UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Desarrollo de una plataforma para la gestión logística de ONGs
1718_153_ISSITI

Autor:Alejandro Moreno Jiménez

Tutor: Simone Santini

Departamento de Ingeniería informática

Escuela politécnica Superior

Universidad Autónoma de madrid

Junio 2018

Resumen

Desarrollo de una App con el nombre de **koriNGO** para facilitar la gestión de los informes de los usuarios actuales de la ONG Koricancha en Perú. Además de con un comedor social, Koricancha cuenta con un programa de apadrinamientos de personas con discapacidad. Hoy en día existen 90 usuarios en el programa de los cuales 77 reciben un apoyo económico mensual de sus respectivos padrinos para la mejora de calidad de vida del ahijado en cuestión. Además de estas 77 personas ya apadrinadas, Koricancha cuenta con un una lista de entre 10 y 20 personas. Para Koricancha es importante conocer lo mejor posible la situación de cada usuario y tener esta información lo más actualizada posible. Los ahijados pueden ser cambiados en el caso de no cumplir un mínimo de reglas y entrar en el programa los que se encuentran en lista de espera. Una vez al año un equipo de voluntarios va un periodo medio de unas dos semanas a este proyecto. Una de las tareas más importantes de estos voluntarios es la de realizar visitas a las familias que están en el programa para actualizar el informe en cuestión o conocer nuevas familias que quieran ser ahijados en el futuro. Es muy importante que los voluntarios realicen las visitas a los niños o niñas de la manera más justa e inteligente pues el número de fichas al año que se crean o se actualiza no llega al veinte por ciento del total. Hoy en día esta gestión de visitas se realiza de manera manual y en algunas ocasiones no se ha visitado a familias cuya visita era importante. Con korinNGO se pretende gestionar las visitas más preferenciales según un algoritmo teniendo en cuenta los requerimientos de algunos voluntarios experimentados para que el orden y la prioridad de estas sean lo más justo e inteligente.

Palabras clave

Desarrollo Web, Cooperación al desarrollo, Visualización de data, MongoDB, JavaScript, Redux, React, Koricancha

Agradecimientos

Para empezar quiero agradecer esta oportunidad a Redradix. Pocas empresas con tan buen nivel son capaces de de hacer y practicar la solidaridad y a la vez de ayudarme y darme la formación que he recibido. Especial agradecimiento al maestro Elías Alonso que me ha tenido que aguantar todos estos meses con una improba paciencia fuera de todo lugar.

Muchas gracias a mi tutor Simone Santini. No es el primero proyecto que hago con él y la verdad que ha sido un placer tenerlo tanto como tutor como de profesor.

Por supuesto agradecer a mi familia, desde la de sangre a la que se elige, por estar todo este tiempo empujando de una o otra manera para que todos estos años acaben de la mejor manera. De todos estos familiares quiero agradecer en especial al mejor de los padres que en numerosas ocasiones ha demostrado tener más energía y pasión que nadie.

Gracias al equipo Koricancha por intentar en cada momento dibujar una sonrisa nueva. Cada día agradezco haber conocido a Sonia Lolo y haberme ensañado otra manera de viajar y de ver las cosas. Gracias a todos esos padrinos que ponen confianza ciega en los que con tan poco consigue tanto. Gracias a toda esa gente que de una forma o de otra ayuda a que todo este proyecto siga adelante.

No estaría escribiendo estas líneas si mi familia peruana no me hubiese dejado formar parte de sus vidas. Mil gracias por dejarme compartir tanto en tan poco tiempo, gracias por dejarme comer en vuestra mesa, gracias por dejarme dormir en vuestra casa y gracias por haberme dado una lección de las que no se olvidan. Para mi, lo que os merecéis no tiene precio.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1.Introducción

A día de hoy se aprecian una serie de problemas en las ONGs en general debido a la falta de **presupuesto** y la **poca rentabilidad** que pueden obtener otras entidades de ellas. En este caso, el estudio del problema se ha analizado siguiendo la referencia de la ONG Koricancha[1].

Koricancha es una ONG con sede en España colabora desde el año 2003 con familias humildes y pobres en la zona de Chiclayo al norte de Perú. Koricancha cuenta con **dos proyectos principales**. Tiene **un comedor social** y **dos escuelas para personas con discapacidad**. Al comedor social van una serie de familias a desayunar todos los días y comer tres veces por semana. Este comedor está financiado por un grupo de padrinos en España que dona una cantidad mensual. Las escuelas para personas con discapacidad tratan de dar clase a todos los usuarios del programa, aunque no es posible que todos se beneficien de estos centros sobre todo debido al espacio limitado y a la incapacidad de algunos de ellos de no tener resuelto el problema de la movilidad a la hora de trasladarse a ellas.

El programa a día de hoy cuenta con **77 apadrinados** y **35 plazas** disponibles entre las dos escuelas. Cabe destacar que en las escuelas además de realizar clase todos los días lectivos por la mañana se imparte terapia física y de lenguaje a lo largo de toda la jornada. Al no poder abastecer a todos los ahijados en las escuelas se facilita una **ayuda** para las familias que no se benefician de estas. Esta ayuda se traduce en una **cuantía económica mensual** para que las familias puedan tener la facilidad de dar una mejor calidad de vida al apadrinado en cuestión del programa ya sea comprando medicamentos, útiles o apoyando las diversas terapias. Las familias siempre tienen que justificar con facturas o comprobantes en que se invierte el dinero que reciben. Cada ahijado del programa tiene asignado un padrino externo que abona una cuantía mensual de 20€. Dependiendo de las necesidades de cada familia, Koricancha intenta repartir el monto económico de la manera más justa. Por ejemplo, si dos padrinos envían los 20€ para sus dos ahijados pero uno de ellos necesite más apoyo que otro se podría repartir de tal manera que a uno le llegasen 25€ y al otro 15€. Este es uno de los motivos por los que se **intenta conocer lo mejor posible las situaciones de cada familia**. Detrás de cada usuario del programa hay una ficha que intenta estar lo **más actualizada** posible.

A día de hoy cada ficha es un documento word que ha sido rellenada por un voluntario que ha entrevistado la familia en Perú. Hay que tener en cuenta que después de quince años hay muchas fichas que tienen que ser actualizadas además de las nuevas incorporaciones al programa. Saber la prioridad de las próximas visitas para actualizar las fichas del programa no es una tarea fácil y hoy en día esta prioridad se hace manera **manual**. Hay muchos factores a la hora de establecer esta actualización de fichas. Entre ellos se pueden destacar los siguientes:

[1] http://koricancha.org

- Tiempo desde que no se actualiza la ficha en cuestión.

- Encarecimiento de la terapia, tratamientos o cualquier tipo de útil físico (ej: pañales, útiles escolares ...)

- Problemas destacables en la familia

    - Maltrato familiar
    - Escasos ingresos económicos de la familia
    - Mala utilización de dinero

Se cree que a pesar de haber hecho el estudio del problema con una ONG en concreto, la solución se puede aplicar a otras ONGs. Hay muchas ONGs que tienen una base de datos de fichas que tienen que actualizar en función de diferentes parámetros.

La empresa de desarrollo web **Redradix**[2] ha dado la oportunidad de realizar este proyecto al estudiante y con ello formándole durante el desarrollo de koriNGO con un tutor personal. Redradix es una empresa puntera en desarrollo web especializada en diseño, maquetación en HTML y CSS y programación en javaScript.

KoriNGO es una App basada totalmente en desarrollo web para facilitar la gestión de los informes que los voluntarios tienen que hacer cuando visitan a las familias de Koricancha. El proyecto entero se puede desglosar en los siguientes apartados:

- Planteamiento de la **metodología** a utilizar. Al principio se planteó el uso de diferentes metodologías y al no haber un cliente de por medio se optó por utilizar **cascada**. No se ha tenido que iterar apenas ya que se ha dedicado gran parte del tiempo al planteamiento del proyecto.

- Recogida de requerimientos: Ya sea viajando a Perú o hablando con otros voluntarios se ha dedicado más de un mes a entender bien el problema y las necesidades del proyecto. Una vez en España se ha tenido que plasmar todos los requerimientos que se han considerado factibles.

- Diseño de la **API REST**[3]. Antes de empezar a programar se ha diseñado la API REST intentado abarcar todos los puntos para el desarrollo posterior de esta.

[2] http://redradix.com/
[3] Ver en en el gloasario

- Desarrollo de la API REST. Se ha programado el back-end de KoriNGO utilizando el framework[4] Express y usando javascript y nodeJS como lenguajes de programación. Como base de datos se ha optado por integrar MongoDb en el proyecto.
- Desarrollo del Front-end. Se ha utilizado React y Redux para establecer la lógica del frontend. Para facilitar la tarea de la realización del HTML y CSS se ha utilizado el Kit de componentes MaterialUI[5]. Esta librería está pensada para una fácil integración con React.

Más adelante se entrará más en detalle en estos puntos.

[4] Ver en el glosario
[5] https://material-ui-next.com/

1.1Motivación

El estudio de las necesidades de los problemas de Koricancha se ha realizado de manera física y personal viajando el estudiante al proyecto Koricancha al Norte de Perú. Ya había ido tres veces al proyecto en el pasado. Aprovechando que tenía que hacer el trabajo de fin de grado decidió ir en enero al proyecto Koricancha y se propuso al tutor el tema. Este último viaje ha sido en parte diferente ya que se ha ido con una predisposición más académica y de alguna forma más objetiva y se ha intentado centrar en la recolección de los requisitos y estudiar las necesidades que iban a hacer falta a la hora de desarrollar el proyecto completo empezando desde cero. Antes de viajar el estudiante se reunió con el tutor del trabajo e hizo la entrevista de trabajo. Por otra parte a a Redradix le pareció bien la idea de formar al estudiante con este proyecto ya que reunía todas las características de algo real. El tutor del trabajo dio al estudiante una serie de pautas y consejos para poder reunir los requerimientos de la manera más completa y posible para empezar a trabajar una vez llegado a España.

Ya se ha comentado que los proyectos solidarios no cuentan con la mejor cobertura debido a la poca rentabilidad económica que se puede obtener de ellos. Este fue el principal motivo por el que el estudiante accedió a realizar este trabajo de fin de grado.

Se puede destacar que además en Perú se ha encontrado mucha más desigualdad de la que puede haber en España. También se ha percibido mucha diferencia entre la parte Norte de Perú y la parte Sur. Esto es debido a que el turismo en el Sur es muy potente a nivel mundial. En cambio, en el Norte apenas hay turismo. En Chiclayo, la ciudad dónde Koricancha trabaja, el estudiante ha sido capaz de ver tan sólo un mochilero en cuatro años diferentes.(falta de turismo)

Además de las pocas ayudas detectadas en el mundo de la cooperación al desarrollo en general, se ha detectado muy poco interés en cooperar con el mundo de la discapacidad. Esto es debido que los problemas que le afectan a las personas que, de una manera o de otra están en este mundo, son una minoría, y consecuentemente hay menos concienciación de la debida y una gran invisibilidad.

Se ha mencionado antes que uno de los principales problemas que tiene Koricancha es controlar lo mejor posible la situación actual de cada familia del programa. Para esto hay que enviar voluntarios p que actualicen o creen los informes de cada usuario del programa. Hoy en día a cada usuario del programa le corresponde una ficha estática que está guardada en formato .doc.[6] Cada vez que un equipo de voluntarios va al proyecto se deciden las próximas visitas de manera manual. Este hecho acarrea una serie de problemas a destacar:

- El orden de las visitas no es justo en numerosas ocasiones.
    - Usuarios en lista de espera pueden quedarse sin visita.
    - Con el paso del tiempo algunas fichas se pueden quedar muy desactualizadas.
    - Usuarios con problemas muy importantes pueden ser no visitados.
    - Algunos usuarios del programa pueden ser visitados con mucha más frecuencia que otros.

[6] Ver en el anexo

- Usuarios en lista de espera que no tienen ficha pueden abandonar el programa por no tener un feedback de algún voluntario.

- Problemas con el formato de guardado de información

    - Muy complicado plasmar los cambios de un usuario en el tiempo debido al formato de la ficha actual.
    - Con el formato de la ficha actual se puede perder mucha informaciń muy valiosa ya que hay mucha información que se sobreescribe.
    - Es muy complicado filtrar las fichas por información de interés para encontrar ciertos usuarios.
    - Las fichas se encuentran en soportes como discos duros o dropbox por lo que es fácil perder información y no siempre está unificada.

Podemos terminar con el apartado de la motivación informando que el estudiante ha realizado un curso presencial de treinta horas de javaScript, React y Redux financiado por Redradix. El material del curso se puede encontrar online[7].

[7] https://github.com/redradix/curso-es6-react

1.2 Objetivos

Los objetivos del proyecto son los siguientes:

- **Unificación de data en una App Online**: Se quiere centralizar todos los informes que contenga la ONG Koricancha en una única plataforma online. Hasta ahora mucha información puede estar una plataforma o en otra según los diferentes voluntarios la tengan.

- **Creación de una base de datos**: Ahora mismo no existe una base de datos con los usuarios del programa y los voluntarios. Lo más parecido a esto es una carpeta con todas fichas existentes. Se pretende crear una base de datos unificada de tal manera que toda la información que suba cada voluntario tenga coherencia con la que ya había.

- **Facilitación de filtros**: Ahora, buscar voluntarios o usuarios es una tarea que no se hace de manera automática. Si por ejemplo, se quiere buscar un voluntario cuya profesión sea la de profesor o un usuario del programa cuya discapacidad sea síndrome de Down, habría que preguntarlo o saberlo directamente. Es muy fácil no poder satisfacer una búsqueda con unos requisitos. Ahora se pretende facilitar el camino para poder encontrar los usuarios o voluntarios en función de algunos filtros.

- **Priorización de visitas**: De igual manera, la priorización de las búsquedas se hace manera manual. Es muy sencillo no ordenar las visitas a las diferentes familias para actualizar o crear nuevas fichas de usuario de la manera correcta. Ahora existirá un apartado de próximas visitas asociados a cada ficha en la que cada una tendrá un **score** asignado en función de algunos parámetros que se consideren importantes.

- **Cambio de soporte de Koricancha:** Se apuesta por el cambio de la utilización del software por lo que hay que formar y testear el nuevo software para que los voluntarios de Koricancha se sientan cómodos con la nueva App.

- **Backup[8] del historial**: Con cómo se ha hecho hasta ahora, es prácticamente imposible saber de los cambios que han experimentado los usuarios en la línea del tiempo. Hasta ahora, si se actualiza una ficha haciendo una visita que ya se hizo en el pasado, los nuevos datos sobreescriben los que ya había de tal manera que se va perdiendo información pasada que en muchas ocasiones puede ser muy valiosa. Ahora se pretende guardar el historial de los cambios en los que suponga una ventaja.

- **Posible reutilización con otras entidades**: Al principio nos centraremos en Koricancha pero lo ideal será poder reutilizar el sistema para que otras entidades puedan usarlo.

[8] Ver en el glosario

2 Estado del arte

En cinco años colaborando con Koricancha no se ha encontrado un software tan personalizado para el fin que se quiere conseguir. No obstante los procedimientos y tecnologías a utilizar ya se llevan usando bastantes años. Ya se ha mencionado que la App realizada es web. Los apartados más destacados a mencionar se citarán a continuación.

2.1 Redradix[9]

Antes se ha mencionado que el estudiante ha realizado el proyecto en la empresa de desarrollo web Redradix. Esta es una empresa con seis años de vida especializada en desarrollo web y cursos relacionados con esta tecnología.

Todos los aspectos que el estudiante ha hecho van íntimamente ligados con el trabajo que realiza Redradix día a día. Esto se debe a que todo el tiempo que el estudiante ha estado realizando este proyecto ha sido tiempo de formación para que el estudiante sea un futuro trabajador de Redradix.
Con más de treinta trabajadores contratados en los últimos años y proyectos realizados a grandes compañías como GMV, Telefónica, Canal + y muchas más de las mismas dimensiones, Redradix ha demostrado día a día ser una de las compañías líderes en el sector en España.

Como ya se mencionó, el estudiante en las primeras semanas realizó un curso de desarrollo web con Redradix.

2.2 Metodología

Entre el tutor de trabajo y el alumno se ha realizado un estudio de qué metodología era más oportuna utilizar en este proyecto. A la hora de elegir una se ha tenido en cuenta, sobre todo, que no había un cliente como tal.Todos y cada una de las decisiones se han tomado entre los tutores y el alumno. Esto se debe a que no ha habido nadie incluyendo requisitos en cada fase. Por eso no se han considerado metodologías ágiles y se ha realizado todo el proceso en **cascada**.

La metodología en cascada es un modelo lineal de diseño de software que emplea un proceso de diseño secuencial. El desarrollo fluye secuencialmente desde el punto inicial hasta el punto final, con varias etapas diferentes: Definición de requerimientos, el análisis del diseño del software, la implementación y las pruebas unitarias, la integración y la prueba del sistema y por último la operación y el mantenimiento del sistema.

Se ha intentado llevar a cabo cada acción en su fase correspondiente sin tener que volver a atrás en cada momento. Para esto se ha tenido que analizar y desarrollar detenidamente cada uno de las fases. Se va a profundizar con cada una de las diferentes fases:

- **Definición de requerimientos**: Desde el mes del viaje terminado hasta las dos primeras semanas se ha realizado el análisis de requerimientos. En esta fase se ha tenido especial

[9] https://redradix.com/

atención a la hora de intentar reunir cuantos más requerimientos mejor. Se preferido contar con más requerimientos que con menos.

- **Análisis del diseño del software**: Una vez realizado la definición de los requerimientos, se ha procedido a hacer un análisis del diseño del software. Se ha planteado detenidamente qué software era más conveniente utilizar en cada fase del proyecto. Teniendo en cuenta que la empresa en la que se ha desarrollado el proyecto está especializada en desarrollo web no ha sido una tarea difícil encontrar un software adecuado para comenzar el desarrollo. Ya se ha mencionado que uno de los objetivos de Redradix con este periodo de pruebas es el de formar estudiante con su tecnología para poder contar con su contratación en el futuro. La tecnología a utilizar se detallará más adelante. Lo primero que se ha diseñado antes de empezar a desarrollar ha sido la API Rest.

- **Implementación y pruebas unitarias**: Una vez diseñada la API Rest se ha procedido a su implementación. Cuando ha sido finalizada y el tutor ha dado el visto bueno se han procedido a realizar las pruebas unitarias de cada módulo independiente. Cuando han pasado todas las pruebas se ha implementado el frontend y de igual manera se han realizado las pruebas unitarias una vez el tutor ha dado el visto bueno.

- **Integración y prueba del sistema**: Una vez hemos contado con el backend y el frontend hemos procedido a integrar junto. Cuando lo hemos tenido, se ha realizado una prueba del sistema lo más exhaustiva posible. También se ha dejado a los voluntarios utilizar la App por un periodo corto de tiempo para ver si todo funcionaba correctamente desde un punto de vista objetivo.

- **Operación y mantenimiento**: Cuando ya se ha tenido toda la App funcionando ha dejado a algunos voluntarios utilizar la App por un periodo largo de tiempo para ver si tiene algún error en lo próximos meses. De esta manera se espera encontrar y corregir algunos errores que no se han tenido en cuenta hasta ahora.

Aquí se puede apreciar de manera visual una representación de cómo funciona la metodología en cascada. Se observa que después de cada parte del proceso no se vuelve a la anterior. Por su puesto que conseguir esto al pie de la letra es muy complicado y en algún momento se ha tenido que dar un paso atrás. En el caso de este proyecto nunca han supuesto cambios de gran escala y nunca ha habido que modificar la estructura del proyecto.

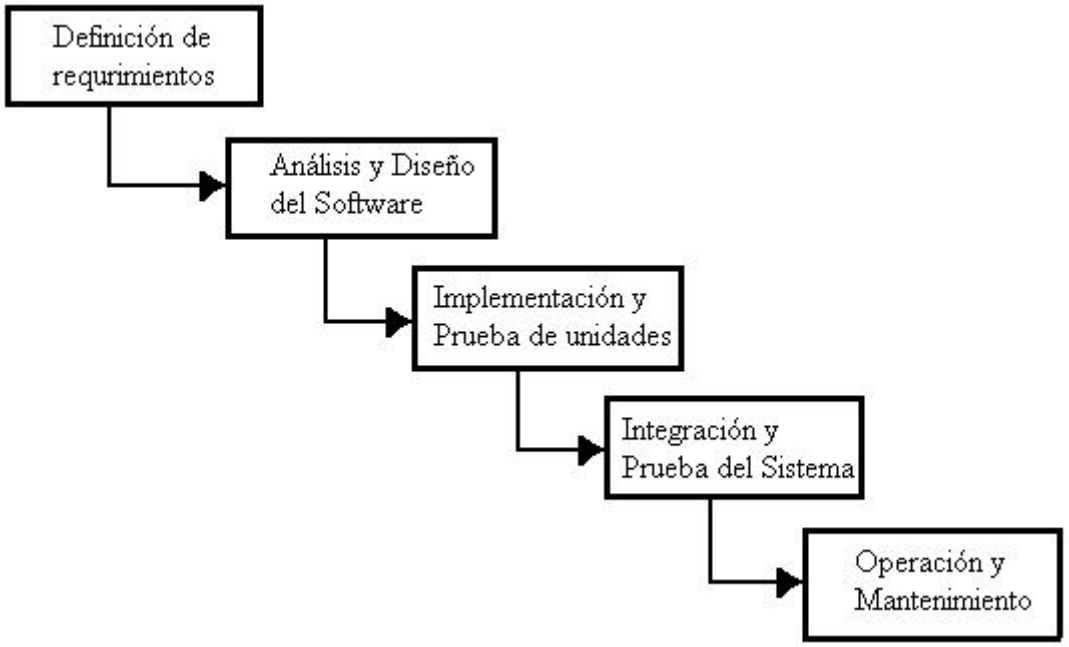

Imagen 1.Metodología cascada. Imagen obtenida en http://primermodelo.blogspot.com.es/

2.3 Backend

Uno de los grandes apartados que de este proyecto que se puede destacar es el backend. La parte del backend es la parte que se encuentra en el lado del servidor. En este proyecto podemos identificar diferentes secciones aquí.

2.3.1 API REST

Una API es un conjunto de comandos, funciones y protocolos que permiten al desarrollador crear programas informáticos sin tener que empezar desde cero.

Dentro de las APIs en este proyecto se va a utilizar **API REST.** REST es una arquitectura de desarrollo web que se apoya se apoya totalmente en el estándar **HTTP**. Esta arquitectura permite crear servicios y aplicaciones que pueden ser utilizadas por cualquier dispositivo que entienda el protocolo HTTP.

HTTP es un protocolo por el cual podemos realizar una petición de datos y recursos. En la base de cualquier intercambio de datos en la Web y un protocolo de estructura cliente-servidor. Esto quiere decir que la petición de datos o recursos es iniciada por un cliente que recibirá estos recursos y datos desde el servidor. Normalmente este cliente es el navegador web. La solicitud se realiza con el método Request y la respuesta con los datos se hacer con el método Response. Aquí se puede apreciar de manera gráfica y muy simplificada como el cliente realiza un Request al servidor y este después de realizar las operaciones pertinentes hace un Response devolviendo un texto plano.

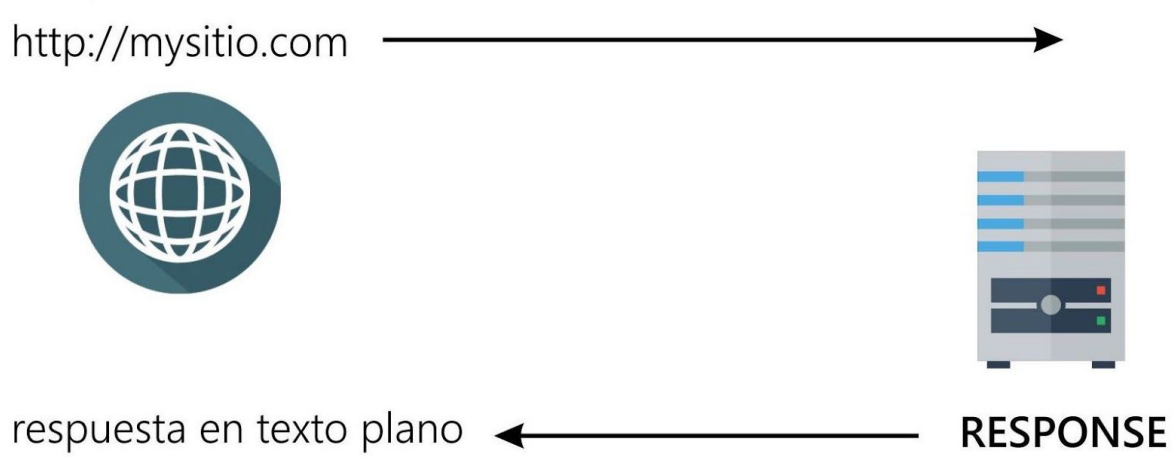

Imagen 2. Representación de HTTP. Fuente: Imagen obtenida utilizando esta referencia[10]

REST tiene una serie de características. Esta arquitectura es más simple y convencional que cualquier alternativa que ha sido usada en los últimos años como puede ser SOAP o XML-RPC. Se puede considerar REST como un framework para facilitar la vida al programador respetando el protocolo HTTP. A pesar de haber sido creada en el año 2000 es la arquitectura más natural y estándar cuando estás trabajando para servicios orientados a internet. Algunos de los conceptos más importante e interesantes e importantes en esta arquitectura son los siguientes:

- **Estado**. Este estado es toda la información que se requiere para mostrar todos los datos solicitados. Uno de las reglas de oro es que en en el protocolo REST este estado siempre está en cada cliente. De esta manera los servidores no se saturan de la misma manera que en lo que el estado se guarda en el servidor y además son totalmente escalables.

- **URL**: Cuando desarrollamos una página web o una aplicación web las URLs nos permiten acceder a cada una de las páginas.

- **Recursos**: Se trata de toda la información a la que se quiere acceder. Esta información puede querer ser mostrada, modificada o eliminada. Las URLs tienen esta forma:
  {protocolo}://{dominio o hostname}[:puerto (opcional)]/{ruta del recurso}?{consulta de filtrado}
  Como sección a destacar se puede ver la ruta del recurso, por la cual se accederá a un recurso en concreto y también podemos ver los filtros que podemos aplicar al recurso en cuestión.

- **Métodos**: Para manipular los recursos REST cuenta con una serie de acciones a las que se llaman métodos. Los métodos más importantes son los siguientes:

  - **GET**: Para consultar y leer recursos

  - **POST**: Para crear recursos

[10] https://somostechies.com/content/images/2016/02/h1-1-1.jpg

    - **PUT**: Para editar recursos
    - **DELETE**: Para eliminar recursos.
    - **PATCH**: Para editar partes concretas de un recurso.

- **Códigos de estado**: Conocidos como Status Code es un número que acompaña la Response e indica si la operación se ha realizado con éxito o no. Los códigos de estado que se han considerado más importantes para este proyecto son los siguientes:
    - **200 OK**: La solicitud ha tenido éxito. El significado de un éxito varía dependiendo del método
    - **201 Created**: La solicitud ha tenido éxito y se ha creado un nuevo recurso como resultado de ello.
    - **400 Bad Request**: Esta respuesta significa que el servidor no pudo interpretar la solicitud dada una sintaxis inválida.
    - **401 Unauthorized**: Es necesario autenticar para obtener la respuesta solicitada. Esta es similar a 403, pero en este caso, autenticación es posible.
    - **403 Forbidden**: El cliente no posee los permisos necesarios para cierto contenido, por lo que el servidor está rechazando otorgar una respuesta apropiada.
    - **404 Not Found**: El servidor no pudo encontrar el contenido solicitado.

Todos los puntos mencionados anteriormente son los que se han considerado remarcables a la hora de construir el proyecto llevado a cabo.

Para ir testeando las diferentes llamadas a la API y sus respectivas respuestas se ha utilizado el plugin de Google Postman[11]. Este Plugin facilita mucho la tarea a la hora de hacer pruebas inmediatas. No es igual de exhaustivo que un test de pruebas pero es mucho más cómodo cuando se quiere realizar una prueba puntual.

[11] https://www.getpostman.com/

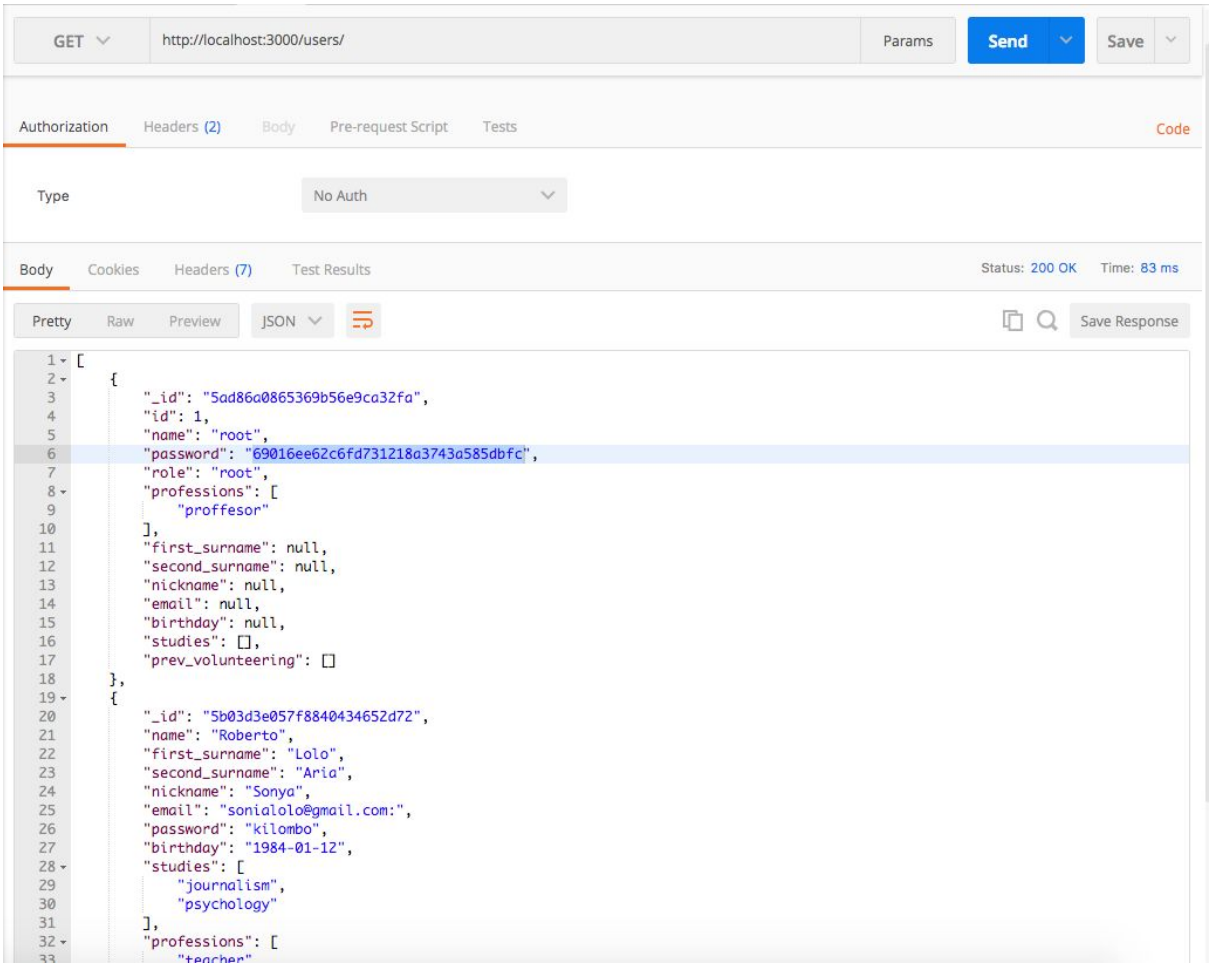

Imagen 3. Postman. Imagen tomada de una petición que se realiza a /users del proyecto real

En la captura se aprecia lo fácil que supone simular cualquier petición que se realice en tu aplicación. En este caso se ha realizado un GET sobre el recurso /users. En el caso de tener un response se puede añadir sin ningún tipo de problema. Las cabeceras deseadas están incluídas en la parte del heder. En el caso de KoriNGO, para realizar las diferentes pruebas, se ha añadido siempre la cabecera con un token válido de autentificación.

2.3.2 Blueprint

Para proceder el diseño de la API se ha utilizado la herramienta Blueprint[12]. Con esta herramienta se ha agilizado la realización y la comprensión de nuestra API. Antes de empezar a diseñar se ha planteado detalladamente los diferentes objetivos para que al continuar con el desarrollo de la API todo estuviera bien estructurado y no hubiese que introducir nueva funcionalidad.

BluePrint traduce un código, que es está está escrito siguiendo unas reglas propias, a un documento .html donde finalmente está está representada de una manera legible la API que se quiere implementar más adelante.

[12] https://apiblueprint.org/

Cuando se explique cómo se ha realizado el diseño de la API se profundizará más adentro en todo el procedimiento poniendo incluso ejemplos visuales

2.3.3 JavaScript

En este proyecto se ha decidido programar todo basado en JavaScript[13]. Este lenguaje es perfecto ya que nuestra App va a ser desarrollada totalmente en web.

JavaScript es un lenguaje de programación que te permite realizar actividades complejas en una página web de manera dinámica. Además está perfectamente integrado con HTML y CSS de los cuales ya se hablará en la parte del frontend. JavaScript es muy conocido porque se ejecuta en el cliente. Bien es cierto que ahora se está hablando del backend pero JavaScript ofrece la posibilidad de poder ejecutarse siempre y cuando usemos NodeJS. JavaScript es uno de los lenguajes más populares ahora en este mundo debido a su compatibilidad web y a su facilidad de uso.

2.3.4 NodeJS

NodeJs[14] es un entorno Javascript del lado del servidor, basado en eventos. Esto último quiere decir la este lenguaje se ejecuta de manera asíncrona. Node ejecuta javascript utilizando el motor V8, desarrollado por Google para uso de su navegador Chrome. Aprovechando el motor V8 permite a Node proporciona un entorno de ejecución del lado del servidor que compila y ejecuta javascript a velocidades increíbles.

2.3.5 NPM[15]

Node Package Manager más conocido como NPM es un gestor de paquetes que facilita el considerablemente el trabajo con Node ya que a través de él podemos tener a nuestra disposición cualquier librería disponible con tan solo una línea de código. NPM ayuda a administrar los paquetes, distribuir paquetes y agregar dependendencias de una manera fácil y sencilla.

En el caso de no tener Node.js instalado se puede hacer desde la línea de comando utilizando:

sudo apt-get install nodejs

Una vez instalado Node.js en el equipo se puede proceder a instalar con el siguiente comando:

sudo apt-get install npm

A partir de este momento se pueden instalar paquetes de manera local en tu proyecto o global en tu proyecto escribiendo en la consola:

npm install nombre_paquete@version

[13] https://javascript.com/
[14] https://nodejs.org/en/
[15] https://www.npmjs.com/

Dependiendo del paquete instalado se podrán acceder a una serie de comandos utilizando también NPM. Para saber cuales son las posibilidades conviene leer y estudiar todos los paquetes instalados.

2.3.6 Eslint[16]

ESLint es un programa que se encarga de revisar el código escrito y es capaz de señalar errores y posibles bugs que podemos corregir para mejorar nuestros programas.

Se puede instalar Eslint desde NPM. Una vez instalas Eslint hay que configurar el archivo .eslintrc.json con una serie de reglas para que de alguna manera el desarrollador se advertido cuando no escribe el código según establecen sus reglas en .eslintrc.json. El mismo archivo de reglas que se usa en todos los proyectos de RedRadix ha sido utilizado con el fin de que el estudiante se acostumbre a la sintaxis de la empresa. Aquí se puede observar un pequeño ejemplo de la forma que tiene el .eslintrc.json:

[16] https://eslint.org/

```
{
    "extends" : "eslint:recommended",
    "root": true,
    "rules": {
        "camelcase": 2,
        "space-infix-ops": 2
    }
}
```

La parte que se ha considerado más importante es la de las reglas. Detrás de cada reglas hay un número que puede tener tres valores del 0 al 2:

- 0. Deshabilitaría la regla
- 1. Genera un warning
- 2. Una violación supondría un error.

En este caso saltaría un error en el caso de que las variables no siguieran el estilo camelCase o que se dejasen espacio entre los operadores.

Se ha considerado importante utilizar un editor de código que sea lo más cómodo y fácil de integrar con Eslint. En el caso de este proyecto se ha utilizado Visual Studio Studio. Desde aquí se se ha podido configurar Eslint de una manera muy sencilla y se han visto los errores y warning a tiempo real mientras se escribía el código. Otra posibilidad Eslint desde consola y ver todos los errores y warnings en esta.

2.3.7 Visual Studio Code[17]

Cómo editor de texto se ha procedido a utilizar Visual Studio Code. Este editor lo usa prácticamente todo el mundo en RedRadix por lo que a la hora de solucionar alguna duda no ha habido problema. Además tiene un instalador desde el mismo editor gráfico y muy sencillo de utilizar. Ya se ha mencionado antes la facilidad de uso de Eslint a tiempo real desde este editor.

2.3.5 Express

Express es el framework más popular en NodeJs. Este framework destaca por su robustez, sencillez, flexibilidad y facilidad a la hora de usarlo. Uno de los grandes motivos por los motivos por los que fue escogido es la potencia y facilidad del uso de los middlewares.

Un middleware es un software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, o paquetes de programas, redes, hardware y/o sistemas operativos. En el caso de KoriNGO facilita mucho la tarea a la hora de autentificar usuarios.

[17] https://code.visualstudio.com/

2.3.6 MongoDB

Para almacenar toda la información a modo de base de datos se ha utilizado MongoDB[18]. MongoDB es una base de datos orientada a documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en formato JSON.

Una de las mayores diferencias que tiene MongoDb respecto a las bases de datos relacionales, es que no es necesario seguir un esquema predefinido. Los elementos de una colección pueden tener esquemas diferentes. En MongoDB existen lo que llamamos databases y estas a su vez contienen colecciones. Estas colecciones en nuestro caso son las que asociamos con los recursos mencionados anteriormente. Podemos tener una colección "usuarios" con esta forma:

```
[
{
  Nombre: "Pedro",
  Apellidos: "Martínez Campo",
  Edad: 22,
  Aficiones: ["fútbol","tenis","ciclismo"],
  Amigos: [
   {
     Nombre:"María",
     Edad:22
   },
   {
     Nombre:"Luis",
     Edad:28
   }
  ]
}
]
```

Se puede observar que cada elemento de la colección es un elemento JSON. A diferencia de las bases de datos relacionales se puede añadir sin ningún problema otro elemento a la colección con este aspecto:

```
{
  Nombre: "Luis",
  Estudios: "Administración y Dirección de Empresas",
  Amigos:12
}
```

[18] https://www.mongodb.com/

Una vez añadido este elemento nos quedaría la colección de la siguiente manera:

```
[
 {
   Nombre: "Pedro",
   Apellidos: "Martínez Campo",
   Edad: 22,
   Aficiones: ["fútbol","tenis","ciclismo"],
   Amigos: [
    {
      Nombre:"María",
      Edad:22
    },
    {
      Nombre:"Luis",
      Edad:28
    }
   ]
 },
 {
   Nombre: "Luis",
   Estudios: "Administración y Dirección de Empresas",
   Amigos:12
 }
]
```

Además de las ventajas mencionadas MongoDb es compatible con todas las tecnologías a utilizar en nuestro desarrollo.

2.3.6 JSON Web Token

Para contar con una validación de acceso de los diferentes usuarios en la aplicación se ha utilizado JSON Web Token[19].

JWT es un estándar basado en JSON creado para la creación de tokens de acceso que permiten la propagación de identidad y de privilegios. Por ejemplo, si un usuario se logea en el sistema el servidor generaría un token para este usuario. Este token tendrá información relevante de quien es ese usuario. En este caso el usuario podría tener privilegios de usuario normal de administrador. Una vez que el cliente tiene este token podría mostrar quien es cada vez que hace una petición al servidor y este en función de quien sea podría responder de una manera o de otra. Este token está firmado por una clave, también llamada llave, que se encuentra en el servidor. Es por esto por lo que tanto el cliente como el servidor pueden verificar si el token es válido o no.

Los JWT normalmente están formados por tres partes:

[19] https://jwt.io/

- **El encabezado o header:** Este identifica qué algoritmo se se está utilizando para generar la firma. Podría tener la siguiente forma:

  header = '{"alg":"HS256","typ":"JWT"}'

  En este caso indica que que el token está formado utilizando el algoritmo HMAC-SHA256.

- **El contenido o payload**: Aquí se encuentra la información que se desee que se encuentre en el token. Puede tener esta forma.

  payload = '{"loggedInAs":"admin","id":1422779638}'

  En este caso se observa que el usuario es administrador y tiene el id 1422779638. También se podrían meter tantos campos como se quisieran.

- **La firma o signature**: La firma está calculada codificando el encabezamiento y el contenido, concatenando ambas partes con un punto como separador. El contenido se traduce a base64url La forma es de la siguiente manera:

```
key = 'secretkey'
unsignedToken=encodeBase64Url(header)+'.'+
encodeBase64Url(payload)
signature = HMAC-SHA256(key, unsignedToken)
```

En el token final se juntan las tres partes ya mencionadas utilizando puntos de la siguiente manera:

```
token=encodeBase64Url(header)+'.'+encodeBase64Url(payload)+'.
'+encodeBase64Url(signature)
```

El resultado final del token podría ser:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJsb2dnZWRJbkFzIjoiYWRt
aW4iLCJpYXQiOjE0MjI3Nzk2Mzh9.gzSraSYS8EXBxLN_oWnFSRgCzcmJmMjL
iuyu5CSpyHI
```

2.3.7 Pruebas unitarias. Mocha[20] y Chai[21]

Para realizar las pruebas unitarias de la API hemos utilizado Mocha y Chai.

Mocha es un framework de pruebas de JavaScript que se ejecuta en Node.js. Nos da la posibilidad de crear tanto tests síncronos como asíncronos de una forma muy sencilla. Nos proporciona muchas utilidades para la ejecución y el reporte de los tests. La idea utilizar Mocha es hacer una batería de

[20] https://mochajs.org/
[21] http://www.chaijs.com/

test. Cada uno de estos test debería probar un módulo independiente. Mocha te permite preparar un escenario antes de ejecutar cada test de tal manera que te da la posibilidad de hacer cualquier tipo de prueba. Una de las funciones más potentes consideradas es la de *before* con la que puedes preparar tu escenario antes de cada pequeño test.

Chai es un librería de aserciones, la cual se puede emparejar con cualquier marco de pruebas de Javascript. Chai tiene varias interfaces: assert, expect y should, que permiten al desarrollador elegir el estilo que le resulte más legible y cómodo a la hora de desarrollar sus tests. Se aprecia una ejemplo de cada una de las interfaces que se puede utilizar siendo saludo una variable con el contenido "hola":

```
Assert:
assert.typeOf( saludo, 'string', saludo is a string' );
assert.equal( saludo, saludo, saludo equal `hola`' );
assert.lengthOf( saludo, 4, saludo`s value has a length of 4'
);
Expect:
expect( saludo ).to.be.a( 'string' );
expect( saludo ).to.equal( hola );
expect( saludo ).to.have.length( 4 );
Should:
saludo.should.be.a( 'string' );
saludo.should.equal( hola );
saludo.should.have.length( 4 );
```

Se puede apreciar que con las tres operaciones se pueden realizar las mismas comparaciones. Chai es realmente flexible y se puede hacer cualquier tipo de comparaciones.
Un pequeño ejemplo de Chai integrado con Mocha puede ser el siguiente:

```
var assert = require("assert");
  before(function (done) {
     const a = 2;
     const b = 3;
    done()
   })
 describe("Vamos a probar las sumas",function(){
  it(Deberia sumar 2 + 3 y devolver 5',function(){
    assert.equal(5, a + b);
  })
});
 describe("Vamos a probar la resta",function(){
  it(Deberia resta 2 - 3 y devolver -1,function(){
    assert.equal(-1, a - b);
  })
});
```

.

.
.

De este pequeño código se pueden desarrollar algunos puntos:

- **before()**: Prepara el escenario antes de cada test. En este caso inicializa a y b con sus respectivos valores. Un caso muy común es poner todas las colecciones, ya sean estáticas o las de que se encuentran en la base de datos, vacías.

- **done()**: Se trata de la función callBack que avisa a before cuando el escenario está preparado.

- **describe**: Realiza una pequeña descripción de la prueba en cuestión que se va a realizar.

- **it** : Dentro de cada prueba puede haber varias posibilidades por ejemplo en la el caso de una resta puedes comprobar si 2 -3 es igual a -1 o si restas un número menos el mismo el valor es 0.

2.4 Frontend

Una vez desarrollado todo el backend se ha procedido a implementar la parte del frontEnd. Esta es toda la parte que se ejecuta en la parte del cliente. Para ello, hemos decidido usar las librerías React y Redux. Cuando hablamos de estas dos librerías damos el lenguaje que vamos a usar es JavaScript. De igual manera se va a proceder a desarrollar el por qué hemos optado por estas opciones.

2.4.1 JavaScript

De igual manera que en el backend se ha decidido utilizar JavaScript como lenguaje de programación de todo el proyecto.

2.4.1 React[22]

React es una biblioteca, desarrollada por FaceBook, escrita en JavaScript para facilitar la creación de componentes interactivos, reutilizables, para interfaces de usuario. Uno de los puntos fuertes que tiene React es el de crear aplicaciones en una sola página. Que una empresa como Facebook se encargue de su mantenimiento habla muy bien del soporte y el potencial que tiene. Instagram también está detrás de React ya que es de Facebook. De hecho, no todo FaceBook está construido con React pero Instagram si.

Uno de los puntos más fuertes de React es su manera de gestionar el cuerpo del html. Los cambios de páginas desde un punto de vista convencional se ha hace renderizando html nuevo por cambio. Utilizando React, sin embargo, se crea una copia de todo el contenido en memoria y se muestra la diferente información deseada desde la misma pantalla. Esto hace que exista una fluidez superior. React facilita la integración con JavaScript de tal manera que la página es totalmente dinámica en todo momento. Además react está pensado para repetir el código lo menos posible y uno de sus fuertes lo

[22] https://reactjs.org/

basa en la existencia de lo que se llaman componentes. Los componentes se podrían definir como elementos visuales de la página que se pueden modificar de manera dinámica. Todos los componentes pueden tener dentro más componentes y así sucesivamente.

En React, cabe destacar que en cada componente hay una variable estado. Esta variable en el momento que cambia refresca todos los componentes que dependían de ella. la variable es un objeto así que no tiene porque cambiar toda. Pueden cambiar las partes que interesen en cada momento.

2.4.2 Material UI [23]

Es una librería de componentes React que sigue el estilo Material Design[24]. Este estilo está muy de moda y es el estilo por excelencia que sigue Google. En el mundo del desarrollo web está muy de moda y hay muchas referencias y guías de estilos donde el desarrollador puede consultar. En este proyecto, no se le ha exigido al estudiante saber maquetar con HTML y CSS por lo que una librería como esta es lo que necesitaba para obtener una interfaz consistente y profesional.
Algunas de las características por las que en este proyecto se eligió utilizar esta guía de estilos son las siguientes:

- La poca información con muchos espacios es una buena manera de resaltar la información relevante y obligarse de alguna manera a no poner información innecesaria.

- La existencia y la interacción de los objetos en el espacio son transmitidas a través de los principios de la luz, la superficie y el movimiento.

- El gran uso de esta guía de estilos hace que muchos usuarios ya estén familiarizados con ella.

- El estudio de todos estos años de Google hace que el acabado sea muy profesional.

[23] https://material-ui.com/
[24] https://material.io/

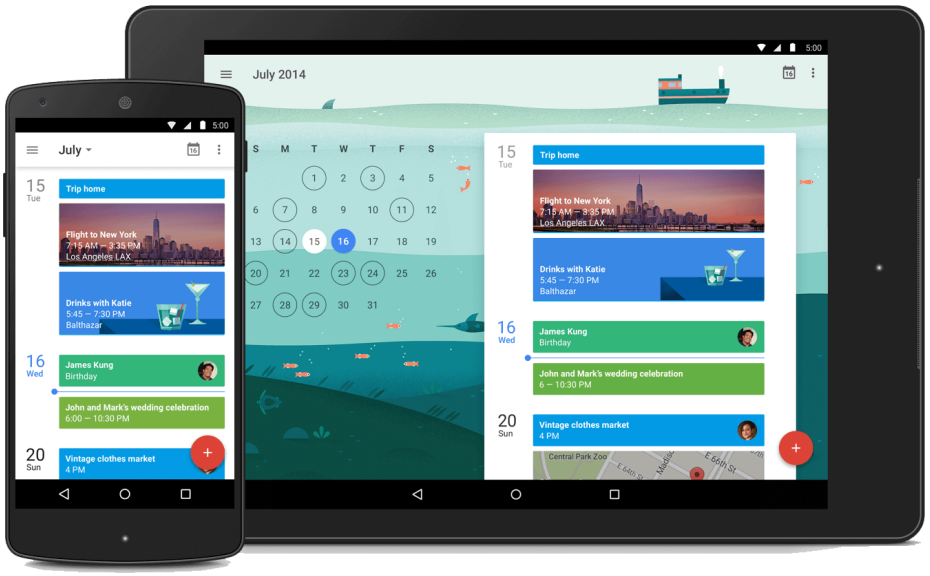

Imagen 4. Representación de Material Design. Imagen obtenida del siguiente enlace[25] que sirve como referencia de el estilo Material Design.

Aquí se puede ver un móvil y una tablet que sigue la guía de estilos de Material Design. Como se puede apreciar es una guía muy reconocida en este mundo. Obviamente las aplicaciones nativas de Android también lo siguen.

2.4.2 HTML y CSS

Como se ha mencionado, ne ha exigido al estudiante saber sobre HTML o CSS. No obstante cabe mencionar que la base de la maquetación de este proyecto se basa en estas dos herramientas. En este caso, Material UI ha sido el encargado de hacer casi todo el HTML y el CSS por nosotros.2.4.3 Redux[26].

Redux es una librería JavaScript que complementa React y tiene como función unificar todos los estado que tiene React en los diferentes componentes. Solamente con React se puede hacer una aplicación web pero Redux te facilita mucho el trabajo ya que todo está centralizado.

[25] https://desarrollador-android.com/
[26] https://redux.js.org/

2.5 Guardado del trabajo

Para guardar todo el código y coordinar todo el trabajo con el tutor se ha utilizado BitBucket[27]. Esta es la habitual entre los trabajadores de Redradix. Antes de empezar con el desarrollo, el estudiante estuvo formándose con estés herramienta de trabajo.

Bitbucket es un servicio de alojamiento basado en web, para los proyectos que utilizan el sistema de control de versiones Git. Bitbucket ofrece planes gratuitos y de pago. Al haberse utilizado el Bitbucket de Redradix no se ha tenido ningún tipo de limitaciones.

A continuación se verán algunos de los puntos más importantes que se pueden destacar de esta herramiento en este proyecto

2.5.1 Bitbucket de trabajo

Redradix dio acceso al estudiante para que este tuviera acceso a todos los proyectos de la empresa. De esta manera el estudiante ha podido consultar código de personal muchos más experimentado. De igual todos los trabajadores tenían acceso a para ver el código subido por el estudiante y hacer cualquier tipo de comentario o sugerencia.

2.5.2 Commits

En Bitbucket tenemos la posibilidad de poder hacer commit a modo de guardado temporal del trabajo realizado. Este commit va acompañado de un comentario poder saber en qué punto estás. El tutor ha insistido en hacer commits por cada hito funcional por pequeño que sea por si hay algún problema en algún momento poder volver atrás y así poder recuperar el código anterior. En más de un ocasión el estudiante ha tenido que volver atrás y tener que recuperar el código anterior. Esto ha ahorra bastante tiempo a la hora de encontrar errores.

2.5.1 Branch

Cada vez que el estudiante se dispone a implementar un módulo nuevo este hacía una nueva rama (branch) para poder trabajar en paralelo a la rama master(la principal del proyecto). De esta manera el código nuevo y sin dependencias directas con otros era desarrollado sin causar ningún tipo de conflicto.

2.5.4 Pull Request

Una Pull Request es la petición para poder juntar una rama con otra. De esta manera cuando el estudiante creía acabado el trabajo realizado en una rama solicitaba una pull request al tutor académico. Este la revisaba y si todo estaba en orden la mergeaba. Cabe destacar que el tutor ha sido muy profesional y exigente en cada una de los avances del proyecto. Es por esto por lo que el estudiante ha tenido que poner especial atención todo detalle del código realizado.

[27] https://bitbucket.org/

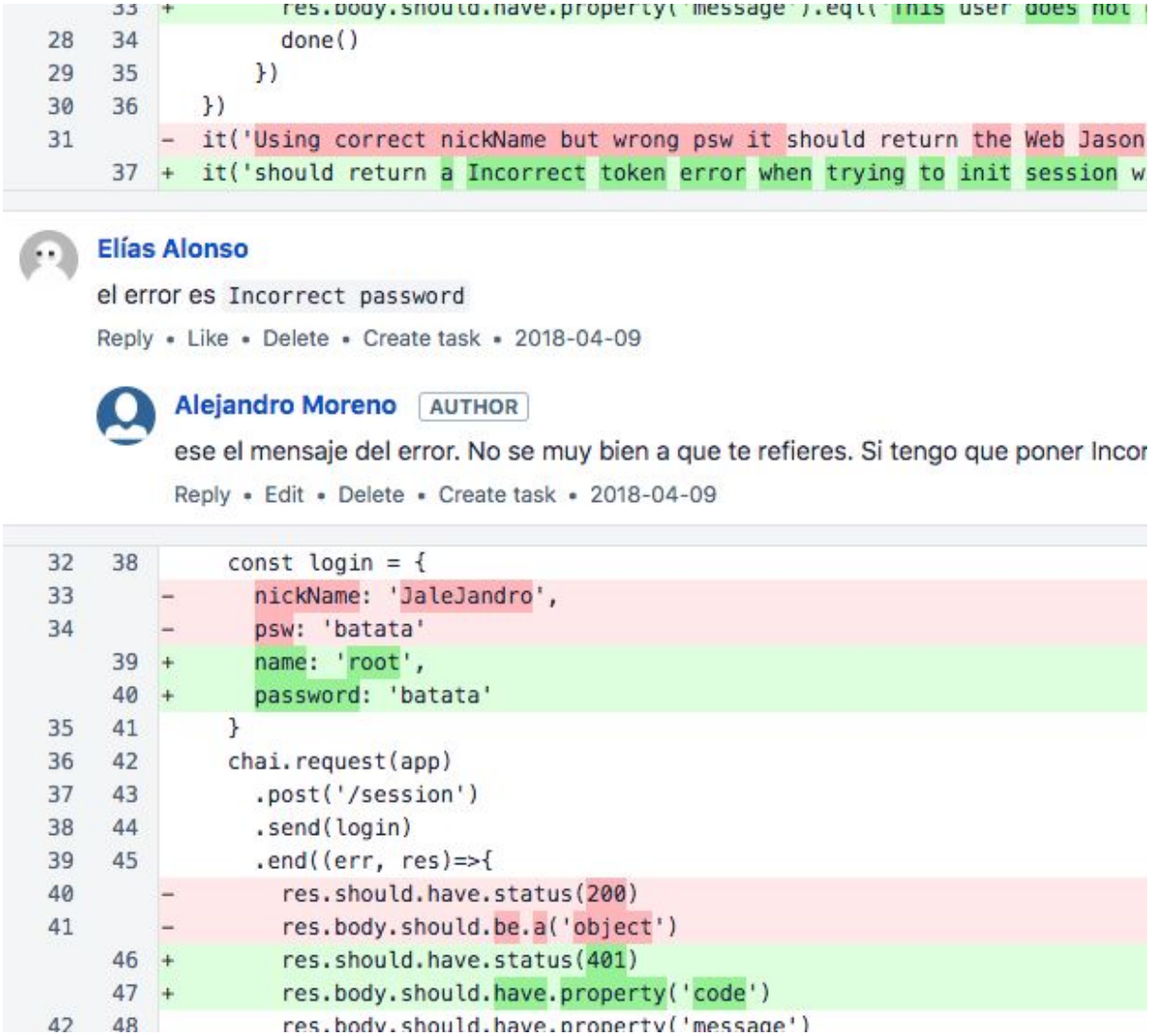

Imagen 5. Pull Request Bitbucket. Fuente: Pantallazo de una Pull Request del proyecto actual

Este es el aspecto que tiene el código cuando haces un Pull Request. En verde se puede apreciar lo nuevo que ha subido tras el último commit y en rojo lo que estaba antes. También se pueden escribir comentarios entre los integrantes que se encuentran en el Pull Request como se aprecia en la imagen. Esto ha sido de gran ayuda a la hora de iterar antes de mergear el código.

2.6 Administrador de tareas

Para guardar comunicación entre el estudiante y el tutor académico a la hora de gestionar las tareas realizadas o a realizar se ha utilizado la herramienta Trello[28].

Trello es un gestor de proyectos en el cual las diferentes tareas se dividen en tarjetas. Estas tarjetas se encuentran en un tablero y van pasando de un estado a otro hasta, finalmente, terminar completas. En Trello pueden estar varios participantes. En el caso de KoriNGO solo se ha encontrado el alumno y su tutor de trabajo. Normalmente el tutor ha ido subiendo tareas, ya sea de lectura o de implementación, para que el estudiante fuese trabajando en ellas. El aspecto del tablero trello usado en este proyecto es el siguiente:

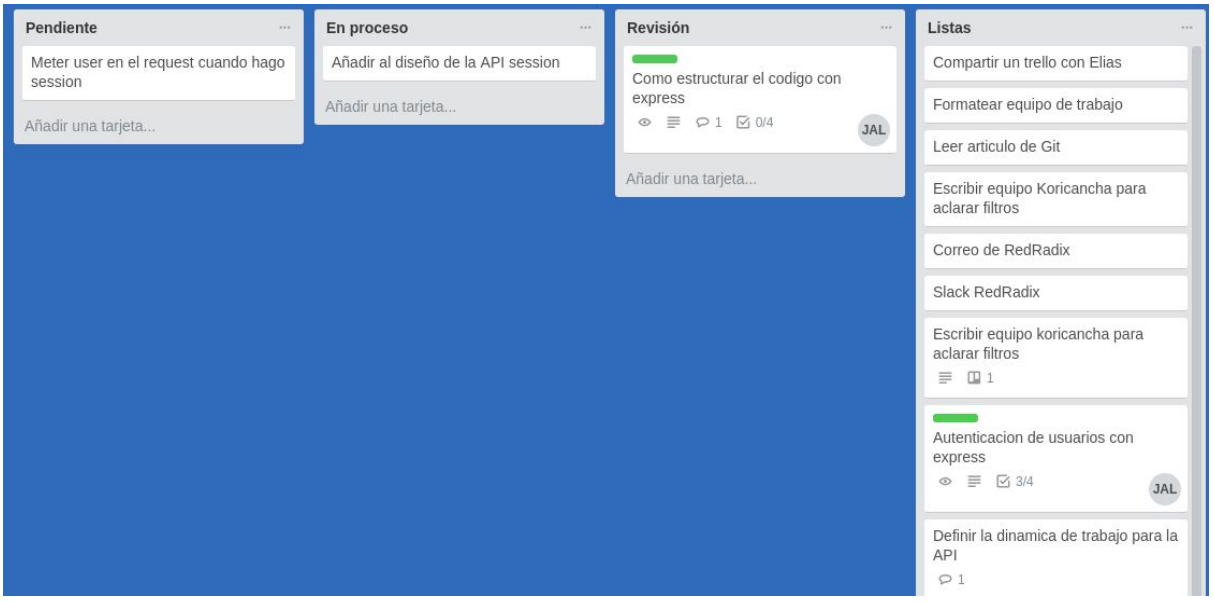

Imagen 6. Tablón de Trello. Fuente: Pantallazo de una tablón de Trello con el proyecto actual

Se puede apreciar el tablero con los diferentes estados Pendiente, En proceso, Revisión y listas. Se va a proceder al desglose de cada uno de estos estados:

- **Pendiente**: Aquí se encuentran todos los procesos que que no se han empezado pero que el tutor cree oportunos listos para ser empezados

- **En proceso**: Aquí se encuentran todos los procesos que se han empezado pero que no se han terminado. El estudiante en este caso es el que decide qué procesos están finalizados.

- **Revisión**: Una vez que el estudiante cree oportuno qué procesos se han terminado los manda a la revisión dónde el tutor considera si el proceso está terminado. En caso de valorar que el proceso no está terminado los puede mandar otra vez el estado "En proceso".

- **Listos**: Finalmente, si el tutor considera terminado definitivamente un proceso lo manda al estado de listos.

[28] https://trello.com/

2.7 Comunicación

Para establecer una comunicación se ha utilizado la herramienta Slack y el correo. En ese caso, el estudiante ha podido comunicarse con todos los trabajadores de la empresa pudiendo preguntar cualquier duda y proponer una sugerencia.

En Slack , además de poder por mensaje privado con quien se desee existen diferentes canales. Esto es bastante interesante ya que según el canal en el que busques puedes obtener diferente información.

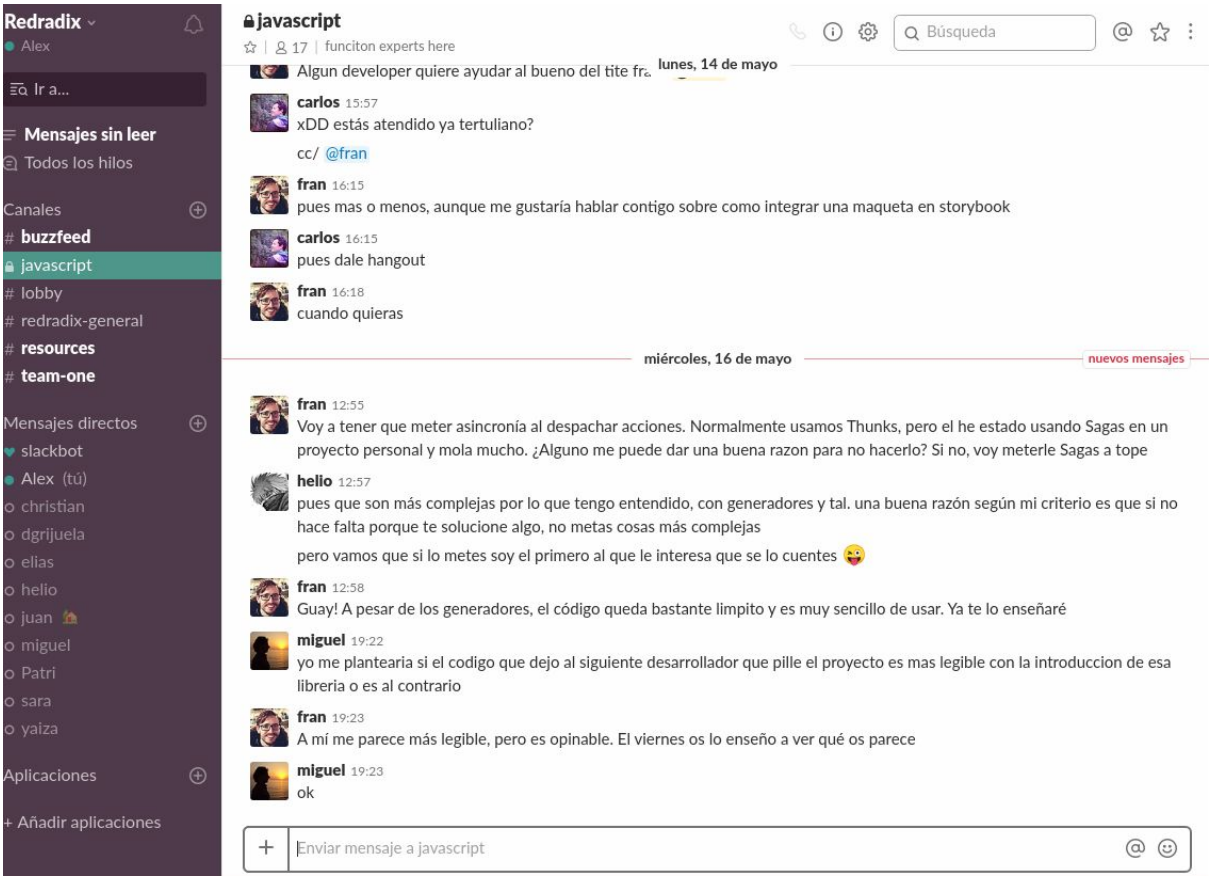

Imagen 7. Tablón de Slack. Fuente: Pantallazo de una conversación por Slack de un grupo de programadores de Redradix

Se pueden apreciar los canales y los hilos privados en el menú de la izquierda. En nuestro caso podemos apreciar canales muy interesantes como el cana javascript o redradix-general.

3 Diseño

3.1 Estudio de necesidades y problemas

El estudiante, además de haber cooperado tres veces (distintos años) en Koricancha, ha viajado al proyecto en el mes de enero del año actual. Esto ha servido para fijarse de manera más atenta en las necesidades y problemas a las personas que forman parte del proyecto. Además se reunió con el tutor académico para tener unas pequeñas pautas. El objetivo de este viaje, entre otros, era el de escribir todo lo que se veía para luego juntar esa información y poder desarrollar un software lo más coherente y completo posible. Las notas más destacables sacadas durante este son estas:

- Nueve de cada diez mujeres sufren violencia de género de algún tipo.
- El veinte por ciento de los usuarios del programa no cumplen la regla de escribir una carta cada dos meses a los respectivos padrinos.
- El treinta por ciento de los usuarios no cumple con la regla de asistir a las reuniones cuando son convocadas.
- Del total de los ahijados registrados en el programa el sesenta por ciento ha sido visitado sólo una vez y el veinte por ciento ha sido visitado cuatro o más veces en el tiempo.
- De los futuros usuarios que solicitan el ingreso al programa sólo se han entrevistado al cincuenta por ciento. Los demás usuarios han dejado de interesarse por el programa.
- De todas las familias entrevistadas el 50 por ciento de las personas no van a la escuela.
- De entre todas las familias entrevistadas del programa el 60 por ciento de personas que necesitan terapias no asisten a ellas.
- De entre todas las familias entrevistadas del programa el 55 por ciento de personas que necesitan tratamiento no tienen acceso a ellos.
- El setenta por ciento de las familias tienen por lo menos una parte de su casa de adobe.
- El diez por ciento de las familias entrevistadas no tiene acceso a agua corriente.
- El veinte por ciento de las familias visitadas han sido víctimas de las pasadas lluvias.

Todos estos puntos mencionados se han tomado desde un punto objetivo y lo más preciso posible. Las diferencias que pueden existir respecto a otros países o zonas son sobre todo cultural y económica.

3.2 Recogida de requisitos funcionales

Antes de desarrollar la API se dejaron muy claros cuáles serían los requisitos funcionales haciendo un previo estudio de las necesidades y problemas existentes. Al haber elegido cascada como metodología de trabajo era muy importante no dejar ningún detalle suelto. Obviamente el tutor académico ha apoyado al estudiante durante la recogida de estos requisitos. Antes de pasar a diseñar la API el tutor de trabajo a corregido la recogida de requisitos funcionales varias veces teniendo el estudiante que iterar sobre las entregas realizadas.

3.2.1 Requisitos Funcionales

Aquí se plasman los requisitos funcionales que se definieron de comenzar a desarrollar la API:

Fichas de participantes

- *El contenido de las fichas se dividirán en dos:*
    - *Datos personales(datos sin historial)*
    - *Datos estado(datos que tienen un histórico)*
- *Se podrán **ver** los detalles de cada ficha*
- *Se podrán **crear nuevas** fichas*
- *Se pueden **modificar** los campos de una ficha*
- *Los campos **obligatorios** para rellenar una ficha son:*
    - *Nombre del participante*
    - *Apellidos*
    - *Teléfono*
    - *Dirección*
- *Todos los demás campos son **opcionales** y se podrán dejar en blanco*
- *Cada ficha contará con un **score**(Para determinar la prioridad que tiene tiene esa ficha de hacer una visita)*
- *El **score** depende de los siguientes parámetros:*
    - *Periodo de tiempo desde que se realizó la ficha(si es que está hecha)*
    - *Condiciones de la vivienda*
    - *Ingresos de la familia*
    - *Costo del tratamiento que tiene el usuario*
    - *Apadrinamiento/lista de espera*
    - *Tiempo en lista de espera*
    - *Cumplimiento del programa (El programa tiene una seria de normas cómo escribir cartas o asistir a las reuniones)*
        - *No lo cumple*
        - *Cumple muy poco*
        - *Cumple la mitad*
        - *Cumple casi todo*
        - *Lo cumple por completo*
- *El **score** se podrá modificar manualmente*
- *Se verá un **listado** de las fichas mostrando en cada **entrada** su información más importante*
- *En cada **entrada** de la lista se podrán visualizar las siguientes miniaturas de cada ficha:*
    - *Fotografía*
    - *Nombre*

- *Apellidos*
- *Teléfono del responsable*
- *Nombre del responsable*
- *Dirección*
- *Tipo de construcción de la casa*
- *Tipo de propiedad*
- *Mensaje para mostrar **información importante***

*Se podrán **eliminar** las fichas de los participantes*

- *Se podrá **filtrar** el listado según algunas características. De no especificarlo se filtrará introduciendo una cadena o número con el valor de cada filtro en cuestión.*
  - *Datos personales*
    - *Nombre*
    - *Localidad*
  - *Educación-Formación*
    - *Centro actual*
    - *Terapias. Tipo de terapia que el participante recibe*
  - *Datos médicos*
    - *Diagnóstico*
    - *Movilidad. Rango de movilidad que el participante tiene. Este rango va del 1 al 4 según las siguientes características:*
      1. *No se puede mover*
      2. *Puede desplazarse de un sitio a otro*
      3. *Puede hacer pequeñas tareas*
      4. *Físicamente puede hacer cualquier tarea*
    - *Silla de ruedas.*
    - *Comunicación. Rango de comunicación que el participante tiene. Se asigna un número según la siguiente lista:*
      1. *No habla*
      2. *Habla palabras sueltas*
      3. *Formula frases simple*
      4. *Se le entiende bien*
      5. *Habla sin problema*
    - *Pruebas realizadas*
    - *Tratamiento*
  - *Datos de la vivienda*
    - *Propiedad/Alquiler. Se podrá rellenar con uno de los campos.*
    - *Tipo de construcción. Se podrá rellenar con uno de los siguientes campos:*
      1. *Estera*
      2. *Triplay*
      3. *Adobe*
      4. *Ladrillo*
    - *Dotaciones. Se podrá elegir cualquiera de las siguientes:*
      1. *Agua*
      2. *Gas*

3. *Luz*
4. *Baño*
5. *TV*
6. *DVD*
7. *Minicadena*

- *Datos económicos*
  - *Apadrinamiento/Año de comienzo. Se podrá rellenar con el año de comienzo del apadrinamiento.*

- *Otros*
  - *Tiempo desde que no se visita. Habrá cuatro intervalos que se decidirán con algunos voluntarios*
  - *Ficha nueva. Se podrá introducir true or false. Si es true vemos las nuevas y si es false las antiguas.*
  - *Veces que se ha visitado . Se podrá introducir el número de veces que se ha visitado anteriormente*

Datos personales

- *Sus datos serán los siguientes:*
  - *Datos personales*
    - *Nombre*
    - *Primer apellido*
    - *Segundo apellido*
    - *DNI*
    - *Zona*
    - *Dirección*
    - *Fecha de nacimiento*
- *A partir de la dirección se guardarán las* ***coordenadas geográficas***

Datos estado

- *Los datos estado siempre están vinculados con una fecha y un usuario y una ficha. Estos datos cambian en el tiempo y se muestran en forma de historias. Nunca se borran los datos estado. Los datos estado siempre se encuentran en las fichas de los usuarios del programa*
- *Sus datos serán los siguientes:*
  - *Datos del responsable del programa*
    - *Nombre*
    - *Primer apellido*
    - *Segundo apellido*
    - *DNI*
    - *Parentesco*
    - *Teléfono*
  - *Datos familiares*
    - *Información de cada familiar*

1. *Nombre*
2. *Primer apellido*
3. *Segundo apellido*
4. *Fecha de nacimiento*
5. *Parentesco*
6. *Estudio/trabajo*
7. *Convivencia*
8. *Observaciones*

- *Observaciones generales*

- *Educación-Formación*
  - *Centro actual*
  - *Año de entrada*
  - *Centros anteriores*
  - *Apoyos especiales/Terapias*
    1. *Centro*
    2. *Comienzo*
    3. *Tipo de terapia*
    4. *Asistencia*
       - *No lo cumple*
       - *Cumple muy poco*
       - *Cumple la mitad*
       - *Cumple casi todo*
       - *Lo cumple por completo*
    5. *Observaciones*
- *Situación social*
  - *Tiempo libre*
  - *Cuidadores*
  - *Apoyos cercanos*
  - *Relaciones con el entorno*
    1. *Nada*
    2. *Un poco*
    3. *Normal*
    4. *Bastante*
    5. *Mucho*
  - *Observaciones*
- *Datos médicos*
  - *Diagnóstico*
  - *Año del diagnóstico*
  - *Movilidad (checkbox)*
    1. *No se puede mover*
    2. *Puede desplazarse de un sitio a otro*
    3. *Puede hacer pequeñas tareas*
    4. *Físicamente puede hacer cualquier tarea*
  - *Silla*
  - *Comunicación (checkbox)*
    1. *No habla*

        2. *Habla palabras sueltas*
        3. *Formula frases simple*
        4. *Se le entiende bien*
        5. *Habla sin problema*
    - *Pruebas realizadas*
        1. *Nombre*
        2. *Año*
        3. *Observaciones*
    - *Tratamiento*
        1. *Nombre*
        2. *Año*
        3. *Observaciones*
    - *Enfermedades de otros miembros de la familia*
        1. *Nombre de la enfermedad*
        2. *Parentesco*
    - *Observaciones*
- *Datos de la vivienda*
    - *Propiedad/Alquiler*
    - *Tipo de construcción*
    - *Estado*
    - *Dotaciones(Agua, Gas, Luz, Baño, TV, DVD, Minicadena)*
        1. *Agua*
        2. *Gas*
        3. *Luz*
        4. *Baño*
        5. *TV*
        6. *DVD*
        7. *Minicadena*
    - *Número de estancias*
    - *Número de camas*
    - *Mobiliario*
    - *Salubridad (Olores, salubridad, limpieza)*
    - *Observaciones*
- *Datos económicos*
    - *Ingresos familiares*
    - *Apoyos externos*
        1. *Apadrinamiento*
            - *Entidad*
            - *Año de comienzo*
            - *cantidad*
        2. *Comedor*
            - *Nombre*
            - *Año de comienzo*
        3. *Otros*
- *Observaciones **generales***
- *Observaciones **manifestadas***

    - *Observaciones **detectadas***
    - ***Pendiente***
    - ***Mensaje importante***
- *Las fichas se podrán marcar como **pendientes** en el caso de que falte algo por editar*

Visitas

- *Habrá dos tipos de visitas:*
    - ***Próximas***
    - ***Realizadas***
- *Las visitas están **asociadas** a una **ficha***
- *Las visitas se organizan en **listados** mostrando su información más importante*
- *Se podrán **eliminar** visitas*
- *Se podrán **modificar** el comentario importante de una visita*

Próximas **visitas**

- *Las próximas visitas son de dos tipos:*
    - *Manuales*
    - *Automáticas*
- *Para **crear** una vista de un **participante nuevo**(que no tenga ficha) habrá que crear una nueva ficha con los siguiente campos:*
    - *Nombre*
    - *Apellidos*
    - *Teléfono*
    - *Dirección*
    - *Comentario importante(Opcional)*
- *La información más importante de cada visita que aparece en cada listado es:*
    - *Información personal de cada participante:*
        - *Nombre*
        - *Apellidos*
        - *Nombre de responsable (Si está registrado)*
        - *Teléfono*
        - *Dirección*
    - *Mensaje para mostrar información importante*
    - ***Score** de la ficha asociada*
    - *Periodo de tiempo desde que se realizó la ficha(si es que está hecha)*
    - *Condiciones de la vivienda*
    - *Ingresos de la familia*
    - *Costo del tratamiento*
    - *Cumplimiento del tratamiento*
    - ***Índice** incremental mostrando la prioridad*
- *Los voluntarios pueden filtrar este listado teniendo en cuentas las siguientes características:*
    - *Orden por cercanía según una dirección*
    - *Apadrinamiento*
    - *Zona*

    - *Estudiar mas parametros*
- *Las visitas se podrán marcar como realizadas o pendientes especificando:*
    - *La fecha*
    - *El usuario*
    - *Comentario*
- *Las visitas realizadas se pueden marcar como pendiente en el caso de que falte algo importante por rellenar*

Próximas visitas **manuales**

- *Se le asignará una prioridad*

Próximas visitas **automáticas**

- *El orden de las visitas estará establecido por el **score** de cada ficha asociada*

Visitas **realizadas**

- *Se mostrará un historial de las visitas realizadas*
- *Se podrá filtrar por lo siguiente:*
    - *Fechas en las que se han realizado.*
    - *Usuario que las ha realizado.*
    - *Localidad*
    - *Nombre del participante*
    - *Intervalo de fechas*

Control de acceso

- *Se podrán dar de alta a los usuarios rellenando los siguientes campos:*
    - *Nombre*
    - *Apellidos*
    - *NickName*
    - *Contraseña*
    - *Email*
    - *Fecha de nacimiento*
    - *Estudios*
    - *Profesión*
    - *Voluntariado previo*
    - *Coche*
    - *Fecha en la que se crea*
    - *Rol*
        - *Administrador*
        - *Usuario*
- *Se podrá **iniciar sesión** introduciendo el NickName y contraseña.*
- *Se podrán **eliminar** usuarios*
- *Se podrá **ver** la información de cada usuario*
- *Se podrá **modificar** la información de cada usuario*

- **El administrador tendrá las **mismas funciones** del usuario además de las suyas*
- *Las funciones que podrá realizar un usuario serán :*
    - *Fichas de participantes*
        - *Ver **listados***
        - *Ver **detalles***
        - ***Crear** fichas*
        - ***Modificar** fichas*
            - *POsibilidad de indicar una prioridad manual que sobreescribe el score*
        - ***Eliminar** fichas*
        - ***Filtrar** fichas en listados*
    - ***Visitas***
        - *Crear visitas*
        - *Modificar visitas*
            - *Modificar score*
        - *Eliminar visitas*
        - *Filtrar visitas*

Como se puede apreciar los requisitos funcionales han sido tomados con total grado de detalle. Esto ha sido clave a la hora de desarrollar ya que apenas se ha tenido que iterar sobre la arquitectura planteada desde un inicio.

3.3 Diseño de la API con BluePrint

Ya se han mencionado la facilidades que ofrece BluePrint a la hora de realizar el diseño de una API REST. Ahora se va hablar de todo lo mencionado anteriormente con relación a nuestro proyecto.

Antes se ha comentado que partes tiene una una API ahora se pretende desarrollar cada una de estas partes aplicadas a KoriNGO.

Para traducir todo el documento escrito en el lenguaje de BluePrint al .html y poder ver la API de una manera sencilla y legible se ha utilizado aglio[29]. Se trata de un paquete de NPM. Una vez instalado este te da varias opciones para elegir el formato en que uno desee crear el .html final. En el caso de esta API se ha decidido sacar un .html con tres columnas diferentes:

- Primera columna: muestra un menú con los diferentes recursos y los métodos que se encuentran en cada uno de estos.

- Segunda columna: Estructura de toda la información con la que cuenta un recurso determinado. También se aprecian los parámetros y cabeceras que pueden ir ligados a un método. En la estructura de la información se observan características como el nombre del campo en cuestión, el tipo de dato o si este dato es requerido o no.

[29] https://www.npmjs.com/package/aglio

- Tercera columna: En esta columna se aprecia la información en sí del recurso que se desea. Por ejemplo el token de validación o el elemento JSON que se solicita o envía según el método deseado.

En la primera te. En la segunda columna se aprecian la estructura de toda la información que tiene cada método asociado a un recurso.

Estas el pinta que tiene el documento .html generado al después de generar todo el diseño con BluePrint.

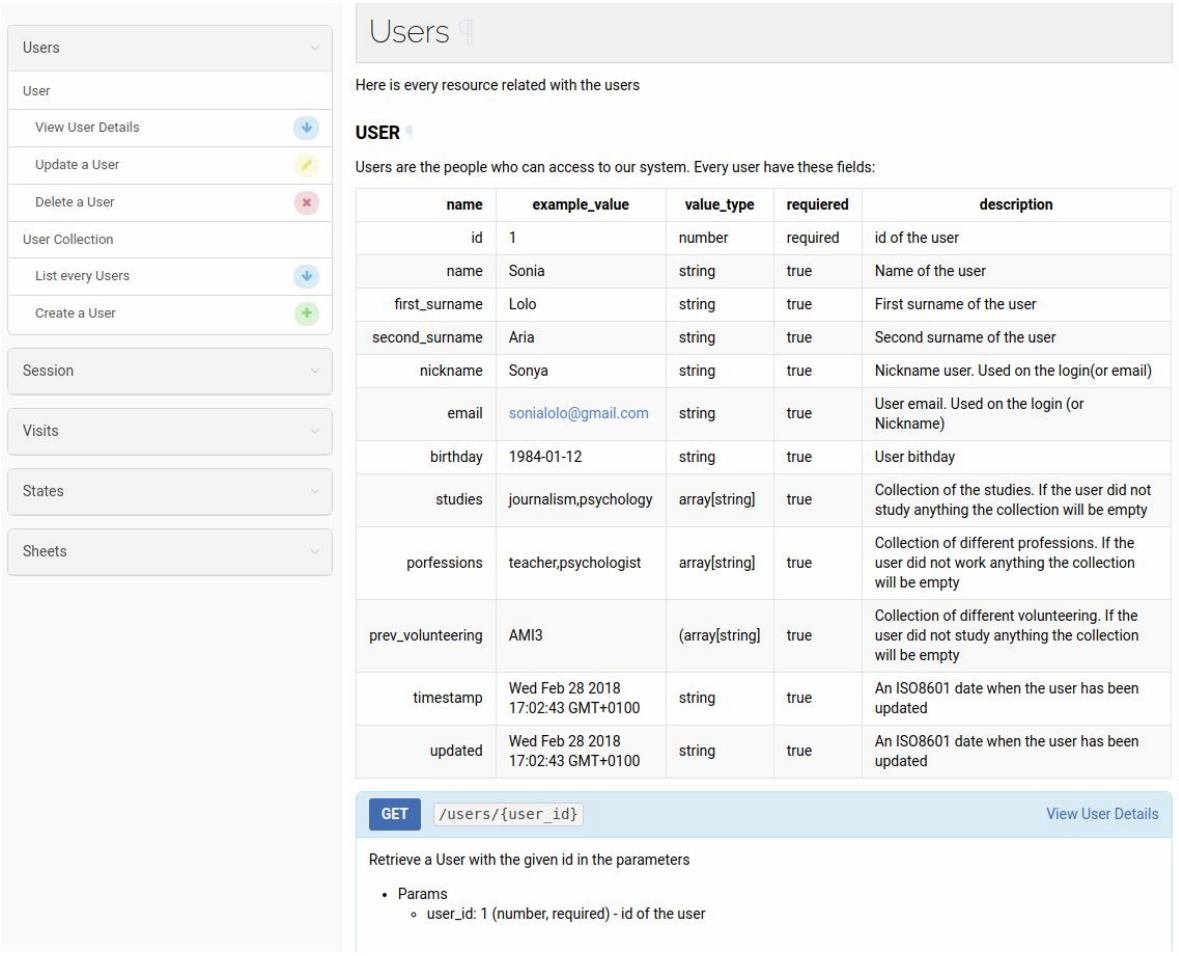

Imagen 8. Api BluePrint Comunas 1 y 2. Pantallazo de las dos primeras columnas de la API del proyecto

En esta imagen se aprecian las dos primeras columnas de nuestra API. En la primera de ellas se puede apreciar el un menú con los diferentes recursos. En este caso el único de estos que está desplegado es users. Se puede ver que que en el caso de users hay dos partes. Los métodos GET y POST no requieren de id. No se necesita ya que el caso de GET devuelve una colección y para crear un nuevo usuario con POST no hace falta especificar un id. En cambio, en los demás métodos si que hace falta especificar que el id del usuario en cuestión. Es por que para eliminar o modificar un usuario si que hace falta especificar el id del usuario en cuestión. Se aprecia que aquí también apareßce el método GET. Esto es porque GET se puede usar para obtener una colección de usuarios o uno en cuestión.

En la segunda columna muestra la estructura de la información que tiene un recurso. Es importante pensar qué campos son requeridos y cuáles no. También están las características de los parámetros pasados.

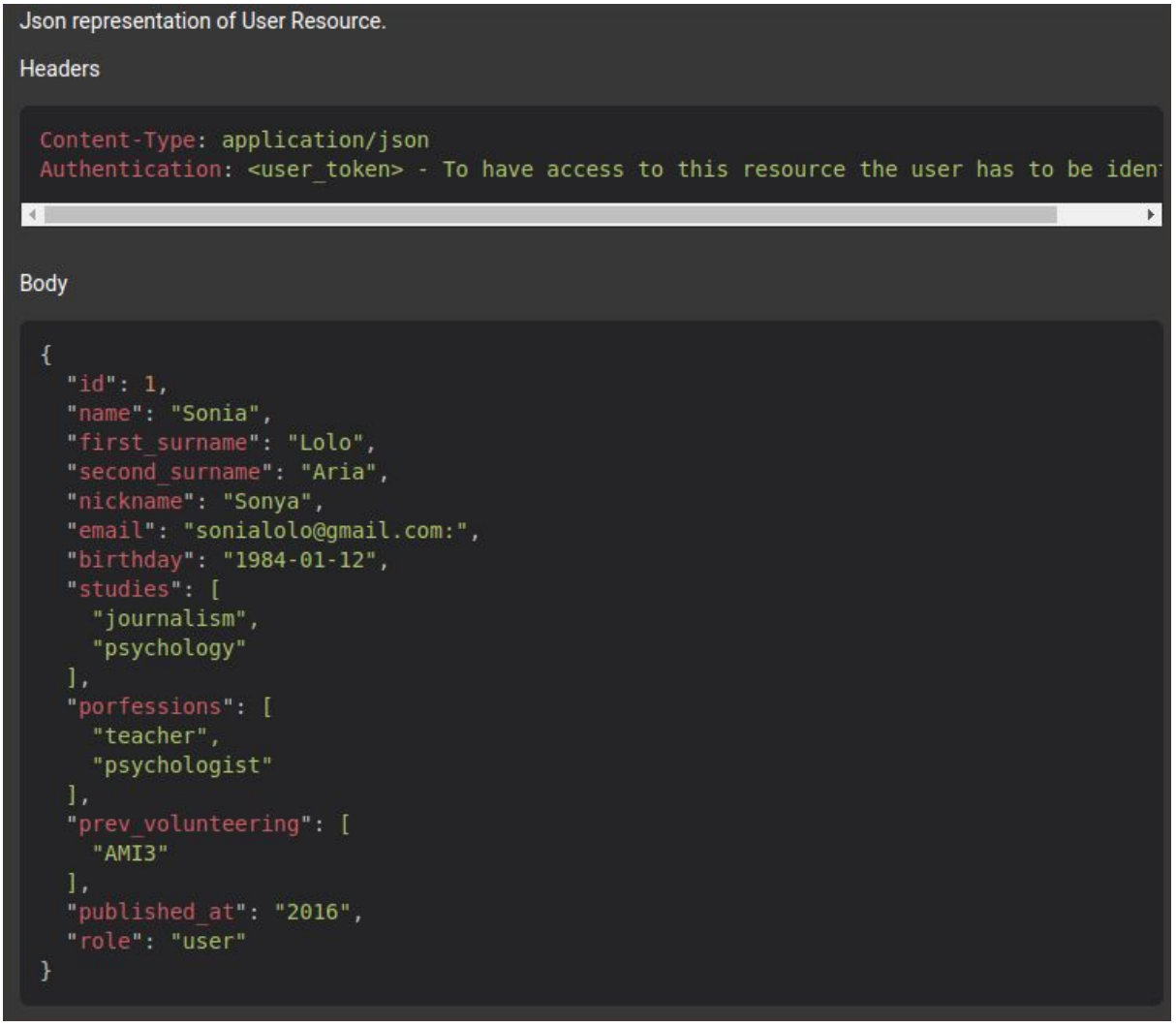

Imagen 9. Api BluePrint Comuna 3.Pantallazo de la última columna de la API generada del proyecto

En la última columna se aprecia un ejemplo de la forma que tiene cada recurso con un ejemplo. Esto resulta muy útil a la hora de hacer pruebas en el código ya que hay muchos ejemplos de request y responses.

3.1.1 Métodos de la API

En una API REST hay una gran variedad de métodos que se pueden utilizar. En este caso hemos utilizado solamente cuatro:

- GET: Este método se usa para obtener la información de un recurso en cuestión. Si se pasa sin parámetros devuelve una colección de elementos. En el caso de ir acompañado de un id te devuelve un elemento si es que existe. Además se pueden pasar filtros para que la response cumpla una sería de condiciones.

- POST: Se utiliza cuando quieres añadir un nuevo elemento a la colección que contiene un recurso. No hace falta acompañarlo de un id pues normalmente se genera automáticamente.

- PATCH: Se usa para modificar un elemento de un recurso. Este si que tiene que ir seguido de un id pues hay que tener claro qué elemento me estoy refiriendo. Un método muy parecido es UPDATE. Se ha decidido no utilizarlo por este cambia todo el elemento. En el caso de PATCH solo se actualizan los campos requeridos.

- DELETE: Este tiene la función de eliminar un elemento de un recurso. Necesitaría ir con un id para saber a qué elemento se desea eliminar.

3.1.2 Recursos usados

Después de pensar los requisitos se ha procedido a deducir los diferentes recursos de nuestra API REST. Estos recursos tiene que ver mucho con las pantallas que tiene la aplicación:

- **Users**: Contiene todos los usuarios que pueden entrar en la aplicación para utilizarla. Antes se tienen que loggear.

- **Session**: Es un recurso intermedio que se usa para autentificar usuario. Sólo tiene el método POST.

- **Sheets**: Contiene todas las fichas que tiene la aplicación contiene datos estáticos y datos variables que llamando States.

- **State**: Contiene todos los datos State. Estos datos no se eliminan y sirven para crear un historial de estos.

- **Visits**: Contienen las pasadas visitas y las futuras.

3.1.3 Cabeceras que se han utilizado

Los métodos pueden ir acompañados de cabeceras que tienen información adicional. En este caso se usan las siguientes:

- **Authentication**: Contiene un token que se decodifica y se puede saber información del usuario que está haciendo la petición. Se puede saber información relevante como qué role tiene el usuario.

- **Content-Type**: Explica que formato tienen los datos. En el caso de esta aplicación siempre serán JSON.

3.1.4 Parámetros requeridos y no requeridos

Cuando se han definido los diferentes recursos ha habido que plantearse qué campos eran requeridos y cuáles no. En el diseño de la API se pueden apreciar cuales son requeridos y cuáles no. Por ejemplo, cuando voy a crear una ficha por primera vez hay que tener en cuenta que el nombre es obligatorio pero que el segundo apellido no.

3.1.5 Validación de usuario

Para validar el usuario hay que logearse en la página de login. Una vez el usuario lo hace, si sus credenciales son correctas el servidor le da un token generado con Json Web Token que se guardará en la cabecera de cliente para que cada vez que este haga una petición el servidor pueda saber quién es el usuario en cuestión. Se puede sacar a partir de este token toda la información del usuario.

3.1.6 Filtros

En la URL de cada petición se pueden añadir los filtros según un estándar. Por ejemplo si se realiza un GET de las visitas una petición con filtros puede ser de la siguiente manera:

```
http://localhost:3000/visits?zone=["Chiclayo"]
```

En este caso se está filtrando por todas las fichas cuya zona sea Chiclayo.

3.4 Diseño de la interfaz

Una vez se contaba con el backend ya terminado se ha procedido a diseñar la parte visual de la aplicación. Se ha considerado que la mejor manera para visualizar las diferentes vistas de la aplicación ha sido con papel y lápiz. Se ha entendido que es más rápido y flexible. Tanto el estudiante como el tutor han visto oportuno desarrollar las diferentes pantallas de esta manera. En los anexos se puede ver una fotografía de la pantalla que contiene las fichas.

3.4.1 Pantallas

La aplicación se compone de una serie de pantallas. Estas pantallas están muy ligados con los recursos mencionados anteriormente. Las pantallas son las siguientes:

- **Pantalla home**: En esta pantalla se encuentra la información más general. Habrá un mapa con todos los usuario en el programa. Se Podrán utilizar unos filtros para cambiar los usuarios mostrados según se quiera. Los filtros son estos:
    - Apadrinados: Se muestran de todos los participantes los que están apadrinados.
    - No apadrinados: De todos los participantes se muestran los que se encuentran en lista de espera.
    - A visitar: Se encuentran las veinte primeras visitas que se encuentran en la lista de espera.
- **Pantalla usuarios**: En esta pantalla se encuentra una lista de los usuarios que pueden acceder al programa. En el caso de ser administrador, habrá un enlace en cada una para ver y editar cada ficha de cada usuario. En esta pantalla hay una serie de filtros.
- **Pantalla usuario**: Se muestra toda la información de un usuario. Se puede ver y editar esta información.
- **Pantalla Fichas**: Se muestra una pantalla con todas las fichas en el sistema. Habrá un botón para ver o editar cada una de las fichas. En esta pantalla hay una serie de filtros.
- **Pantalla ficha**: Es una pantalla para editar o ver una ficha en particular.
- **Pantalla visitas**: Se trata de una pantalla que contiene todas las visitas ordenadas según prioridad.
- **Pantalla no resultados**: Es una pantalla creada para mostrar en el caso de que no se encuentren resultados en una búsqueda.
- **Pantalla de login**: Pantalla para que el usuario pueda loggearse.
- **Pantalla de registro**: Pantalla para poder registrar a un nuevo usuario.

4 Desarrollo

Después de hacer toda la parte de diseño se ha procedido al desarrollo. Esta parte se puede desglosar en las siguientes.

4.1 Backend

Se ha creado un servidor con la ayuda de Espres como ya se ha mencionado antes. Una vez creado el servidor sin nada de funcionalidad se ha procedido a hacer cada uno de los módulos pertinentes. Las tres partes más independientes y que conllevan más funcionalidad son las siguientes:

4.1.1 Router

En esta carpeta se encuentra lo que se ha llamado index.html. Aquí se encuentran los diferentes métodos de los recursos.

```
const UserController = require('../controller/user')
const Session = require('../controller/session')
const middleware = require('../lib/middleware')

// API Server Endpoints
module.exports = function routes(router) {
  // Session group
  router.post('/session', Session.createToken)
  // User Group
         router.post('/users',    middleware.isAuthenticated,    middleware.hasPrivileges,
UserController.create)
  router.get('/users', middleware.isAuthenticated,UserController.getAll)
  router.get('/users/:id', middleware.isAuthenticated,UserController.get)
        router.patch('/users/:id',    middleware.isAuthenticated,middleware.hasPrivileges,
UserController.update)
      router.delete('/users/:id',   middleware.isAuthenticated,   middleware.hasPrivileges,
UserController.delete)
}
```

Aquí se puede ver un extracto del código de index.js que se encuentra en /router. Podemos ver los diferentes métodos del router del recurso users y session. Según el método y la ruta que se introduzca se llamará a una función del módulo controller de users. Se pueden apreciar los middleware entre los parámetros de las funciones. Se aprecian dos:

- middleware.isAuthenticated: Se encarga de comprobar si el usuario está registrado en el sistema.

- middleware.UserController.getAll: Una vez se ha comprobado si el usuario está en el sistema se comprueba si tiene suficientes privilegios.

El segundo de estos se ejecuta sólo en los métodos que lo requieren. Por ejemplo, para ver los usuarios sin modificarlos no se necesita autentificación.

4.1.2 Controller

Este módulo ha sido como medio de comunicación entre entre el servidor y la base de datos. Básicamente traduce toda la información pasada desde el servidor en un formato correcto para llamar a la base de datos. Toda la información innecesaria,por ejemplo, la elimina.

4.1.3 Model

Este módulo está creado para comunicarse directamente con la base de datos. Una vez se tiene la información información preparada desde el módulo controller se utiliza aquí para hacer las respectivas operaciones con la base de datos. En el caso de esta aplicación se conecta con la mongoDB

4.1.4 Gestión de errores

Hay un Módulo llamado error.js en el que se encuentran todos los errores posibles. Estos errores tienen la se representan mostrando un id de error y un mensaje de error.

```
{code: 0, message: 'Incorrect token'}
```

Cada vez que ocurre un error salta uno de estos mensajes.

4.1.5 Middlewares

Una parte muy importante a la hora de realizar el backend es la existencia de los Middlewares. Son un mecanismo para antes de hacer un operación realizar otra a modo de comprobación. En Express están especialmente optimizado. En el caso de esta aplicación se han utilizado para comprobar que cada petición al servidor va seguida de un token. También se comprueba para ciertas operaciones el rolle que tiene el usuario que hace la operación en cuestión. Por ejemplo, el middleware que comprueba los privilegios a la hora de hacer diferentes operaciones tiene esta forma.

```
        hasPrivileges: (req, res, next) => {
          if (req.headers.user) {
            const user = JSON.parse(req.headers.user)
            if (user.role === 'root' || user.role === 'admin' ) {
              next()
            } else {
              res.status(401).send(error.noPrivileges())
            }
          } else {
            res.status(401).send(error.notAuthenticated())
          }
 }
```

Se trata de una función intermedia que se llama antes de llamar a otras para comprobar en este caso si el usuario es riene role admin o root.

4.1.6 Cálculo del score

Ya se ha mencionado que cada ficha tiene un score asociada que implica el grado de prioridad que tiene a la hora se ser visitada. Este score se calcula con un según algunos parámetros que hay que ir regulando con el paso del tiempo según los voluntarios detecten qué es más importantes. En un

principio se tendrá en cuenta el tiempo desde que no se visita la familia, ingresos más bajos, tiempo en el programa y mensaje importante marcado por un voluntario.

4.2 Frontend

Ya se contado que el frontend es la parte que se ejecuta en en el cliente. Como puntos más importantes a desarrollar se han considerado estos:

4.1.1 React

Uno de las características más importantes de React son los componentes. Estos son pequeñas partes de la interfaz que se pueden reutilizar. A estos componentes se les pueden pasar propiedades desde el padre y de esta manera se con un componente se pueden crear diferentes con misma forma pero diferente contenido.
A continuación se aprecia uno de los componentes más sencillos que se encuentran en la aplicación. Hay que destacar que si se desea importar un componente ya creado hay que importarlo al inicio. En el caso del ejemplo de puede ver el MapContaiainer.

```
import React from 'react';
import MapContainer from './MapContainer'

const Home = (props) => {
  return (
    <div className='container'>
      <MapContainer/>
    </div>
   )
}

export default Home
```

En este ejemplo se puede ver que este componente se recibe props. Estas propiedades se pasan de componente en componente y tienen forma de objeto así que se pueden pasar tantas como se quieran.
Los componentes pueden tener funciones que se ejecutan antes o después de ser creados. Esto da mucho juego a la hora de formar componentes nuevos. Por ejemplo, se puede obtener la información de la base de datos antes de que el componente esté montado y una vez la tengamos montarlo.

React destaca por la masificación de desarrorralloder que la están utilizando. Es por eso por lo que es muy sencillo descargar un paquete y usar los componentes de infinidad de funcionalidades. En este proyecto, por ejemplo, se ha usado Google maps y no ha habido complicación alguna para integrarlo.

4.1.2 Redux

Ya hemos mencionado que Redux es un complemento de React para centralizar todo el estado y poder así controlar desde un punto la funcionalidad de toda la aplicación.

Se va a proceder a explicar la forma que tiene la parte del código que se ha realizado en Redux. En todo código Redux estas son las partes muy importantes con un pequeño ejemplo propuesto.

- Store: Al inicio se crea y se genera un estado al que está asociado

  ```
  const store = createStore(counter, 0)
  ```

  En este caso se ha creado con un estado counter inicializado a 0.

- Action: Aquí están las diferentes acciones que se pueden hacer con cada estado. Se definen llamando a dispatch

  ```
  store.dispatch({ type: 'INCREMENT' })
  ```

  En este caso se le aplica al store definido antes la operación INCREMENT. Se pueden definir tantas operaciones como se quiera.

- Reducer. Se trata del paso en el que se cambia el estado tras efectuar la operación. Nunca se debe sobreescribir la operación. Simplemente se devuelve el valor que se quiera.

  ```
  function counter(state, action) {
    switch(action.type) {
      case 'INCREMENT':
        return state + 1
      default:
        return state
    }
  }
  ```

- store.Subscribe: Cuando ya se ha ejecutado la action y el reducer se puede ver que pinta tiene el nuevo estado.

  ```
  store.subscribe(() => {
    console.log(`-> nuevo estado: ${store.getState()}`)
  })
  ```

Después de usar React de puede ver que la gestión de estados es mucho más sencilla centralizándolo con Redux.

5 Integración, pruebas y resultados

Para la realización de pruebas unitarias se han utilizado las ya mencionadas librerías Mocha y Chai. En el código hay una carpeta test dónde se encuentran las diferentes baterías de test. En el caso de querer ejecutar todas a la vez basta con estar en el directorio raíz y ejecutar :

```
npm test
```

Se puede escribir el nombre de uno de los tests al final para referirse a un test en concreto. Se va proceder a mostrar el código considerado más importante que tiene la aplicación que utiliza estas librerías.

```
process.env.NODE_ENV = 'test'
const chai = require('chai')
const chaiHttp = require('chai-http')
const chaiThings = require('chai-things')


chai.use(chaiHttp)
chai.should()
chai.use(chaiThings)
```

Al comienzo se importan los diferentes paquetes requeridos. Después se configura should que es la interfaz que se ha elegido.

```
describe('Checking the privileges', () => {
  beforeEach((done) => {
    User.__emptyCollection__()
      .then(() => done())
      .catch(done)
  })
```

Antes de ejecutar cada batería de test se inicializa el estado que se desea. En este caso se limpia la colección de usuarios.

```
   it('should return a Incorrect token error when trying to access a private
endpoint with an invalid token', (done) =>{
    chai.request(app)
      .get('/users')
      .set('authorization', 'Bearer u25KdsjXHaVU3G3PQgPiFy7KIWbfdIi6NyT6qjIQP3o')
      .set('user', '{"role":"admin"}')
      .end((err, res) => {
        res.should.have.status(401)
        res.body.should.have.property('code')
        res.body.should.have.property('message')
        res.body.should.have.property('message').eql('Incorrect token')
        done()
      })
```

```
})
```

Por último se ve cómo quedaría una prueba del test. En este caso se prueba que pasaría si se intenta obtener una lista de users con un token incorrecto. Se aprecia que el resultado a devolver es 'Incorrect token.

Cabe mencionar que se han encontrado muchísimos errores trascendentales en la aplicación que se habrían pasado por alto de no ser por estos tests. Se ha considerado buena idea crear la batería de tests antes de empezar a desarrollar el código que prueba.

La integración de todo junto, se ha hecho testeándolo manualmente con desarrolladores y con futuros usuarios potenciales. Se les ha dado unas instrucciones de uso ya que la usabilidad es muy sencilla y se ha comprobado si existen errores. Después de un mes de pruebas se les preguntará a los usuarios los problemas que han tenido. Al estar todavía en fase de pruebas no se ha podido testear a fondo el funcionamiento global de la aplicación.

6 Conclusiones y trabajo futuro

Después de todo este tiempo desarrollando esta aplicación considero que tanto para el estudiante como para el beneficiario es una gran oportunidad dedicar tu trabajo de fin de carrera al mundo de la cooperación. Es un mundo con miles de oportunidades para hacer un proyecto real y es bastante agradecido.

6.1Conclusiones

Debido a la motivación extra que suponía un desarrollo como este proyecto se ha intentado estar a la última en cuanto a lo que la tecnología se refiere. No ha sido fácil pero junto con Redradix el estudiante ha dedicado gran parte del tiempo y esfuerzo a la formación propia. El estudiante ya había trabajado con tecnología web en otras ocasiones pero es ahora cuando se ha dado cuenta que las tecnologías antes usadas son inferiores en muchas ocasiones a las que se han utilizadas en este proyecto. Las grandes diferencias que ha encontrado son las siguientes:

- **JavaScript vs PHP**: En general el estudiante se ha sentido mucho más cómodo con JavaScript. La principal ventaja es que se pueda escribir en el lado del cliente.

- **MongoDB vs SQL**: En las bases de datos no relacionales no hay que conocer las estructuras de las tablas desde el principio y esta puede variar durante el desarrollo. Las bases de datos relacionales tienen una serie de operaciones muy potentes que hacen ellas mismas, como por ejemplo los JOINS pero para un proyecto como este se ha preferido MongoDB.

- **React vs HTML convencional**: La ventaja más clara con respecto del HTML convencional es que se reutiliza mucho el código.
- **Cascada vs Metodología ágil**: El estudiante no es especialmente fan de la metodología en cascada pero al no existir un cliente en este proyecto que solicite requisitos con el paso del

tiempo se ha considerado inteligente definir bien el esqueleto desde un principio y no añadir gran funcionalidad nueva durante el proyecto.

6.2 Trabajo futuro

La idea de este trabajo es intentar que sirva para cuántos más proyectos mejor. Ya se ha mencionado que los proyectos altruistas no tienen las mismas ayudas que pueden tener los proyectos con fines económicos de por medio. Es por eso, que se espera que sea lo más escalable posible para otras entidades que lo necesitan.
En cuanto al estudiante, este proyecto ha servido para dominar y entender la tecnología de la empresa. A los tres meses y medio de empezar este proyecto el estudiante ya se encontraba con un proyecto real de Redradix.

Referencias

Todas las referencias que se han utilizado son los enlaces que se han mencionado anteriormente. No se ha utlizado ningún tipo de referencia como tal.

Glosario

1. **API**. Application Programming Interface. Una API es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software.

2. **API REST**. APIU que tiene una arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

3. **HTTP**. Protocolo de transferencia donde se utiliza un sistema mediante el cual se permite la transferencia de información entre diferentes servicios y los clientes que utilizan páginas web.

4. **Backup** (o copia de seguridad). Copia de los datos originales que se realiza con el fin de disponer de un medio de recuperarlos en caso de pérdida.

5. **JSON**. JavaScript Object Notation. Formato ligero de intercambio de datos, completamente independiente del lenguaje de programación

6. **Metodología en cascada**. Metodología en la que el planteamiento de todo se hace al comienzo de empezar sin dar pie a cambios en fases anteriores del desarrollo.

7. **VPS**. Se trata de un método de particionar un servidor físico en varios servidores de tal forma que todo funcione como si se estuviese ejecutando en una única máquina.

8. **Framework**. Es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación

Anexos

6.3Manual de instalación

Se trata de una aplicación web por lo que el usuario final lo único que va a tener que hacer es entrar a esta por mediante una URL. Una vez dentro, se tendrá que logear y en el caso de estar en el sistema tendrá unos permisos u otros.
Al ser tan intuitivo, los voluntarios, de momento serán enseñados de palabra de cómo hacer cada tarea. Este se ha decidido así porque realmente es una aplicación que facilita el trabajo que ya se hacía antes.

6.4Manual del programador

La aplicación se encuentra online por lo que para modificarla habría que descargarse el código y hacer las modificaciones oportunas en localhost para después volver a subirlo. Partiendo de este punto se va a explicar lo que el desarrollador debería saber para poder realizar modificadaciones.

- Arrancar la base de datos. Antes de empezar la base de datos tiene que estar lista. Una vez que el programador tiene creada una base de datos local en mongoDb. Para ello se puede seguir el siguiente tutorial[30] donde se puede ver cómo se hace desde cero. Cuando ya se tiene la base de datos creada hay que modificar, en el caso de que sea necesario, para conectar nuestra base de datos con el servidor. En /lib/config.js podemos encontrar algo como:

  ```
  urlDb: 'mongodb://localhost:27017/koringo'
  ```

  Aquí ponemos la ruta que nos haga falta.

- Instalar dependencias. Una vez descargado el código el desarrollador se tiene que descargar las dependencias. Para ello se utiliza NPM.

  ```
  npm install
  ```

- Arrancar el servidor. Para arrancar el servidor bastará con usar el comando desde la carpeta raiz del servidor.

  ```
  nodemon app.js
  ```

[30] https://docs.mongodb.com/manual/installation/

- Arrancar el cliente. Por último hay que arrancar el cliente para poder hacer las operaciones a nivel de usuario. Esto lo hacemos yendo a la carpeta raíz del cliente y ejecutando.

```
npm start
```

- La aplicación está alojada a un VPS y al acceso es confidencial por lo que quien quiera subir código tendrá que consultar al estudiante.

6.5 Anexos

6.5.2 Diseño de la pantalla de fichas con lápiz

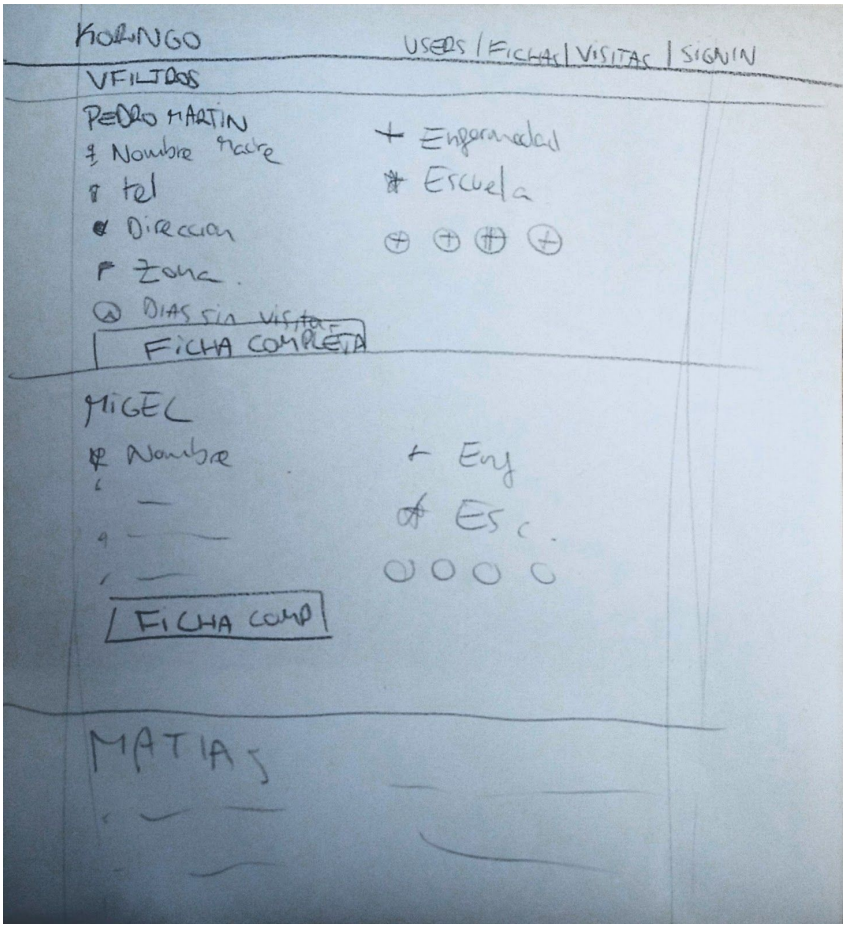

Imagen 10. Fotografía de pantalla fichas. Material propio.

6.5.2 Fichas word antigua

Esta es la ficha que cada voluntario tiene que rellenar cada vez que se realiza una visita.
Si el usuario ya tiene una ficha se escribe sobre esta.

ZONA: FECHA: Nº:

---

1 DATOS PERSONALES

NOMBRE Y APELLIDOS: F. NAC.: D.N.I.:
DATOS DEL RESPONSABLE: (Nombre, DNI y Parentesco)
Nombre: D.N.I.:
Parentesco: Teléfono:
DIRECCIÓN:

2 DATOS FAMILIARES

NOMBRE Y APELLIDOS	FECHA NACIMIENTO	PARENTESCO	ESTUDIOS/ TRABAJO	CONVIVENCIA	OBSERVACIONES

3 EDUCACIÓN - FORMACIÓN

CENTRO ACTUAL: AÑO DE ENTRADA:
CENTROS ANTERIORES:
APOYOS ESPECIALES/TERAPIAS:
ASISTENCIA:
OBSERVACIONES:

4 SITUACIÓN SOCIAL

TIEMPO LIBRE:
CUIDADORES:
APOYOS CERCANOS:
RELACIONES CON EL ENTORNO:

5 DATOS MÉDICOS

DIAGNÓSTICO:
MOVILIDAD:
COMUNICACIÓN:
PRUEBAS REALIZADAS:
TRATAMIENTO:
ENFERMEDADES OTROS MIEMBROS DE LA FAMILIA:
OBSERVACIONES:

6 DATOS DE VIVIENDA

EN PROPIEDAD O ALQUILER: TIPO DE CONSTRUCCIÓN Y ESTADO:
DOTACIONES: Agua Gas Luz Baño TV DVD Mini cadena
Nº Estancias: Nº Camas: Mobiliario:
SALUBRIDAD (Olores-Humedades-Limpieza):
OBSERVACIONES

7 DATOS ECONÓMICOS

Ingresos Familiares:.
Apoyos Externos:
Apadrinamiento Año Comienzo: Comedor (nombre):
Otros:

8 OBSERVACIONES GENERALES

9 NECESIDADES MANIFESTADAS

10 NECESIDADES DETECTADAS