Universidad Autónoma de Madrid
Escuela Politécnica Superior

Proyecto fin de máster

SERVICIO REMOTO DE AUTOMATIZACIÓN DE DOCUMENTOS BASADOS EN PLANTILLAS

Máster en Ingeniería Informática

Guillermo Climent González
Junio 2016

SERVICIO REMOTO DE AUTOMATIZACIÓN DE DOCUMENTOS BASADOS EN PLANTILLAS

AUTOR: Guillermo Climent González
TUTOR: Gonzalo Martínez Muñoz

Dpto. de Ingeniera Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid

Junio 2016

Resumen

Resumen

Actualmente, podría decirse que vivimos en un mundo conectado, la tecnología está presente en todos los ámbitos de la vida diaria, especialmente con los avances de los últimos 10 años. Debido a esto, existen muchos tipos de servicios en la red, pero sin embargo es complicado encontrar herramientas que sirvan para la generación automática de documentos.

Los documentos son muy importantes para muchas aplicaciones, venta de entradas de cine o espectáculos, gestión de mercancías a partir de pegatinas o albaranes, facturas, etc.

Las herramientas habituales no suelen estar diseñadas como servicio, suelen ser simplemente herramientas que ayudan a los programadores a construir documentos dentro de sus aplicaciones y también suelen ser poco verátiles y tipográficamente inexactas. Por eso en este trabajo se presenta una suite de herramientas que permiten desplegar un servicio de generación de documentos basado en plantillas modulares de código LaTeX que solvente algunos problemas de las herramientas existentes.

Palabras Clave

Generación de documentos, plantillas, LaTeX, servicio remoto, aplicaciones web, python, django

Abstract

Nowadays, we live in a connected world, tecnology is present all aspects of our dayly lifes specially after the advances achieved in this last 10 years. Due to this, there are lots of deployed services arround the Internet, but it is still difficult to find automatic document generation tools.

Documents are really important for many modern applications such as ticket selling applications, stickers for package managing, delivery notes, bills...

Usual tools are not designed as a service, they are just tools that help the programer to build documents inside their applications. These tools are usually typographically inexact and little versatile. This reading presents a new set of tools for deploying a service based on modular LaTeX-coded templates for generating documents in applications, solving this way some problems that other tools have.

Key words

Document generation, templates, LaTeX, remote service, web applications, python, django

Agradecimientos

Este trabajo está dedicado a la comunidad, a todo el que necesite una aplicación como la que aquí se presenta. Sigamos haciendo de la web un lugar donde resolver dudas, compartir y ayudarnos unos a otros. También va dedicado a mis padres, a mi hermano, a Noemi y a Gonzalo por ayudarme a terminar este proyecto para poder compartirlo.

Índice general

Índice de figuras

1

Introducción

Vivimos en un mundo conectado. La tecnología está presente en todos los ámbitos de la vida diaria, especialmente con los avances de los últimos 10 años, como los smartphones, las redes sociales y el Internet of Things. Estas nuevas tecnologías, han transformado ampliamente las actividades que diariamente desempeñan las personas, tanto a nivel personal (mensajería instantánea, redes sociales, videojuegos...) como a nivel profesional (organización y compartición de documentos, trabajo colaborativo, BigData, etc.).

La tecnología ha ido sufriendo grandes avances a lo largo de la historia, como la aparición de los primeros ordenadores industriales en los años 70, la aparición del ordenador personal a finales de los 80, Internet, la Web, etc. y ahora nos encontramos en plena revolución. Estamos en un tiempo donde todas las cosas están conectadas, y algo tan simple como un smartphone ofrece infinidad de posibilidades en la palma de la mano y desde cualquier lugar, desde comprar ropa, subir las persianas de casa, llamar a un amigo o compartir en tiempo real lo que está ocurriendo en un lugar determinado. Estamos en la era del aquí y ahora, de todo en cualquier lugar, sin cables.

La tecnología se ha acercado más que nunca al usuario, está pensada para el público, para vender, para sorprender. Esta faceta más comercial y humana de las aplicaciones, está cambiando el paradigma de la informática. Ya no se hacen soluciones monolíticas que resuelven un problema por sí mismas. La necesidad de interconexión entre las aplicaciones está llevando a definirlas como servicios.

Estos servicios se definen como sistemas software diseñados para soportar interacciones máquina-máquina a través de una red **??**. En definitiva, estos sistemas proveen interfaces que permiten a aplicaciones consultar información o solicitar que el servidor realice una cierta tarea.

Existen muchos tipos de servicios en la red, como: servicios de autenticación, desplegado de aplicaciones, datos bursátiles, CDNs, bibliotecas de imágenes, etc.

La gran ventaja de los servicios es que son independientes de la aplicación que contacta con ellos, por lo que se desplaza carga de procesamiento fuera del cliente. También estos servicios se diseñan para ser puedan dar servicio a multitud de aplicaciones independientemente de la naturaleza de estas.

De esta manera, se obtiene por diseño, un sistema interconectado y modular que es fácil de desarrollar, barato de mantener y con una estructura que respeta el hecho de poder disponer de todo lo que la aplicación necesita desde cualquier lugar y en cualquier momento sin carga de procesamiento en el cliente.

1.1. Motivación del Proyecto

Los documentos son algo fundamental para muchas aplicaciones. Como pueden ser la venta de entradas de cine o espectáculos, la gestión de mercancías a partir de pegatinas o albaranes, la expedición facturas, etc.

Las herramientas habituales no están diseñadas como servicios. Únicamente existen librerías que ayudan a los programadores a construir documentos dentro de sus aplicaciones. En el caso de que quieran ser usadas desde un cliente remoto, deben apoyarse en otras tecnologías o incluir esta funcionalidad en otros servicios. Al ser librerías, también es habitual que solo puedan ser usadas empleando un lenguaje de programación determinado.

Otros problemas que suelen tener estas librerías es que normalmente se basan en un solo formato de entrada como HTML y producen un solo formato de salida, usualmente PDF, lo que reduce su versatilidad. También están muy limitadas en cuanto a cómo debe el programador suministrar la información a la librería, normalmente se hace de un modo que mezcla el aspecto físico del documento con su contenido. Este diseño no solo produce que el código sea menos mantenible, ya que hay que realizar cambios en él cada vez que se quiera actualizar el aspecto físico de un documento par aun mismo contenido. Además estas librerías suelen utilizar formatos de entrada propios lo que hace que tengan una funcionalidad de representación limitada.

En cambio, LaTeX es una manera muy fiable de generar documentos. Tiene un lenguaje propio y diseñado especialmente para la descripción de documentos. A través de sus paquetes y extensiones puede colocarse en el documento prácticamente cualquier cosa, como códigos de barras, fórmulas matemáticas, tablas, o incluso grafos. Sin embargo, es una aplicación de consola que para el usuario medio puede resultar difícil de aprender, configurar e instalar.

La motivación principal de este proyecto es proveer a la comunidad de un servicio libre, potente y versátil para la generación de documentos basados en LaTeX, que convine facilidad de uso con una gran funcionalidad de representación.

1.2. Objetivos del proyecto

La solución que se quiere aportar con este TFM tiene por objetivos:

- Crear una solución que pueda ser desplegada en un servidor como servicio de generación de documentos. Además debe poderse usar en forma de librería de la manera tradicional. También debe contener algún elemento que permita generar documentos a través de interfaz de comandos, con el fin de que los usuarios puedan usar la funcionalidad básica sin necesidad de desarrollar una aplicación.
- La solución debe ser versátil. Debe soportar múltiples formatos de entrada de datos y producir diferentes formatos de salida. El servicio que se despliegue debe ser también accesible de una forma sencilla, independientemente de la naturaleza y lenguaje de la aplicación que haga uso de él. También debe utilizar protocolos estándar en la medida de lo posible y contar con las debidas medidas de seguridad.

- Al configurar la aplicación, el contenido del documento debe estar separado de la forma para que la descripción del documento sea más clara. Los documentos deben poder estar creados a partir de componentes reutilizables y anidables que permitan la reutilización.
- La solución debe estar debidamente empaquetada y acompañada de documentación para que la comunidad pueda hacer uso de ella. Debe estar también distribuida bajo una licencia que permita su uso de forma gratuita por parte de la comunidad.

1.3. Solución propuesta

El proyecto implementado, al que hemos denominado Spellbook, es una suite de diferentes herramientas que permiten la generación remota de documentos. Cada una de estas herramientas implementadas, ofrece una capa en torno a la anterior, añadiendo funcionalidad y permitiendo que puedan ser utilizadas por separado dependiendo del las diferentes necesidades:

- **Pdc**: Es un módulo en python que incluye todas las rutinas necesarias para llevar a cabo el proceso básico de generado de documentos. Es el núcleo de todo el sistema y puede incorporarse como librería a aplicaciones de terceros. También contiene un ejecutable que implementa muchas de las funcionalidades de la librería y que permite construir documentos directamente desde la consola, como si de un compilador tradicional se tratara.
- **Pdcs**: Es un servicio REST JSON que envuelve a pdc. Permite utilizar todas las funciones de pdc de forma remota. También incluye una consola web de configuración, gestiona la autenticación y autorización y añade funcionalidades como la posibilidad de editar las plantillas desde el propio navegador web.
- **sPHPellbook**: Es un cliente del servicio REST de pdcs que permite generar documentos de una manera sencilla desde PHP mediante la conexión a un servicio pdcs.

El diseño modular de cada uno de los elementos de Spellbook permiten usar el sistema de diferentes modos, según sea necesario en cada situación:

- Si se necesitan generar los documentos de forma local, puede utilizarse la aplicación por consola.
- Si se pretende construir una aplicación python que genere documentos, puede importarse la librería pdc y utilizarla para generar documentos desde esta.
- Para aplicaciones web, puede usarse el servicio REST remoto de pdcs. De esta forma se puede dar servicio a varias aplicaciones escritas en diferentes lenguajes a la vez y de forma centralizada. Un caso concreto de esto sería utilizar sPHPellbook para conectarse al API JSON de pdcs desde PHP.

Como complemento a estas herramientas principales, se han desarrollado también otros elementos que son necesarios para que otros programadores puedan usar Spellbook como parte de sus soluciones:

- **Environment**: Es un conjunto de scripts bash que instalan un servidor pdcs completo sobre cualquier máquina con Ubuntu 14.04 o superior. Instalan todas las librerías necesarias, configuran el servidor web y de base de datos y dejan el sistema listo para funcionar. Están diseñados para ser utilizados como script de aprovisionamiento para una máquina virtual Vagrant, con la que puede desplegarse un servidor en cuestión de minutos.

- **Aplicación de prueba**: Como pequeña demostración del sistema, se ha construido una aplicación en PHP que hace uso del cliente sPHPellbook para generar dinámicamente albaranes de envío.
- **Documentación web**: Para explicar el funcionamiento completo de Spellbook, se ha desarrollado también una documentación en formato web que expone paso a paso el funcionamiento del sistema a través de la codificación de un ejemplo práctico.

1.4. Público del proyecto

En general Spellbook puede ser útil para el desarrollo de cualquier aplicación que genere documentos, ya sea web, de escritorio o de consola. Sin embargo, destaca sobre todo en los siguientes ámbitos:

- **Industria**: Spellbook puede ser útil para programadores que debido a las necesidades de las soluciones que tengan que desarrollar, necesiten una aplicación de generación de documentos de gran potencia y precisión. Como por ejemplo soluciones industriales complejas como códigos de barras o documentos de identificación que no pueden generarse fácilmente con librerías convencionales.
- **Servicios TI**: El diseño de servicio remoto de Spellbook puede ser de utilidad para los servicios TI de empresas que quieran utilizarlo para generar su formularios y documentación corporativa, centralizando la generación. El carácter modular de la descripción de los documentos en Spellbook permite reutilizar material y cambiar todo el aspecto corporativo de los documentos generados fácilmente.

2

Estado del arte

A continuación, se va a explicar qué circunstancias tecnológicas han inspirado la creación de las herramientas descritas en este documento. En concreto, se hablará del pasado y el presente de los servicios web así como de las diferentes tecnologías de generación de documentos por ordenador y su importancia en las actividades industriales, económicas y domésticas.

2.1. Servicios web

2.1.1. Evolución de la web

Desde que hizo su aparición en 1990 en el CERN de mano de Tim Berners Lee hasta nuestros días, la web ha sido el motor de Internet, experimentado una serie de cambios tanto en tamaño como en funcionalidad que la han transformado en lo que es hoy [9].

Comenzó siendo un medio para científicos y profesionales, sobre el que poder compartir información entre universidades y centros de investigación. Esta web primigenia, era solo accesible para un selecto grupo de personas que tenían los conocimientos y el equipamiento para poder usarla [24]. Más tarde, en 1991, el CERN decidió hacer pública la web, sus protocolos y funcionamiento. La web era accesible a través de Internet a todo el mundo, pero seguía sin serlo para el público en general. Los contenidos que se mostraban en la web eran suministrados por los denominados Webmasters, personas que sabían como crear las páginas y publicarlas, o bien eran de uso puramente científico o empresarial. Era una web abierta, pero aún unidireccional, unos pocos individuos publican contenido y un grupo mucho más numeroso lo consume. Lo que hoy llamamos la web 1.0 [37].

Paralelamente a este uso de Internet, muchas empresas e instituciones empezaron a usar la red para desplegar servicios. Estas aplicaciones debían abastecer una creciente demanda y debían ser capaces de integrar clientes con lenguajes de programación variados. Para hacer frente a este reto, se desarrollaron tecnologías como CGI, CORBA, EDI, RPC o más tarde SOAP que facilitaban la programación de soluciones en diferentes lenguajes que se ejecutaban en distintas máquinas [45].

Poco a poco, la tecnología va haciendo más accesible Internet. Las conexiones domésticas se vuelven más rápidas y baratas, los ordenadores se hacen más potentes y se popularizan nuevas formas de usar Internet, como el correo electrónico. La web empieza a dar un paso hacia la web 2.0, donde las personas conectadas participan no solo consumiendo contenido, sino también aportándolo en formatos como blogs o redes sociales. Este cambio de paradigma, apoyado en cierta medida por los nuevos avances en tecnología doméstica como los smartphones, las tablets y las televisiones inteligentes están ahora cambiando las estructura de Internet de nuevo [31].

2.1.2. Aparición y desarrollo de los servicios web

Tener una aplicación web para que los usuarios la utilicen ya no es suficiente.. Un buén ejemplo de esto son las aplicaciones nativas para iOS y Android de grandes empresas como Facebook o Twitter. Estas aplicaciones fueron lanzadas para mejorar la experiencia de los usuarios en terminales móviles evitando que utilizasen el navegador web ya que con una aplicación nativa, se desperdician menos recursos del dispositivo y por tanto puede ofrecerse mayor funcionalidad y velocidad mejorando así la experiencia de usuario.

Esto fuerza a los desarrolladores a estructurar sus soluciones en capas, situando un backend al que puede conectarse todo tipo de frontend, ya sean aplicaciones web o nativas. De esta forma solo se programa un backend al que se conectan todas las aplicaciones cliente, ahorrando en costes y mejorando la mantenibilidad del sistema.

Esto unido al auge de la computación en cloud y a las tecnologías de virtualización y de contenedores han forzado que las tecnologías de despliegue y comunicación de los servicios se modernicen a gran velocidad, haciendo uso en muchos casos de protocolos simples y accesibles desde cualquier lenguaje como HTTP dando lugar a los denominados servicios REST.

Todas estas circunstancias ha hecho que los desarrolladores de grandes servicios de Internet hagan públicas sus API, como Facebook [8] o Twitter [43] de forma que otras aplicaciones pueden integrarse enriqueciendose entre sí.

Con las herramientas que se presentan en este documento, se pretende hacer de la generación de documentos un servicio más, que permita con un único servidor dar servicio a varias aplicaciones, además de proveer un servicio REST fácil de integrar independientemente del lenguaje de programación del cliente.

2.1.3. Tecnologías en los servicios web

Hoy en día existen multitud de herramientas para crear y desplegar servicios web, asociadas a las tecnologías de desarrollo web más comunes, por lo que hay una gran variedad de posibilidades a elegir. Pero sin duda, la mayor elección que hay que tomar a la hora de diseñar un servicio web es escoger que tipo de servicio se necesita desplegar. Actualmente los dos tipos de servicio más extendidos son los SOAP y los REST.

Servicios SOAP

SOAP (Simple Object Access Protocol) es un protocolo estándar que define la interacción entre objetos de distintos procesos a través del intercambio de mensajes unidireccionales XML sin estado. Este protocolo puede utilizarse para comunicar objetos en diferentes servidores a través de una conexión HTTP dando lugar así a un servicio web.

La naturaleza de los objetos que se intercambian se define en un fichero común, el WSDL (Web Services Description Language). En él se detallan las distintas características de los objetos que se publican mediante el servicio. Cuando el cliente se conecta al servidor, utiliza el WSDL para condificar y decodificar los objetos intercambiados.

La transmisión de los objetos puede hacerse mediante cualquier protocolo que soporte el envío de texto, tradicionalmente se usa HTTP.

Las ventajas que presenta son:

- Al utilizar HTTP es fácilmente integrable, en diferentes sistemas. Existen también librerías que implementan el protocolo para casi todos los lenguajes de programación.
- La transmisión puede utilizar cualquier protocolo de transporte capaz de transmitir texto.

En cambio, también presenta desventajas importantes:

- SOAP depende de XML, cuyo parseo es relativamente lento, además de ser muy dificil de leer para una persona y necesitar una infraestructura software relativamente compleja para interpretar las respuestas. Los datos binarios son forzados a transformarse en texto, lo que ralentiza aún más las operaciones.
- Depende del WSDL para que el cliente y el servidor conozcan la definición de los objetos.

Servicios REST

REST (Representational State Transfer) no es un protocolo propiamente dicho, es un patrón de diseño de arquitectura software orientado a la comunicación vía red y es utilizado para definir servicios.

REST estructura su funcionamiento en torno al concepto de recursos. Un cliente se conecta al servicio para acceder a estos recursos, modificarlos, borrarlos o incluso realizar acciones sobre ellos. Cada uno de estos recursos está identificado por una dirección (URI) que el cliente debe conocer para poder realizar una petición. Las operaciones que se pueden realizar sobre los recursos están bién definidas y se basan en las operaciones CRUD (create, read, update y delete).

Para comunicarse, el cliente se pone en contacto con el servidor utilizando el protocolo HTTP e intercambiándose representaciones de estos recursos. Estas representaciones están codificadas de acuerdo a una especificación concreta, típicamente XML o JSON.

Dicho intercambio de estos recursos es *stateless* corresponde a la aplicación dar sentido y entender los datos intercambiados así como realizar las peticiones necesarias en cada momento.

Los servicios REST presentan una serie de ventajas:

- El punto fuerte de los servicios REST es su simplicidad, al utilizar HTTP no es necesaria una infraestructura software muy grande para navegar por un servicio de este tipo. Por lo que son servicios fáciles de integrar y depurar.
- El hecho de que se pueda definir libremente la arquitectura de las URI, los recursos y la codificación de las representaciones, hace también que las aplicaciones sean más ligeras, al poder definir con versatilidad sus estructuras.
- Tener que definir una estructura y codificación concretas y dependientes de la aplicación también produce que los desarrolladores del servicio se vean obligados a documentar de una manera exahustiva el servicio, lo que luego hace más fácil la labor de integración.

Sin embargo, estas características que hacen de los servicios REST muy adaptados a la aplicación, también suponen que el sistema debe estar bien diseñado. Es fácil cometer fallos de diseño cuando todo desde la codificación de las representaciones hasta la arquitectura de la aplicación depende del programador.

A pesar de esta dificultad, este tipo de servicio ha sido el escogido para implementar el sistema que se presenta en este informe, ya que se busca ante todo la simplicidad y la facilidad de integración independientemente de la naturaleza de la aplicación cliente.

2.2. Generación de documentos

Tratar con documentos es algo habitual en las aplicaciones, especialmente en los ámbitos profesionales. De hecho, unas de las primeras aplicaciones de la todavía informática primigenia, fué una máquina que automatizaba la burocracia necesaria para gestionar el censo de los Estados Unidos [2].

Esta máquina inventada en 1890 por Herman Hollerith usó álgebra de Boole y un documento de entrada en forma de cartulina impresa y perforada donde los usuarios contestaban preguntas binarias. Con este método, el resultado del recuento y análisis censal de los 62.622.250 habitantes estuvo listo en sólo 6 semanas. La empresa que se desarrolló con el éxito de la máquina (figura 2.1), tras muchos cambios, daría lugar a lo que hoy conocemos como IBM.

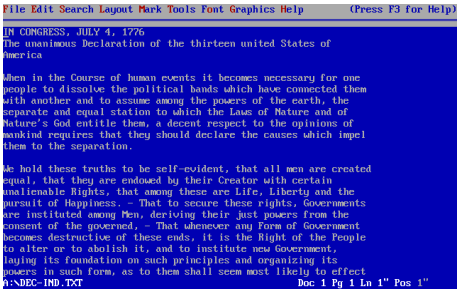
Figura 2.1: Máquina tabuladora de Herman Hollerith

2.2.1. Aplicaciones de procesamiento de textos

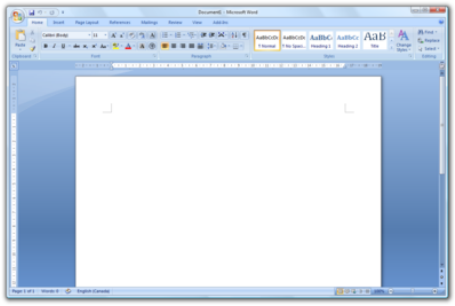
Con la aparición de los primeros ordenadores y su posterior miniaturización hasta los ordenadores personales, se siguieron construyendo soluciones para ayudar a las empresas a manejar el gran volumen de documentación que necesitaban.

Los primeros programas de generación de documentos por ordenador no fueron generadores automáticos, fueron las aplicaciones de procesamiento de textos, que surgieron como alternativas modernas a las máquinas de escribir. Los documentos podían almacenarse digitalmente, por lo que podían modificarse y copiarse.

Una de las primeras aplicaciones de este tipo, WordPerfect (figura 2.2a), hizo su aparición en 1979 [44]. Después lo siguió el Microsoft Word(figura 2.2b) [26] en 1981. Poco a poco, toda la documentación generada por las empresas empezaba a ser digitalizada.



(a) Word Perfect 1.0



(b) Microsoft Word 2007

Figura 2.2: Ejemplos de aplicaciones de procesamiento de textos

Al principio, estos programas de procesamiento de textos solo eran utilizados por personal que había sido formado para usarlos, dada la complejidad de las interfaces de los ordenadores de la época. Un ejemplo de este tipo de interfaces se puede observar en la figura 2.2a, donde está representada la pantalla principal de WordPerfect en su primera versión. No contaba con soporte para ratón y la interacción del usuario era sobre todo a través de atajos de teclado.

Sin embargo, las empresas buscaban que este tipo de programas fueran accesibles al público en general independientemente de su nivel de familiaridad con los ordenadores y así no tener que contratar personal especializado o formar a sus trabajadores.

Esto fomentó que tras la aparición de conceptos como el escritorio, el puntero del ratón o las ventanas, se desarrollara el paradigma What You See Is What You Get (WYSIWYG). Esta forma de diseñar interfaces se basa en mostrar al usuario por pantalla el aspecto real de aquello que está construyendo, y proveerle de herramientas que le permitan editarlo a la vez que ve los cambios que realiza.

De esta forma, los editores de texto evolucionaron hacia interfaces que eran más usables y que permitían al usuario ver en tiempo real una vista de lo que sería su documento al imprimirlo. Un ejemplo de este tipo de interfaz es la figura 2.2b, donde puede verse la pantalla principal del Microsoft Word 2007. Esta versión ya permite colocar los elementos sobre el espacio y hacer todo tipo de configuraciones utilizando el ratón. De esta manera se hace más simple y fácil de utilizar para el público en general.

Sin embargo, al buscar la sencillez y la interactividad con el usuario, desembocó en un software con una funcionalidad más reducida y aplicada al usuario medio. La calidad tipográfica no era suficiente para colectivos como científicos o editoriales.

Los formatos que tratan este tipo de aplicaciones son también inestables entre versiones del mismo programa o programas distintos y no son apropiadas para la publicación de los documentos.

2.2.2. TeX y LaTeX

Paralelamente al desarrollo de las aplicaciones de procesamiento de textos, se desarrollaron otra serie de aplicaciones con una base distinta al WYSIWYG, como es el caso de TeX y LaTeX.

En 1969 Donald Knuth, escribió su libro The Art of Computer Programming. Esta publicación fué inicialmente imprimida utilizando la técnica de presionado de caracteres. Esta técnica era la utilizada en el siglo XIX, y se basaba en imprimir el documento con grandes máquinas de escribir que utilizaban cabezales metálicos con caracteres grabados que eran impregnados en tinta y presionados contra el papel. Esta técnica de impresión producía un resultado clásico, con una gran exactitud y calidad que era del agrado de Knuth.

Sin embargo, una versión posterior fué impresa por fotocomposición ya que la tecnología anterior había sido remplazada y las fuentes originales ya no estaban disponibles. Esta técnica consiste en utilizar negativos fotográficos de cada uno de los caracteres y utilizarlos para componer el texto. Pero la calidad y la exactitud de la tipografía eran muy inferiores al otro método. Esto inspiró a Knuth a crear su propio sistema software de composición de documentos [22], lo que hoy se conoce como TeX.

TeX fué creado con dos objetivos en mente:

- Permitir que cualquiera pueda con el software adecuado crear libros con una calidad gran calidad tipográfica con relativamente poco esfuerzo.
- Hacer que el sistema produzca siempre la misma salida independientemente de la naturaleza del mismo y del momento en el que se realice la generación del documento.

Para alcanzar estos objetivos, Knuth diseñó un lenguaje de descripción vectorial los caracteres de las fuentes, lo que llamó METAFONT [21]. Este sistema, describe las fuentes no como el contorno de los caracteres, sino como el camino de un pincel de una forma determinada sobre el lienzo (figura 2.3). De esta forma, los caracteres están codificados en forma de ecuaciones que regulan el movimiento del pincel, por lo que se les pueden aplicar parámetros como el ratio de aspecto, el tamaño o el grosor de los bordes.

```
%file name: beta.mf
%mode_setup;
% Define a beanlike shape for the character B
beginchar("B",11pt#,11pt#,0);
  % Setup coordinates as an equation system
  y1=y2=y3=0;
  y4=y5=y6=h;
  x1=x4=0;
  x2=x5=w;
  x3=x6=2*w;

  % Define pen
  pickup pencircle xscaled 0.2w yscaled 0.04w rotated 45;

  % Draw the character curve
  draw z1..z3..z6{z2-z6}..z5..{z4-z2}z4..cycle;
endchar;

end
```

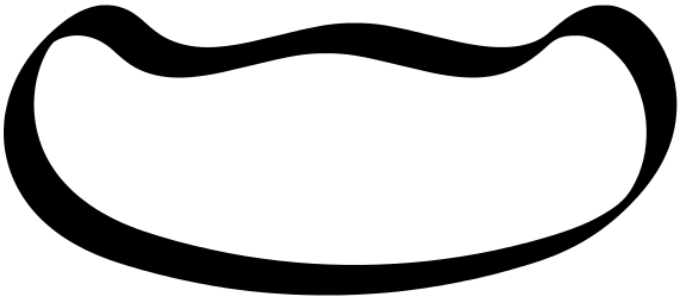


Figura 2.3: Ejemplo de definición de un caracter utilizando una función matemática en METAFONT

Junto con este lenguaje de descripción de fuentes, diseñó también un lenguaje para definir la estructura y el contenido del documento. Este lenguaje está basado en comandos y macros. De esta forma, con un editor de texto plano y un compilador de TeX, cualquiera era capaz de generar un documento con una gran calidad tipográfica a través de un sistema estable y expandible.

Sin embargo, el lenguaje definido por Knuth para el diseño del documento era de muy bajo nivel, pero en 1985 Leslie Lamport creo LaTeX [23]. Esta aplicación derivada de TeX es un sistema de generación de documentos completo, donde el usuario utiliza un lenguaje de marcado de nivel más alto construido sobre comandos de TeX.

Es altamente personalizable, ya que pueden redefinirse los macros y comandos y se pueden añadir nuevos a través de paquetes. Estos son piezas software que se integran en LaTeX y permiten añadir funcionalidad a través de nuevos comandos o macros, como por ejemplo mejorar la sintaxis de ciertos elementos como tablas, o añadir otros como códigos de barras, caracteres de otros idiomas, grafos o fórmulas matemáticas.

Hoy en día LaTeX es muy utilizado en ambientes académicos, científicos y editoriales para obtener de una manera sencilla documentos de cualquier tipo y con una gran variedad de elementos representables sin perjuicio de una alta calidad tipográfica.

2.2.3. Publicación de documentos

Dado el importante aumento de los documentos digitales, y la inestabilidad de los formatos de las diferentes aplicaciones, era necesario transformar los documentos a un formato que fuera estable independientemente del dispositivo donde se visualizara. Para ello se creó PostScript, un lenguaje de descripción de gráficos vectoriales [18]. Fue desarrollado por un equipo en Adobe Systems e hizo su aparición en 1982. Se utilizaba, entre otras cosas, para transformar los documentos en una imagen vectorial cuyo aspecto fuera calculado y representado siempre de la misma manera.

Sin embargo, este proceso de transformación requería mucho tiempo y recursos para llevarse a cabo, de forma que en 1991 Adobe Systems desarrolló el formato de publicación de documento por excelencia, el PDF (Portable Document Format) [40]. Este formato es específico para documentos y permite representarlos en los principales sistemas operativos sin que se modifique el aspecto ni la estructura original, además una vez exportado, no permite la edición del documento.

Hoy en día, la importancia del PDF es muy grande, dado que la gran mayoría de aplicaciones que tratan y publican documentos utilizan este formato.

2.2.4. Edición colaborativa de documentos

Lejos de desaparecer, la importancia de los documentos en la vida diaria va en aumento Una prueba de ello es la aparición multitud de herramientas que muestran nuevas formas de creación de documentos, como por ejemplo es el caso de Google Docs y Overleaf que permiten el desarrollo colaborativo, síncrono y en tiempo real de documentos, convirtiéndolos también en un elemento de comunicación.

El caso de Overleaf es muy interesante, puesto que permite que varias personas editen en tiempo real código LaTeX y ofrece una vista previa del documento que se va construyendo, tal y como se aprecia en la figura 2.4. También tiene un editor de LaTeX especializado que hace más fácil encontrar errores, además de incluir plantillas predefinidas y sistemas integrados de publicación [32].

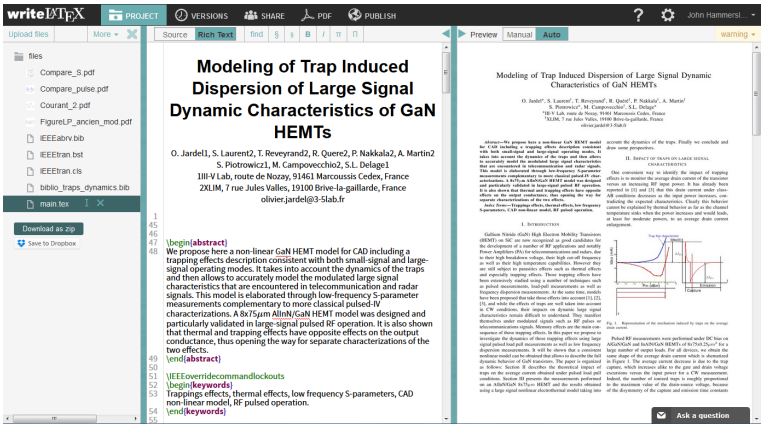


Figura 2.4: Imagen de la interfaz de Overleaf

2.2.5. Herramientas existentes para la generación de documentos

Como se comentaba en el apartado anterior, el hecho de que Internet esté cada vez más al alcance de todos y el auge de los servicios a través de Internet y de estas nuevas formas de hacer uso de documentos, ha producido también herramientas parecidas a la que se presenta en este documento, orientadas a ser incorporadas en estas aplicaciones que necesiten manipular documentos.

Un ejemplo de este tipo de herramientas es jsPDF [17]. Es un framework javascript que se ejecuta en el navegador del cliente permitiendo al programador generar documentos PDF sin hacer uso del servidor. El documento se va generando mediante una serie de rutinas para escribir texto o colocar imágenes dando lugar finalmente a un documento PDF en formato dataURI. Un ejemplo de su uso puede verse en la figura 2.5

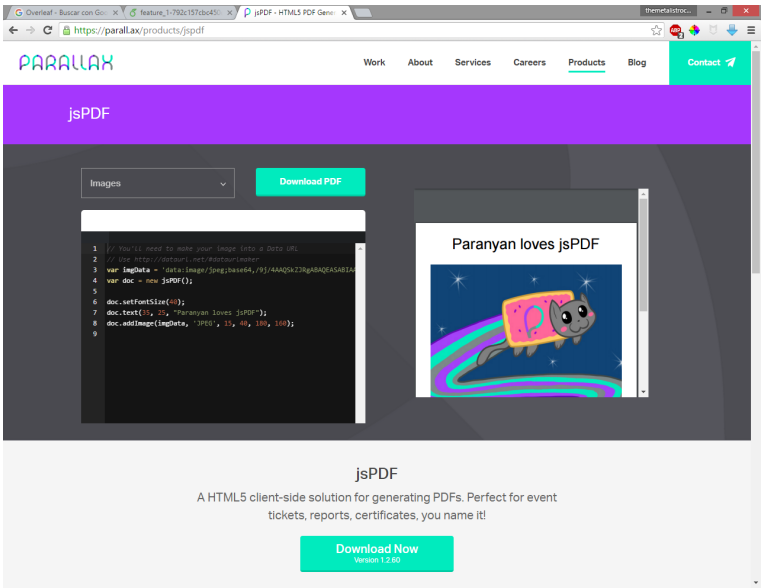


Figura 2.5: Ejemplo de uso de la librería jsPDF

Esta librería presenta varias ventajas, como que es muy sencilla de utilizar y configurar y que todo el trabajo de construcción y renderizado se realiza en el cliente. Sin embargo, las funciones para construir el documento son bastante limitadas y la exactitud y la calidad tipográfica de los documentos obtenidos no pueden compararse con documentos diseñados a mano.

Otra popular herramienta para generar PDFs es WickedPDF. Esta librería recibe un modelo en HTML del documento, renderizándolo a PDF y sirviéndoselo al usuario a través de peticiones HTTP. Está escrita en Ruby y está especialmente diseñada para ser utilizada junto con Rails [36]. Para el renderizado utiliza un binario externo a la aplicación Rails, wkhtmltopdf.

La calidad de los documentos obtenidos con esta herramienta es muy buena, y la sintaxis para controlar el renderizado desde el layout HTML también es muy completa. Sin embargo, esta librería no controla las referencias a ficheros CSS y JS para el renderizado mediante la librería externa y no tiene la variedad de elementos que puede tener LaTeX, como fórmulas matemáticas, códigos de barras, medidas en centímetros...

2.2.6. Conclusiones

En definitiva, estas librerías han sido la inspiración para crear una nueva solución para el generado de documentos que no tenga,en la medida de lo posible, las limitaciones y aspectos negativos de las herramientas que existen:

- Lograr una gran calidad tipográfica y variedad de elementos utilizables haciendo uso de LaTeX e intentando hacer la configuración y el sistema lo más sencillo y autocontenido posible.
- Dividir el documento en vista y contenido, haciendo uso de elementos modulares que ayuden a reutilizar y ordenar las plantillas y que permitan un rápido cambio de estilo de todos los documentos de un mismo tipo.
- Permitir una variedad de formatos de entrada y salida más allá del HTML y el PDF, para dotar de mayor versatilidad a las aplicaciones.

3

Diseño e implementación

En este apartado se va a exponer cómo funcionan cada uno de los componentes que conforman el sistema implementado, así como las tecnologías que se han utilizado y las razones de diseño que han motivado su elección.

En primer lugar se hablará sobre el motor de Spellbook, pdc, encargado de generar los documentos. Después se comentará el funcionamiento de pdcs, el servicio web implementado sobre pdc para poder acceder a sus funciones a través de la web. Una vez explicado el servicio, se continuará exponiendo sPHPellbook, que es un cliente del servicio web escrito en PHP. Por último se comentará la construcción de la aplicación de prueba implementada, la documentación web de todo el sistema y los scripts de desplegado e instalación.

3.1. Pdc: Librería básica y aplicación de consola

Pdc es una librería escrita completamente en python, que implementa la funcionalidad básica de la aplicación, acompañada de un ejecutable también escrito en python que permite el acceso las funciones principales desde la consola.

La razón de escoger este lenguaje de programación para esta parte de la aplicación reside en varios puntos:

- Pdc necesita una librería de plantillas para generar el código de los documentos. En el caso de python, existe Jinja2 que se basa en el motor de plantillas del framework web Django y que permite usarlo fuera de entornos web [34]. Es un motor versátil, optimizado y fácil de utilizar, por eso se decidió seleccionarla como corazón para el generador de documentos.
- Python es un lenguaje que combina una sintaxis clara y sencilla con una funcionalidad, portabilidad y compatibilidad muy grande. Pdc necesita llamar contínuamente a programas externos e interactuar con el SO, estas operaciones pueden realizarse de forma más simple, robusta y rápida en python que en otros idiomas similares como Java o Ruby.
- Python también es un lenguaje muy extendido y goza de una amplia comunidad, lo que hace más fácil el aprendizaje y la búsqueda de información, además de ser también un proyecto de código abierto.

A lo largo de este apartado, se explicará con qué estructuras se definen los documentos a generar y en qué consiste y cómo está implementado el proceso de generación de documentos.

3.1.1. Estructura de los documentos: Vista y datos

Como se ha comentado en apartados anteriores, Spellbook basa el generado de documentos en la diferenciación de la vista y de los datos del documento. De esta manera puede separarse el aspecto físico de un documento de los datos que contiene haciendo la definición de los documentos más sencilla y reutilizable. A continuación van a exponerse estos dos elementos, sus estructuras y su funcionalidad.

Vista: Plantillas y recursos

La vista equivale a la información que permite al generador organizar los datos en el documento, cómo colocarlos y mostrarlos físicamente sobre el espacio. La vista se compone de dos elementos: las plantillas, que se combinan para generar el código LaTeX del documento y los recursos que representan otros elementos como imágenes.

- **Plantillas**: Las plantillas conforman el código LaTeX del documento. Son individuales, pero pueden anidarse y repetirse. Esta estructura permite hacer componentes reutilizables en diferentes documentos con el mismo estilo.

  No hay que olvidar que el objetivo final de la vista es generar un código LaTeX que pueda ser transformado. Esto lleva a estructurar la vista en dos tipos de plantillas:

  - **Componentes**: Son las plantillas que conforman cada una de las partes del documento. La sintaxis de estos componentes es la de código LaTeX corriente, salvo que pueden incluirse estructuras de control como condicionales o bucles y también variables basadas en la sintaxis de Jinja2, a través de las cuales se "rellena" el fragmento de documento con diferente contenido según sean los datos que acompañan al documento. En la figura 3.1 se pueden ver expresiones del tipo {% *conf.variable* %} que son sustituidas por los datos correspondientes.

    Los componentes son anidables, secuenciables y repetibles a lo largo de todo el documento. De esta forma se crea una estructura en forma de árbol que da la forma final al documento como se verá más adelante.
  - **Paquetes**: LaTeX utiliza paquetes para realizar funcionalidades extra. Estos deben instalarse en el sistema, y para ser usados requieren una pequeña configuración en la sección de cabecera del código LaTeX antes del cuerpo del documento, como puede verse en la figura 3.2.

    Los paquetes de pdc son entonces, fragmentos de código especiales que se usan para incluir estos paquetes LaTeX en el código. Se asocian a componentes pero no permiten anidación. Son sólo incluidos en la cabecera del código generado una única vez, independientemente del número de veces que aparezcan en los diferentes componentes que forman el documento.

- **Recursos**: Son cada una de la imágenes que pueden ser incluidas en el documento final. En el futuro se añadirán en esta categoría cualquier otra serie de elementos que acompañen a los datos y puedan ser incluidos en los documentos generados.

```
%-----------------------------------------------------------------------------------
%   DELIVERY NOTE HEADER
%-----------------------------------------------------------------------------------
\textsf{
    \begin{tabular}{m{8cm}rrrr}
        \textbf{\Large {@ conf.title @}\hfill}\hspace*{1.25cm}\large {@ conf.code @}\hfill &
        \hspace*{0.5cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(1cm,1cm)
                    \psbarcode{{@ conf.qr_content @}}{}{qrcode}
            \end{pspicture}} &
        \hspace*{0.25cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(3,1cm)
            \psbarcode{{@ conf.bar_content @}}{includecheck height=0.7}{interleaved2of5}
            \end{pspicture}}
         \\
    \end{tabular}
}
%-----------------------------------------------------------------------------------
%   END DELIVERY NOTE HEADER
%-----------------------------------------------------------------------------------
```

Figura 3.1: Ejemplo de código LaTeX con expresiones de control de Jinja2. Modeliza la cabecera de un documento.

```
\usepackage[spanish]{babel}
\selectlanguage{spanish}
```

Figura 3.2: Ejemplo de código LaTeX de un paquete. Es incluido al inicio del código LaTeX del documento.

Datos: Árbol de contenido

Los datos se corresponden con la información que compone y da sentido al documento, que define qué componentes, paquetes y recursos utilizar y qué información introducir en ellos.

Los datos se expresan mediante un árbol de contenido. Cada uno de sus nodos es un componente con una configuración asociada, unos paquetes y un contenido que incluye otros nodos hijos que se introducen en el interior de ese nodo.

La figura 3.3 representa el árbol de un albarán de entrega, compuesto por un documento base que contiene una cabecera de empresa, una cabecera de albarán y el contenido del albarán.

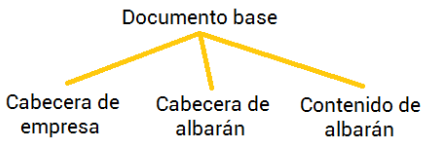


Figura 3.3: Ejemplo de estructura del árbol de documento de un albarán de entrega.

Todos los documentos deben contar con un nodo raíz que actúa como documento base. Esta plantilla es especial y también sirve como modelo para el documento LaTeX. En su interior se escriben las inclusiones de los paquetes LaTeX que necesiten el resto de nodos y el código de todos los componentes del documento.

El árbol de contenido completo del albarán quedaría tal y como se muestra en la figura 3.4. En él puede observarse como se definen cada uno de los nodos, incluyendo qué plantilla modeliza el nodo, los datos a sustituir en dicha plantilla, los paquetes de los que depende y qué otros nodos contiene.


```
{
    "content": [{
        "content": [],
        "packages": [{
            "content": [],
            "template": "packages\/pst-barcode",
            "conf": []
        }, {
            "content": [],
            "template": "packages\/fancyhdr",
            "conf": []
        }, {
            "content": [],
            "template": "packages\/babel",
            "conf": []
        }, {
            "content": [],
            "template": "packages\/tabularx",
            "conf": []
        }],
        "template": "components\/header\/companyHeader",
        "conf": {
            "image": "logo",
            "image_width": 6,
            "image_height": 2,
            "institution_title": "Spellbook, Generadores de documentos",
            "company_address_1": "C\/Alcala S\/N",
            "company_address_2": "28036, Madrid, Spain",
            "company_address_3": "Tlf.: +34 91 568 25 25"
        }
    }, {
        "content": [],
        "packages": [],
        "template": "components\/deliveryNoteHeader",
        "conf": {
            "title": "ALBARAN\t   DE   ENTREGA",
            "code": "A-YYYYMMDD-OR-CLI",
            "qr_content": "A-YYYYMMDD-OR-CLI",
            "bar_content": "201500121"
        }
    }, {
        "content": [],
        "packages": [],
        "template": "components\/deliveryNoteContent",
        "conf": {
            "date": "12\/12\/2016",
            "client": "Spellbook",
            "id": "Nota de prueba!!",
            "products": [{
                "id": "123456",
                "description": "Descripcion del producto",
                "snumber1": "7635438yGTA6785",
                "snumber2": "SN00000001"
            }, {
                "id": "123456",
                "description": "Descripcion del producto",
                "snumber1": "7635438yGTA6785",
                "snumber2": "SN00000001"
            }, {
                "id": "123456",
                "description": "Descripcion del producto",
                "snumber1": "7635438yGTA6785",
                "snumber2": "SN00000001"
            }]
        }
    }],
    "packages": [],
    "template": "components\/document\/basicDocument",
    "conf": {
        "margin_left": 2,
        "margin_right": 2,
        "margin_top": 1,
        "margin_bottom": 4
    }
}
```


Figura 3.4: Ejemplo de árbol de documento en JSON de un albarán de entrega.

3.1.2. Proceso de generado

La vista y los datos son combinados a través de una serie de fases dando lugar así al documento final.

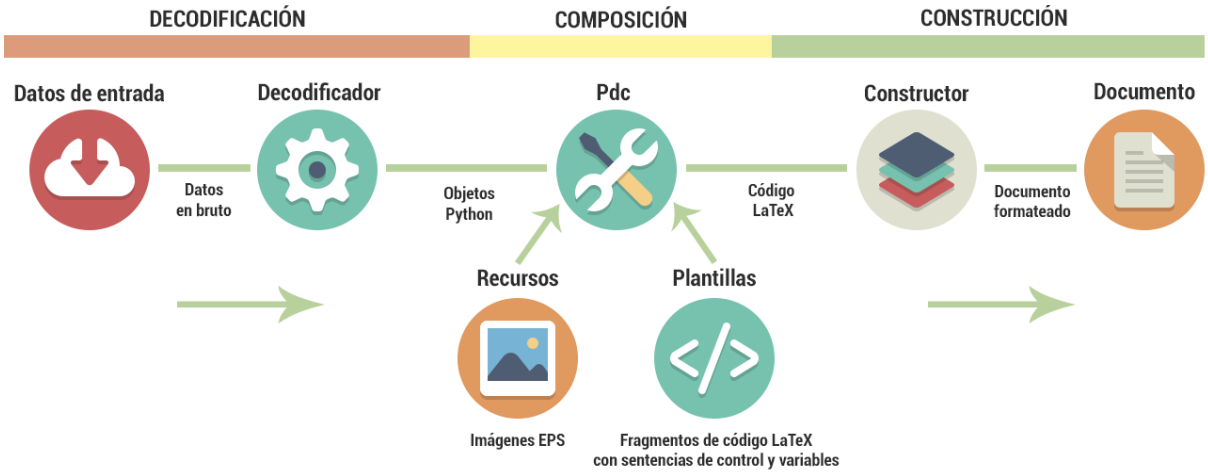


Figura 3.5: Esquema del proceso de generado de documentos.

Como puede verse en la figura 3.5, los datos pasan por tres fases: primero sufren una fase de decodificación que los transforma para poder ser utilizados, luego son utilizados para construir un código LaTeX intermedio y finalmente son transformados en el documento final. A continuación se detallan cada uno de los procesos que conforman estar tres fases.

Fase de decodificación

Durante esta fase, los datos que entran en la aplicación son tratados para ser transformados en estructuras con las que los siguientes procesos puedan trabajar. Esta fase es llevada a cabo por un decodificador, que transforma un formato de entrada concreto como XML o JSON a objetos python que se cargan en memoria para ser procesados más adelante.

Estos decodificadores son intercambiables, lo que permite modificar o ampliar la oferta de formatos soportados por la librería de una manera simple y rápida, además de proveer combinaciones de diferentes configuraciones asociando diferentes módulos, como se verá más adelante.

Fase de composición

En esta fase, a través del sistema de plantillas Jinja2 [34], se componen los diferentes fragmentos de código LaTeX sustituyendo en ellos los datos obtenidos de la fase anterior utilizando un algoritmo de recorrido en profundidad sobre el árbol de contenido, resolviendo todas las estructuras, sentencias e inclusiones que indiquen los datos y las plantillas.

El resultado final es un código LaTeX sin estructuras de control ni variables listo para ser compilado.

Fase de construcción

Para finalizar el proceso de generación, el código obtenido de la fase anterior es pasado a través de un programa externo, usualmente un compilador de LaTeX, que produce un documento en un formato concreto (ej. PDF) que es tomado como salida final del proceso de generado.

Estos constructores son también intercambiables, por lo que es fácil aumentar la variedad de formatos de salida para las soluciones.

3.1.3. Algoritmo de composición de código LaTeX a partir del árbol de contenido

Como ya se ha comentado en apartados anteriores, pdc genera el documento a partir de un árbol de contenido, que es una estructura en forma de árbol cuyos nodos representan los diferentes componentes del documento con su configuración asociada.

El árbol define qué plantillas utilizar, qué datos sustituir en ellas y cómo anidarlas unas dentro de otras. Para transformar un árbol en código LaTeX, hay que recorrerlo transformando cada uno de los nodos en código e ir uniéndo y anidándo dicho código de acuerdo a la estructura del árbol.

El algoritmo también debe ser capaz de ir recogiendo los paquetes LaTeX para ser introducidos en el documento base que actúa como nodo del árbol, para que sean escritos una única vez en el lugar adecuado en al cabecera del código.

La principal ventaja de estructurar el documento en forma de árbol, es que el algoritmo puede construirse de manera recursiva, realizando una búsqueda en profundidad que vaya transformando los diferentes nodos desde lo más profundo hasta la raíz.

El algoritmo es aplicado sobre el nodo raíz del árbol. En primer lugar se comprueba la integridad del nodo, comprobando que contenga todos los campos necesarios tal y como se ilustra en la figura 3.6.

```
def render(self, obj):
    rendered = ''

    # Check the integrity of the object
    if not check_integrity(obj):
        self.logger.error("Integrity check failed in object: \n\n{0}".format(
            json.dumps(obj, sort_keys=True,
                       indent=4, separators=(',', ': '))
        ))
        return rendered
```

Figura 3.6: Comienzo del algoritmo de composición. Comprobación de integridad.

Una vez comprobado que el nodo es apto para la transformación, se busca qué plantilla modeliza el nodo. Si no se encuentra la plantilla o hay algún tipo de error en su sintaxis se reporta el error, tal y como se observa en la figura 3.7.

```
# Search for loaded template
try:
    template = self._env.env.get_template(obj['template'] + '.pct')
except TemplateSyntaxError, e:
    self.logger.error("{0} in {1} at line {2}".format(e.message, e.name, e.lineno))
    return rendered
except TemplateNotFound, e:
    self.logger.error('Template "{0}" not found'.format(obj['template']))
    return rendered
```

Figura 3.7: Búsqueda de la plantilla que modeliza un nodo.

Como se representa en la figura 3.8, si el nodo tiene hijos, se transforman primero, acumulando el código tanto de contenido como de paquetes haciendo una llamada recursiva a la función.

```
# Render packages as normal content. Recursive packages are not allowed.
if 'packages' in obj and obj['packages']:
    for package in obj['packages']:
        # Check if the package have been already rendered
        if package not in self._rendered_package_list:
            content_data, package_data = self.render(package)
            rendered_packages += content_data
            self._rendered_package_list.append(package)
```

Figura 3.8: Detalle de la recursión del algoritmo de composición, si un nodo tiene hijos se transforman primero acumulando el resultado.

Una vez hecho esto, se transforman los paquetes que contiene el nodo que se está procesando, y se añaden a los que provienen de los nodos hijos, comprobando que no hay duplicidades. En la figura 3.9 puede observarse la llamada a la función render la la librería Jinja2.

Por último, se transforma el código del nodo que se está procesando, introduciendo en su interior el código de sus nodos hijos, y si es el nodo raíz también el código de los paquetes del resto de nodos como aparece en el código de la figura 3.10.

De esta forma, el algoritmo ejecutado sobre un ejemplo, como puede ser el de un examen simple, quedaría como se expone en las figura 3.11

```
# Render packages as normal content. Recursive packages are not allowed.
if 'packages' in obj and obj['packages']:
    for package in obj['packages']:
        # Check if the package have been already rendered
        if package not in self._rendered_package_list:
            content_data, package_data = self.render(package)
            rendered_packages += content_data
            self._rendered_package_list.append(package)
```

Figura 3.9: Transformación y acumulación del código de los paquetes.

```
# Render the template. Try to render accumulated package data too.
# Do not forget to resources path.
conf_data = []
if 'conf' in obj and obj['conf']:
    conf_data = obj['conf']
rendered = template.render(
    resources_path=self._env.work_path,
    content=rendered_content,
    conf=conf_data,
    packages=rendered_packages,
)

# Return rendered content and package data
return rendered, rendered_packages
```

Figura 3.10: Composición final del código LaTeX del nodo. Si es el nodo raiz, se sustituye en su interior el código acumulado de los paquetes y de todos los componentes del documento.

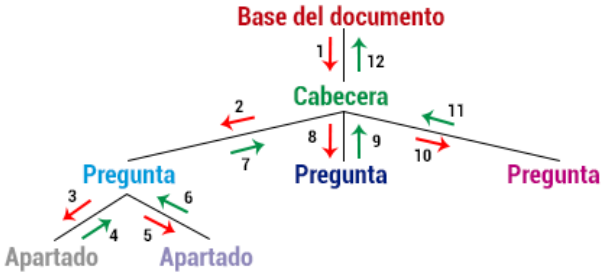


Figura 3.11: Ejemplo del flujo del algoritmo de composición sobre un ejemplo de documento.

Dado este flujo de ejecución, el documento iría componiendose a partir del árbol como se especifica en la tabla 3.1 (página 41). Cada uno de los componentes del documento se representa con su color correspondiente representado en la figura 3.11.

3.1.4. Implementación de pdc: mapa de clases

El proceso de generado mencionado anteriormente, es llevado a cabo por el trabajo conjunto de las diferentes clases que conforman la librería. Como puede verse en la figura 3.12 están divididas en tres grupos. A continuación se explicará cada uno de estos tres módulos que implementan las tres fases del proceso de generado explicadas en apartados anteriores, así como la aplicación pdc que es un programa de línea de comandos que utiliza estas clases para generar documentos desde la consola.

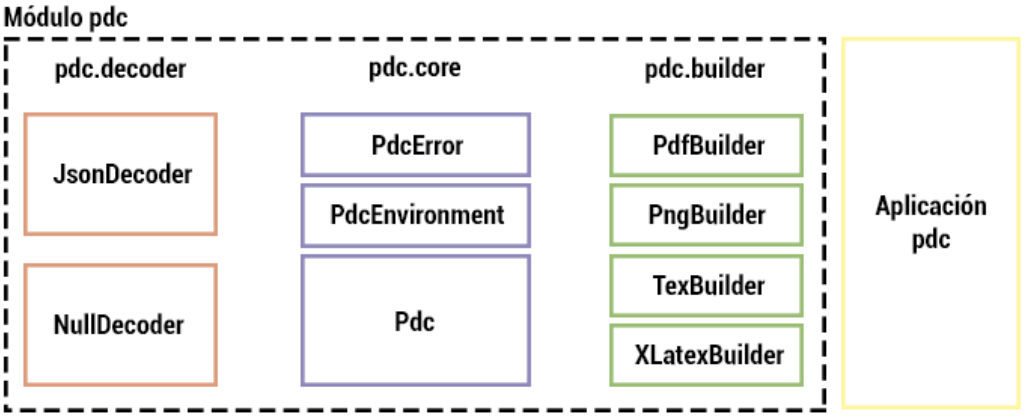


Figura 3.12: Mapa de clases de pdc. Se aprecian tres grupos de clases que se corresponden con las tres fases del proceso de generación.

Módulo pdc

El módulo principal contiene todas las clases necesarias para realizar la generación básica. A continuación se va a exponer la funcionalidad de los tres submódulos en los que se divide, que implementan las tres fases de generación de un documento: decodificación, composición y construcción.

- **Submódulo decoder**: En este módulo se encuentran todas las clases decodificadoras disponibles para la generación. En este punto del proyecto se han implementado 2 decodificadores:
  - **JsonDecoder**: Toma como entrada un fichero de texto plano con el árbol de contenido representado en JSON. La salida es ese mismo árbol de contenido codificado utilizando objetos y arrays de python.
  - **NullDecoder**: Toma como entrada un árbol de contenido codificado utilizando objetos y arrays de python. La salida del decodificador es la misma que la entrada. Este decodificador es necesario para poder utilizar pdc desde el servicio web, ya que como se verá más adelante, en la librería Django Rest Framework que usa pdcs los objetos se reciben ya decodificados.
- **Submódulo core**: Aquí se encuentran las 3 clases principales:
  - **PdcError**: Incluye las definiciones de los errores que la librería puede producir, define nombres y los asocia a códigos. Complementa al sistema de logging incluido en todas las clases del módulo.
  - **PdcEnvironment**: Mantiene un objeto de entorno de Jinja2 que utiliza para buscar en los directorios de las plantillas y cargarlas todas a memoria dejándolas listas para su uso.

También es la clase encargada de definir y controlar los directorios donde se encuentran las plantillas, los recursos, los decodificadores y los constructores. Tiene métodos para instalar cada uno de estos componentes de forma adecuada, gestionando los ficheros en disco y llamando a programas externos si fuera necesario para, por ejemplo, instalar un paquete de LaTeX desde el repositorio cuando se añade una nueva plantilla de paquete. La mayoría de estos métodos de gestión son necesarios para pdcs u otras aplicaciones de terceros, pero no se utilizan en la aplicación de consola.

También se encarga de instanciar en tiempo de ejecución los decodificadores y constructores por nombre de clase. Esto es necesario para poder escoger de forma dinámica qué configuración de generación utilizar en cada caso.

- **Pdc**: Esta clase, dado un determinado entorno, un codificador y un constructor, transforma una entrada en un documento. Se encarga de hacer pasar la información a través de todas las fases del proceso de generación. Implementa también el proceso de composición del código LaTeX intermedio a través del algoritmo sobre el árbol de contenido.

- **Submódulo builder**: En este módulo se encuentran todas las clases constructoras disponibles para la generación. En este punto del proyecto se han implementado 4 constructores:
  - **PdfBuilder**: Utiliza la aplicación pdflatex para transformar el código LaTeX intermedio en un documento en formato PDF.
  - **PngBuilder**: Utiliza la aplicación latex para generar un documento dvi y posteriormente dvipng para transformar el código LaTeX finalmente en una imagen en formato PNG.
  - **TexBuilder**: No realiza ninguna acción sobre el código. Es utilizado para generar el docuemnto en formato de código LaTeX.
  - **XLatexPdfBuilder**: Utiliza la aplicación *xelatex* para transformar el código LaTeX intermedio en un documento en formato PDF. Este proceso permite utilizar ciertos módulos de LaTeX como *pst-barcode* que requieren una transformación intermedia a dvi sin hacer varias llamadas al sistema [46]. Esta es más lenta que usando pdflatex.

Aplicación pdc

Es una aplicación en python que utiliza los componentes anteriormente citados para permitir la generación de documentos. Se basa en el módulo argparse [33] de python para recibir las opciones por línea de comandos.

Para que pdc pueda distribuirse e instalarse de forma sencilla en cualquier entorno, se ha programado un paquete wheel utilizando las herramientas python-setuptools y python-wheel. De esta manera, se puede instalar la librería a través del gestor de paquetes pip. El paquete puede también subirse en un futuro a PyPl y ser descargado entonces desde los repositorios oficiales de python [35].

Para más información técnica sobre cómo utilizar pdc y ejemplos de uso, ver la documentación online accesible mediante los datos presentes en el Anexo I.

3.2. Pdcs: Servicio REST como interfaz de pdc

Pdcs es una envoltura sobre la librería pdc, que permite hacer uso de la generación de documentos y administrar las plantillas, recursos y demás componentes de pdcs a través de la web. Los objetivos que persigue esta envoltura son:

- Aprovechar la librería de generación de documentos para proveer un servicio de generación remoto a través de la web, utilizando un servicio REST que pueda ser integrado sin dificultad independientemente del lenguaje y la condición de la aplicación cliente, y que cuente con la debida fiabilidad y las debidas condiciones de seguridad.
- Proveer de una interfaz de configuración para la librería, de forma que sea sencillo manejar las plantillas, los recursos y las configuraciones. Esta labor debe también poder realizarse desde la web, no solo por el administrador del sistema, sino por las propias aplicaciones que se integren.

El servicio consta de varios módulos, que funcionando conjuntamente, permiten alcanzar las características antes mencionadas. Como se puede observar en la figura 3.13, el servicio cuenta con dos herramientas diferenciadas: un panel de control web y un servicio REST que se corresponden con los dos principales objetivos de diseño.

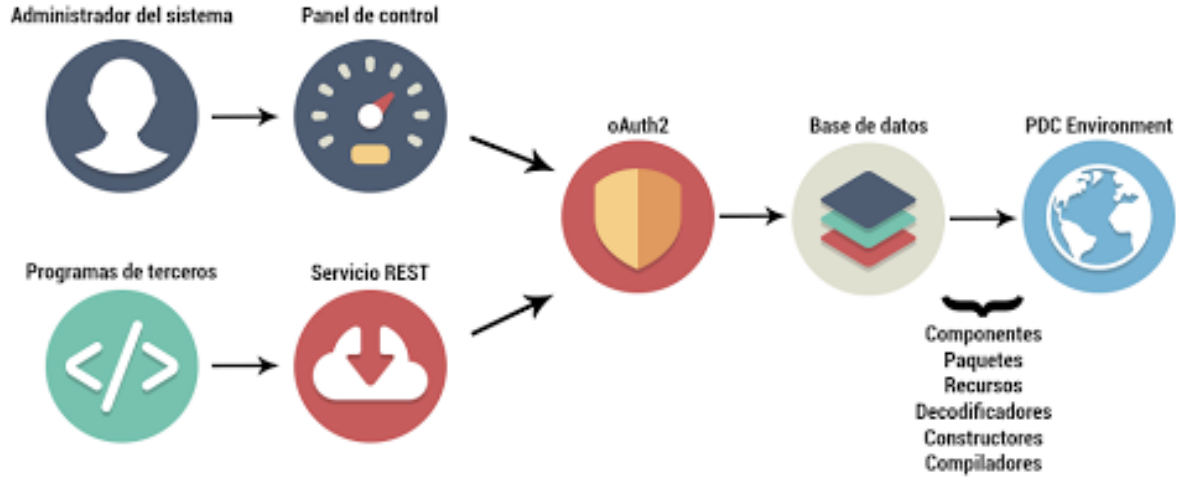


Figura 3.13: Esquema de las partes del servicio REST.

El panel de control web permite a los administradores del sistema configurar de forma sencilla y desde el navegador el entorno de generación de documentos, mientras que el servicio REST provee a aplicaciones de terceros un punto de acceso para realizar configuraciones y generar documentos.

A lo largo de este apartado, van a exponerse las tecnologías utilizadas para implementar el servicio y las razones de su elección así como el funcionamiento del servicio REST y del panel de control.

Tecnologías utilizadas

Tanto el servicio REST como el panel de control están construidas en Python y su framework web Django [6]. Esta decisión de diseño se tomó debido a varias razones:

- Por las ventajas antes comentadas, pdc está escrito en python, de forma que la evolución natural es escribir la envoltura también en python, para que la programación y la integración sean más cómodas.

- Django es un framework estable, escalable, probado en entornos profesionales y con una gran comunidad. Esto hace de Django un framework muy rápido de aprender y muy ágil a la hora de desarrollar. También es una tecnología con la que estoy familiarizado ya que trabajo a diario con ella.
- Django es software libre en continua evolución y revisión por parte de la comunidad. De esta manera se minimiza el riesgo de tener problemas de seguridad derivados de fallos en el framework. Django también tiene una gran variedad de librerías libres desarrolladas y mantenidas por la comunidad que favorecen a un desarrollo más rápido y sencillo.
- Python y Django son una buena elección para aplicaciones web debido a que son independientes de la plataforma. Por tanto, el servicio puede desplegarse en una gran variedad de sistemas con configuraciones distintas sin cambios de arquitectura.

En los siguientes puntos, se desarrollarán más en detalle cada una de las tecnologías utilizadas en las herramientas implementadas.

Servicio REST

El servicio de pdcs ofrece la funcionalidad necesaria para la generación de documentos a cualquier aplicación de terceros, permitiendo la comunicación entre estas aplicaciones y pdc utilizando llamadas HTTP [14] y el lenguaje de descripción de objetos JSON [15].

Tecnologías

Este API está construido sobre la librería Django REST Framework [7] que permite a partir de las entidades de base de datos (en django llamadas modelos) publicar un servicio REST basado en JSON gestionado por la propia librería.

De esta forma, todos los elementos de configuración de pdc tienen su representación en una base de datos, lo que hace más sencillo mantenerla y mostrarla a través del servicio.

Como gestor de base de datos se está utilizando MySQL, debido a su simplicidad de instalación e integración con Django, sin embargo, cualquier otro gestor de bases de datos relacionales es compatible con el sistema.

A través de dicho servicio se puede interactuar con estas entidades de base de datos a través de peticiones HTTP como GET, POST, PUT o DELETE [14], lo que lo hace altamente compatible con cualquier lenguaje de programación o arquitectura de cara a ser integrado en una aplicación de terceros.

La ventaja de este tipo de servicios REST es su simplicidad frente a otras soluciones para la publicación de servicios como SOAP. Al ser más simple y ajustado al problema, consume menos recursos y es más fácil de depurar, probar y desarrollar. También su uso necesita de una documentación que especifique todos los detalles del API, por lo que el servicio estará muy bién documentado.

El servicio cuenta también con una vista web que puede ser de utilidad de cara a probar e integrar el servicio. Como puede verse en la figura 3.14, incluye herramientas de inspección que permiten ver el JSON devuelto de una petición, incluso también presenta formularios de prueba para construir peticiones.

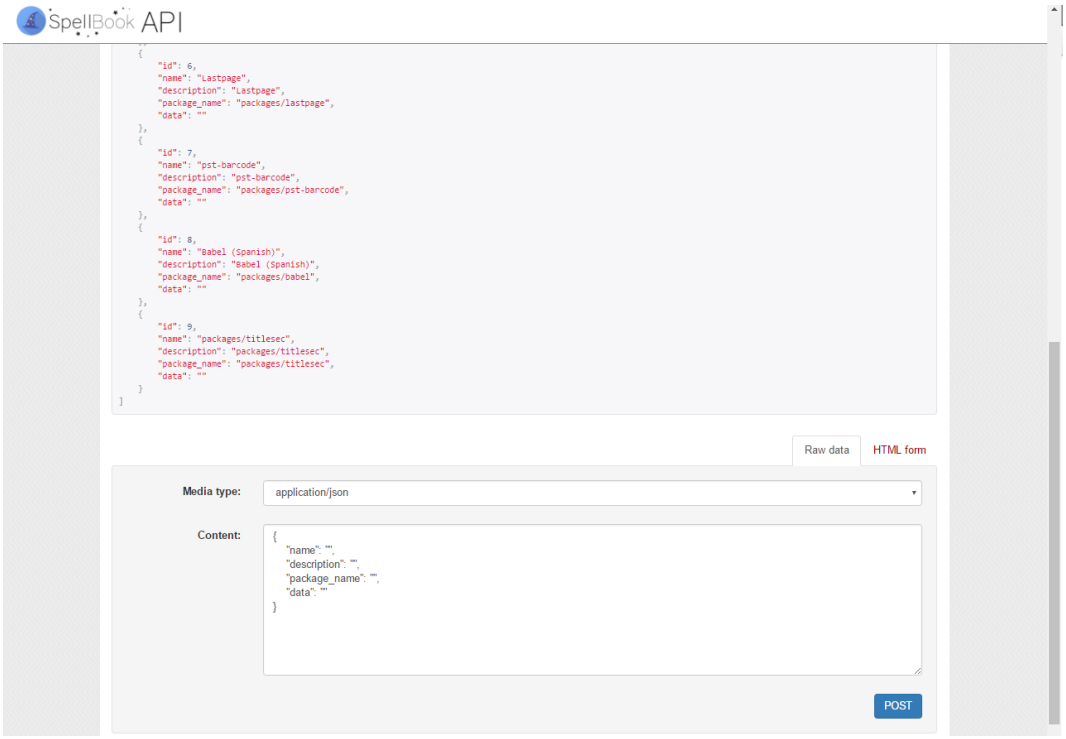


Figura 3.14: Vista web del servicio REST. Puede apreciarse el formulario de prueba en la parte inferior.

Transmisión de la información

El servicio no solo trata con información que puede expresarse en formato JSON, sino que también transporta otros tipos de datos, como imágenes, o documentos en PDF. Para transportar estas cargas especiales a través del API, se utiliza el esquema dataURI especificado en las normas IETF RFC 2397 [41]. En él, los datos son transmitidos en binario codificados en base64 y acompañados de un tipo de dato MIME [29].

De esta forma, los datos binarios son transmitidos de una forma estándar, que no solo es fácil de codificar y descodificar, sino que es empotrable de forma directa en una aplicación web en los navegadores modernos, lo que simplifica mucho, por ejemplo, la presentación de documentos PDF.

Sin embargo, los recursos codificados de acuerdo a esta especificación son aproximadamente un 33 % más grandes que sus equivalentes binarios [39], debido a la codificación por base64. También algunos navegadores limitan el tamaño de las URL haciendo que ficheros muy grandes no puedan ser mostrados utilizando este método y tengan que ser reprocesados.

Seguridad: Autorización y autenticación

Para proveer al servicio REST de seguridad, se ha decidido utilizar el servicio de autorización OAuth2 para complementar la autenticación de Django.

OAuth2 estructura la autorización por aplicaciones. Cuando una aplicación de terceros quiere conectarse a pdcs, el administrador del sistema debe dar de alta dicha aplicación y asignarle un usuario. Una vez hecho esto, se le asignará un identificador de cliente que es necesario para conectarse al servicio.

La aplicación deberá primero conectarse al servicio OAuth2 [16] y presentar su identificador de cliente que verificará la validez y le autorizará a utilizar sus credenciales para recibir un token de acceso, que luego se emplea para obtener los recursos.

Por simplicidad, el servicio de autorización OAuth2 se encuentra por defecto en el mismo host que el API, pero a través de la configuración interna de pdcs puede usarse un servidor externo como fuente de autorización.

De la misma manera, también se ha modificado la autenticación de Django para poder ser centralizada en una base de datos remota, permitiendo que varios servidores pdcs en máquinas diferentes compartan autenticación.

Seguridad: Ejecución maliciosa de código

Desde el servicio REST hay funcionalidades que no están disponibles, pero que sí lo están desde el panel de control. Desde el panel se puede editar el código python de los constructores y decodificadores. Esto es potencialmente peligroso, ya que un usuario malintencionado podría utilizar esta característica en su beneficio. Por eso sólo está disponible para los administradores y no programáticamente a través del servicio.

En cambio, el código LaTeX del interior de las plantillas, si es editable a través del servicio. Esto es porque en el caso del código LaTeX no existe este problema, ya que el código se ejecuta con un nivel de protección (opción write18 activada) que no permite la ejecución de programas externos.

Funcionalidad

El servicio REST está organizado en endpoints, que son URLs que representan cada una de las entidades con las que una aplicación puede interactuar a través de peticiones HTTP.

Todos los endpoints permiten las operaciones GET, POST, PUT, DELETE, OPTIONS y HEAD. A continuación se presenta en detalle cada uno de ellos:

- Paquetes (*/api/packages/*): A través de este endpoint Se pueden listar, editar, borrar y añadir paquetes LaTeX al sistema. Cuando un paquete es añadido, se utiliza el repositorio de LaTeX para añadir todos los ficheros necesarios a la instalación de texlive para que pueda ser utilizado por el compilador de LaTeX. El código de los paquetes puede ser transmitido en texto plano o codificado como dataURI.
- Componentes (*/api/components/*): De la misma manera que los paquetes, a través de este endpoint se puede interactuar con los diferentes componentes disponibles. El código LaTeX puede ser transmitido en texto plano o codificado como dataURI.
- Recursos (*/api/resources/*): Usando este endpoint se puede interactuar con las imágenes disponibles en el servidor para ser utilizadas durante la generación. Todos los intercambios de imágenes se hacen en formato PNG y a través del esquema dataURI.
- Compiladores (*/api/compilers/*): Este endpoint permite seleccionar un compilador para realizar la generación de un documento. Estos son las distintas configuraciones disponibles para la generación de documentos, compuestas por distintas combinaciones de decodificadores y constructores. Este endpoint solo provee acceso a los compiladores y permite usarlos para generar documentos, pero no permite cambiarlos, borrarlos o añadir más.

A efectos de funcionamiento interno, las peticiones HTTP lanzadas contra estos endpoints desencadenan tres tipos diferentes de operaciones:

- **Peticiones de obtención de configuraciones**: El objetivo de estas peticiones es obtener los elementos de configuración del sistema. Por ejemplo listar los componentes disponibles para ver si uno en concreto lo está, o ver qué configuraciones de generación hay pueden utilizarse.

  Estas peticiones son atendidas por la librería Django REST Framework, que directamente responde al cliente con la información almacenada en la base de datos. Al mantener actualizadas todas las configuraciones en ella, no es necesario interactuar directamente con los ficheros en disco, por lo que puede responderse más rápidamente. Un esquema de este funcionamiento puede verse en la figura 3.15.

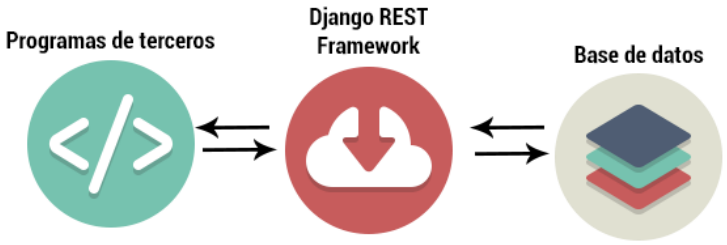


Figura 3.15: Esquema de una petición de obtención de configuraciones.

- **Peticiones de actualización de configuraciones**: Su objetivo es cambiar, borrar o añadir elementos de configuración al sistema, como por ejemplo añadir un nuevo componente, o cambiar el código de un constructor.

  Este tipo de peticiones son atendidas por la librería Django REST Framework quien automáticamente aplica los cambios necesarios sobre la base de datos. Cuando estos cambios se están realizando, una serie de señales llaman a los métodos del entorno de generación pdc para aplicar los cambios necesarios sobre el SO y el disco, tal y como se representa en la figura 3.16.

  En caso de que todas las operaciones concluyan con éxito, las información es introducida definitivamente en base de datos, en caso contrario, los cambios se descartan y se devuelve un error manteniendo así la coherencia entre la base de datos y el entorno de generación.

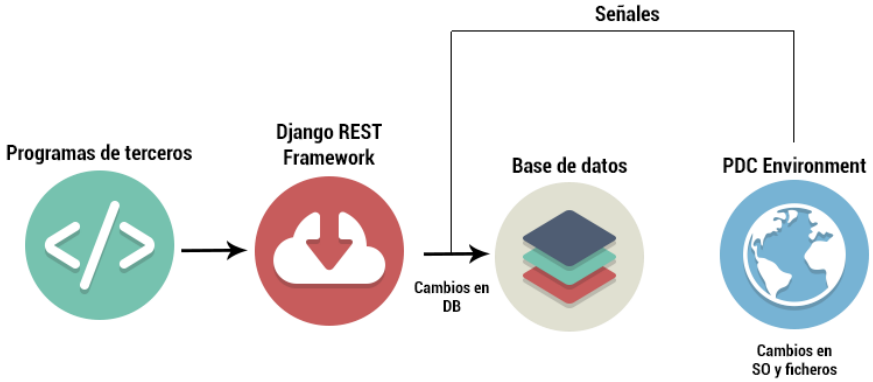


Figura 3.16: Esquema de una petición de actualización de configuraciones.

- **Peticiones de generación de documento**: Se realizan llamando a la función *compile* de un compilador concreto, aportándole el árbol de contenido en la petición. El sistema crea un objeto pdc a partir del entorno de generación y la configuración del compilador. Con él, se genera el documento y se envía de vuelta al cliente codificado en dataURI. Este proceso puede verse esquematizado en la figura 3.18.

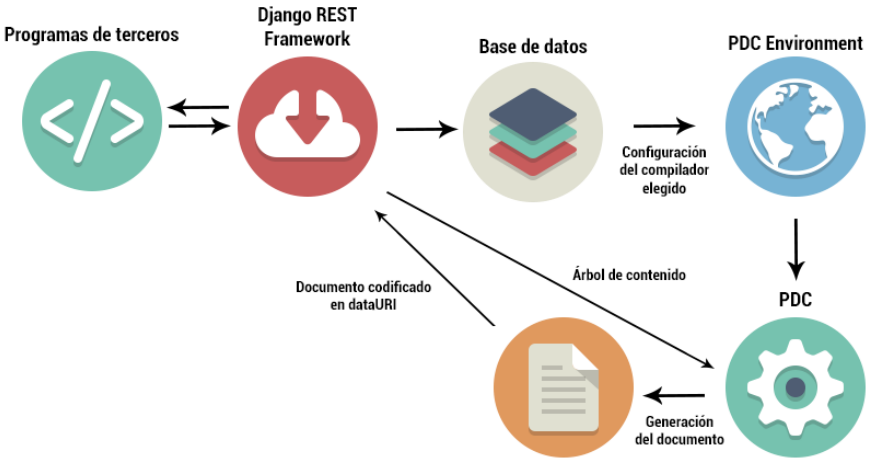


Figura 3.17: Esquema de una petición de generación de documento.

Panel de control

El panel de control es un portal web que permite la configuración del entorno de generación por parte del administrador del sistema.

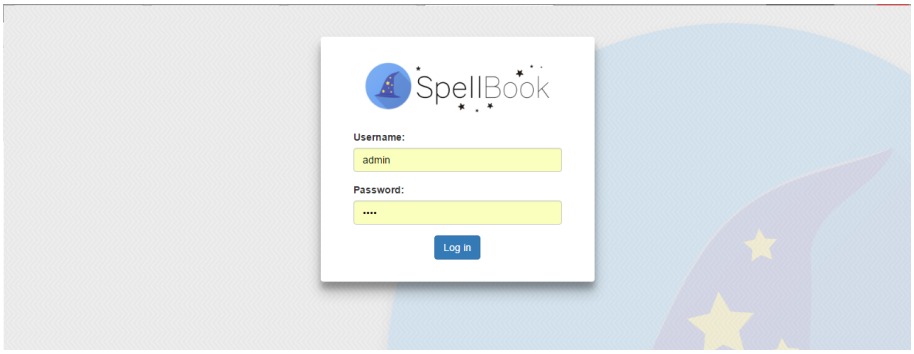


Figura 3.18: Pantalla de login del panel de control web.

Desde este entorno, el administrador es capaz de llevar a cabo las siguientes configuraciones:

- **Manejo de usuarios**: El administrador puede crear nuevos usuarios, tanto para acceder al panel de control como para hacer uso del API. Un ejemplo de la interfaz puede verse en la figura 3.19.

  Se pueden manejar también permisos y grupos, pudiendo controlar así a qué configuraciones tiene acceso cada uno de los usuarios o grupos del sistema a través de una serie de tablas (figura 3.20).

Figura 3.19: Captura de pantalla de la página de gestión de usuarios del panel de control.

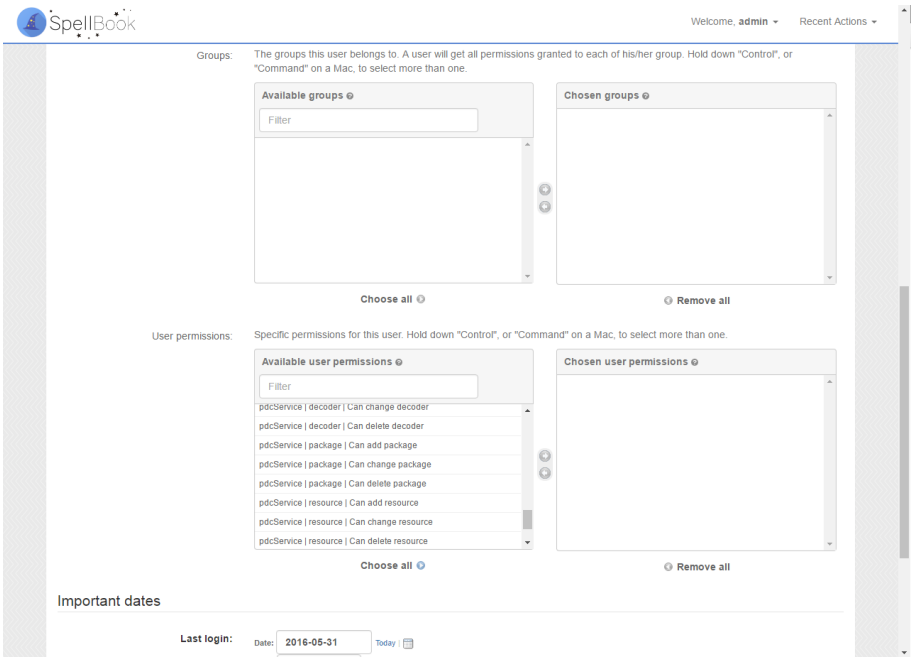


Figura 3.20: Captura de pantalla de la página de gestión de permisos de los usuarios en el panel de control.

- **Manejo de aplicaciones conectadas**: Dado que todas las conexiones al API pasan por oAuth2, es necesario que el administrador pueda crear entradas de acceso para nuevas aplicaciones que quieran conectarse al servicio, además de poder manejar sus claves de acceso. Para ello dispone de una interfaz como la que se observa en la figura 3.21
- **Manejo de plantillas, decodificadores y constructores**: El administrador puede desde un editor de LaTeX integrado en el navegador editar o crear componentes o paquetes para luego ser utilizados para generar documentos. De la misma manera pueden manejarse los decodificadores y los constructores disponibles en el servicio a través de un editor integrado de python (figura 3.22).
- **Manejo de recursos**: El administrador puede subir nuevas imágenes al sistema para ser utilizadas más adelante dentro de los documentos (figura 3.23).
- **Manejo de compiladores**: El administrador del sistema puede combinar en una interfaz similar a las anteriores los decodificadores y constructores presentes en el sistema para formar compiladores que aplicaciones de terceros puedan utilizar.

Este portal web está construido sobre django-admin-tools [5], que es un módulo de Django que provee de una interfaz web simple (figura 3.24) para manejar entidades de base de datos de una manera sencilla.

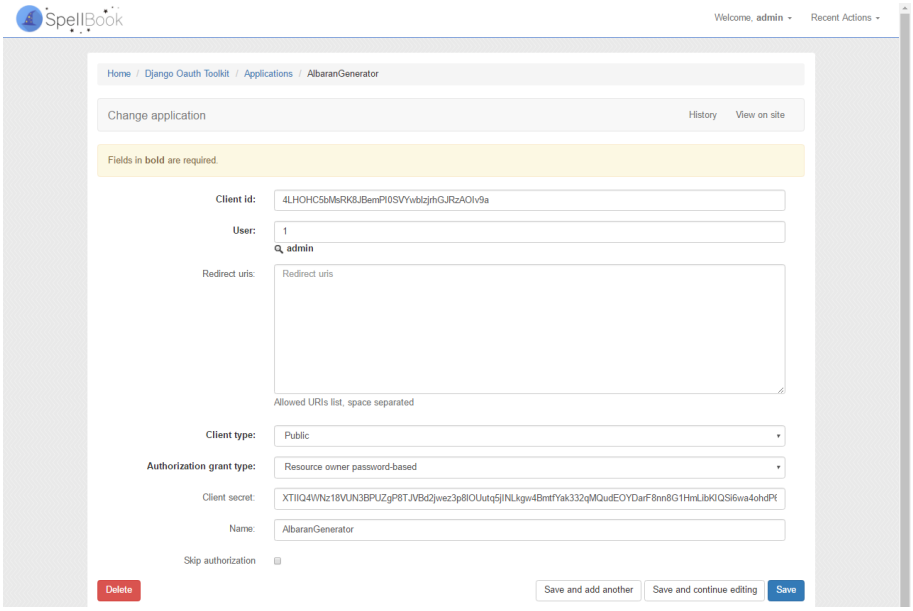


Figura 3.21: Captura de pantalla de la página de gestión de permisos de los usuarios en el panel de control.

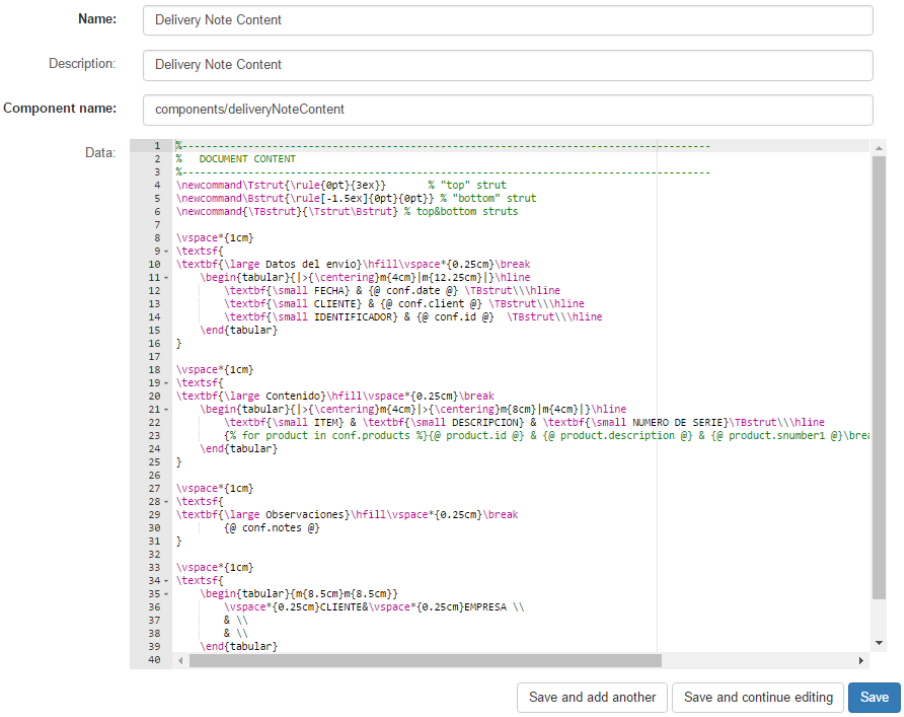


Figura 3.22: Captura de pantalla de la página de gestión de componentes, donde se puede ver el editor de LaTeX.

Para modificar el aspecto del portal por defecto de django-admin-tools, se ha utilizado la librería django-admin-bootstrapped [4]. Este paquete provee de una versión del portal de administración de django construida sobre la librerías javascript Bootstrap3 y jQuery. De esta forma, se dota a la interfaz de un aspecto más moderno (figura 3.25) y más funcionalidad javascript.

Figura 3.23: Captura de pantalla de la página de gestión de recursos.

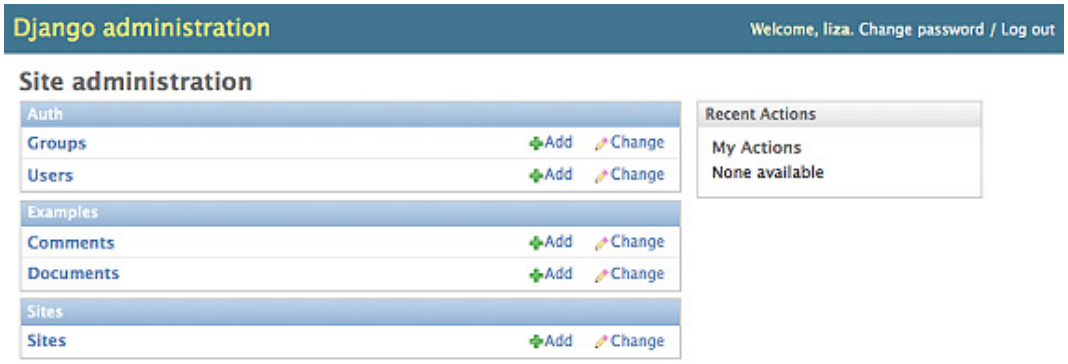


Figura 3.24: Aspecto del portal de administración por defecto de Django.

Sobre esta versión modificada se construyeron nuevas plantillas base y estilos CSS para modificar el aspecto y la funcionalidad del portal. Se han escogido unos colores claros basados en el logo de la plataforma y se han simplificado controles y pantallas para dar un look-and-feel más sencillo y hacer la aplicación más fácil de utilizar. El resultado final es el mostrado en la figura 3.26

Por último, se ha utilizado la librería javascript ace [1] para embeber los diferentes editores de código en el panel de control. Esta librería no solo provee de un editor, sino también de módulos para el resalte de sintaxis de acuerdo al lenguaje de programación, diferentes estilos visuales, corrección de errores sintácticos y otras características.

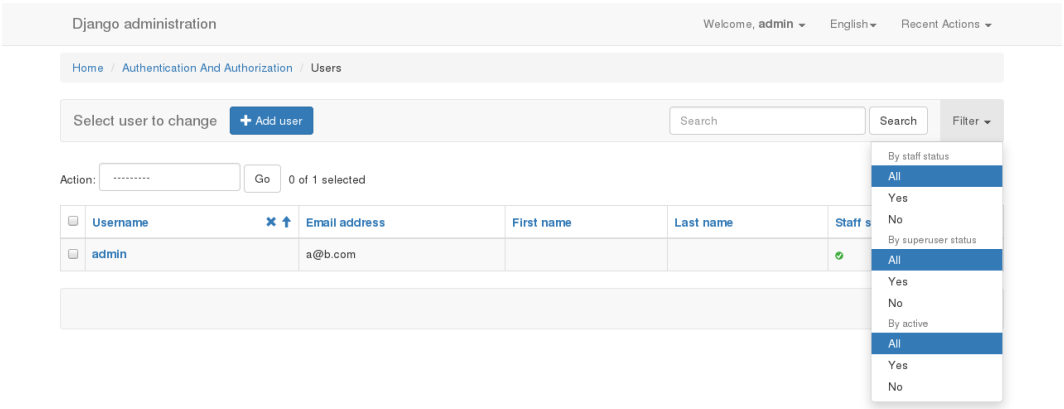


Figura 3.25: Aspecto del portal de administración por utilizando django-admin-bootstrapped.

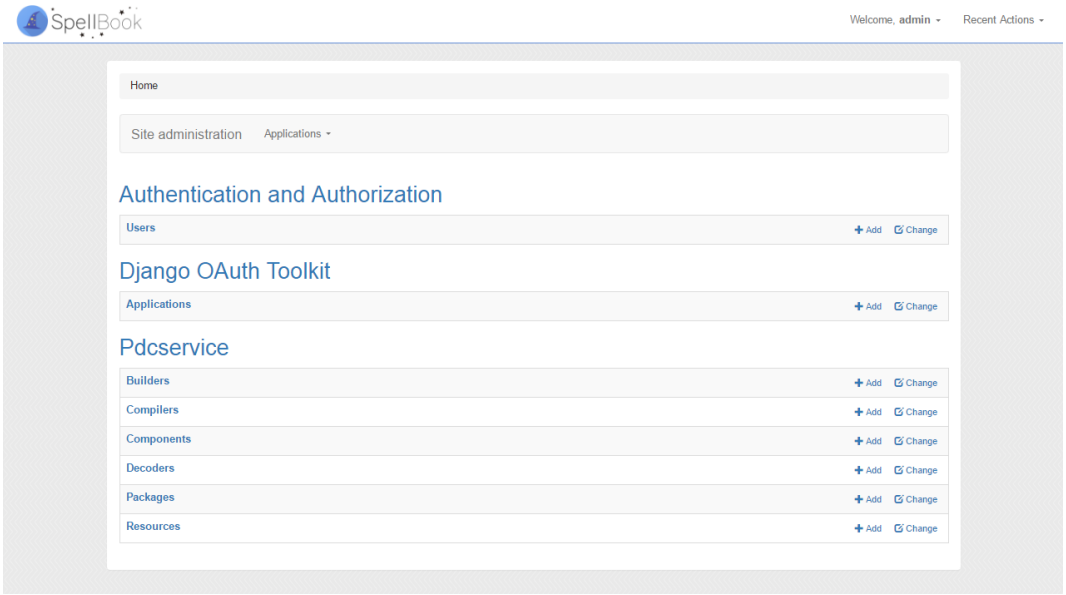


Figura 3.26: Aspecto del portal de administración por utilizando django-admin-bootstrapped y aplicando modificaciones.

3.3. sPHPellbook: Librería cliente para el servicio REST

SPHPellbook es una librería escrita en PHP que implementa todas las funciones del servicio REST pdcs. Simplifica la autenticación oAuth2 y provee al programador de herramientas para representar programáticamente cada los elementos de pdcs.

La librería está dividida en cinco módulos:

- **Spellbook.class.php**: En este módulo se encuentran las clases principales de la librería. La clase Spellbook actúa como objeto principal en las conexiones, se encarga de mantener el estado, guardar las credenciales y gestionar la autenticación en cada una de las peticiones. También se encuentra en este módulo la clase SpellbookError para gestionar los diferentes errores que puedan aparecer. Se basa en la librería cURL de PHP [3].

- **tools.php**: Es un módulo donde se encuentran funciones auxiliares para el resto de módulos. Entre ellas destacan las de transformación de información en forma de arrays asociativos de PHP a objetos StdClass para ser transformadas correctamente a JSON.
- **Component.class.php**: Es el módulo que contiene a la clase Component que representa cada uno de los componentes presentes en el sistema. Esta clase actúa de interfaz entre el programador y el servidor, permitiendo hacer cambios en el objeto de un componente y transmitirlos al servidor.
- **Package.class.php**: De la misma manera que el módulo anterior, este fichero contiene la clase Package que actúa de interfaz para los paquetes disponibles en el servidor.
- **Resource.class.php**: Al igual que los paquetes anteriores, este módulo contiene la clase Resource que hace de interfaz entre el programador y los recursos que puede usar en la generación de documentos.

El núcleo central de este código es la librería cURL de PHP [3], que permite construir y enviar peticiones HTTP con un alto nivel de personalización y gestión de cabeceras y errores.

Aunque esta librería está disponible para muchos lenguajes, finalmente se escogió PHP porque se quería probar la relación del sistema servidor escrito en python con otro lenguaje diferente. Este lenguaje debía implementar cURL o similar y además debía ser web, puesto que Spellbook está diseñado para dar servicio a clientes de este tipo. Todas estas características junto con el hecho de la experiencia previa con PHP, su estabilidad y la gran comunidad de programadores en Internet lo hacía un candidato perfecto para implementar un cliente del servicio REST de pdcs.

3.3.1. Funcionamiento

Para comenzar a utilizar la librería, el primer paso es conectarse al servidor utilizando el objeto principal. Al realizar la conexión se verifican las credenciales y se obtiene un token de acceso.

Cuando el objeto Spellbook es creado, es necesario pasarle las credenciales de usuario y los datos de conexión al servidor. Con estos datos, y tras llamar al método init, el cliente se conecta con el servidor de autorización oAuth2 (por defecto en el propio servidor pdcs) y obtiene un token de acceso. Dicho token es almacenado junto con el token de refresco que es utilizado para obtener otro token de acceso cuando el primero caduque. Estas operaciones pueden verse en el código de la figura 3.27

Una vez obtenido el token, a través del objeto principal pueden obtenerse por nombre o ID cada uno de los componentes paquetes o recursos disponibles en el sistema. Llegado este punto, se puede crear un árbol de contenido utilizando estos objetos interfaz y enviarlo a un compilador para generar un documento.

Para obtener los objetos, se hace una petición al servidor como a aparece en la figura 3.28 solicitando el objeto por nombre. Cuando el servidor responde, los objetos interfaz son creados y rellenados con la información recibida de manera automática. Después el usuario puede introducir la configuración que desee sobre esos objetos.

Como también se observa en la figura 3.28 el programador puede ir anidando objetos unos dentro de otros hasta ir formando el árbol de contenido. Cuando el árbol está listo, se manda al servidor indicando que compilador debe usarse para generar el documento (3.29).

```php
require_once('spellbook/spellbook.class.php');

// Get the data and decode it
$data = $_GET['data'];
$jsonData = base64_decode($data);
$oData = json_decode($jsonData);

$host = 'http://localhost:8001/api';
$username = 'admin';
$password = '1234';
#$clientId = 'gBNxiTnrpWxr46YNTaz4eZqqH4Ch1ttzIPPHlkbH';
$clientId = "qk09v1SUZJJ0LoJsVTYIK8T9CK6hsoN3wYE7gm1L";

$sb = new Spellbook($host, $username, $password, $clientId);
$r = $sb->init();
if ($r !== SpellBookError::Ok){
    print 'An error has ocurred while initialization: ' . SpellBookError::getDescription($r);
    die();
}
```

Figura 3.27: Fragmento de código correspondiente a la conexión de sPHPellnook al servidor pdcs.

```php
# Prepare document base
$basicDocument = $sb->getComponentByName('components/document/basicDocument');
if ($basicDocument === SpellBookError::ObjectNotFound){
    print 'Error instantiating basicDocument';
    die();
}

# Add data
$basicDocument->conf['margin_left'] = 2.0;
$basicDocument->conf['margin_right'] = 2.0;
$basicDocument->conf['margin_top'] = 1.0;
$basicDocument->conf['margin_bottom'] = 4.0;

# Add contained components
array_push($basicDocument->content, $basicHeader);
array_push($basicDocument->content, $deliveryNoteHeader);
array_push($basicDocument->content, $deliveryNoteContent);
```

Figura 3.28: Fragmento de código correspondiente a la obtención de los objetos desde el servidor y a la introducción de la configuración en ellos.

```php
$response = $sb->compile($basicDocument, 4);
```

Figura 3.29: Fragmento de código correspondiente al envío de un árbol de contenido al servidor para generar el documento utilizando el compilador 4.

La respuesta de la petición de generación es recibida, y se devuelve al programador el dataURI que contiene el documento para que sea mostrado al usuario. únicamente es necesario extraer la información de la respuesta tal y como aparece en la figura 3.30.

```php
$decodedResponse = json_decode($response, true);
if(isset($decodedResponse['error'])){
    print $decodedResponse['error'];
}
$documentDataURI = $decodedResponse['data'];
```

Figura 3.30: Fragmento de código correspondiente al procesado de una respuesta de generación de documento.

Otra funcionalidad que también implementa sPHPellbook es la de la ceación, modificación y borrado de los componentes, paquetes y recursos, de la misma forma que pueden realizarse estas mismas tareas de configuración desde el panel de control. Sin embargo, sPHPellbook no implementa el listado y la modificación y el manejo de los constructores, decodificadores y compiladores debido a que no están disponibles a través del servicio REST por motivos de seguridad. Estos elementos solo pueden ser configurados por los administradores a través del panel de control.

3.4. Documentación web

Para poder proveer a los programadores y administradores de documentación de cómo utilizar Spellbook, se ha construido un sitio web a modo de guía de uso. Contiene una pequeña introducción al sistema, y posteriormente explica a través de un ejemplo, la configuración y el uso básicos de pdc, pdcs y sPHPellbook.

El sitio web está generado utilizando la librería mkdocs [28]. Esta librería utiliza un estilo y unas plantillas que dictan el aspecto de la documentación final (figura 3.31), que se rellena con el contenido en formato markdown.

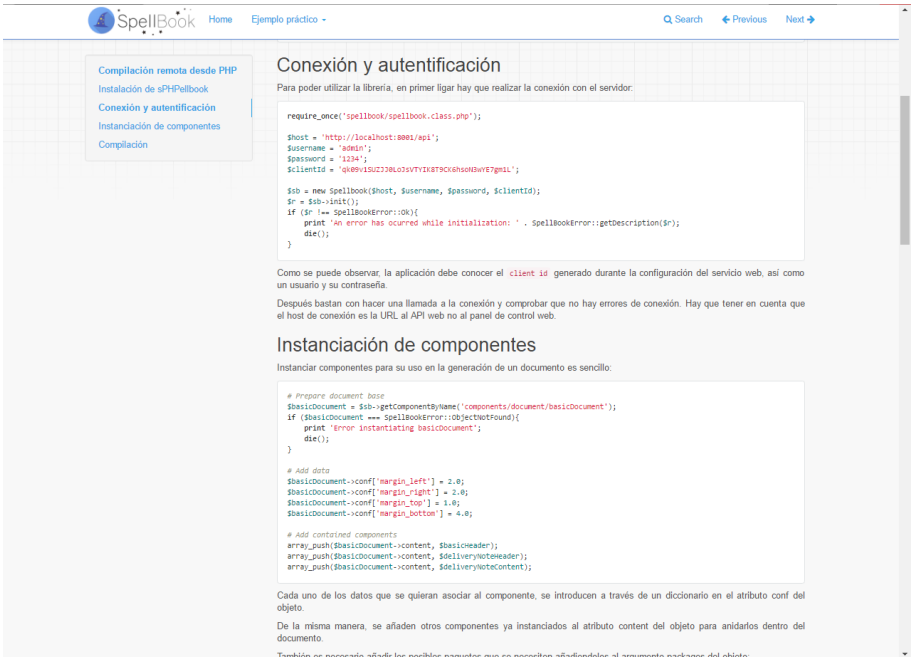


Figura 3.31: Aspecto de la documentación web accedida desde el navegador.

Esta librería permite realizar documentación de una manera ágil, ya que el formato markdown es muy sencillo de aplicar y aprender. También permite poner fragmentos de código, e incluye un resaltadores de sintaxis que soportan multitud de lenguajes diferentes.

Incluye también un generador de HTML que permite exportar la documentación y ser desplegada en Apache, NGINX o similar. Para realizar las pruebas, permite desplegar de manera sencilla un servidor local basado en Tornado que se actualiza automáticamente a cada cambio de los textos.

Para realizar esta documentación, en primer lugar se redactaron los textos en formato markdown, incluyendo imágenes y fragmentos de código. Posteriormente se modificó uno de los temas básicos que se suministran con mkdocs para aplicarle el estilo de la aplicación.

3.5. Environment: Script de instalación y estructura básica

Spellbook es un software complicado, constituido por multitud de piezas y aplicaciones diferentes que deben comunicarse y coordinarse adecuadamente. Por todo ello, desplegar un servidor pdcs no es un trabajo fácil, ya que hay que instalar y configurar muchos elementos, como el servidor web, la conexión de base de datos o los paquetes de LaTeX. Un esquema de las dependencias y relaciones de estos elementos puede verse en la figura 3.32.

Figura 3.32: Esquema de relaciones entre los distintos programas que interactúan en un sistema Spellbook.

Tal y como se comentaba en apartados anteriores, hoy en día es un requisito muy importante poder desplegar y replegar servicios muy rápidamente para adaptarse a la demanda y ahorrar esfuerzos y costes.

Por todo ello, se decidió usar la plataforma Vagrant [12] para crear una máquina virtual base o box que permitiera desplegar un servidor pdcs completo prácticamente sin configuración previa. Además vagrant no solo permite gestionar de forma eficaz las máquinas virtuales, sino desplegarlas en multitud de entornos diferentes, provisionales a partir de scripts u otras herramientas y publicarlas en un repositorio accesible a la comunidad.

Sin embargo, actualmente hay muchas alternativas para desplegar este tipo de aplicaciones, tanto en máquinas físicas como en cloud. Por lo que finalmente se decidió crear un script de instalación más independiente del método de desplegado que fuese capaz de configurar un servidor pdcs básico completo.

Aunque el script está optimizado para provisionar máquinas vagrant, puede usarse también, con unos cambios mínimos, para, por ejemplo, provisionar contenedores docker o para instalar y configurar el servidor en cualquier máquina física o virtual bajo Ubuntu o Debian.

El script se vale de herramientas estándar como apt-get y pip para descargar e instalar los paquetes necesarios. También sincroniza la última versión de cada uno de los componentes de Spellbook haciendo uso de repositorios git.

A continuación se expondrá más en detalle que función lleva a cabo cada uno de estos elementos.

3.5.1. Servidor web

Cuando llega una petición HTTP, es atendida en primer lugar por el servidor web NGINX [30]. Se decidió usar este servidor debido a su ligereza frente a Apache, ya que Spellbook está pensado para funcionar en máquinas más o menos pequeñas.

Sin embargo, NGINX no puede servir una aplicación Django por sí solo, de forma que se necesita un servidor WSGI. El elegido es Gunicorn [10], debido a su amplia compatibilidad, en especial con NGINX y en su buena documentación.

De esta manera, NGINX hace las veces de proxy inverso sirviendo las peticiones que llegan a Gunicorn, sobre el que está funcionando todo el servicio REST de Spellbook. La comunicación entre los dos es a través de un socket UNIX creado para este propósito.

NGINX también actúa de servidor web para los datos estáticos tanto del API, como del panel de control de pdcs, para que no tengan que ser servidos utilizando Gunicorn y mejorar así el rendimiento, ya que NGINX está también optimizado para cachear este tipo de datos.

3.5.2. Mantenimiento de datos y generación de documentos

Para mantener los datos del servicio, pdcs se comunica con una base de datos MySQL que contiene los datos de autentificación, autorización y configuración de todo el servicio.

Para la generación de documentos, como se ha comentado en apartados anteriores, pdc crea procesos que corren los compiladores de LaTeX y recogen su salida para enviarla al cliente.

3.5.3. Sistema operativo

El sistema operativo es el Ubuntu server 14.04 LTS. Se ha elegido debido a que es un S.O que está disponible en prácticamente todos los sistemas de computación en la nube, es fácil de instalar en máquinas físicas y es perfectamente compatible con vagrant y docker. Todo ello de cara a que el script de instalación funcione en el mayor número de plataformas diferentes.

También Ubuntu permite a partir de sus repositorios de paquetes instalar sin problemas y de manera automatizada todo el software necesario para que el servidor funcione.

Para poder garantizar el correcto funcionamiento de todas las piezas que conforman el servidor, se utiliza la librería supervisor [38] de python que controla que todos los procesos estén funcionando correctamente, los reinicia en caso de fallo y mantiene un log con toda la información necesaria para diagnosticar el problema.

3.6. Aplicación de prueba

Como demostración de todo el trabajo realizado, se ha desarrollado una pequeña aplicación PHP que hace uso de sPHPellbook para conectarse a una máquina virtual Spellbook desplegada con Vagrant.

Tal y como se aprecia en la figura 3.33 la aplicación consta de una página HTML que contiene un formulario donde el usuario introduce los datos del albarán que desea generar.

La tabla en la parte inferior del formulario está construida haciendo uso de la librería slickGrid [25] que provee de un motor para crear tablas dinámicas y editables en javascript.

Cuando el usuario pulsa el botón generar, la información es enviada a una página PHP, codificada en JSON y posteriormente en base64, para que no haya problemas de codificación de caracteres durante la petición POST.

La página PHP se conecta al servidor utilizando las credenciales y el ID de cliente configuradas previamente empleando el panel de control de pdcs. La conexión se realiza como se muestra en la figura 3.34. También decodifica la información del formulario que viene de la página anterior.

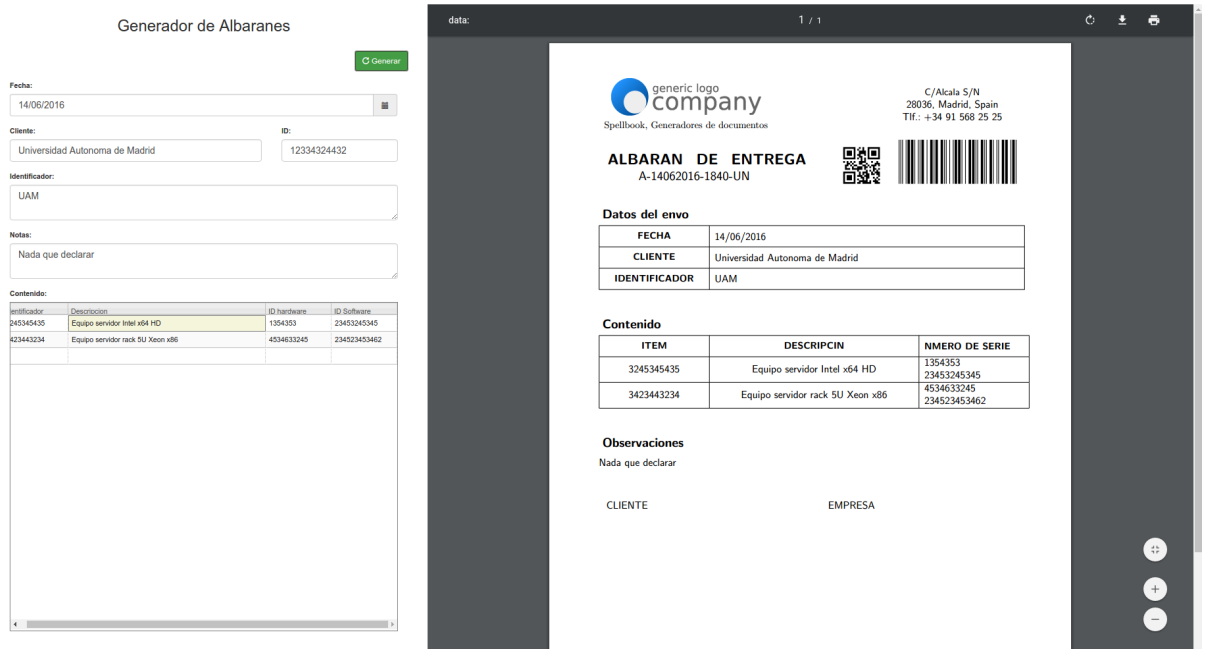


Figura 3.33: Página principal del generador de albaranes.

```php
require_once('spellbook/spellbook.class.php');

// Get the data and decode it
$data = $_GET['data'];
$jsonData = base64_decode($data);
$oData = json_decode($jsonData);

$host = 'http://localhost:8001/api';
$username = 'admin';
$password = '1234';
#$clientId = 'gBNxiTnrpWxr46YNTaz4eZqqH4Ch1ttzIPPHlkbH';
$clientId = "qk09v1SUZJJ0LoJsVTYIK8T9CK6hsoN3wYE7gm1L";

$sb = new Spellbook($host, $username, $password, $clientId);
$r = $sb->init();
if ($r !== SpellBookError::Ok){
    print 'An error has ocurred while initialization: ' . SpellBookError::getDescription($r);
    die();
}
```

Figura 3.34: Detalle del código PHP donde se realiza la conexión con el servidor.

Una vez hecho esto, instancia cada uno de los componentes del documento, previamente configuradas también haciendo uso del panel de configuración de pdcs, utilizando las entidades que provee el API PHP, y los carga con la información que proviene del formulario tal y como puede verse en la figura 3.35.

Por último, se hace la llamada al servidor para recibir el documento generado en formato dataURI, y se muestra dentro de un iframe en la página del formulario utilizando javascript (figura 3.36).

En el anexo II de este documento, se presenta un ejemplo completo del código LaTeX de cada uno de los componentes y paquetes del ejemplo, así como el árbol de contenido completo en JSON y el PDF resultante.

```php
# Prepare document
$basicDocument = $sb->getComponentByName('components/document/basicDocument');
if ($basicDocument === SpellBookError::ObjectNotFound){
    print 'Error instantiating basicDocument';
    die();
}
$basicDocument->conf['margin_left'] = 2.0;
$basicDocument->conf['margin_right'] = 2.0;
$basicDocument->conf['margin_top'] = 1.0;
$basicDocument->conf['margin_bottom'] = 4.0;

# Prepare Company header
$basicHeader = $sb->getComponentByName('components/header/companyHeader');
if ($basicDocument === SpellBookError::ObjectNotFound){
    print 'Error instantiating companyHeader';
    die();
}
$basicHeader->conf['image'] = 'logo';
$basicHeader->conf['image_width'] = 6.0;
$basicHeader->conf['image_height'] = 2.0;
$basicHeader->conf['institution_title'] = "Spellbook, Generadores de documentos";
$basicHeader->conf['company_address_1'] = "C/Alcala S/N";
$basicHeader->conf['company_address_2'] = "28036, Madrid, Spain";
$basicHeader->conf['company_address_3'] = "Tlf.: +34 91 568 25 25";

$packages = [
    $sb->getPackageByName('packages/pst-barcode'),
    $sb->getPackageByName('packages/fancyhdr'),
    $sb->getPackageByName('packages/babel'),
    $sb->getPackageByName('packages/tabularx')
];
$basicHeader->packages = array_merge($basicHeader->packages, $packages);
```

```php
# Prepare note header
$deliveryNoteHeader = $sb->getComponentByName('components/deliveryNoteHeader');
if ($basicDocument === SpellBookError::ObjectNotFound){
    print 'Error instantiating deliveryNoteHeader';
    die();
}

# Generate the not code
$code = "A";
$code .= '-' . str_replace('/', '', $oData->date) . '-' . date('Hi');
$code .= '-' . strtoupper(substr($oData->client, 0, 2));

$deliveryNoteHeader->conf['title'] = 'ALBARAN      DE   ENTREGA';
$deliveryNoteHeader->conf['code'] = $code;
$deliveryNoteHeader->conf['qr_content'] = $code;
$deliveryNoteHeader->conf['bar_content'] = str_replace('/', '', $oData->date) . date('Hi');

# Prepare note content
$deliveryNoteContent = $sb->getComponentByName('components/deliveryNoteContent');
if ($basicDocument === SpellBookError::ObjectNotFound){
    print 'Error instantiating deliveryNoteContent';
    die();
}
$deliveryNoteContent->conf['date'] = $oData->date;
$deliveryNoteContent->conf['client'] = $oData->client;
$deliveryNoteContent->conf['id'] = $oData->id;
$deliveryNoteContent->conf['notes'] = $oData->notes;
$deliveryNoteContent->conf['products'] = array ();

foreach ($oData->content as $product){
    $productArray = array('id' => $product->id, 'description' => $product->description,
        'snumber1' => $product->snumber1, 'snumber2' => $product->snumber2);
    array_push($deliveryNoteContent->conf['products'], $productArray);
}

# Introduce components inside the document
# basicdocument
#  |- basicHeader
#  |- deliveryNoteHeader
#  |- deliveryNoteContent
array_push($basicDocument->content, $basicHeader);
array_push($basicDocument->content, $deliveryNoteHeader);
array_push($basicDocument->content, $deliveryNoteContent);
```

Figura 3.35: Detalle del código PHP donde se obtienen los componentes del documento, se rellenan con los datos y se construye el árbol de contenido.

```php
# Compile using PDF compiler
$response = $sb->compile($basicDocument, 4);
$decodedResponse = json_decode($response, true);
if(isset($decodedResponse['error'])){
        print $decodedResponse['error'];
        die();
}
$documentDataURI = $decodedResponse['data'];

print (sprintf("<script>window.location='%s'</script>", $documentDataURI));
```

Figura 3.36: Detalle del código PHP donde se envía a generar el árbol de contenido y se muestra el PDF resultante.

Cuadro 3.1: Ejemplo del flujo del algoritmo de composición sobre el árbol de la figura 3.11 en la página 21.

	Acciones	Documento
1	Se compone el contenido primero, llamada recursiva.	
2	Se compone el contenido primero, llamada recursiva.	
3	Se compone el contenido primero, llamada recursiva.	
4	El nodo no tiene más hijos, se componen los paquetes y el nodo posteriormente se retorna el resultado.	
5	El nodo tiene más hijos, se compone el siguiente, llamada recursiva.	
6	El nodo no tiene más hijos, se componen los paquetes y el nodo posteriormente se retorna el resultado.	
7	El nodo no tiene más hijos, se componen los paquetes y el nodo, introduciendo en él sus hijos compuestos y se retorna el resultado.	
8	El nodo tiene más hijos, se compone el siguiente, llamada recursiva.	
9	El nodo no tiene más hijos, se componen los paquetes y el nodo posteriormente se retorna el resultado.	
10	El nodo tiene más hijos, se compone el siguiente, llamada recursiva.	
11	El nodo no tiene más hijos, se componen los paquetes y el nodo posteriormente se retorna el resultado.	
12	El nodo no tiene más hijos, se componen los paquetes y el nodo, introduciendo en él sus hijos compuestos y se retorna el resultado.	
13	El nodo no tiene más hijos, se componen los paquetes y el nodo, introduciendo en él sus hijos compuestos y se retorna el resultado. Como es el nodo raíz, también se introducen todos los paquetes acumulados del compuesto de todos los nodos del árbol eliminando duplicidades.	

4

Conclusiones y trabajo futuro

4.1. Conclusiones

En este trabajo, se ha diseñado y construido una suite de herramientas para la generación de documentos basados en plantillas LaTeX.

Estas herramientas se estructuran en torno a pdc, una librería escrita en python que implementa las funciones necesarias para la generación de documentos a partir de las plantillas. Sobre esta, se ha construido un servicio web, pdcs, que permite usar la funcionalidad de pdc de manera remota e independiente de la plataforma. También se ha desarrollado un cliente PHP para dicho servicio, sPHPellbook, que se ha utilizado para construir una aplicacion de prueba que genera documentos haciendo uso de todas las herramientas.

Como complemento a todo esto, se ha construido también una documentación web para ilustrar el funcionamiento de cada una de las piezas implementadas y una serie de scripts que permiten la instalación y desplegado rápido de un servicio pdcs.

Estas nuevas herramientas presentan una serie de ventajas frente a otras ya existentes:

- La estructura de las distintas herramientas permite generar documentos a diferentes niveles, ya sea como librería, como aplicación de consola o como servicio remoto.
- El sistema aporta una excelente calidad tipográfica y una enorme versatilidad de elementos usables en los documentos gracias a que se basa en LaTeX.
- Gracias al carácter modular de los decodificadores y constructores, se pueden utilizar múltiples formatos de entrada y salida con tan solo codificar estos pequeños módulos.
- Permite usar la funcionalidad a partir de un servicio REST fácil de integrar independientemente del lenguaje de programación de la solución cliente. Esto está complementado con un panel de control que permite la configuración vía web y unos scripts que favorecen la instalación.
- La estructuración en componentes reutilizables y la definición del documento en vista y contenido hacen más sencillo el proceso de construcción de un documento y permite cambios de estilo rápidos y reutilización de código.

Sin embargo, para proyectos pequeños, el hecho de usar LaTeX puede ser contraproducente, ya que el tiempo mínimo de configuración del sistema y creación de las plantillas para conseguir una calidad tipográfica que puede no ser necesaria es alto respecto a otras herramientas.

Más allá de las herramientas desarrolladas, este trabajo ha servido entre otras cosas para poner en práctica conceptos muy recurrentes durante la máster, como la estructuración en servicios o la orientación para el despliegue de aplicaciones en cloud. También ha servido para consolidar los conocimientos sobre tecnologías como Python, PHP, Django o Javascript y para completar dicha formación con nuevos conceptos, tecnologías o librerías utilizadas por primera vez en este trabajo.

4.2. Trabajo Futuro

Aunque se ha construido una base sólida y funcional para estas herramientas, aún necesitan ser mejoradas, por lo que se define también un trabajo futuro:

- Crear nuevos módulos que permitan soportar otros formatos para las plantillas y para la salida, como Markdown o HTML que hagan más fácil la configuración del sistema para las situaciones donde la calidad tipográfica de LaTeX no sea necesaria.
- Realizar librerías cliente para el servicio REST en más lenguajes, como C++ o JAVA.
- Revisar el código en estilo y comentarios de cara a poder construir una documentación más completa que contenga una especificación exhaustiva de las funciones de la librería pdc y del servicio REST de pdcs de cara a la construcción de pruebas automatizadas.
- Publicar el proyecto bajo una licencia de código abierto en alguna plataforma pública como Github, para que la comunidad pueda utilizarlo y mejorarlo. Esto permitiría también automatizar pruebas con herramientas como TravisCI [42] y mantener registro de bugs y sugerencias de implementación.

Bibliografía

[1] *ACE How-To guide*. URL: https://ace.c9.io/#nav=howto (visitado 05-2015).

[2] Geoffrey D. Austrian. *Herman Hollerith: Forgotten Giant of Information Processing*. Columbia University Press, 1982, págs. 178-179.

[3] *Biblioteca URL cliente (cURL)*. URL: http://php.net/manual/es/book.curl.php (visitado 05-2015).

[4] *Django admin bootstrapped docs*. URL: https://github.com/django-admin-bootstrapped/django-admin-bootstrapped (visitado 05-2015).

[5] *Django admin tools docs*. URL: https://django-admin-tools.readthedocs.io/en/latest/ (visitado 05-2015).

[6] *Django overview*. URL: https://www.djangoproject.com/start/overview/ (visitado 05-2015).

[7] *Django Rest framework docs*. URL: http://www.django-rest-framework.org/ (visitado 05-2015).

[8] *Facebook API website*. URL: https://developers.facebook.com/ (visitado 05-2016).

[9] World Wide Web Foundation. *History of the Web*. URL: http://webfoundation.org/about/vision/history-of-the-web/ (visitado 05-2016).

[10] *Gunicorn - WSGI server*. URL: http://docs.gunicorn.org/en/stable/index.html (visitado 05-2015).

[11] Hugo Haas. *Web Services Glossary*. 11 de feb. de 2004. URL: https://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/#webservice (visitado 05-2015).

[12] Hashicorp. *Vagrant documentation*. URL: https://www.vagrantup.com/docs/ (visitado 05-2015).

[13] Mitchell Hashimoto. *Vagrant: Up and Running*. O'Reilly, 2013.

[14] IETF. *Hypertext Transfer Protocol – HTTP/1.1*. URL: https://www.ietf.org/rfc/rfc2616.txt (visitado 05-2016).

[15] IETF. *The JavaScript Object Notation (JSON) Data Interchange Format*. URL: https://tools.ietf.org/html/rfc7159 (visitado 05-2016).

[16] IETF. *The OAuth 2.0 Authorization Framework*. Oct. de 1012. URL: https://tools.ietf.org/html/rfc6749 (visitado 05-2016).

[17] Parallax inc. *jsPDF library overview*. URL: https://parall.ax/products/jspdf (visitado 05-2016).

[18] Adobe Systems Incorporated. *PostScript LANGUAGE REFERENCE third edition*. URL: http://partners.adobe.com/public/developer/en/ps/PLRM.pdf (visitado 05-2016).

[19] George J.London. *Idiomatic Django Deployment - The Definitely Definitive Guide*. URL: http://rogueleaderr.com/post/65157477648/the-idiomatic-guide-to-deploying-django-in (visitado 05-2016).

[20] Valery Kholodkov. *The LaTeX Companion, Second Edition*. Packt Publishing, 2015.

[21] Donald Ervin Knuth. *Commemorative lecture for the Kyoto Prize*. Digital Typography. 1986.

[22] Donald Ervin Knuth. *Metafont: The Program*. Addison-Wesley, 1996.

[23] Leslie Lamport. *LaTeX: A document preparation system (2nd edition)*. Addisson-Weley, 1994.

[24] Tim Berners Lee. *A brief history of the web*. 1993. URL: `https://www.w3.org/DesignIssues/TimBook-old/History.html` (visitado 05-2016).

[25] Michael Leibman. *SlickGrid Wiki*. URL: `https://github.com/mleibman/SlickGrid/wiki` (visitado 05-2015).

[26] John Lombradi. "Welcome Microsoft Word in A New Version of Windows". En: *InfoWorld* (). URL: `http://jvlone.com/computerpub/InfoWorld/IW_1990-01-15_x-x_MicrosoftWordWindows10.pdf` (visitado 05-2016).

[27] Frank Mittelbach y Michel Goossens. *The LaTeX Companion, Second Edition*. Addison-Wesley, 2014.

[28] *Mkdocs overview*. URL: `http://www.mkdocs.org/#overview` (visitado 05-2015).

[29] *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. Nov. de 1996. URL: `https://tools.ietf.org/html/rfc2045` (visitado 05-2015).

[30] *NGINX documentation*. URL: `https://nginx.org/en/docs/` (visitado 05-2015).

[31] Tim O'Reilly. *What Is Web 2.0? Design Patterns and Business Models for the Next Generation of Software*. O'Really. URL: `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html` (visitado 05-2016).

[32] Overleaf. *Nginx Essentials*. URL: `https://www.overleaf.com/benefits` (visitado 05-2016).

[33] *Parser for command-line options, arguments and sub-commands*. URL: `https://docs.python.org/3/library/argparse.html` (visitado 05-2015).

[34] The Pallets Projects. *Welcome to Jinja2: Jinja2 docs*. URL: `http://jinja.pocoo.org/docs/dev/` (visitado 05-2016).

[35] Armin Ronacher. *Python on wheels*. 27 de ene. de 2014. URL: `http://lucumr.pocoo.org/2014/1/27/python-on-wheels/` (visitado 05-2016).

[36] Miles Z. Sterrett. *Wicked PDF plugin*. URL: `http://www.mileszs.com/wicked-pdf-plugin/` (visitado 05-2016).

[37] Jonathan Strickland. *Web 1.0 defined*. 2008. URL: `http://computer.howstuffworks.com/web-101.html` (visitado 05-2016).

[38] *Supervisor: A Process Control System*. URL: `http://supervisord.org/` (visitado 05-2015).

[39] Cisco Systems support team. *Why are mail attachments bigger than the original file?* Oct. de 2014. URL: `http://www.cisco.com/c/en/us/support/docs/security/email-security-appliance/118499-qa-esa-00.html` (visitado 05-2015).

[40] *The application/pdf Media Type*. 2004. URL: `https://tools.ietf.org/html/rfc3778` (visitado 05-2015).

[41] *The "data" URL scheme*. URL: `https://www.ietf.org/rfc/rfc2397.txt` (visitado 05-2015).

[42] *TravisCI - Getting started*. URL: `https://docs.travis-ci.com/user/getting-started/` (visitado 05-2015).

[43] *Twitter*. URL: `https://dev.twitter.com/rest/public` (visitado 05-2016).

[44] Funding Universe. *WordPerfect Corporation History*. 1995. URL: `http://www.fundinguniverse.com/company-histories/wordperfect-corporation-history/` (visitado 05-2016).

[45] Information week. *From EDI To XML And UDDI: A Brief History Of Web Services*. 2001. URL: `http://www.informationweek.com/from-edi-to-xml-and-uddi-a-brief-history-of-web-services/d/d-id/1012008?` (visitado 05-2016).

[46] *XeTeX on the Web*. 21 de nov. de 2014. URL: `http://tug.org/xetex/` (visitado 05-2015).

A

Anexos

A.1. Servidor de prueba

Con el fin de que los lectores de este documento tengan una forma de probar todas las herramientas desarrolladas en esta práctica, se ha habilitado un servidor de prueba y unas URL de acceso:

Documentación web: http://tryspellbook.ddns.net:9900/

Panel de control: http://tryspellbook.ddns.net:9999/ Usuario: admin Password: 1234

Aplicación de prueba: http://tryspellbook.ddns.net:9901/

Al utilizar este servidor hay que tener en cuenta:

- Debido a que no está en un servicio de hosting profesional, es posible que el servidor no esté disponible.
- Hay que tener en cuenta que los cambios en la configuración se guardan en el servidor y pueden producir que falle la generación de documentos de otros usuarios que estén haciendo uso del servicio.

A.2. Ejemplo de generación

En este apartado se presenta el código LaTeX de ejemplo para generar un albarán de entrega. Estos mismos códigos y configuraciones son los descritos en la documentación web de la aplicación disponible en las URLs adjuntas en el anexo I.

A.2.1. Código LaTeX de componentes y paquetes

Mediante el panel de control de pdcs, se configuran los siguientes componentes y paquetes necesarios para la generación.

Component name: components/header/companyHeader

```
%-----------------------------------------------------
%   COMPANY HEADER
%-----------------------------------------------------
\setlength{\parindent}{0cm}                 % Remove paragraph indentation
\newcommand{\tab}{\hspace*{2em}}            % Defines a new command for some horizontal space

\fancypagestyle{plain}{                     % Definition of the header
    \fancyhf{}
    \setlength{\headheight}{110pt}
    \renewcommand{\headrulewidth}{0pt}
    \fancyhead[L]{
        \begin{tabular}{ c }
            \includegraphics[width={@ conf.image_width @}cm,height={@ conf.image_height @}cm,keepaspectratio]{{@ conf.image @}} \\
            \small {@ conf.institution_title @} \\
        \end{tabular}
    }
    \fancyhead[R]{\textsf{
        \begin{tabular}{ cc}
            \small {@ conf.company_address_1 @} & \\
            {@ conf.company_address_2 @} & \\
            {@ conf.company_address_3 @} \\
        \end{tabular}
    }}
}
\thispagestyle{plain}

%-----------------------------------------------------
%   END COMPANY HEADER
%-----------------------------------------------------
```

Component name: components/deliveryNoteHeader

```
%-----------------------------------------------------
%   DELIVERY NOTE HEADER
%-----------------------------------------------------
\textsf{
    \begin{tabular}{m{8cm}rrr}
        \textbf{\Large {@ conf.title @}\hfill}\hspace*{1.25cm}\large {@ conf.code @}\hfill &
        \hspace*{0.5cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(1cm,1cm)
                \psbarcode{{@ conf.qr_content @}}{}{qrcode}
            \end{pspicture}} &
        \hspace*{0.25cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(3,1cm)
            \psbarcode{{@ conf.bar_content @}}{includecheck height=0.7}{interleaved2of5}
            \end{pspicture}}
         \\
    \end{tabular}
}
%-----------------------------------------------------
%   END DELIVERY NOTE HEADER
%-----------------------------------------------------
```

Component name: components/document/basicDocument

```
%-----------------------------------------------------
%   BASIC DOCUMENT
%-----------------------------------------------------

% Fixed document configuration and packages
\documentclass[12pt,a4paper]{article}

\usepackage[left={@ conf.margin_left @}cm,right={@ conf.margin_right @}cm,top={@
    conf.margin_top @}cm,bottom={@ conf.margin_bottom @}cm,marginparwidth=3.4cm]{
    geometry}
\setlength{\unitlength}{1mm}          % Set the unit length to millimeters
\usepackage[utf8]{inputenc}

\usepackage{graphicx}
\usepackage{wrapfig}
\graphicspath{ {{@resources_path@}} }

\newenvironment{myindentpar}[1]%
 {\begin{list}{}{%
          {\setlength{\leftmargin}{#1}}%
          \item[]%
 }
{\end{list}}


% Optional packages
{@ packages @}

\begin{document}
{@ content @}
\end{document}

%-----------------------------------------------------
%   END BASIC DOCUMENT
%-----------------------------------------------------
```

Component name: Component name: components/deliveryNoteContent

```
%-----------------------------------------------------
%   DOCUMENT CONTENT
%-----------------------------------------------------
\newcommand\Tstrut{\rule{0pt}{3ex}}         %"top" strut
\newcommand\Bstrut{\rule[-1.5ex]{0pt}{0pt}} %"bottom" strut
\newcommand{\TBstrut}{\Tstrut\Bstrut} %top&bottom struts

\vspace*{1cm}
\textsf{
\textbf{\large Datos del envio}\hfill\vspace*{0.25cm}\break
    \begin{tabular}{|>{\centering}m{4cm}|m{12.25cm}|}\hline
        \textbf{\small FECHA} & {@ conf.date @} \TBstrut\\\hline
        \textbf{\small CLIENTE} & {@ conf.client @} \TBstrut\\\hline
        \textbf{\small IDENTIFICADOR} & {@ conf.id @}  \TBstrut\\\hline
    \end{tabular}
}

\vspace*{1cm}
\textsf{
\textbf{\large Contenido}\hfill\vspace*{0.25cm}\break
    \begin{tabular}{|>{\centering}m{4cm}|>{\centering}m{8cm}|m{4cm}|}\hline
```

```
22          \textbf{\small ITEM} & \textbf{\small DESCRIPCION} & \textbf{\small
    NUMERO DE SERIE}\TBstrut\\\hline
            {% for product in conf.products %}{@ product.id @} & {@ product.
    description @} & {@ product.snumber1 @}\break {@ product.snumber2 @} \\\hline
    {% endfor %}
24      \end{tabular}
}
26
\vspace*{1cm}
28 \textsf{
\textbf{\large Observaciones}\hfill\vspace*{0.25cm}\break
30      \begin{tabular}{|m{17cm}|}\hline
            \\
32          \\
            \\\hline
34      \end{tabular}
}
36
\vspace*{1cm}
38 \textsf{
        \begin{tabular}{m{8.5cm}m{8.5cm}}
40          \vspace*{0.25cm}CLIENTE&\vspace*{0.25cm} \\
            & \\
42          & \\
        \end{tabular}
44 }

46 %-----------------------------------------------------
```

Package name: packages/pst-barcode

```
\usepackage{pst-barcode}
```

Package name: packages/fancyhdr

```
1 \usepackage{fancyhdr}
```

Package name: packages/babel

```
1 \usepackage[spanish]{babel}
\selectlanguage{spanish}
```

Package name: packages/tabularx

```
\usepackage{tabularx}
```

A.2.2. Árbol de contenido

Tras introducir todos los datos provenientes de un formulario explicado en el apartado de aplicación de prueba de este documento, el árbol de contenido del documento resultante es:

```
{
  "content": [{
    "content": [],
    "packages": [{
      "content": [],
      "template": "packages\/pst-barcode",
      "conf": []
    }, {
      "content": [],
      "template": "packages\/fancyhdr",
      "conf": []
    }, {
      "content": [],
      "template": "packages\/babel",
      "conf": []
    }, {
      "content": [],
      "template": "packages\/tabularx",
      "conf": []
    }],
    "template": "components\/header\/companyHeader",
    "conf": {
      "image": "logo",
      "image_width": 6,
      "image_height": 2,
      "institution_title": "Spellbook, Generadores de documentos",
      "company_address_1": "C\/Alcala S\/N",
      "company_address_2": "28036, Madrid, Spain",
      "company_address_3": "Tlf.: +34 91 568 25 25"
    }
  }, {
    "content": [],
    "packages": [],
    "template": "components\/deliveryNoteHeader",
    "conf": {
      "title": "ALBARAN\t DE ENTREGA",
      "code": "A-19062016-1711-UN",
      "qr_content": "A-19062016-1711-UN",
      "bar_content": "190620161711"
    }
  }, {
    "content": [],
    "packages": [],
    "template": "components\/deliveryNoteContent",
    "conf": {
      "date": "19\/06\/2016",
      "client": "Universidad Autonoma de Madrid",
      "id": "UAM",
      "notes": "Nada que declarar",
      "products": [{
        "id": "3245345435",
        "description": "Equipo servidor Intel x64 HD",
        "snumber1": "1354353",
        "snumber2": "23453245345"
      }, {
        "id": "3423443234",
        "description": "Equipo servidor rack 5U Xeon x86",
        "snumber1": "4534633245",
        "snumber2": null
      }]
    }
  }],
```

```
  "packages": [],
  "template": "components\/document\/basicDocument",
  "conf": {
    "margin_left": 2,
    "margin_right": 2,
    "margin_top": 1,
    "margin_bottom": 4
  }
}
```

A.2.3. Documento generado

Tras enviar el anterior árbol de contenido al servidor, el código LaTeX que es introducido al constructor de PDF es:

```
%-------------------------------------------------
%    BASIC DOCUMENT
%-------------------------------------------------

% Fixed document configuration and packages
\documentclass[12pt,a4paper]{article}

\usepackage[left=2cm,right=2cm,top=1cm,bottom=4cm,marginparwidth=3.4cm]{geometry}
\setlength{\unitlength}{1mm}          % Set the unit length to millimeters
\usepackage[utf8]{inputenc}

\usepackage{graphicx}
\usepackage{wrapfig}
\graphicspath{ {/home/guillermo/proyectos/spellbook/pdcs/compiler/pdc/resources/}
    }

\newenvironment{myindentpar}[1]%
 {\begin{list}{}%
         {\setlength{\leftmargin}{#1}}%
         \item[]%
 }
{\end{list}}

% Optional packages
\usepackage{pst-barcode}\usepackage{fancyhdr}\usepackage[spanish]{babel}
\selectlanguage{spanish}\usepackage{tabularx}

\begin{document}
%-------------------------------------------------
%    COMPANY HEADER
%-------------------------------------------------
\setlength{\parindent}{0cm}              % Remove paragraph indentation
\newcommand{\tab}{\hspace*{2em}}          % Defines a new command for some
    horizontal space

\fancypagestyle{plain}{                  % Definition of the header
    \fancyhf{}
    \setlength{\headheight}{110pt}
    \renewcommand{\headrulewidth}{0pt}
    \fancyhead[L]{
        \begin{tabular}{ c }
            \includegraphics[width=6cm,height=2cm,keepaspectratio]{logo} \\
            \small Spellbook, Generadores de documentos \\
        \end{tabular}
```

```
    }
    \fancyhead[R]{\textsf{
        \begin{tabular}{ cc}
            \small C/Alcala S/N & \\
            28036, Madrid, Spain & \\
            Tlf.: +34 91 568 25 25 \\
        \end{tabular}
    }}
}
\thispagestyle{plain}

%----------------------------------------------------------
%    END COMPANY HEADER
%----------------------------------------------------------
%----------------------------------------------------------
%    DELIVERY NOTE HEADER
%----------------------------------------------------------
\textsf{
    \begin{tabular}{m{8cm}rrr}
        \textbf{\Large ALBARAN DE ENTREGA\hfill}\hspace*{1.25cm}\large A-14062016-1901-UN\hfill &
        \hspace*{0.5cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(1cm,1cm)
                \psbarcode{A-14062016-1901-UN}{}{qrcode}
            \end{pspicture}} &
        \hspace*{0.25cm}&
        \raisebox{-.5\height}{
            \begin{pspicture}(3,1cm)
            \psbarcode{140620161901}{includecheck height=0.7}{interleaved2of5}
            \end{pspicture}}
         \\
    \end{tabular}
}
%----------------------------------------------------------
%    END DELIVERY NOTE HEADER
%----------------------------------------------------------
%----------------------------------------------------------
%    DOCUMENT CONTENT
%----------------------------------------------------------
\newcommand\Tstrut{\rule{0pt}{3ex}}         %"top" strut
\newcommand\Bstrut{\rule[-1.5ex]{0pt}{0pt}} %"bottom" strut
\newcommand{\TBstrut}{\Tstrut\Bstrut} %top&bottom struts

\vspace*{1cm}
\textsf{
\textbf{\large Datos del envio}\hfill\vspace*{0.25cm}\break
    \begin{tabular}{|>{\centering}m{4cm}|m{12.25cm}|}\hline
        \textbf{\small FECHA} & 14/06/2016 \TBstrut\\\hline
        \textbf{\small CLIENTE} & Universidad Autonoma de Madrid \TBstrut\\\hline
        \textbf{\small IDENTIFICADOR} & UAM  \TBstrut\\\hline
    \end{tabular}
}

\vspace*{1cm}
\textsf{
\textbf{\large Contenido}\hfill\vspace*{0.25cm}\break
    \begin{tabular}{|>{\centering}m{4cm}|>{\centering}m{8cm}|m{4cm}|}\hline
        \textbf{\small ITEM} & \textbf{\small DESCRIPCION} & \textbf{\small NUMERO DE SERIE}\TBstrut\\\hline
        3245345435 & Equipo servidor Intel x64 HD & 1354353\break 23453245345 \\\hline 3423443234 & Equipo servidor rack 5U Xeon x86 & 4534633245\break 234523453462 \\\hline
```

```
    \end{tabular}
}

\vspace*{1cm}
\textsf{
\textbf{\large Observaciones}\hfill\vspace*{0.25cm}\break
        Nada que declarar
}

\vspace*{1cm}
\textsf{
    \begin{tabular}{m{8.5cm}m{8.5cm}}
        \vspace*{0.25cm}CLIENTE&\vspace*{0.25cm}EMPRESA \\
        & \\
        & \\
    \end{tabular}
}

%-------------------------------------------------------
\end{document}

%-------------------------------------------------------
%    END BASIC DOCUMENT
%-------------------------------------------------------
```

El PDF generado por la aplicación a partir del código anterior se adjunta en la siguiente página.

generic logo company
Spellbook, Generadores de documentos

C/Alcala S/N
28036, Madrid, Spain
Tlf.: +34 91 568 25 25

ALBARAN DE ENTREGA

A-14062016-1913-UN

Datos del envo

FECHA	14/06/2016
CLIENTE	Universidad Autonoma de Madrid
IDENTIFICADOR	UAM

Contenido

ITEM	DESCRIPCIN	NMERO DE SERIE
3245345435	Equipo servidor Intel x64 HD	1354353 23453245345
3423443234	Equipo servidor rack 5U Xeon x86	4534633245 234523453462

Observaciones

Nada que declarar

CLIENTE

EMPRESA