiones de prioridad de caché.

El *service worker* se ejecuta en el navegador directamente, fuera del contexto de la aplicación y se mantiene en ejecución mientras el navegador está activo.

No cuenta hoy en día con una implementación completa de las librerías *javascript* por lo que no es recomendable desarrollar en él algoritmos muy complicados.

El hecho de que no desaparezca cuando la aplicación no está cargada en el navegador lo convierte en el lugar ideal para implementar respuestas a eventos como *push*, además del evento *fetch* comentado anteriormente.

Prioridades

Existen muchas formas de implementar el mecanismo de caché [22], entre otras:

1) **Cache first**: Primero se comprueba que el fichero está en caché, si no está lo descargamos y lo almacenamos. Si está devolvemos el elemento de caché a la aplicación. El almacenamiento y la respuesta a la aplicación pueden hacerse en paralelo.

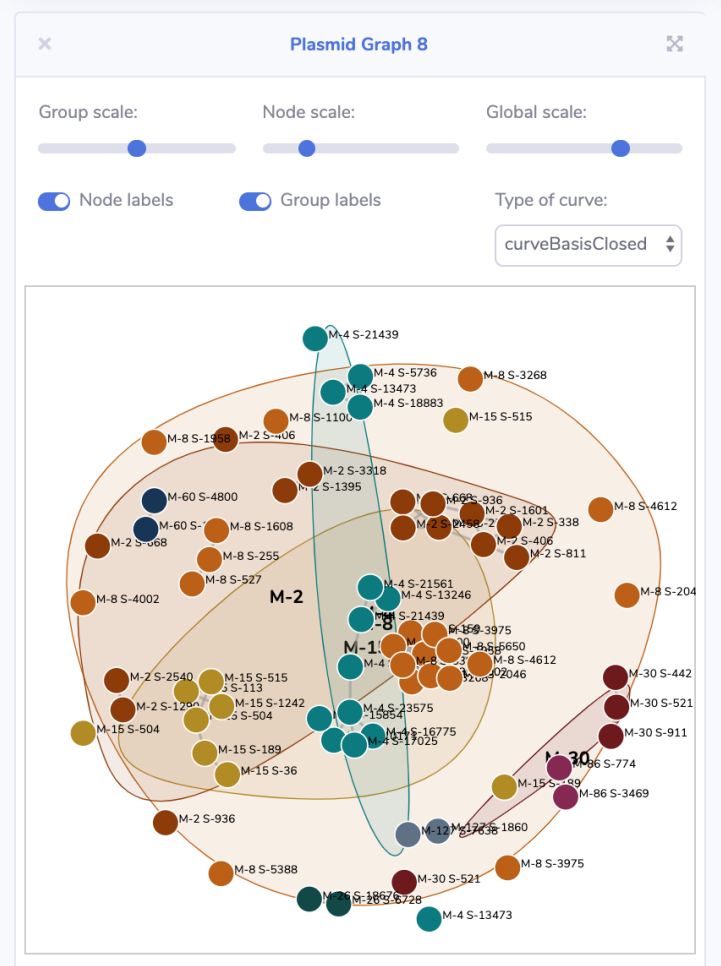

Figura 2.6: Gráfico de visualización de clusters

2) **Network first**: Primero se descarga el fichero desde la red, después se actualiza en el caché, en paralelo a la respuesta al usuario. Si falla la descarga de red, se devuelve el elemento que está en caché.

Nosotros hemos optado por la primera, porque decidimos implementar la aplicación con el criterio *offline first*, que en resumen significa priorizar el funcionamiento desconectado, que no sólo significa que extraemos en primer lugar los datos y componentes desde el caché local, sino que cualquier acceso a la red debe realizarse en segundo plano para que no bloquee el funcionamiento en caso de no estar la red disponible (*timeouts*). Es decir, la experiencia de usuario en cuanto a velocidad de respuesta debe ser siempre como si la aplicación se encontrara ya instalada en su sistema y los datos que utiliza la aplicación también residieran en local. Porque lo que vamos a implementar es una aplicación web progresiva (*PWA*).

En el funcionamiento conectado también obtenemos ventajas de rendimiento accediendo primero al caché local, que ahora no estará controlado por el navegador, sino por nuestro *service worker*.

Pero si priorizamos el contenido en caché ¿cómo sabemos que existe una nueva versión disponible en el servidor?. Existen muchas estrategias desde aplicar tiempos de expiración cuya extinción disparen eventos de renovación en segundo plano, comprobar periódicamente algún registro de versiones contra la propia base de datos remota descargando de nuevo si es necesario (*polling*), informar a los navegadores de la necesidad de descarga mediante tecnología *push*

desde el servidor, o, por qué no, *inpassing*. Inevitablemente, el sistema de caché exige desarrollar también estrategias para garantizar la coherencia de las versiones descargadas cuando los cambios implican a varios módulos.

Instalación

La instalación del *service worker*, se desencadena desde la propia aplicación. Una vez de descargado, se ejecuta activándose la respuesta a una seria de eventos: *install*, *activate* , *fetch* y *push* . Los dos primeros son disparados inmediatamente y sirven para configurar la caché. También para forzar su borrado completo desde el servidor (simplemente cambiando el nombre de la misma).

Evento *fetch*

En el evento fetch interceptamos las llamadas a varios tipos de elementos y realizamos diferentes acciones dependiendo de su naturaleza:

1) Estáticos almacenables en caché suministrados por nuestro servidor. Son los ficheros *js*, *css* y *html* que se descargan con la página principal. Están sujetos a control de versiones.
2) Estáticos no almacenables en caché suministrados por otros servidores. No están sujetos a control de versiones. En nuestro caso son unos pocos ficheros de fuentes tipográficas de la aplicación. Consideramos que la aplicación deberá suministrar estos ficheros en el futuro.
3) Elementos no almacenables en caché. En nuestro caso están relacionados con llamadas al *proxy browsersync* y son puramente instrumentales para un entorno de desarrollo y no bloquean si la aplicación se encuentra sin conexión.
4) Módulos dinámicos (suministrados por nuestro servidor). Son ficheros *js* que cargamos dinámicamente como módulos. Están sujetos al control de versiones pero requieren un tratamiento especial para evitar la caché del navegador (parámetro *mtime*) lo que nos obliga a eliminar este parámetro antes de almacenarlos en caché. También tienen un tratamiento especial porque no los incluimos en la lista inicial de *url* a almacenar.

Lo primero que comprobamos en todos los casos es si la caché tiene asignada una prioridad menor respecto a la información almacenada en remoto. Si lo está se devuelve el dato directamente desde la red a la vez que se actualiza el elemento en cache con su versión descargada. En caso contrario devolvemos el fichero a la aplicación desde la caché y en paralelo lanzamos contra el servidor una consulta para comprobar si existe una nueva versión (cabecera http *x-plasmidnet-subversion*). Si el elemento debe volver a descargarse se marca en una matriz. Este es el primer paso del *inpassing*. Con esta marca la caché se comporta con una prioridad menor para este elemento y en el siguiente acceso será descargado, devuelto a la aplicación y almacenado en la caché (en segundo plano) con la nueva versión. Este es el segundo paso del *inpassing*.

En definitiva, la técnica *inpassing* consiste en este caso:

1) Primer acceso al elemento: comprobar la necesidad de actualización del elemento.
2) Siguiente acceso al elemento: descargarlo y actualizar la caché.

La gestión de versiones se efectúa intercambiando la cabecera *x-plasmidnet-subversion* entre cliente y servidor. En el servidor todos los elementos son informados con el código criptográfico *md5* de su contenido y devuelto en esta cabecera en la respuesta. En la comprobación de versión

el *service worker* envía al servidor la cabecera mediante una consulta *REST*. El servidor retorna un objeto *json* que incluye un indicador lógico.

La caché de artefactos injertables se gestiona en base a una estrategia similar.

Para la caché de datos no existe control de versión. Desde la interfaz el usuario podrá solicitar datos nuevos en la pantalla de búsqueda, aunque en cualquier caso siempre serán almacenados en la caché para que estén disponibles en modo desconectado.

Limitaciones y orientación futura

La descarga de doble paso debe simplificarse en uno. Ahora está reutilizando la lógica de prioridades de caché, pero es una economía que no justifica la doble interacción.

Por otra parte la recarga de un componente se efectúa independientemente de las demás y a veces tendremos que gestionar dependencias. No hemos desarrollado ningún mecanismo para sincronizar las versiones de diferentes componentes cuando estas deban aplicarse de forma solidaria. Con el fin de cubrir todas las casuísticas con seguridad, este es el nuevo diseño a priorizar en la evolución del sistema:

1) Incluir un nuevo campo que indique el código de versión de despliegue que acompaña al código específico de *subversion*. Dos o más componentes que deban distribuirse conjuntamente serán marcadas con el mismo código de versión de despliegue.
2) Incluir en la base de datos una lista de los componentes que van asociados a la nueva versión, indicando si necesitará acciones del usuario o no (si se trata de módulos javascript seguramente no será necesario, si se trata de cualquier otro componente el usuario deberá reiniciar la pestaña en el navegador).
3) Las nuevas versiones serán descargadas por el navegador sobre una caché alternativa.
4) Cuando la aplicación comprueba que todas las componentes están descargadas, y si es necesaria la acción del usuario, lo in dicará.

Notificaciones *push*

La tecnología *push* capacita a un servidor *web* para enviar un mensaje directamente al navegador, mensaje que puede por ejemplo lanzar una ventana emergente informando al usuario de cualquier aspecto relevante, de forma muy parecida al funcionamiento de los mensajes que nos llegan al móvil. Para ello el navegador debe registrarse en un elemento de red y recibir un identificador que informa a su servidor. El servidor puede enviar mensajes *push* hacia este identificador utilizando también el mismo elemento de red que realiza la intermediación. El problema actual es que cada fabricante utiliza su propio servicio de comunicaciones *push* [23].

Nosotros hemos utilizado la tecnología *push* sólo en local únicamente para modificar en tiempo de ejecución dos comportamientos de la aplicación:

1) Nivel de *log* en consola de navegador. Por defecto está desactivado. Su activación depende del contenido de una cadena de tres caracteres que indica la activación/desactivación de los *logs* de error, información y depuración. Por defecto, en la aplicación y el *service worker* su valor es *NNN*.
2) Activación/desactivación de caché. De la misma forma se indican en una cadena de cuatro posiciones si la caché correspondiente está invalidada o no. Por defecto todas las cachés son válidas (*NNNN*). Si queremos despriorizar alguna de ellas debemos marcarla como *Y*.

Estas facilidades son muy importantes para las pruebas.

El push lo lanzamos desde una funcionalidad provista por Chrome 2.7 mediante la cual podemos emitir un mensaje push hacia el *service worker*, que será capturado por su evento. El *service worker* actualiza sus niveles de traza y las prioridades de caché e informa a la aplicación para que haga lo mismo.

El mensaje que enviamos no es otro que la concatenación de las cadenas de validez de cachés y de niveles de traza, con un separador.

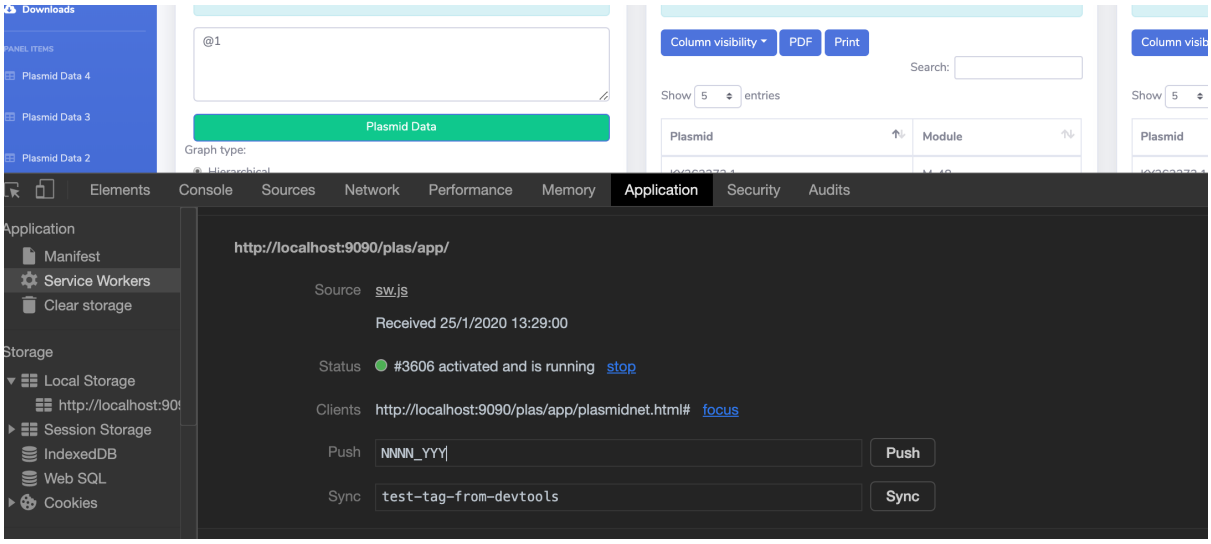

Figura 2.7: Push directo desde Chrome

2.8. Subsistema multiversión

El subsistema multiversión debe ser compatible con las políticas de caché. De hecho hemos conseguido que funcione ortogonalmente a él.

El objetivo de este sistema es que un desarrollador pueda utilizar versiones diferentes de la *web app* y versiones diferentes del software del servidor. Es decir, mientras un usuario normal está ejecutando las versiones publicadas, el desarrollador sería capaz de realizar las últimas comprobaciones de una nueva versión de la aplicación que todavía no es pública, aunque de alguna forma está ya disponible en producción.

Podríamos pensar que diferir pruebas a un entorno productivo no es buena práctica ya que deberían realizarse en entornos previos de certificación o de integración. Pero la realidad llega a ser muy lejana al caso ideal. En nuestro caso, por ejemplo, existe un aspecto muy difícil de verificar convenientemente en entornos previos: el funcionamiento sobre *https*. A veces también es necesario realizar una intervención rápida para corregir un problema, y esta sin duda es una nueva vía. Es importante señalar que esta capacidad de nuestro sistema para ejecutar varias versiones concurrentes no está muy extendida en otros sistemas *web*. De hecho es una capacidad muy difícil de implementar a posteriori, con el sistema ya construido.

Para nosotros ha resultado relativamente sencillo porque lo hemos tenido en cuenta desde el principio. Habitualmente, y en el mejor de los casos, se cuenta con instancias del servidor provistas de las nuevas versiones, donde el desarrollador puede probar. Este tipo de soluciones no suelen ser perfectas porque precisan de la modificación adicional de elementos de red como balanceadores de carga, proxy caché u otros que se sitúan entre el servidor de la aplicación e *Internet*. Del mismo modo si estos elementos no han sido configurados con la multi-versión en mente, resulta muy difícil configurarlos después, sobre todo por el riesgo de afectar al entorno

productivo. PlasmidNet no depende de configuraciones externas porque es la propia aplicación la que gestiona qué versión se asocia a cada usuario.

La problemática *https* deriva de la dificultad y el coste de conseguir certificados avalados por una entidad certificadora. Es cierto que podríamos generar certificados locales, que los navegadores rechazarían, pero cuyo rechazo logramos soslayar otorgando confianza al certificado. Sin embargo no es el caso de los *service workers*. No hay forma de autorizar en los navegadores un *service worker* bajo conexión *https* no comprobable en una autoridad certificadora, ni tampoco nos sirve utilizar *http* bajo otro dominio que no sea el de *localhost*. De hecho el *service worker* no funcionará salvo en estas limitadas casuísticas. Después veremos hasta dónde conseguimos llegar en las pruebas con entornos de desarrollo, pero es importante señalar que, como debe ser, el no funcionamiento del *service worker* sólo afecta al modo desconectado, la aplicación funcionará perfectamente bajo conexión, eso sí, sin sistema de caché de estáticos y módulos (la ausencia de carga dinámica de módulos no se ve afectada porque cargamos todo el software en el arranque de la página).

¿Cómo consigue un desarrollador acceder a nuevas versiones?. Simplemente editando una *cookie* en el navegador: *pn_ version*.

Esta *cookie* contiene dos campos separados por el carácter _. El primero de ellos indica la versión de la *web app* que por defecto es la versión *v0* que es la versión pública en todo momento y el segundo la versión de software que también por defecto es *v0*. Esta *cookie* no se activa para un usuario normal, ya que el servidor interpreta que si la cookie no está informada, su valor es *v0_ v0*. Estas versiones son independientes: no quiere decir que la versión *v1* del servidor tenga que estar asociada a la *v1* de la *web app*.

Cuando, después de las pruebas queramos activar la nueva versión en *v0* no hay más que redistribuir el nuevo software en esta ubicación (lo veremos en el capítulo relacionado con el servidor) o activar un indicador ad-hoc en el servidor que le indica cuál es la versión pública en cada momento.

El control de caché se ve afectado sólo en lo necesario, con los cambios de versión de la *web app* porque cada elemento de servidor tiene asignado un *md5* independiente para cada versión. Si es el mismo para las dos versiones implicadas, el componente de la caché no se verá modificado. O sea, sólo se refrescan los elementos modificados.

2.9. Aplicación web progresiva

PlasmidNet es una aplicación web progresiva. Esto implica, para un usuario final, que puede instalarse en su dispositivo móvil (teléfono o tablet) de forma que se comporte como una aplicación nativa:

1) La aplicación cuenta con un icono visible en las pantallas de aplicaciones.
2) Cuando accede a la aplicación y aunque esta se ejecute en el navegador, los diversos menús e iconos del navegador quedan ocultos al usuario: la aplicación ocupa toda la pantalla del navegador.
3) Si el usuario está fuera de red (frecuente sobre todo en dispositivos *tablet*) la aplicación está operativa alimentándose de los datos almacenados en su caché local.

La principal dificultad técnica es la implementación del modo desconectado a través del *service worker*.

Adicionalmente es necesario generar los iconos de la aplicación en diferentes tamaños e informarlos en el manifiesto, junto con otros aspectos [24]. El manifiesto es un fichero *json* con el siguiente aspecto:

```
{
  "short_name": "PlasmidNet",
  "name": "Plasmid Modules Portal",
  "icons": [{
    "src": "images/icon_128.png",
    "type": "image/png",
    "sizes": "128x128"
  ...
  },{
    "src": "images/icon_512.png",
    "type": "image/png",
    "sizes": "512x512"
  }],
  "background_color": "#2196F3",
  "display": "standalone",
  "orientation": "any",
  "theme_color": "#2196F3",
  "start_url": "/plas/app/plasmidnet.html"
}
```

Existe un plugin de auditoria para *Chrome* (*Lighthouse*) que nos validad la idoneidad de nuestra aplicación como *web app*.

En la figura 2.8 se muestra el aspecto de la aplicación en *iPadOS 13*.

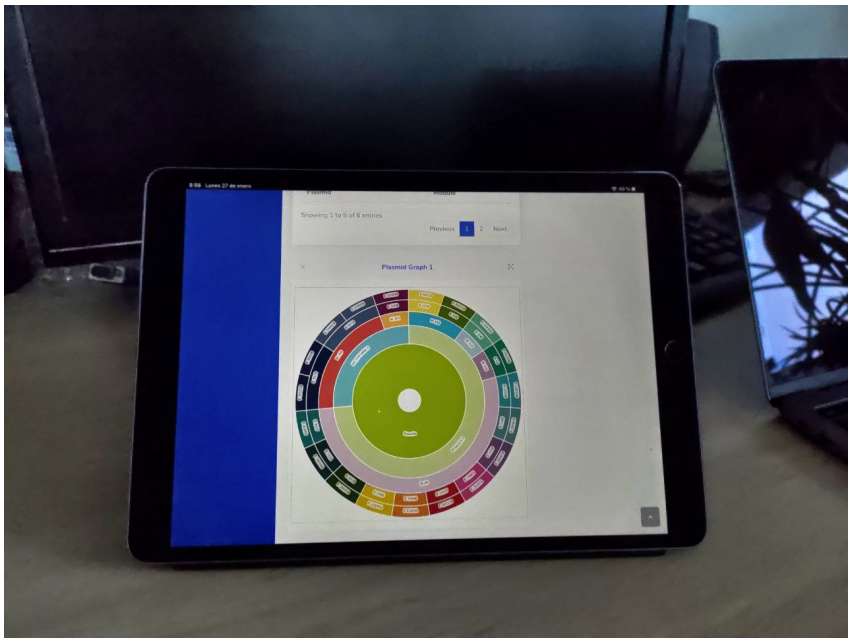

Figura 2.8: PlasmidNet en iPadOS como web app

En la figura 2.9 se muestra el icono en el panel principal y en la barra de aplicaciones: no existe diferencia con una aplicación nativa.

Figura 2.9: PlasmidNet en iPadOS como web app: panel y barra.

En la figura 2.10 se muestran las aplicaciones activas. A diferencia de la ejecución en navegador, no existe rastro de elementos de navegación: barra de búsquedas, botones, ...

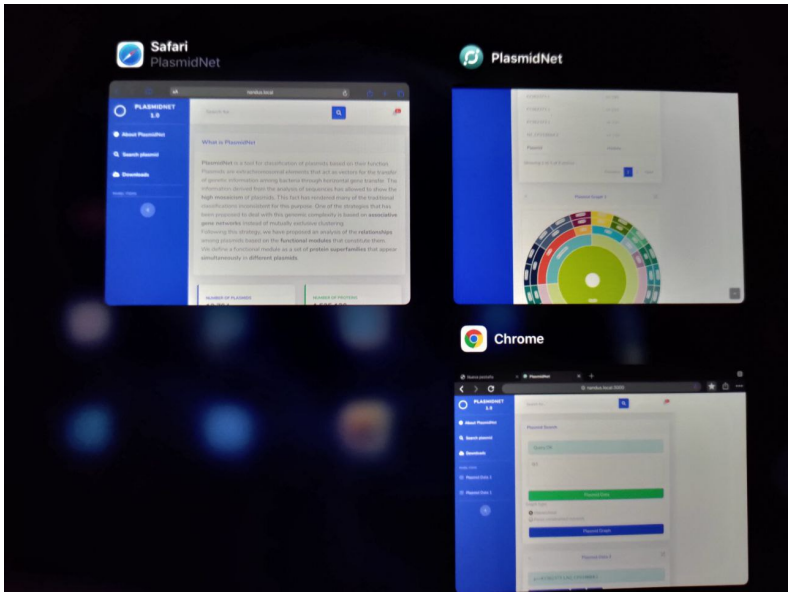

Figura 2.10: PlasmidNet en iPadOS como web app: aplicaciones activas

2.10. Diseño de la aplicación

Componentes:

1) Página única con tres visualizaciones:
   - Informativa donde se presenta la aplicación.

- Búsquedas sobre la base de datos de módulos funcionales de plásmidos.
- Descargas.

2) Cuatro componentes principales:

   - Menú vertical.
   - Barra horizontal de búsquedas y avisos.
   - Cuerpos compuestos de tarjetas de información (o descarga) y una tarjeta de búsqueda (en esta visualización).

3) Tres tipos de tarjetas informativas:

   - Datos.
   - Gráficos jerárquicos circulares interactivos (*sunburst*), donde los nodos de la estructura se muestran en anillos concéntricos.
   - Gráficos de red interactivos: análisis de *clusters*.

Funcionalidades:

1) El menú puede contraerse y en visualizaciones móviles se puede ocultar.
2) Las tarjetas pueden contraerse, expandirse o borrarse.
3) Las tarjetas cuentan con accesos rápidos desde el menú.
4) Existe un botón flotante para volver a la parte superior de la pantalla.
5) Las búsquedas se implementan mediante un lenguaje ad-hoc (*DSL*) muy versátil ideado para poder ser utilizado en un dispositivo móvil (sintaxis muy compacta).
6) Las tarjetas de datos están dotadas de paginación, ordenaciones, ocultación de columnas y exportación a *pdf* y *csv*.

La figura 2.11 evidencia las diferentes zonas de la aplicación:

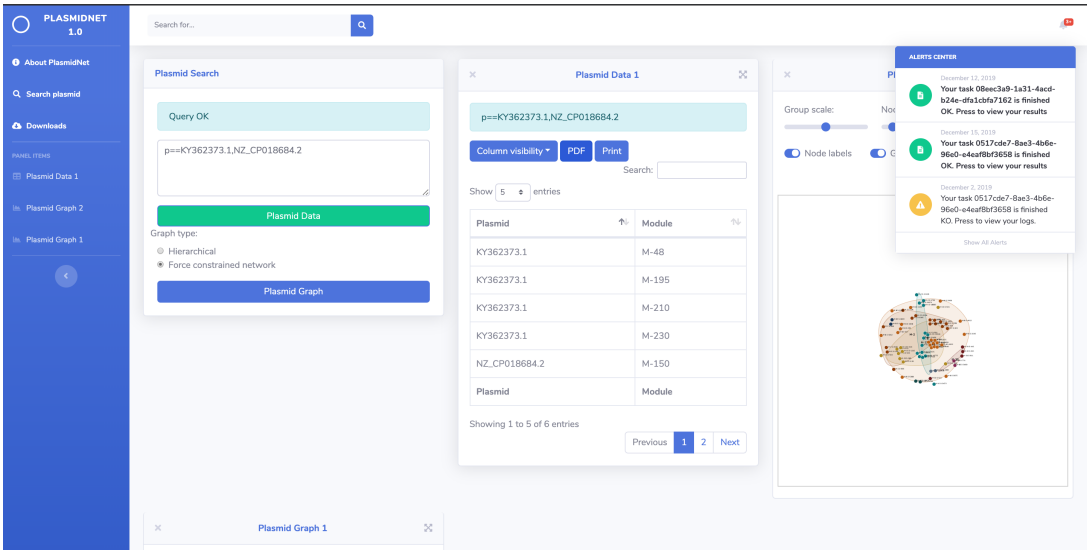

Figura 2.11: Visualización de búsquedas de plásmidos

La figura 2.12 muestra la pantalla informativa:

El diseño de la aplicación y sus elementos básicos se apoyan en una aplicación de administración construida sobre *bootstrap 4*: *SB-Admin2* [26].

Este tipo de plantillas, ideadas para tareas administrativas, implementan un panel central donde se presenten diversas tarjetas informativas, que pueden colapsarse, expandirse o borrarse.

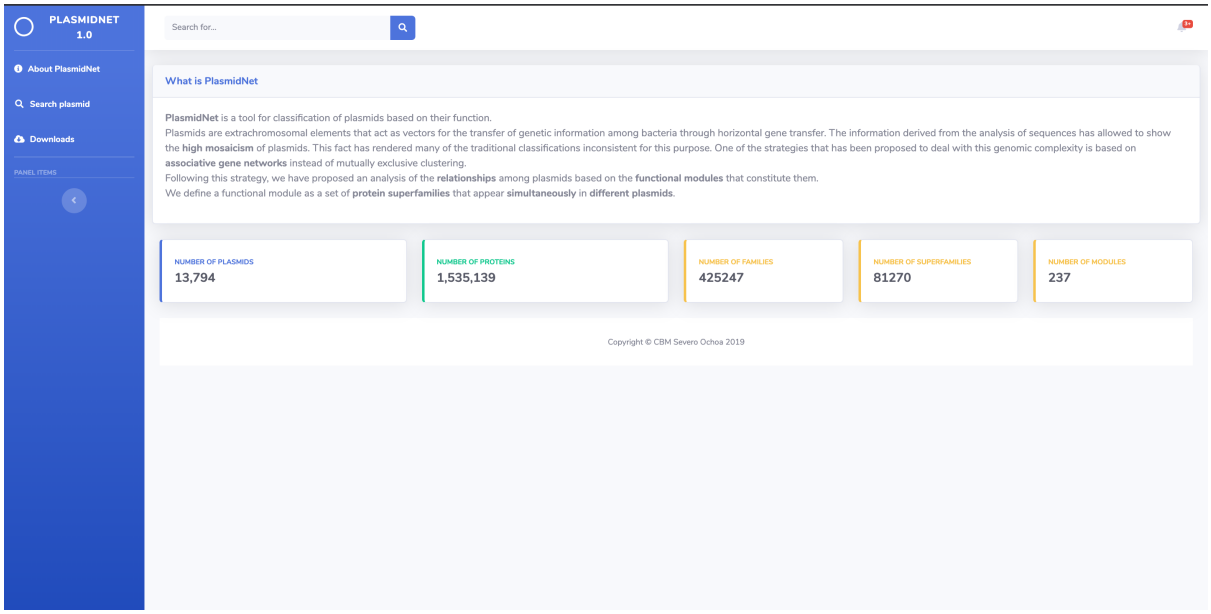

Figura 2.12: Información de la aplicación

Este comportamiento es ideal para sustituir una navegación convencional sobre varias páginas *html*. Además permite comparar, en la misma pantalla, información generada para diversos plásmidos. Adoptamos esta plantilla sobre todo por su sencillez, ya que preveíamos la necesidad de modificarla en gran medida.

2.11. Documentación como PWA

La documentación también puede ser publicada como *web app*:

- En modo desconectado se comporta como un documento local.
- Reutilizamos el *service worker* con su sistema de caché y multiversión.
- Reutilizamos las posibilidades de conversión documentales del software *pandoc* [9] y de todo nuestro sistema automatizado.
- Independencia de aplicaciones: no se necesita otra aplicación para visualizarla.
- Independencia de "estándares" documentales como *epub3* que no acaban de ser adoptados de forma homogénea por las plataformas.
- Multidispositivo.

Características de la implementación:

1) Barra lateral con enlaces a diversas partes del documento.
2) Barra de navegación con sistema de búsqueda.
3) La barra lateral no se desplaza con el resto del documento cuando se alcanza el final de la pantalla.
4) La barra de búsqueda permanece siempre visible.
5) Presentación en *web* mediante *revealjs*.

Figura 2.13: Web app documental

En la figura 2.13 se muestra la pantalla principal de la aplicación documental.

En la figura 2.14 se muestra la presentación *revealjs*.

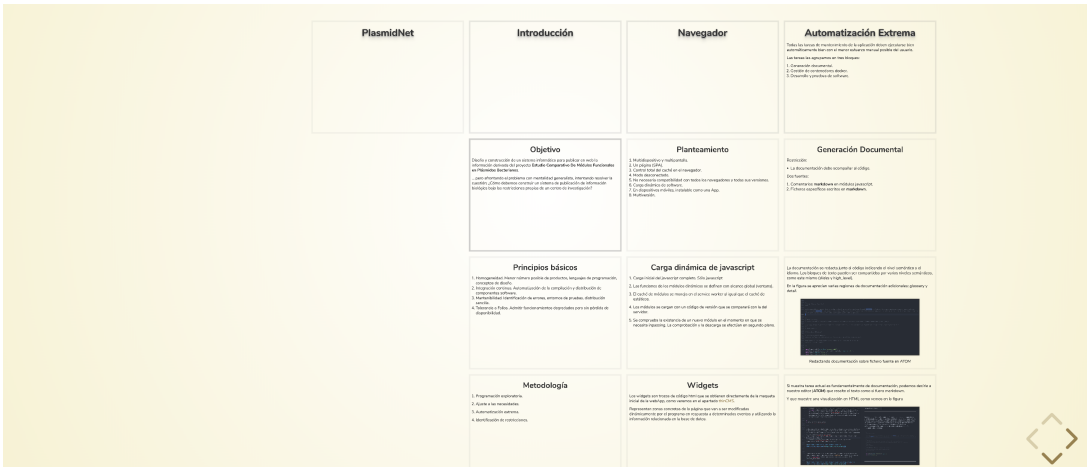

Figura 2.14: Web app documental: revealjs

En la figura 2.15 se muestran varias visualizaciones multidispositivo: búsqueda, tabla, código e imágenes.

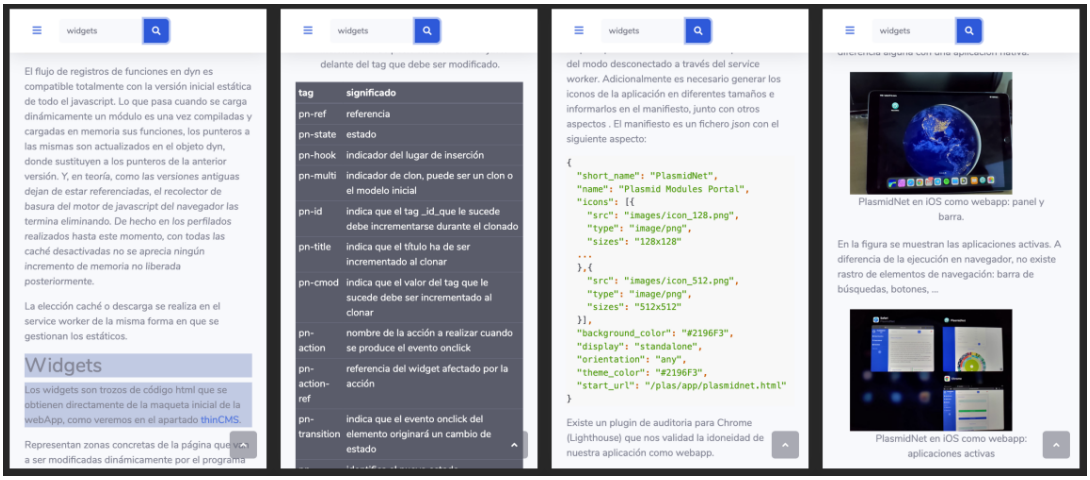

Figura 2.15: Web app documental: mobile

3

Servidor *web*

3.1. Planteamiento

El desarrollo se ha realizado sobre *nodejs* [38] y *express* [6]. El uso de *nodejs* nos otorga inmediatamente capacidades multiplataforma, de forma que el software desarrollado puede ser distribuido sin cambios en cualquier sistema operativo donde *nodejs* sea compatible. Actualmente *linux*, *macos* y *windows* están totalmente soportados.

Construir un servidor *web* con *nodejs* nos ofrece un alto nivel de control del desarrollo pero a la vez la necesidad de lidiar a bajo nivel (*http*) con las comunicaciones respecto a configurar un servidor convencional como *apache* o *weblogic*. De hecho la principal diferencia entre estos dos tipos de desarrollos se resume en el concepto **programación frente a configuración** (*POC*, ver glosario). Si hubiéramos utilizado *apache* muchas líneas de código se resumirían (o expandirían) en la sintaxis, muchas veces farragosa de ficheros como *http.config* y con suerte dos o tres ficheros más. Ya que otros servidores *web* o de aplicaciones requieren la configuración precisa de una pléyade de estos ficheros de configuración o alternativamente interactuar con una aplicación (*web*) de administración, prolija hasta límites difíciles de aceptar para muchos usuarios que sólo buscan una mínima parte de lo que allí se ofrece.

En realidad habríamos necesitado dos piezas, presentes en la mayoría de los sistemas:

1) Servidor *web* para suministrar ficheros estáticos (*html*, *javascript*, *css*, ...)
2) Servidor de aplicaciones para la ejecución de código, en este caso *javascript*.

Incluso puede proveerse un tercer servidor especializado en descargas de ficheros.

Una combinación típica en entornos *java* (*J2EE*) es utilizar *apache* como servidor *web* y *JBOSS* o *Weblogic* como servidores de aplicaciones, capaces de ejecutar máquinas virtuales de *java*.

En nuestro caso, todos los ámbitos de operación las implementamos bajo la misma arquitectura *nodejs+express*. Podríamos haber encaminado los estáticos hacia *apache* pero atentábamos contra el principio de homogeneidad y nos dificultaba la construcción de un servidor multiversión.

El sistema está sustentado sobre un conjunto de tareas atómicas, con toda la independencia posible del sistema global, con el fin de poder probar cada elemento independientemente, dentro

de un ámbito de pruebas tanto automático como manual. Para este fin hemos utilizado **gulp [?**], una librería de *nodejs* que proporciona un sistema de tareas también basado en los principios *POC*. En el manual de usuario se detallan todas las operaciones que pueden ser ejecutadas bajo este sistema.

3.2. Requisitos principales

1) Toda función podrá ser verificada aisladamente (fuera del contexto del resto de la aplicación).
   - Facilita pruebas unitarias automatizadas.
2) Sistema configurable, capaz de especializarse en varios tipos de servidores *http*:
   - Servidor para la *web app*.
   - Servidor *REST*.
   - Servidor de comunicaciones entre nodos de orquestación.
   - Entornos de pruebas/producción: niveles de *log*.
3) Sistema distribuible en contenedores *docker*:
   - Control de versiones de productos de terceros.
   - Multiplataforma.
4) Facilitar al máximo las pruebas en todos los entornos y la disponibilidad del sistema:
   - Carga dinámica de un porcentaje alto de módulos sin necesidad de reiniciar el servidor.
   - Multiversión: un desarrollador podría usar una versión distinta a la de un usuario.
5) Homogeneidad de entornos:
   - El sistema es idéntico en todos los entornos (desarrollo/pruebas y producción). Las variaciones entre los mismos dependen sólo de la configuración.

3.3. Implementación

La construcción sobre *nodejs* se ha estructurado en seis tipos de ficheros fuente *javascript*. Algunos ficheros pertenecen a más de una categoría:

1) Estructurales. Módulos que sirven para amalgamar las tareas de otros módulos.

- **gulpfile.js** Fichero principal de la aplicación: integra las tareas definidas por el resto de módulos.
- **gulpfile_app.js** Integra el sistema de *logs* y los diferentes encaminamientos.
- **proxy_modules.js** Gestiona la carga dinámica de módulos.

2) Módulos de tareas *gulp*. Todos los ficheros fuente que comienzan con **gulpfile**:

- **gulpfile.js** Define la tarea por defecto.
- **gulpfile_app.js**. Tareas de arranque de los servicios *web*.
- **gulpfile_cms.js**. Tareas relacionada con el ciclo de vida de contenidos *web*.

- **gulpfile_dist.js**. Tareas de distribución de software del servidor.
- **gulpfile_doc.js**. Tareas de generación automática de documentación.
- **gulpfile_docker.js**. Tareas de creación y arranque de *dockers*.
- **gulpfile_misc.js**. Tareas *git* y copias de seguridad.
- **gulpfile_bio.js**. Tareas relacionadas con consultas a la base de datos de módulos funcionales de plásmidos.
- **gulpfile_pipelines.js**. Tareas del ciclo de vida de la orquestación de procesos.
- **gulpfile_site.js**. Tareas de distribución de software de la *web app*.
- **gulpfile_task_model_loki.js**. Tareas relacionadas con la base de datos de orquestación. Desarrollo sobre **LokiJS** [36]
- **gulpfile_task_model_redis.js**. Tareas relacionadas con la base de datos de orquestación. Desarrollo sobre **redisjs** [11].
- **gulpfile_task_model_sqlite.js**. Tareas relacionadas con la base de datos de orquestación. Desarrollo sobre **SQlite3** [13]

3) Módulos de rutas http:

- **router.js**. Implementa las respuestas a las llamadas http para obtener los ficheros estáticos de la *web app*.
- **router_crt.js**. Implementa la respuesta para la validación *certbot* y obtener los certificados de *Let's Encrypt* para pruebas.
- **router_pipelines.js**. Implementa el protocolo (*http*) de comunicación entre los nodos de orquestación.
- **router_services.js**. Implementa las respuesta a las llamadas *http* de los servicios *REST*.

4) Módulos de log:

- **logger_app.js**. Módulo de *logs* de la aplicación.
- **logger_express.js**. Módulo de *logs* de la infraestructura exprés.

5) Proxy de carga dinámica

- **proxy_modules.js**. Gestión de carga dinámica de módulos.

6) Configuración:

- **pipelines.js**. Definiciones de los procesos orquestables: ficheros de entrada y salida, programas a ejecutar, orden de ejecución, ...

3.4. Configuración

El sistema es configurable mediante diversos parámetros de forma que podemos lanzar varios tipos de servidores *web* (especializaciones), ajustar el nivel de traza y definir puertos y protocolos (*http(s)*).

Para desarrollo y por defecto el servidor es totipotente: se arrancan todos los servicios y todos los puertos, también el del *proxy browsersync*.

Si un parámetro se configura con valor *Y*, significa que está activada la funcionalidad.

Para que la *web app* sea operativa deben de estar activados los parámetros *app*, *services* y *pipelines* pero si sólo queremos levantar un nodo que participe en pipelines se puede desactivar *app*. Si sólo queremos que el nodo actúe como servidor de servicios *REST*, basta activar *services*, de acuerdo a la siguiente tabla:

Cuadro 3.1: Parámetros de configuración del servidor *web*.

parámetro	descripción
http	arranca servicio http de la *web app*
httpdoc	arranca servicio http alternativo, utilizado para el servidor documental
https	arranca servicio https por puerto 443
app	publica las rutas necesarias para los estáticos de la *web app*
services	publica las rutas de servicios *REST*
pipelines	publica las rutas del protocolo de comunicación en el orquestador
certbot	ruta para la validación del servidor para la instalación de certificados
browsersync	activa o no el *proxy* para facilitar pruebas simultáneas
loglevel	nivel de traza: *debug*, *info* o *error*

3.5. Encaminamientos

Se sustentan en varios módulos que generan las respuestas ante los diferentes tipos de solicitudes.

Estos módulos son cargados dinámicamente desde *gulpfile_ app.js* comprobándose la versión en cada solicitud (lo detallaremos en otro apartado). Esto implica que podemos realizar modificaciones en las rutas de encaminamiento o añadir o borrar rutas existentes en caliente, sin reiniciar el servidor.

3.5.1. Solicitudes de estáticos

Corresponden a las solicitudes de ficheros *html*, *css*, *javascript*, fuentes e imágenes.

Las respuestas son creadas en el módulo *router.js*.

Los ficheros se descargan del directorio físico asociada a la versión de la *web app*. El directorio tiene el mismo nombre que la versión (*COC*). La versión no se solicita en la *URL*, llega en la *cookie pn_ version* y por defecto es *v0*, la versión pública a la que acceden los usuarios.

En una cabecera *http* de la respuesta se devuelve el *md5* del contenido del fichero para que el sistema de caché de la *web app* pueda decidir si existe una nueva versión disponible.

3.5.2. Solicitudes de servicios *REST*

Contamos con varias tipologías de servicios:

1) Servicio de prueba.
2) Servicios de datos. Devuelven resultados de consultas a la base de datos de módulos funcionales de plásmidos. El objeto *json* se crea adaptado ya al tipo de presentación en parte para simplificar esta tarea en la *web app*. En cierto modo estamos adoptando implícitamente un modelo *MVCVC* repartiendo el trabajo entre el servidor y el cliente. La principal

desventaja es que estaríamos almacenando en las cachés locales de los navegadores formatos diferentes del mismo conjunto de datos, en lugar de almacenar los datos directamente, sin embargo, se presentan otras ventajas: el servicio resulta más útil para clientes *REST* del mismo que deseen implementar otra interfaz en local.

- Consultas para tablas de datos. El formato está adaptado a la representación en tabla.
- Datos para gráficos jerárquicos. El formato se adapta para la presentación en estructura jerárquica.
- Datos para gráficos de red. El formato se adapta para la presentación en estructura de red.

3) Servicio de contenidos. Devuelve los injertables desde la base de datos del *CMS*.
4) Servicio de módulos. Retorna los módulos dinámicos *javascript* que serán importados desde el navegador.
5) Servicios informativos de la aplicación. Realizan funciones de control como indicar si el dato en caché ya no es válido.

3.6. Sistema de *log*

El sistema de *log* se apoya en el paquete *winston* [20] que nos ofrece la posibilidad de configurar el sistema definiendo transportes (salidas como consola o fichero), formatos de fichero y niveles de log.

En PlasmidNet hemos aglutinado toda la información en dos módulos de configuración/programación (*POC*), uno para la aplicación en general, y otro empotrado en *express*.

Utilizamos tres niveles de log: *debug*, *info* y *error*, que se acotan a *error* si el entorno es el productivo.

Definimos dos salidas paralelas, una salida a fichero y otra a consola.

El entorno se especifica, valga la redundancia, con la variable de entorno *ENV*.

En el *log* incluimos el identificador único de *request* de forma que es fácil identificar en los ficheros de *log* todas las entradas correspondientes a una misma solicitud de usuario.

Construimos también un *log* para los accesos *http* hacia el servidor, tipo *access* típico de los servidores *web*. En este *log* grabamos también el identificador de la *request* con el fin de poder cruzar la información de todos los *log*.

3.7. Carga dinámica de módulos

Este subsistema nos permite recargar en el servidor sin reiniciarlo versiones nuevas de módulos *javascript* que sustituyen a las que están operativas en ese momento. Esta capacidad facilita enormemente las pruebas porque evitamos efectuar tediosos reinicios y además en producción en una gran parte de cambios de software mantendremos la disponibilidad.

Además, en esta estrategia están engranadas también las capacidades multiversión. Un desarrollador podría, indicando el identificador de versión en la *cookie pn_version*, solicitar al servidor que ejecute los módulos desde una versión distinta a la utilizada por los usuarios (*v0*). Esto le habilitaría para verificar en el entorno productivo el funcionamiento de la aplicación como último paso de la puesta en marcha de una nueva evolución del software de servidor.

¿Todos los cambios de software están exentos de reinicios?. No, y no es fundamental. Los módulos estructurales: *gulpfile.js*, *gulpfile_ app.js* y *proxy_ modules.js* requieren reiniciar el servidor, pero conseguimos no afectar a la disponibilidad del sistema en un amplio espectro de casuísticas.

La versión es comprobada utilizando el *timestamp* del fichero y su valor almacenado en local.

Alternativamente contamos con un nuevo algoritmo que utiliza la versión generada en la distribución de la misma forma que se utiliza en la *webapp*. De momento está en fase experimental.

4

Bases de Datos

4.1. Planteamiento

El afloramiento de multitud de opciones de servidores de *bd* no *sql* nos hicieron apostar inicialmente por una de estas nuevas arquitecturas. De hecho, comenzamos a utilizar *Lokijs* y después *redis* para el sistema de orquestación de tareas de usuario.

Pronto nos dimos cuenta que la sintaxiz de acceso *noSQL* resultaba un tanto farragosa para situaciones estándar de utilización como era el nuestro, tanto en el caso de la orquestación como en el caso de la propia base de datos de módulos funcionales de plásmidos. Las búsquedas en lenguaje *sql* nos parecieron más legibles que las propuestas por sus incipientes competidoras.

También contábamos con dos gestores de amplia utilización y de probada fiabilidad: *SQlite* [13] y *PostgreSQL* [10], que destacan por el avanzado desarrollo de su intérprete *SQL* que nos permite abordar consultas imposibles en otros gestores relacionales. De hecho pueden realizarse cruces de datos y agrupaciones de mucho más fácilmente que con cualquier otro lenguaje de programación como veremos. No deja de ser sorprendente que más de treinta años después de la invención de este lenguaje todavía resulte atractivo para muchos proyectos.

La simplicidad administrativa de *SQLite* nos tentó en gran medida, pero su falta de concurrencia en escritura podría poner en riesgo su idoneidad en producción. A pesar de ello, y teniendo en cuenta que la concurrencia en escritura de este sistema puede no ser un factor determinante, hemos construido esta primera versión sobre este gestor de base de datos. Aceleraremos el sistema gracias a la relativamente reciente configuración *WAL* [14] que acelera hasta un factor diez las actualizaciones, minimizando los problemas de encolamiento.

La adaptación a *Postgresql* será realizada en una versión posterior, sobre una instalación que se ejecute dentro de contenedor para minimizar los costes de instalación y configuración. La adaptación es trivial ya que las sentencias *SQL* esperamos que sean totalmente compatibles y todas las consultas y actuaciones contra las *BD* están encapsuladas en módulos *javascript* específicos. Probablemente no migremos todas las bases de datos, sólo la de módulos funcionales.

De todas formas la implementación *SQLite* no es un desperdicio y se va a mantener como alternativa. En entornos de desarrollo puede ser más adecuada y en entornos de sólo consulta o con actualizaciones escasas su adecuación es indudable.

4.2. Implementación

Contamos con tres bases de datos:

1) Módulos funcionales de plásmidos: Datos de la jerarquía funcional de los plásmidos.
   - módulos
   - superfamilias
   - familias
   - proteínas
   - plásmidos
2) Contenidos (*CMS*). Datos de contenidos (*widgets*).
   - contenidos
   - versiones
3) Orquestación. Gestión de tareas de usuario.
   - tareas
   - pasos

4.3. Módulos funcionales de plásmidos

Contiene varias tablas que albergan la jerarquía funcional elucidada en el proyecto [27].

Los plásmidos y proteínas cargados son los utilizados en el citado proyecto.

tabla	descripción
module_superfam	superfamilias asociadas a cada módulo
plasmid	identificador GID y secuencia de bases de los plásmidos
plasmid_module	módulos asociados a cada plásmido
plasmid_rel	puntuaciones de similitud entre secuencias de dos plásmidos
plasmid_superfam	superfamilias asociadas a cada plásmido
protein	proteínas: identificador GID, función y secuencia de aminoácidos
protein_norep	proteínas y proteínas representativas asociadas
protein_rep	proteína representativa de una familia, identificadores, localización y plásmido
protein_superfam	superfamilias asociadas a proteínas
superfam_rel	puntuaciones de similitud entre superfamilias

4.4. DSL de consulta

Nos hemos propuesto facilitar el acceso del usuario a la información mediante una interfaz de línea de comandos, basada en un lenguaje ad hoc de consulta (*Domain Specific Language* o *DSL*) extremadamente compacto ideado para los teclados reducidos de los dispositivos móviles.

La alternativa habría sido construir el típico formulario provisto de diferentes campos de entrada, con despliegues de elementos gráficos mediante selecciones *AND* y *OR* y similares. Estas interfaces ocupan mucho espacio, y resultan incómodas de manejar, sobre todo en pantallas pequeñas. La utilidad se amplifica si la sintaxis es fácil de recordar, pero existen otras ventajas.

Para el usuario:

1) Las consultas son muy fáciles de almacenar (*copy + paste*), incluso aunque la interfaz carezca de sistema de almacenamiento (de momento así es).
2) Las consultas son más fáciles de crear a partir de otras ya existentes. A diferencia de volver a introducir todos los campos en un formulario.
3) Es posible compartir consultas con otros usuarios.
4) La consulta es más fácil de interpretar.

Para el programador:

1) Aunque la implementación inicial no es necesariamente más sencilla, ya que a cambio de no diseñar un formulario es inevitable configurar un resaltador de sintaxis, sí es más fácil extender el lenguaje con nuevas capacidades.
2) Constituye una barrera idónea para defendernos de los ataques de inyección *SQL*, porque la implementación realiza de forma natural un conjunto de validaciones y conversiones antes de obtener la sentencia *SQL* final.

4.4.1. Componentes.

El lenguaje se desarrolla sobre tres tipos de componentes:

1) **Operadores**: igual, contenido en, mayor que, empieza por, y, o, no, ...
2) **Operandos**: identifican campos del modelo de datos biológico: plasmid, superfam, protein, ...
3) **Valores**: son las variables contra las que los operadores comparan los operandos.

No existe referencia a las tablas, el modelo es presentado al usuario como una caja negra, dependiendo de la consulta, el sistema es capaz de deducir qué tablas o combinaciones de tablas deben devolver los resultados. Entre otras ventajas esto permite cambiar el modelo subyacente sin cambiar el lenguaje.

En la versión actual tampoco requerimos los campos a devolver del modelo, intentamos deducirlo por la forma de la consulta. En la práctica no está resultando suficiente y vamos a ampliar la DSL para que el usuario pueda explicitar los campos de la consulta.

La adaptación a teclados multidispositivo se apoya en varios criterios de diseño:

1) Todos los campos se consideran textuales: no existen tipos de datos, por tanto no es necesario ni se acepta ningún tipo de comillas o delimitadores de campo (en el modelo actual tampoco contamos con la necesidad de admitir espacios en blanco, dadas las características de la información, por tanto en una extensión del lenguaje tendremos que habilitar algún sistema de delimitado o de definición de espacios). Este hecho también protege de la inyección *SQL*.

2) La mayoría de los operadores (operadores tipo *multi*) actúan contra listas de valores separados por comas (sin paréntesis). Se interpreta que el operador se aplica al operando y valor para cada uno de los valores estableciendo un *OR* lógico entre cada expresión o *AND* en el caso de que el operador sea de caracter negativo (*not equal*). Los operadores que implementan comparaciones aritméticas como $>=$ actúan también sobre listas de valores, por mantener la homogeneidad, pero realmente el valor determinante es sólo uno de ellos, ya que van enlazados con *AND*.

3) Los operandos cuentan con varios alias, algunos tan reducidos como *p* para plásmido, *s* para superfamilia y *m* para módulo funcional.

   Por ejemplo, la expresión:

```
plasmid == KY362373.1, NZ_CP018684.2
```

   se interpreta como:

```
plasmid == KY362373.1 OR plasmid == NZ_CP018684.2
```

4) Se establecen **alias** para todos los operadores, que constan de letras y el punto decimal ., de forma que pueden teclearse con la pantalla flotante por defecto del cualquier móvil.

El lenguaje admite además operaciones lógicas binarias (*and* y *or*) y unarias (*not*) que pueden combinarse en niveles arbitrarios de profundidad.

La traducción a *SQL* de la consulta no genera una sentencia completa sino un inserto que se acopla a una estructura de consulta predefinida donde se informan las cláusulas *SELECT*, *FROM* y *WHERE* y por tanto todas las referencias necesarias al modelo de datos.

Por ejemplo:

```
p = NZ_LT960791.1 | p =\* AZ
```

se transforma en este inserto SQL:

```
(p.plasmid_id LIKE '%NZ_LT960791.1%') OR (p.plasmid_id LIKE 'AZ%')
```

que el proceso completa automáticamente así:

```
SELECT p.plasmid_id, "M-" || module_id AS module_id
FROM plasmid_module p
WHERE (p.plasmid_id LIKE '%NZ_LT960791.1%') OR (p.plasmid_id LIKE 'AZ%')
ORDER BY p.plasmid_id ASC, p.module_id ASC
LIMIT 0, 1000;
```

En la *web app* hemos utilizado el paquete *ace* [16] para el resaltado de código, configurando las reglas particulares de nuestro caso.

Cuadro 4.2: Operadores expandibles de la DSL de consulta.

operador	alias	tipo	expansión	descripción
==	.e.	multi	OR	igual estricto al valor (**e**qual)
=	.i.	multi	OR	contiene el valor (**i**n)
=	.ew.	multi	OR	termina por el valor (**e**nds **w**ith)
=*	.sw.	multi	OR	empieza por el valor (**s**tarts **w**ith)

operador	alias	tipo	expansión	descripción
!==	.ne.	multi	AND	no igual escricto (**n**ot **e**qual)
!=	.ni.	multi	AND	no contiene el valor (**n**ot **i**n)
!*=	.new.	multi	AND	no termina por el valor (**n**ot **e**nds **w**ith)
!=*	.nsw.	multi	AND	no empieza por el valor (**n**ot **s**arts **w**ith)
>	.gt.	multi	OR	mayor que el valor (**g**reat **t**han)
<	.lt.	multi	OR	menor que el valor (**l**ess **t**han)
>=	.ge.	multi	OR	mayor o igual que el valor (**g**reat or **e**qual)
<=	.le.	multi	OR	menor o igual que el valor (**l**ess or **e**qual)

Cuadro 4.3: Operadores lógicos de la DSL de consulta.

operador	alias	tipo	descripción
&	.a.	binario	operador lógico **a**nd
|	.o.	binario	operador lógico **o**r
n	.n.	unario	operador lógico **n**ot

En la figura 4.1 se muestra una sesión de trabajo.

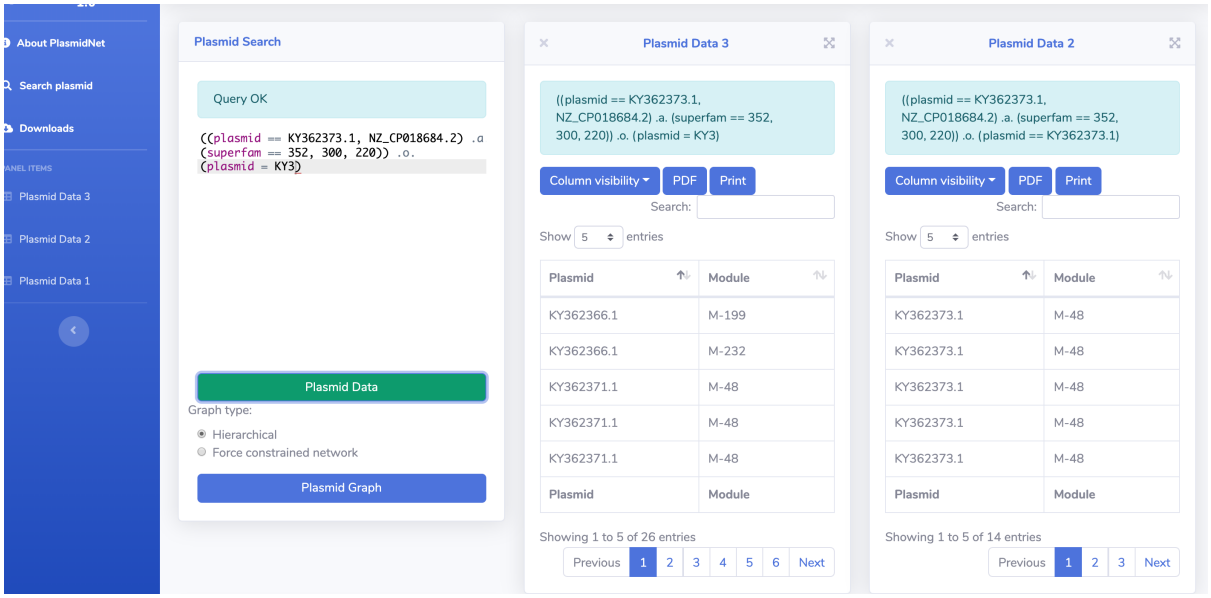

Figura 4.1: Sesión de trabajo de búsquedas

4.5. Carga inicial de la base de datos

La carga inicial de la base de datos de módulos funcionales de plásmidos se ha construido para ser ejecutada desde un flujo orquestado.

Desde el flujo se lanzaría una de las operaciones en secuencia.

La forma de definir el comando es la siguiente:

```
node load_plasmidnet_db --step test [--max_lines number]
```

Donde en el parámetro *step* se le indica nombre del paso de carga a ejecutar:

- create_model
- load_representatives
- load_protein_AA
- load_protein_norep
- load_module_superfam
- load_superfam
- load_plasmid_superfam
- load_plasmid_module

Si queremos ejecutarlos todos, por este orden se indica como *all*. Esta funcionalidad es de interés para pruebas.

En el parámetro *max_lines* le indicamos el máximo número de filas de los ficheros de entrada a cargar en cada tabla. Es de utilidad para las pruebas. Por defecto se cargan todas las filas.

5

Gestor de contenidos

5.1. Planteamiento

La organización del trabajo técnico en torno a plataformas *web* ha venido siguiendo diversas recomendaciones organizativas más o menos atinadas tal y como se comprobaba después en la práctica. En primera aproximación e incluso dentro de empresas de tamaño medio o proyectos sencillos, se consideran necesarios dos perfiles.

1) Experto en software que se encarga de todo lo relacionado con el sistema informático subyacente.
2) Experto en usabilidad que se ocupa del diseño de las interfaces: aspecto, usabilidad, accesibilidad, ...

Desde luego ambos perfiles pueden subdividirse a su vez, dependiendo del tamaño de la empresa o proyecto. No es extraño encontrar un perfil de *creativo* con la responsabilidad de diseñar el aspecto gráfico *look&feel* de la aplicación.

Esta división es la de mínimos porque incluso en un proyecto donde una sola persona aborda todos los aspectos, lo hace sin duda desempeñando estos dos roles. Incluso nos atreveríamos a decir que es mucho más importante en este caso separar al máximo ambos ámbitos.

Veamos los pasos de una 'cadena de montaje' de máximos para contar con una mejor perspectiva. Voy a reducir deliberadamente los aspectos documentales (existen muchos más documentos involucrados):

1) Creación del manual de usuario.
2) Creación de la interfaz en modo *wireframes* (modelo de alambres), mediante la cual reproducimos la interfaz preocupándonos por evidenciar todo su contenido y relaciones entre sus partes, olvidando su aspecto gráfico.
3) Creación de maqueta *web*, semi-operativa.
4) Incorporación de la maqueta a la aplicación final: a) Una parte es administrada en el gestor de contenidos, normalmente elementos estáticos: literales, imágenes, hojas de estilos, *html*. b) Otra parte se incorpora al código con las modificaciones necesarias, normalmente todo

lo relacionado con la dinamización de la interfaz: accesos al modelo de datos, respuestas a eventos, ...

¿Cómo aparece aquí, casi por arte de magia, el mentado gestor de contenidos?. El origen de la necesidad puede deberse a que el administrador de esta herramienta es capaz de introducir cambios sin la intervención de los programadores. O, si yo mismo me encargo de todo, mediante esta herramienta puedo cambiar cosas sin tocar mis programas ni pasar por el inevitable circuito de diseño, construcción, pruebas, implantación y actualización de la documentación. Porque además al *CMS* se le otorgan privilegios en producción: los administradores de sistemas no tienen inconveniente alguno en habilitar rutas de acceso para que deposite sus contenidos en el entorno en cualquier momento. Sin riesgo. O casi. Porque esto tiene mucho que ver son sistemas de alta disponibilidad 24x7.

Pero ¿qué estamos haciendo?, un elemento estático, que a veces es incluso un inserto *javascript* disimulado o no dentro de un fichero *html*, ¿no debería seguir el mismo circuito de desarrollo de software, por lo menos con su batería de pruebas. La respuesta es sí, y por supuesto se hace. Se cuenta habitualmente con un clon del *CMS* en entornos de pruebas donde se verifican los cambios en los elementos estáticos. Una utilidad de migración que proporciona la propia herramienta sirve para actualizar los contenidos en los entornos productivos, normalmente en dos pasos, uno de replicación del contenido en el casi idéntico *CMS* de destino y otro de despliegue de los contenidos para que sean accesibles por el servidor *web*.

Pero la realidad frustra muy fácilmente todas las expectativas. Fijémonos que debemos mantener en le evolución del sistema sincronizados todos estos elementos, y dejo al margen la documentación:

1) Modelo de *wireframes*.
2) Maqueta *web*.
3) Estáticos.
4) Código dinamizador.
5) *CMS* de producción y *CMS* de desarrollo/pruebas.

No voy a entrar en las dificultades que entrañan las diferentes sincronizaciones, pero en esencia es fácil abocarse a diversos escenarios no deseados:

1) El modelo de *wireframes* no se actualiza ni en la primera evolución de la aplicación.
2) La maqueta *web* es más resistente pero termina siguiendo el mismo camino.
3) Los estáticos se impregnan de código *javascript* dinamizador que debería seguir el mismo ciclo de vida que el resto del código, pero no lo hace.
4) Por tanto es más difícil mantener sincronizados los dos repositorios de componentes (el de los programadores y el de los administradores) por lo que mantener un alto flujo de cambios se vuelve cada vez más arriesgado.
5) Los *CMS* de producción y de desarrollo van dejando de parecerse: errores de sincronización no solventados por falta de prioridad, premura por subir los cambios inmediatamente a producción incluso sin pasar por el entorno previo, bien por temas comerciales, bien por incidencias, incapacidades técnicas de la herramienta, bien porque no tiene un sistema de clonado o porque el que tiene no es fácil de entender y de hacer funcionar correctamente...

Pero no sería justo dejar de comentar que el *CMS* también aporta un entorno donde es relativamente fácil reutilizar los componentes en unas y otras páginas, ofrece interfaces para la estructuración de los diferentes portales dela empresa/usuario, mantenimiento de literales, traducciones, facilidades de posicionamiento en buscadores (*SEO+SEM*), etc...

¿Cuál es el enfoque que seguiremos en PlasmidNet?. En nuestro objetivo de abrazar la simplicidad al máximo hemos construido el embrión (*thinCMS*) de lo que podrá ser un *CMS* más elaborado. Hemos conservado únicamente un aspecto que consideramos crucial: separar los roles de diseño del sitio web del rol de programación del mismo. En una frase **la maqueta del sitio es la base de la construcción** y por tanto se mantiene constantemente actualizada.

5.2. Implementación

Todo parte de la maqueta, la versión inicial del portal y las posteriores modificaciones. Cada modificación en un elemento gráfico, hoja de estilos, imagen, etc. . . , se desarrolla sobre la maqueta.

A partir de aquí el flujo de la información es el siguiente:

1) Los archivos de estilos *css* se exportan al entorno del servidor *web* tal cuál están definidos en la maqueta. Esto no es del todo trivial, porque es necesario asegurarse desde el principio que el sistema de nombrado de los elementos *css* no va a colisionar con ninguna librería *javascript* de las librerías que se utilizan o se van a utilizar en programación. Para ello es fundamental prefijar adecuadamente los distintos elementos.
2) El html es fraccionado en un *html* global (de la página; recordemos que estamos en el ámbito de un proyecto de una única página) y diferentes fragmentos de identidad funcional que hemos denominado (artefactos) injertables o *widgets* a lo largo de este documento. Todos estos elementos son almacenados en la base de datos (específica, diferenciada de la base de datos biológica). Los *widgets* son eliminados de la página principal dejando en su lugar puntos de inserción. Para todos los elementos se genera un código de versión que es una reducción criptográfica *md5* del contenido *html*.
3) La aplicación (*web app*) carga la página principal, como si fuera un *html* convencional suministrada por el servidor *web* y carga los injertables desde la base de datos a medida que los va necesitando.

Puntos de inserción, *widgets* y la página como un todo cuentan con un conjunto de etiquetas *tags html* específicas de PlasmidNet, que se prefijan con *pn-* por parte del diseñador de la maqueta y que obedecen a un contrato entre el diseñador y el programador. Estas etiquetas son meras marcas que finalmente ayudan al programa a decidir qué hacer en cada momento: mostrar o no el *widget*, de qué forma, qué datos se muestran. Usando intensivamente estrategias *COC* pueden llegar a automatizarse una buena parte de este contrato, reutilizándose con facilidad partes del código.

La operatividad de la maqueta es laxa, no es una exigencia. Las acciones que no se implementen, basta con que sean documentadas. En realidad deberíamos denominar la maqueta como 'versión estática del sitio *web*'. Sí conviene implementar algunas funciones en respuesta a algunos eventos muy ligadas a la presentación. En nuestro caso, por ejemplo, está implementada la acción del botón superpuesto que nos lleva al inicio de la página. Estas funciones se integran en un *javascript* que debe ser incorporado a la aplicación tal cuál está implementado, sin modificaciones o mediante modificaciones automáticas.

Todos estos acuerdos obedecen únicamente al criterio de que la maqueta sea la base de intercambio de información funcional entre diseñadores y programadores. La semántica de *tags* puede ser más o menos prolija, y debe ir evolucionando hacia una reutilización extrema de código ya existente, pero no es un elemento que deba estar cerrado a priori.

La fragmentación en injertables almacenados en la *bd* junto a su versión calculada nos ofrece una serie de ventajas propias de un gestor de contenidos tradicional y otras propias de PlasmidNet:

1) Los *widgets* pueden ser reutilizados entre páginas o sitios diferentes y editados individualmente.
2) Los *widgets* pueden ser actualizados dinámicamente desde la *web app* si se detecta algún cambio en su versión (se descargan desde la *bd*). En el capítulo *Navegador* explicamos su interpretación desde la *web app*.
3) Pueden versionarse como otros elementos de la aplicación, de forma que un programador tenga acceso a una versión distinta a la que están utilizando los usuarios.

El entorno de administración gráfico que suministran los portales puede implementarse de dos maneras:

1) Administrando los *widgets* en aplicaciones especializadas en la maquetación de sitios *web* sencillos (como *Bootstrap Studio*).
2) Utilizando las utilidades de PlasmidNet.

La aplicación provee un proxy *browsersync* que facilita enormemente las pruebas ya que está configurado para reiniciar automáticamente el navegador ante cambios de *javascript* y *html* y sin necesidad de reinicio para inyectar en caliente los cambios en los *css*.

Un flujo de trabajo de construcción *web* en PlasmidNet es como sigue:

1) Levantar dos servidores *web*, uno con la aplicación y otro con la maqueta, ambos interceptados por *browsersync*.
2) Cualquier cambio en el software la maqueta dispara dos flujos, el refresco del *proxy* de manera que se visualiza inmediatamente y la generación de los *widgets* y la página principal que se distribuye automáticamente al sistema de archivos del servidor *web* de la aplicación.
3) El *proxy* del servidor *web* reacciona a los cambios y es reiniciado automáticamente.

El propio navegador *Chrome* ofrece muchas posibilidades para cambiar elementos *HTML5* al vuelo sobre el propio navegador.

En definitiva, con las herramientas de automatización extrema, entre las que incluimos *browsersync* PlasmidNet proporciona un entorno muy ágil de trabajo que evita la necesidad de adquirir o incorporar nuevos productos.

6

Orquestador distribuido de tareas

6.1. Planteamiento

No es suficiente que PlasmidNet sea capaz de resolver peticiones de usuario de respuesta inmediata. Es cierto que las operaciones de generación de los resultados de una consulta a la base de datos o la construcción de gráficos basados en esos datos pueden resolverse interactivamente con el usuario porque requieren unos pocos segundos de espera. Pero existen solicitudes que inevitablemente conllevarían tiempos de espera inasumibles para una interacción en tiempo real.

Necesitamos que nuestro sistema admita datos que el propio usuario aporte: nuevas secuencias de proteínas, nuevas secuencias de plásmidos, que deben ser analizadas funcionalmente, siguiendo un camino parecido a las secuencias que sirvieron para construir las agrupaciones funcionales originales de la aplicación, basadas en familias, superfamilias y módulos. Se requiere entonces volver a ejecutar procesos de alineamiento y clasificación, como los utilizados para la carga inicial de nuestra base de datos.

Del mismo modo es necesario que el mismo proceso inicial de carga de la base de datos de PlasmidNet sea fácilmente reproducible para adaptarlo con el mínimo esfuerzo a la evolución de las fuentes de información (bases de datos de plásmidos, proteínas, anotaciones) y a la mejora de las herramientas utilizadas (*DFAST*, *MMseqs2*, *HHblits*, ...) o a evoluciones de nuestro propio software de proceso.

Como estos procesos se componen de varios pasos en cada uno de los cuales se realiza una acción determinada y estos pasos han de ejecutarse en un determinado orden, necesitamos una forma de definir estas secuencias y una forma de lanzar ordenadamente su ejecución. Necesitamos un **orquestador de tareas**. También podríamos referirnos a este tipo de sistema como motor de flujos de trabajo, pero preferimos el término, más moderno, de orquestador porque explicita su función central: la coordinación.

Otro aspecto fundamental que queremos tener en cuenta es el de la **reproducibilidad**. No sólo nuestro orquestador debe poseer capacidades ejecutivas, también debe tener memoria. Es crucial contar con un sistema de descripción y configuración de flujos que podamos almacenar como parte de la documentación del proyecto. Si además incluimos uno o más pasos de estos flujos en imágenes *docker*, incluido el propio orquestador, tendríamos todos los elementos necesarios

para reproducir los resultados en cualquier plataforma.

Además, en el servidor web de la *webapp*, necesitamos:

1) Recepción de solicitudes diferidas del usuario, junto a todos los datos relacionados: secuencias, ficheros,...
2) Identificar la solicitud con un código e informarlo al usuario.
3) Que el usuario pueda consultar el estado de su solicitud y consultar los resultados si ésta ha finalizado.
4) Que el sistema informe al usuario cuando la solicitud haya finalizado.

Nuestra primera tentación fue la de integrar una plataforma existente; *NextFlow* [25] parecía una buena opción. Opera sobre una máquina virtual de *java*, utiliza el lenguaje *groovy* y aporta una *DSL* para la definición de los flujos de trabajo. Pero la herramienta es compleja y ofrece mucho más de lo que en principio necesitamos: atentábamos inmediatamente contra el principio de homogeneidad: no solo precisábamos de otra plataforma, sino también de todo un entorno *java* para poder ejecutarla. El empaquetado mediante contenedores de este entorno facilitaría las cosas, pero ¿y si reutilizábamos de alguna forma lo que ya teníamos desarrollado en *nodejs*?. ¿Por qué no desarrollar las comunicaciones entre los diferentes contenedores implicados en el flujo reutilizando la capa de servicios *REST* que hemos implementado?.

6.2. Definiciones

Nodo PlasmidNet o simplemente **nodo**, se refiere una instancia de servidor *web* PlasmidNet. Como tal escucha en un puerto determinado, y normalmente está desplegado sobre el sistema anfitrión como contenedor *docker*.

Flujo de trabajo o simplemente **flujo** o *pipeline* que es el término utilizado en el desarrollo. El flujo de trabajo es el conjunto de tareas a realizar para obtener un determinado resultado e incluye las dependencias de ejecución entre las mismas. Por tanto incluye una *especificación de flujo*.

Paso de ejecución, o simplemente **paso**, o *step* que es el término utilizado en el desarrollo. Constituye el elemento mínimo de ejecución. Es el elemento mínimo visto desde la coordinación, porque algunos de estos pasos despliegan internamente a su vez otro flujo interno, como en los provistos por *DFAST* o *MMseqs2* en sus imágenes *docker*, donde se encadena la ejecución de varios trabajos.

Una **solicitud de usuario**, desencadena la ejecución de un flujo (*pipeline*), es decir, la ejecución de una **tarea de usuario**, o simplemente **tarea** (*task* como se denomina en el código). Cada tarea lleva asociada una especificación de flujo de trabajo (*pipeline*) y consta de pasos (*steps*) de ejecución.

El término tarea para referirnos a una operación solicitada por un usuario es un tanto desafortunado ya que también denominamos tareas a las diferentes operaciones publicadas sobre *gulp* (donde nos hemos dejado influenciar por la propia terminología de la plataforma *gulp task*). Incluso nos planteamos inicialmente utilizar el propio *gulp* para desarrollar los flujos. Esperamos que no cause confusión, y lo vamos a incluir como aspecto a mejorar en futuras refactorizaciones del código.

6.3. Roles

Un nodo PlasmidNet puede desempeñar varios roles en el sistema de orquestación:

1) Gestor de solicitudes de usuario, proveyendo un conjunto de servicios *REST* para recoger la solicitud, requerir la solicitud al nodo coordinador e informarle del resultado de la misma. Le llamaremos **gestor** (*manager*).
2) Proveedor de pasos. Lo llamaremos **ejecutor** (*executor*).
3) Coordinador de flujo. Lo denominaremos **coordinador** (*orchestrator*).

En la figura 6.1 se muestra la arquitectura lógica.

En la realización física de la misma podemos construir topologías en las que cada uno de los nodos se ubica en un servidor *web* distinto, topologías donde todo los nodos lógicos están ubicados en el mismo servidor *web* y topologías con grados de dispersión intermedios entre los dos anteriores.

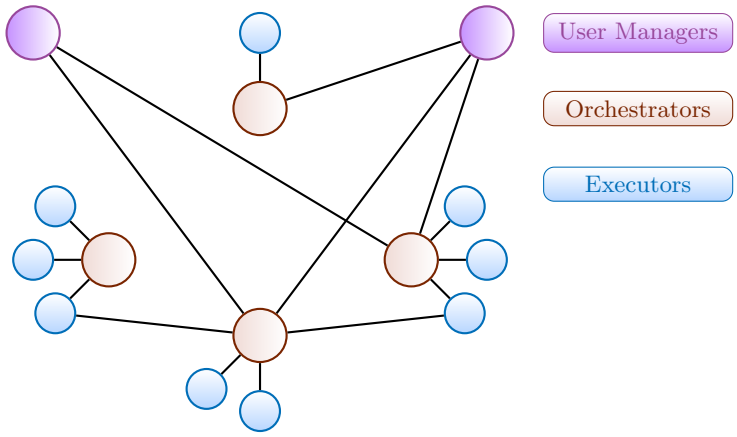

Figura 6.1: Topología lógica de orquestación

6.4. Casos de uso

Vamos a describir el funcionamiento del sistema en base a la redacción de casos de uso, entrando progresivamente en los detalles y en las bifurcaciones. Nos centraremos en las casuísticas más complejas.

Nuestros actores serán los siguientes: 1) *usuario* de la aplicación. 2) *aplicación* cliente PlasmidNet (*web app*) que está utilizando el usuario en el navegador. 3) nodos plasmidnet, tal y como los definimos anteriormente: *gestor*, *coordinador*, *ejecutor*.

El rol de gestor, lo desempeña el servidor de la *webapp*. No queremos denominarlo *servidor* porque implicaría otro tipo de rol que poseen todos los nodos participantes*.

El contexto de los casos tratados aquí es el siguiente:

El usuario se encuentra utilizando la *web app* y necesita obtener la clasificación funcional de un plásmido que no está previamente clasificado en la base de datos de PlasmidNet. Esta clasificación va a requerir al sistema la ejecución del flujo de trabajo P que consta de dos pasos.

6.4.1. Ejecución de una tarea

1) El usuario solicita a la aplicación la clasificación del plásmido que ha proporcionado.

2) La aplicación solicita al gestor la ejecución de una nueva tarea de flujo *P*.
3) El gestor solicita al coordinador la ejecución de una nueva tarea de flujo *P*.
    - El gestor lanza una solicitud *REST* al servicio */task_ create* del coordinador.
4) El coordinador construye la tarea (y sus pasos) a partir de la información maestra del flujo *P* y solicita al ejecutor del primero de los pasos, la ejecución del mismo.
    - El coordinador ejecuta la función *task_ create*, que finaliza realizando una solicitud *REST* al servicio /step_do del ejecutor.
5) El ejecutor ejecuta el paso solicitado.
    - El ejecutor lanza la función *step_ do*.
6) El paso termina correctamente.
7) El ejecutor informa al coordinador de la finalización correcta del paso.
    - El ejecutor lanza el servicio *REST /task_ update* contra el coordinador.
8) El coordinador actualiza la información de la tarea y solicita al ejecutor del segundo paso la ejecución del mismo.
    - El coordinador lanza la función *task_ update* que a su vez lanza *task_ continue* que decide el siguiente paso a ejecutar y llama a *step_ start* que construye el nuevo paso y que a través de *step_ do* finaliza lanzando el servicio *REST* /step_do del ejecutor.
9) Como el paso (5).
10) Como el paso (6).
11) Como el paso (7).
12) El coordinador actualiza la información de la tarea y como se trata del último paso informa al gestor que la tarea ha terminado.

No hemos todavía decidido si vamos a utilizar alguna estrategia proactiva para informar al usuario del estado de una solicitud. Probablemente la *web app* o el *service worker* realizarán *polling* contra el servidor para actualizar el área de avisos. También valoraremos la nueva funcionalidad que *Chrome* en su versión 80 acaba de implementar (*Periodic Background Sync*) que permite a las aplicaciones registrar tareas que se ejecutarán a intervalos periódicos con conectividad de red. También debemos revisar el estado actual de las tecnologías *push* que de momento hemos descartado sobre todo a causa de la dispersión de plataformas.

Debemos tener en cuenta que lo que estamos describiendo son roles que pueden estar desempeñados por el mismo nodo. El sistema es suficientemente flexible para configurarse de la forma que se considere más conveniente en cada caso. En el caso más extremo, los nodos PlasmidNet gestor, coordinador y ejecutor se ejcutan sobre una misma máquina por lo que sólo tendríamos un servidor. Podría pensarse que esta estructura será la ideal para el entorno de pruebas, pero no es necesario, resulta muy fácil automatizar una tarea *gulp* para levantar todos los nodos que hagan falta, en la misma o en diferentes máquinas físicas o virtuales, provistas del mismo sistema operativo o no (siempre que dispongan de la plataforma *docker* instalada).

Pero necesitamos profundizar con más detalle en lo que ocurre en cada uno de los pasos.

Vamos a desarrollar el paso *9* como un caso de uso en sí mismo, cuando el ejecutor se dispone a ejecutar el paso. Como actores intervienen: el coordinador, el ejecutor del paso, los ejecutores de los pasos previos que han generado los ficheros que son necesarios para la ejecución del paso. A estos últimos les llamaremos ejecutores predecesores o, para abreviar más *predecesores*. Como en este caso sólo tenemos dos pasos, nos referiremos a un único *predecesor*, sin perder generalidad. Vamos a considerar de momento que el predecesor está en otra máquina y lo que

es más importante, que no comparte con el ejecutor el sistema de archivos donde almacena los datos (podría hacerlo por *nfs* por ejemplo).

Veremos que el paso fluirá por una serie de estados. Las transiciones están definidas en un objeto de la aplicación:

```
const STATE_STEP_TRANSITIONS = {
  'INIT' : STATE_MD5_DOWNLOAD_PENDING,
  'MD5_DOWNLOAD__FINISHED': STATE_MD5_CHK_PENDING,
  'MD5_CHK__FINISHED': STATE_MD5_COMPARE_PENDING,
  'MD5_COMPARE__FINISHED': STATE_DOWNLOAD_PENDING,
  'DOWNLOAD__FINISHED': STATE_MD5_DOWN_PENDING,
  'MD5_DOWN__FINISHED': STATE_MD5_VERIFY_PENDING,
  'MD5_VERIFY__FINISHED': STATE_EXEC_PENDING,
  'EXEC__FINISHED': STATE_MD5_OUT_PENDING
}
```

En la situación inicial de la ejecución el estado del paso es *INIT*, y que su siguiente estado es STATE_MD5_DOWNLOAD_PENDING.

6.4.2. Ejecución de un paso

1) El ejecutor solicita al predecesor los ficheros *md5* correspondientes a los ficheros que va a necesitar descargar posteriormente (ficheros que son los de salida del predecesor).
   - Para ello el ejecutor actualiza el estado del paso a *STATE_MD5_DOWNLOAD_PENDING* y lanza el servicio REST */step_md5_download* para cada uno de los ficheros que necesita
2) El ejecutor descarga los ficheros *md5* necesarios solicitándolos al predecesor. La descarga se efectúa mediante *http*. Estos ficheros contienen los resúmenes *md5* de los ficheros de entrada necesarios para la ejecución del paso.
3) El ejecutor comprueba que todos los ficheros se han descargado correctamente.
   - El ejecutor actualiza el estado el paso a *MD5_DOWNLOAD_FINISHED*.
4) El ejecutor verifica que no tiene los ficheros *md5* calculados por él disponibles en su sistema de archivos (podrían estarlo si estuviéramos realizando una nueva ejecución) y solicita al predecesor los ficheros de entrada (no los *md5*, que acaba de descargar).
5) El predecesor sirve los ficheros al ejecutor.
6) El ejecutor verifica que la descarga ha sido correcta y lanza el cálculo de los *md5* de los ficheros recibidos.
   - El estado del paso cambia a *STATE_MD5_DOWN_PENDING* en el momento en que termina la descarga.
7) El ejecutor verifica que el cálculo de los *md5* ha finalizado correctamente y lanza la comparación de los *md5* calculados con los *md5* recibidos.
   - El estado del paso cambia a *STATE_MD5_VERIFY_PENDING*.
8) El ejecutor verifica que la comparación no arroja diferencias y lanza la ejecución del paso.
   - El estado del paso cambia a *STATE_EXEC_PENDING*.

9) El ejecutor verifica que la ejecución ha terminado sin errores e informa al coordinador de que el paso ha terminado correctamente.
    - El estado del paso cambia a *EXEC_ FINISHED*.

Errores en la ejecución

Si en cualquiera de las verificaciones el ejecutor detecta un error, el ejecutor informa al coordinador de la contingencia, junto a toda la información relevante del error (estado donde se produce, mensaje de sistema,...) que es devuelta en la misma estructura *json* del paso.

Ficheros ya recibidos

Si los ficheros de entrada ya están disponibles en los directorios del ejecutor, el número de pasos se reduce:

1) El ejecutor descarga igualmente los ficheros *md5* del predecesor, con el fin de comprobar la validez de los ficheros de entrada. Si además tiene disponibles los ficheros *md5* locales, los compara con ellos. En caso contrario los vuelve a calcular. Si coinciden procede a ejecutar el paso.
2) Si en la situación de inicio anterior los *md5* de alguno de los ficheros no coincide, procede a la descarga del mismo y de su *md5* y después a las comparaciones y prosigue como en los casos anteriores.

El predecesor es el propio ejecutor

Luego, los ficheros ya se han recibido, porque en realidad no ha sido necesario moverlos. Estaríamos exactamente en el caso 1 anterior porque se vuelven a verificar los *md5*. El sistema es compatible con la casuística más trivial en que todos los pasos de la tarea son ejecutados sobre el mismo sistema de archivos. Aún bajo estas circunstancias la orquestación puede estar distribuida en varios nodos que entonces estarían compartiendo ese sistema de archivos, o al menos el directorio de datos de ejecución.

6.5. Definición de flujos

Se sustenta en objetos *javascript*, que definen cinco entidades:

1) *pipeline* : flujo.
2) *step*: paso de ejecución.
3) *provider*: proveedor de ejecución.
4) *input*: ficheros de entrada asociados a un *step*.
5) *output*: ficheros de salida asociados a un *step*.

Vamos explicar el procedimiento de definición sobre una serie de ejemplos de complejidad creciente.

Flujo elemental

Veamos cómo se define una *pipeline* sencilla:

```
let provider_url = { default : `http://${HOST}:9091`,
                     plasmid01 : `http://${HOST}:9091`,
                     plasmid02 : `http://${HOST}:9092`,
                     plasmid03 : `http://${HOST}:9093`
                   }
pipelines['test'] = {
  desc : "Obtain modules for a plasmid set",
  steps : [
    {name: 'plas01_test', command : 'plas01_test.pl', providers : ['plasmid01'
    ↪ ]},
    {name: 'plas02_test', command : 'plas02_test.pl', providers : ['plasmid02']}
  ]
}
```

Primeramente se declaran las *URL* de los proveedores. Cada proveedor es un nodo PlasmidNet. Después se define la *pipeline* asignándole un identificador 'test'. Se incluyen la descripción *desc* y la matriz de pasos *steps*. Cada paso consta de un identificador *name*, el nombre el programa a ejecutar *command* y la matriz de proveedores capaces de ejecutar el paso *providers*.

El orden en que están los pasos declarados en la matriz *steps* es el orden de ejecución.

Ahora podemos definir los ficheros de entrada y salida para cada *step.name*.

```
outputs['plas01_test'] = ["test01.txt"];
inputs['plas01_test'] = [];
inputs['plas02_test'] = ["test01.txt"];
outputs['plas02_test'] = ["centos7.vdi", "test01.txt", "centos7_bis.vdi", "test.
    ↪ jpg"];
```

Esta definición es global, todos los *steps* que tengan el mismo nombre, aunque estén en *pipelines* distintas, reutilizan estas definiciones, que también pueden especificarse a nivel de *step* dentro de una *pipeline*.

Esta *pipeline* es interpretada por el sistema de esta forma:

Se debe ejecutar primero el paso de nombre *plas01_ test* lanzando el programa perl *plas01_ test.pl* en el nodo *plasmid01* si están disponibles el fichero *test01.txt*.

En cuanto termine, debe lanzarse el paso *plas02_ test* que consiste en la ejecución del programa perl *plas02_ test.pl* en el nodo *plasmid02*.

Flujo de cómputo de los módulos funcionales de plásmidos

Veamos los primeros *steps* que utilizan exhaustivamente todos los campos disponibles para las definiciones:

```
pipelines['plasmid_modules'] = {
  desc : "Plasmid modules pipeline",
  steps : [
    {
      name: 'plas00', command: 'plas00_plasmid_gi__PLSDB_extract.tcsh',
      desc: 'Extract tsv of plamids from file downloaded from PLSDB',
      providers : ['plasmid01'],
```

```
      cmd_data : ['__TASK_ID__',
                  '30', //plasmid_number
                  '2018_09_14'] //plsdb_version
    },
    {
      name: 'plas01', command: 'plas01_plasmid_protein__NCBI_download.pl',
      desc: 'Obtain from NCBI the sequences of proteins related to plasmids',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    ... más pasos
    {
      pipeline: 'mmseq2_create_cluster'
    },
    {//step 11 begins
      pipeline: 'mmseq2_update_cluster'
    }
  ]
}
```

En el primer paso definimos una matriz *cmd_ data*. En ella y por este orden, se declara el valor de los parámetros a pasar al *command* (en este caso se trata de un *shell script*).

Dentro de las matrices de parámetros aparece un ubicuo ***TASK_ ID***. Esta etiqueta se refiere al código de solicitud del usuario de la *web app* (coincide con el código que asignamos a las solicitudes *http* y que el sistema refleja en todos los *logs*). Este código se le transmite al usuario e identifica a la tarea solicitada en todas nuestras comunicaciones con él. Ese código también se informa en los *logs* de ejecución de las *pipelines*, con lo que garantizamos la máxima trazabilidad de las operaciones.

Recursividad

Los últimos pasos de la *pipeline* anterior referencian otra entidad *pipeline*. Estas pipelines las definimos después:

```
pipelines['mmseq2_update_cluster'] = {
  desc : "MMSEQ2 update cluster",
  steps : [
    {//11
      name: 'plas11_mmseqs__createdb',
      command: 'mmseqs',
      cmd_data: 'createdb __DIR__/plasmid_protein__combined.fasta __DIR__/
   ↪ plasmid_protein_ini.DB --max-seq-len __max-seq-len__',
      inputs: ['plasmid_protein__combined.fasta'],
      outputs: ['plasmid_protein_ini.DB'],
      providers: ['plasmid03'],
      subs: {'__max-seq-len__': 2600000}
    },
    {//12
      name: 'plas12_mmseqs__update_cluster', command: '',
      desc: 'MMSEQ2 update cluster',
      providers: ['plasmid03'],
```

```
    cmd_data: `
      pwd
      whoami
      rm  __TASK_DIR__/plasmid_protein_new.*
      rm -rf __TASK_DIR__/tmp
      mmseqs clusterupdate \
      __PIPE_DIR__/plasmid_protein.DB \
      __TASK_DIR__/plasmid_protein_ini.DB \
      __PIPE_DIR__/plasmid_protein.CLU \
      __TASK_DIR__/plasmid_protein_new.DB \
      __TASK_DIR__/plasmid_protein_new.CLU \
      __TASK_DIR__/tmp -v 2 -c 0.8 --min-seq-id 0.7 --threads 1 --max-seq-len
↪ 2600000`,
    inputs: ['__PIPE_DIR__/plasmid_protein.DB',
             'plasmid_protein_ini.DB',
             '__PIPE_DIR__/plasmid_protein.CLU'],
    outputs: ['plasmid_protein_new.DB', 'plasmid_protein_new.CLU']
  },
  ...
```

O sea, una *pipeline* puede actuar como si fuese una subrutina dentro de otra *pipeline*. La estructura de inclusión está implementada recursivamente con lo que podemos incrustar varios niveles.

Esta característica nos sirve para incrementar el grado de reutilización y la legibilidad *pipelines* complejas.

En esta *pipeline* contemplamos otra posibilidad del orquestador: la ejecución de lanzar una batería de comandos del sistema operativo. La batería se informa en *cmd_ data* y *command* se deja en blanco o no se informa. Alternativamente puede desplegarse en el nodo de ejecución un *shell script* que implemente esta batería. Aquí hemos decidido informarlo de esta manera porque así tenemos visibilidad directa de la ejecución que estamos solicitando del programa *MMSEQS2*, pero es una cuestión de conveniencia.

Por otra parte vemos que aparecen nuevas etiquetas:

1) *PIPE_ DIR*: hace referencia al subdirectorio de la *pipeline* relativo al punto de montaje del directorio de datos. Es un subdirectorio de *DATA* con el mismo nombre que la *pipeline*, que contiene objetos (ficheros, bases de datos) que van a ser utilizados por cualquier tarea de usuario asociada a una *pipeline* del mismo nombre.
2) *TASK_ DIR*: hace referencia al subdirectorio específico de la tarea relativo al punto de montaje del directorio de datos. Es otro subdirectorio de *DATA* nombrado mediante el **TASK_ID**.

También aparece un nuevo objeto en el *step*: *subs* que nos permite implementar sustituciones o etiquetas personalizadas en el ámbito del *step*. En este caso declaramos la etiqueta ***max-seq-len*** y le asociamos el valor *2600000*. La etiqueta se utiliza como parámetro del mismo nombre en el comando. Antes de lanzar la ejecución del comando, el sistema realiza la sustitución de todas las etiquetas, personalizadas y genéricas.

En la figura 6.2 se muestra una estructura típica del directorio *DATA* con las carpetas correspondientes a las tareas del usuario:

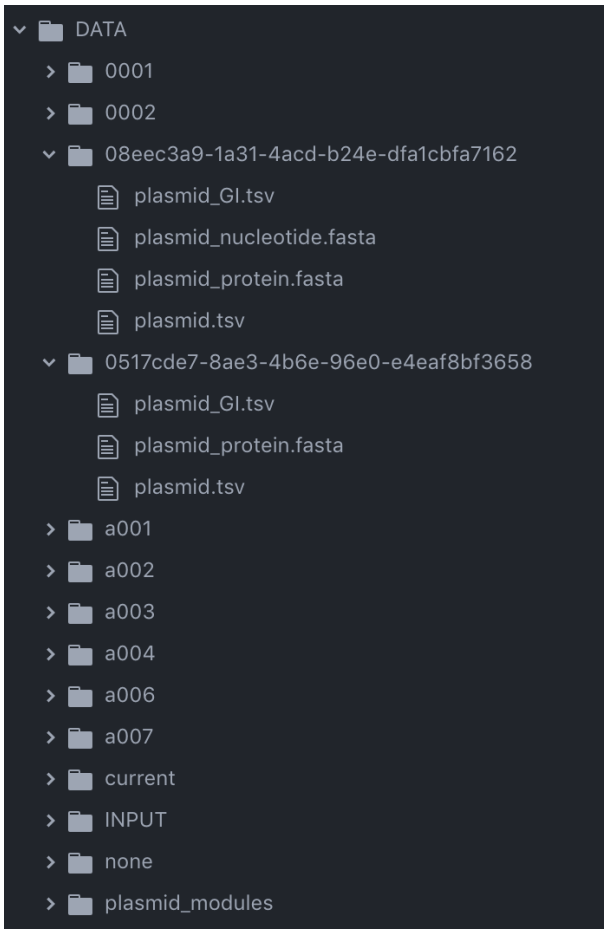

Figura 6.2: Directorios de ejecución de flujos

6.6. Base de datos de orquestación

Se han desarrollado tres soportes en bases de datos intercambiables.

1) **Sqlite3**. Relacional SQL: empotrada en la aplicación.
2) **Lokijs**. NoSQL, en memoria con serialización a disco: empotrada en la aplicación.
3) **Redis**. No SQL, en memoria con serialización a disco, sobre servidor externo.

La variable de entorno MODEL *sirve para especificar el tipo, en tiempo de arranque de la aplicación.*

El modelo compatible estructuralmente con los tres tipos de *bd* (forzando *sqlite* a un modelo fuertemente desnormalizado).

- *tasks*
  - *task_ id.* referencia al código de tarea de usuario.
  - *task.* estructura JSON que incluye el identificador de *pipeline.*
- *steps*
  - *task_ id.* referencia al código de tarea de usuario.
  - *step_ id.* secuencial que indica el orden de ejecución.

- *step* estructura JSON que incluye todos los campos definidos para el *step* del mismo nombre en la *pipeline* asociada a la *task*.

API compatible con los tres modelos:

Cuadro 6.1: API de acceso a la base de datos de orquestación.

función	descripción
start	Arrancar la base de datos
upsert_task	Actualizar tarea o insertar tarea nueva
upsert_step	Actualizar paso o insertar nuevo paso
update	Actualizar/insertar tarea y paso
get_tasks	Obtener todas las tareas
get_task	Obtener tarea por clave de tarea
get_steps_all	Obtener todos los pasos
get_step	Obtener paso por clave
get_steps	Obtener todos los pasos de una tarea
exists_task	Comprobar la existencia de una tarea por clave

6.7. Comunicaciones

Todas las comunicaciones entre los nodos PlasmidNet que participan en la ejecución de las tareas de usuario se realiza vía protocolo *http(s)*, soportadas por servicios *REST-json* definidos de forma similar a los servicios que soportan la *web app*, reutilizando la misma infraestructura.

Las *URL* de los servicios tienen esta forma:

1) */plas/pipeline/task_[servicio]*. Servicios relacionados con la tarea de usuario: creación, cambios de estado, información.

2) */plas/pipeline/step_[servicio]*. Servicios relacionados con los pasos de la tarea de usuario: creación, cambios de estado, información.

7

Automatización Extrema

Todas las tareas de mantenimiento de la aplicación deben ejecutarse bien automáticamente bien con el menor esfuerzo manual posible del usuario.

Las tareas las agrupamos en tres bloques:

1. Generación documental.
2. Gestión de contenedores (*docker*).
3. Desarrollo y pruebas de software.

7.1. Generación Documental

7.1.1. Primeros pasos

Cuando afrontamos por primera vez la tarea de generación de la documentación identificamos que debíamos construir varios tipos de documentos:

1. Documento relacionado con el trabajo de fin de master, exportable a *pdf*, esto es, imprimible. Contamos con una plantilla en *latex*.

2. Documento para la presentación del proyecto en la defensa pública del mismo.

3. Documentación detallada del código. *Acordémonos que en nuestra deriva exploratoria decidimos aplazar la documentación esperando a contar con una base de código suficientemente estable.*

En el documento de fin de master podíamos distinguir tres partes: una **documentación de alto nivel** que constituiría el núcleo de este trabajo, un anexo **manual de programador** y un anexo **manual de usuario**.

Decidimos que el manual del programador sería del nivel semántico de la documentación detallada del código.

Siguiendo las prácticas habituales decidimos que la documentación detallada debía acompañar al código, es decir debía redactarse sobre los propios ficheros fuente y debíamos idear alguna forma de transformar automáticamente esta información en un formato *latex* integrable en el anexo correspondiente del TFM, de acuerdo a las exigencias de la Automatización Extrema.

La documentación de alto nivel al principio previmos redactarla sobre un fichero *latex*, fuera del contexto del código fuente. La presentación de diapositivas la redactaríamos en un momento posterior.

La conversión de la documentación de detalle, forzosamente textual porque procedería de comentarios insertos en el propio código fuente, la realizaríamos con el programa **pandoc** [9], un transformador universal de formatos.

La información de detalle la redactaríamos en el formato típico dictado por herramientas como *Doxygen*.

Llenaríamos el código con fragmentos de texto como este:

```
/// @function Sync.load
  /// Loads a resource
  /// @param {String} id the GUID of the resource
  /// @param {Function} success callback to be executed with the data on
  ↪ succeed
  /// @param {Function} error callback to be executed with error description
  ↪ in case of failure
  /// Loads a resource
```

Aquí comenzaron nuestras dudas ¿de verdad queremos documentar prolijamente lo que es obvio en el código?. Podríamos adoptar incluso la postura extrema de pensar que el código es autoexplicativo y que por tanto se documenta a sí mismo. Y si el código no se entiende por sí mismo es que está mal programado, o estructurado y debería revisarse. Esta es la perspectiva que la metodología *Agile* trajo asociada al resto de sus regalos.

Muchos técnicos, comenzaron a replantearse este tema seriamente. Es de destacar este artículo de Bran Selic [39], del que incluyo una cita muy reveladora y que anticipa el enfoque que hemos seguido en el proyecto:

"Clearly, documenting software at the code level is foolish; this simply results in the nightmare of trying to keep duplicated information consistent (code comments often fall into this useless category). Instead, we need design documentation at a higher level of abstraction, stripped of unnecessary technological detail and closely coupled to application concepts and requirements. These should incorporate design rationale, including descriptions of rejected design alternatives. These are, in fact, architectural specifications: technology-independent descriptions of the higher-level structure and behavior of systems along with key design principles"

7.1.2. Enfoque inicial

Nos debemos centrar en el objetivo: **que el código se entienda**, que otro programador pueda hacerse cargo de él porque comprende cómo funciona y los condicionantes o restricciones que han llevado a su compañero a optar por tal o cual estrategia. Y aprovechar un hecho relevante: el código se explica a sí mismo, si bien no totalmente, sí en una gran medida que hemos de aprovechar para no repetirnos (e incurrir en los desperdicios que *lean* nos conmina a evitar).

¿Cuál es entonces el nivel semántico a seguir en un diseño detallado?. Creemos que deben de quedar claros al menos los siguientes puntos:

1. El enfoque global enmarcándolo en otras alternativas que se hayan desechado, ya que esto puede ayudar no sólo a la comprensión, también a afrontar posibles mejoras o corregir errores.
2. Clarificar el objetivo de cada función, priorizando las de mayor carga algorítmica.
3. Ser prolijos en los aspectos algorítmicos que parecen más enrevesados.
4. No comentar nada que parezca obvio en el propio código.

Todos estos puntos nos llevaron a la necesidad de una redacción más literaria, menos técnica, sin miedo a resultar excesiva, orientada a un fin: la comprensión. Además nos dimos cuenta que la frontera entre el diseño de alto nivel y el diseño detallado se difuminaba un poco, y que, tal vez, debíamos incluir también los comentarios de diseño de alto nivel acompañando al propio código, también cerca de él porque también contribuían a su comprensión.

Entonces nos topamos de lleno con las ideas de Donald Knuth, el creador de *tex*, que propone un paradigma de documentación-programación denominado *Literate Programming* [34].

Knuth concibe un fichero de código (*WEB* en su terminología) como fuente de dos procesos de sistema: el que genera la documentación (proceso *weave*) y el que compila o interpreta el código para poder ser ejecutado (proceso *tangle*).

El fichero *WEB* que utiliza Knuth como ejemplo alberga muy pocas líneas de código inmersas en una estructura documental muy desarrollada. No pretendíamos llegar tan lejos, nos queríamos encaminar hacia la consecución de código documentado, no tanto a un documento que ilustra código a modo de un *jupyter notebook*, o *R-Markdown*, pero no cabe duda que estábamos en una línea ya explorada anteriormente y no nos sería difícil encontrar herramientas específicas de nuestro entorno **nodejs**.

Así fue, probamos **docco** [18] y **groc** [35], dos proyectos nodejs capaces de extraer comentarios en **markdown** contenidos en el código fuente y exportar una página *html* con dos columnas, una con el código y otra con los comentarios asociados, en una vista paralela 7.1

No era suficiente, aunque coincidíamos en utilizar **markdown**, el mismo lenguaje de marcado utilizado por *jupyter* o *R*, de sintaxis muy sencilla y altamente legible sin ningún tipo de formato, texto plano, que *pandoc* sería capaz de transformar a multitud de formatos, entre ellos los que necesitábamos.

Uno de los problemas del desarrollo de *docco* es que sólo interpreta en *javascript* los comentarios del tipo '\' de forma que un fichero comentado por *docco* tiene este aspecto, que resulta bastante incómodo de leer (y escribir):

```
// **Docco** is a quick-and-dirty documentation generator, written in
// [Literate CoffeeScript](http://coffeescript.org/#literate).
// It produces an HTML document that displays your comments intermingled with
  ↪ your
// code. All prose is passed through
```

Nosotros preferíamos limitarnos a los bloques de código multilínea que además de facilitar la legibilidad nos permiten reservar los comentarios monolínea, tipo // en javascript, para aquellos que no se reflejan en la documentación, más que como parte del mismo código.

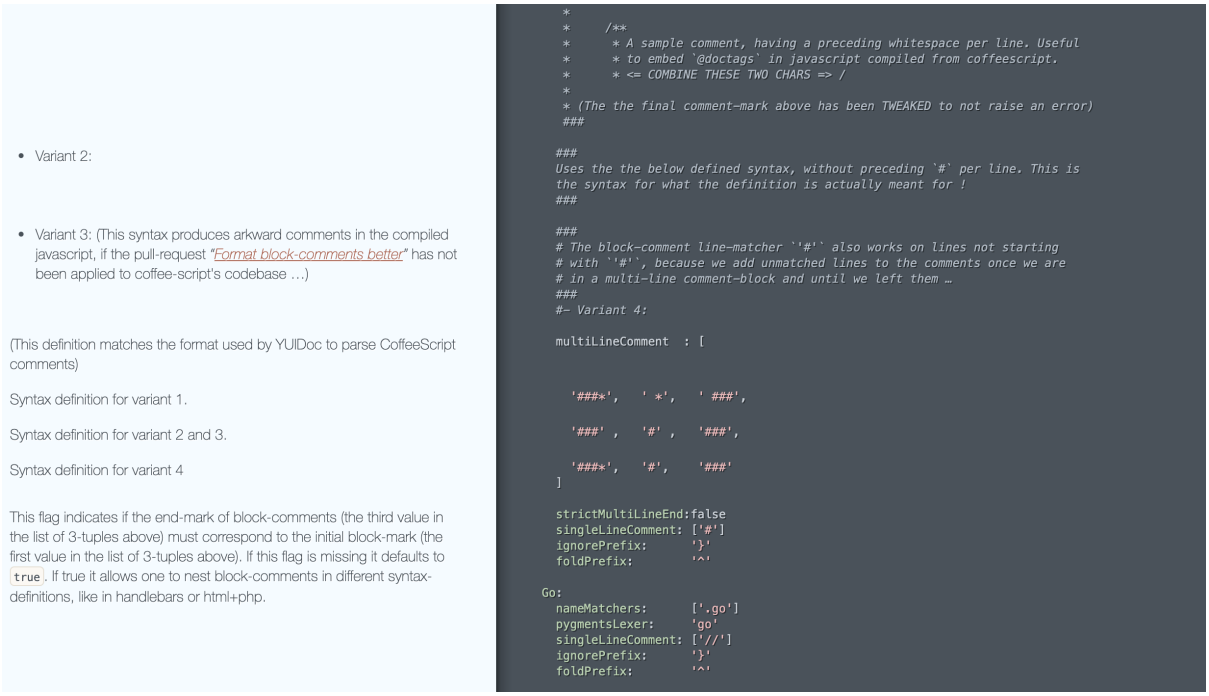

Figura 7.1: Documentación generada por groc

Comprobamos que utilizar *markdown* como formato de origen contaba con más de un adepto. Dennis Tenen y Grant Wythoff desarrollan con bastante detalle esta idea [43], proponiendo utilizar *pandoc* como conversor.

Agarwal nos muestra en [17] un caso de uso de utilización de *pandoc* para la elaboración de *pdf*.

¿Íbamos por buen camino?. Sí, pero *pandoc* tal y como se nos presentaba, no era suficiente, ni siquiera a través de la modificación de sus plantillas y filtros.

Porque también nos planteábamos generar *html* para su publicación en *web*, con un índice de contenido adecuado y con capacidad para ser leído desde cualquier dispositivo y en cualquier tamaño de pantalla. Necesitábamos de alguna forma adaptar la salida de *pandoc* a un ambiente *bootstrap* como el que habíamos utilizado en la aplicación.

Nos encontramos con que conseguir en *pandoc* una salida *latex* adecuada para la impresión, era en algunos casos incompatible con obtener desde *pandoc* una salida *html* adecuada para su visualización en navegador. En el manual del programador se explican alguna de las inconsistencias.

En definitiva, sobre estos sólidos cimientos conceptuales: documentación en texto plano acompañando al código y *literate programming*, construimos nuestro sistema documental.

7.1.3. Implementación

Nuestra estrategia se basa en la definición de dos tipos de ámbitos, el semántico y el estructural. El primero tiene que ver con el contenido del documento y el segundo con el continente, con la forma en la que queremos presentarlo.

El nivel semántico está relacionado con la forma en que debemos redactar, qué terminología debemos incluir, a qué nivel de detalle queremos profundizar. Detectamos cinco niveles semánticos principales:

1. Diseño de alto nivel. Es el nivel correspondiente a un análisis funcional en la metodología tradicional. Consiste en una descripción lógica del funcionamiento del sistema, sin descender al detalle máximo de implementación. Aunque en el diseño funcional tradicional al documento de arquitectura es otro documento, en PlasmidNet lo incluimos dentro de este apartado. *Estas mismas palabras que estoy redactando sirven de ejemplo de este nivel semántico.* En la interpretación que hemos realizado de este tipo de diseño hemos considerado muy conveniente enmarcar donde fuera necesario cada una de las decisiones tomadas frente a otras alternativas de mercado. No esperéis un gran nivel de exhaustividad, no se trata de realizar un estado del arte completo, sino de que sirva de marco para facilitar la comprensión de la argumentación. No existe por tanto separación neta entre estado del arte e implementación, según vamos mostrando las soluciones, vamos realizando las comparaciones pertinentes. Este diseño corresponde al núcleo del documento del Proyecto de Fin de Master.

2. Diseño detallado o manual de programación. Este es el nivel de máximo detalle, es decir incluye el código fuente y las explicaciones sobre el mismo donde creemos que el código no basta por sí solo para explicarse.

3. Manual de usuario. En este manual describimos cómo se invocan las diferentes acciones o comandos que permite ejecutar el sistema. Es un manual orientado a la operación sobre el terminal: comandos disponibles y parámetros.

4. Presentación. Este es sin duda el nivel de máximo resumen, va orientado a presentar el proyecto de forma ágil resaltando sus partes fundamentales.

5. Glosario. En este nivel se redacta la lista de términos y acrónimos de la aplicación.

En el ámbito estructural hemos construido estos tipos de salidas:

1. Formato base: *markdown*. Este es un formato de salida manual, aunque se generan *markdown* intermedios agregados para combinar los *markdown* de los diferentes documentos que intervienen en un documento agregado. Este formato, aunque sirva de origen de conversión para los demás, debe ser un fin en sí mismo, al fin y al cabo, al estar cerca del código es el que va a ser más consultado, presumiblemente . Debemos garantizar también que se lee sin dificultad, que podemos escribirlo con comodidad desde la misma herramienta con la que escribimos el código fuente de la aplicación. Cada editor o *IDE* puede necesitar de ciertos plugins y configuraciones adicionales. En el caso de **ATOM** que es el editor que utilizamos, instalamos 'Markdown Writer' y 'Markdown Preview Plus', además configuramos los editores de *javascript* y *markdown* para que automáticamente continuen una línea en la siguiente (*wrapping*) cuando se supere un determinado número de caracteres. 'Markdown Preview Plus' habilita una conversión en *html* al vuelo para que tengamos una referencia aproximada de cómo se mostraría en los otros formatos de salida.

2. Formato web: *HTML5*. Permite visualizar la documentación en el entorno de un navegador. Existen dos subformatos web implementados: el relacionado con el nivel semántico de presentación (*revealjs*) y el relacionado con el resto de los niveles semánticos.

3. Formato de impresión: *latex*. Es el formato intermedio para la generación de salidas impresas o documentos *pdf*. También existen implementadas dos tipos de salidas *pdf*, la relacionada con el nivel semántico presentación (*beamer*) y la relacionada con el agregado *latex*, que ahora veremos.

En el ámbito estructural distinguimos también lo que llamamos tipologías:

1. No agregada. Es el de los ficheros nativos procedentes de la conversión desde *pandoc*.

2. Agregada. Hemos construido dos agregaciones de documentos base que agrupan en un mismo "documento", documentos individuales procedentes de niveles semánticos distintos. El *agregado documental tipo webapp* configura una página web mediante la cual publicamos los ficheros *html* generados por *pandoc* para cada uno de los niveles semánticos. El agregado documental tipo *latex* agrega los documentos latex generados por *pandoc* de todos los niveles semánticos menos el de presentación.

   Esta tipología nos provee de documentos que funcionalmente van un paso más allá de *pandoc*. Es probable que programando en los entresijos de *pandoc* pueda construirse también a base de filtros, plantillas y hojas de estilo, pero requeriría penetrar en las profundidades del lenguaje *Haskell*, y dispersaríamos inevitablemente la plataforma, en contra del principio de homogeneidad. Por ello desarrollamos en *javascript* sobre *nodejs* todas las rutinas necesarias para hacer esto posible.

La documentación se redacta junto al código indicando el nivel semántico y el idioma. Los bloques de texto pueden ser compartidos por varios niveles semánticos, como este mismo (*slides* y *high_ level*).

En la figura 7.2 se aprecian varias regiones de documentación adicionales: *glossary* y *detail*.

Figura 7.2: Redactando documentación sobre fichero fuente en ATOM

Si nuestra tarea en curso es de documentación, podemos decirle a nuestro editor (**ATOM**) que resalte el texto como si fuera *markdown*.

Y que muestre una visualización en HTML como vemos en la figura 7.3

Esta visualización un tanto forzada no es el verdadero *markdown* que generamos en el primer paso de las conversiones documentales, y se resaltan de forma extraña los inicios y fin de comentarios que estamos utilizando como parte de nuestro *pre-markdown*, sin embargo hemos comprobado que este procedimiento no es del todo inadecuado.

Sería posible crear un lenguaje en *ATOM* que interpretara directamente nuestro *pre-markdown* y lo resaltara de forma precisa.

Otros editores de texto requerirían un tratamiento diferente pero conceptualmente similar.

Figura 7.3: Redactando documentación sobre fichero fuente resaltando markdown en ATOM

Otra utilidad importantísima presente en todos los editores es el colapsado de código. Esto nos habilita una visión resumida del fichero, ahora un tanto complicado ya que contiene el código fuente y todos los comentarios de todos los niveles semánticos e idioma, como se muestra en la figura 7.4

Figura 7.4: Visualización de código colapsado en ATOM

Existe un plugin para *ATOM* [28] que permite colapsar sólo los comentarios. La utilidad de este plugin en este escenario es extraordinaria, tal y como hemos podido comprobar durante este proyecto.

7.1.4. Bibliografía

La mejor automatización bibliográfica la hemos conseguido con Zotero [15] y su plugin de *Chrome*, desde donde pueden generarse directamente referencias bibliográficas de una página,

que pueden ser editadas para completarlas de forma muy sencilla dentro de la aplicación.

Desde Zotero exportamos a un fichero en formato *biblatex* que utilizamos como referencia en *markdown* para enlazar a los diferentes documentos.

Este fichero lo tenemos incluido en la plantilla *tex* agregada con lo cual conseguimos completar el circuito.

Generación de manual de uso para la línea de comandos

El usuario necesita una referencia rápida de las tareas *gulp* disponibles en cada ámbito de la aplicación: *doc*, *docker*, *pipelines*, ... Para conseguirlo sería muy importante generarla automáticamente a partir de alguno de los niveles semánticos, sin crear otro nivel adicional y por tanto sin generar duplicidades. El nivel semántico más adecuado es sin duda alguna el nivel *user*, o sea, el correspondiente al Manual de Usuario.

Para lograrlo hemos construido un conversor capaz de extraer la información del mismo *markdown* generado. La misma función de conversión es una tarea más del entorno *gulp*. En la figura 7.5 se muestra el aspecto de la ayuda sobre un terminal:

```
man scope doc lang es

Extrae los comentarios markdown del fichero file de entrada, relacionados con doctype y lang.
doc_extract_md --file [gulpfile.js] --doc_type [high_level, slides, detail, user] --lang [es, en]

Convierte a latex el fichero markdown asociado a file mediante pandoc.
El nivel de los capítulos y secciones se ajusta en base al parámetro shift que se envía a pandoc sin modificaciones.
doc_md_to_tex --shift [0] --file [gulpfile.js]

Convierte a slides reveal.js el fichero markdown asociado a file mediante pandoc.
doc_md_to_slides --file [gulpfile.js]

Convierte a slides beamer el fichero markdown asociado a file mediante pandoc.
doc_md_to_slides_pdf --file [gulpfile.js]

Convierte a html el fichero markdown asociado a file mediante pandoc.
doc_md_to_html --shift [0, 1, -1, ...] --file [gulpfile.js]

Genera el pdf global desde el agregado latex vía pdflatex.
El pdf global se encuentra sobre una plantilla latex externa al sistema pandoc.
doc_latex_to_pdf --shift [0, 1, -1, ...] --file [gulpfile.js]

Convierte un fichero latex tikz en pdf y svg, que serán utilizados respectivamente por los ficheros de salida latex y html.
tikz_to_svg --file [gulpfile.js]

Genera la webapp html a partir del tipo de documento doc_type: detail, user, highlevel, slides, e idioma lang: es, en.
doc_html_to_webapp --doc_type [high_level, slides, detail, user] --lang [es, en]

Genera toda la documentación asociada al proyecto. Primero extrae los markdown de los ficheros fuente. También de los fuentes que son markdown puros, después lanza las transform
ciones pandoc y por último el agregado para la webapp y la generación del pdf agregado.
doc_html_to_webapp --doc_type [high_level, slides, detail, user] --lang [es, en]

Imprime por consola la ayuda asociada a las tareas gulp relacionadas comn un determinado 'scope' (doc, docker, ...).
Por defecto imprime las tareas gulp de documentación.
man --scope [doc, docker, ...] --lang [es, en]

[16:18:37] Finished 'man' after 4.81 ms
```

Figura 7.5: Manual de uso sobre un terminal

Multiidioma

No queríamos renunciar a un sistema que nos permitiera redactar en Español facilitándonos la traducción a otras lenguas como el Inglés. De hecho al más bajo nivel (código fuente, comentarios del código fuente no gestionados por las transformaciones documentales), hemos redactado en esta lengua internacional porque no tenía sentido ni era posible (código fuente) mantener un bilingüismo.

La traducción no puede realizarse automáticamente, quiero decir, no es recomendable. No es un problema derivado sólo de la incapacidad de los traductores para, digamos, acertar con las correspondencias. Es sobre todo un problema de nuestra redacción en Español, muy poco cuidadosa a la hora de construir las frases, redacción que roza muchas veces el filo de la coherencia interna en nuestro propio idioma, y la rompe por completo al trasladarse al Inglés. Poco cuidadosa y versada. Necesitaríamos más conocimientos para saber redactar en nuestro idioma de forma que al menos un 95 % del texto pudiera ser traducido sin problemas por un traductor automático.

Creemos que esto es alcanzable ya con la tecnología actual en el marco de la documentación técnica y científica. Ahora lo vamos a abordar de manera supervisada.

En la figura 7.6 se muestra una sesión de trabajo. La pantalla la tenemos dividida: a la izquierda el editor y a la derecha el navegador *Chrome* sobre la página de *Google Translator*.

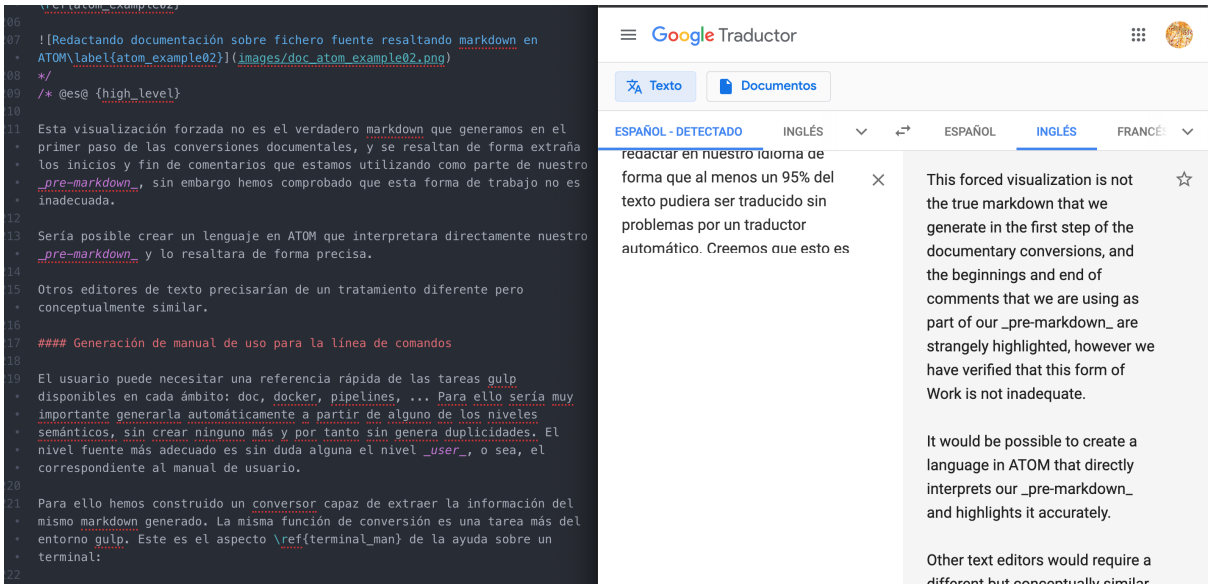

Figura 7.6: Sesión de trabajo de traducción

El procedimiento es muy sencillo:

1. Seleccionamos en el editor el bloque de texto que deseamos traducir.
2. *Copy* en el editor y *Paste* en el traductor.
3. Traducir y realizar el *Copy+Paste* inverso sobre el editor en un nuevo bloque, ahora asociado al idioma Inglés.
4. Corregir manualmente los errores. Pueden corregirse de dos formas: modificando la prosa original en Español para que la traducción sea más precisa o modificar la versión traducida. En la primera opción es más idóneo corregir sobre la ventana de Google y sustituir sobre el editor ambas versiones: la original y la traducida.

7.1.5. Conclusiones

Hemos construido un sistema multi-documental integrado en el código. Mediante la definición de niveles semánticos es posible redactar paralelamente sobre los ficheros fuentes de los diversos documentos que contendrán la información de nuestro proyecto, reutilizando en muchos casos secciones de texto comunes.

Esta aproximación es novedosa hasta donde sabemos. A partir de nuestro propio uso en este proyecto creemos que puede ser un camino a seguir con un amplio espectro de mejora:

1. Extensión a otros lenguajes de programación.
2. Resaltado multi-lenguaje: código fuente y *markdown*.
3. Mejora de la *web app* documental.
4. Mejoras en las plantillas en general, en particular *beamer* y *revealjs*.
5. Traducción totalmente automática.
6. Otras mejoras en los editores (colapso de todos los comentarios o todo el código, ...).
7. Ampliar las capacidades de configuración del sistema.

7.2. Pruebas y despliegue de software

Uno de los aspectos más importantes de la automatización es la capacidad de generar los paquetes de software para su distribución en el sistema de archivos del servidor *web* y que esta distribución se desencadene automáticamente en el entorno de desarrollo en cuanto se detecte un cambio en los ficheros fuentes.

Si además conseguimos que el servidor *web* dispare en el navegador una actualización automática, el proceso de codificación y pruebas se acelera notablemente.

Hemos automatizado dos conjuntos de ficheros de software:

1) Automatizaciones de gestión de software de cliente (*web app*).
2) Automatizaciones de gestión de software de servidor.

Las automatizaciones que implican actuaciones sobre los servidores *web* se apoyan en dos tipos de utilidades:

1) *Filewatchers*. Observadores de eventos de modificación de ficheros en el sistema de archivos que disparan los procesos de distribución correspondientes. Son proporcionados por *gulp* [7].
2) Proxy *browsersync* [2]. Utilidad que nos permite trasladar las acciones de teclado desde un navegador cliente de la aplicación a los demás, de forma que pueden probarse sincronizadamente varias versiones de navegadores desde dispositivos distintos.

Automatizaciones cliente. Distribución.

1) Concatenado de ficheros *javascript*: ficheros de la aplicación.
2) Minimización (*uglify*). Dificulta la comprensión del código a un tercero y reduce el tamaño de los ficheros:
   - Sustituyendo nombres de variable por nombres más cortos.
   - Eliminando tabuladores, espacios y saltos de línea.
   - Eliminando comentarios.
3) Transformación de ficheros *sass* en *css* y concatenación.
4) Minimización de *css*.
5) Generación de mapas de código fuente (entorno de desarrollo).
6) Actualización de versión en el *CMS*.

Automatizaciones cliente. Pruebas.

1) Arranque y parada de proxy *browsersync*.
2) Desencadenamiento de acciones ante cambios en ficheros fuente:
   - Despliegue automático.
   - Actualización de *css* en la página.
   - Reinicio automático del navegador ante cambios de *javascript* o *html*.

Automatizaciones de servidor

1) Eliminación de comentarios para entorno de producción.
2) Generación de versión (subversion) en el *CMS* para los módulos *javascript*. Necesaria para la carga dinámica de módulos en el servidor.
3) Distribución de código sobre los directorios de archivos del servidor *web*.

7.3. Despliegue basado en contenedores

Las distribución de software y el control de versiones, son tareas capitales en el flujo de vida de un proyecto informático, que requieren la máxima precisión y atención al detalle. Es relativamente habitual que surjan problemas inesperados debido a las diferencias entre los entornos de desarrollo o pruebas y los entornos productivos. Estos problemas pueden ser minimizados si podemos empaquetar al máximo nuestra aplicación junto a todos los componentes de software de los que depende, de forma que no tengamos colisiones con otras versiones de los mismos distribuidos en el nodo productivo anfitrión. Las colisiones pueden darse tanto en instalación como en ejecución. Además, siendo estrictos, las pruebas realizadas dejarían de tener validez completa si alteramos algo del software subyacente.

La independencia o aislamiento de nuestra aplicación respecto a su entorno están relacionados con otro aspecto clave: la seguridad. Cuanto más aislado esté nuestra aplicación menos posibilidades existen de afectar al anfitrión si alguien aprovecha una vulnerabilidad de nuestro sistema. Y al contrario, la explotación de una vulnerabilidad en el anfitrión tendría menos posibilidades de afectarnos. Obviamente siempre serán necesarios ámbitos compartidos: red, sistema de archivos,..., pero debemos tratar de reducir la superficie de contacto.

Una tercer aspecto y no menos importante en el trabajo científico es la repetibilidad de resultados. Si almacenamos adecuadamente las imágenes que hemos utilizado para una investigación concreta, podremos en el futuro ser capaces de reproducir la parte informática del experimento sin temor a que una actualización posterior de uno de los paquetes nos altere los resultados.

Distribución *nodejs*

Es cierto que el haber elegido *nodejs* como plataforma de desarrollo podría asegurarnos en gran medida este objetivo, y tal vez llegaríamos a conformarnos con la distribución de los ficheros de la aplicación junto a todas sus librerías *javascript* de *nodejs*. Habría que asegurar como mínimo que la máquina anfitriona ejecuta la misma versión de *nodejs* que la utilizada para el desarrollo y pruebas de nuestra aplicación.

Las capacidades de aislamiento de *nodejs* son las siguientes:

1) Todo el código fuente de la aplicación, *javascript* y por supuesto estáticos (*css*, *html*), es compatible con cualquier sistema operativo que ejecute la misma versión de *nodejs*. Esto nos permite, por una parte, desarrollar y probar bajo *macos* o *linux ubuntu* y distribuir en entornos *linux redhat* o *windows* sin pruebas adicionales confiando en esa compatibilidad.
2) Podríamos desplegar un *tar* con todo el software de la aplicación y de sus dependencias o bien utilizar el fichero *package.json* que nos ha generado *nodejs* en el entorno de desarrollo y donde *npm* ha dejado constancia de todos los paquetes y versiones utilizadas. Ejecutando un comando *npm install* en un directorio vacío de despliegue donde hayamos depositado este fichero, se descargarían automáticamente todas las dependencias. Después de esto sólo faltaría desplegar el software de la aplicación.

Hasta aquí todo parece bastante aceptable, y como primera aproximación podríamos conformarnos, pero si nuestra aplicación o sus paquetes de software asociados incluyen código que debe ser compilado para cada sistema operativo (en nuestro caso por ejemplo la base de datos *sqlite*, que es un programa escrito en *C*), no podemos desplegar el software completo, la instalación debe realizarse con el segundo método: *npm* y *package.json*.

Siendo estrictos, el ejecutable de estos programas nativos no es exactamente el mismo que el binario utilizado en nuestras pruebas. Nos podemos fiar, pero no es el mismo.

Y siguiendo estos criterios, tampoco deberíamos ser absolutamente confidentes si en el entorno productivo nos esperan binarios de *nodejs* diferentes a los utilizados en las pruebas (aunque sean de la misma versión).

Distribución mediante máquina virtual

Un aislamiento más estricto y una mayor fiabilidad de nuestras pruebas se alcanzarían si utilizamos una máquina virtual como contenedora de nuestra aplicación y de todas sus dependencias. Por ejemplo, podríamos construir una máquina virtual en cualquier distribución de linux y desplegar la máquina completa, que es tan sencillo como distribuir los discos virtuales y los ficheros de configuración. Cada nodo anfitrión debería contar con un sistema de ejecución de máquinas virtuales, lo más parecido posible al que utilizamos en desarrollo. En cualquier caso ya estamos seguros que todos los componentes de la aplicación, binarios o no, son exactamente los que hemos utilizado en el desarrollo.

La virtualización ofrece un buen conjunto de ventajas adicionales frente a un despliegue físico, y algunas desventajas relacionadas con los requerimientos de memoria, disco y capacidad de proceso añadidos que se le exigen al anfitrión, pero sobre todo una desventaja crucial que nos ha hecho rechazar esta estrategia porque complicaría enormemente nuestro sistema: la máquina virtual debe estar correctamente configurada para garantizar la estabilidad, el rendimiento y la seguridad de nuestra aplicación. Y configurar correctamente una máquina virtual es al menos tan complejo como configurar una máquina real. Y nosotros sólo necesitamos desplegar una aplicación. Estaríamos atentando contra los principios fundamentales de economía que exigimos desde el principio a este proyecto.

Jaulas *chroot* y *knoppix*

Pero, ¿realmente es un requerimiento que PlasmidNet pueda ser ejecutado en cualquier sistema operativo?. La respuesta es que no. En producción nos esperan nodos uniformemente *Centos 7* y contamos con máquinas de desarrollo en estos entornos y en cualquier caso sí podemos instalar una máquina virtual *Centos* en cualquier sistema operativo sin costes de licencias (utilizando *VirtualBox* de *Oracle*, por ejemplo).

Entonces, distribuyendo todo el directorio de la aplicación y sus dependencias tenemos solucionado el despliegue. Para evitar colisiones con otras posibles instalaciones de *nodejs* quizá en versiones distintas a la utilizada por nosotros, basta con desplegar también adicionalmente el ejecutable *node* que hemos utilizado en nuestro desarrollo y asegurarnos de que la aplicación utilice este y no otro. No es necesario desplegar nada más. Y estamos con un nivel de aislamiento máximo respecto al entorno.

Pero ¿y el entorno con respecto a nosotros?. Nuestra aplicación tendría acceso por defecto a todo el sistema de archivos del anfitrión. Sí, podemos protegerlo con una configuración adecuada de políticas de usuarios y grupos, pero físicamente están accesibles, lo que en caso de una brecha de seguridad en nuestro sistema comprometería con relativa facilidad al sistema anfitrión. Y viceversa, como ya hemos comentado.

¿Con qué métodos de aislamiento contamos?. Precisamente en entornos *linux* podemos enjaular la parte del sistema de archivos en los que se despliega la aplicación, de forma que sólo es capaz de percibir este conjunto de directorios, como si la máquina física estuviera constituida únicamente por esa estructura de archivos. Esto se consigue mediante el comando *chroot* que modifica el raíz del sistema de archivos antes de invocar a nuestra aplicación. Sólo queda habilitar explícitamente la parte del sistema de archivos del anfitrión que debe ser compartida con nuestra sistema. Con esto ya hemos reducido la superficie de contacto.

Históricamente las estrategias de aislamiento basadas en sistemas de archivos independientes tuvo su máximo desarrollo en las distribuciones *linux* que se ejecutaban desde un sistema de almacenamiento de sólo lectura: *CD* o *DVD*.

La distribución *knoppix* [33] se apoya en imágenes comprimidas de sistemas de archivos (*cloop*) que permiten empaquetar todo un sistema en un único fichero (como los tradicionales archivos-sistemas *loop* pero comprimidos). Un módulo específico del *kernel* se utiliza para montar, desmontar y leer los datos de este fichero que es presentado a las aplicaciones como si se tratase de un sistema de archivos convencional. La escritura tienen lugar automáticamente en la memoria del sistema, por lo que cualquier cambio se termina perdiendo. Sin embargo la distribución provee de otro módulo multicapa que permite superponer sobre las imágenes *cloop* una nueva capa de escritura con persistencia a disco. Este sistema mezcla efectivamente la capa de lectura y escritura que son presentadas como un sistema de archivos único a usuarios y aplicaciones. Estos módulos del *kernel* se denominan módulos de unión y existen diversas implementaciones (*unionfs*, *aufs*, *overlayfs*). La escritura se basa en muchos casos en una estrategia *cow* (*copy on write*), mediante la cual, cuando un fichero sufre una modificación es copiado automáticamente a la capa de escritura, desde donde será proporcionado en accesos sucesivos. La estrategia permite además agregar un número indeterminado de capas superpuestas.

No sería difícil utilizar estas tecnologías en nuestra plataforma *linux CentOS*, empaquetando nuestra aplicación y dependencias como un archivo *cloop* y montándolo para su ejecución unido a una capa de escritura. Incluso sería factible idear una estrategia multicapa, con niveles incrementales de lectura que se superpondrían a la capa base original y que contendrían sólo los archivos modificados por una versión posterior, evitando así distribuir toda la aplicación de nuevo. Si fuese necesario compartir archivos (bases de datos, *logs*) con el anfitrión desde una carpeta específica, fuera del entramado de uniones, bastaría otorgar permisos de montado de esa carpeta al usuario de ejecución de la aplicación.

Contenedores

Sin duda esta última hubiera sido una opción satisfactoria hace unos años, pero ahora contamos con un conjunto de desarrollos que facilitan la implementación.

Nosotros hemos decidido utilizar uno de ellos *docker* [8], que se apoya en los mismos fundamentos que hemos descrito respecto a *knoppix* pero provee de un conjunto de herramientas (*docker desktop*) multiplataforma para generar, montar, ejecutar y monitorizar las imágenes de la aplicación. La generación puede hacerse desde cero o tomando de referencia otra imagen. La unión de una imagen con su capa de escritura se denomina contenedor y es el concepto que da nombre a toda esta línea tecnológica. Un contenedor puede incluir varias imágenes diferentes y existen muchos otros aspectos recogidos en la literatura, pero lo que hemos utilizado en PlasmidNet abarca estos tipos de operaciones:

1) Creación de imágenes.
2) Compartir directorios con el anfitrión: *log*, datos, código fuente.
3) Compartir las interfaces de red con el anfitrión.

Antes de adentrarnos en estos aspectos, es importante destacar que hemos recuperado la capacidad multiplataforma como efecto colateral beneficioso. Las mismas imágenes *docker* pueden ser ejecutadas desde *windows*, *macos* o cualquier sistema *linux* que tengan instalada la plataforma *docker*. Esto es así porque las imágenes se apoyan siempre en sistemas *linux* reducidos basadas en distribuciones existentes recortadas al máximo posible. Porque *docker* lanza las aplicaciones de los contenedores siempre sobre un sistema *linux*, el del propio anfitrión en *linux*, o en *windows* o *macos* sobre una máquina virtual reducida (*hypervisor*). Por tanto, podemos desarrollar y probar en cualquier sistema operativo y trasladar después las imágenes de nuestra aplicación junto a sus dependencias y la distribución reducida de *linux* que hayamos escogido.

Parece entonces que volvemos a necesitar, como en la estrategia de máquinas virtuales, la configuración de un sistema operativo. Sin embargo estos sistemas son muy sencillos, carecen

de los mecanismos de arranque de servicios y apenas aportan un conjunto de comandos básicos de *unix*, a menudo una instalación de *busybox*. *Docker* nos provee de ficheros de configuración para abordar con facilidad estas tareas, además de la reutilización de imágenes ya existentes preconfiguradas. Es necesario prestar sobre todo atención a la configuración del usuario de la aplicación y de sus permisos.

7.3.1. Creación de imágenes

Docker proporciona varios métodos. Nosotros hemos escogido la configuración basada en *Dockerfiles*, que no son más que ficheros donde podemos ejecutar comandos del sistema operativo. Estos comandos se lanzan en tiempo de creación de la imagen, no en tiempo de ejecución de la aplicación.

Las tareas principales que necesitamos acometer en la creación son las siguientes:

1) Importar la imagen del sistema operativo *linux* que vamos a tomar de base. Nosotros hemos elegido *alpine*.
2) Crear el usuario de la aplicación (hemos decidido *plasmiduser*), sin *password*. No es conveniente operar como *root*, aunque se trate de un mero contenedor.
3) Instalar adicionalmente el software de *nodejs* y *npm* de la versión que decidamos.
4) Instalar en el directorio de aplicación escogido los paquetes requeridos de *nodejs*. Se hace en dos pasos:
   - Copiar el fichero *package.json* en el directorio de la aplicación.
   - Ejecutar *npm install* desde este directorio.
5) Desplegar el software de la aplicación
6) A todos los ficheros y directorios de la aplicación asignar el usuario de la aplicación como propietario.

Servidor web para la *web app*

Este es el fichero *Dockerfile* para un entorno en el que sólo queremos arrancar un nodo de tipo servidor de *web app*. Hemos escogido como imagen de base la de una distribución de linux *alpine* con *node.10.16.3* preinstalado. El mismo entorno *docker desktop* nos conecta con el repositorio de imágenes y descarga automáticamente la imagen referida en *FROM*. A partir de ahí seguimos los pasos indicados más arriba. La instalación de *python* es imprescindible para el sistema de compilación de componentes nativas en *nodejs*, así como *make*, *g++* y *gcc*, pero son instalados con la opción *virtual* del gestor de paquetes, de forma que son borrados después de ejecutado el comando *RUN* y tras la última línea con *apk*:

```
FROM node:10.16.3-alpine
RUN addgroup -g 6000 plasmiduser  && adduser -u 6000 -G plasmiduser -s /bin/sh -
    ↪ D plasmiduser \
    && mkdir -p /home/plasmiduser/app/node_modules && chown -R plasmiduser:
    ↪ plasmiduser /home/plasmiduser/app
WORKDIR /home/plasmiduser/app
COPY --chown=plasmiduser:plasmiduser package.json .
RUN apk --no-cache --virtual build-dependencies add \
    python \
    make \
    g++ \
    gcc \
```

```
    && npm install \
    && apk del build-dependencies
COPY --chown=plasmiduser:plasmiduser . .
USER plasmiduser
ENV PATH="node_modules/.bin:${PATH}"
```

Servidor de web de orquestación

Para construir un nodo del tipo servidor de orquestacón, tendríamos que incluir la instalación de *perl* y *tcsh* con el fin de que pueda ejecutar los pasos de PlasmidNet que utilizan estos lenguajes.

Las ejecuciones necesarias en *Dockerfile* son de esta forma:

```
...
RUN apk --no-cache --virtual build-dependencies add \
    python \
    make \
    g++ \
    gcc \
    && apk --no-cache add tcsh perl \
    && npm install \
    && apk del build-dependencies

RUN apk --no-cache add perl-log-log4perl
```

Si quisiéramos un servidor web plenipotenciario, capaz de ser utilizado como servidor de la *web app*, nodo de orquestación y proveedor de servicios *REST* utilizaríamos también esta última versión de la *Dockerfile*.

Servidor web totipotente

Recordemos que podemos siempre configurar el servidor web a partir de variables de entorno como *PN_CONFIG_PORT* o *MODEL*, y estas variables pueden ser definidas tanto en tiempo de instalación como en tiempo de ejecución. Sin embargo es recomendable en producción ajustar las capacidades del sistema a cada caso con el fin de reducir la superficie de vulnerabilidad. En la versión actual de PlasmidNet no hemos llevado este criterio hasta sus últimas consecuencias, pero debemos avanzar en un sistema de configuración más granulado, sin perder la capacidad de definir un *docker* de desarrollo con todas sus capacidades, como este, donde instalamos herramientas como *git* o el propio *docker*:

```
RUN apk --no-cache --virtual build-dependencies add \
    python \
    make \
    g++ \
    gcc \
    && apk --no-cache add tcsh perl \
    && npm install \
    && apk del build-dependencies
RUN apk --no-cache add perl-log-log4perl
RUN apk --no-cache add docker
RUN addgroup plasmiduser docker
```

```
RUN apk --no-cache add git
RUN apk --no-cache add mc
```

Docker dentro de *docker*

Instalamos el cliente *docker* porque queremos que el entorno de desarrollo empaquetado como *docker* también sea capaz de construir y ejecutar *dockers*. Porque consideramos muy importante que el propio entorno de desarrollo pueda ser distribuido como imagen de *docker* para asegurarnos que cualquier miembro del equipo trabaja con las mismas versiones, independientemente de su sistema operativo de trabajo.

Las instalaciones de *docker* no pueden ejecutar *docker* dentro de *docker*, salvo con una serie de estratagemas no recomendadas por los propios desarrolladores del producto, que entre otras inconveniencias comprometen la seguridad del sistema. Pero no lo necesitamos, basta que seamos capaces de gestionar los *docker* del anfitrión, y esto se consigue compartiendo el *socket* del servicio *docker* desde el anfitrión. Así:

```
let param_string = `run -itd \
...
-v /var/run/docker.sock:/var/run/docker.sock \
...
${image} sh`;
```

Docker ajenos

Como pasos de orquestados necesitamos a veces ejecutar software de terceros. Una posibilidad es incluir la instalación dentro de un *docker* especializado, junto a nuestro software de servidor *web*. Una alternativa, y la que hemos escogido, es utilizar los propios *docker* que proprocionan los desarrolladores de las diferentes herramientas, práctica cada vez más habitual. La gran ventaja es que se nos ofrece un paquete ya probado con la instalación precisa de todas sus dependencias. Esto lo hemos realizado para las siguientes herramientas bioinformáticas utilizadas para construir la base de datos de plásmidos:

1) **DFAST**. Anotaciones automáticas [42].
2) **MMSEQS2**. Comparación y agrupamientos de secuencias de proteínas [41].

Este tipo de *dockers* debemos incluirlos como pasos orquestables. Como hemos visto en este subsistema de flujos, nuestro objetivo es que todas las comunicaciones se realicen mediante *http(s)* entre sus nodos, o sea, no nos planteamos que desde los anfitriones se lancen pasos como la ejecución de otros dockers. Tampoco queremos utilizar en un entorno productivo una aproximación *docker dentro de docker*. Por tanto debemos incluir en los *docker* de terceros lo necesario para ejecutar nuestro servidor *web* de orquestación.

Así hemos resuelto la creación del *docker DFAST*:

```
FROM plasmidnet_image:v.2019.10 AS plasmidnet
FROM quay.io/biocontainers/dfast:1.2.3--h8b12597_2
RUN addgroup -g 6000 plasmiduser  && adduser -u 6000 -G plasmiduser -s /bin/sh -
    ↪ D plasmiduser \
    && mkdir -p /home/plasmiduser/app/node_modules && chown -R plasmiduser:
    ↪ plasmiduser /home/plasmiduser/app
```

```
WORKDIR /home/plasmiduser/app
COPY --from=plasmidnet /home/plasmiduser/app .
COPY --from=plasmidnet /usr/local/bin/node /usr/local/bin
...
```

Y así hemos resuelto la creación del *docker MMSEQS2*:

```
FROM plasmidnet_glibc_image:v.2019.10 AS plasmidnet
FROM soedinglab/mmseqs2:latest
RUN addgroup --gid 6000 plasmiduser  && adduser --uid 6000 --gid 6000 --shell /
    ↪ bin/sh --disabled-password plasmiduser \
  && mkdir -p /home/plasmiduser/app/node_modules \
  && chown -R plasmiduser:plasmiduser /home/plasmiduser/app
WORKDIR /home/plasmiduser/app
COPY --from=plasmidnet /home/plasmiduser/app .
COPY --from=plasmidnet /usr/local/bin/node /usr/local/bin
...
```

La estrategia central es fusionar los archivos de nuestra imagen plasmidnet con la imagen proporcionada por el proveedor de la herramienta. Además aprovechamos para crear un usuario *no root* con el que ejecutar los procesos (estas imágenes vienen todas configuradas con usuarios *root*).

Como puede comprobarse en uno de los casos partimos de una imagen *plasmidnet_ glibc*. Esto es así porque contamos con dos tipos de distribuciones de *linux* base, las que utilizan la librería de funciones estándar de sistema *linux glibc* que son la mayoría y las que utilizan, como *alpine linux*, la más ligera *uClibc*, diseñada para sistemas empotrados. Ambas son incompatibles, de tal forma que los ejecutables de un tipo de distribución no funcionan en la otra.

Nuestra imagen *alpine* de la aplicación la hemos utilizado para el caso de *DFAST*, que también utiliza una distribución *uClibc*. Pero para el caso de *MMSEQS2* debimos crear una imagen de la aplicación alternativa basada en *glibc*:

```
FROM  node:10.17-slim
RUN addgroup --gid 6000 plasmiduser  && adduser --uid 6000 --gid 6000 --shell /
    ↪ bin/sh --disabled-password plasmiduser\
    && mkdir -p /home/plasmiduser/app/node_modules && chown -R plasmiduser:
    ↪ plasmiduser /home/plasmiduser/app
WORKDIR /home/plasmiduser/app
COPY --chown=plasmiduser:plasmiduser package.json .

RUN apt-get update
RUN apt-get install -y make g++ gcc
RUN apt-get install -y python tcsh perl fossil
RUN npm install
RUN apt-get remove -y --purge make g++ gcc \
    && apt-get clean
RUN apt-get install -y liblog-log4perl-perl
...
```

Hemos utilizado una imagen de *linux slim* y hemos debido de ajustar los comandos al gestor de paquetes *apt* y la sintaxis de los comandos de creación de usuarios y grupos. Por lo demás, una vez incluido como tarea automatizada bajo *gulpjs*, tenemos la base para construir de esta forma cualquier herramienta de terceros.

En caso de no existir ningún *docker* provisto por el desarrollador deberíamos generar uno desde cero con todas las dependencias necesarias.

La imagen de la aplicación no contiene todo

Debemos dejar fuera como mínimo los ficheros grandes de alta frecuencia de actualización, ya que con la estrategia *copy on write* también utilizada por *docker* tendríamos retrasos importantes en ese primer copiado. Esto deja fuera a los *logs* de la aplicación, las bases de datos y las copias de seguridad.

Tampoco es buena idea que se incluyan ficheros de los que debemos conservar sus actualizaciones, ya que nos obligaría a conservar la capa de escritura y extraer de ella con posterioridad la información relevante. Esto deja fuera a las base de datos y las copias de seguridad.

Por último tampoco debemos incluir todo aquello que deba ser actualizado desde el exterior, en nuestro caso el código fuente de la aplicación.

Por tanto en tiempo de ejecución montamos estos directorios compartidos desde el host, mediante el parámetro *mount bind*.

```
...
let param_string = `run -itd --rm \
--mount type=bind,src=${bind}/LOG,dst=${bind_internal}/LOG \
--mount type=bind,src=${bind}/DATA,dst=${bind_internal}/DATA \
--mount type=bind,src=${bind_code}/CODE,dst=${bind_internal}/CODE \...
```

7.3.2. Ejecución de contenedores

PlasmidNet aporta también varias tareas de ejecución de contenedores. Con estas tareas puede regularse la potencialidad de los contenedores (servicios activos), configuración de puertos, nombres de nodos, bases de datos a utilizar, directorios compartidos, *sockets* compartidos, ...

Las tareas utilizan intensamente la utilidad *screen* que nos capacita para acceder a varios pseudo-terminales desde un único terminal anfitrión. Esto nos permite gestionar más fácilmente la operación simultánea de un número variable de nodos de PlasmidNet, imprescindible para pruebas del sistema de orquestación.

7.4. Otras automatizaciones

Existe otro conjunto de tareas de ámbito diverso que no se han incluido en los tres bloques anteriores:

1) Generar empaquetados de toda la aplicación.
2) Interacción con el sistema git de control de versiones.
3) Limpieza de logs.
4) Creación de *symlinks*

    - Sistema documental: *images*.

- Estructura del volumen de sistema (*dockers*)
  - CODE
  - DATA
  - LOG

8

Conclusiones y trabajo futuro

PlasmidNet aporta un sistema completo para la resolución del reto planteado en la introducción: la publicación en Internet de los datos y procesos resultantes de proyectos de investigación de ámbito bioinformático. Diseñado bajo criterios de homogeneidad, nos permite configurar un servidor *web* para abordar todas las funcionalidades necesarias, habitualmente proporcionadas por plataformas y productos distintos.

Un servidor *web* PlasmidNet puede considerarse como un servidor totipotente capaz de adquirir muy variadas especializaciones mediante configuración:

1) Servidor de aplicación web progresiva de datos y procesos.
2) Servidor de aplicación web progresiva documental.
3) Servidor de servicios *REST*.
4) Servidor de descargas.
5) Servidor proxy de pruebas sincronizadas multidispositivo.
6) Servidor participante en flujos orquestados de tareas, asumiendo tres roles diferentes:
    1) Ejecutor de tareas.
    2) Gestor de tareas de usuario.
    3) Coordinador de la orquestación.

La implementación bajo protocolos *http* facilita las comunicaciones a través de la redes *tcp*, siendo compatible con *proxys*, balanceadores, encriptadores y otros elementos de la infraestructura.

Los nodos pueden ser distribuidos en máquinas físicas, virtuales y sobre cualquier plataforma en la nube compatible con contenedores *docker*. Y en ausencia de plataforma *docker* subyacente, la construcción sobre *nodejs* posibilita enormemente el despliegue sobre sistemas operativos diversos.

Por otra parte, la homogeneidad de implementación facilita el mantenimiento del sistema por parte de los mismos perfiles técnicos.

En los siguientes meses nos planteamos varias líneas de continuación, en gran medida paralelas:

- Completar las pruebas del sistema, actualizándolo inicialmente a las últimas versiones estables de *nodejs* (actualmente versión 12) y *docker*.
- Adaptar la aplicación web al *look&feel* del nuevo portal corporativo del *CBM*.
- Implantar el sistema en el entorno productivo del *CBM*.
- Mejorar la calidad de la clasificación funcional de los plásmidos incorporando nuevos pasos a los flujos teniendo en cuenta los últimos avances en la predicción de la estructura 3D de proteínas y priorizando las funcionalidades relacionadas con la resistencia a antibióticos.
- Nutrir a la aplicación *web* con más tipos de gráficos interactivos y tablas de datos.
- Posibilitar el almacenamiento local de sentencias de consulta a la base de datos que el usuario pueda seleccionar, también en modo desconectado, así como un conjunto de consultas de ejemplo.
- Ampliar el sistema de generación documental para incluir un nuevo tipo de documento: ayuda contextual de la *web app*.
- Construcción de una orquestación alternativa no centralizada donde todos los nodos sean equivalentes, con ausencia del rol central de coordinador. Las solicitudes de los usuarios se encaminarían a una red de nodos PlasmidNet y estos se repartirían por consenso los pasos a ejecutar.
- Analizar la posibilidad de publicar la información y procesos de otros proyectos del CBM, y al mismo tiempo evolucionar hacia una versión de PlasmidNet más generalizable.

Glosario de términos y acrónimos

- **Web App** Aplicación *web* habilitada para poder ser utilizada desde cualquier dispositivo y con cualquier tamaño de pantalla y con cualquier navegador. Utiliza por ello tecnología *HTML5* (javascript, hojas de estilos(*css*) y *html*).
- **PWA** Aplicación *web* progresiva. Se trata de una *web app* que es capaz de trabajar en modo desconectado de su servidor, en base al almacenamiento de datos y software en local. Puede instalarse dentro del dispositivo móvil y recibir notificaciones como si se tratase de una APP nativa.
- **devops** Conjunto de principios de ingeniería de software orientados a la unificación de los procesos del ciclo de vida de desarrollo de software (Dev) y de la operación del software (Ops), automatizando al máximo todos los pasos.
- **DSL** *Domain Specific Language* , es un lenguaje de programación dedicado a resolver un problema muy acotado, muchas veces orientado hacia la configuración de sistemas. Se diferencia de un lenguaje de propósito general (*java*, *C*, ..) en que la sintaxis es mucho más reducida y muy próxima a la semántica de su ámbito de aplicación. Como ejemplos tenemos: *SQL*, lenguaje de macros de Excel o configuración de flujos en *TensorFlow*.
- **MVC**. Modelo Vista Controlador es un patrón de diseño de software que separa una aplicación en tres partes: modelo de datos, presentación (vista) y control (controlador). El controlador aplica la vista a los datos.
- **SPA**. *Single Page App*, aplicación de página única es una aplicación web que utiliza una sola página y por ello está construida como si fuese una aplicación de escritorio.
- **NODEJS** Entorno de ejecución javascript multiplataforma, de código abierto, basado en el motor V8 de google, el mismo motor que incorporan los navegadores Chrome y otros.
- **API** *Application Programming Interface* Conjunto de métodos/funciones que proporciona un sistema informático para su consumo por cualquier otro sistema, incluido él mismo. Este conjunto suele estar debidamente documentado para facilitar su utilización.
- **DOM** *Document Object Model*, modelo de objetos, accesible por los programas, que constituye la representación de una página web dentro de los navegadores.
- **LokiJS** Base de datos *noSQL* que almacena sus datos en objetos *javascript* dentro del *Local Storage* del navegador.
- **POC** *Programming Over Configuration* o *Coding Over Configuration* es un principio de diseño y construcción de *software* que prioriza la implementación en base a programación ad hoc frente a la alternativa de crear programas más generales altamente configurables acompañados de uno o más ficheros de configuración.
- **COC** *Convention Over Configuration*. Principio de diseño y construcción de *software* que alienta la utilización de convenios de nombrado de entidades (nombres de funciones, módulos, clases, tablas,. . .) que evite la administración de las relaciones entre estas entidades

(por ejemplo, asignando el mismo nombre a todas las entidades que se refieren a lo mismo, independientemente de lo que sean). Sirve para simplificar la administración, que se vuelve en cierta medida implícita, por lo que también cuenta con detractores.

- **REST** Interfaz de intercambio de datos entre sistemas sobre protocolo *HTTP*. Los datos pueden transferirse en cualquier formato acordado entre cliente y servidor. Significa una ampliación del ámbito de uso de los protocolos *HTTP*, inicialmente ideados para el envío de ficheros (*html*, *js*, *css*, ...). Actualmente uno de los formatos más utilizados es *json*. Ha estado reemplazando paulatinamente otras arquitecturas como *SOAP* (basada en *XML*) o *DCOM*.
- **JSON** *Javascript Object Notation*. Consiste en la representación en texto plano de objetos *javascript* y comenzó utilizándose como una forma de almacenarlos o transmitirlos. Actualmente su uso se ha extendido hacia otros lenguajes, con preferencia a *XML* u otros formatos específicos.
- **MD5** *Message-Digest Algorithm 5*, algoritmo de reducción criptográfico de 128 bits dotado de la suficiente garantía de unicidad para realizar comprobaciones de integridad de ficheros.
- **CMS** *Content management system*. Gestor de contenidos. Es un sistema informático que facilita construir, probar y publicar contenido web a usuarios sin perfil técnico.
- **SEO** *Search Engine Optimization* es un conjunto de estrategias cuyo objetivo es mejorar el posicionamiento de un sitio web en la lista de resultados de los buscadores.
- **SEM** *Search Engine Marketing* es el conjunto de acciones encaminadas a promover comercialmente un determinado producto en Internet: mejora de posicionamiento en buscadores(*SEO*), inserción de anuncios en páginas muchas veces personalizados, envío de notificaciones push a navegadores, ...
- **Look&Feel** Aspecto gráfico de una aplicación: colores, fuentes, disposición de los elementos, ...
- **TEX** sistema de tipografía basado en macros inventado por Donald Knuth en 1985. El nombre deriva del griego *tech*.
- **LEAN** método de producción basado en la mejora continua y en la eliminación de desperdicios que no aportan valor al producto final.

Bibliografía

[1] AngularJS — superheroic JavaScript MVW framework, . URL https://angularjs.org/.

[2] Browsersync - time-saving synchronised browser testing, . URL https://www.browsersync.io/.

[3] D3.js - data-driven documents, . URL https://d3js.org/.

[4] DataTables | table plug-in for jQuery, . URL https://datatables.net/.

[5] Dynamic imports, . URL https://javascript.info/modules-dynamic-imports.

[6] Express - infraestructura de aplicaciones web node.js, . URL https://expressjs.com/es/.

[7] gulp.js - the streaming build system, . URL https://gulpjs.com/.

[8] Orientation and setup, . URL https://docs.docker.com/get-started/.

[9] Pandoc - pandoc user's guide, . URL https://pandoc.org/MANUAL.html.

[10] PostgreSQL: The world's most advanced open source database, . URL https://www.postgresql.org/.

[11] Redis, . URL https://redis.io/.

[12] RequireJS, . URL https://requirejs.org/.

[13] SQLite home page, . URL https://www.sqlite.org/index.html.

[14] Write-ahead logging, . URL https://www.sqlite.org/wal.html.

[15] Zotero | your personal research assistant, . URL https://www.zotero.org/.

[16] ace. Ace - the high performance code editor for the web. URL https://ace.c9.io/#nav=higlighter.

[17] Sundeep Agarwal. Customizing pandoc to generate beautiful pdfs from markdown. URL https://learnbyexample.github.io/tutorial/ebook-generation/customizing-pandoc/.

[18] Jeremy Ashkenas. jashkenas/docco. URL https://github.com/jashkenas/docco. original-date: 2010-03-01T00:42:38Z.

[19] Mathias Bynens. Dynamic import() · v8. URL https://v8.dev/features/dynamic-import.

[20] Alderson Chris Charlie Robbins; Cruger Jarrett, Hyde David. winstonjs/winston. URL https://github.com/winstonjs/winston. original-date: 2010-12-29T18:49:51Z.

[21] José Luis Cristina. Qué es DevOps (y sobre todo qué no es DevOps). URL https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops/.

[22] Google Developers. Caching files with service worker | web, . URL `https://developers.google.com/web/ilt/pwa/caching-files-with-service-worker?hl=es`.

[23] Google Developers. Introduction to push notifications | web, . URL `https://developers.google.com/web/ilt/pwa/introduction-to-push-notifications?hl=es`.

[24] Google Developers. Your first progressive web app, . URL `https://codelabs.developers.google.com/codelabs/your-first-pwapp/#2`.

[25] Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. 35(4):316–319. ISSN 1546-1696. doi: 10.1038/nbt.3820. URL `https://www.nature.com/articles/nbt.3820`.

[26] Blackrock Digita. BlackrockDigital/startbootstrap-sb-admin-2. URL `https://github.com/BlackrockDigital/startbootstrap-sb-admin-2`. original-date: 2014-10-25T00:36:48Z.

[27] Abia David Donoso Ana. Estudio comparativo de módulos funcionales en plásmidos bacterianos.

[28] Jeff Escalante. jescalan/atom-fold-comments. URL `https://github.com/jescalan/atom-fold-comments`. original-date: 2014-08-04T23:46:22Z.

[29] WebAssembly Working Group. WebAssembly. URL `https://webassembly.org/`.

[30] Luke Joliat. Do we still need JavaScript frameworks? URL `https://www.freecodecamp.org/news/do-we-still-need-javascript-frameworks-42576735949b/`.

[31] JS Foundation js.foundation. The jQuery team | jQuery foundation. URL `https://jquery.org/team/`.

[32] Mary Beth Kery, Amber Horvath, and Brad Myers. Variolite: Supporting exploratory programming by data scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 1265–1276. Association for Computing Machinery. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025626. URL `https://doi.org/10.1145/3025453.3025626`. event-place: Denver, Colorado, USA.

[33] Knopper Klaus. KNOPPIX - live linux filesystem on CD. URL `https://www.knopper.net/knoppix/index-en.html`.

[34] D. E. Knuth. Literate programming. 27(2):97–111. ISSN 0010-4620. doi: 10.1093/comjnl/27.2.97. URL `https://doi.org/10.1093/comjnl/27.2.97`.

[35] Ian MacLeod. nevir/groc. URL `https://github.com/nevir/groc`. original-date: 2011-11-26T09:29:14Z.

[36] Joe Minichino. techfort/LokiJS. URL `https://github.com/techfort/LokiJS`. original-date: 2013-07-25T07:39:20Z.

[37] Mireia More. ¿qué es el lean manufacturing o producción ajustada? URL `https://www.iebschool.com/blog/que-es-lean-manufacturing-negocios-internacionales/`.

[38] Node.js. Node.js. URL `https://nodejs.org/en/`.

[39] Bran Selic. Agile documentation, anyone? 26(6):11–12. ISSN 1937-4194. doi: 10.1109/MS.2009.167.

[40] Carlos Solís. La saga de angular, episodios: JS, 2, 3, 4 y 5. URL `https://www.linkedin.com/pulse/la-saga-de-angular-episodios-js-2-3-4-y-5-carlos-solis`.

[41] Martin Steinegger and Johannes Söding. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. 35(11):1026–1028. ISSN 1546-1696. doi: 10.1038/nbt.3988. URL `https://www.nature.com/articles/nbt.3988`.

[42] Yasuhiro Tanizawa, Takatomo Fujisawa, and Yasukazu Nakamura. DFAST: a flexible prokaryotic genome annotation pipeline for faster genome publication. 34(6):1037–1039. ISSN 1367-4811. doi: 10.1093/bioinformatics/btx713.

[43] Dennis Tenen and Grant Wythoff. Sustainable authorship in plain text using pandoc and markdown. URL `https://programminghistorian.org/en/lessons/sustainable-authorship-in-plain-text-using-pandoc-and-markdown`.

[44] Joan Vila. Replicación en sistemas distribuidos. page 41.

[45] MDN web docs. Control de acceso HTTP (CORS). URL `https://developer.mozilla.org/es/docs/Web/HTTP/Access_control_CORS`.

[46] Brian Woodward. doowb/ansi-colors. URL `https://github.com/doowb/ansi-colors`. original-date: 2015-11-30T17:42:04Z.

A

Manual de utilización

A.1. Servidor *web*

parse_config

Conversión a objeto de la configuración compactada en una cadena

parse_config –config [*http_ Y|httpdoc_ Y|https_ Y|app_ Y|services_ Y|pipelines_ Y|certbot_ N*]

start_server

Arranque de los servidores *web*: *web app* en *port*, *web app* documental en *doc_port*, y *https* en 443, dependiendo de los valores de la configuración *config*.

start_server –config [*http_ Y|httpdoc_ Y|https_ Y|app_ Y|services_ Y|pipelines_ Y|certbot_ N*] –node [*plasmid01*] –port [*9090*] –doc_port[*9099*]

A.2. Bases de Datos

parse_dsl_query

Traduce a sql la query en plasmidnet DSL pasada por parámetro.

parse_dsl_query –query [*p = NZ_LT960791.1 | p = AZ*]

do_hierarchy

Obtiene la estructura *json* tipo *hierarchy* de los datos devueltos por la consulta especificada.

hierarchy –query [*p = NZ_LT960791.1 | p = AZ*]

do_datatable

Obtiene la estructura *json* tipo *datatable* de los datos devueltos por la consulta especificada.

datatable –query [*p = NZ_LT960791.1 | p = AZ*]

do_force_network

Obtiene la estructura json tipo *force_network* de los datos devueltos por la consulta especificada.

force_network –query [*score > 0.8*]

A.3. Gestor de contenidos

cms_model_create

Crea el modelo de datos del gestor de contenidos.

cms_model_create

cms_model_drop

Borrado (*DROP*) de las tablas del gestor de contenidos.

cms_model_drop

start_cms

Arranca la base de datos de contenidos.

start_cms

upsert_content

Actualiza el contenido de nombre *name* , con html *body* para la versión de aplicación *version*.

upsert_content –name [*sidebar*] –pversion [*v0*] –body

get_content

Recupera el contenido de nombre *name* , y versión de aplicación *version*. Devuelve el cuerpo, la versión de aplicación y la versión del contenido.

get_content –name [*sidebar*] –pversion [*v0*]

extract_content_from_html

Obtención de los contenidos *html* por deconstrucción de la página estática de la aplicación.

extract_content_from_html –context [*plasmidnet*] –html [*v0*] –pversion[*v0*]

do_upsert_content

Lanza la extracción de contenidos desde la maqueta 'plasmidnet_cms.html' para la versión *pversion* para la *web app* PlasmidNet.

do_upsert_content –pversion [*v0*]

do_upsert_content_doc

Lanza la extracción de contenidos desde la maqueta *plasmidnet_doc_cms.html* sobre la versión *pversion* para la *web app* documental de PlasmidNet.

do_upsert_content –name [*sidebar*] –pversion [*v0*]

upsert_version

Actualiza la versión del objeto referenciado por *object*, para la versión de aplicación *pversion* y versión de objeto *subversion*.

upsert_version –object [*plasmidnet_datatables.js*] –pversion [*v0*] –subversion

get_subversion

Obtiene la versión del objeto referenciado por *object*, para la versión de aplicación *pversion*.

get_subversion –object [*plasmidnet_datatables.js*] –pversion [*v0*] –subversion

get_cache

Obtiene la validez de la cache referenciada por *cache*.

get_cache –cache [*modules*, widgets, data, static]

create_home

Crea la página principal de la aplicación *plasmidnet*, descargándola de la tabla de contenidos.

create_home –pversion [*v0*]

create_home_doc

Crea la página principal de la aplicación *plasmidnet_doc*, descargándola de la tabla de contenidos.

create_home_doc –pversion [*v0*]

A.4. Orquestador distribuido de tareas

expand_path

Función de utilidad que dado un directorio que se utiliza de modelo *template*, devuelve todos los nombres de los ficheros que terminan en *md5*.

expand_path –template

md5_calc

Calcula los *md5* de todos los ficheros pasados en la matriz *files* y se almacenan en ficheros cuyo nombre utiliza el sufijo *sufix* especificado, el nombre del fichero y el identificador del paso *step_id*.

md5_calc –task –step –legend –sufix –files –state_ok –state_ko

file_download

Transferencia de ficheros entre ejecutores de pasos. El ejecutor solicita a cada uno de los ejecutores predecesores los ficheros de entrada (de salida desde el punto de vista del predecesor).

La relación completa de ficheros a descargar se especifica en el parámetro de entrada *files*. Se informan también los siguientes campos:

- **task** Contenido completo de la tarea
- **step** Contenido completo del step a ejecutar
- **legend** Texto aclaratorio del tipo de transferencia que aparecerá en los log.
- **sufix_from** Sufijo de los ficheros de origen que queremos modificar en destino.
- **sufix_to** Sufijo de destino que sustituye al *sufix_from*
- **files** Matriz de ficheros a descargar.
- **state_ok** Identificador del estado al que se debe actualizar el *step* si el proceso termina sin errores.
- **state_ko** Identificador del estado al que se debe actualizar el *step* si el proceso termina con errores.

Para identificar los ficheros se pasa por parámetro los datos completos de tarea *task* y paso *step*.

file_download –task –step –legend –sufic_from –sufix_to –files –state_ok –state_ko

md5_compare

Compara todos los ficheros *md5* descargados (sufijos *down.md5*) con los *md5* calculados *chk.md5*.

Provee dos formas de ejecución, con *type* igual a *verify* cambia además el estado del paso a estado erróneo.

md5_compare –task –step –type –files –legend –state_ok –state_ko

step_md5_download

Descarga desde el directorio del proveedor de los ficheros hasta el directorio local (del elecutor del *step*) todos los ficheros *md5*, y se renombran cambiando el sufijo *out.md5* a *down.md5*. Para ello utiliza la función *file_download* donde informa también el estado OK y el estado KO de este subproceso.

Para identificar los ficheros se pasan por parámetro los datos completos de tarea *task* y paso *step*.

step_md5_download –task –step

step_download

Descarga desde el directorio del proveedor de los ficheros hasta el directorio local (del ejecutor del *step*) todos los ficheros *md5*, y se renombran cambiando el sufijo *out.md5* a *down.md5*.

Los ficheros que deben descargarse son los que el paso almacena en la matriz *download_required*.

step_download –task –step

step_md5_chk

Calcula los *md5* de todos los ficheros de entrada del paso y los almacena en ficheros con sufijo *chk*.

step_md5_chk –task –step

step_md5_out

Calcula los *md5* de todos los ficheros de salida del paso y los guarda en local con el sufijo *out*.

step_md5_out –task –step

step_md5_down

Calcula los *md5* de todos los ficheros de entrada del paso y los guarda en local con el sufijo *chk*.

step_md5_down –task –step

step_md5_compare

Compara los *md5* calculados para todos los ficheros de entrada del paso con los *md5* descargados desde los proveedores de origen de los ficheros. Sirve para determinar, a través de *md5_compare* la lista de ficheros necesarios para poder lanzar la ejecución del paso. El paso no se marca como erróneo si los *md5* no coinciden.

step_md5_compare –task –step

step_md5_verify

Equivalente a *step_md5_compare* pero en este caso el paso se marca como erróneo si los *md5* no coinciden.

step_md5_verify –task –step

resolve_cmd

Interpreta la cadena del comando de sistema especificado en el paso en el campo *step.command*. Si este campo no está especificado o está vacío, se entiende que el comando está definido en *cmd_data* y ajusta las opciones de ejecución en consecuencia. Se devuelven el comando a ejecutar y las opciones de ejecución.

resolve_cmd –task –step

step_exec

Ejecuta el comando de sistema cuya cadena obtiene de la función *resolve_cmd*. Actualiza los estados de la tarea y paso dependiendo del resultado.

step_exec –task –step

task_create

Solicita al coordinador *task_provider* la ejecución de un flujo del tipo especificado en *pipeline*. Se indican los pasos de inicio y fin (por defecto *0*, que indica que queremos lanzar ya el primer paso). Utiliza el servicio *plas/pipeline/task_create*. Se informa también el identificador de tarea *task_id*. Si la tarea ya existe, el sistema volverá a ejecutar todos los pasos de la misma entre *step_from* y *step_to*.

task_create –task_provider –pipeline –task_id –step_from [*0*] –step_to [*0*]

step_do

Solicita al proveedor del paso *task*, *step* la ejecución del mismo.

step_do –task –step

task_info

Solicita al coordinador *task_owner* de la tarea *task_id* toda la información de la misma.

task_info –task_owner –task_id

step_info

Solicita al coordinador *task_owner* de la tarea *task_id* toda la información del paso identificado por *step_id*.

step_info –task_owner –task_id –step_id

task_update

Actualiza el contenido del paso (parámetro *step*) de la tarea *task_id* en el proveedor ejecutor del paso.

task_update –task_id –step

step_start

Crea y arranca el paso identificado por *step_id* de la tarea *task*.

step_start –task –step_id

task_continue

Crea y arranca el siguiente paso del flujo para la tarea *task* y el paso ejecutado *step*.

task_continue –task –step

task_start

Crea en la base de datos, la tarea de identificador *task_id* y de nodo solicitante (nodo gestor) *task_client* asociándola al flujo especificado (*pipeline*) y definiendo también los pasos de inicio (*step_from*) y fin (*step_to*).

task_start –task_id –task_client –pipeline –step_from –step_to

A.4.1. Base de datos de orquestación

sqlite_upsert_task

Actualiza o inserta el registro *task* en la tabla *tasks* conforme a su clave *task_id*.

sqlite_upsert_task –task [*{task_id : "test", pipeline: "plasmid_modules"}*]

sqlite_upsert_step

Actualiza o inserta el registro *step* en la tabla *steps* conforme a su clave *task_id + step_id*.

upsert_step –step [*{task_id : "test", step_id: "2", "command": "plas.pl", "providers" : {"plasmid_GI.tsv" : {"provider" : "plasmid01", "step_id" : "0"}}*]

sqlite_start

Arranca la base de datos *sqlite* de tareas de usuario y crea las tablas si no existen.

El modelo contiene las tablas *tasks*, *steps* y *log*.

start

sqlite_update

Upsert combinado de la tarea *task* y el paso *step* pasados por parámetro. Pueden llegar vacíos.

update –task [*{task_id : "test", pipeline: "plasmid_modules"}*] –step [*{task_id : "test", step_id: "2", "command": "plas.pl", "providers" : {"plasmid_GI.tsv" : {"provider" : "plasmid01", "step_id" : "0"}}*]

sqlite_exists_task

Devuelve *true* si la tarea identificada por *task_id* existe en la base de datos y *false* en caso contrario.

exists_task –task_id [*test*]

sqlite_get_tasks

Devuelve en una matriz todas las tareas de la base de datos.

get_tasks

sqlite_get_steps_all

Devuelve en una matriz todos los *steps* de la base de datos.

sqlite_get_steps_all

sqlite_get_task

Devuelve la tarea cuyo identificador *task_id* se pasa por parámetro.

get_task –task_id [*test*]

sqlite_get_steps

Devuelve en una matriz todos los *steps* de la tarea cuyo identificador *task_id* se pasa por parámetro.

get_steps –task_id [*test*]

sqlite_get_step

Devuelve el step cuyo identificador *task_id* + *step_id* se pasa por parámetro.

get_step –task_id [*test*] –step_id [*plas*]

sqlite_test

Función para verificar la creación de la base de datos y las operaciones de obtención de *tasks* y *steps*.

sqlite_test

sqlite_test_upserts

Función para verificar las funciones de *upserts*

sqlite_test_upserts

A.5. Automatización Extrema

A.5.1. Generación Documental

doc_extract_md

Extrae los comentarios *markdown* del fichero *file* de entrada, relacionados con *doc_type* y *lang*.

doc_extract_md –file [*gulpfile.js*] –doc_type [*high_level*, slides, detail, user] –lang [*es*, en]

doc_md_to_tex

Convierte a latex el fichero *markdown* asociado a *file* mediante *pandoc*. El nivel de los capítulos y secciones se ajusta en base al parámetro *shift* que se envía a *pandoc* sin modificaciones.

doc_md_to_tex –shift [0] –file [*gulpfile.js*]

doc_md_to_slides

Convierte a slides reveal.js el fichero *markdown* asociado a *file* mediante *pandoc*.

doc_md_to_slides –file [*gulpfile.js*]

doc_md_to_slides_pdf

Convierte a slides beamer el fichero *markdown* asociado a *file* mediante *pandoc*.

doc_md_to_slides_pdf –file [*gulpfile.js*]

doc_md_to_html

Convierte a html el fichero *markdown* asociado a *file* mediante *pandoc*.

doc_md_to_html –shift [*0*, 1, -1, ...] –file [*gulpfile.js*]

doc_latex_to_pdf

Genera el *pdf* global desde el agregado latex vía *pdflatex*.

El *pdf* global se encuentra sobre una plantilla latex externa al sistema *pandoc*.

Es necesario ejecutar *bibtex* en primer lugar con el fin de actualizar los enlaces de las citas, imágenes y tablas.

Y dos veces *pdflatex* para que se resuelvan los links. En la primera ejecución sólo se actualiza la lista de imágenes

doc_latex_to_pdf –shift [*0*, 1, -1, ...] –file [*gulpfile.js*]

tex_to_svg

Convierte un fichero latex *tikz*, *smartdiagram*, ... en *pdf* y *svg*, que serán utilizados respectivamente por los ficheros de salida latex y html. Los ficheros de entrada se buscan en resources/tikz

tex_to_svg –texgraph [*doc_ diagram_ webapp*]

doc_html_to_webapp

Genera la webapp html a partir del tipo de documento **doc_type**: *detail*, *user*, *high_ level*, *slides*, e idioma **lang**: *es*, *en*.

doc_html_to_webapp –doc_type [*high_ level*, slides, detail, user, all] –lang [*es*, en]

doc_gendoc

Genera toda la documentación asociada al proyecto. Primero extrae los *markdown* de los ficheros fuentes. También de los fuentes que son *markdown* puros, después lanza las transformaciones *pandoc* y por último el agregado para la *web app* y la generación del *pdf* agregado.

doc_gendoc –doc_type [*high_level*, slides, detail, user, all] –lang [*es*, en] –pdf [*N*, Y]

man

Imprime por consola la ayuda asociada a las tareas *gulp* relacionadas con un determinado *scope* (*doc*, *docker*, ...). Por defecto imprime las tareas *gulp* de documentación.

man –scope [*doc*, docker, site, dist, pipelines, bio, misc, app, cms] –lang [*es*, en]

A.5.2. Pruebas y despliegue de software

browserSync

Arranca el proxy *broswersync* [2], como intermediario del servidor de la aplicación, que debe estar escuchando en el puerto especificado en *port*.

browserSync –port [*9090*]

site_css

Genera el fichero *css* de la aplicación y lo distribuye, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_css –pversion [*v0*]

site_js

Genera el fichero *js* de la aplicación y lo distribuye, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_js –pversion [*v0*]

site_js_doc

Genera el fichero js de la aplicación y lo distribuye, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_js_doc –pversion [*v0*]

js_other

Distribuye el resto de ficheros: html, *javascript* externo, fuentes y ficheros de test, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_js_other –pversion [*v0*]

js_modules

Distribuye como módulos los ficheros *javascript* de la aplicación.

site_js_modules –pversion [*v0*]

js_doc_other

Distribuye el resto de ficheros de la *web app* documental, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_js_other –pversion [*v0*]

site_manifest

Distribuye el manifiesto de la web app, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_manifest –pversion [*v0*]

site_manifest_doc

Distribuye el manifiesto de la web app documental, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_manifest –pversion [*v0*]

site_icons

Distribuye los iconos de las *web app*, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

site_icons –pversion [*v0*]

site_bs

Desencadena la construcción y distribución completamente, después arranca los *filewatchers* y el proxy *browsersync*.

site_bs –pversion [*v0*] –port [*9090*]

site_watcher

Desencadena la construcción y distribución completamente, después arranca los *filewatchers*.

site_watcher –pversion [*v0*] –port [*9090*]

site_build

Desencadena la construcción y distribución completas de la *web app* y de la *web app* documental.

site_build –pversion [*v0*]

dist_root

Distribuye el fichero central del servidor *gulpfile.js*.

dist_root

dist_package

Distribuye el fichero de dependencias *package.json* desde el directorio principal del servidor hasta el directorio de código.

dist_package

dist_build

Distribuye el los ficheros de construcción de *dockers*.

dist_build

dist_pipelines

Distribuye el los ficheros de código de pipelines.

dist_pipelines

dist_modules

Distribuye los ficheros *javascript* de módulos y genera versión en el *CMS*.

dist_modules –pversion [*v0*]

dist_proxy_modules

Distribuye *proxy_modules.js*.

dist_proxy_modules

watcher

Arranca los *filewatchers* para la distribución automática en cuanto se producen cambios en los ficheros fuentes. Genera automáticamente los contenidos si detecta un cambio en la maqueta.

watcher –pversion [*v0*]

dist_modules_all

Distribuye todos los módulos, incluido *proxy_modules.js*.

dist_modules_all –pversion [*v0*]

A.5.3. Despliegue basado en contenedores

docker_sh

Ejecuta en *screen* una shell lanzada desde el contenedor identificado por parámetro *node*.

docker_sh –node [*plasmid01*]

docker_server

Arranca la tarea por defecto *gulp* en el contenedor identificado por parámetro *node*.

docker_server –node [*plasmid01*]

docker_rm

Borra el contenedor identificado por parámetro *node*.

docker_rm –node [*plasmid01*]

docker_stop

Detiene la ejecución del contenedor identificado por parámetro *node*.

docker_stop –node [*plasmid01*]

docker_build

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

docker_build –dist_version [*v.2020.02*]

docker_build_dev

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile.dev* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

docker_build_dev –dist_version [*v.2020.02*]

docker_build_dfast

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile.dfast* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

docker_build_dfast –dist_version [*v.2020.02*]

docker_build_glibc

Construye la imagen de la aplicación DFAST de acuerdo a las especificaciones del fichero *Dockerfile.dfast* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

docker_build_glibc –dist_version [*v.2020.02*]

docker_build_mmseq2

Construye la imagen de la aplicación MMSEQS2 de acuerdo a las especificaciones del fichero *Dockerfile.mmseq2* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

docker_build_mmseq2 –dist_version [*v.2020.02*]

docker_run

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web.

docker_run –node [*plasmid01*] –pimage [**] –port [*9090*] –dist_version [*v.2020.02*] –bind –bind_code –bind_internal

docker_run_dev

Crea y ejecuta el contenedor asociándole el identificador *node*, para la imagen *plasmidnet_dev_image* de código de versión *dist_version*, con puerto de servidor web *port*.

docker_run_dev –node [*plasmid01*] –port [*9090*] –dist_version [*v.2020.02*]

docker_run_redis

Crea y ejecuta un contenedor para la imagen de la base de datos *redis*, asociándole el identificador *node*, con puerto de servidor redis_port_.

docker_run_redis –node [*plasmid01*] -redis_port [*6379*]

docker_run_all

Crea y ejecuta varios contenedores creando un entorno de pruebas de pipelines:

1) Un contenedor plasmidnet glibc.
2) Un contenedor *redis*.
3) Un contenedor *dfast*.
4) Un contenedor *mmsegs2*.

docker_run_all -dist_version [*v.2020.02*] –bind_code –bind_internal

docker_start_sh

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web. Además arranca en el contenedor una shell *sh* y la empareja con un terminal, de *screen*.

docker_start_sh –node [*plasmid01*] –pimage [**] –port [*9090*] –dist_version [*v.2020.02*] –bind –bind_code –bind_internal

docker_start_server

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web. Además arranca en el contenedor el servidor de la aplicación (arrancado mediante la tarea por defecto de *gulp*).

docker_start_server –node [*plasmid01*] –pimage [**] –port [*9090*] –dist_version [*v.2020.02*] –bind –bind_code –bind_internal

docker_build_all

Construye todos los tipos de docker asociándoles la versión *dist_version*: producción, dev, dfast, mmseq2

docker_build_all –dist_version [*v.2020.02*]

A.5.4. Otras automatizaciones

zip_app

Realiza un backup (*tar.gz*) de todo el directorio de la aplicación.

zip_app –all [*true*]

zip

Realiza un backup (*tar.gz*) del directorio *dir* (CODE, LOG, DATA)

zip –dir [*CODE*, LOG, DATA]

git_add

Ejecuta el comando *git add* sobre el directorio raíz del código fuente.

git_add

git_commit

Ejecuta el comando *git commit -am* incluyendo el mensaje especificado en *message* sobre el directorio raíz del código fuente.

git_commit –message

git_push

Ejecuta el comando *git push origin master* sobre el directorio raíz del código fuente.

git_push

git_execute

Ejecuta el comando *git* con los parámetros especificados en *param_string*.

git_execute

git_credentials

Solicita al usuario sus credenciales *github* y las almacena en el entorno.

git_credentials

git_config_email

Añade la dirección de correo de la cuenta de git (*email*) a la configuración global del sistema.

git_config_email

git_config_name

Añade el usuario de la cuenta de git (*name*) a la configuración global del sistema.

git_config_email

github_config

Configura la cuenta de github a parftir de los datos de credential.helper.

github_config

test

Tarea de test para comprobar que el servidor responde.

test

link_LOG

Genera el *symlink* del directorio de logs *LOG*.

link_LOG

link_LOG

Genera el *symlink* del directorio de datos *DATA*.

link_DATA

link_CODE

Genera el *symlink* del directorio de código fuente *CODE*.

link_CODE

link_BACKUP

Genera el *symlink* del directorio de backup *BACKUP*.

link_BACKUP

data_clean_dry

Simula el borrado del directorio *DATA* como verificación de ficheros afectados antes de un borrado real. No se borran: el directorio raíz, el directorio de la bd *plasmid_modules*, *INPUT*.

data_clean_dry

data_clean

Borrado del directorio *DATA*. No se borran: el directorio raíz, el directorio de la bd plasmid_modules, *INPUT*.

data_clean

log_clean_dry

Simula el borrado del directorio de logs *LOG* como verificación de ficheros afectados antes de un borrado real.

log_clean_dry

log_clean

Borra el directorio de logs *LOG* como verificación de ficheros afectados antes de un borrado real.

log_clean

git_all

Ejecuta en serie los comandos *git*: *add*, *commit* y *push*, asociando al *commit* el mensaje espevificado en *message*.

git_all –message

B

Manual del programador

B.1. Estructura Global

B.1.1. Módulo principal

El módulo **gulpfile.js** es el módulo principal del sistema de tareas **gulpjs**.

Este módulo carga dinámicamente todos los módulos de tareas, prefijados por convenio como 'gulpfile'.

Se arrancan los siguientes servicios:

1. Servidores *web*.
2. *Filewatchers* para la detección de cambios en ficheros.
3. Inicializaciones de base de datos.

```
global.__basedir = __dirname;
```

MODEL es la variable de entorno que especifica la base de datos utilizada para las tareas de flujo. Tenemos tres opciones: * sqlite * redis * loki.

Por defecto se utiliza *sqlite*.

```
global.__model = process.env.MODEL || 'sqlite';
const {series, parallel} = require('gulp');
const {require_dyn} = require('./modules/proxy_modules');
```

Esta función define la carga de los métodos del módulo. En cada módulo, el objeto **shares** incluye las funciones y objetos exportados.

```
function task_exporter(mod) {
  let module_dyn = require_dyn(mod);
```

```
  for (let share in module_dyn.shares) {
    if (share !== 'shares') {
      exports[share] = module_dyn[share];
    }
  }
}
```

Exportaciones En cada llamada a *task_ exporter* se publican los métodos de cada uno de los módulos de forma que son accesibles al usuario como tareas de **gulpjs**.

```
task_exporter("gulpfile_misc");
...
task_exporter("gulpfile_site");
task_exporter("gulpfile_task_model_" + __model);
```

Y ahora definimos como tarea por defecto todas las tareas de inicio, que serán lanzadas en paralelo por *gulpjs*.

```
exports.default = parallel(exports.start, exports.start_bio, exports.start_cms,
    ↪ exports.watcher, exports.site_bs, exports.start_server);
```

B.2. Navegador

B.2.1. plasmidnet.js

Este módulo define las variables globales de la aplicación (nivel de ventana) y el comportamiento en el evento *ready*.

```
const REGEXP_SPACES = new RegExp(/\s+/g);
var db;
var db_widgets;
var db_info;
var editor;
var dyn = {}; // Dynamic functions or functions used by dynamic modules
var DEBUG = false; // Activate/deactivate console debug
var ERROR = false; // Activate/deactivate console error
var INFO  = false; // Activate/deactivate console info
```

** bypass_cache **

La cadena indica en cada posición qué datos deben ser obtenidos del servidor (S) con prioridad a la caché.

Las posiciones indican:

1. Caché de ficheros *js*, *css* o *html*. Compete sólo al *service worker*. El valor *S* expresa aquí que se obtienen del servidor los ficheros *plasmidnet.*.js* y *plasmidnet.*.css* que aglutinan el software propio de la aplicación, los ficheros de los proveedores se siguen obteniendo principalmente de la caché. Si el valor es *A* todos los ficheros se recuperan del servidor.
2. Caché de injertables (*widgets*) (colección referenciada por *db_widgets* de *Lokijs*).
3. Caché de datos: módulos, familias, superfamilias, ... (colección referenciada por *db_info*).
4. Caché de módulos dinámicos: cuya carga no implica reinicio del navegador.

Por defecto la marcamos con prioridad caché. Esta prioridad puede ser modificada dinámicamente, como veremos al respecto del *service worker*.

El ciclo de vida de este objeto se describe en el *service worker*.

```
var bypass_cache = 'NNNN';
```

load_module

Carga dinámicamente desde el servidor, sin reiniciar el navegador, el módulo *module*. Se recupera con un dato variable para que no sea interceptado por ningún caché, ni de servidor ni de navegador. Las funciones de este módulo son exportadas para sustituir a las cargadas inicialmente. Los módulos que estamos utilizando en carga dinámica son:

- plasmidnet_graphs
- plasmidnet_datatables

```
function load_module(module) {
  // Load module code
  DEBUG && console.log('load_module ' + module);
  //import('/plas/service/module/' + module + '?mtime=' + Date.now())
  import('/plas/service/module/' + module + '?mtime=a')
  .then(mod => {
    dyn[module] = mod;
    DEBUG && console.log('load_module promise ' + module);
  })
  .catch(error => ERROR && console.log("load_module ERROR ", module, error.
    ↪ message));
}
```

document.ready

Cargamos la base de datos de *loki*, que es un proceso asíncrono. Una vez cargada activamos los *widgets* cuyo estado es el inicial (pn-state = 'onLoad') o corresponden al estado que hemos asociado a la presentación de la aplicación (pn-state = 'onAbout').

```
$(document).ready(function() {
  loki_load_db()
  .then((successMessage) => {
```

```
    INFO && console.log('loki_load_db ', successMessage);
    var active_states = ['onLoad', 'onAbout'];
    activate_widgets(active_states)
    .then(
      function(value) {
        INFO && console.log(value);
      },
      function(reason) {
        INFO && console.log(reason);
      }
    );
  })
  .catch(error => {
    ERROR && console.log(`loki_load_db start app plasmidnet ${error.message}`);
    ERROR && console.log(error);
  });
});
```

B.2.2. plasmidnet_widgets.js

Este módulo maneja los injertables de la aplicación, actualizándolos, clonándolos o borrándolos dependiendo del estado. Implementa de hecho una máquina de estados. Determinados eventos del sistema: navegación, carga inicial, modifican el estado de la aplicación. En cada evento el programa muestra sólo los injertables cuyo estado es compatible y oculta los demás.

Si los injertables son creados por primera vez, también genera las funciones asociadas a los eventos de los injertables.

parse_tag

Busca en la cadena *s* (*html*) el valor de la variable *tag*, teniendo en cuenta que en esa cadena la variable aparece en la forma :

```
nombre_variable = "valor_variable"
```

```
function parse_tag(tag, s) {
  var regexp = new RegExp(tag + '="(.*?)"');
  var result = regexp.exec(s);
  if (result && 1 in result) return result[1];
  else return "";
}
```

replace_tag_append

Sustituye en la cadena de *html target* todos los valores incrustados entre las cadenas *tag* + *sep_left* y *sep_right*. La sustitución se efectúa añadiendo al valor existente el sufijo *append_text*.

Esta es una invocación típica:

```
html_new = replace_tag_append('pn-link href', html_new, '_' + new_seq_ref, '="',
    ↪  '"');
```

En este caso una cadena con la forma '...pn-link href="reference"...' se sustituiría por '...pn-link href="reference#1"...' si *append_text* está definido como *#1*.

```
function replace_tag_append(tag, target, append_text, sep_left, sep_right) {
  var regexp = new RegExp(tag + sep_left + '(.*?)' + sep_right, 'g');
  var matches = target.match(regexp);
  if (matches) {
    for (var i = 0; i < matches.length; i++) {
      var match = matches[i];
      var regexp = new RegExp(tag + sep_left + '(.*?)' + sep_right);
      var result = regexp.exec(match);
      if (result && 1 in result) {
         var old_value = tag + sep_left + result[1] + sep_right;
         var new_value = tag + sep_left + result[1] + append_text + sep_right;
         target = target.replace(old_value, new_value);
       }
    }
  }
  return target;
}
```

replace_tag

Reemplaza en la cadena *s* (html) el valor de la variable *tag*, teniendo en cuenta que en esa cadena la variable aparece en la forma:

```
nombre_variable = "valor_variable"
```

El valor existente es reemplazado por el que contiene la variable *new_value*.

```
function replace_tag(tag, s, new_value) {
  var regexp = new RegExp(tag + '="(.*?)"');
  var result = regexp.exec(s);
  if (result && 1 in result) {
    var old_value = tag + '="' + result[1] + '"';
    return s.replace(old_value, tag + '="' + new_value + '"');
  } else {
    return s;
  }
}
```

parse_ref

Obtiene el valor de los *tags* 'pn-ref' y 'pn-state' en la cadena *html*. Devuelve el primer valor encontrado.

Estas etiquetas identifican un injertable plasmidnet (pn-ref) y su estado (pn-state).

```
function parse_ref(html) {
  var object_id = parse_tag('pn-ref', html);
  var state = parse_tag('pn-state', html);
  return {state: state, id: object_id};
}
```

clone_delete

Borra un clon de un injertable referenciado por *pn_ref*. Se borra del DOM y de la base de datos local.

```
function clone_delete(pn_ref) {
  DEBUG && console.log("clone_delete ", pn_ref);
  let included_objects = $('[pn-ref*="' + pn_ref + '"]');
  for (var i = 0; i < included_objects.length; i++) {
    DEBUG && console.log("clone_delete ", included_objects[i]);
    var object_ref = parse_ref(included_objects[i].outerHTML);
    loki_remove_widget(object_ref.id, object_ref.state);
    included_objects[i].remove();
  }
}
```

card_expand

Cambia la visualización del injertable *pn-ref* tipo *card* a modo expandido. En este modo abarca más columnas en la visualización.

```
function card_expand(pn_ref) {
  let included_object = $('[pn-ref="' + pn_ref + '"]');
  INFO && console.log('card_expand');
  included_object.toggleClass("col-xl-4 col-xl-12");
  included_object.toggleClass("col-lg-8 col-lg-12");
}
```

activate_transitions

Crea la respuesta al evento *onclick* del elemento *DOM* del injertable referenciado por *pn_ref*: cualquier elemento que pueda responder a un evento *onclick*. La respuesta depende del estado contenido en el tag *pn-state-transition*.

El evento activa los injertables cuyo estado coincide con el recuperado del tag y oculta el resto de injertables.

```
function activate_transitions(pn_ref) {
  // Create observer state transitions for main links and buttons
  DEBUG && console.log('activate_transitions', pn_ref);
  //$("[pn-ref=" + pn_ref + "]")
  var pn_transitions = $('[pn-ref="' + pn_ref + '"]').find("[pn-state-transition
    ↪ ]");
  for (var i = 0; i < pn_transitions.length; i++) {
    pn_transitions[i].onclick = function() {
      var new_state = parse_tag('pn-state-transition', this.outerHTML);
      DEBUG && console.log('clicked ' + new_state);
      activate_widgets([new_state]);
      mute_dom_widgets([new_state]);
    }
  }
}
```

manage_searchPlasmid_errors

La primera operación que se realiza cuando se activa un injertable complejo consiste en obtener los datos desde la base de datos o desde la caché (dependiendo de las prioridades establecidas).

Si la respuesta es errónea el resultado se muestra al usuario en la tarjeta *searchPlasmid*.

```
function manage_searchPlasmid_errors(response) {
  INFO && console.log("manage_searchPlasmid_errors RESPONSE:", response.data);
  if (response.data.errorcode && response.data.errorcode != "") {
    ERROR && console.log("manage_searchPlasmid_errors ERROR: ", response.data);
    $('#searchPlasmid .alert-warning').text(response.data.message);
    $('#searchPlasmid .alert-warning').show();
    $('#searchPlasmid .alert-info').hide();
    return false;
  } else {
    DEBUG && console.log("manage_searchPlasmid_errors OK");
    $('#searchPlasmid .alert-warning').hide();
    $('#searchPlasmid .alert-info').text("Query OK");
    $('#searchPlasmid .alert-info').show();
    return true;
  }
}
```

activate_actions

Crea la respuesta al evento *onclick* del elemento *DOM* contenido en el injertable referenciado por *pn_ref*. Este elemento se etiqueta con *pn-action* que puede estar en cualquier elemento que responda a un evento *onclick*. La respuesta depende del estado contenido en la etiqueta citada.

Este es un ejemplo de cómo aparece en el *html* del injertable plasmidSearch asociado al botón 'Plasmid Data':

```
<a href="#" pn-action="addPlasmidData" pn-action-ref="plasmidData" class="btn
    ↪ btn-primary btn-block bg-success">
   Plasmid Data
</a>
```

Esta sintaxis de alto nivel informa al programa que cuando se produzca el evento *onclick* del botón, debe desencadenar la acción *addPlasmidData*, insertando una nueva tarjeta de datos cuyo molde está referenciado por *pn_ action_ ref*. Esta acción se programa dinámicamente en esta función.

Las acciones posibles son:

1) *addPlasmidData*
2) *addPlasmidGraph*
3) *deletePlasmidGraph*
4) *deletePlasmidGraph*
5) *expandPlasmidGraph*
6) *expandPlasmidData*

En el evento onclick de *addPlasmidData* y *addPlasmidGraph* se efectúa primeramente una llamada a *manage_ searchPlasmid_ errors* para validar los datos. En caso de error se informa en la misma tarjeta de búsqueda.

Hemos definido dos consultas por defecto para facilitar las pruebas (@1 y @2) que se traducen al vuelo (antes de invocar al servidor) en dos consultas que devuelven datos adecuados para presentar respectivamente los gráficos interactivos jerárquicos o los gráficos de red.

```
function activate_actions(pn_ref) {
  // Create observer state transitions for main links and buttons
  INFO && console.log('activate_actions', pn_ref);
  var pn_actions = $('[pn-ref="' + pn_ref + '"]').find("[pn-action]");
  //var pn_actions = $("[pn-action]");
  DEBUG && console.log('ACTIONS', pn_actions.length);
  if (pn_ref === "searchPlasmid") {
    editor = ace.edit("examplePlasmidID");
    $(".ace_editor").css("height", "300px");
    editor.setTheme("ace/theme/chrome");
    editor.session.setMode("ace/mode/plas");
    editor.renderer.setShowGutter(false);
    editor.session.setUseWrapMode(false);
    editor.setOptions({
      fontSize: "11pt",
      tabSize: 0
    });
    editor.setValue("plasmid == KY362373.1, NZ_CP018684.2");
  }
  for (var i = 0; i < pn_actions.length; i++) {
    pn_actions[i].onclick = function() {
      var pn_action = parse_tag('pn-action', this.outerHTML);
```

```
      var pn_action_ref = parse_tag('pn-action-ref', this.outerHTML);
      if (pn_action === "addPlasmidData" || pn_action === "addPlasmidGraph") {
        DEBUG && console.log('activate_actions clicked action ', pn_action,
    ↪ pn_action_ref);
        //let query = $('#searchPlasmid #examplePlasmidID').val();
        let query = editor.getValue();
        DEBUG && console.log("activate_actions query ", query);
        // Validate query
        let search_service = "";
        if (pn_action === "addPlasmidData") search_service = "datatable";
        else search_service = $('#searchPlasmid input:radio:checked').val();
        // Default queries for testing
        if (query === "@1") query = "p==KY362373.1,NZ_CP018684.2";
        if (query === "@2") query = "score > 0.8";
        dyn.get_data(search_service, query, bypass_cache).then(response => {
          INFO && console.log("RESPONSE:", response.data.data);
          if (manage_searchPlasmid_errors(response)) {
            loki_upsert_info(search_service, query, response.data);
            create_widget(pn_action_ref, query, search_service, response.data);
          }
        })
        .catch(error => {
          ERROR && console.log(`activate_actions  ${error.message}`);
          ERROR && console.log(error);
        });
      } else if (pn_action == "deletePlasmidGraph" || pn_action == "
    ↪ deletePlasmidData") {
        DEBUG && console.log('activate_actions clicked action ', pn_action,
    ↪ pn_action_ref);
        clone_delete(pn_action_ref);
      } else if (pn_action == "expandPlasmidGraph" || pn_action == "
    ↪ expandPlasmidData") {
        DEBUG && console.log('activate_actions clicked action ', pn_action,
    ↪ pn_action_ref);
        card_expand(pn_action_ref);
      } else {
        DEBUG && console.log('activate_actions clicked action default ',
    ↪ pn_action);
      }
    }
  }
}
```

activate_sidebar_toggle

Genera la función asociada al evento *onclick* del injertable *sidebar*. Esta función gestiona la disminución y el aumento del ancho de la *sidebar*.

```
function activate_sidebar_toggle(pn_ref) {
```

```
  $(`[pn-ref="${pn_ref}"]`).find("#sidebarToggle, #sidebarToggleTop").on('click'
    ↪ , function(e) {
    INFO && console.log('clicked sidebar');
    $("body").toggleClass("sidebar-toggled");
    $(".sidebar").toggleClass("toggled");
    if ($(".sidebar").hasClass("toggled")) {
      $('.sidebar .collapse').collapse('hide');
    };
  });
}
```

mute_dom_widgets

Oculta los injertables cuyo estado no está activado, o sea, no está en la lista de *active_states*.

Se considera que los injertables en estados *all* e *init* deben estar siempre visibles.

```
function mute_dom_widgets(active_states) {
  var excluded_objects = $("[pn-state]");
  active_states.push('all');
  active_states.push('init');
  for (var i in active_states) {
    excluded_objects = excluded_objects.not("[pn-state=" + active_states[i] +
    ↪ "]");
  }
  for (var i = 0; i < excluded_objects.length; i++) {
    var excluded_object = excluded_objects[i];
    var object_ref = parse_ref(excluded_object.outerHTML);
    //if (content !== "") {
      $(excluded_object).hide();
    //}
  }
}
```

activate_dom_widgets

Se marcan como visibles los injertables cuyo estado ('pn-state') debe estar activado, o sea, cuyo estado está en la lista de *active_states*.

Excluimos los injertables que tienen informado la etiqueta gancho *pn-hook*, que indica que el injertable se utiliza como ancla para concatenar sobre él injertables dinámicos obtenidos por clonado, por ejemplo las tarjetas de datos o gráficos. Estos injertables modelo no deben estar visibles nunca.

Este es un injertable tipo gancho:

```
<div pn-ref="plasmidGraph" pn-hook="0" pn-multi pn-state="onSearch"></div>
```

Tampoco deben estar visibles los injertables que sirven de molde de clonado, que son aquellos que contienen la etiqueta *pn-multi*, y carecen de referencia incremental (*seq_ref*). Esta referencia incremental se genera automáticamente cada vez que se crea un clon.

Este es un injertable clon que debe visualizarse (cuando el estado de la aplicación sea *onSearch*). El sufijo numérico de la referencia *pn-ref* es el que nos indica que se trata de un clon.

```
<div pn-ref="plasmidGraph_1" pn-multi pn-state="onSearch">
...
</div>
```

Este es un injertable de origen de clonado que no debe visualizarse. En este caso no existe sufijo numérico en la referencia.

```
<div pn-ref="plasmidGraph" pn-multi pn-state="onSearch">
...
</div>
```

Esta función sólo opera con los injertables ya presentes en el *DOM*.

```
function activate_dom_widgets(active_states) {
  let included_objects = $();
  for (let i in active_states) {
    included_objects = included_objects.add($("[pn-state=" + active_states[i] +
    ↪ "]"));
  }
  included_objects = included_objects.not("[pn-hook]");
  for (let i = 0; i < included_objects.length; i++) {
    var included_object = included_objects[i];
    var object_ref = parse_ref(included_object.outerHTML);
    let pn_multi = $(included_object).attr('pn-multi');
    var seq_ref = object_ref.id.split('_')[1];
    // Exclude
    if (pn_multi === "") {
      if (seq_ref) $(included_object).show();
      else $(included_object).hide();
    } else {
      $(included_object).show();
    }
  }
}
```

activate_widgets

A diferencia de la función *activate_dom_widgets* esta función también activa los injertables que no se han incorporado al *DOM* todavía. Para ello debe descargar los injertables desde las bases de datos, bien de la caché de injertables o desde el servidor dependiendo de las prioridades definidas. Sólo se descargan los injertables de los estado activos.

Los injertables a recuperar son todos aquellos compatibles con los estados indicados en *active_states* y marcados como *pn_hook*, es decir injertables gancho. El *html* de los injertables gancho será reemplazado por el *html* descargado.

La descarga sustituye también en el *DOM* los injertables gancho por los injertables descargados.

Al finalizar la activación contamos con un *DOM* completo, no totalmente visualizable: los injertables que son molde de clonado no se dibujan en la pantalla.

```
function activate_widgets(active_states) {
  return new Promise((resolve, reject) => {
    // Show hidden objects already loaded in dom
    active_states = ['all'].concat(active_states);
    activate_dom_widgets(active_states);
    let included_objects = $();
    // Find hook widgets
    for (let i in active_states) {
      included_objects = included_objects.add($("[pn-hook][pn-state=" +
    ↪ active_states[i] + "]"));
    }
    // Create the promises to download the new widgets from the BD (or caché see
    ↪  _get_content_)
    if (included_objects.length > 0) {
      let pn_loads = []; // Promise array with all the load intructions
      let pn_refs = [];
      for (let i = 0; i < included_objects.length; i++) {
        var included_object = included_objects[i];
        var object_ref = parse_ref(included_object.outerHTML);
        pn_loads.push(get_content(object_ref.id, object_ref.state));
        pn_refs.push(object_ref);
      }
      axios.all(pn_loads).then(axios.spread((...responses) => {
        for (let i in responses) {
          let pn_ref = responses[i].data.pn_ref;
          let pn_content = responses[i].data.pn_content;
          // Update widget in DOM
          let content_dom = $(`[pn-hook][pn-ref="${pn_ref}"]`);
          content_dom.replaceWith(pn_content);
          // Update widget in cache
          loki_upsert_widget(pn_ref, pn_refs[i].state, responses[i].data);
          // Activate transitions && actions
          activate_sidebar_toggle(pn_ref);
          activate_transitions(pn_ref);
          activate_actions(pn_ref);
        }
        activate_widgets(active_states)
        .then(
          function(value) {
            DEBUG && console.log(value);
            resolve(value);
          },
          function(reason) {
```

```
            DEBUG && console.log(reason);
            reject(reason);
          }
        );
      })).catch(errors => {
        ERROR && console.log(errors); // react on errors.
        reject("NOK");
      });
    } else {
      resolve('No content to process');
    }
  });
}
```

generate_inner_widget_structure

Aquí generamos el *html* interno del injertable complejo (apoyado en las librerías *datatables* o *d3js*).

Si la respuesta es correcta se ejecuta el método pasado por parámetro *search_service* que utilizará las librerías para terminar de construir el injertable.

El método *search_service* tiene el mismo nombre que el servicio utilizado previamente para obtener los datos del servidor.

Aprovechamos en este punto para lanzar una carga dinámica diferida de los módulos que generan las estructura interna.

Como comentamos en el diseño de alto nivel, la carga es asíncrona, o mejor dicho, se efectúa en paralelo a la llamada que se realiza en la última línea. En la siguiente invocación que se realice del módulo actualizado estaría disponible la nueva versión.

```
function generate_inner_widget_structure(pn_ref, new_seq_ref, query,
    ↪ search_service, data) {
  INFO && console.log('generate_inner_widget_structure', pn_ref, new_seq_ref);
  if (search_service === "hierarchy") {
    load_module('plasmidnet_graphs.js');
    $(`#plasmidGraph_${new_seq_ref} [pn-option="plasmidForceNetwork"]`).hide();
  } else if (search_service === "force_network") {
    load_module('plasmidnet_graphs.js');
  } else if (search_service === "datatable") {
    load_module('plasmidnet_datatables.js');
    $('#query_string_' + new_seq_ref).text(query);
  }
  dyn[search_service](data.data, pn_ref, new_seq_ref, query);
}
```

create_widget

Esta función genera los injertables que son clones de un injertable de tipo *plasmidData* o de tipo *plasmidGraph*, además de los clones de sus enlaces asociados en *sidebar*.

Asigna a cada nuevo clon un número de secuencia incremental para diferenciarlo del resto de clones y del molde de clonado.

El molde de clonado permanece en el *DOM* no visible, no es sustituido por ninguno de sus clones y se utiliza como referencia de inserción de los clones que se van generando por este medio (*insertion_point*)

Por último debemos añadir toda la funcionalidad del injertable. En nuestro caso tenemos un tipo de injertable digamos sencillo, *html* puro que son los enlaces desde el *sidebar* y dos tipos de injertables complejos, el de hojas de datos apoyado en el paquete *dataTables* y el de gráficos *SVG* apoyado en el paquete *d3js* (*generate_inner_widget_structure*).

```
function create_widget(pn_action_ref, query, search_service, data) {
  var included_objects = $("[pn-ref*=" + pn_action_ref + "]");
  INFO && console.log('create_widget included_objects', included_objects);
  var root_refs = {};
  // Determine next sequence
  for (let i = 0; i < included_objects.length; i++) {
    var included_object = included_objects[i];
    var object_ref = parse_ref(included_object.outerHTML);
    var ref_parts = object_ref.id.split('_');
    var seq_ref = parseInt(ref_parts[1]) || 0;
    var root_ref = ref_parts[0];
    if (!(root_refs[root_ref])) root_refs[root_ref] = seq_ref;
    if (seq_ref > root_refs[root_ref]) {
      root_refs[root_ref] = seq_ref;
    }
    DEBUG && console.log("create_widget root_refs", root_refs);
  }
  // Modify template html to generate clon html
  for (let root_ref in root_refs) {
    let seq_ref = root_refs[root_ref];
    var new_seq_ref = seq_ref + 1;
    var new_id = root_ref + "_" + new_seq_ref;
    let widget = loki_get_widget(root_ref, object_ref.state);
    let html_new = widget.pn_content;
    html_new = replace_tag_append('pn-link href', html_new, '_' + new_seq_ref, '
    ↪ ="', '"');
    html_new = replace_tag_append('pn-id id', html_new, '_' + new_seq_ref, '="',
    ↪  '"');
    html_new = replace_tag_append('pn-cmod for', html_new, '_' + new_seq_ref, '
    ↪ ="', '"');
    html_new = replace_tag_append('pn-title', html_new, ' ' + new_seq_ref, '>',
    ↪ '<');
    html_new = replace_tag_append('pn-ref', html_new, '_' + new_seq_ref, '="', '
    ↪ "');
    html_new = replace_tag_append('pn-cmod pn-action-ref', html_new, '_' +
    ↪ new_seq_ref, '="', '"');
    html_new = replace_tag('pn-hook', html_new, new_seq_ref);
```

```
    var insertion_point = $('[pn-ref="' + root_ref + '"]');
    // Generate inner widget html and functions
    $(html_new).insertAfter(insertion_point);
    activate_actions(new_id);
    activate_transitions(new_id);
    loki_upsert_widget(new_id, object_ref.state, {pn_content: html_new});
    if (root_ref === 'plasmidGraph' || root_ref === 'plasmidData') {
      generate_inner_widget_structure(new_id, new_seq_ref, query, search_service
    ↪ , data);
    }
  }
}
```

B.2.3. plasmidnet_loki.js

Este módulo incluye todas las funciones que gestionan la base de datos local **LokiJS**. En este repositorio se incluyen dos tipos de objetos: injertables y datos, cada uno en su colección.

loki_upsert_widget

En esta función actualizamos o insertamos si no existe un objeto injertable. La clave del objeto es *pn_ref* y *pn_state* y por ella se determina si existe o no previamente en la BD. Los campos pn_ref y pn_state se correponden con *pn-ref* y *pn-state* que se utilizan en el *html* de la página. El estado es parte de la clave porque un mismo injertable puede tener varias versiones de visualización dependiendo del estado. Se trata en realidad de dos injertables distintos y podrían etiquetarse de otra forma, pero así permite manejar funciones comunes basado sólo en *pn_ref*.

Cuadro B.1: Campos del objeto *widget*.

campo	descripción
pn_ref	referencia del injertable
pn_state	estado del injertable
pn_content	html
pn_version	version de la aplicación
pn_subversion	versión del injertable
pn_reload_pending	*true* si existe una nueva versión del injertable disponible

```
function loki_upsert_widget(pn_ref, pn_state, widget) {
  INFO && console.log("plasmidnet_loki.loki_upsert_widget ", pn_ref, pn_state,
    ↪ widget);
  if (widget.pn_reload_pending === undefined) widget.pn_reload_pending = false;
  let pn_object = {pn_ref: pn_ref, pn_state: pn_state, pn_content: widget.
    ↪ pn_content, pn_version: widget.pn_version, pn_subversion: widget.
    ↪ pn_subversion, pn_reload_pending: widget.pn_reload_pending};
  let pn_old = db_widgets.findOne({pn_ref: pn_object.pn_ref, pn_state: pn_object
    ↪ .pn_state});
  if (pn_old) {
```

```
    let pn_new = pn_old;
    for (let key in pn_object) {
      pn_new[key] = pn_object[key];
    }
    db_widgets.update(pn_new);
  } else {
    db_widgets.insert(pn_object);
  }
}
```

loki_get_widget

En esta función recuperamos un injertable desde la base de datos a partir de su clave.

```
function loki_get_widget(pn_ref, pn_state) {
  let pn_object = db_widgets.findOne({pn_ref: pn_ref, pn_state: pn_state});
  INFO && console.log("plasmidnet_loki.loki_get_widget", pn_object);
  if (pn_object) {
    return pn_object;
  } else {
    return null;
  }
}
```

loki_upsert_info

Actualización o inserción de los datos de una consulta *info*, que están en formato json. La clave está formada por el nombre del servicio utilizado en la recuperación de datos *search_service* y por la propia consulta *query* de la que eliminamos los espacios para evitar en lo posible almacenar conjuntos de datos idénticos para dos consultas que son la misma. El servicio es necesario porque el objeto de datos devuelto por el servidor depende de si el dato se va a utilizar por un determinado tipo de gráfico o dentro de una tabla.

```
function loki_upsert_info(search_service, query, info) {
  // Upsert with query without spaces to have the same key for the same data
  query = query.replace(REGEXP_SPACES, '');
  INFO && console.log('plasmidnet_loki.loki_upsert_info', search_service, query,
    ↪  info);
  let pn_info_object = {search_service: search_service, query: query, info: info
    ↪ };
  let pn_old = db_info.findOne({search_service: search_service, query: query});
  if (pn_old) {
    let pn_new = pn_old;
    for (let key in pn_info_object) {
      pn_new[key] = pn_info_object[key];
    }
    db_info.update(pn_new);
```

```
  } else {
    db_info.insert(pn_info_object);
  }
}
```

loki_get_info

Recupera un objeto de datos desde el cache **LokiJS** a partir de su clave *search_service* y *query*.

```
function loki_get_info(search_service, query) {
  // get with query without spaces
  query = query.replace(REGEXP_SPACES, '');
  let pn_info_object = db_info.findOne({search_service: search_service, query:
    ↪ query});
  INFO && console.log("plasmidnet_loki.loki_remove_widgetloki_get_info",
    ↪ pn_info_object);
  if (pn_info_object) {
    return pn_info_object;
  } else {
    return null;
  }
}
```

loki_get_widgets_by_state

Recupera todos los injertables asociados a un estado *pn_state*.

```
function loki_get_widgets_by_state(pn_state) {
  return db_widgets.find({pn_state : pn_state}) || {};
}
```

loki_remove_widget

Borra el injertable de clave *pn_ref* y pn_state_.

```
function loki_remove_widget(pn_ref, pn_state) {
  let pn_object = db_widgets.findOne({pn_ref: pn_ref, pn_state: pn_state});
  INFO && console.log("plasmidnet_loki.loki_remove_widget ", pn_object);
  if (pn_object) {
    db_widgets.remove(pn_object);
  } else {
    ERROR && console.log('plasmidnet_loki.loki_remove_widget ERROR removing
    ↪ object');
  }
```

```
}
```

loki_load_db

Arranca la base de datos *LokiJS*. El arranque es asíncrono, lo encapsulamos en una *Promise* que se devuelve al llamante. Hemos renunciado a utilizar el adaptador *LokiIndexedAdapter* porque no funciona en la versión actual de *Firefox*, sí en la versión *nightly*, y no presenta problemas en *Safari* o *Chrome*.

```
function loki_load_db() {
  let promise = new Promise((resolve, reject) => {
    //let adapter = new LokiIndexedAdapter(); //incompatible con Firefox 73.08b
    db = new loki("plasmidnet.db", {
      autosave: true,
      autosaveInterval: 4 * 1000
      //adapter: adapter
    });

    db.loadDatabase({}, (err) => {
      if (err) {
        INFO && console.log('plasmidnet_loki.loadDatabase', err);
        reject('LOKI DB loading failed');
      } else {
        db_widgets = db.getCollection("widgets") || db.addCollection("widgets",
    ↪ { indices: ['pn_ref', 'pn_state'] });
        db_info = db.getCollection("info") || db.addCollection("info", { indices
    ↪ : ['pn_ref'] });
        DEBUG && console.log("plasmidnet_loki.loki_load_db reenter", db_widgets.
    ↪ find(), db_info.find());
        resolve("LOKI DB loaded");
      }
    });
  });
  return promise;
}
```

cache_reload

Comprueba si existe una versión del injertable en el servidor distinta de la que tenemos en local, para ello envía al servidor la clave del injertable *pn_ref* y la versión del mismo (*subversion*).

```
function cache_reload(widget, subversion) {
  fetch(`/plas/service/cache/reload`, { headers:{
    'x-plasmidnet-subversion': subversion,
    'x-plasmidnet-object': widget.pn_ref
  }})
  .then(function(response) {
```

```
    return response.json(); // Is a Promise
  })
  .then(function(data) {
    DEBUG && console.log('plasmidnet_loki.cache_reload ', widget.pn_ref,
    ↪ subversion, data.reload);
    widget.pn_reload_pending = data.reload;
    loki_upsert_widget(widget.pn_ref, widget.pn_state, widget);
  })
  .catch(function(error) {
    ERROR && console.log('plasmidnet_loki.cache_reload ERROR: ' + error.message)
    ↪ ;
  });
}
```

get_content

Carga el injertable referenciado por *content_name* y el estado *state* desde el gestor de contenidos o desde la caché local (almacenada en *LokiJS*). Tiene preferencia el contenido en caché salvo que está activado el indicador correspondiente a la base de datos de injertables local (posición 1 de la matriz *bypass_cache*).

```
function get_content(content_name, state) {
  let stored_widget = loki_get_widget(content_name, state);
  INFO && console.log('plasmidnet_loki get_content', content_name, state,
    ↪ stored_widget);
  if (bypass_cache[1] === 'N' && stored_widget && !stored_widget.
    ↪ pn_reload_pending) {
    // Need reload?
    cache_reload(stored_widget, stored_widget.pn_subversion);
    DEBUG && console.log('plasmidnet_loki get_content: widget from cache:',
    ↪ bypass_cache, content_name);
    return new Promise((resolve, reject) => {resolve( {data : stored_widget} )})
    ↪ ;
  } else {
    DEBUG && console.log('plasmidnet_loki get_content: widget from server:',
    ↪ content_name);
    let data_provider = "";
    return axios.get(`${data_provider}/plas/service/cms/content/${content_name}`
    ↪ , {});
  }
}
```

Dinamización de la función *loki_get_info* que es utilizada por módulos dinámicos y para ello es necesario declararla en el objeto global *dyn*. En caso contrario no podría ser utilizada por los módulos dinámicos.

```
dyn.loki_get_info = loki_get_info;
```

get_data

Carga los datos referenciados por *search_service* y *query* desde la base de datos del servidor o desde la caché local (almacenada en *LokiJS*). Tiene preferencia el contenido en caché salvo que está activado el indicador correspondiente a la base de datos local (posición 2 de la cadena *bypass_cache*);

```
function get_data(search_service, query, bypass_cache) {
  let info_object = dyn.loki_get_info(search_service, query);
  INFO && console.log('get_data', info_object);
  if (bypass_cache[2] === 'N' && info_object && info_object.info) {
    DEBUG && console.log('get_data: data from cache:', info_object.query);
    return new Promise((resolve, reject) => {resolve({data : info_object.info})
    ↪ });
  } else {
    DEBUG && console.log('get_data: data from server:', query);
    return axios.post(`/plas/service/${search_service}`, {
      query: query
    });
  }
}
```

datatable

Recibe los datos *data* como objetos *javascript* devueltos por el servidor y los carga en el objeto *Datatable*, que construye el html dinámico necesario para el manejo de los datos.

Con *pn_ref* y *new_seq_ref* se identifica el objeto del *DOM* donde debe visualizarse la información y la *query* se recibe la consulta original con el fin de mostrarla en lo alto de la tarjeta.

Esta función dibuja la tabla correspondiente a una consulta que devuelve los datos de plásmido, superfamilia y módulo, ocultando por defecto la columna superfamilia (que se puede mostrar después en la interfaz). También admite resultados de consultas con sólo el código de plásmido y el módulo.

Nuevas consultas requerirían funciones similares.

```
function datatable(data, pn_ref, new_seq_ref, query) {
  INFO && console.log('datatable @@H@@');
  $.fn.dataTable.ext.errMode = 'none';
  let table = $('#dataTable_' + new_seq_ref).DataTable({
    processing: true,
    serverSide: false,
    responsive: true,
    dom: 'Bfrltip',
    buttons: {
      name: 'primary',
      buttons: [
        'colvis',
        'pdf',
        'print'
```

```
      ],
      dom: {
        container: {
          className: 'dt-buttons flex-wrap btn-block'
        },
        button: {
          className: 'btn btn-primary'
        },
        collection: {
          className: 'btn btn-primary'
        }
      }
    },
    searching: true,
    lengthMenu: [[5, 10, 15, 25, 50, 100], [5, 10, 15, 25, 50, 100]],
    data: data,
    columns: [
      { data: 'plasmid_id' },
      { data: 'module_id' },
      { data: 'superfam_id' }
    ]
  });
  // If the column is present we hide it.
  if (!query.includes('superfam_id')){
    var column = table.column(2);
    column.visible(false);
  } else {
    var column = table.column(2);
    column.visible(true);
  }
  table.buttons().container().appendTo( $('.col-sm-6:eq(0)', table.table().
    ↪ container() ) );
}
```

Exportación de las funciones dinámicas.

```
dyn.datatable = datatable;
dyn.get_data = get_data;
```

B.2.4. plasmidnet_graphs.js

En este módulo definimos las funciones relacionadas con los gráficos vectoriales **d3js**.

```
const dark = [
  '#B08B12',
  '#BA5F06',
  '#8C3B00',
  '#6D191B',
```

```
  '#842854',
  '#5F7186',
  '#193556',
  '#137B80',
  '#144847',
  '#254E00'
];

const mid = [
  '#E3BA22',
  '#E58429',
  '#BD2D28',
  '#D15A86',
  '#8E6C8A',
  '#6B99A1',
  '#42A5B3',
  '#0F8C79',
  '#6BBBA1',
  '#5C8100'
];

const light = [
  '#F2DA57',
  '#F6B656',
  '#E25A42',
  '#DCBDCF',
  '#B396AD',
  '#B0CBDB',
  '#33B6D0',
  '#7ABFCC',
  '#C8D7A1',
  '#A0B700'
];

const palettes = [light, mid, dark];
const lightGreenFirstPalette = palettes
  .map(d => d.reverse())
  .reduce((a, b) => a.concat(b));

const darkFirstPalette = [dark, mid, light]
    .map(d => d.reverse())
    .reduce((a, b) => a.concat(b));
```

force_network

A partir de los datos en formato json *data* que devuelve el servicio del mismo nombre *force_network*, y con la clave identificativa de la tarjeta *html pn_ref* y *seq_ref*, construye el gráfico vectorial de tipo *network* donde la posición idónea de los nodos se calcula mediante la simulación de fuerzas físicas.

El gráfico se liga a un conjunto de controles que permiten modificar interactivamente la presentación: tamaño de los nodos, tamaño de las agrupaciones, escala global, forma de las líneas de agrupación y mostrar u ocultar las etiquetas de los nodos.

```
function force_network(data, pn_ref, seq_ref) {
  const DEFAULT_RADIUS = 12;
  const SCALE_FACTOR_INIT = 1.2;

  // Neccessary because force field updates the link and affect the data saved
    ↪ in localstorage cache
  let graph = JSON.parse(JSON.stringify(data));
  DEBUG && console.log('generate_graph_force_network', pn_ref, seq_ref, graph);
  let width = 2000;
  let height = 2000;
  //var svg = d3.select("#graph").append('svg').attr("width", width).attr("
    ↪ height", height).style("font", "12px Nunito");
  d3.select("#" + pn_ref).select(".graph").html("");
  let svg = d3.select("#" + pn_ref).select(".graph").append('svg').attr("viewBox
    ↪ ", [0, 0, width, height]).style("font", "12px Nunito");
  const forceX = d3.forceX(width / 2).strength(0.020);
  const forceY = d3.forceY(height / 2).strength(0.020);
  const color = d3.scaleOrdinal(darkFirstPalette);

  var valueline = d3.line()
      .x(function(d) { return d[0]; })
      .y(function(d) { return d[1]; })
      .curve(d3.curveBasisClosed),
    paths,
    groups,
    groupIds,
    scaleFactor = SCALE_FACTOR_INIT,
    polygon,
    centroid,
    node,
    link,
    simulation = d3.forceSimulation()
      .force('link', d3.forceLink().id(function(d) { return d.id; }))
      .force('charge', d3.forceManyBody())
      .force('x', forceX)
      .force('y',  forceY)
      .force('center', d3.forceCenter(width / 2, height / 2));

  $('#scaleFactorSettings_' + seq_ref).on('input', function() {
    DEBUG && console.log('clicked', $(this).val());
    scaleFactor = $(this).val();
    //d3.select('#scaleFactorLabel').text(scaleFactor);
    updateGroups();
  });

  $('#curveSettings_' + seq_ref).on('change', function() {
    DEBUG && console.log('select', $(this).val());
    valueline.curve(d3[$(this).val()]);
```

```
  updateGroups();
});

$('#scaleFactorNodeSettings_' + seq_ref).on('input', function() {
  DEBUG && console.log('clicked', $(this).val());
  d3.select("#" + pn_ref).select(".graph").selectAll('circle').attr('r', $(
  ↪ this).val());
  d3.select("#" + pn_ref).select(".graph").selectAll('.nodes g text').attr('x'
  ↪ , $(this).val());
  //d3.select('#scaleFactorLabel').text(scaleFactor);
  updateGroups();
});

$('#nodeSwitch_' + seq_ref).on('change', function (event, state) {
  let checked = $(this).is(':checked');
  let newOpacity = checked ? 1 : 0;
  DEBUG && console.log('select', newOpacity);
  d3.select("#" + pn_ref).select(".graph").selectAll('.nodes g text').style("
  ↪ opacity", newOpacity);
});

$('#groupSwitch_' + seq_ref).on('change', function (event, state) {
  let checked = $(this).is(':checked');
  let newOpacity = checked ? 1 : 0;
  DEBUG && console.log('select', newOpacity);
  d3.select("#" + pn_ref).select(".graph").selectAll('.groups g text').style("
  ↪ opacity", newOpacity);
});

var zoom = d3.zoom()
  .scaleExtent([.1, 4])
  .on("zoom", zoomed);

var container = svg.append("g");

svg.call(zoom);

var slider = d3.select('#scaleGlobalSettings_' + seq_ref)
  .datum({})
  .attr("type", "range")
  .attr("value", 1.0)
  .attr("min", zoom.scaleExtent()[0])
  .attr("max", zoom.scaleExtent()[1])
  .attr("step", (zoom.scaleExtent()[1] - zoom.scaleExtent()[0]) / 100)
  .on("input", slided);

groups = container.append('g').attr('class', 'groups');

link = container.append('g')
    .attr('class', 'links')
  .selectAll('line')
```

```
  .data(graph.links)
  .enter().append('line')
    .attr('stroke-width', function(d) { return Math.sqrt(d.value); });

var g = container.append("g")
    .attr("class", "nodes")
  .selectAll("g")
  .data(graph.nodes)
  .enter().append("g")

var node = g.append("circle")
    .attr("r", DEFAULT_RADIUS)
    .attr("fill", function(d) { return color(d.group); })
    .call(d3.drag()
        .on("start", dragstarted)
        .on("drag", dragged)
        .on("end", dragended));

// Count members of each group. Groups with less
// than 3 member will not be considered (creating
// a convex hull need 3 points at least)
groupIds = d3.set(graph.nodes.map(function(n) { return n.group; }))
  .values()
  .map( function(groupId) {
    return {
      groupId : groupId,
      count : graph.nodes.filter(function(n) { return n.group == groupId; }).
 ↪ length
    };
  })
  .filter( function(group) { return group.count > 2;})
  .map( function(group) { return group.groupId; });

paths = groups.selectAll('.path_placeholder')
  .data(groupIds, function(d) { return +d; })
  .enter()
  .append('g')
    .attr('class', 'path_placeholder')
  .append('path')
    .attr('stroke', function(d) { return color(d); })
    .attr('fill', function(d) { return color(d); })
    .attr('opacity', 0)

paths
  .transition()
  .duration(2000)
  .attr('opacity', 1);

// add interaction to the groups
groups.selectAll('.path_placeholder')
  .call(d3.drag()
```

```
    .on('start', group_dragstarted)
    .on('drag', group_dragged)
    .on('end', group_dragended)
  );

g.append("text")
  .text(function(d) { return d.group + " " + d.id; })
  .style("font", "12px Nunito")
  .attr("x", DEFAULT_RADIUS)

node.append('title')
  .text(function(d) { return d.group + " " + d.id; });

groups.selectAll('.path_placeholder').append('title')
  .text(function(d) { return d; });

groups.selectAll('.path_placeholder').append('text')
  .text(function(d) { return d; })
  .style("font", "16px Nunito")
  .style("font-weight", "bold")
  .attr('x', 0)
  .attr('y', 0);

simulation
  .nodes(graph.nodes)
  .on('tick', ticked)
  .force('link')
  .links(graph.links);

function ticked() {
  link
      .attr('x1', function(d) { return d.source.x; })
      .attr('y1', function(d) { return d.source.y; })
      .attr('x2', function(d) { return d.target.x; })
      .attr('y2', function(d) { return d.target.y; });
  node
      .attr('cx', function(d) { return d.x; })
      .attr('cy', function(d) { return d.y; });

  g.select("text")
    .attr('transform', (function(d) { return `translate(${d.x}, ${d.y})`; }));

  updateGroups();
}

// Select nodes of the group, retrieve its positions and return the convex
  ↪ hull of the specified points (3 points as minimum, otherwise returns null
var polygonGenerator = function(groupId) {
  var node_coords = node
    .filter(function(d) { return d.group == groupId; })
    .data()
```

```
    .map(function(d) { return [d.x, d.y]; });
  return d3.polygonHull(node_coords);
};

function updateGroups() {
  groupIds.forEach(function(groupId) {
    var path = paths.filter(function(d) { return d == groupId;})
      .attr('transform', 'scale(1) translate(0,0)')
      .attr('d', function(d) {
        polygon = polygonGenerator(d);
        centroid = d3.polygonCentroid(polygon);
        // to scale the shape properly around its points:
        // move the 'g' element to the centroid point, translate
        // all the path around the center of the 'g' and then
        // we can scale the 'g' element properly
        return valueline(
          polygon.map(function(point) {
            return [  point[0] - centroid[0], point[1] - centroid[1] ];
          })
        );
      });
    d3.select(path.node().parentNode).attr('transform', 'translate('  +
  ↪ centroid[0] + ',' + (centroid[1]) + ') scale(' + scaleFactor + ')');
  });
}
// drag nodes
function dragstarted(d) {
  //d3.event.sourceEvent.stopPropagation();
  d3.event.sourceEvent.stopPropagation();
  if (!d3.event.active) simulation.alphaTarget(0.3).restart();
  d.fx = d.x;
  d.fy = d.y;
}

function dragged(d) {
  d.fx = d3.event.x;
  d.fy = d3.event.y;
}

function dragended(d) {
  if (!d3.event.active) simulation.alphaTarget(0);
  d.fx = null;
  d.fy = null;
}

// drag groups
function group_dragstarted(groupId) {
  d3.event.sourceEvent.stopPropagation();
  if (!d3.event.active) simulation.alphaTarget(0.3).restart();
  d3.select(this).select('path').style('stroke-width', 3);
}
```

```
  function group_dragged(groupId) {
    node
      .filter(function(d) { return d.group == groupId; })
      .each(function(d) {
        d.x += d3.event.dx;
        d.y += d3.event.dy;
      })
  }

  function group_dragended(groupId) {
    if (!d3.event.active) simulation.alphaTarget(0.3).restart();
    d3.select(this).select('path').style('stroke-width', 1);
  }

  function zoomed(){
    container.attr("transform", d3.event.transform);
    slider.property("value", d3.event.transform.k);
  }

  function slided(d) {
    zoom.scaleTo(svg, d3.select(this).property("value"));
  }
}
```

hierarchy

A partir de los datos en formato json *data* que devuelve el servicio del mismo nombre *force_network*, y con la clave identificativa de la tarjeta html *pn_ref*, construye el gráfico vectorial de tipo *sunburst*.

```
function hierarchy(data, pn_ref) {
  DEBUG && console.log('generate_graph_d3js_sunburst', pn_ref, data);
  var height = 600;
  var width  = 600;
  const maxRadius = Math.min(width, height) / 2 - 5;
  const formatNumber = d3.format(',d');
  const x = d3
    .scaleLinear()
    .range([0, 2 * Math.PI])
    .clamp(true);

  const y = d3.scaleSqrt().range([maxRadius * 0.1, maxRadius]);

  const color = d3.scaleOrdinal(lightGreenFirstPalette);

  const partition = d3.partition();

  const arc = d3
```

```
  .arc()
  .startAngle(d => x(d.x0))
  .endAngle(d => x(d.x1))
  .innerRadius(d => Math.max(0, y(d.y0)))
  .outerRadius(d => Math.max(0, y(d.y1)));

const middleArcLine = d => {
  const halfPi = Math.PI / 2;
  const angles = [x(d.x0) - halfPi, x(d.x1) - halfPi];
  const r = Math.max(0, (y(d.y0) + y(d.y1)) / 2);

  const middleAngle = (angles[1] + angles[0]) / 2;
  const invertDirection = middleAngle > 0 && middleAngle < Math.PI; // On
  ↪ lower quadrants write text ccw
  if (invertDirection) {
    angles.reverse();
  }

  const path = d3.path();
  path.arc(0, 0, r, angles[0], angles[1], invertDirection);
  return path.toString();
};

const textFits = d => {
  const CHAR_SPACE = 6;

  const deltaAngle = x(d.x1) - x(d.x0);
  const r = Math.max(0, (y(d.y0) + y(d.y1)) / 2);
  const perimeter = r * deltaAngle;

  return d.data.child.length * CHAR_SPACE < perimeter;
};

// Remove previous content (spinner)
d3.select("#" + pn_ref).select(".graph").html("");
const svg = d3.select("#" + pn_ref).select(".graph").append("svg")
  .style("font", "8px Nunito")
  .attr('viewBox', `${-width / 2} ${-height / 2} ${width} ${height}`)
  .on('click', () => focusOn()); // Reset zoom on canvas click

let stratify = d3.stratify()
  .id(d => d["child"])
  .parentId(d => d["parent"]);

let root_raw = stratify(data);

DEBUG && console.log('generate_graph_d3js_sunburst root', root_raw);

let root = root_raw.copy().count();
DEBUG && console.log('generate_graph_d3js_sunburst root', root);
```

```
const slice = svg.selectAll('g.slice').data(partition(root).descendants());

slice.exit().remove();

const newSlice = slice
  .enter()
  .append('g')
  .attr('class', 'slice')
  .on('click', d => {
    d3.event.stopPropagation();
    focusOn(d);
  });

newSlice
  .append('title')
  .text(d => d.data.child + ' ' + d.data.info);

newSlice
  .append('path')
  .attr('class', 'main-arc')
  .style('fill', d => color((d.children ? d : d.parent).data.child))
  .attr('d', arc);

newSlice
  .append('path')
  .attr('class', 'hidden-arc')
  .attr('id', (_, i) => `${pn_ref}_hiddenArc${i}`)
  .attr('d', middleArcLine);

const text = newSlice
  .append('text')
  .attr('display', d => (textFits(d) ? null : 'none'));

// Add white contour
text
  .append('textPath')
  .attr('startOffset', '50%')
  .attr('xlink:href', (_, i) => `#${pn_ref}_hiddenArc${i}`)
  .text(d => d.data.child)
  .style('fill', 'none')
  .style('stroke', '#ffffff')
  .style('stroke-opacity', 0.7)
  .style('stroke-width', 8)
  .style('stroke-linejoin', 'round');

text
  .append('textPath')
  .attr('startOffset', '50%')
  .attr('xlink:href', (_, i) => `#${pn_ref}_hiddenArc${i}`)
  .text(d => d.data.child);
```

```
  function focusOn(d = { x0: 0, x1: 1, y0: 0, y1: 1 }) {
    // Reset to top-level if no data point specified
    const transition = svg
      .transition()
      .duration(600)
      .tween('scale', () => {
        const xd = d3.interpolate(x.domain(), [d.x0, d.x1]),
          yd = d3.interpolate(y.domain(), [d.y0, 1]);
        return t => {
          x.domain(xd(t));
          y.domain(yd(t));
        };
      });

    transition.selectAll('path.main-arc').attrTween('d', d => () => arc(d));

    transition
      .selectAll('path.hidden-arc')
      .attrTween('d', d => () => middleArcLine(d));

    transition
      .selectAll('text')
      .attrTween('display', d => () => (textFits(d) ? null : 'none'));

    moveStackToFront(d);

    function moveStackToFront(elD) {
      svg
        .selectAll('.slice')
        .filter(d => d === elD)
        .each(function(d) {
          this.parentNode.appendChild(this);
          if (d.parent) {
            moveStackToFront(d.parent);
          }
        });
    }
  }
}

dyn.force_network = force_network;
dyn.hierarchy = hierarchy;
```

B.2.5. plasmidnet_global.js

En este módulo se definen las funciones relacionadas con los comportamientos generales de la aplicación ante diferentes eventos.

```
(function($) {
```

```
"use strict";

var searchText = "none";
var color = "#c1cae3";
var reset = true;
var elements = [];
var search_pos = 0;

// Toggle the side navigation
$("#sidebarToggle, #sidebarToggleTop").on('click', function(e) {
  $("body").toggleClass("sidebar-toggled");
  $(".sidebar").toggleClass("toggled");
  if ($(".sidebar").hasClass("toggled")) {
    $('.sidebar .collapse').collapse('hide');
  };
});

// Close any open menu accordions when window is resized below 768px
$(window).resize(function() {
  if ($(window).width() < 768) {
    $('.sidebar .collapse').collapse('hide');
  };
});

// Prevent the content wrapper from scrolling when the fixed side navigation
 ↪ hovered over
$('body.fixed-nav .sidebar').on('mousewheel DOMMouseScroll wheel', function(e)
 ↪  {
  if ($(window).width() > 768) {
    var e0 = e.originalEvent,
      delta = e0.wheelDelta || -e0.detail;
    this.scrollTop += (delta < 0 ? 1 : -1) * 30;
    e.preventDefault();
  }
});

// Scroll to top button appear
$(document).on('scroll', function() {
  var scrollDistance = $(this).scrollTop();
  if (scrollDistance > 100) {
    $('.scroll-to-top').fadeIn();
  } else {
    $('.scroll-to-top').fadeOut();
  }
});

// Smooth scrolling using jQuery easing
$(document).on('click', 'a.scroll-to-top', function(e) {
  var $anchor = $(this);
  $('html, body').stop().animate({
    scrollTop: ($($anchor.attr('href')).offset().top)
```

```
    }, 100, 'easeInOutExpo');
    e.preventDefault();
  });

  function contains(text_one, text_two) {
      return text_one.toLowerCase().indexOf(text_two.toLowerCase()) != -1;
  }
```

event searchText.keyup

Mediante la interceptación del evento *keyup* borramos cualquier selección previa en en componente de búsqueda

```
  $("#searchText").on('keyup', function() {
    if (!reset) {
      console.log("reset");
      $("#content span, #content p, #content h1, #content h2").each(function() {
        if (contains($(this).text(), searchText)) {
          $(this).removeAttr("style");
        }
      });
      reset = true;
      elements = [];
    }
  });
```

event do_search.click

Interceptando el evento *click* lanzamos la búsqueda textual en los elementos susceptibles de contener texto. En el caso de encontrarlo el elemento se marca con otro color de fondo.

```
  $("#do_search").on('click', function() {
      searchText = $("#searchText").val();
      console.log(searchText);
      $("#content span, #content p, #content h1, #content h2").each(function() {
          if (contains($(this).text(), searchText)) {
            elements.push($(this));
            $(this).css("background-color", color);
          }
      });
      reset = false;
      search_pos = -1;
  });
```

event do_search.keypress

Desactiva el evento *enter*.

```
  // Deactivate keypresses on
  $("#do_search").on('keypress', function() {
      return false;
  });
```

event search_next_down.click

En este evento se avanza el cursor hasta el siguiente texto encontrado por la búsqueda.

```
  $("#search_next_down").click(function() {
      if (elements.length === 0) return;
      search_pos++;
      search_pos = search_pos % (elements.length);
      $([document.documentElement, document.body]).animate({
        scrollTop: $(elements[search_pos]).offset().top
      }, 500);
  });
```

event search_next_down.click

En este evento se avanza el cursor hasta el texto previo encontrado por la búsqueda.

```
  $("#search_next_up").click(function() {
      if (elements.length === 0) return;
      if (search_pos > 0) search_pos--;
      $([document.documentElement, document.body]).animate({
        scrollTop: $(elements[search_pos]).offset().top
      }, 1000);
  });

})(jQuery); // End of use strict
```

B.2.6. plasmidnet_sw.js

En este módulo gestiona la instalación del *service worker*.

sw_install

Efectúa el registro habitual de un service worker, pero además añade un *listener* de eventos que será disparado por el *service worker* ante la llegada de un mensaje vía *push*.

En el mensaje nos llega la configuración de prioridades de caché *bypass_cache* y la configuración del *log* de errores *log_deb_info_err*.

```
function sw_install() {
  INFO && console.log('sw_install init');
  if ('serviceWorker' in navigator) {
    window.addEventListener('load', function() {
      navigator.serviceWorker.register('/plas/app/sw.js').then(function(
    ↪ registration) {
        // Registration was successful
        INFO && console.log('sw_install werviceWorker registration successful
    ↪ with scope : ', registration.scope);
        navigator.serviceWorker.addEventListener('message', event => {
          INFO && console.log('sw_install ', event.data);
          bypass_cache = event.data.bypass_cache;
          let log_deb_info_err = event.data.log_deb_info_err;
          if (log_deb_info_err) {
            if (log_deb_info_err[0] === 'S') DEBUG = true; else DEBUG = false;
            if (log_deb_info_err[1] === 'S') INFO = true; else INFO = false;
            if (log_deb_info_err[2] === 'S') ERROR = true; else ERROR = false;
          }
        });
      }, function(err) {
        // registration failed :(
        ERROR && console.log('sw_install serviceWorker registration failed: ',
    ↪ err);
      });
    });
  }
}

if (!DEBUG) var DEBUG = false;
if (!INFO) var INFO = false;
if (!ERROR) var ERROR = false;
sw_install();
```

B.2.7. sw.js

Implementamos aquí una parte del sistema de caché, mediante la interceptación de las llamadas *http* desde la aplicación hacia la red (evento *fetch*) y la respuesta al método *push* (con funciones meramente de reconfiguración local).

```
var VERSION = 'v0';
var CACHENAME = 'plasmidnet_cache-' + VERSION;
var SITE_ROOT = '/plas/app';
var bypass_cache = 'NNNN';
var DEBUG = false; // Activate/deactivate console debug
var ERROR = false; // Activate/deactivate console error
var INFO  = false; // Activate/deactivate console info
var FILES = [
  SITE_ROOT + '/manifest.json',
  SITE_ROOT + '/manifest_doc.json',
```

```
  SITE_ROOT + '/css/slides.css',
  SITE_ROOT + '/vendor/all.min.css',
  SITE_ROOT + '/css/plasmidnet.css',
  SITE_ROOT + '/css/plasmidnet.min.css',
  SITE_ROOT + '/vendor/dataTables.min.css',
  SITE_ROOT + '/vendor/jquery.min.js',
  SITE_ROOT + '/vendor/bootstrap.bundle.min.js',
  SITE_ROOT + '/vendor/jquery.easing.min.js',
  SITE_ROOT + '/vendor/axios.min.js',
  SITE_ROOT + '/vendor/lokijs.min.js',
  SITE_ROOT + '/vendor/loki-indexed-adapter.min.js',
  SITE_ROOT + '/vendor/loki-indexed-adapter.js',
  SITE_ROOT + '/vendor/d3.min.js',
  SITE_ROOT + '/js/plasmidnet.min.js',
  SITE_ROOT + '/js/plasmidnet_doc.min.js',
  SITE_ROOT + '/vendor/dataTables.min.js',
  SITE_ROOT + '/webfonts/fa-brands-400.eot',
  SITE_ROOT + '/webfonts/fa-brands-400.pdf',
  SITE_ROOT + '/webfonts/fa-brands-400.ttf',
  SITE_ROOT + '/webfonts/fa-brands-400.woff',
  SITE_ROOT + '/webfonts/fa-brands-400.woff2',
  SITE_ROOT + '/webfonts/fa-regular-400.eot',
  SITE_ROOT + '/webfonts/fa-regular-400.pdf',
  SITE_ROOT + '/webfonts/fa-regular-400.ttf',
  SITE_ROOT + '/webfonts/fa-regular-400.woff',
  SITE_ROOT + '/webfonts/fa-regular-400.woff2',
  SITE_ROOT + '/webfonts/fa-solid-900.eot',
  SITE_ROOT + '/webfonts/fa-solid-900.pdf',
  SITE_ROOT + '/webfonts/fa-solid-900.ttf',
  SITE_ROOT + '/webfonts/fa-solid-900.woff',
  SITE_ROOT + '/webfonts/fa-solid-900.woff2',
  SITE_ROOT + '/vendor/ace.js',
  SITE_ROOT + '/vendor/theme-chrome.js',
  SITE_ROOT + '/vendor/mode-text.js',
  SITE_ROOT + '/vendor/worker-javascript.js'
];

var reload_pending = {};
```

install event

```
self.addEventListener("install", function(event) {
  INFO && console.log('sw.js: event install');
  event.waitUntil(
    caches.open(CACHENAME).then(function(cache) {
      INFO && console.log('sw.js: Opened cache');
      return cache.addAll(FILES);
    })
  );
```

```
});
```

activate event

En este evento aprovechamos para renovar la caché si se ha producido un cambio de nombre. Normalmente lo hacemos provocando un cambio ad hoc en *VERSION*. Es útil para las pruebas, o para soslayar algún error en la versión de software que haya dejado en un estado irreparable las cachés de los navegadores.

Lo que haremos en el futuro es modificar un indicador en el propio servidor, que consultado por el *service worker* desencadenará el borrado.

```
self.addEventListener('activate', function(event) {
  var version = VERSION;
  event.waitUntil(
    caches.keys()
      .then(cacheNames =>
        Promise.all(
          cacheNames
            .map(c => c.split('-'))
            .filter(c => c[0] === 'plasmidnet_cache')
            .filter(c => c[1] !== version)
            .map(c => caches.delete(c.join('-')))
        )
      )
  );
});
```

url_cacheable

Comprueba si una *URL* deba almacenarse o servirse por el caché o no. Se utiliza la lista de estáticos, la ubicación de las fuentes y el servicio de módulos.

```
function url_cacheable(url) {
  var cacheable = false;
  for (var i = 0; i < FILES.length; i++) {
    if (url.indexOf(FILES[i]) >= 0) {
      cacheable = true;
      break;
    }
  }
  // Modules are cacheable also
  if (url.indexOf("service/module") >= 0) cacheable = true;
  // Externals cacheable : fonts from fonts.gstatic.com cacheables
  if (url.indexOf("fonts.gstatic.com") >= 0) cacheable = true;
  if (url.indexOf(SITE_ROOT + "/plasmidnet") >= 0) cacheable = true;
  if (url.indexOf(SITE_ROOT + "/fonts") >= 0) cacheable = true;
  if (url.indexOf(SITE_ROOT + "/images") >= 0) cacheable = true;
```

```
  // Externals from revealjs slides
  if (url.indexOf(SITE_ROOT + "/revealjs.com") >= 0) cacheable = true;
  return cacheable;
}
```

cache_reload

Consulta en el servidor si un determinado elemento *object* necesita ser recargado porque su versión *subversion* no está ya vigente.

```
function cache_reload(object, subversion) {
  fetch(`/plas/service/cache/reload`, { headers:{
    'x-plasmidnet-subversion': subversion,
    'x-plasmidnet-object': object
  }})
  .then(function(response) {
    return response.json();
  })
  .then(function(data) {
    INFO && console.log('sw.cache_reload ', object, subversion, data.reload);
    reload_pending[object] = data.reload;
  })
  .catch(function(error) {
    INFO && console.log('sw.cache_reload ERROR: ' + error.message);
  });
}
```

fetch event

Devuelve el elemento desde el caché o desde la red dependiendo de diversos factores:

1) Si el elemento es almacenable en la caché (llamando a *url_cacheable*).
2) La prioridad de la caché (A, Y o N en la posición correspondiente de *bypass_cache*)
3) Si el servidor contiene una nueva versión del elemento, mediante *cache_reload*.

El valor Y de la caché de estáticos baja la prioridad de la caché de estáticos sólo para los elementos de la aplicación (que contienen *plasmidnet*). El valor A de este elemento indica que la caché de estáticos tiene menor preferencia para todos los elementos.

```
self.addEventListener('fetch', function(event) {
  let url_cache = event.request.url;
  // Modules cache entry without mtime
  let ignoreSearch = false;
  if (event.request.url.indexOf('service/module') >= 0) {
    url_cache = event.request.url.split('?mtime')[0];
    ignoreSearch = true;
  }
```

```
event.respondWith(
  caches.open(CACHENAME).then(function(cache) {
    return cache.match(event.request, {ignoreSearch: ignoreSearch}).then(
↪ function (response) {
      DEBUG && console.log("sw.fetch cache match ", event.request.url,
↪ url_cache);
      let from_cache;
      // Content in cache
      if (response) {
        // Modules
        DEBUG && console.log('sw.fetch subversion ', response.headers.get('x-
↪ plasmidnet-subversion'));
        if (response.url.indexOf('service/module') >= 0) {
          if (bypass_cache[3] && bypass_cache[3] === 'Y') {
            DEBUG && console.log('sw.fetch module from network ' +
↪ bypass_cache + " " + response.url);
            from_cache = false;
          } else {
            DEBUG && console.log('sw.fetch module from cache ' + bypass_cache
↪ + " " + response.url);
            from_cache = true;
          }
        // Statics
        } else {
          if (bypass_cache[0] === 'N') {
            DEBUG && console.log('sw.fetch static from cache ' + bypass_cache
↪ + " " + response.url);
            from_cache = true;
          } else if (bypass_cache[0] === 'A') {
            DEBUG && console.log('sw.fetch static from network ' +
↪ bypass_cache + " " + response.url);
            from_cache = false;
          } else if (bypass_cache[0] === 'Y' && response.url.indexOf('
↪ plasmidnet.') >= 0) {
              // bypass selective
              DEBUG && console.log('sw.fetch static plasmidnet from network '
↪ + bypass_cache + " " + response.url + " " + response.url.indexOf('
↪ plasmidnet.'));
              from_cache = false;
          } else {
            DEBUG && console.log('sw.fetch from cache ' + bypass_cache + " " +
↪  response.url);
            from_cache = true;
          }
        }
      // Content not in cache, must download
      } else {
        from_cache = false;
        DEBUG && console.log('sw.fetch from network because not in cache ' +
↪ event.request.url);
      }
```

```
        if (from_cache) {
          // Verify local version <> server version. Async = not blocking.
          cache_reload(response.url, response.headers.get('x-plasmidnet-
    ↪ subversion'));
          // Check if reload pending
          if (reload_pending[response.url]) {
            from_cache = false;
            reload_pending[response.url] = false;
          } else {
            return response;
          }
        }
        if (!from_cache) {
          return fetch(event.request).then(function(response) {
            if (url_cacheable(response.url)) {
              // Put in cache
              DEBUG && console.log('sw.fetch subversion ', response.headers.get(
    ↪ 'x-plasmidnet-subversion'));
              if (event.request.method != 'POST') cache.put(url_cache, response.
    ↪ clone());
              DEBUG && console.log('sw.fetch from network ' + event.request.url
    ↪ + ' and update cache ' + url_cache);
            } else {
              DEBUG && console.log('sw.fetch from network because not cacheable
    ↪ ' + event.request.url);
            }
            return response;
          });
        }
      });
    })
  );
});
```

push event function

El mensaje recibido es una cadena con esta esta estructura **SWDM_DIE**:

1) **SWDM** se refiere a la habilitación de los distintos cachés. Cada carácter indica si está habilitado el caché de estáticos, widgets, datos o módulos de carga dinámica. Por habilitado se entiende que tiene preferencia el dato en caché frente al dato en el servidor.

2) **DIE** indica si los *console.log* de *DEBUG*, *INFO* o *ERROR* deben habilitarse (*Y*) o no (*N*).

```
self.addEventListener('push', function(event) {
  INFO && console.log(`sw.js: push had this data: "${event.data.text()}"`);
  let message = event.data.text().split('_');
  bypass_cache = message[0];
```

```
  let log_deb_info_err = message[1];
  if (log_deb_info_err) {
    if (log_deb_info_err[0] === 'Y') DEBUG = true; else DEBUG = false;
    if (log_deb_info_err[1] === 'Y') INFO = true; else INFO = false;
    if (log_deb_info_err[2] === 'Y') ERROR = true; else ERROR = false;
  }
  self.clients.matchAll().then(function(clients) {
    clients.forEach(function(client) {
      client.postMessage({bypass_cache: bypass_cache, log_deb_info_err:
    ↪ log_deb_info_err});
    });
  });
});
```

B.3. Servidor *web*

B.3.1. gulpfile_app.js

Este es el módulo principal. Desde aquí el servidor enlaza a los diferentes módulos dinámicos incluidos los encaminadores de *express*, se definen los diferentes puertos y protocolos en los que se arranca el servicio y crea las tareas de arranque.

Express [6] es una infraestructura web de direccionamiento y *middleware*.

```
const uuidv4 = require('uuid');
var compression = require('compression');
const fs = require('fs');
const https = require('https');
const http = require('http');
const express = require('express');
const cors = require('cors');
const DATA_DIR = './DATA';
const DEFAULT_NODE = 'plasmid01';
const DEFAULT_PORT = '9090'; // Web server http default port
const DEFAULT_DOC_PORT = '9099'; // Doc server http default
```

La configuración compacta del servicio define una serie de parejas parámetro valor que permite indicar qué servicios se arrancan.

En la configuración por defecto (válida para pruebas) arrancamos los tres servidores http, damos accesibilidad a las rutas de la *web app*, los servicios *REST* y las *pipelines* y deshabilitamos el *certbot*.

```
const DEFAULT_CONFIG = 'http_Y|httpdoc_Y|https_Y|app_Y|services_Y|pipelines_Y|
    ↪ certbot_N';
const argv = require('yargs')
  .default('node', process.env.plasmidnode)
  .default('port', process.env.port)
  .default('doc_port', process.env.doc_port)
```

```
  .default('config', process.env.config || DEFAULT_CONFIG)
  .argv;
```

Certificados para el servicio https.

```
const privateKey = fs.readFileSync('./DATA/none/privkey.pem', 'utf8');
const certificate = fs.readFileSync('./DATA/none/cert.pem', 'utf8');
const ca = fs.readFileSync('./DATA/none/chain.pem', 'utf8');

const credentials = {
    key: privateKey,
    cert: certificate,
    ca: ca
};
```

Versiones por defecto para la *web app* y para el servidor. Estas son las versiones publicadas y accesibles por un usuario normal de la aplicación.

```
const default_app_version = "v0";
const default_server_version = "v0"
const {require_dyn} = require('../proxy_modules');
```

httpContext es un *middleware* de *express* que nos permite almacenar en un objeto la información que necesitamos que sea accesible en todo el contexto de una *request http*.

```
const httpContext = require('express-http-context');
const cookieParser = require('cookie-parser');
let pn_config = {}; // global config

const app = express();
app.use(httpContext.middleware);
```

Este *middleware* analiza las *cookies* que llegan en la *request* (en una cadena) y las inserta en la misma *request* dentro del objeto *req.cookies*, lo que facilita su análisis por los *middlewares* posteriores.

```
app.use(cookieParser());
```

get_version

Carga en el contexto de una *request* que *express* pasa por parámetro (*req*), la versión de la aplicación y del servidor asociados a la misma. Estos valores se obtienen de la cookie *pn_ version*. Si no viniese la cookie se toman los valores por defecto.

Todos los *middleware* de *express* tienen la misma forma: el sistema nos pasa el objeto *request http* (*req*) y el objeto *response http* (*res*) y un puntero a la siguiente función a ejecutar.

La infraestructura *express* nos permite encadenar varias de estas funciones (*middlewares* en la terminología *express*) de manera que podemos estructurar adecuadamente el código.

El camino de la información es muy sencillo. Tenemos una *request* que nos llega desde la aplicación cliente y vamos construyendo una respuesta en una cadena de pasos antes de devolverla al cliente. También podemos incorporar pasos que no generan ninguna parte de la respuesta, como este mismo o como los que activan los *logs*.

```
app.use(function get_version(req, res, next) {
  var app_version;
  var server_version;
  if (req.cookies !== undefined && req.cookies.pn_version !== undefined) {
    app_version = req.cookies.pn_version.split('_')[0];
    server_version = req.cookies.pn_version.split('_')[1];
    require_dyn("logger_app").info(`version from cookie ${app_version} ${
    ↪ server_version}`);
  } else {
    app_version = default_app_version;
    server_version = default_server_version;
    require_dyn("logger_app").info(`default version ${app_version} ${
    ↪ server_version}`);
  }
  httpContext.set('app_version', app_version);
  httpContext.set('server_version', server_version);
  next();
});
```

asign_uuid

En este *middleware*, al que en el código hemos asociado una función anónima, introducimos en el contexto un identificador único que utilizaremos para referenciar la llamada en el *log* e identificar los pasos de las *pipelines* del sistema de orquestación.

```
app.use(function(req, res, next) {
  req.plasmidnet_reqid = uuidv4();
  httpContext.set('reqId', req.plasmidnet_reqid);
  next();
});
```

Middleware de compresión de la *response*. Activo si en la *request* el cliente informa el campo *filter* en las cabeceras.

```
app.use(compression({ filter: should_compress }));
function should_compress(req, res) {
  if (req.headers['x-compress']) {
    //require_dyn("logger_app").info("Compressing " + req.url);
    return compression.filter(req, res);
  } else {
    //require_dyn("logger_app").info("Not compressing " + req.url);
```

```
    return false;
  }
}
```

Middleware que aplica a cualquier fichero solicitado dentro del directorio *web files* la infraestructura para la descarga de estáticos de *express*. Esto nos evita la necesidad de implementar nosotros mismos servicios basado en *streams*. El directorio físico donde se encuentran los ficheros en el servidor está definido por *DATA_DIR*.

```
app.use('/files', express.static(DATA_DIR));
```

Implementa automáticamente las cabeceras *cors* [45] necesarias para que los servicios *REST* puedan ser consultados desde dominios distintos al nuestro. En principio no tenemos en PlasmidNet ninguna limitación de acceso: todos los dominios origen están permitidos.

```
app.use(cors());
```

Middleware que convierte los cuerpos json de una *request* en objetos *javascript*. Nos evita tener que realizar la conversión.

```
app.use(express.json({ limit: '50mb' }));
```

Middleware que convierte los cuerpos *urlencoded* de una request en objetos *javascript*. Nos evita tener que realizar la conversión. Gracias a estos dos *middlewares* tratamos todas los cuerpos de llamadas de la misma forma en los pasos siguientes de la cadena de *middlewares*. Estos *middlewares* se apoyan en la cabecera *content-type* para decidir si es necesaria la conversión.

```
app.use(express.urlencoded({ limit: '50mb', extended: true }));
```

Analiza el formato de la *URL* de forma que todas las que comienzn por '/plas/app' son encaminadas al módulo de encaminamiento *router.js*, donde se genera la respuesta. Estas *URL* corresponden a la *web app*.

```
app.all(/^\/plas\/app/, function(req, res, next) {
  if (pn_config.app) {
    require_dyn("router")(req, res, next);
  } else {
    res.status(403);
    return res.send({message: "Not authorized", error: "Not authorized"});
  }
});
```

Encamina las *URL* que comienzan por '/plas/service' al módulo *router_services.js* donde se elaborarán las respuestas. Estas *URL* corresponden a los servicios *REST* de la aplicación.

```
app.all(/^\/plas\/service/, function(req, res, next) {
  if (pn_config.services) {
    require_dyn("router_services")(req, res, next);
  } else {
    res.status(403);
    return res.send({message: "Not authorized", error: "Not authorized"});
  }
});
```

Encamina las *URL* que comienzn por '/plas/pipeline' al módulo de encaminamiento *router_pipelines.js* donde se elaborarán las respuestas. Estas *URL* corresponden al protocolo de comunicación entre los nodos implicados en la orquestación.

```
app.all(/^\/plas\/pipeline/, function(req, res, next) {
  if (pn_config.pipelines) {
    require_dyn("router_pipelines")(req, res, next);
  } else {
    res.status(403);
    return res.send({message: "Not authorized", error: "Not authorized"});
  }
});
```

Encaminamiento para la verificación de *host* para la obtención del certificado *Lets-Encrypt*.

```
app.all(/^\/.well-known\/acme-challenge/, function(req, res, next) {
  if (pn_config.certbot) {
    require_dyn("router_crt")(req, res, next);
  } else {
    res.status(403);
    return res.send({message: "Not authorized", error: "Not authorized"});
  }
});
```

Tratamiento de error de objeto no encontrado.

```
app.use(function(req, res, next) {
  let error = new Error('Route '+ req.url);
  error.status = 404;
  throw error;
});
```

Gestión del resto de errores.

```
app.use(function(err, req, res, next) {
  let message;
  let status = err.status || 500;
  res.status(status);
```

```
  if(status === 403) {
     message = 'Action forbidden';
  }
  if(status === 404) {
     message = 'Not found';
  }
  if(status === 500) {
    require_dyn("logger_app").error('gulpfile_app.error_500');
    require_dyn("logger_app").error(err);
    message = 'Internal Server Error';

  }
  return res.send({message: message, error: err.message});
});
```

parse_config

A partir de la configuración en forma compacta genera el objeto *pn_config* para que sea utilizado en el módulo.

```
function parse_config(cb, config=argv.config) {
  let config_params = config.split('|');
  for (param of config_params) {
    let [param_name, param_value] = param.split('_');
    pn_config[param_name] = param_value === 'Y' ? true : false;
  }
  if(cb) {console.log('parse_config Object', pn_config); return cb()};
}
```

register_cleanup

Registra la función de cierre ordenado de los servicios *web*, definida más abajo y captura cualquier posible error no controlado.

```
function register_cleanup(callback) {

  // attach user callback to the process event emitter
  // if no callback, it will still exit gracefully on Ctrl-C
  callback = callback || noOp;
  process.on('cleanup',callback);

  // do app specific cleaning before exiting
  process.on('exit', function () {
    process.emit('cleanup');
  });

  // Signals to trap
```

```
  ['SIGINT', 'SIGTERM', 'SIGUSR1', 'SIGUSR2']
  .forEach(signal => process.on(signal, () => {
    process.exit(2);
  }));

  // Trace exception then normal exit
  process.on('uncaughtException', function(e) {
    console.log('Uncaught Exception...');
    console.log(e.stack);
    process.exit(99);
  });
}
```

cleanup

En esta función programamos una salida ordenada. En ella cerramos todas las bases de datos utilizadas. Es necesario para liberar ficheros como el *Write Ahead Log* que utilizamos para acelerar la concurrencia en *sqlite*.

```
function cleanup() {
  console.log("gulpfile_app exiting closing cms db");
  __sqlite_cms_db.close();
  console.log("gulpfile_app exiting closing bio db");
  __sqlite_bio_db.close();
  console.log("gulpfile_app exiting closing task db");
  __sqlite_db.close();
}
```

start_server

Definición de la tarea *gulp* de arranque de los servidores *http*. El máximo número de servidores que puede arrancarse es: servicio de *web app*, servicio de documentación del proyecto y servicio *https*(443). Depende de la configuración.

```
let servers=[];
function start_server(cb, node=argv.node, port=argv.port, doc_port=argv.doc_port
    ↪ , config=argv.config) {
  parse_config(undefined, config=argv.config);
  node = node || DEFAULT_NODE;
  port = port || DEFAULT_PORT;
  doc_port = doc_port || DEFAULT_DOC_PORT;
  // Register exit cleanup
  register_cleanup(cleanup);

  if (pn_config.http) {
    http.createServer(app).listen(port, function() {
```

```
      require_dyn("logger_app", default_server_version).info(`Web Server ${node}
    ↪  listening on port ${port} configuration ${config}`);
    });
  };
  if (pn_config.httpdoc) {
    http.createServer(app).listen(doc_port, function() {
      require_dyn("logger_app", default_server_version).info(`Web Server ${node}
    ↪  listening on port ${doc_port} configuration ${config}`);
    });
  };
  if (pn_config.https) {
    https.createServer(credentials, app).listen(443, function() {
      require_dyn("logger_app", default_server_version).info(`Web Server ${node}
    ↪  listening on port ${443} configuration ${config}`);
    });
  };
  cb();
}

module.exports.start_server = start_server;
module.exports.parse_config = parse_config;
module.exports.shares = module.exports;
```

B.3.2. routes.js

En este módulo se definen las respuestas ante las solicitudes de ficheros estáticos:

1) *html*. Únicamente la única página de la aplicación *plasmidnet.html* aunque es posible retornarla bajo otros alias como *home.html* o *index.html*.

2) *css*, *js*, imágenes y fuentes de la aplicación.

3) *css*, *js* y fuentes de los diferentes suministradores (*vendors*) de API para el navegador:

- **jQuery**. Modificador del *DOM* que utilizamos para dinamizar la página.
- **bootstrap**. Provee un conjunto de *tags* que adaptan automáticamente nuestro *HTML* a cualquier dispositivo y pantalla.
- **d3js**. Modificador del *DOM* que utilizamos para generar gráficos interactivos.
- **dataTables**. Presentación de tablas de datos.
- **lokijs**. Base de datos *noSQL* para los cachés de datos y *widgets* del navegador
- **axios**. Cliente *http*.

```
const {require_dyn} = require('../proxy_modules');
const httpContext = require('express-http-context');
const express = require('express');
const router = express.Router();
const path_module = require('path');

router.use(require_dyn("logger_express").info);
```

Respuesta de error interno.

```
router.use(function (err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: {}
  });
});
```

Página de la aplicación

Devuelve la página principal, de nombre canónico *plasmidnet.html*. Se admiten otros alias convencionales para soslayar errores de tecleo del usuario.

Los ficheros se marcan en el header *x-plasmidnet-subversion* con el *md5* de versión que tenemos almacenado en la base de datos. Es utilizado por la *web app* para comprobar si es necesario descargar una nueva versión invalidando la existente en caché.

El fichero descargado utiliza la versión contenida en el contexto *app_version*, que recordemos puede proceder de la *cookie pn_version*, para decidir desde qué directorio suministrar el fichero. Cada versión lleva asociado implícitamente un directorio del mismo nombre donde el sistema de distribución de software deposita los ficheros relacionados con la versión. También en la base de datos se almacenan los códigos de versión (*subversion*) del fichero (*md5* de su contenido), ligados a la versión de software.

Estos convenios internos de nomenclatura los utilizamos extensivamente en PlasmidNet para evitar configuraciones, conforme a *COC* (*Convention over Configuration*).

Todas las versiones del portal se despliegan en el directorio *site*.

```
router.get(/^\/plas\/app\/(|index.html|index.htm|home.html|home.htm|plasmidnet.
    ↪ html|plasmidnet.htm)$/, function(req,res){
  require_dyn("logger_app").info("HTTP " + req.url);
  let cms = require_dyn("gulpfile_cms");
  let version = httpContext.get('app_version');
  let path  = __basedir + '/site/' + version;
  let subversion = cms.get_subversion(undefined, 'plasmidnet.html', version);
  res.set({'x-plasmidnet-subversion' : subversion});
  res.sendFile(path + '/plasmidnet.html');
});
```

Resto de ficheros estáticos

Retorna el resto de ficheros, siguiendo las mismas pautas expresadas más arriba.

```
router.get(/^\/plas\/app\/(|js|css|vendor|fonts|images).*/, function(req,res){
  require_dyn("logger_app").info("HTTP " + req.url);
  let cms = require_dyn("gulpfile_cms");
  let version = httpContext.get('app_version');
  let path  = __basedir + '/site/' + version + req.url.split('/plas/app')[1];
  let cache_name = path_module.basename(req.url);
  let subversion = cms.get_subversion(undefined, cache_name, version);
  res.set({'x-plasmidnet-subversion' : subversion});
  res.sendFile(`${path}`);
});
```

Ping

Servicio para verificar que el servidor está activo.

```
router.get('^/plas/app/ping', function (req, res) {
  res.send("<p>pong</p>");
});

router.use(require_dyn("logger_express").error);

module.exports = router;
```

B.3.3. router_services.js

En este módulo se definen las respuestas ante las solicitudes de datos *REST*. En todos los casos devolvemos la respuesta en formato *json* y la solicitud puede llegar de muchas formas. En el caso de los servicios de datos (bajo método *POST*) en el cuerpo de la solicitud es factible informarlo en formato *json* o *urlencoded*.

```
const {require_dyn} = require('../proxy_modules');
const httpContext = require('express-http-context');
const express = require('express');
const uuidv4 = require('uuid');
const none = function(){};
const router_services = express.Router();
const fs = require('fs');
const path = require('path');
let logger, bio, cms;

router_services.use(require_dyn("logger_express").info);
```

resolve_dyns

En esta función se cargan dinámicamente los módulos necesarios. Recordemos que el intento de recarga se realiza en tiempo de solicitud (*inpassing*).

```
router_services.use(function resolve_dyns(req, res, next) {
  logger = require_dyn("logger_app");
  logger.info("router_services URL: " + req.url);
  cms = require_dyn("gulpfile_cms");
  bio = require_dyn("gulpfile_bio");
  next();
});
```

Servicio test

Servicio de prueba al que indicamos la versión y el número de iteraciones y realizamos un update reiterado de la base de datos *CMS* (tabla *version*).

Lo utilizamos para pruebas de concurrencia en desarrollo.

```
router_services.get('^/plas/service/test/:table/:version/:iter', async function
    ↪ (req, res, next) {
  try {
    let time_ini = (new Date()).getTime();
    let iter;
    for(iter=0; iter<req.params.iter; iter++) {
      cms.upsert_version_testing(undefined, req.params.table, 'test', req.params
    ↪ .version, iter);
    }
    let time_fin = (new Date()).getTime();
    let elapsed = (time_fin - time_ini)/1000;
    res.json({version: req.params.version, iter: req.params.iter, elapsed:
    ↪ elapsed, time_ini: time_ini/1000, time_fin: time_fin/1000});
  } catch(error) {
    errorcode = "TEST0001";
    message = `service.test internal error: ${error.message}`;
    logger.error(message + ' ' + errorcode + ' ' + error.errorcode);
    if (error.stack) logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode + '.' + error.
    ↪ errorcode});
  }
});
```

Sercicio *content*

Servicio de devolución de *widgets* (injertables). La respuesta se compone de los siguientes campos, que son los mismos con los que se almacena en la base de datos local en el navegador:

Cuadro B.2: Campos *json* del servicio de descarga de injertables.

campo	descripción
pn_ref	referencia interna
pn_content	contenido html
pn_version	versión del servidor
pn_subversion	versión (*md5*) del *widget*
pn_reload_pending	indicador de caché inválido

```
router_services.get('^/plas/service/cms/content/:content_name', async function (
    ↪ req, res, next) {
  try {
    let content_name = req.params.content_name;
    let content = cms.get_content(undefined, content_name, httpContext.get('
    ↪ app_version'));
    res.json({pn_ref: content_name, pn_content: content.body, pn_version:
    ↪ content.version, pn_subversion: content.subversion, pn_reload_pending:
    ↪ false});
  } catch(error) {
    errorcode = "CMS0001";
    message = `service.cms.content internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio *datatable*

Servicio de devolución de datos para presentar en local con dataTables. El servicio invoca al módulo de base de datos (en *gulpfile_bio.js*) que es donde realmente se realiza el acceso a la base de datos y la generación del *json*.

Al módulo de acceso a base de datos se le pasa la consulta confeccionada por el usuario en la tarjeta de búsqueda.

En caso de error se devuelve un *status* 500, el mensaje de error y el código de error.

```
router_services.post('^/plas/service/datatable', async function (req, res, next)
    ↪  {
  try {
    res.json(bio.datatable(undefined, req.body.query));
  } catch(error) {
    errorcode = "PLAS0002";
    message = `service.datatable internal error ${error.message}`;
    logger.error(message + ' ' + errorcode + ' ' + error.errorcode);
    if (error.stack) logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode + '.' + error.
    ↪ errorcode});
```

```
  }
});
```

Servicio hierarchy

Servicio de devolución de datos jerárquicos para presentar en local con *d3js*. La implementación es similar a la del servicio anterior.

```
router_services.post('^/plas/service/hierarchy', async function (req, res, next)
    ↪  {
  try {
    res.json(bio.hierarchy(undefined, req.body.query));
  } catch(error) {
    errorcode = "PLAS0003";
    message = `service.hierarchy internal error: ${error.message}`;
    logger.error(message + ' ' + errorcode + ' ' + error.errorcode);
    if (error.stack) logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode + '.' + error.
    ↪ errorcode});
  }
});
```

Servicio force_network

Servicio de devolución de datos en red para presentar en local con *d3js*. La implementación es similar a la del servicio anterior.

```
router_services.post('^/plas/service/force_network', async function (req, res,
    ↪ next) {
  try {
    res.json(bio.force_network(undefined, req.body.query));
  } catch(error) {
    errorcode = "PLAS0020";
    message = `service.force_network internal error: ${error.message}`;
    logger.error(message + ' ' + errorcode + ' ' + error.errorcode);
    if (error.stack) logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode + '.' + error.
    ↪ errorcode});
  }
});
```

Servicio module

Servicio de devolución de módulos *javascript*. En la respuesta, además del contenido del módulo, se informa la cabecera *x-plasmidnet-subversion* con la versión del objeto para comprobaciones de validez del caché dentro del navegador.

```
router_services.get('^/plas/service/module/:module', function (req, res, next) {
  try {
    let version = httpContext.get('app_version');
    let path  = __basedir + '/site/' + version;
    let file = path + '/js/modules/' + req.params.module;
    let subversion = cms.get_subversion(undefined, req.params.module, version);
    res.set({'x-plasmidnet-subversion' : subversion, 'ETag': subversion});
    res.sendFile(file);
  } catch(error) {
    errorcode = "PLAS0100";
    message = `service.module internal error: ${error.message}`;
    logger.error(message + ' ' + errorcode + ' ' + error.errorcode);
    if (error.stack) logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode + '.' + error.
    ↪ errorcode});
  }
});
```

Servicio reload

Verifica si un objeto debe ser recargado desde el navegador al ser obsoleta la versión descargada. Para ello se envía al servicio, en la cabecera *x-plasmidnet-subversion*, la versión actual del objeto en la caché del navegador.

El servicio comprueba la coincidencia de versiones, si no coinciden devuelve en el *json* el indicador *reload* como *true* que indica que existe una nueva versión disponible que debe ser descargada.

```
router_services.get('^/plas/service/cache/reload', async function (req, res,
    ↪ next) {
  try {
    // Debe llegar object, version y subversion, pero la version viene de la
    ↪ cookie?
    let object_id = req.headers['x-plasmidnet-object'];
    // Modules have a mtime ad-hoc parameter to avoid caching on browser and
    ↪ proxies
    let object_raw = object_id.split('?mtime')[0];
    let object = path.basename(object_raw);
    // Delete mtime tail from module url
    let subversion = cms.get_subversion(undefined, object, httpContext.get('
    ↪ app_version'));
    logger.info("router_services.reload object " + object + " " + subversion);
    let reload;
    if (subversion && subversion !== req.headers['x-plasmidnet-subversion'])
    ↪ reload = true;
    else reload = false;
    res.json({reload: reload});
  } catch(error) {
    errorcode = "CACHE0001";
```

```
    message = `router_services.reload internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});

router_services.use(require_dyn("logger_express").error);

router_services.use(function(err, req, res, next) {
  res.status(500);
  res.json({message: "Services: generic Internal Error"});
  next();
});

module.exports = router_services;
```

B.3.4. logger_app.js

En este módulo definimos el *log* de aplicación sobre el API *winston* de *nodejs*.

Creamos dos tipos de objetos *logger*:

1) Error. Orientado a la impresión de la pila de ejecución.
2) Resto.

```
var winston = require('winston');
const ENV = process.env.ENV || 'dev';
const LEVEL = ENV === 'dev' ?  'debug' : 'error';
const winstonLogger = winston.createLogger({
  transports: [
    new winston.transports.Console({level: LEVEL}),
    new winston.transports.File({level: LEVEL, filename: './LOG/web/
    ↪ plasmidnet_app.log'})
  ],
  datePattern: 'YYYY-MM-DD-HH',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.json()
    //winston.format.prettyPrint()
  )
});

const winstonErrorLogger = winston.createLogger({
  transports: [
    new winston.transports.Console({level: 'error'}),
    new winston.transports.File({level: 'error', filename: './LOG/web/
    ↪ plasmidnet_error.log'})
  ],
```

```
  datePattern: 'YYYY-MM-DD-HH',
  format: winston.format.combine(
    winston.format.errors({ stack: true }),
    winston.format.timestamp(),
    winston.format.json()
  )
});

var httpContext = require('express-http-context');
```

formatMessage

En este punto incrustamos en los *logs* el identificador de *request* que tenemos almacenado en el contexto de la llamada *httpContext*.

```
// Wrap Winston logger to print reqId in each log
var formatMessage = function(message) {
    var reqId = httpContext.get('reqId');
    message = reqId ? message + " request id: " + reqId : message;
    return message;
}

var logger = {
    log: function(level, message) {
      winstonLogger.log(level, formatMessage(message));
    },
    debug: function(message) {
      winstonLogger.debug(formatMessage(message));
    },
    info: function(message) {
      winstonLogger.info(formatMessage(message));
    },
    error: function(message) {
      winstonErrorLogger.error(message);
    }
}

module.exports = logger;
```

logger_express.js

Aquí definimos el *log* de errores relacionado con el servidor web *express*.

Como en el caso de la aplicación contamos con dos transportes, el de salida a consola y el de salida a fichero.

Definimos un *log* de errores y un *log* tipo *access* con el fin de registrar todos los accesos al servidor *web*.

El identificador de *request* se graba también en el *log access*. De este modo se puede cruzar perfectamente con el *log* de aplicación.

El fichero de *log* de error es el mismo que el *log* de la aplicación.

```
const express = require('express');
var winston = require('winston'),
    expressWinston = require('express-winston');
const LEVEL = "info";
const ENV = process.env.ENV || 'dev';
```

logger_express.info

Definición del *logger* tipo *access*. El transporte tipo consola es desactivado para el entorno de producción.

```
let transports_access;
if (ENV === "dev") {
  transports_access = [
    new winston.transports.Console({level: LEVEL}),
    new winston.transports.File({level: LEVEL, filename: __basedir + '/LOG/web/
    ↪ plasmidnet_access.log' })
  ];
} else {
  transports_access = [
    new winston.transports.File({level: LEVEL, filename: __basedir + '/LOG/web/
    ↪ plasmidnet_access.log' })
  ];
};
const info = expressWinston.logger({
    transports: transports_access,
    format: winston.format.combine(
      winston.format.timestamp(),
      winston.format.json()
    ),
    meta: false,
    msg: "{{req.plasmidnet_reqid}} HTTP {{req.method}} {{req.url}} {{res.
    ↪ statusCode}} {{res.responseTime}}ms",
    expressFormat: false, // Use the default Express/morgan request formatting.
    ↪ Enabling this will override any msg if true. Will only output colors with
    ↪  colorize set to true
    colorize: false, // Color the text and status code, using the Express/morgan
    ↪  color palette (text: gray, status: default green, 3XX cyan, 4XX yellow,
    ↪ 5XX red).
    ignoreRoute: function (req, res) { return false; } // optional: allows to
    ↪ skip some log messages based on request and/or response
});
```

logger_express.error

Definición del *logger* de errores para *express*. El transporte tipo consola es desactivado para el entorno de producción.

```
let transports_error;
if (ENV === "dev") {
  transports_error = [
    new winston.transports.Console({level: LEVEL}),
    new winston.transports.File({level: LEVEL, filename: __basedir + '/LOG/web/
    ↪ plasmidnet_error.log' })
  ];
} else {
  transports_error = [
    new winston.transports.File({level: LEVEL, filename: __basedir + '/LOG/web/
    ↪ plasmidnet_error.log' })
  ];
};
const error = expressWinston.errorLogger({
  transports: transports_error,
  format: winston.format.combine(
    winston.format.errors({ meta: false }),
    winston.format.timestamp(),
    winston.format.json()
  ),
  msg: "version0 {{req.plasmidnet_reqid}}"
});
if (ENV === "prod") {

}
var logger_express = {info : info, error: error}
module.exports = logger_express;
```

B.3.5. proxy_modules.js

Este módulo intercepta todas las llamadas a todas las funciones de los módulos dinámicos, comprobando en ese momento si existe una nueva versión del módulo desplegada. Si es así, procede a cargarla y devuelve la respuesta con el nuevo módulo.

La versión es comprobada utilizando el *timestamp* del fichero y su valor almacenado en el objeto *timestamps*.

El objeto *modules* almacena en memoria los punteros a las versiones vigentes de los módulos.

```
const fs = require('fs');
const httpContext = require('express-http-context');
const default_server_version = "v0";
const ENV = process.env.ENV || 'dev';
const timestamp = false;
var timestamps = {};
var modules = {};
```

```
var subversions = {};
```

require_dyn_timestamp

Esta función devuelve la versión vigente del módulo. A diferencia del *require* de *nodejs*, comprueba antes si existe en el sistema de archivos una versión más actualizada (utilizando el *timestamp* de modificación), si es así, antes de devolver la respuesta esta versión es cargada vía *require* después de borrar la caché de *require*. Este módulo lo enlazamos en la estructura *modules* y lo devolvemos al programa.

Si el módulo no ha sufrido cambios, se devuelve el puntero al módulo que hemos almacenado en *modules*.

```
var require_dyn_timestamp = function(mod, version) {
  var server_version = version === undefined ? httpContext.get('server_version')
    ↪  : version;
  if (server_version === undefined) {
    server_version = default_server_version;
  }
  var logger_app;
  if (modules[server_version] && modules[server_version]["logger_app"] !==
    ↪ undefined) {
    logger_app = modules[server_version]["logger_app"];
  } else {
    logger_app = console;
  }
  var module_path = './' + server_version + '/' + mod
  var complete_module_path = require.resolve(module_path + '.js');
  var current_timestamp = fs.statSync(complete_module_path).mtimeMs;
  if (timestamps[server_version] === undefined) {
    timestamps[server_version] = {};
    modules[server_version] = {};
  }
  if (timestamps[server_version][mod] === undefined || timestamps[server_version
    ↪ ][mod] !== current_timestamp) {
    ENV === 'dev' && console.log('require_dyn_timestamp @@@@@ Reloading module:
    ↪ ' + mod);
    timestamps[server_version][mod] = current_timestamp;
    delete require.cache[complete_module_path];
    modules[server_version][mod] = require(module_path);
  }
  return modules[server_version][mod];
}
```

require_dyn_subversion

A diferencia de la función anterior, ahora la comprobación de versión se realiza sobre la base de datos *CMS* (tabla *version*), si la versión (*subversion*) en esta *bd* es diferente a la almacenada

en local en el objeto *subversions* se procede a la recarga desde fichero.

Si el módulo no ha sufrido cambios, se devuelve el puntero al módulo que hemos almacenado en *modules*.

Este es el algoritmo que utilizaremos en el futuro, por homogeneidad con la forma de trabajar en la *web app*. Presenta además la ventaja de que si modificamos manualmente la versión en la *bd* podemos forzar una recarga.

El principal problema es que tal vez el acceso de comprobación a la *bd* es más lento que el *stats* sobre fichero, aunque esta *bd* está empotrada (*sqlite*) en nuestro programa y los retardos vayan a ser mínimos.

```
var require_dyn_subversion = function(mod, version) {
  var server_version = version === undefined ? httpContext.get('server_version')
    ↪  : version;
  if (server_version === undefined) {
    server_version = default_server_version;
  }
  var logger_app;
  if (modules[server_version] && modules[server_version]["logger_app"] !==
    ↪ undefined) {
    logger_app = modules[server_version]["logger_app"];
  } else {
    logger_app = console;
  }
  var module_path = './' + server_version + '/' + mod
  var complete_module_path = require.resolve(module_path + '.js');
  let current_subversion;
```

Necesitamos acceder a la *bd CMS* para comprobar la versión, pero en los primeros momentos del arranque puede no estar accesible, tampoco el módulo, en estos casos marcamos la versión como *0*, que será diferente de cualquier versión real.

```
  if (modules[server_version] && modules[server_version]["gulpfile_cms"] !==
    ↪ undefined) {
    try {
      current_subversion = modules[server_version]["gulpfile_cms"].
    ↪ get_subversion(undefined, mod + ".js", server_version);
    } catch (error) {
      current_subversion = 0;
    }
  } else {
    current_subversion = 0;
  }
  if (subversions[server_version] === undefined) {
    subversions[server_version] = {};
    modules[server_version] = {};
  }
  if (subversions[server_version][mod] === undefined || subversions[
    ↪ server_version][mod] !== current_subversion) {
    ENV === 'dev' && console.log('require_dyn_subversion @@@@@Reloading module:
    ↪ ' + mod);
```

```
    subversions[server_version][mod] = current_subversion;
    delete require.cache[complete_module_path];
    modules[server_version][mod] = require(module_path);
  }
  return modules[server_version][mod];
}
```

Por conveniencia, exportamos varios alias para la función de recarga dinámica.

```
var proxy_modules = {call: require_dyn_timestamp, require_dyn:
    ↪ require_dyn_timestamp, dyn: require_dyn_timestamp}

module.exports = proxy_modules;
```

B.4. Bases de Datos

B.4.1. gulpfile_bio.js

Este es el módulo que amalgama todas las consultas y operaciones sobre la base de datos de módulos funcionales de plásmidos. Las consultas están implementadas en *SQL* contra una base de datos *SQlite*. En la interfaz de usuario las consultas se escriben con la sintaxis de un lenguaje reducido (*DSL: Domain Specific Language*) muy ajustado a las casuística que nos ocupa.

Estas consultas son transformadas a otro lenguaje específico de dominio: *SQL*. Durante la transformación se valida la sintaxis de la consulta, de forma que nos protegemos contra posibles intentos de inyección *SQL*.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const sqlite = require('better-sqlite3');
const axios = require('axios');
const fs = require('fs');
const cheerio = require('cheerio');
const LIMIT = 1000; // Max number of table registers to return
const argv = require('yargs')
  .default('plasmid_id', 'NZ_LT960791.1')
  .default('query', 'p = NZ_LT960791.1 | p = AZ')
  .default('query_force_network', 'score > 0.8')
  .argv;
```

DSL: lista de operadores. Están todos duplicados. Las versiones literales entre . están pensadas para facilitar el tecleo sobre dispositivos móviles.

Se denominan expandibles porque admiten uno o varios argumentos.

```
const operators_expandable = [
  '==', '.e.', '=', '.i.', '*=', '.ew.', '=*', '.sw.',
  '!==', '.ne.', '!=', '.ni.', '!*=', '.new.', '!=*', '.nsw.',
```

```
  '>=', '.ge.', '<=', '.le.', '<', '.lt.', '>', '.gt.'
];

const ternary_operators = [
  '(', ')'
];

const operators_pair = {
  '(': ')'
};

const binary_operators = [
  '|', '.o.', '&', '.a.'
];

const unary_operators = [
  '^', '.n.'
];

const alias = {
  '.ge.': '>=',
  '.le.': '<=',
  '.lt.': '<',
  '.gt.': '>',
  '.a.': '&',
  '.o.': '|',
  '.n.': 'NOT',
  '^': 'NOT'
}
```

Los operandos se relacionan con campos específicos de base de datos. Los prefijos de los campos son los alias que debemos utilizar en la *SQL* resultante de la traducción al dominio *SQL*. Para facilitar el tecleo definimos varios alias que se refieren al mismo campo.tabla de la BD.

```
const operands = {
  plasmid: 'p.plasmid_id',
  pl: 'p.plasmid_id',
  p: 'p.plasmid_id',
  m: 'm.module_id',
  mod: 'm.module_id',
  module: 'm.module_id',
  s: 's.superfam_id',
  sf: 's.superfam_id',
  superfam: 's.superfam_id',
  f: 'f.id',
  fam: 'f.id',
  pr: 'f.protein_id',
  prot: 'f.protein_id',
  protein: 'f.protein_id',
  sc: 'r.score',
```

```
  score: 'r.score'
};

global.__sqlite_bio_db;
```

start_bio

Arranque de la base de datos de información biológica. Es ejecutada en el arranque de la aplicación.

```
function start_bio(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_bio_db = new sqlite('./DATA/plasmid_modules/db/plasmid_modules.db', {
    ↪  verbose: logger.debug });
  if (cb) {cb()};
}
```

close_bio

Cierre de la base de datos de información biológica.

```
function close_bio(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_bio_db.close();
  if (cb) {cb()};
}
```

datatable

Esta función devuelve los datos requeridos en la consulta *query*. Lo primero que hace es convertir la consulta en su equivalente *SQL*.

El equivalente es en realidad una parte de la consulta que se ubica dentro de la cáusula *WHERE* de una consulta ya formada. Contamos con dos estructuras de consultas sobre la jerarquía plásmido, superfamilia y módulos:

1) Con campo superfamilia.
2) Sin campo superfamilia.

Entendemos que el usuario quiere realizar una consulta sobre superfamilias si ha incluido una condición de búsqueda para este campo. El número de superfamilias es muy elevado: no podemos en principio admitir una consulta que las requiera y no las especifique.

El nombre de la función corresponde con el nombre del servicio y con la función en la que se llama al servicio en la aplicación cliente (*COC*).

```
function datatable(cb, query) {
  try {
    let logger = require_dyn("logger_app");
    logger.debug('datatable');
    let parsed_query = parse_dsl_query(undefined, query);
    if (parsed_query.errorcode != "") {
      return parsed_query;
    }
    if (parsed_query.query === '') throw {message: 'Empty query', errorcode: '
    ↪ PLAS0009'};
    let sql_superfam = `
      SELECT p.plasmid_id as plasmid_id,
        "M-" || p.module_id as module_id,
        "S-" || s.superfam_id as superfam_id
      FROM plasmid_module p, module_superfam s
      WHERE p.module_id = s.module_id AND
      (${parsed_query.query})
      ORDER BY p.plasmid_id ASC, p.module_id ASC, s.superfam_id ASC
      LIMIT 0, ${LIMIT};`;

    let sql_default = `SELECT p.plasmid_id, "M-" || module_id AS module_id
      FROM plasmid_module p
      WHERE ${parsed_query.query}
      ORDER BY p.plasmid_id ASC, p.module_id ASC
      LIMIT 0, ${LIMIT}`;

    let sql = sql_default;
    if (parsed_query.query.includes('superfam_id')) sql = sql_superfam;

    let stmt = __sqlite_bio_db.prepare(sql);
    let modules = {data: []};
    let nrecords = 0;
    for (let record of stmt.iterate()) {
      modules.data.push(record);
      nrecords++;
    }
    modules.recordsTotal = nrecords;
    modules.recordsFiltered = nrecords;
    if (cb) {
      console.log(modules);
      cb();
    } else {
      return modules;
    }
  } catch(error) {
    if (cb) {console.log(error);cb();} else {return error;}
  }
}
```

force_network

Devuelve los datos requeridos en la consulta (parámetro *query*) en un formato *json* adecuado para presentarse en un gráfico *force_ network*.

En este caso el *json* lo construimos en *SQL* directamente sin utilizar *javascript*, porque es más rápido de construir y de probar.

```
function force_network(cb, query=argv.query_force_network) {
  try {
    let logger = require_dyn("logger_app");
    logger.debug('force_network');
    let parsed_query = parse_dsl_query(undefined, query);
    if (parsed_query.errorcode != "") {
      return parsed_query;
    }
    logger.info('gulpfile_bio.force_network transformed query' + parsed_query);
    if (parsed_query.query === '') throw {message: 'Empty query', errorcode: '
   ↪ PLAS0012'};
    let sql_default = `
      SELECT '{"nodes": ['  AS json
      UNION ALL
      SELECT DISTINCT '{"id":"S-' || superfam_id_1 || '", "group":"M-' || s.
   ↪ module_id || '"},'
        FROM superfam_rel r, module_superfam s
        WHERE r.superfam_id_1 = s.superfam_id
        AND ${parsed_query.query}
      UNION ALL
      SELECT DISTINCT '{"id":"S-' || superfam_id_2 || '", "group":"M-' || s.
   ↪ module_id || '"},'
        FROM superfam_rel r, module_superfam s
        WHERE r.superfam_id_2 = s.superfam_id
        AND ${parsed_query.query}
      UNION ALL
      SELECT '],"links": ['
      UNION ALL
      SELECT '{"source":"S-' || r.superfam_id_1 || '", "target":"S-' || r.
   ↪ superfam_id_2 ||  '", "value": ' ||  round(r.score*10) || '},'
        FROM superfam_rel r, module_superfam s, module_superfam s2
        WHERE r.superfam_id_1 = s.superfam_id AND r.superfam_id_2 = s2.
   ↪ superfam_id
        AND ${parsed_query.query}
      UNION ALL
      SELECT ']}'
    `;
    let sql = sql_default;
    logger.info('gulpfile_bio.force_network final query ' + parsed_query);
    let stmt = __sqlite_bio_db.prepare(sql);
    let nrecords = 0;
    let json_concat = "";
    for (let json of stmt.pluck().iterate()) {
      json_concat += json;
      nrecords++;
```

```
    }
    json_concat = json_concat.replace(/,\]/g, ']').replace(/,\}/g, '}');
    let modules = {data: JSON.parse(json_concat)};
    modules.recordsTotal = nrecords;
    modules.recordsFiltered = nrecords;
    if (cb) {
      console.log(modules);
      cb();
    } else {
      return modules;
    }
  } catch(error) {
    if (cb) {console.log(error);cb();} else {logger.error('gulpfile_bio.
    ↪ force_network ERROR');logger.error(error); return {message: 'Internal
    ↪ Error', errorcode: "PLAS00030"};}
  }
}
```

hierarchy

Devuelve los datos requeridos en la consulta (parámetro *query*) en un formato *json* adecuado para presentarse en un gráfico jerárquico.

Del mismo modo que en *force_ network* el json de salida lo construimos en *SQL* directamente sin utilizar *javascript*, porque es más rápido de construir y de probar.

```
function hierarchy(cb, query) {
  try {
    let logger = require_dyn("logger_app");
    let parsed_query = parse_dsl_query(undefined, query);
    if (parsed_query.errorcode != "") {
      return parsed_query;
    }
    logger.info('gulpfile_bio.hierarchy transformed query' + parsed_query);
    if (parsed_query.query === '') throw {message: 'Empty query', errorcode: '
    ↪ PLAS0008'};
    let sql_default = `
      SELECT  DISTINCT "" AS parent,  "Plasmids" AS child, "" AS info
      UNION
      SELECT  DISTINCT "Plasmids" AS parent,  p.plasmid_id AS child, "" AS info
        FROM plasmid_module p WHERE
        (${parsed_query.query})
      UNION
      SELECT DISTINCT p.plasmid_id AS parent, "M-" || p.module_id AS child, ""
    ↪ AS info
        FROM plasmid_module p WHERE
        (${parsed_query.query})
      UNION
      SELECT DISTINCT "M-" || p.module_id AS parent, "S-" || s.superfam_id AS
    ↪ child,  "" AS info
```

```
        FROM module_superfam s,  plasmid_module p
        WHERE p.module_id = s.module_id
        AND (${parsed_query.query})
      UNION
      SELECT DISTINCT "S-" || s.superfam_id AS parent, "F-" || f.id AS child, pr
    ↪ . protein_id AS info
        FROM module_superfam s,  plasmid_module p, protein_superfam f ,
    ↪ protein_rep pr, protein_norep pnr
        WHERE p.module_id = s.module_id AND s.superfam_id = f.superfam_id
        AND f.id = pr.id AND pnr.id_expanded = pr.id_expanded AND pnr.plasmid_id
    ↪  = p.plasmid_id
        AND (${parsed_query.query})
      LIMIT 0, 500
    `;
    let sql = sql_default;
    let stmt = __sqlite_bio_db.prepare(sql);
    let modules = {data: []};
    let nrecords = 0;
    for (let record of stmt.iterate()) {
      modules.data.push(record);
      nrecords++;
    }
    modules.recordsTotal = nrecords;
    modules.recordsFiltered = nrecords;
    if (cb) {
      console.log(modules);
      cb();
    } else {
      return modules;
    }
  } catch(error) {
    if (cb) {console.log(error);cb();} else {logger.error('gulpfile_bio.
    ↪ hierarchy ERROR');logger.error(error); return {message: 'Internal Error',
    ↪  errorcode: "PLAS00031"};}
  }
}
```

parse_operator

Traducción a *SQL* de una expresión formada por el operando *left_parsed*, el operador *operator* y la lista de valores *right_parsed*.

```
function parse_operator(left_parsed, operator, right_parsed, ref_pos) {
  // Process expandable operators
  if (left_parsed === "" || right_parsed === "") {
    let total_lenght = left_parsed.length + operator.length + right_parsed.
    ↪ length;
    throw {message: 'Bad expression', position: ref_pos, length: total_lenght,
    ↪ errorcode: 'PARSE011'};
```

```
  }
  let parsed = "";
  let right_parsed_parts = right_parsed.split(',');
  // Parse multi right operand operators (expandable operators)
  for (operand of right_parsed_parts) {
    if (parsed != "") {
      if (operator.includes('!') || operator.includes('.n')) parsed += ' AND ';
      else parsed += ' OR ';
    }
    // escaping '_' for LIKE sentences
    let operand_like = operand.replace(/_/g, "\_");
    if (operator === "=" || operator === ".i.") {
      parsed += `${operands[left_parsed]} LIKE '%${operand_like}%'`;
    } else if (operator === "!=" || operator === ".ni.") {
        parsed += `${operands[left_parsed]} NOT LIKE '%${operand_like}%'`;
    } else if (operator === "!==" || operator === ".ne.") {
        parsed += `${operands[left_parsed]} != '%${operand}%'`;
    } else if (operator === "==" || operator === ".e.") {
      parsed += `${operands[left_parsed]} = '${operand}'`;
    } else if (operator === "=*" || operator === ".sw.") {
      parsed += `${operands[left_parsed]} LIKE '${operand_like}%'`;
    } else if (operator === "!=*" || operator === ".nsw.") {
      parsed += `${operands[left_parsed]} NOT LIKE '${operand_like}%'`;
    } else if (operator === "*=" || operator === ".ew.") {
      parsed += `${operands[left_parsed]} LIKE '%${operand_like}'`;
    } else if (operator === "!*=" || operator === ".new.") {
      parsed += `${operands[left_parsed]} NOT LIKE '%${operand_like}'`;
    } else if (['>=', '<=', '<', '>', '&', '|'].includes(operator) &&
   ↪ right_parsed_parts.length === 1) {
      parsed = `${operands[left_parsed]} ${operator} '${operand}'`;
    } else if (['.ge.', '.le.', '.lt.', '.gt.', '.a.', '.o.'].includes(operator)
   ↪  && right_parsed_parts.length === 1) {
      parsed = `${operands[left_parsed]} ${alias[operator]} '${operand}'`;
    } else { //Operator not found
      throw {message: 'Operator not found', position: ref_pos, length: operator.
   ↪ length, errorcode: 'PARSE006'};
    }
  }
  return "(" + parsed + ")";
}
```

process_search_item

Verifica el formato de la lista de valores de la parte derecha de un operador. Se admite utilizar {} para agrupar la lista, pero no es obligatorio. Puede ser interesante si facilita la legibilidad.

```
function process_search_item(query, position, ref_pos) {
  if (query.trim() === "") return "";
  if (position === 'left') {
```

```
    for (let operand in operands) {
      if (operand === query.trim()) return operand;
    }
    throw {message: 'Unknown field ' + query, position: ref_pos, length: 1,
   ↪ errorcode: 'PARSE010'};
  } else if (position === 'right') {
    // Verify group parenthesis coherence
    if (query[0] === '{' && query[query.length-1] != '}') {
      throw {message: 'Unpaired opening bracket {', position: ref_pos, length:
   ↪ 1, errorcode: 'PARSE001'};
    } else if (query[0] != '(' && query[query.length-1] === ')') {
      throw {message: 'Unpaired closing bracket {', position: ref_pos + query.
   ↪ length-1, length: 1, errorcode: 'PARSE002'};
    }
    // Replace possibly group parentesis
    let query_new = query.replace('{', '').replace('}', '');
    let query_parts = query_new.split(',');
    let query_parts_parsed = [];
    for (let part of query_parts) {
      part = part.trim();
      if (/^[a-z0-9_.]+$/i.test(part))
        query_parts_parsed.push(part);
      else {
        let part_pos = ref_pos + query.indexOf(part);
        throw {message: 'Fields with invalid characters', position: part_pos,
   ↪ length: part.length, errorcode: 'PARSE003'};
      }
    }
    let right_parsed = query_parts_parsed.join(',');
    return right_parsed;
  }
}
```

search_close_bracket

Búsqueda del paréntesis de cierre de un paréntesis de apertura.

```
function search_close_bracket(query, operator, pos) {
  let operator_pair = operators_pair[operator];
  let counter = 1;
  let pos_end = -1;
  for (let i = pos + 1; i < query.length; i++) {
    if (query[i] === operator) counter++;
    else if (query[i] === operator_pair) counter--;
    if (counter === 0) {
      // We have arrived to the closed bracket
      pos_end = i;
      break;
    }
```

```
  }
  return pos_end;
}
```

parse_dsl

Función principal en la conversión de la consulta a formato *SQL*. Se ha implementado recursivamente: es llamada desde cada nivel de paréntesis. En la lista de operadores éstos se informan por orden de precedencia. La posición de referencia *ref_pos* se utiliza en todas estas funciones para informar al usuario del lugar del error.

```
function parse_dsl(query, operators, position='left', ref_pos) {
  if (query === "") return query;
  query += "";
  let found = false;
  for (let operator_idx in operators) {
    let operator = operators[operator_idx];
    let pos = query.indexOf(operator);
    if (pos >= 0) {
      found = true;
      // The previous indexes were not found in query, we discard them
      operators = operators.slice(operator_idx);
      let query_parts = query.split(operator);
      let left = query_parts[0];
      let right = query_parts[1];
      let parsed;
      if (ternary_operators.includes(operator)) {
        if (operator == ')') throw {message: 'Unpaired closing bracket',
    ↪ position: ref_pos + pos, length: operator.length, errorcode: 'PARSE007'};
        pos_end = search_close_bracket(query, operator, pos);
        if (pos_end === -1) throw {message: 'Unpaired opening bracket', position
    ↪ : ref_pos + pos, length: operator.length, errorcode: 'PARSE008'};
        let inter = query.slice(pos + 1, pos_end);
        right = query.slice(pos_end + 1);
        let left_parsed = parse_dsl(left, operators, '', ref_pos);
        let inter_parsed = parse_dsl(inter, operators, '', ref_pos + pos +
    ↪ operator.length);
        let right_parsed = parse_dsl(right, operators, '', ref_pos + pos_end +
    ↪ operator.length);
        parsed = left_parsed + operator + inter_parsed + operators_pair[operator
    ↪ ] + right_parsed;
      } else if (unary_operators.includes(operator)) {
        let right_parsed = parse_dsl(right, operators, 'right', ref_pos + pos +
    ↪ operator.length);
        if (operator === "^" | operator === '.n.') {
          parsed = `${alias[operator]} ${right_parsed}`;
        }
      } else if (binary_operators.includes(operator)) {
        let left_parsed = parse_dsl(left, operators, 'left', ref_pos);
```

```
        let right_parsed = parse_dsl(right, operators, 'right', ref_pos + pos +
    ↪ operator.length);
        // Pure binary operators
        if (operator === "&" || operator === '.a.') {
          return `${left_parsed} AND ${right_parsed}`;
        } else if (operator === "|" || operator === '.o.') {
          return `${left_parsed} OR ${right_parsed}`;
        }
      } else {
        // Operators with database field left operand
        let left_parsed = parse_dsl(left, operators, 'left', ref_pos);
        let right_parsed = parse_dsl(right, operators, 'right', ref_pos + pos +
    ↪ operator.length);
        parsed = parse_operator(left_parsed, operator, right_parsed, ref_pos);
      }
      return parsed;
    }
  }
  // Not found operator, all the string is a left operand or a right operand
    ↪ group: (op,op,..)
  if (!found) {
    return process_search_item(query, position, ref_pos);
  }
}
```

parse_dsl_query

Esta es la función de alto nivel para lanzar la traducción desde la línea de comandos. Sirve para probar el algoritmo unitariamente.

```
function parse_dsl_query(cb, query=argv.query) {
  try {
    let operators_sorted = operators_expandable.sort(function(a, b){
      return b.length - a.length;
    });
    let operators_by_reverse_precedence = ternary_operators.concat(
    ↪ binary_operators).concat(unary_operators).concat(operators_sorted);
    let parsed_query = parse_dsl(query, operators_by_reverse_precedence, 'left',
    ↪  0);
    let ret = {query: parsed_query, errorcode: ''};
    if (cb) {console.log(ret.query);cb();} else {return ret;}
  } catch(error) {
    if (cb) {console.log(error);cb();} else {return error;}
  }
}

module.exports.start_bio = start_bio;
module.exports.close_bio = close_bio;
module.exports.datatable = datatable;
```

```
module.exports.hierarchy = hierarchy;
module.exports.force_network = force_network;
module.exports.parse_dsl_query = parse_dsl_query;
module.exports.do_hierarchy = series(start_bio, hierarchy, close_bio);
module.exports.do_datatable = series(start_bio, datatable, close_bio);
module.exports.do_force_network = series(start_bio, force_network, close_bio);
module.exports.shares = module.exports;
```

B.4.2. load_plasmidnet_db.js

Carga de la base de datos de PlasmidNet a partir de los ficheros obtenidos en el flujo orquestado de proceso inicial.

```
const logger = require('../../modules/logger.js');
const sqlite = require('better-sqlite3');
const fs = require('fs');
const path = require('path');
const readline = require('readline');
const DB_NAME = './DATA/plasmid_modules/db/plasmid_modules.db';
const MAX_LINES = 1000;
// const none = function(){};
const argv = require('yargs')
  .default('max_lines', MAX_LINES)
  .default('step', 'test')
  .default('erase', 'N')
  .argv;

let plasmid_modules_db;
```

create_read_interface

Crea el *stream* de lectura de un fichero de entrada. Necesario para procesar línea por línea los ficheros de entrada, de forma similar a cómo se realiza en otros lenguajes como *python* o *perl*.

```
function create_read_interface(input, method) {
  let readInterface = readline.createInterface({
    input: fs.createReadStream(input),
    crlfDelay: Infinity
  });

  readInterface.input.on('error', function(error) {
    logger.error(method + ' .readInterface.input error');
    logger.error(error);
  });

  return readInterface;
}
```

create_model

Sentencias de creación del modelo de datos.

```
function create_model(erase=argv.erase) {
  logger.info("create_model Init");
  fs.mkdirSync(path.dirname(DB_NAME), {recursive:true});
  if (erase === 'Y' && fs.existsSync(DB_NAME)) fs.unlinkSync(DB_NAME);
  plasmid_modules_db = new sqlite(DB_NAME, { verbose: logger.debug });
  let sql_create_model =
    `CREATE TABLE IF NOT EXISTS protein_rep(
      id INTEGER PRIMARY KEY,
      id_expanded TEXT,
      location TEXT,
      protein_id TEXT,
      plasmid_id TEXT);

      CREATE UNIQUE INDEX IF NOT EXISTS id
        ON protein_rep(id);

      CREATE INDEX IF NOT EXISTS plasmid_id
        ON protein_rep(plasmid_id);

      CREATE TABLE IF NOT EXISTS protein(
        protein_id TEXT PRIMARY KEY,
        function TEXT,
        sequence_AA TEXT);

      CREATE UNIQUE INDEX IF NOT EXISTS protein_id
        ON protein(protein_id);

      CREATE TABLE IF NOT EXISTS plasmids(
        plasmid_id TEXT,
        plasmid_desc TEXT,
        sequence_NU TEXT);

      CREATE UNIQUE INDEX IF NOT EXISTS plasmid_id
        ON plasmids(plasmid_id);

      CREATE TABLE IF NOT EXISTS protein_norep(
        id_expanded_norep TEXT,
        id_expanded TEXT,
        plasmid_id TEXT);

      CREATE UNIQUE INDEX IF NOT EXISTS id_expanded_norep
        ON protein_norep(id_expanded_norep);

      CREATE INDEX IF NOT EXISTS plasmid_id_2
```

```
        ON protein_norep(plasmid_id);

      CREATE TABLE IF NOT EXISTS protein_superfam(
        id INTEGER,
        superfam_id INTEGER);

      CREATE UNIQUE INDEX IF NOT EXISTS id_2
        ON protein_superfam(id);

      CREATE TABLE IF NOT EXISTS module_superfam(
        module_id INTEGER,
        superfam_id INTEGER);

      CREATE INDEX IF NOT EXISTS module_id
        ON module_superfam(module_id);

      CREATE UNIQUE INDEX IF NOT EXISTS superfam_id
        ON module_superfam(superfam_id);

      CREATE TABLE IF NOT EXISTS plasmid_superfam(
        plasmid_id TEXT,
        superfam_id INTEGER);

      CREATE INDEX IF NOT EXISTS plasmid_id_3
        ON plasmid_superfam(plasmid_id);

      CREATE INDEX IF NOT EXISTS superfam_id_2
        ON plasmid_superfam(superfam_id);

      CREATE TABLE IF NOT EXISTS plasmid_module(
        plasmid_id TEXT,
        module_id INTEGER);

      CREATE INDEX IF NOT EXISTS plasmid_id_4
        ON plasmid_module(plasmid_id);

      CREATE INDEX IF NOT EXISTS module_id_2
        ON plasmid_module(module_id);
      `
  let ret = plasmid_modules_db.exec(sql_create_model);
  logger.info("create_model Finish");
  //logger.debug(ret);
}
```

load_representatives

Carga la tabla *protein_rep*, proteínas base de las familias.

```
function load_representatives(max_lines=argv.max_lines) {
```

```
logger.info("load_representatives Init");
const readline = require('readline');
const INPUT = './DATA/plasmid_modules/load_init/fam.pr_code.plasmid_protein.
  ↪ idx';
//const DB_OPTIONS = { verbose: logger.debug };
const DB_OPTIONS =  {}
plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
let nproteins = 0;
let insert_protein_rep = plasmid_modules_db.prepare("INSERT OR REPLACE INTO
  ↪ protein_rep VALUES(@id, @id_expanded, @location, @protein_id, @plasmid_id
  ↪ )");
try {
  // Line format example:
  //    pr_347  >lcl|CP000620.1_prot_ABO60461.1_34 [
  //   locus_tag=Bcep1808_7586] [db_xref=InterPro:IPR003346]
  //   [protein=transposase IS116/IS110/IS902 family protein]
  //   [protein_id=ABO60461.1] [location=complement(36627..37964)]
  //   [gbkey=CDS]`
  const readInterface = readline.createInterface({
    input: fs.createReadStream(INPUT),
    crlfDelay: Infinity
  });

  readInterface.on('line', function(line) {
    // Normalize the fields
    if (nproteins >= max_lines) {
      readInterface.close();
      return;
    } else if (nproteins%10000 === 0) {
      logger.info(`load_representatives.readInterface.on ${nproteins} proteins
  ↪  loaded`);
    }
    let line_split = line.replace('>lcl|', '[id_expanded=').replace('>pr_', '
  ↪ id=').replace(/\]/g, ' ').split('[');
    let fields = {};
    for (field of line_split) {
      let field_trimmed = field.trim();
      matches =  field_trimmed.match(/^(.*)=(.*)/);
      if (matches && matches[2]) {
        fields[matches[1]] = matches[2];
      }
    }
    let ids = fields['id_expanded'].split('_prot_');
    fields['plasmid_id'] = ids[0];
    // There are registers like this:
    // >pr_386415 >lcl|CP031372.1_prot_PR80 hypothetical protein [Klebsiella
  ↪ pneumoniae]
    // We need to take as protein_id PR80 and as expanded CP031372.1_prot_PR80
    let split_protein = ids[1].split(/\s+/);
    if (split_protein[1]) {
      fields['id_expanded'] = fields['plasmid_id'] + '_prot_' + split_protein
```

```
    ↪ [0];
        fields['protein_id'] = split_protein[0];
      }
      // Compute protein code
      if (!('protein_id' in fields)) {
        fields['protein_id'] = 'prot_' + ids[1];
      }

      // Compute location
      if (!('location' in fields)) {
        fields['location'] = '0';
      }

      //console.log(fields);
      insert_protein_rep.run(fields);
      nproteins++;
    })
    .on('close', function() {
      logger.info(`load_representatives.readInterface.close: ${nproteins}
    ↪ proteins loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_representatives Finish");
    })
    .on('error', function(error) {
      logger.error('load_representatives.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_representatives');
    logger.error(error);
  }
}
```

load_protein_AA

Carga la tabla *protein* de secuencias aminoacídicas de proteínas.

```
function load_protein_AA(max_lines=argv.max_lines) {
  logger.info("load_protein_AA Init");
  const readline = require('readline');
  const PR_IDX = './DATA/plasmid_modules/load_init/fam.pr_code.plasmid_protein.
    ↪ fasta';
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nproteins = 0;
  let sequence_AA, fields;
  let insert_protein = plasmid_modules_db.prepare("INSERT OR REPLACE INTO
    ↪ protein VALUES(@protein_id, @function, @sequence_AA)");
  try {
```

```
// Line format example:
//>lcl|NZ_CP025014.1_prot_WP_105009573.1_473
// [locus_tag=CUJ84_RS31885]
// [protein=hypothetical protein]
// [protein_id=WP_105009573.1]
//[location=complement(495027..495632)]
// [gbkey=CDS]
// MKIVNLSQREEDWLDWRRQGVTA... (over several lines)
const readInterface = readline.createInterface({
  input: fs.createReadStream(PR_IDX),
  crlfDelay: Infinity
});

readInterface.on('line', function(line) {
  if (line[0] === '>') {// header line
    if (nproteins !== 0) {
      fields['sequence_AA'] = sequence_AA;
      insert_protein.run(fields);
    }
    let line_split = line.replace('>lcl|', '[id_expanded=').replace(/\]/g, '
↪  ').split('[');
    fields = {};
    for (field of line_split) {
      let field_trimmed = field.trim();
      matches =  field_trimmed.match(/^(.*)=(.*)/);
      if (matches && matches[2]) {
        fields[matches[1]] = matches[2];
      }
    }
    let ids = fields['id_expanded'].split('_prot_');
    let split_protein = ids[1].split(/\s+/);
    if (split_protein[1]) {
      fields['protein_id'] = split_protein[0];
      fields['function'] = split_protein[1];
    }
    //compute protein code
    if (!('protein_id' in fields)) {
      fields['protein_id'] = 'prot_' + ids[1];
    }
    if (!('function' in fields)) {
      fields['function'] = fields['protein'];
    }
    sequence_AA = "";
    nproteins++;
    if (nproteins >= max_lines) {
      readInterface.close();
      return;
    } else if (nproteins%10000 === 0) {
      logger.info(`load_protein_AA.readInterface.on ${nproteins} protein
↪ loaded`);
    }
```

```
      } else {
        //console.log(fields);
        sequence_AA += line;
      }
    })
    .on('close', function() {
      // insert last protein
      fields['sequence_AA'] = sequence_AA;
      insert_protein.run(fields);
      logger.info(`load_protein_AA.readInterface.close: ${nproteins} proteins
    ↪ loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_protein_AA Finish");
    })
    .on('error', function(error) {
      logger.error('load_protein_AA.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_protein_AA');
    logger.error(error);
  }
}
```

load_protein_norep

Carga la tabla *protein_norep* que relaciona todas las proteínas de los plásmidos con sus proteínas representativas.

```
function load_protein_norep(max_lines=argv.max_lines) {
  logger.info("load_protein_norep Init");
  const readline = require('readline');
  const INPUT = './DATA/plasmid_modules/load_init/plasmid_protein_rep.
    ↪ plasmid_protein.idx';
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nproteins = 0;
  let fields;
  let insert_protein_norep = plasmid_modules_db.prepare("INSERT OR REPLACE INTO
    ↪ protein_norep VALUES(@protein_expanded_norep, @protein_expanded,
    ↪ @plasmid_id)");
  try {
    // Line format example:
    //NZ_CP023000.1_prot_WP_095437876.1_182   NZ_CP013567.1_prot_WP_064824145.1
    ↪ _888
    const readInterface = readline.createInterface({
      input: fs.createReadStream(INPUT),
      crlfDelay: Infinity
    });
```

```
    readInterface.on('line', function(line) {
      // Normalize the fields
      if (nproteins >= max_lines) {
        readInterface.close();
        return;
      } else if (nproteins%50000 === 0) {
        logger.info(`load_protein_norep.readInterface.on ${nproteins} proteins
    ↪ loaded`);
      }
      let line_split = line.split(/\s+/);
      fields = {};
      fields['protein_expanded'] = line_split[0];
      fields['protein_expanded_norep'] = line_split[1];
      let ids = fields['protein_expanded_norep'].split('_prot_');
      fields['plasmid_id'] = ids[0];
      insert_protein_norep.run(fields);
      nproteins++;
    })
    .on('close', function() {
      logger.info(`load_protein_norep.readInterface.close: ${nproteins} proteins
    ↪  loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_protein_norep Finish");
    })
    .on('error', function(error) {
      logger.error('load_protein_norep.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_protein_norep');
    logger.error(error);
  }
}
```

load_superfam

Carga la tabla *protein_superfam* que relaciona todas las proteínas representativas con sus superfamilias.

```
function load_superfam(max_lines=argv.max_lines) {
  logger.info("load_superfam Init");
  const readline = require('readline');
  const INPUT = './DATA/plasmid_modules/load_init/superfam.pr_code.idx';
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nsuperfams = 0;
  let fields;
  let insert_protein_superfam = plasmid_modules_db.prepare("INSERT OR REPLACE
    ↪ INTO protein_superfam VALUES(@id, @superfam_id)");
```

```
  try {
    // Line format example:
    //pr_1007 pr_100891 pr_101177 ...
    const readInterface = readline.createInterface({
      input: fs.createReadStream(INPUT),
      crlfDelay: Infinity
    });

    readInterface.on('line', function(line) {
      // Normalize the fields
      if (nsuperfams >= max_lines) {
        readInterface.close();
        return;
      } else if (nsuperfams%10000 === 0) {
        logger.info(`load_superfam.readInterface.on ${nsuperfams} proteins
    ↪ loaded`);
      }
      fields = {};
      nsuperfams++;
      fields['superfam_id'] = nsuperfams;
      let line_split = line.replace(/pr_/g, '').split(/\s+/);
      for (let id of line_split) {
        fields['id'] = id;
        insert_protein_superfam.run(fields);
      }
    })
    .on('close', function() {
      logger.info(`load_superfam.readInterface.close: ${nsuperfams} superfams
    ↪ loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_superfam Finish");
    })
    .on('error', function(error) {
      logger.error('load_superfam.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_superfam');
    logger.error(error);
  }
}
```

load_module_superfam

Carga la tabla *module_superfam* que relaciona todas las superfamilias con sus módulos.

```
function load_module_superfam(max_lines=argv.max_lines) {
  logger.info("load_module_superfam Init");
  const readline = require('readline');
  const INPUT = './DATA/plasmid_modules/load_init/module_superfam.idx';
```

```
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nmodules = 0;
  let fields;
  let insert_module_superfam = plasmid_modules_db.prepare("INSERT OR REPLACE
    ↪ INTO module_superfam VALUES(@module_id, @superfam_id)");
  try {
    // Line format example:
    //15      1014 1043 113 1170 1242 136 146 ...
    const readInterface = readline.createInterface({
      input: fs.createReadStream(INPUT),
      crlfDelay: Infinity
    });

    readInterface.on('line', function(line) {
      // Normalize the fields
      if (nmodules >= max_lines) {
        readInterface.close();
        return;
      } else if (nmodules%100 === 0) {
        logger.info(`load_module_superfam.readInterface.on ${nmodules} modules
    ↪ loaded`);
      }
      fields = {};
      nmodules++;
      let line_split = line.split(/\s+/);
      fields['module_id'] = line_split[0];
      //console.log(line_split);
      for (let id in line_split) {
        if (id >= 1) {
          fields['superfam_id'] = line_split[id];
          insert_module_superfam.run(fields);
        }
      }
    })
    .on('close', function() {
      logger.info(`load_module_superfam.readInterface.close: ${nmodules} modules
    ↪  loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_module_superfam Finish");
    })
    .on('error', function(error) {
      logger.error('load_module_superfam.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_module_superfam');
    logger.error(error);
  }
}
```

load_plasmid_superfam

Carga la tabla *plasmid_superfam* que relaciona todos los plásmidos con las superfamilias de sus proteínas.

```
function load_plasmid_superfam(max_lines=argv.max_lines) {
  logger.info("load_plasmid_superfam Init");
  const INPUT = './DATA/plasmid_modules/load_init/plasmid_superfam.mat';
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nplasmids = 0;
  let insert_plasmid_superfam = plasmid_modules_db.prepare("INSERT OR REPLACE
    ↪ INTO plasmid_superfam VALUES(@plasmid_id, @superfam_id)");
  try {
    // Format example:
    //Plasmids        1       2       3       4       5
    //NZ_CP015942.1   0       0       0       0       0       0
    //NZ_CP014521.1   0       0       0       0       0       0
    //...
    let readInterface = create_read_interface(INPUT, 'load_plasmid_superfam');
    let superfams;
    let init = true;
    readInterface.on('line', function(line) {
      if (nplasmids >= max_lines) {
        readInterface.close();
        return;
      }
      let fields = {};
      let line_split = line.split(/\s+/g);
      //console.log(line_split);
      if (init) {
        superfams = line_split;
        init = false;
      } else {
        for (let superfam_idx in line_split) {
          //console.log(superfam_idx);
          if (superfam_idx == 0) {
            fields['plasmid_id'] = line_split[0];
          } else if (line_split[superfam_idx] === '1') {
            fields['superfam_id'] = superfams[superfam_idx];
            insert_plasmid_superfam.run(fields);
          }
        }
        nplasmids++;
      }
      if (nplasmids%200 === 0) {
        logger.info(`load_plasmid_superfam.readInterface.on ${nplasmids}
    ↪ plasmids loaded`);
      }
    })
    .on('close', function() {
```

```
      logger.info(`load_plasmid_superfam.readInterface.close: ${nplasmids}
    ↪ plasmids loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_plasmid_superfam Finish");
    })
    .on('error', function(error) {
      logger.error('load_plasmid_superfam.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_plasmid_superfam');
    logger.error(error);
  }
}
```

load_plasmid_module

Carga la tabla *plasmid_ module* que relaciona todos los plásmidos con los módulos funcionales que contienen.

```
function load_plasmid_module(max_lines=argv.max_lines) {
  logger.info("load_plasmid_module Init");
  const INPUT = './DATA/plasmid_modules/load_init/plasmid_module.mat';
  //const DB_OPTIONS = { verbose: logger.debug };
  const DB_OPTIONS =  {}
  plasmid_modules_db = new sqlite(DB_NAME, DB_OPTIONS);
  let nplasmids = 0;
  let insert_plasmid_module = plasmid_modules_db.prepare("INSERT OR REPLACE INTO
    ↪  plasmid_module VALUES(@plasmid_id, @module_id)");
  try {
    // Format example:
    //Plasmids        1       2       3       4       5
    //NZ_CP015942.1   0       0       0       0       0       0
    //NZ_CP014521.1   0       0       0       0       0       0
    //...
    let readInterface = create_read_interface(INPUT, 'load_plasmid_module');
    let modules;
    let init = true;
    readInterface.on('line', function(line) {
      if (nplasmids >= max_lines) {
        readInterface.close();
        return;
      }
      let fields = {};
      let line_split = line.split(/\s+/g);
      //console.log(line_split);
      if (init) {
        modules = line_split;
        init = false;
      } else {
```

```
        for (let module_idx in line_split) {
          if (module_idx == 0) {
            fields['plasmid_id'] = line_split[0];
          } else if (line_split[module_idx] === '1') {
            fields['module_id'] = modules[module_idx];
            insert_plasmid_module.run(fields);
          }
        }
        nplasmids++;
      }
      if (nplasmids%200 === 0) {
        logger.info(`load_plasmid_module.readInterface.on ${nplasmids} plasmids
    ↪ loaded`);
      }
    })
    .on('close', function() {
      logger.info(`load_plasmid_module.readInterface.close: ${nplasmids}
    ↪ plasmids loaded. Params: max_lines => ${max_lines}`);
      logger.info("load_plasmid_module Finish");
    })
    .on('error', function(error) {
      logger.error('load_plasmid_module.readInterface.error');
      logger.error(error);
    });
  } catch(error) {
    logger.error('load_plasmid_module');
    logger.error(error);
  }
}
```

test

Función de prueba.

```
function test(max_lines=argv.max_lines) {
  logger.info("load_plasmidnet_db test " + max_lines);
}

module.exports.test = test;
module.exports.create_model = create_model;
module.exports.load_representatives = load_representatives;
module.exports.load_protein_AA = load_protein_AA;
module.exports.load_protein_norep = load_protein_norep;
module.exports.load_module_superfam = load_module_superfam;
module.exports.load_superfam = load_superfam;
module.exports.load_plasmid_superfam = load_plasmid_superfam;
module.exports.load_plasmid_module = load_plasmid_module;
```

main

Esta función es un envoltorio para la ejecución del resto de funciones.

Primero ajusta el valor de *max_lines*. Después lanza la función cuyo nombre se pasa en el parámetro *step*.

Permite la ejecución de todos los pasos en secuencia si el parámetro *step* está informado como *all*.

```
if (argv.max_lines === 'all' || argv.max_lines === 'ALL' || argv.max_lines === '
    ↪ A' || argv.max_lines === 'a') {
  argv.max_lines = Number.MAX_SAFE_INTEGER;
}
if (argv.step === 'all') {
  create_model();
  load_representatives();
  load_protein_AA();
  load_protein_norep();
  load_module_superfam();
  load_superfam();
  load_plasmid_superfam();
  load_plasmid_module();
} else {
  logger.info("load_plasmidnet_db main: Executing step " + argv.step);
  module[argv.step]();
}
```

B.4.3. mode-plas.js

Definiciones de los estados del editor *ace* para interpretar las consultas de la DSL propia de PlasmidNet.

```
ace.define("ace/mode/plas_highlight_rules",["require","exports","module","ace/
    ↪ lib/oop","ace/mode/text_highlight_rules"], function(require, exports,
    ↪ module) {
"use strict";

var oop = require("../lib/oop");
var TextHighlightRules = require("./text_highlight_rules").TextHighlightRules;

var PlasHighlightRules = function() {
    var keywords = (
        "plasmid|p|pl|m|mod|module|s|sf|superfam|f|fam|pr|prot|protein|sc|score"
    );

    var keywordMapper = this.createKeywordMapper({
        "keyword": keywords
    }, "identifier", true);

    this.$rules = {
```

```
        "start" : [
        {
            token : keywordMapper,
            regex : "[a-zA-Z][a-zA-Z0-9_]*\\b",
            next: "expanded_operator"
        }, {
            token : "keyword.operator.unary",
            regex : "\\^|\\.n\\.",
            next: "start"
        },
        {
            token : "lbracket.pn",
            regex : "\\(",
            next: "start"
        }],
        "binary_operator": [{
            token : "keyword.operator.binary",
            regex : "\\||\\.o\\.|&|\\.a\\.",
            next: "start"
        }],
        "expanded_operator": [{
            token : "keyword.operator.expandable",
            regex : "==|\\.e\\.|=|\\.i\\.|\\*=|\\.ew\\.|=\\*|\\.sw\\.|!==|\\.ne
↪ \\.|!=|\\.ni\\.|!\\*=|\\.new\\." +
                    "|!=\\*|\\.nsw\\.|>=|\\.ge\\.|<=|\\.le\\.|<|\\.lt\\.|>|\\.gt
↪ \\.",
            next: "expansion"
        }],
        "expansion" : [ {
            token : "value",
            regex : "[a-zA-Z0-9][a-zA-Z0-9_\\.]*\\b",
            next: "series"
        }],
        "series": [
        {
            token : "comma",
            regex : "\\,",
            next: "expansion"
        }, {
            token : "rbracket.pn",
            regex : "\\)",
            next: "binary_operator"
        }]
    };
    this.normalizeRules();
};

oop.inherits(PlasHighlightRules, TextHighlightRules);

exports.PlasHighlightRules = PlasHighlightRules;
});
```

```
ace.define("ace/mode/plas",["require","exports","module","ace/lib/oop","ace/mode
    ↪ /text","ace/mode/plas_highlight_rules"], function(require, exports,
    ↪ module) {
"use strict";

var oop = require("../lib/oop");
var TextMode = require("./text").Mode;
var PlasHighlightRules = require("./plas_highlight_rules").PlasHighlightRules;

var Mode = function() {
    this.HighlightRules = PlasHighlightRules;
    this.$behaviour = this.$defaultBehaviour;
};
oop.inherits(Mode, TextMode);

(function() {

    this.lineCommentStart = "--";

    this.$id = "ace/mode/plas";
}).call(Mode.prototype);

exports.Mode = Mode;

});                 (function() {
                     ace.require(["ace/mode/plas"], function(m) {
                         if (typeof module == "object" && typeof exports == "
    ↪ object" && module) {
                             module.exports = m;
                         }
                     });
                 })();
```

B.5. Gestor de contenidos

B.5.1. gulpfile_cms.js

Este módulo abarca todas las funciones de generación y distribución de contenidos partiendo de la versión estática del sitio (maqueta).

Se generan dos tipos de contenidos por deconstrucción de la maqueta:

1) Página principal del sitio.
2) Artefactos injertables (*widgets*).

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
```

```
const sqlite = require('better-sqlite3');
const axios = require('axios');
const fs = require('fs');
const cheerio = require('cheerio');
const path = require('path');
const forge = require('node-forge');

const argv = require('yargs')
  .default('plasmid_id', 'NZ_LT960791.1')
  .default('pversion', 'v0')
  .default('subversion', 'v1')
  .default('object', 'plasmidnet_datatables.js')
  .default('name', 'sidebar')
  .default('cache', 'modules')
  .argv;

global.__sqlite_cms_db;
```

cms_model_create

Sentencias *sql* de creación del modelo de datos: tablas de contenidos *content*, tabla de versiones *version* y tabla de invalidación de las cachés *cache*.

```
function cms_model_create(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_cms_db = new sqlite('./DATA/cms_model_sqlite.db', { verbose: logger.
    ↪ debug });
  let sql_create_model =
    `CREATE TABLE IF NOT EXISTS content(
      name text NOT NULL,
      version text NOT NULL,
      body text NOT NULL,
      timestamp INTEGER,
      PRIMARY KEY(name, version));
     CREATE TABLE IF NOT EXISTS version(
      object text NOT NULL,
      version text NOT NULL,
      subversion text NOT NULL,
      PRIMARY KEY(object, version));
    CREATE TABLE IF NOT EXISTS cache(
      type text PRIMARY KEY,
      valid text NOT NULL);
    INSERT INTO cache VALUES("statics", "true");
    INSERT INTO cache VALUES("widgets", "true");
    INSERT INTO cache VALUES("data", "true");
    INSERT INTO cache VALUES("modules", "true");
  `;
  __sqlite_cms_db.exec(sql_create_model);
  if (cb) {cb()};
```

```
}
```

cms_model_drop

Sentencias *sql* de borrado (*DROP*) de las tablas del gestor de contenidos.

```
function cms_model_drop(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_cms_db = new sqlite('./DATA/cms_model_sqlite.db', { verbose: logger.
    ↪ debug });
  let sql_drop_model =
    `DROP TABLE content;
     DROP TABLE version;
     DROP TABLE cache;
  `;
  __sqlite_cms_db.exec(sql_drop_model);
  if (cb) {cb()};
}
```

start_cms

Arranca la base de datos de contenidos.

```
function start_cms(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_cms_db = new sqlite('./DATA/cms_model_sqlite.db', { verbose: logger.
    ↪ debug });
  if (cb) {cb()};
}
```

upsert_content

Actualiza el contenido de nombre *name* , con html *body* para la versión de aplicación *version*.

```
function upsert_content(cb, name=argv.name, body, version=argv.pversion) {
  let stmt = __sqlite_cms_db.prepare(`
    INSERT INTO content VALUES(?, ?, ?, ?)
    ON CONFLICT(name, version) DO UPDATE SET
    timestamp = excluded.timestamp,
    body = excluded.body `);
  stmt.run(name, version, body, (new Date()).getTime());
  if (cb) {cb()};
}
```

get_content

Recupera el contenido de nombre *name* , y versión de aplicación *version*. Devuelve el cuerpo, la versión de aplicación y la versión del contenido.

```
function get_content(cb, name=argv.name, pversion=argv.pversion) {
  let stmt = __sqlite_cms_db.prepare("SELECT body, content.version, subversion
    ↪ FROM content, version WHERE version.object = content.name AND content.
    ↪ name= ? AND content.version = ?");
  let ret = stmt.get(name, pversion);
  if (cb) {console.log(ret);cb();} else {return ret;}
}
```

extract_content_from_html

Función recursiva que interpreta el contenido html *html* de la versión estática del sitio, para la aplicación *context* y la versión de aplicación *pversion*. Los contenidos se identifican a partir de su referencia *pn-ref*. Todo el *html* contenido en este *tag*, incluido el del propio *tag* (*outerHtml* en la jerga del navegador), es almacenado en la tabla *content* de la base de datos *cms* y sustituido en la página por un inserto tipo gancho (marcado con el tag *hook*). Estos ganchos serán utilizados por la *web app* para recomponer dinámicamente el código *html* de la página.

Por último, la página de la aplicación es almacenada también en la tabla *content*.

```
function extract_content_from_html(cb, context, html, pversion=argv.pversion) {
  let $ = cheerio.load(html);
  // Main app html page
  $('[pn-ref]').each(function (i, e) {
    if (i > 0) {
      let attr = $(this)[0].attribs;
      let outer_html = cheerio.html($(this));
      let inner_html = $(this).html();
      let pn_ref = attr['pn-ref'];
      let pn_state = attr['pn-state'] || "onLoad";
      //let pn_hook = attr['pn-hook'] || "0";
      let pn_multi = attr['pn-multi'];
      extract_content_from_html(undefined, pn_ref, outer_html, pversion);
      // Create hooks on parent html
      if (pn_multi === "")
        $.root().find(`[pn-ref="${pn_ref}"]`).replaceWith(`<div pn-ref=${pn_ref}
    ↪  pn-hook=${0} pn-multi pn-state=${pn_state}></div>`);
      else
        $.root().find(`[pn-ref="${pn_ref}"]`).replaceWith(`<div pn-ref=${pn_ref}
    ↪  pn-hook=${0} pn-state=${pn_state}></div>`);
    }
  });
  let context_content;
  // Distintion between whole page and internal contents based on head
  // cheerio loads in some point the inclosed tags body and html
  if ($('head').html()) context_content = $.root().html();
  else context_content = $('body').html()
```

```
  upsert_content(undefined, context, context_content, pversion);
  set_widget_version(context, context_content);
  if (cb) {cb()};
}
```

do_upsert_content

Lanza la extracción de contenidos desde la maqueta 'plasmidnet_cms.html' para la versión *pversion* para la *web app* PlasmidNet.

```
function do_upsert_content(cb, pversion=argv.pversion) {
  let plasmidnet = fs.readFileSync('./CODE/plasmidnet/web/plasmidnet_cms.html');
  extract_content_from_html(undefined, 'plasmidnet', plasmidnet, pversion);
  if (cb) {cb()};
}
```

do_upsert_content_doc

Lanza la extracción de contenidos desde la maqueta *plasmidnet_doc_cms.html* sobre la versión *pversion* para la *web app* documental de PlasmidNet.

```
function do_upsert_content_doc(cb, pversion=argv.pversion) {
  let plasmidnet_doc = fs.readFileSync('./CODE/plasmidnet/web/plasmidnet_doc_cms
    ↪ .html');
  extract_content_from_html(undefined, 'plasmidnet_doc', plasmidnet_doc,
    ↪ pversion);
  if (cb) {cb()};
}
```

set_version

Actualiza la versión de un fichero (parámetro *file*). La versión se calcula como reducción criptográfica *md5* del contenido.

```
function set_version(file, t) {
  let logger = require_dyn("logger_app");
  try {
    let filename = path.basename(file.path);
    let md = forge.md.md5.create();
    md.update(file.contents);
    let md5_sum = md.digest().toHex();
    upsert_version(undefined, filename, argv.pversion, md5_sum);
  } catch(error) {
    logger.error(`gulpfile.cms.set_version  ${file} ${error.message}`);
    logger.error(file);
```

```
    logger.error(file.path);
  }
}
```

set_widget_version

Actualiza la versión de un *widget* de nombre _content_name y html *content*. La versión se calcula como *md5* del contenido y se almacena en el campo *subversion* de la tabla *version*.

```
function set_widget_version(content_name, content, t) {
  let logger = require_dyn("logger_app");
  try {
    let md = forge.md.md5.create();
    md.update(content);
    let md5_sum = md.digest().toHex();
    upsert_version(undefined, content_name, argv.pversion, md5_sum);
  } catch(error) {
    logger.error(`gulpfile.cms.set_widget_version  ${content_name} ${error.
    ↪ message}`);
  }
}
```

upsert_version

Actualiza la versión del objeto referenciado por *object*, para la versión de aplicación *pversion* y versión de objeto *subversion*.

```
function upsert_version(cb, object=argv.object, pversion=argv.pversion,
    ↪ subversion=argv.subversion) {
  let stmt = __sqlite_cms_db.prepare(`
    INSERT INTO version VALUES(?, ?, ?)
    ON CONFLICT(object, version) DO UPDATE SET
    subversion = excluded.subversion `);
  stmt.run(object, pversion, subversion);
  if (cb) {cb()};
}
```

get_subversion

Obtiene la versión del objeto referenciado por *object*, para la versión de aplicación *pversion*.

```
function get_subversion(cb, object=argv.object, pversion=argv.pversion) {
  let logger = require_dyn("logger_app");
  let stmt = __sqlite_cms_db.prepare("SELECT subversion FROM version WHERE
    ↪ object = ? AND version = ?");
```

```
  let ret = stmt.pluck().get(object, pversion);
  let subversion;
  if (ret) subversion = ret.toString();
  else subversion = undefined;
  if (cb) {logger.error(subversion);cb();} else {return subversion;}
}
```

get_cache

Obtiene la validez de la cache referenciada por *cache*.

```
function get_cache(cb, cache=argv.cache) {
  let logger = require_dyn("logger_app");
  let stmt = __sqlite_cms_db.prepare("SELECT valid FROM cache WHERE cache = ?");
  let ret = stmt.pluck().get(cache);
  let valid;
  if (ret) valid = ret.toString();
  else valid = undefined;
  if (cb) {console.log(valid);cb();} else {return valid;}
}
```

create_home

Crea la página principal de la aplicación *plasmidnet*, descargándola de la tabla de contenidos.

```
function create_home(cb, pversion=argv.pversion) {
  let content = get_content(undefined, 'plasmidnet', pversion).body;
  fs.writeFileSync('./CODE/plasmidnet/web/plasmidnet.html', content);
  if (cb) {cb()};
}
```

create_home_doc

Crea la página principal de la aplicación *plasmidnet_doc*, descargándola de la tabla de contenidos.

```
function create_home_doc(cb, pversion=argv.pversion) {
  let content = get_content(undefined, 'plasmidnet_doc', pversion).body;
  fs.writeFileSync('./CODE/plasmidnet/web/plasmidnet_doc.html', content);
  if (cb) {cb()};
}

module.exports.cms_upsert_content = series(start_cms, do_upsert_content,
    ↪ create_home);
```

```
module.exports.cms_upsert_content_doc = series(start_cms, do_upsert_content_doc,
    ↪  create_home_doc);
module.exports.cms_model_create = cms_model_create;
module.exports.cms_model_drop = cms_model_drop;
module.exports.start_cms = start_cms;
module.exports.create_home = create_home;
module.exports.create_home_doc = create_home_doc;
module.exports.do_upsert_content = do_upsert_content;
module.exports.do_upsert_content_doc = do_upsert_content_doc;
module.exports.upsert_version = upsert_version;
module.exports.get_subversion = get_subversion;
module.exports.get_content = get_content;
module.exports.get_cache = get_cache;
module.exports.set_version = set_version;
module.exports.test_create_home = series(start_cms, create_home);
module.exports.test_upsert_version = series(start_cms, upsert_version);
module.exports.test_get_subversion = series(start_cms, get_subversion);
module.exports.test_get_cache = series(start_cms, get_cache);
module.exports.test_get_content = series(start_cms, get_content);
module.exports.shares = module.exports;
```

B.6. Orquestador distribuido de tareas

B.6.1. gulpfile_pipelines.js

Implementa las tareas *gulp* del orquestador de flujos de trabajo.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const axios = require('axios');
const execa = require('execa');
const fs = require('fs');
const path = require('path');
const forge = require('node-forge');
```

Las siguientes constantes definen los estados por los que avanza cada *step*.

```
const STATE_MD5_DOWNLOAD_PENDING = 'MD5_DOWNLOAD__PENDING'; //md5 download of
    ↪ outputs pending
const STATE_MD5_DOWNLOAD_FINISHED = 'MD5_DOWNLOAD__FINISHED'; //md5 download of
    ↪ outputs finished
const STATE_MD5_CHK_PENDING = 'MD5_CHK__PENDING'; //md5 calculation of inputs
    ↪ pending'
const STATE_MD5_CHK_FINISHED = 'MD5_CHK__FINISHED'; //md5 calculation of outputs
    ↪  finished'
const STATE_MD5_COMPARE_PENDING = 'MD5_COMPARE__PENDING'; //md5 compare between
    ↪ chk and down pending'
```

```
const STATE_MD5_COMPARE_FINISHED = 'MD5_COMPARE__FINISHED'; //md5 compare
    ↪ between chk and down finished'
const STATE_DOWNLOAD_PENDING = 'DOWNLOAD__PENDING'; //md5 download files pending
    ↪ '
const STATE_DOWNLOAD_FINISHED = 'DOWNLOAD__FINISHED'; //md5 download files
    ↪ finished'
const STATE_MD5_DOWN_PENDING = 'MD5_DOWN__PENDING'; //md5 calculation of
    ↪ download inputs pending'
const STATE_MD5_DOWN_FINISHED = 'MD5_DOWN__FINISHED'; //md5 calculation of
    ↪ download inputs finished'
const STATE_MD5_VERIFY_PENDING = 'MD5_VERIFY__PENDING'; //md5 compare with new
    ↪ downloads pending'
const STATE_MD5_VERIFY_FINISHED = 'MD5_VERIFY__FINISHED'; //md5 compare with new
    ↪  downloads finished'
const STATE_RUNNING = 'RUNNING';
const STATE_EXEC_PENDING = 'EXEC__PENDING';
const STATE_EXEC_FINISHED = 'EXEC__FINISHED';
const STATE_MD5_OUT_PENDING = 'MD5_OUT__PENDING'; //md5 calculation of outputs
    ↪ pending'
const STATE_FINISHED = 'FINISHED'; //md5 calculation of outputs finished'
const STATE_ERROR = 'ERROR'; //error
const STATE_INIT = 'INIT'; //initial state
const STATE_TASK_RUNNING = 'TASK_RUNNING'; //task state RUNNING
const STATE_TASK_ERROR = "TASK_ERROR";
```

STATE_STEP_TRANSITIONS define las transiciones entre estados.

```
const STATE_STEP_TRANSITIONS = {
  'INIT' : STATE_MD5_DOWNLOAD_PENDING,
  'MD5_DOWNLOAD__FINISHED': STATE_MD5_CHK_PENDING,
  'MD5_CHK__FINISHED': STATE_MD5_COMPARE_PENDING,
  'MD5_COMPARE__FINISHED': STATE_DOWNLOAD_PENDING,
  'DOWNLOAD__FINISHED': STATE_MD5_DOWN_PENDING,
  'MD5_DOWN__FINISHED': STATE_MD5_VERIFY_PENDING,
  'MD5_VERIFY__FINISHED': STATE_EXEC_PENDING,
  'EXEC__FINISHED': STATE_MD5_OUT_PENDING
}

const DEFAULT_NODE = "plasmid01";
const DEFAULT_TASK_ID = "another";
const DATA_DIR = './DATA';
const WEB_DIR = 'files';
const PIPELINE_DIR = 'pipelines';
const DEFAULT_PIPELINE = 'plasmid_modules';
const TEST_PIPELINE = 'test';
const argv = require('yargs')
  .default('pipeline', DEFAULT_PIPELINE)
  .default('task_provider', DEFAULT_NODE)
  .default('task_owner', DEFAULT_NODE)
  .default('task_client', DEFAULT_NODE)
  .default('task_id', DEFAULT_TASK_ID)
```

```
  .default('command', 'plas01_test.pl')
  .default('step_from', 0)
  .default('step_to', 0)
  .default('step_id', 0)
  .default('step', '{"step_id": "2", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]}')
  .default('steps', '[{"step_id":"0", "provider":"plasmid01","command":"
    ↪ plas00_plasmid_gi__PLSDB_extract.tcsh","stdout":[]},{"step_id":"1","
    ↪ provider":"plasmid01","command":"plas01_plasmid_protein__NCBI_download.pl
    ↪ ","stdout":[]},{"step_id":"2","provider":"plasmid01","command":"
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "stdout":[]}]')
  .default('task', `{"task_id" : ${DEFAULT_TASK_ID}, "pipeline": "
    ↪ plasmid_modules"}`)
  .default('input', 'plasmid_GI.tsv')
  .default('state', STATE_INIT)
  .default('template', '^gulp/')
  .argv;
const none = function(){};
const PROMISE_ADHOC = new Promise((resolve, reject) => {resolve("   No files to
    ↪ process")});

let pipelines, inputs, outputs, provider_url, task_model;
```

load_models

Carga el módulo de acceso a la base de datos de tareas definido en la variable global *__model* (por defecto *sqlite*) y el módulo de definición de flujos (*pipelines*).

```
function load_models() {
  task_model = require_dyn("gulpfile_task_model_" + __model);
  let pipeline_model = require_dyn("pipelines");
  pipelines = pipeline_model.pipelines;
  inputs = pipeline_model.inputs;
  outputs = pipeline_model.outputs;
  provider_url = pipeline_model.provider_url;
}
```

assign_provider

En esta función se asigna el proveedor del *step*. Los proveedores candidatos están definidos en la información del *step*. Actualmente se devuelve el primero de los candidatos, pero en el futuro se aplicará algún tipo de algoritmo *round robin* o se tendrá en cuenta el estado de recursos de sistema (CPU, disco) de los diferentes proveedores.

```
function assign_provider(step_data) {
  // TODO assign algorithm
```

```
  return step_data.providers[0];
}
```

get_step_outputs

Devuelve todos los ficheros de salida de la tarea y paso identificados por *task* y *step*. Se les concatena la constante interna *DIR* del sistema de orquestación que hace referencia al directorio raíz de datos. Tiene en cuenta las precedencias: los ficheros definidos a nivel *step* tienen precedencia sobre los definidos a nivel de *pipeline*.

```
function get_step_outputs(task, step) {
  let step_outputs;
  // Outputs at step level override outputs at step.name level
  if (pipelines[task.pipeline].steps[step.step_id].outputs)
    step_outputs = pipelines[task.pipeline].steps[step.step_id].outputs.slice(0)
    ↪ ;
  else if (step.name in outputs)
    step_outputs = outputs[step.name].slice(0);
  // If filename doesn't include task_path, we insert it
  else
    step_outputs = [];
  for (let index in step_outputs) {
    if (step_outputs[index].substr(0, 2) !== '__') {
      step_outputs[index] = '__DIR__/' + step_outputs[index];
    }
  }
  return step_outputs;
}
```

get_step_inputs

Devuelve todos los ficheros de entrada de la tarea y paso identificados por *task* y *step*. Se les concatena la constante interna *DIR* del sistema de orquestación que hace referencia al directorio raíz de datos si no viene referenciado en el fichero. Tiene en cuenta las precedencias: los ficheros definidos a nivel *step* tienen precedencia sobre los definidos a nivel de *pipeline*.

```
function get_step_inputs(task, step) {
  let step_inputs;
  // Inputs at step level override inputs at step.name level
  if (pipelines[task.pipeline].steps[step.step_id].inputs)
    step_inputs = pipelines[task.pipeline].steps[step.step_id].inputs.slice(0);
  else if (step.name in inputs)
    step_inputs = inputs[step.name].slice(0);
  else
    step_inputs = [];
  for (let index in step_inputs) {
    if (step_inputs[index].substr(0, 2) !== '__') {
```

```
      step_inputs[index] = '__DIR__/' + step_inputs[index];
    }
  }
  return step_inputs;
}
```

get_step_input_provider

Devuelve el proveedor del fichero de entrada (*input*) del paso, cuya información completa se envía en *step*, de la tarea cuya información completa se envía en *task*.

```
async function get_step_input_provider(cb, input=argv.input, task=JSON.parse(
    ↪ argv.task), step=JSON.parse(argv.step)) {
  let logger = require_dyn("logger_app");
  let provider_name;
  let provider_step_id;
  let command = step.command;
  let steps = await task_model.get_steps(cb, task.task_id);
  for (step of steps) {
    let step_outputs = get_step_outputs(task, step);
    if (step_outputs.includes(input)) {
      provider_name = step.provider;
      provider_step_id = step.step_id;
      break;
    }
  }
  let provider = {provider: provider_name, step_id: provider_step_id}
  logger.debug("Provider is: " + provider);
  cb();
  return provider;
}
```

get_step_providers

Devuelve los proveedores de todos los ficheros de entrada del paso, cuya información completa se envía en *step*, de la tarea cuya información completa se envía en *task*.

```
async function get_step_providers(cb, task=JSON.parse(argv.task), step=argv.step
    ↪ ) {
  let logger = require_dyn("logger_app");
  load_models();
  let providers = {};
  let command = step.command;
  let name = step.name;
  for (input of get_step_inputs(task, step)) {
    let provider = await get_step_input_provider(cb, input, task, step);
    providers[input] = provider;
```

```
  }
  logger.debug(providers);
  cb();
  return providers;
}
```

step_update_error

Actualiza el estado del paso, cuya información completa se envía en *step*, de la tarea cuya información completa se envía en *task*.

El estado se actualiza de acuerdo a lo informado en *state_error*. A la información del error se añade el contenido del parámetro *comment*.

También se actualiza el estado del *step* en el nodo coordinador mediante *task_update*.

```
async function step_update_error(cb, logger, comment, error, task, step,
    ↪ state_error) {
  try {
    logger.error(comment);
    if (error) {
      //logger.error(error.stack.split('\n'));
      logger.error(error);
      if (error.stderr !== undefined) {
        step.stderr = error.stderr.split('\n');
      }
      step.error_message = error.message;
    }
    step.state = state_error;
    if (Array.isArray(comment)) {
      step.stdout = step.stdout.concat(comment);
    } else if (comment !== undefined) {
      step.stdout = step.stdout.concat(comment.split('\n'));
    }
    step.finish_time = new Date().toISOString().replace(/T/, ' '),
    //logger.error(step);
    await task_model.update(cb, task, step);
    task_update(cb, task.task_id, step);
    cb(error);
  } catch(err) {
    logger.error('step_update_error');
    logger.error(err.stack);
  }
}
```

task_update_error

Actualiza el estado del la tarea, cuya información completa se envía en *task*.

El estado se actualiza de acuerdo a lo informado en *errorcode*. A la información del error se añade el contenido de los parámetros *stack*, *message* y *errorcode*. En el log de errores se incluye el contenido del parámetro *from* que hace referencia al servicio que ha ejecutado la función.

```
async function task_update_error(cb, from, logger, task, stack, message,
    ↪ errorcode) {
  logger.error(`task_update_error ${from} ${errorcode} ${message}`);
  let new_stack = stack.split('\n').slice(0,3);
  logger.error(new_stack);
  task.state = STATE_TASK_ERROR;
  task.message = message;
  task.errorcode = errorcode;
  task.stack = new_stack;
  await task_model.update(cb, task, '');
  cb(error);
}
```

step_update_ok

Actualiza el estado del paso, cuya información completa se envía en *step*, de la tarea cuya información completa se envía en *task*.

El estado se actualiza de acuerdo a lo informado en *state_ ok*. A la información de ejecución (*stdout*) se añade el contenido del parámetro *comment*. Se ajusta el estado de la tarea de acuerdo al estado del paso.

También se actualiza el estado del *step* en el nodo coordinador mediante *task_ update*.

```
async function step_update_ok(cb, logger, comment, task, step, state_ok) {
  try {
    logger.debug('gulpfile_pipelines.step_update_ok init');
    if (Array.isArray(comment)) {
      step.stdout = step.stdout.concat(comment);
    } else if (comment !== undefined) {
      step.stdout = step.stdout.concat(comment.split('\n'));
    }
    step.state = state_ok;
    task.state = state_ok;
    step.finish_time = new Date().toISOString().replace(/T/, ' '),
    logger.debug('gulpfile_pipelines.step_update_ok: ' + step.step_id);
    await task_model.update(cb, task, step);
    task_update(cb, task.task_id, step);
    logger.debug('gulpfile_pipelines.step_update_ok finish');
    cb();
  } catch(err) {
    logger.error('step_update_ok');
    logger.error(err.stack);
  }
}
```

check_completion

Verifica la finalización de la lista de *promises* declarada en entrada para la tarea informada en el parámetro *task* y el paso informado en *step*. Dependiendo del resultado, correcto o no, actualiza la información del paso en base a la función correspondiente.

```
function check_completion(cb, logger, promises, legend, state_ok, state_error,
    ↪ task, step) {
  Promise.all(promises)
    .then(function(data) {
      let comment = [`gulpfile_pipelines.check_completion ${legend}: all files
    ↪ processed ok`].concat(data);
      step_update_ok(cb, logger, comment, task, step, state_ok);
    })
    .catch(function(data) {
      logger.error('gulpfile_pipelines.check_completion' + data);
      let comment = [`gulpfile_pipelines.check_completion ${legend}: there are
    ↪ files with process ko`].concat(data.comment);
      step_update_error(cb, logger, comment, data.error, task, step, state_error
    ↪ );
    });
}
```

create_download_promise

Crea una *promise* para envolver un *stream* de escritura (parámetro *write_stream*) para el fichero especificado por el parámetro *file*.

Es utilizado por la función *file_download* para optimizar la memoria utilizada para escribir el fichero recibido.

```
function create_download_promise(write_stream, file) {
  return new Promise((resolve, reject) => {
    write_stream
      .on('finish', function() {
        try {
          let comment = `   gulpfile_pipelines.create_download_promise ${file}
    ↪ OK`;
          resolve(comment);
        } catch(error) {
          let comment = `   gulpfile_pipelines.create_download_promise ${file}
    ↪ KO. INTERNAL`;
          reject({error: error, comment: comment});
        }
      })
      .on('error', function (error) {
        let comment = `   gulpfile_pipelines.create_download_promise File ${file
    ↪ } KO`;
        reject({error: error, comment: comment});
      });
  });
```

```
}
```

create_md5_calc_promise

Crea una *promise* para envolver un *stream* de lectura que se crea en la misma función, con el fin de leer el fichero *file* y generar su código *md5*, que se graba finalmente en el fichero *file_out*.

```
function create_md5_calc_promise(logger, file, file_out) {
  return new Promise((resolve, reject) => {
    let read_stream = fs.createReadStream(file);
    let err;
    let md = forge.md.md5.create();
    read_stream
      .on('data', function(chunk) {
        try {
          md.update(chunk);
        } catch(error) {
          err = error;
        }
      })
      .on('end', function() {
        if (err) {
          let comment = `   gulpfile_pipelines.create_md5_calc_promise for file
    ↪ ${file} not calculated. INTERNAL`;
          reject({error: err, comment: comment});
        } else {
          try {
            //throw new Error('AAAAAUCH!');
            let md5_sum = md.digest().toHex();
            fs.writeFileSync(file_out, md5_sum);
            let comment = `   gulpfile_pipelines.create_md5_calc_promise ${file}
    ↪  ${file_out} with md5 ${md5_sum} calculated`;
            resolve(comment);
          } catch(error) {
            let comment = `   gulpfile_pipelines.create_md5_calc_promise on end.
    ↪  md5 for file ${file} not calculated. INTERNAL 2`;
            reject({error: error, comment: comment});
          }
        }
      })
      .on('error', function (error) {
        comment = `   gulpfile_pipelines.create_md5_calc_promise: ${file} not
    ↪ calculated`;
        reject({error: error, comment: comment});
      });
  });
}
```

resolve_filename

Modifica los nombres de las etiquetas (*PIPE_DIR*, *DIR*, *TASK_DIR*) del nombre del fichero *filename* por sus nombres correspondientes en el sistema de archivos. Estas etiquetas son utilizadas en la especificación de los flujos.

Se devuelven dos nombres, el físico del fichero de archivos y el que verá el servidor web, que se diferencian en el punto de montaje especificado en *DATA_DIR*.

```
function resolve_filename(filename, task) {
  let file_web = filename
    .replace(/__PIPE_DIR__/gi, `${task.pipeline}`)
    .replace(/__DIR__/gi, `${task.task_id}`)
    .replace(/__TASK_DIR__/gi, `${task.task_id}`);
  let file_disk = `${DATA_DIR}/${file_web}`;
  // Create directories
  fs.mkdirSync(path.dirname(file_disk), {recursive:true});
  return [file_web, file_disk];
}
```

expand_path

Función de utilidad que dado un directorio que se utiliza de modelo *template*, devuelve todos los nombres de los ficheros que terminan en *md5*.

```
function expand_path(cb, template=argv.template) {
  let path_name = path.dirname(template);
  let regexp = path.basename(template + '(?!.*md5)');
  let files = fs.readdirSync(path_name, {withFileTypes: true})
    .filter(item => !item.isDirectory())
    .map(item => item.name)
    .filter(name => name.match(new RegExp(regexp)))
    .map(name => path_name + '/' + name);
  cb();
  return files;
}
```

md5_calc

Calcula los *md5* de todos los ficheros pasados en la matriz *files* y se almacenan en ficheros cuyo nombre utiliza el sufijo *sufix* especificado, el nombre del fichero y el identificador del paso *step_id*.

```
function md5_calc(cb, task, step, legend, sufix, files, state_ok, state_ko) {
  let logger = require_dyn("logger_app");
  try {
    logger.info(`${legend} ${task.task_id} ${step.step_id}`);
    let task_id = task.task_id;
```

```
    let step_id = step.step_id;
    let promises = [];
    let index = 0;
    promises[index] = PROMISE_ADHOC; //ad-hoc to force verification if not files
    ↪  pending
    for (let filename of files) {
      let [file_web, file_disk] = resolve_filename(filename, task);
      let files_expanded = expand_path(none, file_disk);
      logger.debug('++++');
      logger.debug(files_expanded);
      for (let file of files_expanded) {
        if (fs.existsSync(file)) {
          let file_md5 = `${file}.${step_id}.${sufix}.md5`;
          promises[index] = create_md5_calc_promise(logger, file, file_md5);
          logger.debug('gulpfile_pipelines.md5_calc created promise for ' +
    ↪ file_md5);
          index++;
        }
      }
    }
    check_completion(cb, logger, promises, legend, state_ok, state_ko, task,
    ↪ step);
  } catch(error) {
    step_update_error(cb, logger, legend + ' error', error, task, step, state_ko
    ↪ );
  }
}
```

file_download

Transferencia de ficheros entre ejecutores de pasos. El ejecutor solicita a cada uno de los ejecutores predecesores los ficheros de entrada (de salida desde el punto de vista del predecesor).

La relación completa de ficheros a descargar se especifica en el parámetro de entrada *files*. Se informan también los siguientes campos:

- **task** Contenido completo de la tarea
- **step** Contenido completo del step a ejecutar
- **legend** Texto aclaratorio del tipo de transferencia que aparecerá en los log.
- **sufix_from** Sufijo de los ficheros de origen que queremos modificar en destino.
- **sufix_to** Sufijo de destino que sustituye al *sufix_from*
- **files** Matriz de ficheros a descargar.
- **state_ok** Identificador del estado al que se debe actualizar el *step* si el proceso termina sin errores.
- **state_ko** Identificador del estado al que se debe actualizar el *step* si el proceso termina con errores.

Para identificar los ficheros se pasa por parámetro los datos completos de tarea *task* y paso *step*.

La función lanza en paralelo asíncronamente todas las solicitudes de descarga. Para ello emplea la utilidad *axios*, que es el mismo paquete que utilizamos en la *web app*. Para cada solicitud registra una *promise* mediante la función *create_download_promise*. Finalmente se comprueba el resultado global de todas las descargas solicitadas mediante la función *check_completion*.

Ante cualquier error se actualiza el estado del *step* al *state_ko* pasado por parámetro (*step_update_error*).

Los proveedores de cada fichero se consultan en la estructura del *step* (*step.providers*), que antes se ha construido a partir de la información maestra de *pipelines* y la propia información del *step* (función *get_step_inputs*).

```
async function file_download(cb, task, step, legend, sufix_from, sufix_to, files
    ↪ , state_ok, state_ko)  {
  let logger = require_dyn("logger_app");
  try {
    logger.info(`gulpfile_pipelines.file_download ${legend} ${task.task_id}_${
    ↪ step.step_id}`);
    let task_id = task.task_id;
    let step_id = step.step_id;
    let command = step.command;
    let name = step.name;
    let promises = [];
    let index = 0;
    promises[index] = PROMISE_ADHOC;
    for (let filename of files) {
      logger.debug('gulpfile_pipelines.file_download ' + filename);
      let [file_web_ini, file_disk_ini] = resolve_filename(filename, task);
      let files_expanded = expand_path(none, file_disk_ini);
      logger.debug('++++D');
      logger.debug(files_expanded);
      for (let file_disk of files_expanded) {
        let file_out;
        if (sufix_to === '')
          file_out = `${file_disk}`;
        else
          file_out = `${file_disk}.${step_id}.${sufix_to}`;
        logger.debug('gulpfile_pipelines.file_download ' + legend + ' ' +
    ↪ file_out);
        if (fs.existsSync(file_out)) fs.unlinkSync(file_out);
        let provider = step.providers[filename].provider;
        logger.debug("gulpfile_pipelines.file_download PROVIDER " + provider);
        if (provider !== undefined) {
          logger.debug('gulpfile_pipelines.file_download ' + legend + ' ' +
    ↪ provider);
          let file_web = file_disk.replace(DATA_DIR, WEB_DIR);
          let file;
          if (sufix_from === '')
            file = `${file_web}`;
          else
            file = `${file_web}.${step.providers[filename].step_id}.${sufix_from
    ↪ }`;
          let write_stream = fs.createWriteStream(file_out);
```

```
          write_stream.on('error', function (error) {
            logger.error('gulpfile_pipelines.file_download' + ' ' + error.
    ↪ message);
          });
          let response = await axios({ url: `${provider_url[provider]}/${file}`,
            method: 'GET', responseType: 'stream', headers: { 'x-compress' :
    ↪ 1}});
          promises[index] = create_download_promise(write_stream, file_out);
          index++;
          response.data.pipe(write_stream);
        }
      }
    }
    check_completion(cb, logger, promises, legend, state_ok, state_ko, task,
    ↪ step);
  } catch(error) {
    step_update_error(cb, logger, legend + " error", error, task, step, state_ko
    ↪ );
  }
}
```

md5_compare

Compara todos los ficheros *md5* descargados (sufijos *down.md5*) con los *md5* calculados *chk.md5*.

Los ficheros para los que no existe coincidencia son almacenados en el campo *download_required* del paso. La existencia de ficheros en esta estructura indica al orquestador que es necesaria la descarga de estos ficheros para poder ejecutar el paso.

Provee dos formas de ejecución, con *type* igual a *verify* cambia además el estado del paso a estado erróneo.

```
function md5_compare(cb, task, step, type, files, legend, state_ok, state_ko) {
  let logger = require_dyn("logger_app");
  try {
    logger.info(`${legend} ${task.task_id} ${step.step_id}`);
    let task_id = task.task_id;
    let step_id = step.step_id;
    let command = step.command;
    let name = step.name;
    let download_required = [];
    for (let filename of files) {
      let [file_web, file_disk] = resolve_filename(filename, task);
      let files_expanded = expand_path(none, file_disk);
      for (let file of files_expanded) {
        let file_md5_down = `${file}.${step_id}.down.md5`;
        let file_md5_chk = `${file}.${step_id}.chk.md5`;
        if (fs.existsSync(file_md5_down) && fs.existsSync(file_md5_chk)) {
          let md5_down = fs.readFileSync(file_md5_down);
          let md5_chk = fs.readFileSync(file_md5_chk);
```

```
          if (Buffer.compare(md5_down, md5_chk)) {
            comment = "gulpfile_pipelines.step_md5_compare unmatch " + md5_down
    ↪ + " " + md5_chk;
            logger.debug(comment);
            download_required.push(filename);
          }
        } else {
          download_required.push(filename);
        }
      }
    }
    step.download_required = download_required;
    if (download_required.length > 0 && type === 'verify') {
      step_update_error(cb, logger, legend + " unmatch", null, task, step,
    ↪ state_ko);
    } else {
      step_update_ok(cb, logger, [legend + " ok"], task, step, state_ok);
    }
  } catch(error) {
    step_update_error(cb, logger, legend + " error", error, task, step, state_ko
    ↪ );
  }
}
```

step_md5_download

Descarga desde el directorio del proveedor de los ficheros hasta el directorio local (del elecutor del *step*) todos los ficheros *md5*, y se renombran cambiando el sufijo *out.md5* a *down.md5*. Para ello utiliza la función *file_download* donde informa también el estado OK y el estado KO de este subproceso.

Para identificar los ficheros se pasan por parámetro los datos completos de tarea *task* y paso *step*.

```
function step_md5_download(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.
    ↪ step)) {
  load_models();
  file_download(cb, task, step, 'step_md5_download', 'out.md5', 'down.md5',
    ↪ get_step_inputs(task, step), STATE_MD5_DOWNLOAD_FINISHED, STATE_ERROR);
}
```

step_download

Descarga desde el directorio del proveedor de los ficheros hasta el directorio local (del ejecutor del *step*) todos los ficheros *md5*, y se renombran cambiando el sufijo *out.md5* a *down.md5*.

Los ficheros que deben descargarse son los que el paso almacena en la matriz *download_required*.

```
function step_download(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.step
    ↪ )) {
  load_models();
  file_download(cb, task, step, 'step_download', '', '', step.download_required,
    ↪  STATE_DOWNLOAD_FINISHED, STATE_ERROR);
}
```

step_md5_chk

Calcula los *md5* de todos los ficheros de entrada del paso y los almacena en ficheros con sufijo *chk*.

```
function step_md5_chk(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.step)
    ↪ ) {
  load_models();
  md5_calc(cb, task, step, 'step_md5_chk', 'chk', get_step_inputs(task, step),
    ↪ STATE_MD5_CHK_FINISHED, STATE_ERROR);
}
```

step_md5_out

Calcula los *md5* de todos los ficheros de salida del paso y los guarda en local con el sufijo *out*.

```
function step_md5_out(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.step)
    ↪ ) {
  load_models();
  md5_calc(cb, task, step, 'step_md5_out', 'out', get_step_outputs(task, step),
    ↪ STATE_FINISHED, STATE_ERROR);
}
```

step_md5_down

Calcula los *md5* de todos los ficheros de entrada del paso y los guarda en local con el sufijo *chk*.

```
function step_md5_down(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.step
    ↪ )) {
  load_models();
  md5_calc(cb, task, step, 'step_md5_down', 'chk', get_step_inputs(task, step),
    ↪ STATE_MD5_DOWN_FINISHED, STATE_ERROR);
}
```

step_md5_compare

Compara los *md5* calculados para todos los ficheros de entrada del paso con los *md5* descargados desde los proveedores de origen de los ficheros. Sirve para determinar, a través de *md5_compare* la lista de ficheros necesarios para poder lanzar la ejecución del paso. El paso no se marca como erróneo si los *md5* no coinciden.

```
function step_md5_compare(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.
    ↪ step)) {
  load_models();
  md5_compare(cb, task, step, '', get_step_inputs(task, step), 'step_md5_compare
    ↪ ', STATE_MD5_COMPARE_FINISHED, STATE_ERROR);
}
```

step_md5_verify

Equivalente a *step_md5_compare* pero en este caso el paso se marca como erróneo si los *md5* no coinciden.

```
function step_md5_verify(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.
    ↪ step)) {
  load_models();
  md5_compare(cb, task, step, 'verify', get_step_inputs(task, step), '
    ↪ step_md5_verify', STATE_MD5_VERIFY_FINISHED, STATE_ERROR);
}
```

resolve_cmd

Interpreta la cadena del comando de sistema especificado en el paso en el campo *step.command*. Si este campo no está especificado o está vacío, se entiende que el comando está definido en *cmd_data* y ajusta las opciones de ejecución en consecuencia. Se devuelven el comando a ejecutar y las opciones de ejecución.

```
function resolve_cmd(cb, task, step) {
  let cmd_string, execa_options;
  if (step.command && step.command !== '') {
    execa_options = {};
    let cmd_path = `${PIPELINE_DIR}/${task.pipeline}/${step.command}`;
    if (fs.existsSync(cmd_path)) {
      cmd_string = cmd_path + ' ';
    } else {
      cmd_string = step.command + ' ';
    }
  } else {
    // Command defined completely in cmd_data, that might contain several
    ↪ commands separated by ||, is necessary to execute in a shell
    execa_options = {shell: true};
    cmd_string = '';
```

```
  }
  if (Array.isArray(step.cmd_data)) {
    cmd_string += step.cmd_data.join(' ');
  } else if (step.cmd_data !== undefined) {
    cmd_string += ' ' + step.cmd_data;
  }
  cb();
  let global_subs = {
    ' +': ' ',
    '__DIR__': `${DATA_DIR}/${step.task_id}`,
    '__TASK_DIR__': `${DATA_DIR}/${step.task_id}`,
    '__PIPE_DIR__': `${DATA_DIR}/${task.pipeline}`,
    '__TASK_ID__': step.task_id
  }
  for (let sub in global_subs) {
    cmd_string = cmd_string.replace(new RegExp(sub,'ig'), global_subs[sub]);
  }
  if (step.subs) {
    for (let sub in step.subs) {
      cmd_string = cmd_string.replace(new RegExp(sub,'ig'), step.subs[sub]);
    }
  }
  return [execa_options, cmd_string]
}
```

step_exec

Ejecuta el comando de sistema cuya cadena obtiene de la función *resolve_cmd*. Actualiza los estados de la tarea y paso dependiendo del resultado.

```
async function step_exec(cb, task=JSON.parse(argv.task), step=JSON.parse(argv.
    ↪ step)) {
  let logger = require_dyn("logger_app");
  logger.info(`step_exec ${task.task_id} ${step.step_id}`);
  try {
    load_models();
    let task_id = task.task_id;
    let step_id = step.step_id;
    // Replace tokens in cmd_data
    let [execa_options, cmd_string] = resolve_cmd(none, task, step);
    const {stdout, stderr} = await execa.command(cmd_string, execa_options);
    step_update_ok(cb, logger, stdout, task, step, STATE_EXEC_FINISHED);
  } catch (error) {
    step_update_error(cb, logger, 'step_exec error ' + step.command, error, task
    ↪ , step, STATE_ERROR);
  }
}
```

task_create

Solicita al coordinador *task_provider* la ejecución de un flujo del tipo especificado en *pipeline*. Se indican los pasos de inicio y fin (por defecto *0*, que indica que queremos lanzar ya el primer paso). Utiliza el servicio *plas/pipeline/task_create*. Se informa también el identificador de tarea *task_id*. Si la tarea ya existe, el sistema volverá a ejecutar todos los pasos de la misma entre *step_from* y *step_to*.

```
function task_create(cb, task_provider=argv.task_provider, pipeline=argv.
    ↪ pipeline, task_id=argv.task_id, step_from=argv.step_from, step_to=argv.
    ↪ step_to) {
  let logger = require_dyn("logger_app");
  load_models();
  let task_provider_url = provider_url[task_provider];
  axios.post(`${task_provider_url}/plas/pipeline/task_create`, {
    from: process.env.PLASMIDNODE,
    pipeline: pipeline,
    task_id: task_id,
    step_from: step_from,
    step_to: step_to
  })
    .then(response => {
      logger.info(response.data);
      cb();
    })
    .catch(error => {
      logger.error(`task_create ${error}`);
      cb(error);
    });
}
```

step_do

Solicita al proveedor del paso *task*, *step* la ejecución del mismo.

Al ejecutor del paso se le informan en el cuerpo la tarea y el paso completos y en el campo *from* el identificador del nodo solicitante.

```
function step_do(cb, task=argv.task, step=argv.step) {
  let logger = require_dyn("logger_app");
  load_models();
  let new_state = STATE_STEP_TRANSITIONS[step.state];
  let method = 'step_' + new_state.split('__')[0].toLowerCase();
  step.state = new_state;
  logger.info(`Step ${method} required to provider ${step.provider} ${task.
    ↪ task_id} ${step.step_id}`);
  let task_provider_url = provider_url[step.provider];
  axios.post(`${task_provider_url}/plas/pipeline/${method}`, {
    from : process.env.PLASMIDNODE,
    task : task,
    step : step
```

```
  })
    .then(response => {
      logger.debug(response.data);
      cb();
    })
    .catch(error => {
      logger.error(`gulpfile_pipelines.step_do ${method} ${error}`);
      //logger.error(error);
      cb(error);
    });
}
```

task_info

Solicita al coordinador *task_owner* de la tarea *task_id* toda la información de la misma.

```
function task_info(cb, task_owner=argv.task_provider, task_id=argv.task_id) {
  let logger = require_dyn("logger_app");
  load_models();
  let task_owner_url = provider_url[task_owner];
  axios.post(task_owner_url + '/plas/pipeline/task_info', {
    from: process.env.PLASMIDNODE,
    task_id: task_id
  })
    .then(response => {
      logger.info(response.data);
      cb();
    })
    .catch(error => {
      logger.error(`task_info ${error}`);
      cb(error);
    });
}
```

step_info

Solicita al coordinador *task_owner* de la tarea *task_id* toda la información del paso identificado por *step_id*.

```
function step_info(cb, task_owner=argv.task_provider, task_id=argv.task_id,
    ↪ step_id=argv.step_id) {
  let logger = require_dyn("logger_app");
  load_models();
  let task_owner_url = provider_url[task_owner];
  axios.post(task_owner_url + '/plas/pipeline/step_info', {
    from: process.env.PLASMIDNODE,
    task_id: task_id,
```

```
    step_id: step_id
  })
    .then(response => {
      logger.info(response.data);
      cb();
    })
    .catch(error => {
      logger.error(`step_info ${error}`);
      cb(error);
    });
}
```

task_update

Actualiza el contenido del paso (parámetro *step*) de la tarea *task_id* en el proveedor ejecutor del paso.

El proveedor ejecutor del paso está almacenado en el campo *step.owner*.

```
function task_update(cb, task_id=argv.task_id, step=argv.step) {
  let logger = require_dyn("logger_app");
  load_models();
  let step_owner = step.owner;
  let step_owner_url = provider_url[step_owner];
  axios.post(`${step_owner_url}/plas/pipeline/task_update`, {
    from: process.env.PLASMIDNODE,
    task_id: task_id,
    step: step
  })
    .then(response => {
      logger.debug(response.data);
      cb();
    })
    .catch(error => {
      logger.error(error);
      cb(error);
    });
}
```

step_start

Crea y arranca el paso identificado por *step_id* de la tarea *task*.

El paso se construye a partir de la información maestra del flujo *pipeline* asociado a la tarea *task*. Durante la creación se resuelven el proveedor ejecutor del paso (*assign_provider*) y todos los ejecutores proveedores de ficheros de entrada (*get_step_providers*). Por último se solicita la ejecución del paso al proveedor ejecutor del mismo (*step_do*).

```
async function step_start(cb, task=JSON.parse(argv.task), step_id=argv.step_id)
    ↪ {
  load_models();
  if (pipelines[task.pipeline].steps.length > step_id) {
    let step = {
      task_id: task.task_id,
      step_id: step_id,
      state: STATE_INIT,
      provider: assign_provider(pipelines[task.pipeline].steps[step_id]),
      command: pipelines[task.pipeline].steps[step_id].command,
      cmd_data: pipelines[task.pipeline].steps[step_id].cmd_data,
      subs: pipelines[task.pipeline].steps[step_id].subs,
      outputs: pipelines[task.pipeline].steps[step_id].outputs,
      inputs: pipelines[task.pipeline].steps[step_id].inputs,
      name: pipelines[task.pipeline].steps[step_id].name,
      owner: process.env.PLASMIDNODE || DEFAULT_NODE,
      start_time: new Date().toISOString().replace(/T/, ' '),
      stdout: []
    }
    task.step = step_id;
    step.providers = await get_step_providers(cb, task, step);
    await task_model.update(cb, task, step);
    step_do(cb, task, step);
  } else {
    cb();
  }
}
```

task_continue

Crea y arranca el siguiente paso del flujo para la tarea *task* y el paso ejecutado *step*.

Actualmente el paso a ejecutar es simplemente el siguiente del flujo (*step_id + 1*). En un desarrollo más completo deberían quedar liberados, para su ejecución en paralelo, todos los pasos que presenten todas las dependencias de entrada resueltas.

```
async function task_continue(cb, task=argv.task, step=argv.step) {
  let logger = require_dyn("logger_app");
  try {
    load_models();
    if (step.state === STATE_ERROR) {
      task.state = STATE_ERROR;
    } else if (step.state === STATE_FINISHED) {
      step_id = step.step_id + 1;
      if (task.step_to === -1 || step_id <= task.step_to) {
        step_start(cb, task, step_id);
      }
    } else {
      task.step = step.step_id;
```

```
      await task_model.update(cb, task, step);
      step_do(cb, task, step);
    }
  } catch(error) {
    step_update_error(cb, logger, 'gulpfile_pipelines.task_continue error',
    ↪ error, task, step, STATE_ERROR);
    cb(error);
  }
}
```

task_start

Crea en la base de datos, la tarea de identificador *task_id* y de nodo solicitante (nodo gestor) *task_client* asociándola al flujo especificado (*pipeline*) y definiendo también los pasos de inicio (*step_from*) y fin (*step_to*).

Actualmente el paso a ejecutar es simplemente el siguiente del flujo (*step_id + 1*). En un desarrollo más completo deberían quedar liberados, para su ejecución en paralelo, todos los pasos que presenten todas las dependencias de entrada resueltas.

```
async function task_start(cb, task_id=argv.task_id, task_client=argv.task_client
    ↪ , pipeline=argv.pipeline, step_from=argv.step_from, step_to=argv.step_to)
    ↪  {
  load_models();
  let task = {
    task_id: task_id,
    client: task_client,
    owner: process.env.PLASMIDNODE,
    pipeline: pipeline,
    step: step_from,
    step_from: step_from,
    step_to: step_to,
    state: STATE_TASK_RUNNING
  }
  await task_model.update(cb, task, '');
}

module.exports.task_create = task_create;
module.exports.task_start = task_start;
module.exports.step_do = step_do;
module.exports.task_continue = task_continue;
module.exports.task_info = task_info;
module.exports.step_info = step_info;
module.exports.task_update = task_update;
module.exports.step_exec = step_exec;
module.exports.step_md5_chk = step_md5_chk;
module.exports.step_start = step_start;
module.exports.step_md5_down = step_md5_down;
module.exports.step_md5_out = step_md5_out;
module.exports.step_md5_compare = step_md5_compare;
```

```
module.exports.step_md5_verify = step_md5_verify;
module.exports.step_md5_download = step_md5_download;
module.exports.step_download = step_download;
module.exports.get_step_providers = get_step_providers;
module.exports.expand_path = expand_path;
module.exports.shares = module.exports;
```

B.6.2. pipelines.js

Define las *pipelines* y funciones de transformación que resuelven dependencias recursivas entre las *pipelines*.

```
const {require_dyn} = require('../proxy_modules');
let pipelines = {};
let pipelines_compiled = {};
let inputs = {};
let outputs = {};
let step_ref = {};
global.__pipelines;
const HOST = 'nandus.local';
let provider_url = { default : `http://${HOST}:9091`,
                     plasmid01 : `http://${HOST}:9091`,
                     plasmid02 : `http://${HOST}:9092`,
                     plasmid03 : `http://${HOST}:9093`
                   }
```

Pipelines de ejemplo: con dos pasos, con un paso y dos casos de *pipelines* vacías. El significado de los diferentes campos está documentado en el diseño de alto nivel.

```
pipelines['test'] = {
  desc : "Obtain modules for a plasmid set",
  steps : [
    {name: 'plas01_test', command : 'plas01_test.pl', providers : ['plasmid01'
    ↪ ]}, //0
    {name: 'plas02_test', command : 'plas02_test.pl', providers : ['plasmid02']}
  ]
}

pipelines['test_simple'] = {
  desc : "Test pipeline",
  steps : [
    {name: 'plas03_test', command : 'plas03_test.pl', providers : ['plasmid01']}
    ↪  //0
  ]
}

outputs['plas01_test'] = ["test01.txt"];
inputs['plas01_test'] = [];
```

```
inputs['plas02_test'] = ["test01.txt"];
inputs['plas03_test'] = ["centos7.vdi", "test01.txt", "centos7_bis.vdi", "test.
    ↪ jpg"];

pipelines['KO01'] = {
  desc : "Pipeline without steps",
  steps : []
}

pipelines['KO02'] = {
  desc : "Pipeline without steps 2"
}
```

Pipeline de carga de la base de datos de módulos funcionales de plásmidos.

```
pipelines['plasmid_modules'] = {
  desc : "Plasmid modules pipeline",
  steps : [
    {
      name: 'plas00', command: 'plas00_plasmid_gi__PLSDB_extract.tcsh',
      desc: 'Extract tsv of plamids from file downloaded from PLSDB',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__',
                  '30', //plasmid_number
                  '2018_09_14'] //plsdb_version
    },
    {
      name: 'plas01', command: 'plas01_plasmid_protein__NCBI_download.pl',
      desc: 'Obtain from NCBI the sequences of proteins related to plasmids',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    {
      name: 'plas02', command: 'plas02_plasmid_nucleotide__NCBI_download.pl',
      desc: 'Obtain from NCBI the nucleotidic sequences related to plasmids',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    {
      name: 'plas03', command: 'plas03_plasmid_nucleotide__DFAST_input.pl',
      desc: 'Obtain from NCBI the nucleotidic sequences related to plasmids',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    {
      name: 'plas04_dfast', command: 'dfast',
      desc: 'DFAST annotation',
      providers: ['plasmid02'],
      cmd_data: '--genome __DIR__/plasmid_nucleotide__DFAST_input.fasta --out
    ↪ __DIR__/DFAST_OUT --force'
    },
```

```
    {
      name: 'plas05', command: 'plas05_plasmid_protein__DFAST_output.pl',
      desc: 'DFAST output to fasta',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    {
      name: 'plas06', command: 'plas06_plasmid_protein__NCBI_DFAST_combined.pl',
      desc: 'Combined NCBI and DFAST annotations',
      providers : ['plasmid01'],
      cmd_data : ['__TASK_ID__']
    },
    {
      pipeline: 'mmseq2_create_cluster'
    },
    {//step 11 begins
      pipeline: 'mmseq2_update_cluster'
    }
  ]
}

pipelines['mmseq2_create_cluster'] = {
  desc : "MMSEQ2 create cluster",
  steps : [
    {//7
      name: 'make_pipeline_dir', command: '',
      desc: 'make pipeline directory in data',
      providers: ['plasmid03'],
      inputs: ['plasmid_protein__combined.fasta'],
      outputs: ['plasmid_protein__trimmed.fasta'],
      cmd_data: `mkdir __PIPE_DIR__
                 awk '/^>/{seqCount++;} {if (seqCount <= 1000) {print $0;}} \
                 ' __DIR__/plasmid_protein__combined.fasta > __DIR__/
    ↪ plasmid_protein__trimmed.fasta`

    },
    {//8
      name: 'plas07_mmseqs__createdb', command: 'mmseqs',
      desc: 'MMSEQ2 createdb',
      providers: ['plasmid03'],
      //cmd_data: 'createdb __DIR__/plasmid_protein__combined.fasta __PIPE_DIR__
    ↪ /plasmid_protein.DB --max-seq-len 2600000',
      cmd_data: 'createdb __DIR__/plasmid_protein__trimmed.fasta __PIPE_DIR__/
    ↪ plasmid_protein.DB --max-seq-len 2600000',
      inputs: ['plasmid_protein__trimmed.fasta'],
      outputs: ['__PIPE_DIR__/plasmid_protein.DB']
    },
    {//9
      name: 'plas08_mmseqs__create_cluster', command: '',
      desc: 'MMSEQ2 clustering',
      providers: ['plasmid03'],
```

```
      inputs: ['__PIPE_DIR__/plasmid_protein.DB'],
      outputs: ['__PIPE_DIR__/plasmid_protein.CLU'],
      cmd_data: `rm __PIPE_DIR__/plasmid_protein.CLU*
                 mmseqs cluster __PIPE_DIR__/plasmid_protein.DB \
                 __PIPE_DIR__/plasmid_protein.CLU __PIPE_DIR__/tmp \
                 -c 0.8 --min-seq-id 0.7 --threads 1 --max-seq-len 2600000`
    },
    {//10
      name: 'plas09_mmseqs__create_tsv', command: 'mmseqs',
      desc: 'MMSEQ2 tsv generation',
      providers: ['plasmid03'],
      inputs: ['__PIPE_DIR__/plasmid_protein.DB','__PIPE_DIR__/plasmid_protein.
    ↪ CLU'],
      outputs: ['__PIPE_DIR__/plasmid_protein.CLU.tsv'],
      cmd_data: 'createtsv __PIPE_DIR__/plasmid_protein.DB __PIPE_DIR__/
    ↪ plasmid_protein.DB __PIPE_DIR__/plasmid_protein.CLU __PIPE_DIR__/
    ↪ plasmid_protein.CLU.tsv'
    }
  ]
}

pipelines['mmseq2_update_cluster'] = {
  desc : "MMSEQ2 update cluster",
  steps : [
    {//11
      name: 'plas11_mmseqs__createdb',
      command: 'mmseqs',
      cmd_data: 'createdb __DIR__/plasmid_protein__combined.fasta __DIR__/
    ↪ plasmid_protein_ini.DB --max-seq-len __max-seq-len__',
      inputs: ['plasmid_protein__combined.fasta'],
      outputs: ['plasmid_protein_ini.DB'],
      providers: ['plasmid03'],
      subs: {'__max-seq-len__': 2600000}
    },
    {//12
      name: 'plas12_mmseqs__update_cluster', command: '',
      desc: 'MMSEQ2 update cluster',
      providers: ['plasmid03'],
      cmd_data: `
        pwd
        whoami
        rm  __TASK_DIR__/plasmid_protein_new.*
        rm -rf __TASK_DIR__/tmp
        mmseqs clusterupdate \
        __PIPE_DIR__/plasmid_protein.DB \
        __TASK_DIR__/plasmid_protein_ini.DB \
        __PIPE_DIR__/plasmid_protein.CLU \
        __TASK_DIR__/plasmid_protein_new.DB \
        __TASK_DIR__/plasmid_protein_new.CLU \
        __TASK_DIR__/tmp -v 2 -c 0.8 --min-seq-id 0.7 --threads 1 --max-seq-len
    ↪ 2600000`,
```

```
      inputs: ['__PIPE_DIR__/plasmid_protein.DB',
               'plasmid_protein_ini.DB',
               '__PIPE_DIR__/plasmid_protein.CLU'],
      outputs: ['plasmid_protein_new.DB', 'plasmid_protein_new.CLU']
    },
    {//13
      name: 'plas13_mmseqs__create_tsv', command: 'mmseqs',
      desc: 'MMSEQ2 tsv generation',
      providers: ['plasmid03'],
      cmd_data: `createtsv __TASK_DIR__/__DB_NAME__.DB \
                 __TASK_DIR__/__DB_NAME__.DB \
                 __TASK_DIR__/__DB_NAME__.DB_CLU \
                 __TASK_DIR__/__DB_NAME__.DB_CLU.tsv`,
      inputs: ['plasmid_protein_new.DB', 'plasmid_protein_new.DB_CLU'],
      outputs: ['plasmid_protein_new.DB_CLU.tsv'],
      subs: {'__DB_NAME__': 'plasmid_protein_new'}
    }
  ]
}

outputs['plas00'] = ['plasmid_GI.tsv'];
inputs['plas01'] = ['plasmid_GI.tsv'];
outputs['plas01'] = ['plasmid_protein.fasta'];
inputs['plas02'] = ['plasmid_GI.tsv'];
outputs['plas02'] = ['plasmid_nucleotide.fasta'];
inputs['plas03'] = ['plasmid_protein.fasta', 'plasmid_nucleotide.fasta'];
outputs['plas03'] = ['plasmid_nucleotide__DFAST_input.fasta'];
inputs['plas04_dfast'] = ['plasmid_nucleotide__DFAST_input.fasta'];
outputs['plas04_dfast'] = ['DFAST_OUT/genome.gbk', 'DFAST_OUT/protein.faa'];
inputs['plas05'] = outputs['plas04_dfast'];
outputs['plas05'] = ['plasmid_protein__DFAST.fasta'];
inputs['plas06'] = ['plasmid_protein__DFAST.fasta', 'plasmid_protein.fasta', '
    ↪ plasmid_nucleotide.fasta']
outputs['plas06'] = ['plasmid_protein__combined.fasta'];
inputs['plas07_mmseqs__createdb'] = ['plasmid_protein__combined.fasta'];
outputs['plas07_mmseqs__createdb'] = ['__PIPE_DIR__/plasmid_protein.DB'];
inputs['plas08_mmseqs__create_cluster'] = outputs['plas07_mmseqs__createdb'];
outputs['plas08_mmseqs__create_cluster'] = ['__PIPE_DIR__/plasmid_protein.DB_CLU
    ↪ .index'];
inputs['plas09_mmseqs__create_tsv'] = ['__PIPE_DIR__/plasmid_protein.DB_CLU.
    ↪ index', '__PIPE_DIR__/plasmid_protein.DB'];
outputs['plas09_mmseqs__create_tsv'] = ['__PIPE_DIR__/plasmid_protein.DB_CLU.tsv
    ↪ '];
```

Las siguientes funciones implementan el preproceso de la información de *pipelines* y generan una nueva estructura *pipelines_compiled* que será la utilizada por el orquestador. En estas funciones se despliegan las referencias de *pipelines* dentro de *pipelines* y se mezclan los *steps* para los que se han declarado varias versiones.

Este preproceso se podía haber diferido a la orquestación, pero de esta forma es más fácil de depurar y además disminuimos la complejidad de la propia orquestación.

merge_step_with_reference

Esta función mezcla dos versiones de un paso, la nueva sustituye en la antigua los valores comunes y se mantienen los valores antiguos que no han sido modificados. Es una implementación de un mecanismo de herencia y está orientado a la reutilización: un paso puede definirse declarando sólo las diferencias con un paso previamente definido.

```
function merge_step_with_reference(step_new) {
  let step = {};
  //console.log("STEP_NEW", step_new);
  let step_old = step_ref[step_new.name];
  if (step_old) {
    for (let key in step_old) {
      step[key] = step_old[key];
    }
  }
  if (step_new) {
    // Override with the new fields
    for (let key in step_new) {
      step[key] = step_new[key];
    }
  }
  if (!step.command && !step.cmd_data) {
    throw new Error('Step without required key: "command" or "cmd_data" are
    ↪ mandatory');
  }
  return step;
}
```

resolve_step_references

Genera un nuevo *step* en la estructura global *step_ref* tras mezclar dos versiones del mismo.

```
function resolve_step_references(step) {
  let new_step = {};
  if (step.name) {
    new_step = merge_step_with_reference(step);
    step_ref[step.name] = new_step;
  } else {
    throw new Error('Required step key: "name"');
  }
  return new_step;
}
```

get_steps

Función recursiva que sustituye en una *pipeline* que se pasa por parámetro las *pipelines* referenciadas por sus pasos correspondientes. El resultado es una *pipeline* sin referencias donde

todos los pasos son explícitos.

```
function get_steps(pipeline) {
  let steps = [];
  if (pipelines[pipeline].steps) {
    for (let step of pipelines[pipeline].steps) {
      if (step.pipeline) {
        if (!pipelines[step.pipeline]) throw new Error('Unreferenced pipeline');
        // if there is a compiled version of included pipeline we take the
    ↪ compiled version
        // because it has included pipelines resolved
        let new_steps = get_steps(step.pipeline);
        for (let step_new of new_steps) {
          steps.push(resolve_step_references(step_new));
        }
      } else {
        steps.push(resolve_step_references(step));
      }
    }
  }
  return steps;
}
```

compile_pipelines_recursive

Función principal que obtiene la estructura de *pipelines* final para uso del sistema de orquestación.

```
function compile_pipelines_recursive() {
  for (pipeline in pipelines) {
    pipelines_compiled[pipeline] = {desc: pipelines[pipeline].desc, steps:
    ↪ get_steps(pipeline)};
  }
}
```

Función implícita *main*. Si falla la compilación de las *pipelines* mantenemos la última versión.

Este programa *pipelines.js* es posible cargarlo dinámicamente en el servidor sin reiniciarlo. Por tanto es factible modificar en caliente la definición de una *pipeline*. Como el proceso entraña cierto riesgo, en caso de error de compilación , el sistema conserva la versión anterior del módulo.

```
try {
  compile_pipelines_recursive();
} catch(error) {
  require_dyn("logger_app").error(error);
  throw new Error('Pipelines not compiled, last compiled version is in use');
}
```

La versión compilada se exporta como pipelines.

```
module.exports.pipelines = pipelines_compiled;
module.exports.inputs = inputs;
module.exports.outputs = outputs;
module.exports.provider_url = provider_url;
```

B.6.3. gulpfile_task_model_sqlite.js

Módulo de tareas *gulp* sobre la base de datos de flujos de tareas de usuario. Desarrollada sobre la base de datos *sqlite*.

Incluye las operaciones de creación del modelo de datos y todas las actualizaciones sobre las diferentes tablas.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const sqlite = require('better-sqlite3');
const fs = require('fs');

const none = function(){};
const argv = require('yargs')
  .default('step', {task_id : 'test', step_id: "2", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]})
  .default('task', {task_id : 'test', pipeline: "plasmid_modules"})
  .default('task_id', 'test')
  .default('step_id', '0')
  .argv;

global.__sqlite_db;
```

sqlite_upsert_task

Actualiza o inserta el registro *task* en la tabla *tasks* conforme a su clave *task_id*.

```
function sqlite_upsert_task(cb, task=argv.task) {
  let stmt = __sqlite_db.prepare(`
    INSERT INTO tasks VALUES(?, ?, ?)
    ON CONFLICT(task_id) DO UPDATE SET
    timestamp = excluded.timestamp,
    task = excluded.task `);
  stmt.run(task.task_id, (new Date()).getTime(), JSON.stringify(task));
  cb();
}
```

sqlite_upsert_step

Actualiza o inserta el registro *step* en la tabla *steps* conforme a su clave *task_id* + *step_id*.

```
function sqlite_upsert_step(cb, step=argv.step) {
  let stmt = __sqlite_db.prepare(`
    INSERT INTO steps VALUES(?, ?, ?, ?)
    ON CONFLICT(task_id, step_id) DO UPDATE SET
    timestamp = excluded.timestamp,
    step = excluded.step `);
  stmt.run(step.task_id, step.step_id, (new Date()).getTime(), JSON.stringify(
    ↪ step));
  cb();
}
```

sqlite_start

Arranca la base de datos *sqlite* de tareas de usuario y crea las tablas si no existen.

El modelo contiene las tablas *tasks*, *steps* y *log*.

```
function sqlite_start(cb) {
  let logger = require_dyn("logger_app");
  __sqlite_db = new sqlite('./DATA/task_model_sqlite.db', { verbose: logger.
    ↪ debug });
  let sql_create_model =
    `CREATE TABLE IF NOT EXISTS tasks(
      task_id text PRIMARY KEY,
      timestamp INTEGER,
      task text NOT NULL);
      CREATE UNIQUE INDEX IF NOT EXISTS task_id
      ON tasks(task_id);

      CREATE TABLE IF NOT EXISTS steps(
        task_id text,
        step_id text,
        timestamp INTEGER,
        step text NOT NULL,
        PRIMARY KEY(task_id, step_id),
        FOREIGN KEY(task_id)
        REFERENCES tasks(task_id)
         ON DELETE CASCADE
         ON UPDATE NO ACTION);
      CREATE UNIQUE INDEX IF NOT EXISTS step_id
      ON steps(task_id, step_id);

      CREATE TABLE IF NOT EXISTS log(
        log_id TEXT,
        provider TEXT,
        timestamp INTEGER,
        method TEXT);
```

```
  `
  let ret = __sqlite_db.exec(sql_create_model);
  cb();
}
```

sqlite_update

Upsert combinado de la tarea *task* y el paso *step* pasados por parámetro. Pueden llegar vacíos.

```
function sqlite_update(cb, task=argv.task, step=argv.step) {
  if (task && task !== '') {
    sqlite_upsert_task(none, task);
  }
  if (step && step !== '') {
    sqlite_upsert_step(none, step);
  }
  cb();
}
```

sqlite_exists_task

Devuelve *true* si la tarea identificada por *task_id* existe en la base de datos y *false* en caso contrario.

```
function sqlite_exists_task(cb, task_id=argv.task_id) {
  let task_exists;
  if (sqlite_get_task(none, task_id)) task_exists = true;
  else task_exists = false;
  return task_exists;
  cb();
}
```

sqlite_get_tasks

Devuelve en una matriz todas las tareas de la base de datos.

```
function sqlite_get_tasks(cb) {
  let stmt = __sqlite_db.prepare("SELECT task FROM tasks");
  let tasks = [];
  for (let task of stmt.pluck().iterate()) {
    tasks.push(JSON.parse(task));
  }
  return tasks;
  cb();
```

```
}
```

sqlite_get_steps_all

Devuelve en una matriz todos los *steps* de la base de datos.

```
function sqlite_get_steps_all(cb) {
  let stmt = __sqlite_db.prepare("SELECT step FROM steps");
  let steps = [];
  for (let step of stmt.pluck().iterate()) {
    steps.push(JSON.parse(step));
  }
  return steps;
  cb();
}
```

sqlite_get_task

Devuelve la tarea cuyo identificador *task_id* se pasa por parámetro.

```
function sqlite_get_task(cb, task_id=argv.task_id) {
  let stmt = __sqlite_db.prepare("SELECT task FROM tasks WHERE task_id = ?");
  let task = stmt.pluck().get(task_id);
  return task ? JSON.parse(task) : {};
  cb();
}
```

sqlite_get_steps

Devuelve en una matriz todos los *steps* de la tarea cuyo identificador *task_id* se pasa por parámetro.

```
function sqlite_get_steps(cb, task_id=argv.task_id) {
  let stmt = __sqlite_db.prepare("SELECT step FROM steps WHERE task_id = ?");
  let steps = [];
  for (let step of stmt.pluck().iterate()) {
    steps.push(JSON.parse(step));
  }
  return steps;
  cb();
}
```

sqlite_get_step

Devuelve el step cuyo identificador *task_id* + *step_id* se pasa por parámetro.

```
function sqlite_get_step(cb, task_id=argv.task_id, step_id=argv.step_id) {
  let stmt = __sqlite_db.prepare("SELECT step FROM steps WHERE task_id = ? AND
    ↪ step_id = ?");
  let step = stmt.pluck().get(task_id, step_id);
  return step ? JSON.parse(step) : {};
  cb();
}
```

sqlite_test

Función para verificar la creación de la base de datos y las operaciones de obtención de *tasks* y *steps*.

```
async function sqlite_test(cb) {
  let logger = require_dyn("logger_app");
  let step = {task_id : 'test5', step_id: "3", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]};
  let task = {task_id : 'test5', pipeline: "plasmid_modules"};
  __sqlite_db = new sqlite('./DATA/task_model_sqlite.db', { verbose: logger.
    ↪ debug });
  let sql_create_model =
    `CREATE TABLE IF NOT EXISTS tasks (
      task_id text PRIMARY KEY,
      timestamp INTEGER,
      task text NOT NULL);
      CREATE TABLE IF NOT EXISTS steps (
        task_id text,
        step_id text,
        timestamp INTEGER,
        step text NOT NULL,
        PRIMARY KEY(task_id, step_id),
        FOREIGN KEY(task_id)
        REFERENCES task (task_id)
         ON DELETE CASCADE
         ON UPDATE NO ACTION);
  `;
  __sqlite_db.exec(sql_create_model);
  console.log(sqlite_get_task(none, 'test_6'));
  console.log(sqlite_get_task(none, 'test_5'));
  cb();
}
```

sqlite_test_upserts

Función para verificar las funciones de *upserts*

```
function sqlite_test_upserts(cb) {
  let step = {task_id : 'test', step_id: "3", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]};
  let task = {task_id : 'test', pipeline: "plasmid_modules"};
  sqlite_upsert_task(none, task);
  sqlite_upsert_task(none, task);
  sqlite_upsert_task(none, task);
  sqlite_upsert_step(none, step);
  sqlite_upsert_step(none, step);
  console.log("TASKS ", sqlite_get_tasks(none));
  console.log("STEPS ", sqlite_get_steps_all(none));
  cb();
}
```

En las exportaciones los nombres de las funciones se renombran a los nombres comunes utilizados por los diferentes modelos de base de datos.

De esta forma todos los módulos que implementan el modelo de tareas son intercambiables.

```
module.exports.start = sqlite_start;
module.exports.upsert_task = sqlite_upsert_task;
module.exports.upsert_step = sqlite_upsert_step;
module.exports.update = sqlite_update;
module.exports.get_tasks = sqlite_get_tasks;
module.exports.get_task = sqlite_get_task;
module.exports.get_steps_all = sqlite_get_steps_all;
module.exports.get_step = sqlite_get_step;
module.exports.get_steps = sqlite_get_steps;
module.exports.exists_task = sqlite_exists_task;
```

Funciones de pruebas.

```
module.exports.get_steps_test = series(sqlite_start, sqlite_get_steps);
module.exports.sqlite_test_upsert = series(sqlite_start, sqlite_test_upserts);
module.exports.sqlite_testing = series(sqlite_start, sqlite_test);
module.exports.sqlite_test = sqlite_test;
module.exports.sqlite_test_exist_task = series(sqlite_start, sqlite_exists_task,
    ↪  sqlite_get_tasks);

module.exports.shares = module.exports;
```

B.6.4. gulpfile_task_model_loki.js

Módulo de tareas de base de datos de flujos basado en *LokiJS* [36].

El *API* es equivalente al definido para *sqlite*, por lo tanto sirve de referencia lo documentado para este modelo.

Estas son las diferencias más importantes:

1) Los objetos de base de datos son objetos *javascript* que contienen todos los campos, no se definen las claves como en *sqlite*, donde tenemos el contenido de tareas y pasos serializados en *json* al igual que en *loki*, además de los campos de clave *task_id* y *step_id*.
2) La base de datos *loki* trabaja en memoria, con persistencia a disco a intervalos programables. La persistencia se especifica en arranque (*autosaveInterval*, *autosave*).
3) Las búsquedas no utilizan SQL sino una sintaxis compatible con *MongoDB*.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const loki = require("lokijs");
const lfsa = require('lokijs/src/loki-fs-structured-adapter.js');
const fs = require('fs');
const redis = require('redis');
const none = function(){};
const argv = require('yargs')
  .default('state', 'none')
  .default('autosave', 'Y')
  .default('autosaveInterval', 6000)
  .default('step', {task_id : 'test', step_id: "2", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]})
  .default('task', {task_id : 'test', pipeline: "plasmid_modules"})
  .argv;

global.__task_model;
global.__loki_tasks;
global.__loki_steps;

function loki_upsert_task(cb, task) {
  let task_old = __loki_tasks.findOne({task_id: task.task_id});
  if (task_old) {
    let task_new = task_old;
    for (key in task) {
      task_new[key] = task[key];
    }
    __loki_tasks.update(task_new);
  } else {
    __loki_tasks.insert(task);
  }
  cb();
}

function loki_upsert_step(cb, step=argv.step) {
  let step_old = __loki_steps.findOne({task_id : step.task_id, step_id: step.
    ↪ step_id});
  if (step_old) {
    let step_new = step_old;
```

```
    for (key in step) {
      step_new[key] = step[key];
    }
    __loki_steps.update(step_new);
  } else {
    __loki_steps.insert(step);
  }
  cb();
}

function loki_start(cb, autosave=argv.autosave, autosaveInterval=argv.
    ↪ autosaveInterval) {
    let adapter = new lfsa();
    let loki_options =  {
      adapter: adapter,
      autoloadCallback : databaseInitialize,
      autoload: true,
      autosave: true,
      autosaveInterval: parseInt(autosaveInterval) || 6000
    }
    if (autosave === 'N') {
      loki_options.autosave = false;
    }
    __task_model = new loki('./DATA/task_model_loki.db', loki_options);

  function databaseInitialize() {
    let logger = require_dyn("logger_app");
    __loki_tasks = __task_model.getCollection("tasks") ||
                        __task_model.addCollection("tasks", { indices: ['task_id'
    ↪ ] });
    __loki_steps = __task_model.getCollection("steps") ||
                        __task_model.addCollection("steps", { indices: ['task_id'
    ↪ , 'step_id'] });
    logger.info("Loki DB ready");
    logger.info(loki_options);
    cb();
  }
}

function loki_test_inserts(cb) {
  console.log("TASKS ", loki_get_tasks(none));
  console.log("TASKS ", __loki_tasks.data);
  __loki_tasks.insert({task_id : 'test', pipeline: "plasmid_modules"});
  __loki_tasks.insert({task_id : 'test', pipeline: "plasmid_modules"});
  __loki_tasks.insert({task_id : 'test', pipeline: "plasmid_modules"});
  console.log("TASKS ", loki_get_tasks(none));
  cb();
}

async function loki_test_upserts(cb) {
  console.log("TASKS ", loki_get_tasks(none));
```

```
  let step = {task_id : 'test', step_id: "3", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]};
  let task = {task_id : 'test', pipeline: "plasmid_modules"};
  loki_upsert_task(none, task);
  loki_upsert_task(none, task);
  loki_upsert_task(none, task);
  loki_upsert_task(none, task);
  loki_upsert_step(none, step);
  loki_upsert_step(none, step);
  loki_update(none, task, step);
  console.log("TASKS ", loki_get_tasks(none));
  console.log("STEPS ", loki_get_steps_all(none));
  cb();
}

function loki_update(cb, task=argv.task, step=argv.step) {
  if (task && task !== '') {
    loki_upsert_task(none, task);
  }
  if (step && step !== '') {
    loki_upsert_step(none, step);
  }
  cb();
}

function loki_exists_task(cb, task_id=argv.task_id) {
  let task_exists;
  if (__loki_tasks.findOne({task_id : task_id})) task_exists = true;
  else task_exists = false;
  cb();
  return task_exists;
}

function loki_get_tasks(cb) {
  cb();
  return __loki_tasks.find();
}

function loki_get_steps_all(cb) {
  cb();
  return __loki_steps.find();
}

function loki_get_task(cb, task_id=argv.task_id) {
  cb();
  return __loki_tasks.findOne({task_id : task_id}) || {};
}

function loki_get_steps(cb, task_id=argv.task_id) {
  cb();
```

```
  return __loki_steps.find({task_id : task_id}) || {};
}

function loki_get_step(cb, task_id=argv.task_id, step_id=argv.step_id) {
  cb();
  return __loki_steps.findOne({task_id : task_id, step_id: step_id}) || {};
}
```

Funciones del API.

```
module.exports.start = loki_start;
module.exports.upsert_task = loki_upsert_task;
module.exports.upsert_step = loki_upsert_step;
module.exports.update = loki_update;
module.exports.get_tasks = loki_get_tasks;
module.exports.get_task = loki_get_task;
module.exports.get_steps_all = loki_get_steps_all;
module.exports.get_step = loki_get_step;
module.exports.get_steps = loki_get_steps;
module.exports.exists_task = loki_exists_task;
```

Funciones de pruebas.

```
module.exports.get_steps_test = series(loki_start, loki_get_steps);
module.exports.loki_test = series(loki_start, loki_test_inserts);
module.exports.loki_test_upsert = series(loki_start, loki_test_upserts);
module.exports.loki_test_inserts = loki_test_inserts;
module.exports.loki_test_upserts = loki_test_upserts;
module.exports.loki_test_exist_task = series(loki_start, loki_exists_task,
    ↪ loki_get_tasks);
module.exports.shares = module.exports;
```

B.6.5. gulpfile_task_model_redis.js

Módulo de tareas de base de datos de flujos basado en *redis* [11].

El *API* es equivalente al definido para *sqlite* y *loki*, por lo tanto sirve de referencia lo documentado para este modelo.

La implementación es más próxima al modelo de *loki*, diferenciándose en los siguientes aspectos:

1) La base de datos no está empotrada como en *loki*, es publicada por un servidor externo que expone un puerto configurable. Contamos con una versión de este servidor en unna imagen *docker*.
2) A diferencia de *loki* y *sqlite* todos los accesos son asíncronos. Para compatibilizar el *API* hemos utilizado la utilidad *promisify* de *nodejs* para convertirlos en versiones síncronas.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const fs = require('fs');
const redis = require('redis');
const {promisify} = require('util');
const none = function(){};
const argv = require('yargs')
  .default('state', 'none')
  .default('autosave', 'Y')
  .default('autosaveInterval', 6000)
  .default('step', {task_id : 'test', step_id: "2", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]})
  .default('task', {task_id : 'test', pipeline: "plasmid_modules"})
  .argv;
const HOST = 'macdoide-2.local';

global.__redis_cli;
global.__redis_set;
global.__redis_get;
global.__redis_exists;
global.__redis_keys;

async function redis_upsert_task(cb, task) {
  await __redis_set('task_' + task.task_id, JSON.stringify(task));
  cb();
}

async function redis_upsert_step(cb, step=argv.step) {
  await __redis_set('step_' + step.task_id + '_' + step.step_id, JSON.stringify(
    ↪ step));
  cb();
}

function redis_start(cb) {
  let logger = require_dyn("logger_app");
  __redis_cli  = redis.createClient({
    port      : 6379,               // replace with your port
    host      : HOST,        // replace with your hostanme or IP address
    //password  : ''   // replace with your password
    // optional, if using SSL
    // use `fs.readFile[Sync]` or another method to bring these values in
    // tls       : {
    //   key  : stringValueOfKeyFile,
    //   cert : stringValueOfCertFile,
    //   ca   : [ stringValueOfCaCertFile ]
  })
  .on('connect', function() {
    console.log('Connected to Redis Server');
    __redis_get = promisify(__redis_cli.get).bind(__redis_cli);
```

```
    __redis_set = promisify(__redis_cli.set).bind(__redis_cli);
    __redis_exists = promisify(__redis_cli.exists).bind(__redis_cli);
    __redis_keys = promisify(__redis_cli.keys).bind(__redis_cli);
    console.log('Client ready');
    cb();
  })
  .on('error', function (err) {
    logger.error("gulpfile_task_model");
    logger.error(err);
    cb();
  });
}

async function redis_test_upserts(cb) {
  console.log("TASKS ", redis_get_tasks(none));
  let step = {task_id : 'test', step_id: "3", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]};
  let task = {task_id : 'test', pipeline: "plasmid_modules"};
  await redis_upsert_task(none, task);
  redis_upsert_task(none, task);
  redis_upsert_task(none, task);
  redis_upsert_task(none, task);
  redis_upsert_step(none, step);
  redis_upsert_step(none, step);
  redis_update(none, task, step);
  console.log("TASKS ", await redis_get_tasks(none));
  console.log("STEPS ", await redis_get_steps_all(none));
  cb();
}

function redis_update(cb, task=argv.task, step=argv.step) {
  if (task && task !== '') {
    redis_upsert_task(none, task);
  }
  if (step && step !== '') {
    redis_upsert_step(none, step);
  }
  cb();
}

async function redis_exists_task(cb, task_id=argv.task_id) {
  let task_exists;
  if (await __redis_exists('task_' + task_id) === 1) task_exists = true;
  else task_exists = false;
  return task_exists;
  cb();
}

async function redis_get_tasks(cb) {
  let task_keys = await __redis_keys('*task*');
```

```
  let tasks = [];
  for (key of task_keys) {
    tasks.push(JSON.parse(await __redis_get(key)));
  }
  return tasks;
  cb();
}

async function redis_get_steps_all(cb) {
  let step_keys = await __redis_keys('*step*');
  let steps = [];
  for (key of step_keys) {
    steps.push(JSON.parse(await __redis_get(key)));
  }
  return steps;
  cb();
}

async function redis_get_task(cb, task_id=argv.task_id) {
  return JSON.parse(await __redis_get('task_' + task_id)) || {};
  cb();
}

async function redis_get_steps(cb, task_id=argv.task_id) {
  let step_keys = await __redis_keys('*step_' + task_id + '*');
  let steps = [];
  for (key of step_keys) {
    steps.push(JSON.parse(await __redis_get(key)));
  }
  return steps;
  cb();
}

async function redis_get_step(cb, task_id=argv.task_id, step_id=argv.step_id) {
  return JSON.parse(await __redis_get('step_' + task_id + '_' + step_id)) || {};
  cb();
}

async function redis_test(cb) {
  let step = {task_id : 'test5', step_id: "3", "command": "
    ↪ plas02_plasmid_nucleotide__NCBI_download.pl", "providers" : {"plasmid_GI.
    ↪ tsv" : {"provider" : "plasmid01", "step_id" : "0"}} , "stdout":[]};
  let task = {task_id : 'test5', pipeline: "plasmid_modules"};
  await redis_upsert_task(none, task);
  await redis_upsert_task(none, task);
  await redis_upsert_task(none, task);
  await redis_upsert_task(none, task);
  await redis_upsert_step(none, step);
  const task_new = await redis_get_task(none, task.task_id);
  console.log(task_new);
  const step_new = await redis_get_step(none, step.task_id, step.step_id);
```

```
  console.log(step_new);
  console.log(await redis_get_tasks(none));
  console.log(await redis_get_steps_all(none));
  console.log(await redis_get_steps(none, task.task_id));
  console.log('Quit redis');
  await console.log("a");
  __redis_cli.quit();
  cb();
}
```

Funciones del API.

```
module.exports.start = redis_start;
module.exports.upsert_task = redis_upsert_task;
module.exports.upsert_step = redis_upsert_step;
module.exports.update = redis_update;
module.exports.get_tasks = redis_get_tasks;
module.exports.get_task = redis_get_task;
module.exports.get_steps_all = redis_get_steps_all;
module.exports.get_step = redis_get_step;
module.exports.get_steps = redis_get_steps;
module.exports.exists_task = redis_exists_task;
```

Funciones de pruebas.

```
module.exports.get_steps_test = series(redis_start, redis_get_steps);
module.exports.redis_test_upserts = redis_test_upserts;
module.exports.redis_test_upsert = series(redis_start, redis_test_upserts);
module.exports.redis_testing = series(redis_start, redis_test);
module.exports.redis_test = redis_test;
module.exports.redis_test_exist_task = series(redis_start, redis_exists_task,
    ↪ redis_get_tasks);
module.exports.shares = module.exports;
```

B.6.6. router_pipelines.js

Módulo de servicios *REST* relacionados con la orquestación de flujos.

Todas las funciones asociadas son asíncronas porque deben utilizar *await* contra el *API* del modelo de datos.

```
const {require_dyn} = require('../proxy_modules');
const httpContext = require('express-http-context');
const express = require('express');
const uuidv4 = require('uuid');
const none = function(){};
const ENV = process.env.ENV || 'dev';
const router_pipelines = express.Router();
```

```
let logger, pipelines, gulpfile_pipelines, task_model;

router_pipelines.use(require_dyn("logger_express").info);
```

resolve_dyns

Recarga los módulos dinámicos implicados a través de *proxy_modules*. La necesidad de recarga la decide *require_dyn* de *proxy_modules*. Se comprueba una vez en cada *request*.

```
router_pipelines.use(function resolve_dyns(req, res, next) {
  logger = require_dyn("logger_app");
  logger.info("router_pipelines URL: " + req.url);
  gulpfile_pipelines = require_dyn("gulpfile_pipelines");
  task_model = require_dyn("gulpfile_task_model_" + __model);
  let pipeline_model = require_dyn("pipelines");
  pipelines = pipeline_model.pipelines;
  next();
});
```

Servicio /plas/pipeline/task_model

http: *GET*

Devuelve toda la estructura de tareas (servicio habilitado para el entorno de desarrollo).

```
ENV === 'dev' && router_pipelines.get('^/plas/pipeline/task_model', async
    ↪ function (req, res) {
  try {
    res.json({tasks: await task_model.get_tasks(none), steps: await task_model.
    ↪ get_steps_all(none)});
  } catch(error) {
    errorcode = "PIPE0008";
    message = `get.task_model internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio /plas/pipeline/task_create

http: *POST*

Crea una tarea o rearranca una tarea existente a partir de los siguientes datos del cuerpo de la solicitud *request.body*:

- ***pipeline*** identificador del flujo asociado.
- ***step_from*** índice que identifica el *step* de inicio (0, en tareas nuevas).
- ***step_to*** índice que identifica el *step* de final de ejecución.
- ***task_id*** identificador de tarea de usuario.

Si la tarea es nueva (se envía con *task_id = 0* se le asigna un identificador único. Tras la creación/arranque de la tarea (*task_start*), se crea el primer *step* y se solicita al *provider* (*step_start*).

```
router_pipelines.post('^/plas/pipeline/task_create', async function (req, res,
    ↪ next) {
  try {
    let pipeline = req.body.pipeline;
    let step_from = req.body.step_from;
    let step_to = req.body.step_to;
    let task_id = req.body.task_id;
    if (task_id === '0' && !await task_model.exists_task(cb, task_id)) {
      task_id = uuidv4();
    }
    // Request first step to step provider
    if (pipeline in pipelines) {
      if (pipelines[pipeline].steps && pipelines[pipeline].steps.length >
    ↪ step_from) {
        await gulpfile_pipelines.task_start(none, task_id, req.body.from,
    ↪ pipeline, step_from, step_to);
        await gulpfile_pipelines.step_start(none, await task_model.get_task(none
    ↪ , task_id), step_from);
        res.json({task: await task_model.get_task(none, task_id), step: await
    ↪ task_model.get_step(none, task_id, step_from), message: `Task ${task_id}
    ↪ created` , errorcode: "OK0001"});
      } else {
        message = `pipeline ${pipeline} step from out of range of pipeline steps
    ↪ `;
        errorcode = "PIPE0001";
        await gulpfile_pipelines.task_update_error(none, 'post.task_create',
    ↪ logger, task, "", message, errorcode);
        res.status(500).json({message: message, errorcode: errorcode});
      }
    } else {
        message = `pipeline ${pipeline} not defined`;
        errorcode = "PIPE0002";
        await gulpfile_pipelines.task_update_error(none, 'post.task_create',
    ↪ logger, task, "", message, errorcode);
        res.status(500).json({message: message, errorcode: errorcode});
    }
  } catch(error) {
    errorcode = "PIPE0003";
    message = `post.task_create internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
```

```
});
```

Servicio /plas/pipeline/step_info

http: *POST*

A partir de los identificadores de tarea **task_id** y de paso **step_id** revuelve la estructura *json* completa de información para esa tarea y paso desde la base de datos de orquestación.

```
router_pipelines.post('^/plas/pipeline/step_info', async function (req, res,
    ↪ next) {
  try {
    let task_id = req.body.task_id;
    let step_id = req.body.step_id;
    logger.info('HTTP POST /step_info ' + task_id);
    res.send({task : await task_model.get_task(none, task_id), steps : await
    ↪ task_model.get_step(none, task_id, step_id)});
  } catch(error) {
    errorcode = "PIPE0020";
    message = `post.step_info internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio /plas/pipeline/pipelines

http: *GET*

Devuelve toda la estructura compilada de *pipelines*.

```
router_pipelines.get('^/plas/pipeline/pipelines', async function (req, res, next
    ↪ ) {
  try {
    logger.info('HTTP POST /pipelines ');
    res.send(pipelines);
  } catch(error) {
    errorcode = "PIPE0021";
    message = `get.pipelines internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/step_

http: *POST*

Ejecuta el servicio identificado por el parámetro *method* de la *request*, de acuerdo a los siguientes datos que llegan en el cuerpo:

- ***step_id*** identificador del paso dentro de la tarea
- ***task_id*** identificador de tarea de usuario
- **step** estructura de información asociada al paso
- **task** estructura de información asociada a la tarea

La ejecución se deriva a la función de *gulpfile_pipelines* del mismo nombre que el indicado en el parámetro *method*.

```
router_pipelines.post('^/plas/pipeline/step_:method', async function (req, res,
    ↪ next) {
  try {
    let method = 'step_' + req.params.method;
    logger.debug( `HTTP ${req.params.method} ${req.body.task.task_id}_${req.body
    ↪ .step.step_id}`);
    let step = req.body.step;
    let task = req.body.task;
    let task_id = task.task_id;
    let step_id = step.step_id;
    await gulpfile_pipelines[method](none, task, step);
    res.send({from: process.env.PLASMIDNODE, message: `Task ${task_id} Pipeline
    ↪ ${task.pipeline} Step ${step_id} resolving deps`, errorcode: "OK0001"});
  } catch(error) {
    errorcode = "PIPE0004";
    message = `router_pipelines.step ${req.params.method} internal error ${error
    ↪ .message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/task_info

http: *POST*

Devuelve la información completa de la tarea con todos sus pasos. La tarea se identifica por su ***task_id*** que se informa en el cuerpo de la *request*.

```
router_pipelines.post('^/plas/pipeline/task_info', async function (req, res,
    ↪ next) {
  try {
    let task_id = req.body.task_id;
    logger.info('HTTP POST /task_info ' + task_id);
```

```
    res.send({task : await task_model.get_task(none, task_id), steps : await
    ↪ task_model.get_steps(none, task_id)});
  } catch(error) {
    errorcode = "PIPE0009";
    message = `post.task_info internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/task_info

http: *GET*

Versión bajo método http *GET* con la misma funcionalidad que el servicio anterior. Ahora el identificador de tarea se pasa por parámetro.

```
router_pipelines.get('^/plas/pipeline/task_info/:task_id', async function (req,
    ↪ res, next) {
  try {
    let task_id = req.params.task_id;
    logger.info('HTTP GET /task_info ' + task_id);
    res.send({task : await task_model.get_task(none, task_id), steps : await
    ↪ task_model.get_steps(none, task_id)});
  } catch(error) {
    errorcode = "PIPE0010";
    message = `get.task_info internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio /plas/pipeline/task/:task_id

http: *GET*

Devuelve la información de la tarea (sólo de la tarea, no de sus pasos) identificada por el parámetro *task_id*.

```
router_pipelines.get('^/plas/pipeline/task/:task_id', async function (req, res,
    ↪ next) {
  try {
    logger.info('HTTP /task ' + req.params.task_id);
    res.json({task : await task_model.get_task(none, req.params.task_id)});
  } catch(error) {
    errorcode = "PIPE0012";
```

```
    message = `get.task internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/steps/:task_id

http: *GET*

Devuelve la información de todos los pasos de la tarea identificada por el parámetro *task_id*.

```
router_pipelines.get('^/plas/pipeline/steps/:task_id', async function (req, res,
    ↪  next) {
  try {
    logger.info('HTTP /steps ' + req.params.task_id);
    res.json({steps : await task_model.get_steps(none, req.params.task_id)});
  } catch(error) {
    errorcode = "PIPE0013";
    message = `get.steps internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio /plas/pipeline/step/:task_id/:step_id

http: *GET*

Devuelve la información del paso identificado por los parámetros *task_id*, *step_id*.

```
router_pipelines.get('^/plas/pipeline/step/:task_id/:step_id', async function (
    ↪ req, res, next) {
  try {
    logger.info('HTTP /step ' + eq.params.task_id + ' ' + req.params.step_id);
    res.json({step : await task_model.get_step(none, req.params.task_id, req.
    ↪ params.step_id)});
  } catch(error) {
    errorcode = "PIPE0014";
    message = `get.step internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/task_update

http: *POST*

Actualiza o inserta un paso concreto (vía *upsert* dentro de una tarea). Los datos identificativos llegan en el cuerpo:

- ***task_id*** identificador de tarea de usuario a modificar.
- **step** estructura completa de la nueva información del paso.
- **from** identificador del *provider* del paso.

Una vez actualizado el paso, se continúa la actualización de la tarea, mediante la invocación de la función *task_continue*.

```
router_pipelines.post('^/plas/pipeline/task_update', async function (req, res,
    ↪ next) {
  try {
    logger.info('HTTP /task update', req.body.task_id);
    let task_id = req.body.task_id;
    let step = req.body.step;
    let from = req.body.from;
    let task = await task_model.get_task(none, task_id);
    let step_id = step.step_id;
    await task_model.upsert_step(none, step);
    await gulpfile_pipelines.task_continue(none, task, step);
    res.send({message: `Task ${task_id} Pipeline ${task.pipeline} Step ${req.
    ↪ body.step.step_id} updated`, errorcode: "OK0002"});
  } catch(error) {
    errorcode = "PIPE0005";
    message = `post.task_update internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error);
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicio /plas/pipeline/step/step_insert

http: *GET*

Crea una paso nuevo dentro de la tarea *test* para pruebas de las operaciones de *upsert*. El servicio sólo está activo en el entorno de desarrollo.

```
ENV === "dev" && router_pipelines.get('^/plas/pipeline/step_insert', async
    ↪ function(req, res, next) {
  try {
    logger.info('HTTP /step_insert ');
    let step = {
      task_id: 'test',
      step_id: '0',
      state: 'TEST',
```

```
      provider: 'plasmid01',
      command: 'none',
      exec_type: '',
      owner: 'plasmid01',
      stdout: []
    };
    await task_model.upsert_step(none, step);
    res.send({step : await task_model.get_step(none, step.task_id, step.step_id)
    ↪ });
  } catch(error) {
    errorcode = "PIPE0015";
    message = `get.step_insert internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

Servicios /plas/pipeline/test/task_upsert

http: *GET*

Crea una tarea nueva para pruebas de las operaciones de *upsert*. El servicio sólo está activo en el entorno de desarrollo.

```
ENV === "dev" && router_pipelines.get('^/plas/pipeline/test/task_upsert', async
    ↪ function (req, res, next) {
  try {
    logger.info('HTTP /task_insert ');
    let task = {
      task_id: 'test_001',
      client: 'plasmid01',
      owner: process.env.PLASMIDNODE,
      pipeline: 'test',
      step: 1,
      step_from: 1,
      step_to: 1,
      state: 'TEST',
      creation_date: (new Date()).getTime()
    }
    await task_model.upsert_task(none, task);
    res.send({task : await task_model.get_task(none, task.task_id)});
  } catch(error) {
    errorcode = "PIPE0011";
    message = `get.task_insert internal error ${error.message}`;
    logger.error(message + ' ' + errorcode);
    logger.error(error.stack.split('\n'));
    res.status(500).json({message: message, errorcode: errorcode});
  }
});
```

```
router_pipelines.use(require_dyn("logger_express").error);

router_pipelines.use(function(err, req, res, next) {
  res.status(500);
  res.json({message: "Generic Internal Error"});
  next();
});

module.exports = router_pipelines;
```

B.7. Automatización Extrema

Todas las tareas de mantenimiento de la aplicación deben ejecutarse bien automáticamente bien con el menor esfuerzo manual posible del usuario.

Las tareas las agrupamos en tres bloques:

1. Generación documental.
2. Gestión de contenedores (*docker*).
3. Desarrollo y pruebas de software.

B.7.1. Generación Documental

El módulo **gulpfile_doc.js** contiene las funciones relacionadas con la generación automática de documentación.

Como módulo de tipo *gulp*, muchas de sus funciones son llamables directamente desde la línea de comandos.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const execa = require('execa');
const clean = require('gulp-clean');
const fs = require('fs');
const path = require('path');
const readline = require('readline');
const c = require('ansi-colors');
const WATCHER_OPTIONS = {events: 'all', interval: 2000, usePolling: true,
    ↪ ignoreInitial: false};
const DEFAULT_ROOT_CODE = "/Users/nandoide/PLASMIDBUILD/CODE/plasmidnet";
const DEFAULT_ROOT_DOC = "/Users/nandoide/PLASMIDBUILD/CODE/plasmidnet/doc";
const PROJECT = "plasmidnet";
const TITLE = "PlasmidNet";
const CODE_OPEN = "```javascript";
const CODE_CLOSE = "```";
const cheerio = require('cheerio');
```

En las definiciones por defecto para la línea de comandos utilizamos como fichero de ejemplo el propio gulpfile_doc.js. Esto nos facilita el desarrollo ya que en el mismo fichero sobre el que estamos realizando las modificaciones, estamos a la vez redactando la documentación.

```
const argv = require('yargs')
  .default('file', 'gulpfile_doc.js')
  .default('doc_type', 'detail')
  .default('scope', 'doc')
  .default('lang', 'es')
  .default('texgraph', 'doc_diagram_webapp')
  .default('pdf', 'N')
  .argv;
var state;
var glossary = {};
```

Estructura de las definiciones documentales

En el objeto **structures** definimos qué documentos vamos a generar y qué ficheros contribuyen a cada uno de ellos y en que orden (**files**).

En **label** indicamos los títulos que se añadirán a la documentación, en los idiomas correspondientes. Utilizamos las abreviaturas universales de dos caracteres ('es' para Español, 'en' para inglés, ...)

El campo **tex_shift** indica si en la transformación a *latex* los niveles de capítulos y secciones cuántos niveles deben desplazarse hacia arriba o hacia abajo (necesario para adecuar el documento a un agregado que incorpore ya los niveles superiores). Necesitamos en *detail* y *user* que los capítulos pasen a ser secciones, y las secciones a subsecciones, ya que el agregado incluye en un capítulo todo el documento. Por ello debemos marcar *shift* como *1*, en *detail* y *user*.

El campo *parser* indica la función utilizada para analizar cada línea del fichero, donde se decide entre otras cosas si se trata de una línea de comentario o de una línea de código.

```
let common_files = [
  "gulpfile.js",
  "web/js/plasmidnet/plasmidnet.js",
  "web/js/plasmidnet/plasmidnet_widgets.js",
  "web/js/plasmidnet/plasmidnet_loki.js",
  "web/js/plasmidnet/plasmidnet_datatables.js",
  "web/js/plasmidnet/plasmidnet_graphs.js",
  "web/js/plasmidnet/plasmidnet_global.js",
  "web/js/plasmidnet/plasmidnet_sw.js",
  "web/sw.js",
  "modules/gulpfile_app.js",
  "modules/router.js",
  "modules/router_services.js",
  "modules/logger_app.js",
  "modules/logger_express.js",
  "modules/proxy_modules.js",
  "modules/gulpfile_bio.js",
  "pipelines/plasmid_modules/load_init/load_plasmidnet_db.js",
  "web/js/plasmidnet/mode-plas.js",
  "modules/gulpfile_cms.js",
```

```
  "modules/gulpfile_pipelines.js",
  "modules/pipelines.js",
  "modules/gulpfile_task_model_sqlite.js",
  "modules/gulpfile_task_model_loki.js",
  "modules/gulpfile_task_model_redis.js",
  "modules/router_pipelines.js",
  "modules/gulpfile_doc.js",
  "modules/gulpfile_site.js",
  "modules/gulpfile_dist.js",
  "modules/gulpfile_docker.js",
  "modules/gulpfile_misc.js",
  "doc/resources/roadmap.md"];
let structures = {
  high_level: {label: {es: "PlasmidNet", en: "PlasmidNet"},
            files: ["doc/resources/introduccion.md"].concat(common_files),
            tex_shift: 0,
            parser: parser_other},
  detail: {label: {es: "Manual del Programador", en: "Programmer Manual"},
            files: common_files,
            tex_shift: 1,
            parser: parser_detail},
  slides: {label: {es: "Presentación", en: "Slides"},
            files: ["doc/resources/introduccion.md"].concat(common_files),
            parser: parser_other},
  user: {label: {es: "Manual de Usuario", en: "User Manual"},
            files: ["modules/gulpfile_app.js", "modules/gulpfile_bio.js", "
    ↪ modules/gulpfile_cms.js",
            "modules/gulpfile_pipelines.js",
            "modules/gulpfile_task_model_sqlite.js",
            "modules/gulpfile_task_model_loki.js",
            "modules/gulpfile_task_model_redis.js",
            "modules/gulpfile_doc.js", "modules/gulpfile_site.js", "modules/
    ↪ gulpfile_dist.js", "modules/gulpfile_docker.js", "modules/gulpfile_misc.
    ↪ js"],
            tex_shift: 1,
            parser: parser_other},
  glossary: {label: {es: "Glosario", en: "Glossary"},
            files: ["doc/resources/introduccion.md", "web/js/plasmidnet/
    ↪ plasmidnet.js", "modules/gulpfile_app.js",   "modules/gulpfile_cms.js", "
    ↪ modules/gulpfile_doc.js"],
            tex_shift: 1,
            parser: parser_other}
}
```

Metadata

En la estructura *metadata* configuramos información adicional necesaria para las conversiones, como los literales del capítulo bibliográfico.

```
let metadata = {
  title: "PlasmidNet",
```

```
  date: "1 de Enero de 2020",
  biblio_label: {es: "Bibliografía", en: "Bibliography"}
}
```

doc_execute

Ejecuta el comando de sistema indicado en el string de entrada *cmd* Se utiliza *execa* sincronizado con la construcción *async+await*.

Los comandos típicos que llegan a esta función son las llamadas a *pandoc*.

```
async function doc_execute(cmd, options={}) {
  let logger = require_dyn("logger_app");
  let execa_options = {stdin: 'inherit'};
  let command;
  try {
    command = await execa.command(cmd, options);
    logger.info(`Command ${cmd} ${command.stdout.slice(-500,-1)}`);
  } catch (error) {
    let error_str = `ERROR DOC00001: gulpfile_doc.doc_execute: ${error.stderr} $
    ↪ {cmd}`;
    logger.error(error_str);
    throw error_str;
  }
}
```

open_code

Es utilizada por los analizadores de líneas para insertar automáticamente los caracteres *markdown* que indican el inicio de un bloque de código (tres comillas seguidas del lenguaje de programación). Están definidos en la constante *CODE_OPEN*)

```
function open_code(output, output_type) {
  if (output_type.slice(-1).pop() !== "code") {
    output.push(CODE_OPEN);
    output_type.push("markdown");
  }
}
```

close_code

Es utilizada por los analizadores de líneas para insertar automáticamente los caracteres *markdown* que indican el fin de un bloque de código (tres comillas). Están definidos en la constante *CODE_CLOSE*)

```
function close_code(output, output_type) {
  if (output_type.slice(-1).pop() === "code") {
    output.push(CODE_CLOSE);
    output_type.push("markdown");
  }
}
```

parser_other

Realiza el análisis de la línea de documento *line* filtradas por el tipo de documento *doc_type* (slides, high_level, ...) y el idioma. Decide qué líneas se almacenan en la matriz *output*, diferenciando en la matriz *output_type* cuáles son *markdown* y cuáles son de código.

El idioma se detecta en la línea de apertura de comentarios multilínea, rodeado de dos símbolos arroba:

```
@es@ {detail}
```

El tipo de documento debe estar descrito también en esa primera línea, sin ningún formato en particular, pero estamos utilizando el convenio de incluirlos entre llaves *{}* separados por comas (puede declararse más de un tipo de documento para cada fragmento de *markdown*)

La inclusión de las líneas de código, a diferencia del tipo de documento *detail*, debe ser explícita, y por ello se señaliza entre dos líneas comentadas con *//+*, la primera de ellas debe incluir también el tipo de documento. El idioma no se analiza en este caso: entendemos que todas las líneas aplican para todos los idiomas.

En el caso del tipo *slides* los separadores de slides pueden indicarse (de acuerdo al convenio de *pandoc*) con una línea de guiones: —— en *markdown*, cuatro como mínimo. Hay que tener en cuenta que *pandoc* separa las *slides* automáticamente en cada declaración de nivel # o ##, pero no lo hace en niveles inferiores. Para que estas separaciones ad hoc no ensucien la prosa de otros tipos de documentos que compartan el fragmento, ignoramos estas líneas en todos los demás tipos.

```
function parser_other(line, output, output_type, doc_type, lang) {
  let label = '@' + lang + '@';
  if (line.match(/^\/\/\-/));
  else if (line.match(/^\-\-\-\-+/) && doc_type !== "slides");
  else if (line.match(new RegExp(`^\\s*\\/\\*.*${doc_type}`))
    && line.match(new RegExp(`^\\s*\\/\\*.*${label}`))) {
    state = "markdown";
  } else if (line.match(/^\s*\/\*\s*/)) {
    state = "ignore";
  } else if (line.match(/^\s*\*\/\s*/)) {
    state = "ignore";
  } else if (line.match(new RegExp(`^\\/\\/\\+.*${doc_type}`))) state = "code";
  else if (line.match(/^\/\/\+/)) state = "ignore";
  else if (state === "code") {
    open_code(output, output_type);
```

```
    output.push(line);
    output_type.push(state);
  } else if (state === "markdown") {
    close_code(output, output_type);
    output.push(line);
    output_type.push(state);
  }
}
```

parser_detail

Realiza el análisis de la línea de documento *line* filtradas por el tipo de documento *detail* y el idioma. El análisis es muy similar al de la función anterior, pero aquí las líneas de código son incluidas automáticamente. Si queremos que alguna línea de código no se tenga en cuenta, es necesario delimitar el bloque de líneas entre comentarios //-.

Ese bloque de código eliminado se refleja en la salida como puntos suspensivos.

```
function parser_detail(line, output, output_type, _, lang) {
  let label = '@' + lang + '@';
  if (line.match(/^\/\/\+/));
  else if (line.match(/^\-\-\-\-+/));
  else if (line.match(/\/\/\*\*\*\*/)) state = "ignore";
  else if (line.match(/^\s*\/\*.*detail/)
    && line.match(new RegExp(`^\\s*\\/\\*.*${label}`))) {
    state = "markdown";
  } else if (line.match(/^\s*\/\*\s*/)) state = "ignore_markdown";
  else if (line.match(/^\s*\*\/\s*/)) state = "code";
  else if (line.match(/^\/\/\-/) && state === "ignore_code") {
    state = "code";
    line = "...";
    open_code(output, output_type);
    output.push(line);
    output_type.push(state);
  } else if (line.match(/^\/\/\-/)) state = "ignore_code";
  else if (state === "code") {
    open_code(output, output_type);
    output.push(line);
    output_type.push(state);
  } else if (state === "markdown") {
    close_code(output, output_type);
    output.push(line);
    output_type.push(state);
  }
}
```

doc_extract_md

En esta función construimos los ficheros *markdown* del fichero fuente *file* teniendo en cuenta su tipo de documento y su idioma, lo que sirve para filtrar la información multi-documento presente en el fichero fuente.

Utilizamos un *readStream* encapsulado en una *Promise* como hemos hecho en otros puntos de la aplicación. En cada caso se llama al analizador asociado al *doc_type* (*parser_detail* o *parser_other*), que está definido en la estructura.

La función devuelve dos salidas: un fichero *markdown* que se almacena en el *path* md de la documentación y un *string* que se devuelve al programa con el mismo contenido *markdown*, con el fin de facilitar procesos posteriores.

La función no se limita a extraer el *markdown* desde el código (filtrado por *doc_type* y *lang*), también, apoyado en los *parser*, incorpora a la salida los fragmentos de código fuente, de forma que los ficheros resultantes constituyen una fuente de consulta en sí misma, más cómoda documentalmente.

Estos ficheros tienen la forma: *nombreFicheroFuente_tipoDocumento_idioma.md*

También son procesados por esta función los ficheros *markdown* puros, sin código fuente, como *introduccion.md*, que estamos utilizando para redactar documentación de alto nivel que no es fácilmente asociable a un fichero de código fuente.

En realidad estos ficheros *markdown* específicos no son *markdown* puros porque utilizan también las convenciones que indican el tipo de documento y el idioma.

```
async function doc_extract_md(cb, file=argv.file, doc_type=argv.doc_type, lang=
    ↪ argv.lang) {
  let logger = require_dyn("logger_app");
  try {
    let file2convert = `${DEFAULT_ROOT_CODE}/${file}`;
    const readInterface = readline.createInterface({
      input: fs.createReadStream(file2convert),
      crlfDelay: Infinity
    });
    let nlines = 0;
    state = "ignore";
    let parser = structures[doc_type].parser;
    let output = [];
    let output_type = [];
    output_type.push("markdown");
    return new Promise((resolve, reject) => {
      readInterface.on('line', function(line) {
        parser(line, output, output_type, doc_type, lang);
        nlines++;
      })
      .on('close', function() {
        logger.info(`doc_extract_md.readInterface.close: ${nlines} lines
    ↪ processed`);
        logger.info("doc_extract_md Finish");
        close_code(output, output_type);
        let md = output.join('\n') + '\n';
        fs.writeFileSync(`${DEFAULT_ROOT_DOC}/md/${path.basename(file)}_${
    ↪ doc_type}_${lang}.md`, md);
```

```
        resolve(md);
      })
      .on('error', function(error) {
        logger.error('doc_extract_md.readInterface.error');
        logger.error(error);
        reject(error);
      });
    });
  } catch(error) {
    logger.error(error);
    if (cb) {cb()};
  }
}
```

doc_md_to_tex

Esta es la transformación a formato *latex* mediante *pandoc* de un fichero fuente (puro) en *markdown*. que antes ha sido construido por la función *doc_extract_md*.

Incluimos las referencias bibliográficas que incluimos en *biblio.bib* en formato *latex*.

Tenemos la posibilidad de subir o bajar el nivel de los capítulos en base al parámetro *shift*. Lo estamos utilizando sólo para bajar el *detail* porque necesitamos incorporarlo en un documento agregado donde el *detail* se incrusta en un capítulo del documento final (anexo).

Aplicamos filtros de ajuste de la plantilla no viables con *pandoc* o que requerirían adentrarse en la utilización a bajo nivel del paquete (programación de filtros en AST *abstract_syntax_tree*).

Eliminamos las extensiones *.pdf* ya que *includegraphics* no es capaz de procesar correctamente esta extensión (requiere que no se utilice la extensión, lo que es un problema porque sí se necesita en *html*). O modificábamos el fichero *.tex* o modificábamos el fichero *.html*. Al final optamos por enviar a *latex* los *svg* como *pdf* con extensión.

La otra modificación implementada es cambiar las /section por /lsection para adscribirnos al formato del documento agregado que queremos obtener.

```
async function doc_md_to_tex(cb, shift=0, file=argv.file) {
  let cmd = `
    pandoc -f markdown+auto_identifiers+table_captions \
    --top-level-division chapter \
    --natbib --bibliography=${DEFAULT_ROOT_DOC}/biblio.bib \
    --shift-heading-level-by=${shift} \
    ${DEFAULT_ROOT_DOC}/md/${file}.md \
    -o ${DEFAULT_ROOT_DOC}/tex/${file}_raw.tex -t latex --listings
  `;
  await doc_execute(cmd);
  let raw_tex = fs.readFileSync(`${DEFAULT_ROOT_DOC}/tex/${file}_raw.tex`, {
    ↪ encoding: "utf8"});
  let tex = raw_tex.toString().replace(/\.pdf/g, '.pdf')
            .replace(/\\lsection/g, '\\lsection');
  fs.writeFileSync(`${DEFAULT_ROOT_DOC}/tex/${file}.tex`, tex, {encoding: "utf8
    ↪ "});
```

```
  if (cb) {cb()};
}
```

doc_md_to_slides

Realiza la transformación desde *markdown* a slides *revealjs* mediante *pandoc*.

Utilizamos un template revealjs con leves modificaciones sobre el original y un *css* también específico donde hemos variado algunas cosas como el que no utilice un tema por defecto si no se informa, como es el caso.

En el css definimos que la alineación del texto sea a la izquierda.

```
async function doc_md_to_slides(cb, file=argv.file) {
  let cmd = `
    pandoc -f markdown+auto_identifiers \
    -M title=${metadata.title} \
    -V revealjs-url=https://revealjs.com \
    --template=${DEFAULT_ROOT_DOC}/resources/template.revealjs \
    --css=css/slides.css \
    -V center=false \
    ${DEFAULT_ROOT_DOC}/md/${file}.md \
    -o ${DEFAULT_ROOT_DOC}/html/${file}.html -t revealjs -s \
    --slide-level 2
  `;
  await doc_execute(cmd);
  if (cb) {cb()};
}
```

doc_md_to_slides_pdf

Realiza la transformación desde *markdown* a slides *beamer* mediante *pandoc*.

Utilizamos un template beamer.yaml para definir el paquete de latex *fvextra* necesario para la presentación del código fuente. También redefinimos el caption para que no aparezca aquí el texto *Figure* y el comando *ref* para que no se muestre la referencia (el enlace a la figura no tiene sentido en slides).

Hay que tener en cuenta que *pandoc* realiza la conversión a *pdf* pasando antes por *latex*, ya que beamer es en realidad un entorno para definir *slides* desde *latex*.

Y que no tiene de referencia ningún directorio de trabajo, por lo que hemos de indicárselo en el parámetro *resource-path*. En caso contrario no será capaz de localizar las imágenes.

```
async function doc_md_to_slides_pdf(cb, file=argv.file) {
  let cmd = `
    pandoc -f markdown+auto_identifiers \
    --metadata-file ${DEFAULT_ROOT_DOC}/resources/beamer.yaml \
    -M title=${metadata.title} \
    -V theme=metropolis -V colortheme=rose \
```

```
    -V fontsize=9pt \
    --resource-path=${DEFAULT_ROOT_DOC} \
    --toc \
    ${DEFAULT_ROOT_DOC}/md/${file}.md \
    -o ${DEFAULT_ROOT_DOC}/html/${file}.pdf -t beamer -s \
    --slide-level 2
  `;
  await doc_execute(cmd);
  if (cb) {cb()};
}
```

doc_md_to_html

Realiza la transformación desde *markdown* a *html* mediante *pandoc*.

Utilizamos una definición de sintaxis javascript para obtener mejores resultados en el resaltado de sintaxis.

```
async function doc_md_to_html(cb, shift=0, file=argv.file) {
  let cmd = `
    pandoc -f markdown -s \
    --to html5  \
    --highlight-style=tango \
    --wrap=auto --columns=80 \
    --syntax-definition=${DEFAULT_ROOT_DOC}/resources/javascript.xml \
    --filter=pandoc-citeproc \
    -V link-citations=true \
    --reference-links \
    --natbib --bibliography=${DEFAULT_ROOT_DOC}/biblio.bib \
    --shift-heading-level-by=${shift} \
    ${DEFAULT_ROOT_DOC}/md/${file}.md \
    -o ${DEFAULT_ROOT_DOC}/html/${file}.html
  `;
  await doc_execute(cmd);
  if (cb) {cb()};
}
```

doc_latex_to_pdf

Genera el *pdf* global desde el agregado latex vía *pdflatex*.

El *pdf* global se encuentra sobre una plantilla latex externa al sistema *pandoc*.

Es necesario ejecutar *bibtex* en primer lugar con el fin de actualizar los enlaces de las citas, imágenes y tablas.

Y dos veces *pdflatex* para que se resuelvan los links. En la primera ejecución sólo se actualiza la lista de imágenes

```
async function doc_latex_to_pdf(cb) {

  // Update bibliographic links
  let cmd = `
    /Library/TeX/texbin/bibtex ${PROJECT}.aux
  `;
  await doc_execute(cmd, {cwd: DEFAULT_ROOT_DOC});
  // Create pdf
  cmd = `
    /Library/TeX/texbin/pdflatex -synctex=1 \
    -interaction=nonstopmode \
    ${PROJECT}.tex
  `;
  await doc_execute(cmd, {cwd: DEFAULT_ROOT_DOC});
  // It`s mandatory to execute _pdflatex_ two more times to resolve all links
  await doc_execute(cmd, {cwd: DEFAULT_ROOT_DOC});
  await doc_execute(cmd, {cwd: DEFAULT_ROOT_DOC});
  if (cb) {cb()};
}
```

tex_to_svg

Convierte un fichero latex *tikz*, *smartdiagram*, ... en *pdf* y *svg*, que serán utilizados respectivamente por los ficheros de salida latex y html. Los ficheros de entrada se buscan en resources/tikz

```
async function tex_to_svg(cb, texgraph=argv.texgraph) {
  let cmd = `
    pdflatex -synctex=1 -interaction=nonstopmode -output-directory ${
    ↪ DEFAULT_ROOT_DOC}/images -shell-escape \
    ${DEFAULT_ROOT_DOC}/resources/tikz/${texgraph}.tex
  `;
  await doc_execute(cmd);
  cmd = `
    pdf2svg ${DEFAULT_ROOT_DOC}/images/${texgraph}.pdf \
    ${DEFAULT_ROOT_DOC}/images/${texgraph}.pdf
  `;
  await doc_execute(cmd);
  if (cb) {cb()};
}
```

generate_toc

Genera el índice de contenidos para la versión agregada del documento html de la aplicación. Las plantillas están definidas al principio de la función.

Las entradas de la lista de contenidos se obtienen de los tags html h1 y h2 que aparecen en la salida html de la conversión realizada por *pandoc*. El resultado de esta conversión previa se recibe en el parámetro *content*.

En *part_ name* recibe el nombre del documento a agregar (viene dado por el campo *label* de las estructuras documentales definidas más arriba).

```
function generate_toc(doc_type, content, part_name) {
  const template_toc_part_header = `
  <a class="nav-link collapsed" href="#collapse_part_toc_##part_name_idx##" data
    ↪ -toggle="collapse" data-target="#collapse_part_toc_##part_name_idx##"
    ↪ aria-expanded="false" aria-controls="#collapse_part_toc_##part_name_idx
    ↪ ##">
     <span class="pn-toc">##part_name##</span>
  </a>
  <div class="nav-item collapsed collapse" id="collapse_part_toc_##part_name_idx
    ↪ ##">`;
  const template_toc_chapter = `
  <li class="nav-item">
    <a class="nav-link collapsed" href="#collapse##item##" data-toggle="collapse
    ↪ " data-target="#collapse##item##" aria-expanded="true" aria-controls="#
    ↪ collapse##item##">
      <span pn-title>##chapter##</span>
    </a>
    <div id="collapse##item##" class="collapse" aria-labelledby="collapse" data-
    ↪ parent="#accordionSidebar">
      <div class="bg-white py-2 collapse-inner rounded">
        <h6 class="collapse-header" >##chapter##</h6>
        ##sections##
      </div>
    </div>
  </li>
  `;
  const template_toc_part_tail = `</div>`;
  const template_toc_section = `<a class="collapse-item" href="###section_ref
    ↪ ##">##section_name##</a>`;
  pos = 0;
  let part_name_idx = doc_type + '_' + part_name.replace(/\s+/g, '')
  let html_toc = template_toc_part_header.replace(/##part_name##/g, part_name).
    ↪ replace(/##part_name_idx##/g, part_name_idx);
  let chapter_idx = 0;
  let html_section_toc = "";
  let html_section_toc_subsections = "";
  content = content.toString();
  while (pos < content.length) {
    let ini_chapter = content.indexOf('<h1 id=', pos);
    let ini_section = content.indexOf('<h2 id=', pos);
    if (ini_chapter > -1 && (ini_section === -1 || ini_chapter < ini_section)) {
      if (html_section_toc != "") {
        html_section_toc = html_section_toc.replace("##sections##",
    ↪ html_section_toc_subsections);
        html_toc += html_section_toc;
```

```
      }
      chapter_idx++;
      let end_chapter = content.indexOf('</h1>', ini_chapter);
      let [chapter_ref , chapter_name] = content.slice(ini_chapter + 8,
    ↪ end_chapter).split('">');
      html_section_toc = template_toc_chapter.replace(/##chapter##/g,
    ↪ chapter_name).replace(/##item##/g, doc_type + '_' + chapter_idx);
      html_section_toc_subsections = "";
      pos = end_chapter;
    } else {
      if (ini_section > -1) {
        let end_section = content.indexOf('</h2>', ini_section);
        let [section_ref , section_name] = content.slice(ini_section + 8,
    ↪ end_section).split('">');
        html_section_toc_subsections += template_toc_section.replace(/##
    ↪ section_ref##/g, section_ref)
                                                .replace(/##section_name##/g,
    ↪ section_name);
        pos = end_section;
      } else {
        pos = content.length;
      }
    }
  }
  if (html_section_toc != "") {
    html_section_toc = html_section_toc.replace("##sections##",
    ↪ html_section_toc_subsections);
    html_toc += html_section_toc;
  }
  return html_toc + template_toc_part_tail;
}
```

generate_biblio

En esta función creamos los enlaces a la bibliografía, enlaces que no es capaz de generar *pandoc* directamente. Además debemos incluir el *header* del capítulo.

```
function generate_biblio(content, lang) {
  let $ = cheerio.load(content);
  $('.citation').each(function (i, e) {
    let attr = $(this)[0].attribs;
    let ref = "ref-" + attr["data-cites"];
    $(this).contents().wrap(`<a href="#${ref}"></a>`);
  });
  $(`<h1 id="biblio">${metadata.biblio_label[lang]}</h1>`)
    .insertBefore('.references');
  return $.html();
}
```

images_set_mods

En esta función modificamos las imágenes para incluirles la clase bootstrap 'img-fluid' que facilita su escalado multidispositivo y multipantalla

```
function images_set_mods(content) {
  let $ = cheerio.load (content);
  $('img').each(function (i, e) {
    $(this).addClass('img-fluid mx-auto d-block');
  });
  return $.html();
}
```

tables_format

Modificamos las tablas para incluirles clases bootstrap que facilitan su escalado multidispositivo y multipantalla

```
function tables_format(content) {
  let $ = cheerio.load (content);
  $('table').each(function (i, e) {
    $(this).addClass('table table-sm table-dark');
  });
  return $.html();
}
```

headers_id_modify

Es necesario especificar los id de los headers h1 y h2 que genera *pandoc*, con el fin de diferenciarlos cuando agregamos los documentos y antes de generar la TOC. En caso contrario todos los enlaces a los capítulos y secciones desde el TOC apuntarán al primero de los documentos agregados. Creemos que el doc_type es suficiente para los casos de uso que manejamos actualmente. Este es un punto que requiere revisión en posibles generalizaciones de este módulo.

```
function headers_id_modify(content, doc_type) {
  let $ = cheerio.load (content);
  $('h1, h2').each(function (i, e) {
    let attr = $(this)[0].attribs;
    attr['id'] = doc_type + '_' + attr['id'];
  });
  return $.html();
}
```

generate_content_part

Esta función, adapta un html de entrada (*content*) de un determinado tipo de documento (*doc_type*) mediante una cabecera colapsable, que es el inserto html que utilizamos para componer el agregado documental html5 comandado por la función *doc_html_to_webapp*.

```
function generate_content_part(doc_type, content, part_name) {
  const template_content_part_header = `
  <a class="nav-link collapsed" pn-link href="#collapse_##part_name_idx##" data-
    ↪ toggle="collapse" data-target="#collapse_##part_name_idx##" aria-expanded
    ↪ ="true" aria-controls="#collapse_##part_name_idx##">
     <p class="pn-content-header">##part_name##</p>
  </a>
  <div class="collapsed collapse show" id="collapse_##part_name_idx##">`;
  const template_content_part_footer = `</div>`;
  let part_name_idx = doc_type + '_' + part_name.replace(/\s+/g, '')
  let content_part = template_content_part_header.replace(/##part_name##/g,
    ↪ part_name).replace(/##part_name_idx##/g, part_name_idx) +  content +
    ↪ template_content_part_footer;
  return content_part;
}
```

doc_html_to_webapp

Genera la webapp html en base al *doc_type* declarado. Si se indica como *all*, entonces se agregan los tipos *high_level*, *detail* y *user*. También incluye un enlace a la presentación generada por *pandoc* en formato revealjs. Utiliza muchas de las funciones que hemos definido anteriormente y una plantilla *template_doc.html*, que define la estructura de la página, muy parecida a propia de la webapp PlasmidNet, y por tanto basada en *bootstrap* así como enlaces al css y al js *plasmidnet.doc.js* donde se desarrollan las respuestas a los diferentes eventos de la página.

```
function doc_html_to_webapp(cb, doc_type=argv.doc_type, lang=argv.lang) {
  let doc_types;
  if (doc_type === "all") {
    doc_types = Object.keys(structures);
    // all in this case don't include slides
    doc_types = doc_types.filter(x => !['slides'].includes(x));
  } else {
    doc_types = [doc_type];
  }
  let template = fs.readFileSync(`${DEFAULT_ROOT_DOC}/resources/template_doc.
    ↪ html`);
  let bs = template.toString('utf8')
            .replace(/##lang##/g, lang)
            .replace(/##label_slides##/g, structures.slides.label[lang]);
  let content_parts = "";
  let pandoc_css = "";
  let html_toc = "";
  for (let doc_type of doc_types) {
      let pandoc = fs.readFileSync(`${DEFAULT_ROOT_DOC}/html/${PROJECT}_${
    ↪ doc_type}_${lang}.html`);
      // Content
      let ini_content = pandoc.indexOf("<body>");
      let end_content = pandoc.indexOf("</body>", ini_content);
      let content = pandoc.slice(ini_content + 6, end_content - 1);
```

```
      content = generate_biblio(content, lang);
      content = images_set_mods(content);
      content = tables_format(content);
      content = headers_id_modify(content, doc_type);
      content_parts += generate_content_part(doc_type, content, structures[
    ↪ doc_type].label[lang]);
      // css from pandoc output
      let ini_css = pandoc.indexOf("<style");
      let end_css_1 = ini_css;
      let end_css = pandoc.indexOf("</style>", end_css_1 + 8);
      pandoc_css += pandoc.slice(ini_css, end_css + 8);
      // TOC
      html_toc += generate_toc(doc_type, content, structures[doc_type].label[
    ↪ lang]);
  }
  bs = bs.replace('##content##', content_parts)
      .replace('##pandoc_css##', pandoc_css)
      .replace('##chapter_links##', html_toc);
  fs.writeFileSync(`${DEFAULT_ROOT_CODE}/web/${PROJECT}_doc_${doc_type}_${lang}.
    ↪ html`, bs);
  if (cb) {cb()};
}
```

doc_gendoc

Genera toda la documentación asociada al proyecto. Primero extrae los *markdown* de los ficheros fuentes. También de los fuentes que son *markdown* puros, después lanza las transformaciones *pandoc* y por último el agregado para la *web app* y la generación del *pdf* agregado.

```
async function doc_gendoc(cb, doc_type=argv.doc_type, lang=argv.lang, pdf=argv.
    ↪ pdf) {
  // Generate all markdown pieces in the order dictated by **document_structure
    ↪ **
  let doc_types;
  if (doc_type === "all") {
    doc_types = Object.keys(structures);
  } else {
    doc_types = [doc_type];
  }
  for (let doc_type of doc_types) {
    let markdown = "";
    for (let file of structures[doc_type].files) {
      let md = await doc_extract_md(undefined, file, doc_type, lang);
      markdown += md;
    }
    fs.writeFileSync(`${DEFAULT_ROOT_DOC}/md/${PROJECT}_${doc_type}_${lang}.md`,
    ↪  markdown);
  }
  // Define deep (shift) for tex outputs
```

```
  for (let doc_type of doc_types) {
    if (doc_type === "slides") {
      await doc_md_to_slides(undefined, `${PROJECT}_${doc_type}_${lang}`);
      await doc_md_to_slides_pdf(undefined, `${PROJECT}_${doc_type}_${lang}`);
    } else {
      await doc_md_to_html(undefined, 0, `${PROJECT}_${doc_type}_${lang}`);
      await doc_md_to_tex(undefined, structures[doc_type].tex_shift, `${PROJECT}
    ↪ _${doc_type}_${lang}`);
    }
  }
  // Aggregate html to webapp
  if (doc_type !== "slides") {
    await doc_html_to_webapp(undefined, doc_type, lang);
  }
  // Aggregate latex to pdf
  if (doc_type === "all" && pdf === 'Y') {
    await doc_latex_to_pdf();
  }
  if (cb) {cb()};
}
```

man

Genera la ayuda *online* a partir del *markdown* del manual de usuario del ámbito informado en el campo *scope*. El ámbito puede ser *doc*, *docker*, *site*, *dist*, *pipelines*, *bio*, *misc*, *app*, *cms*, o sea, se refiere a cualquier módulo de tareas *gulp* que por convenio están desarrolladas en los ficheros javascript de la forma *gulpfile_ scope.js*.

Para ello realiza un análisis de este *markdown* resaltando las diferentes partes y aplicando los estilos *bold* o *italic* con la ayuda del paquete *ansi-colors* [46].

```
function man(cb, scope=argv.scope, lang=argv.lang) {
  try {
    let user_manual = fs.readFileSync(`${DEFAULT_ROOT_DOC}/md/gulpfile_${scope}.
    ↪ js_user_${lang}.md`).toString().split('\n');
    console.log('\n' + c.bold.cyan("man ") + c.green("scope ") + scope + c.green
    ↪ (" lang ") + lang + '\n');
    for (let line of user_manual) {
      if (line[0] === '>') {
        let task_regexp = /\*\*(.*)\*\*/;
        let task = line.match(task_regexp);
        let params_regexp = / (--[a-zA-Z0-9_]+) /g;
        let values_regexp = /\[([a-zA-Z0-9_,.\-\* \\\|\=>\{\}\"\:]*)\]/g;
        let params = line.match(params_regexp);
        let values = line.match(values_regexp);
        let command = c.bold.cyan(task[1]);
        if (params) {
          for (let i = 0; i< params.length; i++) {
            command += c.green(params[i]);
            let values_array = values[i].split(',');
```

```
            let values_string = '[';
            // Default value to bold
            for (let k in values_array) {
              if (values_array[k].includes('*')) {
                values_array[k] = c.bold.yellow.italic(values_array[k].replace
↪ (/\*/g,'').replace('[', '').replace(']', ''));
              } else {
                values_array[k] = c.yellow.italic(values_array[k].replace('[', '
↪ ').replace(']', ''));
              }
            }
            values_string += values_array.join(',') + ']';
            command += values_string;
          }
        }
        console.log(command);
        console.log("");
      } else if (line[0] !== '#' && line.trim() !== "") {
        // Help lines
        let words = line.split(/\s+/);
        // Managing bold and italic
        for (let k in words) {
          if (words[k][0] === '*') {
            words[k] = c.bold(words[k].replace(/\*/g,''));
          } else if (words[k][0] === '_') {
            words[k] = c.italic(words[k].replace(/\_/g,''));
          }
        }
        console.log(words.join(' '));
      }
    }
    cb();
  } catch(error) {
    console.log(c.red("ERROR DOC00005. Probably scope without associated help ")
↪  , error.message, error);
    cb();
  }
}

module.exports.doc_md_to_tex = doc_md_to_tex;
module.exports.doc_extract_md = doc_extract_md;
module.exports.doc_md_to_html = doc_md_to_html;
module.exports.doc_latex_to_pdf = doc_latex_to_pdf;
module.exports.doc_html_to_webapp = doc_html_to_webapp;
module.exports.man = man;
module.exports.tex_to_svg = tex_to_svg;
module.exports.doc_gendoc = doc_gendoc;
module.exports.shares = module.exports;
```

B.7.2. Pruebas y despliegue de software

gulpfile_site.js

Este módulo incluye las automatizaciones relacionadas con la distribución de software de las *web app* PlasmidNet y la *web app* documental.

```
"use strict";
const {require_dyn} = require('../proxy_modules');
const autoprefixer = require("gulp-autoprefixer");
const browsersync = require("browser-sync").create();
const cleanCSS = require("gulp-clean-css");
const del = require("del");
const gulp = require("gulp");
const header = require("gulp-header");
const merge = require("merge-stream");
const plumber = require("gulp-plumber");
const rename = require("gulp-rename");
const tap = require("gulp-tap");
const sass = require("gulp-sass");
const uglify = require('gulp-uglify-es').default;
const gulpif = require('gulp-if');
const concat = require('gulp-concat');
const sourcemaps = require('gulp-sourcemaps');
const path = require('path');
const DEFAULT_VERSION = 'v0';
const PLASMID_MOUNTS = '.';
const ROOT_SRC_DIR = `${PLASMID_MOUNTS}/CODE/plasmidnet/web`;
const ROOT_DOC_DIR = `${PLASMID_MOUNTS}/CODE/plasmidnet/doc`;
const WATCHER_OPTIONS = {events: 'all', interval: 2000, usePolling: true,
    ↪ ignoreInitial: false};
const ENV = process.env.ENV || 'dev';
const DEFAULT_PORT = '9090'; // Web server http default port
const DEFAULT_PORT_BS = '3000'; // Browsersync default proxy port
const argv = require('yargs')
  .default('pversion', DEFAULT_VERSION)
  .default('port', process.env.port)
  .argv;

// Load package.json for banner
const pkg = require(__basedir + '/package.json');

// Set the banner content
const banner = ['/*!\n',
  ' * Plasmidnet - <%= pkg.title %> v<%= pkg.version %> (<%= pkg.homepage %>)\
    ↪ n',
  ' * Copyright 2020-' + (new Date()).getFullYear(), ' <%= pkg.author %>\n',
  ' * Licensed under <%= pkg.license %>\n',
  ' * PlasmidNet dynamic version 1.0\n',
  ' */\n',
  '\n'
].join('');
```

```
let cms = require_dyn("gulpfile_cms");
let vendor_dist = [
  ROOT_SRC_DIR + '/vendor/fontawesome-free/css/all.min.css',
  ROOT_SRC_DIR + '/vendor/datatables/dataTables.min.css',
  ROOT_SRC_DIR + '/vendor/jquery/jquery.min.js',
  ROOT_SRC_DIR + '/vendor/bootstrap/js/bootstrap.bundle.min.js',
  ROOT_SRC_DIR + '/vendor/jquery-easing/jquery.easing.min.js',
  ROOT_SRC_DIR + '/vendor/axios/axios.min.js',
  ROOT_SRC_DIR + '/vendor/lokijs/lokijs.min.js',
  ROOT_SRC_DIR + '/vendor/lokijs/loki-indexed-adapter.min.js',
  ROOT_SRC_DIR + '/vendor/lokijs/loki-indexed-adapter.js',
  ROOT_SRC_DIR + '/vendor/d3js/d3.min.js',
  ROOT_SRC_DIR + '/vendor/datatables/dataTables.min.js',
  ROOT_SRC_DIR + '/vendor/ace/ace.js',
  ROOT_SRC_DIR + '/vendor/ace/theme-chrome.js',
  ROOT_SRC_DIR + '/vendor/ace/mode-text.js',
  ROOT_SRC_DIR + '/vendor/ace/worker-javascript.js'
];

let vendor_dist_webfonts = [
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-brands-400.eot',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-brands-400.pdf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-brands-400.ttf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-brands-400.woff',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-brands-400.woff2',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-regular-400.eot',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-regular-400.pdf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-regular-400.ttf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-regular-400.woff',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-regular-400.woff2',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-solid-900.eot',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-solid-900.pdf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-solid-900.ttf',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-solid-900.woff',
  ROOT_SRC_DIR + '/vendor/fontawesome-free/webfonts/fa-solid-900.woff2'
]
```

browserSync

Arranca el proxy *broswersync* [2], como intermediario del servidor de la aplicación, que debe estar escuchando en el puerto especificado en *port*.

```
function browserSync(done, port=argv.port) {
  port = port || DEFAULT_PORT;
  browsersync.init({
    proxy: {
      target: `http://localhost:${port}/plas/app/plasmidnet.html`,
      ws: true
```

```
    },
    port: DEFAULT_PORT_BS || 3000
  });
  done();
}
```

css

Genera el fichero *css* de la aplicación para su distribución, compilando los *sass*, concatenándolos, añadiendo el *banner* de autoría, eliminando comentarios, asignando al fichero un código de versión en la base de datos para la versión especificada en el parámetro *pversion* y distribuyéndolo en el directorio del servidor *web* correspondiente a la versión indicada en este parámetro.

Además informa al *proxy browsersync* de que el *css* ha sido modificado. El *proxy* reacciona inyectando automáticamente el *css* en todos los navegadores cliente sin reiniciarlos: los cambios de presentación se reflejan inmediatamente.

```
function css(cb, pversion=argv.pversion) {
  return gulp
    .src(ROOT_SRC_DIR + "/scss/**/*.scss")
    .pipe(plumber())
    .pipe(sass({
      outputStyle: "expanded",
      includePaths: "./node_modules",
    }))
    .on("error", sass.logError)
    .pipe(autoprefixer({
      cascade: false
    }))
    .pipe(header(banner, {
      pkg: pkg
    }))
    .pipe(tap(cms.set_version))
    .pipe(gulp.dest(`site/${pversion}/css`))
    .pipe(rename({
      suffix: ".min"
    }))
    .pipe(cleanCSS())
    .pipe(tap(cms.set_version))
    .pipe(gulp.dest(`site/${pversion}/css`))
    .pipe(browsersync.stream());
}
```

js

Genera el fichero *javascript* de la aplicación para su distribución, concatenándo los ficheros fuentes *javascript*, añadiendo el banner de autoría, eliminando comentarios, añadiendo los mapas

fuente (entorno de desarrollo), asignando al fichero resultante un código de versión en la base de datos asociado a la versión especificada en el parámetro *pversion* y distribuyéndolo en el directorio del servidor *web* correspondiente a la versión indicada en este parámetro.

Además informa al *proxy browsersync* de que el *js* ha sido modificado. El *proxy* reacciona reiniciando automáticamente todos los navegadores cliente: los cambios de presentación se reflejan inmediatamente.

```
function js(cb, pversion=argv.pversion) {
  return gulp
    .src([
      ROOT_SRC_DIR + '/js/plasmidnet/*.js'
    ])
    .pipe(gulpif(ENV === 'dev', sourcemaps.init()))
      .pipe(concat('plasmidnet.min.js'))
      .pipe(uglify())
      .pipe(header(banner, {
        pkg: pkg
      }))
    .pipe(gulpif(ENV === 'dev', sourcemaps.write()))
    .pipe(tap(cms.set_version))
    .pipe(gulp.dest(`site/${pversion}/js`))
    .pipe(gulpif(ENV === 'dev',browsersync.stream()), null);
}
```

js_doc

Genera el fichero *javascript* de la aplicación documental para su distribución, concatenándo los ficheros fuentes *javascript*, añadiendo el banner de autoría, eliminando comentarios, añadiendo los mapas fuentes (entorno de desarrollo), asignando al fichero resultante un código de versión en la base de datos asociado a la versión especificada en el parámetro *pversion* y distribuyéndolo en el directorio del servidor *web* correspondiente a la versión indicada en este parámetro.

En este caso no contamos con *proxy browsersync*. No lo hemos considerado necesario.

```
function js_doc(cb, pversion=argv.pversion) {
  return gulp
    .src([
      ROOT_SRC_DIR + '/js/plasmidnet/plasmidnet_global.js',
      ROOT_SRC_DIR + '/js/plasmidnet/plasmidnet_sw.js'
    ])
    .pipe(gulpif(ENV === 'dev', sourcemaps.init()))
      .pipe(concat('plasmidnet_doc.min.js'))
      .pipe(uglify())
      .pipe(header(banner, {
        pkg: pkg
      }))
    .pipe(gulpif(ENV === 'dev', sourcemaps.write()))
    .pipe(tap(cms.set_version))
    .pipe(gulp.dest(`site/${pversion}/js`))
}
```

other

Distribuye el resto de ficheros: 1) Las páginas html. 2) El *javascript service worker*. 3) El *javascript* externo de las diferentes API(*vendor*). 4) Fuentes. 5) Ficheros misceláneos para pruebas (sólo en desarrollo).

```
function other(cb, pversion=argv.pversion) {
  let cms = require_dyn("gulpfile_cms");
  var root = gulp.src([`${ROOT_SRC_DIR}/sw.js`, `${ROOT_SRC_DIR}/plasmidnet.html
    ↪ `, `${ROOT_SRC_DIR}/plasmidnet_doc_all*.html`]).pipe(tap(cms.set_version)
    ↪ ).pipe(gulp.dest(`site/${pversion}`));
  var root_versions = gulp.src([`${ROOT_SRC_DIR}/plasmidnet_cms.html`]).pipe(tap
    ↪ (cms.set_version));
  var vendor = gulp.src(vendor_dist).pipe(tap(cms.set_version)).pipe(gulp.dest(`
    ↪ site/${pversion}/vendor`));
  var webfonts = gulp.src(vendor_dist_webfonts).pipe(tap(cms.set_version)).pipe(
    ↪ gulp.dest(`site/${pversion}/webfonts`));
  var fonts = gulp.src([`${ROOT_SRC_DIR}/fonts/**/*`]).pipe(tap(cms.set_version)
    ↪ ).pipe(gulp.dest(`site/${pversion}/fonts`));
  var test = gulp.src([`${ROOT_SRC_DIR}/test/*.js`, `${ROOT_SRC_DIR}/test/*.css`
    ↪ ]).pipe(tap(cms.set_version)).pipe(gulpif(ENV === 'dev', gulp.dest(`site/
    ↪ ${pversion}/test`)));
  var test_html = gulp.src([`${ROOT_SRC_DIR}/test/*.html`]).pipe(tap(cms.
    ↪ set_version)).pipe(gulpif(ENV === 'dev', gulp.dest(`site/${pversion}/`)))
    ↪ ;
  return merge(root, vendor, test, fonts, webfonts, test_html, root_versions);
  cb();
}
```

modules

Distribuye como módulos los ficheros *javascript* de la aplicación y actualiza su versión en el *CMS*. También los minimiza y añade los mapas fuente en desarrollo. Esto es imprescindibler para que se puedan cargar dinámicamente desde las *web app*.

```
function modules(cb, pversion=argv.pversion) {
  return gulp
    .src([`${ROOT_SRC_DIR}/js/plasmidnet/*.js`])
    .pipe(gulpif(ENV === 'dev', sourcemaps.init()))
    .pipe(uglify())
    .pipe(tap(cms.set_version))
    .pipe(gulpif(ENV === 'dev', sourcemaps.write()))
    .pipe(gulp.dest(`site/${pversion}/js/modules`));
}
```

doc_other

Distribuye el resto de ficheros de la *web app* documental: 1) Imágenes. 2) *html* de la presentación en *revealjs*. 3) *css*.

```
function doc_other(cb, pversion=argv.pversion) {
  var images = gulp.src([`${ROOT_DOC_DIR}/images/doc_*`]).pipe(tap(cms.
    ↪ set_version)).pipe(gulp.dest(`site/${pversion}/images`));
  var slides = gulp.src([`${ROOT_DOC_DIR}/html/plasmidnet_slides*.html`]).pipe(
    ↪ tap(cms.set_version)).pipe(gulp.dest(`site/${pversion}`));
  var slides_css = gulp.src([`${ROOT_DOC_DIR}/css/*.css`]).pipe(tap(cms.
    ↪ set_version)).pipe(gulp.dest(`site/${pversion}/css`));
  return merge(images, slides, slides_css);
  cb();
}
```

manifest

Distribuye el manifiesto de la *web app*, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

```
function manifest(cb, pversion=argv.pversion) {
  return gulp.src([`${ROOT_SRC_DIR}/manifest.json`]).pipe(tap(cms.set_version)).
    ↪ pipe(gulp.dest(`site/${pversion}`));
}
```

manifest_doc

Distribuye el manifiesto de la *web app* documental, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*.

```
function manifest_doc(cb, pversion=argv.pversion) {
  return gulp.src([`${ROOT_SRC_DIR}/manifest_doc.json`]).pipe(tap(cms.
    ↪ set_version)).pipe(gulp.dest(`site/${pversion}`));
}
```

icons

Distribuye el los iconos de las *web app*, actualizando el sistema de archivos del servidor *web* correspondiente a la versión *pversion*. Los iconos están referenciados en los manifiestos.

```
function icons(cb, pversion=argv.pversion) {
  return gulp.src([`${ROOT_SRC_DIR}/images/*`]).pipe(tap(cms.set_version)).pipe(
    ↪ gulp.dest(`site/${pversion}/images`));
}
```

watchFiles

Activa los *filewatchers* para detectar los cambios en los ficheros fuente.

```
function watchFiles(cb, pversion=argv.pversion) {
  gulp.watch([ROOT_SRC_DIR +  "/scss/**/*"], WATCHER_OPTIONS, css);
  gulp.watch([ROOT_SRC_DIR + "/js/plasmidnet/*.js"], WATCHER_OPTIONS, gulp.
    ↪ series(js, js_doc, modules));
  gulp.watch([`${ROOT_SRC_DIR}/test/*`, `${ROOT_SRC_DIR}/test/*`, `${
    ↪ ROOT_SRC_DIR}/vendor/**/*`,`${ROOT_SRC_DIR}/fonts/**/*`,`${ROOT_SRC_DIR}/
    ↪ sw.js`, `${ROOT_SRC_DIR}/plasmidnet.html`, `${ROOT_SRC_DIR}/
    ↪ plasmidnet_cms.html`, `${ROOT_SRC_DIR}/plasmidnet_doc*.html`],
    ↪ WATCHER_OPTIONS, other);
  gulp.watch([`${ROOT_DOC_DIR}/images/doc_*`, `${ROOT_DOC_DIR}/html/
    ↪ plasmidnet_slides*`, `${ROOT_DOC_DIR}/css/*.css`], WATCHER_OPTIONS,
    ↪ doc_other);
  gulp.watch([`${ROOT_SRC_DIR}/manifest.json`], WATCHER_OPTIONS, manifest);
  gulp.watch([`${ROOT_SRC_DIR}/manifest_doc.json`], WATCHER_OPTIONS,
    ↪ manifest_doc);
  gulp.watch([`${ROOT_SRC_DIR}/images/*`], WATCHER_OPTIONS, icons);
}

const build = gulp.series(gulp.parallel(css, js, js_doc, other, doc_other,
    ↪ modules, manifest, icons));
const bs = gulp.series(build, gulp.parallel(watchFiles, browserSync));
const watcher = gulp.series(build, watchFiles);

// Export tasks
exports.site_css = gulp.series(cms.start_cms, css);
exports.site_js = gulp.series(cms.start_cms, js);
exports.site_modules = gulp.series(cms.start_cms, modules);
exports.site_js_doc = gulp.series(cms.start_cms, js_doc);
exports.site_other = gulp.series(cms.start_cms, other);
exports.site_manifest = gulp.series(cms.start_cms, manifest);
exports.site_manifest_doc = gulp.series(cms.start_cms, manifest_doc);
exports.site_icons = gulp.series(cms.start_cms, icons);
exports.site_bs = bs;
exports.site_watcher = gulp.series(cms.start_cms, watcher);
exports.site_default = gulp.series(cms.start_cms, build);
exports.site_build = gulp.series(cms.start_cms, build);

module.exports.shares = module.exports;
```

gulpfile_dist.js

Contiene las automatizaciones relacionadas con la distribución de software en el servidor. La eliminación de comentarios se efectúa con el API *strip* de *gulpjs* [7].

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
```

```
const merge = require("merge-stream");
const execa = require('execa');
const tap = require("gulp-tap");
const strip = require('gulp-strip-comments');
const gulpif = require('gulp-if');
const DEFAULT_VERSION = 'v0';
const PLASMID_MOUNTS = '.';
const WATCHER_OPTIONS = {events: 'all', interval: 1000, usePolling: true,
    ↪ ignoreInitial: false};
const ENV = process.env.ENV || 'dev';
const argv = require('yargs')
  .default('pversion', DEFAULT_VERSION)
  .argv;

let cms = require_dyn("gulpfile_cms");
```

dist_root

Distribuye el fichero central del servidor *gulpfile.js*.

```
function dist_root() {
  return src([`${PLASMID_MOUNTS}/CODE/plasmidnet/gulpfile.js`])
    .pipe(gulpif(ENV === 'prod', strip()))
    .pipe(dest(`.`));
}
```

dist_package

Distribuye el fichero de Distribuye el fichero de dependencias *package.json* desde el directorio principal del servidor hasta el directorio de código.

La distribución de este fichero es a la inversa porque se actualiza en el sistema de archivos del servidor, cada vez que se instala un nuevo paquete de *nodejs* vía *npm install*.

```
function dist_package() {
  return src([`./package.json`])
    .pipe(dest(`${PLASMID_MOUNTS}/CODE/plasmidnet/`));
}
```

dist_build

Distribuye el los ficheros de construcción de *dockers*.

```
function dist_build() {
  return src([`${PLASMID_MOUNTS}/CODE/plasmidnet/build/*`])
    .pipe(dest(`.`));
```

```
}
```

dist_pipelines

Distribuye el los ficheros de código de pipelines.

```
function dist_pipelines() {
  return src([`${PLASMID_MOUNTS}/CODE/plasmidnet/pipelines/**`])
    .pipe(dest(`pipelines`));
}
```

modules

Distribuye los ficheros *javascript* de módulos en el directorio de despliegue del servidor relacionado con la versión *pversion* y genera los códigos de versión en el *CMS*.

```
function modules(cb, pversion=argv.pversion) {
  return src([`${PLASMID_MOUNTS}/CODE/plasmidnet/modules/*`, `!CODE/plasmidnet/
    ↪ modules/proxy_modules.js`])
    .pipe(gulpif(ENV === 'prod', strip()))
    .pipe(tap(cms.set_version))
    .pipe(dest(`modules/${pversion}`));
  cb();
}
```

dist_proxy_modules

Distribuye *proxy_ modules.js*.

```
function dist_proxy_modules(cb) {
  return src([`${PLASMID_MOUNTS}/CODE/plasmidnet/modules/proxy_modules.js`])
    .pipe(gulpif(ENV === 'prod', strip()))
    .pipe(dest('modules'));
  cb();
}
```

watcher

Arranca los *filewatchers* para la distribución automática en cuanto se producen cambios en los ficheros fuentes. Genera automáticamente los contenidos si detecta un cambio en la maqueta.

```
function watcher(cb, pversion=argv.pversion) {
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/modules/*`, `!CODE/plasmidnet/
    ↪ modules/proxy_modules.js`], WATCHER_OPTIONS, modules);
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/modules/proxy_modules.js`],
    ↪ WATCHER_OPTIONS, dist_proxy_modules);
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/gulpfile.js`], WATCHER_OPTIONS,
    ↪ dist_root);
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/build/*`], WATCHER_OPTIONS,
    ↪ dist_build);
  watch([`./package.json`], WATCHER_OPTIONS, dist_package);
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/pipelines/**`], WATCHER_OPTIONS,
    ↪ dist_pipelines);
  watch([`${PLASMID_MOUNTS}/CODE/plasmidnet/web/plasmidnet_cms.html`],
    ↪ WATCHER_OPTIONS, series(cms.do_upsert_content, cms.create_home));
}

module.exports.watcher = watcher;
module.exports.dist_all = parallel(dist_root, modules, dist_proxy_modules,
    ↪ dist_pipelines, dist_package);
module.exports.dist_root = dist_root;
module.exports.dist_package = dist_package;
module.exports.dist_build = dist_build;
module.exports.dist_pipelines = dist_pipelines;
module.exports.dist_modules = series(cms.start_cms, modules);
module.exports.dist_proxy_modules = dist_proxy_modules;
```

dist_modules_all

Distribuye todos los módulos, incluido *proxy_modules.js*.

```
module.exports.dist_modules_all = parallel(cms.start_cms, modules,
    ↪ dist_proxy_modules);
module.exports.shares = module.exports;
```

B.7.3. Despliegue basado en contenedores

gulpfile_docker.js

En este módulo definimos todas las tareas relacionadas con la gestión de dockers. Principalmente:

1) Tareas de creación de imágenes.
2) Tareas de ejecución de contenedores unitarios.
3) Tareas de ejecución masiva de contenedores.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const execa = require('execa');
const clean = require('gulp-clean');
const DEFAULT_NODE = 'plasmid01';
const DEFAULT_PORT = '9091';
const DEFAULT_DIST_VERSION = 'v.2020.02';
const DEFAULT_BIND_VOLUMES_ROOT = "/Users/nandoide/PLASMIDBUILD";
const DEFAULT_BIND_VOLUMES_ROOT_CODE = "/Users/nandoide/PLASMIDBUILD";
const DEFAULT_BIND_VOLUMES_ROOT_INTERNAL = "/home/plasmiduser/app";
const DEFAULT_IMAGE = '';
const argv = require('yargs')
  .default('dist_version', DEFAULT_DIST_VERSION)
  .default('node', DEFAULT_NODE)
  .default('port', DEFAULT_PORT)
  .default('redis_port', 6379)
  .default('pimage', DEFAULT_IMAGE)
  .default('bind', DEFAULT_BIND_VOLUMES_ROOT)
  .default('bind_code', DEFAULT_BIND_VOLUMES_ROOT_CODE)
  .default('bind_internal', DEFAULT_BIND_VOLUMES_ROOT_INTERNAL)
  .argv;
```

docker_execute

Ejecuta el comando docker con los parámetros pasados en *param_string*.

```
async function docker_execute(param_string) {
  let logger = require_dyn("logger_app");
  let params = param_string.split(/\s+/);
  logger.info(params);
  let execa_options = {env: {'DOCKER_BUILDKIT': 1}};
  try {
    logger.info(`Docker command ${params.join(' ')}`);
    const {stdout, stderr} = await execa('docker', params, execa_options);
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

docker_execute_shell

Ejecuta en una shell el comando pasado en el parámetro *cmd*. Este tipo de ejecución evita que un comando interactivo como el enviado en *docker_sh* falle debido a que el sistema no encuentra un terminal de entrada (*the input device is not a TTY*).

```
async function docker_execute_shell(cmd) {
  let logger = require_dyn("logger_app");
  let execa_options = {stdin: 'inherit'};
  try {
    logger.info(`Docker command ${cmd}`);
    execa.command(cmd, execa_options).stdout.pipe(process.stdout);
  } catch (error) {
    logger.error(`Cannot execute docker command ${error.message}`);
  }
}
```

docker_sh

Ejecuta en *screen* una shell lanzada desde el contenedor identificado por parámetro *node*.

```
async function docker_sh(cb, node=argv.node) {
  let cmd = `screen -dmaS ${node} docker exec -it ${node} sh`;
  await docker_execute_shell(cmd);
  cb();
}
```

docker_server

Arranca la tarea por defecto *gulp* en el contenedor identificado por parámetro *node*.

```
async function docker_server(cb, node=argv.node) {
  let cmd = `screen -dmaS ${node} docker exec -it ${node} gulp`;
  await docker_execute_shell(cmd);
  cb();
}
```

docker_rm

Borra el contenedor identificado por parámetro *node*.

```
async function docker_rm(cb, node=argv.node) {
  let param_string = `rm ${node}`;
  await docker_execute(param_string);
  cb();
}
```

docker_stop

Detiene la ejecución del contenedor identificado por parámetro *node*.

```
async function docker_stop(cb, node=argv.node) {
  let param_string = `stop ${node}`;
  await docker_execute(param_string);
  cb();
}
```

docker_build

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

```
async function docker_build(cb, dist_version=argv.dist_version) {
  let image = `plasmidnet_image:${dist_version} -f Dockerfile`;
  let param_string = `build -t ${image} .`;
  await docker_execute(param_string);
  cb();
}
```

docker_build_dev

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile.dev* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

```
async function docker_build_dev(cb, dist_version=argv.dist_version) {
  let image = `plasmidnet_dev_image:${dist_version} -f Dockerfile.dev`;
  let param_string = `build -t ${image} .`;
  await docker_execute(param_string);
  cb();
}
```

docker_build_dfast

Construye la imagen de la aplicación de acuerdo a las especificaciones del fichero *Dockerfile.dfast* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

```
async function docker_build_dfast(cb, dist_version=argv.dist_version) {
  let image = `plasmidnet_dfast_image:${dist_version} -f Dockerfile.dfast`;
  let param_string = `build -t ${image} .`;
  await docker_execute(param_string);
  cb();
}
```

docker_build_glibc

Construye la imagen de la aplicación DFAST de acuerdo a las especificaciones del fichero *Dockerfile.dfast* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

```
async function docker_build_glibc(cb, dist_version=argv.dist_version) {
  let image = `plasmidnet_glibc_image:${dist_version} -f Dockerfile.glibc`;
  let param_string = `build -t ${image} .`;
  await docker_execute(param_string);
  cb();
}
```

docker_build_mmseq2

Construye la imagen de la aplicación MMSEQS2 de acuerdo a las especificaciones del fichero *Dockerfile.mmseq2* incluyendo en el nombre de la imagen la versión pasada en el parámetro *dist_version*.

```
async function docker_build_mmseq2(cb, dist_version=argv.dist_version) {
  let image = `plasmidnet_mmseq2_image:${dist_version} -f Dockerfile.mmseq2`;
  let param_string = `build -t ${image} .`;
  await docker_execute(param_string);
  cb();
}
```

docker_run

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web.

```
async function docker_run(cb, node=argv.node, pimage=argv.pimage, port=argv.port
    ↪ , dist_version=argv.dist_version, bind=argv.bind, bind_code=argv.
    ↪ bind_code, bind_internal=argv.bind_internal) {
  if (pimage === '') pimage = '_';
  else pimage = `_${pimage}_`;
  let image = `plasmidnet${pimage}image:${dist_version}`;
  let param_string = `run -itd --rm \
  --mount type=bind,src=${bind}/LOG,dst=${bind_internal}/LOG \
  --mount type=bind,src=${bind}/DATA,dst=${bind_internal}/DATA \
  --mount type=bind,src=${bind_code}/CODE,dst=${bind_internal}/CODE \
  --env PORT=${port} \
  --env PLASMIDNODE=${node} \
  --name ${node} \
  -p ${port}:${port} \
  ${image}`;
  await docker_execute(param_string);
```

```
  cb();
}
```

docker_run_dev

Crea y ejecuta el contenedor asociándole el identificador *node*, para la imagen *plasmidnet_dev_image* de código de versión *dist_version*, con puerto de servidor web *port*.

```
async function docker_run_dev(cb, node=argv.node, port=argv.port, dist_version=
    ↪ argv.dist_version) {
  let image = `plasmidnet_dev_image:${dist_version}`;
  let param_string = `run -itd \
  --mount type=bind,src=${DEFAULT_BIND_VOLUMES_ROOT}/LOG,dst=${
    ↪ DEFAULT_BIND_VOLUMES_ROOT_INTERNAL}/LOG \
  --mount type=bind,src=${DEFAULT_BIND_VOLUMES_ROOT}/DATA,dst=${
    ↪ DEFAULT_BIND_VOLUMES_ROOT_INTERNAL}/DATA \
  --mount type=bind,src=${DEFAULT_BIND_VOLUMES_ROOT}/CODE,dst=${
    ↪ DEFAULT_BIND_VOLUMES_ROOT_INTERNAL}/CODE \
  -v /var/run/docker.sock:/var/run/docker.sock \
  --env PORT=${port} \
  --env PLASMIDNODE=${node} \
  --name ${node}_dev \
  -p ${port}:${port} \
  ${image} sh`;
  await docker_execute(param_string);
  cb();
}
```

docker_run_redis

Crea y ejecuta un contenedor para la imagen de la base de datos *redis*, asociándole el identificador *node*, con puerto de servidor redis_port_.

```
async function docker_run_redis(cb, node=argv.node, redis_port=argv.redis_port)
    ↪ {
  let image = `bitnami/redis:latest`;
  let param_string = `run -itd --rm \
  --name ${node} \
  --env ALLOW_EMPTY_PASSWORD=yes \
  --mount type=bind,src=${DEFAULT_BIND_VOLUMES_ROOT}/DATA,dst=/bitnami/redis/
    ↪ data \
  -p ${redis_port}:${redis_port} \
  ${image}`;
  await docker_execute(param_string);
  cb();
}
```

docker_run_all

Crea y ejecuta varios contenedores creando un entorno de pruebas de pipelines:

1) Un contenedor plasmidnet glibc.
2) Un contenedor *redis*.
3) Un contenedor *dfast*.
4) Un contenedor *mmsegs2*.

```
async function docker_run_all(cb, dist_version=argv.dist_version, bind_code=argv
    ↪ .bind_code, bind_internal=argv.bind_internal) {
  let node = 'plasmid03';
  let bind = `${DEFAULT_BIND_VOLUMES_ROOT}/${node}`;
  await docker_stop(cb, node);
  await docker_run(cb, node, 'mmseq2', 9093, dist_version, bind, bind_code,
    ↪ bind_internal);
  await docker_server(cb, node);

  node = 'plasmid02';
  bind = `${DEFAULT_BIND_VOLUMES_ROOT}/${node}`;
  await docker_stop(cb, node);
  await docker_run(cb, node, 'dfast', 9092, dist_version, bind, bind_code,
    ↪ bind_internal);
  await docker_server(cb, node);

  node = 'plasmid01';
  bind = `${DEFAULT_BIND_VOLUMES_ROOT}`;
  await docker_stop(cb, node);
  await docker_run(cb, node, 'glibc', 9091, dist_version, bind, bind_code,
    ↪ bind_internal);
  await docker_server(cb, node);

  node = 'redis';
  await docker_stop(cb, node);
  await docker_run_redis(cb, node);
}

module.exports.docker_run = docker_run;
module.exports.docker_run_dev = docker_run_dev;
module.exports.docker_run_redis = docker_run_redis;
module.exports.docker_build = docker_build;
module.exports.docker_build_dev = docker_build_dev;
module.exports.docker_build_dfast = docker_build_dfast;
module.exports.docker_build_mmseq2 = docker_build_mmseq2;
module.exports.docker_build_glibc = docker_build_glibc;
module.exports.docker_rm = docker_rm;
module.exports.docker_sh = docker_sh;
module.exports.docker_stop = docker_stop;
module.exports.docker_server = docker_server;
module.exports.docker_quit = series(docker_stop, docker_rm);
```

docker_start_sh

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web. Además arranca en el contenedor una shell *sh* y la empareja con un terminal, de *screen*.

```
module.exports.docker_start_sh = series(docker_run, docker_sh);
```

docker_start_server

Crea y ejecuta el contenedor asociádole el identificador *node*, para la imagen identificada en el parámetro *pimage* y de código de versión *dist_version*. Se identifican también los directorios de *bind* internos y externos de logs, código y datos y el puerto de ejecución del servidor web. Además arranca en el contenedor el servidor de la aplicación (arrancado mediante la tarea por defecto de *gulp*).

```
module.exports.docker_start_server = series(docker_run, docker_server);
module.exports.docker_run_all = docker_run_all;
```

docker_build_all

Construye todos los tipos de docker asociándoles la versión *dist_version*: producción, dev, dfast, mmseq2

```
module.exports.docker_build_all = series(docker_build, docker_build_dev,
    ↪ docker_build_dfast, docker_build_mmseq2);
module.exports.shares = module.exports;
```

B.7.4. Otras automatizaciones

gulpfile_misc.js

En este módulo definimos automatizaciones adicionales.

```
const {require_dyn} = require('../proxy_modules');
const {series, parallel, src, dest, watch, symlink} = require('gulp');
const execa = require('execa');
const prompt = require('gulp-prompt');
const del = require('del');

const PLASMID_MOUNTS = '/Users/nandoide/PLASMIDBUILD';
//const PLASMIDBUILD = 'PLASMIDBUILD';
const PLASMIDBUILD_TGZ = './BACKUP/plasmidbuild.app.tar.gz';
const DEFAULT_GIT_MESSAGE = 'alpha WIP';
const GIT_ROOT = 'CODE/plasmidnet';
```

```
const argv = require('yargs')
  .default('message', DEFAULT_GIT_MESSAGE)
  .default('dir', 'CODE')
  .default('all', 'true')
  .argv;
```

zip_app

Realiza un backup (*tar.gz*) de todo el directorio de la aplicación.

```
async function zip_app(cb, all=argv.all){
  let logger = require_dyn("logger_app");
  //let path = 'app';
  let follow_sym = '';
  if (all === 'true') {
    follow_sym = '-h'
  }
  let cmd = `tar -czvf ./app/BACKUP/plasmidbuild.app.tar.gz --exclude app/BACKUP
    ↪ ${follow_sym} app`;
  try {
    const {stdout, stderr} = await execa.command(cmd, {cwd: '..'});
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

zip

Realiza un backup (*tar.gz*) del directorio *dir* (CODE, LOG, DATA)

```
async function zip(cb, dir=argv.dir){
  let logger = require_dyn("logger_app");
  let cmd = `tar -czvf ./BACKUP/plasmidbuild.${dir}.tar.gz -h ${dir}`;
  logger.info(cmd);
  try {
    const {stdout, stderr} = await execa.command(cmd);
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

git_add

Ejecuta el comando *git add* sobre el directorio raíz del código fuente.

```
async function git_add(cb){
  let logger = require_dyn("logger_app");
  try {
    const {stdout, stderr} = await execa('git', ['add', '.'], {cwd: GIT_ROOT});
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

git_commit

Ejecuta el comando *git commit -am* incluyendo el mensaje especificado en *message* sobre el directorio raíz del código fuente.

```
async function git_commit(cb, message=argv.message) {
  let logger = require_dyn("logger_app");
  try {
    const {stdout, stderr} = await execa('git', ['commit', '-am', message], {cwd
    ↪ : GIT_ROOT});
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

git_push

Ejecuta el comando *git push origin master* sobre el directorio raíz del código fuente.

```
async function git_push(cb) {
  let logger = require_dyn("logger_app");
  try {
    const {stdout, stderr} = await execa('git', ['push', 'origin', 'master'], {
    ↪ cwd: GIT_ROOT});
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
```

```
  }
}
```

git_execute

Ejecuta el comando *git* con los parámetros especificados en *param_string*.

```
async function git_execute(param_string) {
  let logger = require_dyn("logger_app");
  let params = param_string.split(/\s+/);
  logger.info(params);
  try {
    logger.info(`Git command ${params.join(' ')}`);
    const {stdout, stderr} = await execa('git', params);
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
}
```

git_credentials

Solicita al usuario sus credenciales *github* y las almacena en el entorno.

```
function git_credentials(cb) {
  return src( 'gulpfile.js' )
    .pipe(prompt.prompt({
      type: 'input',
      name: 'email',
      message: 'Which is your git email'
    }, function(res){
      process.env.git_email = res.email;
      console.log(res.email);
    }))
    .pipe(prompt.prompt({
      type: 'input',
      name: 'name',
      message: 'Which is your git username'
    }, function(res){
      process.env.git_name = res.name;
      console.log(res.name);
    }))
  cb();
}
```

git_config_email

Añade la dirección de correo de la cuenta de git (*email*) a la configuración global del sistema.

```
async function git_config_email(cb, email=process.env.git_email) {
  let param_string = `config --global user.email "${email}"`;
  await git_execute(param_string);
  cb();
}
```

git_config_name

Añade el usuario de la cuenta de git (*name*) a la configuración global del sistema.

```
async function git_config_name(cb, name=process.env.git_name) {
  let param_string = `config --global user.name "${name}"`;
  await git_execute(param_string);
  cb();
}
```

github_config

Configura la cuenta de github a parftir de los datos de credential.helper.

```
async function github_config(cb) {
  let param_string = `config --global credential.helper cache --timeout=3600`;
  await git_execute(param_string);
  cb();
}
```

test

Tarea de test para comprobar que el servidor responde.

```
async function test(cb){
  let logger = require_dyn("logger_app");
  try {
    const {stdout, stderr} = await execa('echo', ['gulp test task']);
    logger.info(stdout.split('\n'));
    logger.info(stderr.split('\n'));
  } catch (error) {
    logger.error(error.stderr.split('\n'));
    logger.error(error.message);
  }
  cb();
}
```

link_LOG

Genera el *symlink* del directorio de logs *LOG*.

```
function link_LOG() {
  return src(`${PLASMID_MOUNTS}/LOG`)
    .pipe(symlink(`.`));
}
```

link_LOG

Genera el *symlink* del directorio de datos *DATA*.

```
function link_DATA() {
  return src(`${PLASMID_MOUNTS}/DATA`)
    .pipe(symlink(`.`));
}
```

link_CODE

Genera el *symlink* del directorio de código fuente *CODE*.

```
function link_CODE() {
  return src(`${PLASMID_MOUNTS}/CODE`)
    .pipe(symlink(`.`));
}
```

link_BACKUP

Genera el *symlink* del directorio de backup *BACKUP*.

```
function link_BACKUP() {
  return src(`${PLASMID_MOUNTS}/BACKUP`)
    .pipe(symlink(`.`));
}
```

data_clean_dry

Simula el borrado del directorio *DATA* como verificación de ficheros afectados antes de un borrado real. No se borran: el directorio raíz, el directorio de la bd *plasmid_modules*, *INPUT*.

```
async function data_clean_dry(cb) {
  let logger = require_dyn("logger_app");
  const deleted = await del(['DATA/**', '!DATA', '!DATA/INPUT', '!DATA/
    ↪ plasmid_modules'], {dryRun: true});
  logger.info('Files and directories that would be deleted: ' + deleted.join('\n
    ↪ '));
}
```

data_clean

Borrado del directorio *DATA*. No se borran: el directorio raíz, el directorio de la bd plasmid_modules, *INPUT*.

```
async function data_clean(cb) {
  let logger = require_dyn("logger_app");
  const deleted = await del(['DATA/**', '!DATA', '!DATA/INPUT', '!DATA/
    ↪ plasmid_modules'], {dryRun: false});
  logger.info('Files and directories that have been deleted: ' + deleted.join('\
    ↪ n'));
}
```

log_clean_dry

Simula el borrado del directorio de logs *LOG* como verificación de ficheros afectados antes de un borrado real.

```
async function log_clean_dry(cb) {
  let logger = require_dyn("logger_app");
  const deleted = await del(['LOG/pipelines/*', 'LOG/web/*'], {dryRun: true});
  logger.info('Files and directories that would be deleted: ' + deleted.join('\n
    ↪ '));
}
```

log_clean

Borra el directorio de logs *LOG* como verificación de ficheros afectados antes de un borrado real.

```
async function log_clean(cb) {
  let logger = require_dyn("logger_app");
  const deleted = await del(['LOG/pipelines/*', 'LOG/web/*'], {dryRun: false});
  logger.info('Files and directories that have been deleted: ' + deleted.join('\
    ↪ n'));
}
```

```
module.exports = {test: test, zip_app: zip_app, zip: zip, link_BACKUP:
    ↪ link_BACKUP, link_LOG: link_LOG, link_DATA: link_DATA, link_CODE:
    ↪ link_CODE};
module.exports.git_commit = git_commit;
module.exports.git_push = git_push;
module.exports.git_add = git_add;
module.exports.git_credentials = git_credentials;
```

git_all

Ejecuta en serie los comandos *git*: *add*, *commit* y *push*, asociando al *commit* el mensaje espevificado en *message*.

```
module.exports.git_all = series(git_add, git_commit, git_push);
module.exports.git_config = series(git_credentials, git_config_email,
    ↪ git_config_name);
module.exports.github_config = github_config;
module.exports.data_clean = data_clean;
module.exports.data_clean_dry = data_clean_dry;
module.exports.log_clean = log_clean;
module.exports.log_clean_dry = log_clean_dry;
module.exports.shares = module.exports;
```