UNIVERSIDAD POLITÉCNICA DE MADRID

Proyecto de Fin de Grado
Grado en Ingeniería de Software

Desarrollo de una app de visualización de PDF con guardado en la nube

Autor:
Álvaro Morales González
José María Puerta González

Supervisor:
Luis Fernando de Mingo Lopez

Curso Académico:
2018/2019

escuela técnica superior de ingeniería de sistemas informáticos

Resumen

Este Proyecto de Fin de Grado propone una solución a la correcta y rápida visualización y renderización de archivos en formato PDF en el contexto de la computación ubicua, concretamente para dispositivos iOS, incorporando además el guardado en la nube.

El problema que se intenta resolver parte de una necesidad real que se planteó en una aplicación de lectura basada en la nube llamada Nubico. Nubico contaba con un gran número de libros en formato PDF de todo tipo. Aunque los visores que había en el mercado eran capaces de abrirlos, no conseguían buenos tiempos de carga, y en algunos casos las renderizaciones eran incorrectas. La idea detrás de este proyecto es crear un visor que sea capaz de optimizar al máximo la carga y la renderización de los archivos a través de algoritmos que permitan paralelizar el trabajo y priorizar las tareas. Añadiendo la posibilidad de guardado en la nube, aumentamos la posibilidad de hacer una aplicación competitiva y darle al usuario la posibilidad de disponer de los archivos en el momento que lo requiera.

La solución de este Proyecto, se basa por tanto en resolver 3 subproblemas principales.

- El primero de ellos es la renderización de los archivos, concretamente de las páginas de estos archivos. Aunque existen varias opciones, se ha optado por hacer uso de CoreGraphics. CoreGraphics hace uso de la tecnología Quartz para llevar a cabo una representación 2D ligera, y de alta fidelidad.

- El segundo problema tiene su fundamento en el manejo de la concurrencia de la renderización de esas páginas.

- Finalmente, el último problema planteado consiste en el análisis y selección del algoritmo más óptimo, que haciendo uso del control del sistema concurrente, sea capaz de obtener mejores resultados en la experiencia y uso del usuario final.

Este renderizador es un mero componente que forma parte de un conjunto global formado por una aplicación que utiliza este componente, y un servidor.

El documento se estructura de la siguiente forma: primero se exponen el contexto que existe sobre la renderización de PDFs para después continuar con objetivos que se pretender conseguir. A continuación se hace un análisis de los requisitos para así exponer de forma clara como se ha diseñado el sistema, como la arquitectura y las tecnologías usadas. Para concluir, el documento cierra con unas conclusiones y una visión de como podría evolucionar el proyecto.

Abstract

In this proyect suggest a solution for a correct and fast display and render from PDF files with ubiquitous computing, exactly for iOS device and backend.

The problem comes from one app with cloud calls Nubico. It has many books with PDF format. However the actual displays that we can find in the market, can not open correctly and it does not have good render times. In some cases the render from the file is not correct. The idea behind this proyect is create a new display can optimize the render from the file with new algorithms and let work in paralel to reduce the threath and priorize task.

The solution is solve three principal problems.

- First, render a file, exactly only the pages for those files. Howerver exist some options, in this case is used CoreGraphics, which use Quartz-
- The second problem is how use concurrency and renderization in those pages.
- Finally, the last problem is analize and select the most optimal algorithm, which adapt to our program and give us the less time to render moreover the less complexity-

The document is structured in the following way: the context that exists about the rendering of PDF files is exposed. Then, the objectives that are intended to be achieved. An analysis of the requirements to explain clearly how the system was designed, and the chosen architecture. To conclude, the document closes with some conclusions and a vision of how the project could evolve.

Índice

Índice de Tablas

Índice de Figuras

Capítulo 1

Introducción

El uso del formato PDF (Portable Document Format) está ampliamente extendido. Fue inicialmente desarrollado por Adobe System a principios de 1991. Sin embargo, no fue hasta 1993 cuando se comercializó la primera versión, y hasta 2008 cuando se lanzó oficialmente como estándar abierto.

Este formato se desarrolló con el único objetivo de presentar documentos de una manera independiente al software, hardware o sistema operativo que se utilice. De esta forma, cada archivo PDF encapsula toda la información que necesita para poder mostrarse, como la fuente, los colores o las imágenes.

A diferencia de otros formatos parecidos, el formato PDF es un documento fixed-layout. Esto significa que tanto su tamaño como la disposición de sus elementos está definido y es inmutable.

La longevidad del formato PDF unido a su amplio uso, lo convierte en un formato robusto pero complejo de tratar. Aunque mostrar una simple página no es una tarea excesivamente compleja, tratar con su sistema de links o su tabla de contenidos si que lo es. Además, conseguir todo esto utilizando una interface de usuario amigable es un trabajo que conlleva mucho esfuerzo.

Esta es la principal razón por la cual no existen demasiadas librerías para visualizar PDFs y tratar con ellos. Las librerías de software libre tienen bastantes puntos de mejora como puede ser el soporte o la calidad de las páginas. Las comerciales, por su parte, tienen precios elevados que no están al alcance de todos los desarrolladores.

Es cierto que en la mayoría de los casos de uso no es necesario tener un visor demasiado potente. Pero en aquellos casos en los que si es necesario, el objetivo principal de cualquier software es transmitir calidad al usuario final.

En casos de computación híbrida, la veda se abre un poco para dejar paso a las vistas basadas en web que pueden potenciar la visualización de este formato haciendo uso de librerías en otros lenguajes. Sin embargo, aquello que se mejora en un sentido (con el visor) se pierde en la experiencia global de usuario al utilizar la aplicación

Por otro lado, desde los inicios del año 2000, cuando los móviles empezaron a incluir un sistema operativo capaz de soportar aplicaciones básicas como correo electrónico o mensajería instantánea, se han ido creando teléfonos los cuales son capaces de realizar la mayoría de las necesidades básicas de las personas. El término que se conoce para estos dispositivos es el de smartphone.

Un smartphone es un celular con pantalla táctil, que permite al usuario conectarse a internet, gestionar cuentas de correo electrónico e instalar otras aplicaciones y recursos a modo de pequeño computador.

Es por ello, que la capacidad de estos dispositivos para poder ofrecer esos recursos cotidianos, en un solo aparato, hace que el uso actual en todo el mundo sea de más de un 70 % de la población. Los smartphones han superado a los ordenadores y a los portátiles en uso y han pasado a ser los principales dispositivos usados para conectarse a internet.

El uso de los smartphones ha pasado a ser casi fundamental y necesario en la vida de muchas personas, es normal que se puedan encontrar aplicaciones para todo tipo de usos, desde juegos para el entretenimiento, calendarios para organización, apps para poder tomar notas, etc. Sin embargo, sigue habiendo sectores y aplicaciones que no terminan de satisfacer (o de gustar) a los usuarios... El desarrollo de apps para dispositivos iOS es más difícil que para sistemas Android dado que requiere un lenguaje de programación más personalizado y el uso de un programa especifico para desarrollar apps como es Xcode. Además, requiere de una licencia para publicar aplicaciones en la AppStore (Aplicación donde se pueden descargar las aplicaciones del dispositivo iOS) y cuesta más que una licencia para publicar en PlayStore (Aplicación donde se pueden descargar las aplicaciones del dispositivo Android).

Todo esto hace que muchos programadores y empresas opten primero en la creación de apps para un sistema Android , que para sistemas iOS. Esto abre una gran posibilidad de poder desarrollar apps que aún no estén en el mercado o de ofrecer un servicio mejorado a las que ya están.

Los dos principales formatos de lectura a día de hoy son ePub y PDF. Construir un visor ePub conlleva un desarrollo más complejo ya que es necesario utilizar un navegador web embebido que sea capar de renderizar el formato.

Capítulo 2

Estado del arte

2.1. PDF

Actualmente, existen varias librerías para cargar archivos PDF en dispositivos iOS. Hay opciones Open Source y comerciales. Cada una de ellas con sus pros y sus contras.

El framework comercial más conocido actualmente es PSPDFKit. Es rápido abriendo PDFs simples y además tiene muchas features como anotaciones, thumbnails o selección de elementos entre otros. Sin embargo, es lento cuando se intenta abrir un PDF de gran tamaño o con varias capas de imágenes. En muchos casos, esta y otras versiones comerciales no dejan acceso al código fuente, aunque si tienen soporte personalizado. Por su parte, el frameworks Open Source más conocido es PDF Reader Core. Este framework es más sencillo que PSPDFKit. Se trata de un visor más básico que solo soporta links y cambios de orientación. Además, es más lento en la carga de archivos simples, y mucho más lento en la carga de archivos pesados, llegando en algunos casos incluso a colgarse. Como contrapartida, hay que añadir que la actividad de este y la mayor parte de los repositorios de visores de PDF sostenidos por la comunidad deja bastante que desear.

La idea detrás de este proyecto es conseguir y perfeccionar lo mejor de ambas partes. Es decir, conseguir un visor sencillo pero muy rápido, donde el enfoque final sea la lectura. Los archivos, en este caso libros, deben abrirse casi instantáneamente.

Por otro lado, Apple ha sacado recientemente un framework por encima de Core-

Graphics para la renderización de PDFs. Esta disponible a partir de versiones iOS 11 o superiores y macOS 10.4 o superiores. CoreGraphics es la capa sobre la que está construido el renderizador, se trata de APIs a más bajo nivel para pintar sobre diferentes layers, las páginas del PDF.

Algunas de las aplicaciones más importantes a las que se le podría hacer competencia son PDFPro 3, iBooks, AdobeAcrobarReader y PDFViewer pro.

2.2. Computación ubicua

Mark Weiser es el autor de este concepto, que se está volviendo cada vez más importante en los últimos años teniendo en cuenta la apariencia de los teléfonos inteligentes, tablets y dispositivos portátiles (por ejemplo, relojes inteligentes).

Presentó tres modelos que podrían usarse para desarrollar sistemas ubicuos:

- **Pestañas:** dispositivos portátiles del tamaño de un centímetro que se pueden usar.
- **Almohadillas:** dispositivos portátiles del tamaño de decímetros que se ajustan a las manos de una persona.
- **Tableros:** dispositivos del tamaño de un metro.

La idea principal es que la informática puede aparecer en cualquier lugar y en cualquier momento. Puede ocurrir en cualquier dispositivo, ubicación y formato. Los usuarios pueden usar los dispositivos sin problemas para realizar cualquier tarea.

La computación ubicua se relaciona con la computación distribuida y móvil, dos temas que serán explorados profundamente en este proyecto.

Capítulo 3

Objetivos y metas

En este capitulo se introducirán los conceptos básicos necesarios para entender el problema que se intenta resolver, y arrojar un poco de luz sobre sus posibles soluciones.

3.1. Motivación

La principal motivación para este proyecto es la adquisición de conocimientos a la hora de implementar una aplicación con una arquitectura y tecnologías específicas, y el trabajo que ello conlleva. Para este caso, se usan varias tecnologías que actualmente están en crecimiento constante, lo que favorece al desarrollo y a la futura evolución de la aplicación. Otra motivación es el uso de tecnologías ágiles en el desarrollo, con la distribución de roles y las planificaciones adecuadas que hacen que el equipo consiga desarrollar el proyecto de manera eficaz y próspera.

3.2. Metas

La principal meta para este proyecto es el diseño y desarrollo de una aplicación móvil desarrollada en iOS, que ofrece la posibilidad de visualizar PDF's y poder almacenarlos en la nube. De esta manera se quiere crear una app que pueda ser competitiva a la que actualmente se ofrece por parte de la plataforma nativa iOS. También se quiere adquirir más conocimientos en la tecnología NodeJS la cual permite la creación de

API's implementando el propio servidor, sin requerir de un sistema externo. Por otro lado, para el desarrollo del proyecto, se han elegido el uso de metodologías ágiles. Otras de las metas es la adquisición de conocimientos de desarrollo, que puedan ser aplicados a entornos de trabajo y que a su vez sean métodos usados por grandes compañías.

3.3. Objetivos

- **Análisis de las posibles tecnologías que se pueden usar para el backend y la base de datos.** Actualmente en el sector existen varias tecnologías factibles para este proyecto. Realizar un buen análisis de las necesidades y los requisitos será fundamental para optar que tecnología será la utilizada.

- **Implementación de una API Rest**. Aprender cuales son las principales características y la arquitectura adecuada para una API eficiente.

- **Diseño y desarrollo del backend.** El sistema tiene que poder dar un servicio rápido y asíncrono para poder utilizarlo con múltiples usuarios sin que se pueda notar una deficiencia en el rendimiento.

- **Modelar un renderizador de PDFs.** Quizá la pieza más importante del proyecto. Puesto que es el elemento principal que el usuario final va a utilizar. Es necesario que tenga una carga rápida, que sea liviano y que tenga una capacidad de respuesta elevada a interacciones del usuario

    1. Carga rápida y liviana.
    2. Simplicidad.
    3. Diseño de un algoritmo concurrente.
    4. Renderización y zoom de las páginas.
    5. Mantener el punto de lectura.
    6. Manejo de links.
    7. Tabla de contenidos.

8. Orientación.

- **Diseñar y desarrollar un cliente iOS.** Teniendo especial cuidado en su arquitectura así como en la reutilización, en la medida de lo posible, del código en iOS, Mac y watchOS. Para ello, la plataforma móvil se desarrollará usando la arquitectura VIPER (View-Interactor-Presenter-Entity-Routing).

- **Explorar el lenguaje de programación Swift y justificar su elección sobre Objective-C.** Swift es un lenguaje de programación más moderno enfocado en la seguridad, la expresividad y el rendimiento.

- **Diseño e implementación de los test.** Test unitarios y test funcionales. Aprendizaje de los frameworks más usados para testing. Valoración del uso de frameworks que se adapten de manera acorde a las tecnologías que se usan, observando si son factibles o no a los tiempos de desarrollo.

- **Trabajar con metodologías ágiles.** El trabajo en equipo solo es satisfactorio cuando se realiza con compromiso por todas las partes. En este proyecto el uso de las metodologías ágiles supone un factor importante tanto en el desarrollo como en la comunicación de todos los integrantes, reforzando cualidades positivas que favorecen al ambiente profesional.

Capítulo 4

Análisis y requisitos

En esta sección se describen los requisitos asociados con la implementación del proyecto.

4.1. Análisis tecnológico: Elección de tecnologías

En la implementación de cualquier proyecto existen siempre varios pasos que se deben de seguir para poder realizarlo de manera satisfactoria. El primero de ellos, es el stack tecnológico.

Actualmente existen varias tecnologías que son las más utilizadas, ya sea por flexibilidad, facilidad en el desarrollo, variedad de frameworks, compatibilidad con varios plugins, etc. Para este proyecto, se ha elegido el uso de MongoDB para la base de datos, NodeJS en la parte backend y Swift para el desarrollo de la app móvil.

4.1.1. Base de datos

En la mayoría de los proyectos se apuestan por base de datos relacionales o denominadas a partir de ahora BDR. Esto se debe a la cantidad de datos que se deben soportar y a la cantidad de entidades que se poseen. Una BDR es un conjunto de tablas, formada por registros y campos. Estos registros representan a cada uno de los objetos descritos en la tabla, mientras que los campos corresponden con los atributos de los objetos. En aplicaciones con gran cantidad de entidades, que requieren de relaciones entre ellas,

este modelo es el más adecuado y el que más se suele adaptar a las necesidades de los clientes.

Principales características de una BDR que aplica para este proyecto:

- El lenguaje utilizado es el denominado SQL o lenguaje de consulta estructurada. Es el lenguaje estándar y todas las bases de datos relacionarles lo soportan. Se utiliza para agregar, actualizar y eliminar datos. Es el lenguaje más conocido, lo que facilita la resolución de problemas en las comunidades.

- Integridad de los datos. Las BDR utilizan restricciones que permiten relacionar los datos de manera coherente e integra. Para ello se hace uso de claves como son las claves primarias o las claves únicas. Teniendo claves se asegura el no duplicado de registros, además del uso de las mismas en las consultas para realizarlas de manera más dinámica.

Principales desventajas para este proyecto:

- Definición de un esquema. Para tener una BDR se requiere definir al principio del desarrollo un esquema estructurado y que posea todas las entidades que se van a soportar en la aplicación. Esto es un impedimento para los desarrollos evolutivos, ya que el insertar una nueva entidad, puede suponer que el esquema definido tenga que ser reajustado por completo.

Para este caso, se ha apostado por una base de datos no SQL por los siguientes motivos:

- La información que se desea almacenar no es compleja. Actualmente las entidades que se tienen en el sistema son usuarios y pdf. Esto hace que el tratamiento de los datos sea más versátil y las consultas puedan hacerse de manera más veloz.

- Se requiere la inserción de datos de manera no estructurada. La ausencia de un esquema en los registros permiten incluir atributos que no habían sido definidos con anterioridad. Esta cualidad permite poder almacenar datos extras procedentes del PDF en el momento de ejecución, o tener distintos objetos con atributos no compartidos.

- Escalabilidad horizontal. Mientras que en una BDR, la creación de una nueva entidad implica la creación de una o varias tablas para adaptar al correcto funcionamiento, con una base de datos noSQL se puede incluir la información simplemente creando un nuevo nodo en la base de datos.

- Consultas tipo Map-Reduce. Tener varios nodos podría suponer una deficiencia en el sistema, sin embargo, con este tipo de consultas se ejecuta las sentencias en todos los nodos ejecutado únicamente una porción de los datos y reuniendo la información final antes de ser devuelta al cliente.

4.1.2. Servidor

Muchos puntos a tener en cuenta al construir la parte servidor en una aplicación son: flexibilidad, rendimiento, monitorización, escalabilidad, mantenimiento...

Java es la base para muchos tipos de aplicaciones además de un estándar global para desarrollo y distribución. Es un lenguaje probado por una gran comunidad de desarrolladores y está diseñado para permitir el desarrollo de aplicaciones portátiles de elevado rendimiento. Al estar testeado por una gran variedad de arquitectos y programadores hace que Java sea de los lenguajes más robustos existentes hasta el momento.

PHP se trata de un lenguaje de programación interpretado en el servidor. En la actualidad es ampliamente usado en entornos de desarrollo por su facilidad de uso e integración con HTML. Uno de los puntos mas desfavorables de este lenguaje es la modularidad. La modularidad de un sistema es la capacidad de separar el sistema o aplicación en diversos módulos pudiendo tener un modelo vista controlador (MVC). PHP almacena todas las capas lógicas implementadas en un mismo archivo lo que impedimenta la separación en módulos.

NodeJS es un programa servidor, basado en el concepto de módulos que se pueden agregar al núcleo principal. Está desarrollado en JavaScript, lo que permite que dicho motor interprete el código y lo ejecute directamente, haciendo que sea mucho más rápido.

Uno de los factores que más han influido en la elección ha sido la gran cantidad de información que se puede obtener por parte de la comunidad. NodeJS tiene una gran

comunidad open source que ofrece módulos y respuestas a una amplia cantidad de problemas que se pueden presentar en el desarrollo de cualquier aplicación. Es importante ser eficiente en el tiempo de implementación, quedarse atascado en un problema puede suponer un retraso en las entregas del producto. Poder encontrar varias soluciones, hacen que el desarrollador aprenda a la vez que se siente motivado al ver como su producto avanza y no se atasca.

El siguiente punto tenido en cuenta, es la multiplataforma. NodeJS posibilita el uso en cualquier sistema operativo, no siendo una tecnología que dependa del tipo de sistema operativo.

Por último, se valora la escalabilidad. Al ser una tecnología orientada a módulos y eventos, favorece el posible crecimiento de la aplicación sin que su complejidad aumente. Poder tener separada la aplicación en módulos, facilita el testing, disminuye la complejidad del código y ofrece una alta estabilidad funcional.

4.1.3. Cliente

A nivel de cliente existen varias plataformas sobre las que se podría haber realizado el proyecto: Windows Phone, Web, Android o iOS entre otros. Sin embargo, para este proyecto solo se ha elegido uno de ellos.

El proyecto se ha realizado sobre la plataforma de Apple para iOS por dos razones. La primera son los conocimientos previos disponibles por los integrantes del proyecto. La segunda se puede computar como un factor que relaciona la velocidad de desarrollo y la cantidad de conocimientos adquiridos. Esto es así porque Apple tiene una buena base sobre la que desarrollar un visor para PDFs.

En ningún caso se han valorado como opciones aquellas plataformas con mejores librerías de terceros, ya que el proyecto en sí a nivel de cliente tiene como objetivo construir algo desde cero.

Por otro lado, también se ha descartado el uso de la plataforma Android; a pesar de tener más facilidades con librerías de terceros, las posibles soluciones a los problemas de renderizado y computación complicaban el desarrollo.

4.2. Arquitectura elegida: justificación

4.2.1. Servidor

La arquitectura usada se basa en los conceptos de **Clean Architecture** y **Arquitectura Hexagonal** que se adapta a la mayoría de los proyecto y escala de manera apropiada. Se escoge Express como framework para la API Rest y MongoDB para la base de datos, ya que el stack MEAN (Mongo + express + angular + Node) es bastante popular y posee bastante información al respecto.

Clean Architecture

Esta arquitectura se basa en la separación de capas para gestionar las peticiones de la API y las consultas a la base de datos. Ficheros separados en módulos que se integran de manera fácil con express además de la idea de que el código sea fácilmente testeable y que los test no estén ligados a ningún elemento tecnológico externo.

Clean Architecture[Mar08] (por Robert C. Martin) intenta seguir los principios SOLID de Michael Feathers para conseguir esta separación por capas.

- **Single Responsibility.** Una clase, modulo, método, etc, debe tener una y solo una razón para cambiar. Esto significa que una entidad de software debe tener solo una responsabilidad y hacer únicamente la tarea para la cual ha sido diseñada. De lo contrario si se asume más de una responsabilidad existirá un alto acoplamiento provocando que nuestra lógica de programación sea frágil ante cualquier cambio.

- **Open / Closed.** Las entidades de software deben estar abiertas para su extensión, pero cerradas para su modificación. Esto significa que una entidad de software debe ser fácilmente extensible sin necesidad de modificar su código existente. Si diseñamos sistemas extensibles serán menos propensos a errores ante cambios en los requisitos.

- **Liskov substitution.** Principio elaborado por Barbara Liskov que viene a decir que los objetos deben ser reemplazables por instancias de sus subtipos sin alterar el correcto funcionamiento del sistema. Por ejemplo, si tenemos una clase “Coche”

que tiene un método "arrancarMotor" y también tenemos una clase "Deportivo" que extiende de "Coche", podemos sustituir las referencias de "Coche" por referencias de "Deportivo". Aplicando este principio conseguimos validar que nuestras abstracciones sean correctas.

- **Interface segregation.** Tener interfaces específicas es mejor que tener una interfaz de propósito general. Este principio nos dice que nunca debemos implementar una interfaz que no vayamos a necesitar. Incumplir este principio conlleva que en nuestras implementaciones tengamos dependencias de métodos que no necesitamos pero nos vemos obligados a definir. Por lo tanto una interfaz la define el cliente que la consume, y no debe tener métodos que este no vaya a implementar.

- **Dependency inversion.** Se debe depender de abstracciones, y no de implementaciones. Significa que una clase concreta, no debe depender directamente de otra clase sino de una abstracción (interfaz) de esta. Aplicar este principio nos ayuda a reducir la dependencia en implementaciones específicas y así lograr que el código sea más reutilizable. No hay que confundir este principio con la "inyección de dependencias" que es una técnica que nos ayuda a la hora de aplicar este principio y conseguir que las colaboraciones entre clases no conlleven dependencias entre ellas.

En los últimos años, muchas arquitecturas se han creado a partir de estos principios. Todas ellas con la única finalidad de separar conceptos.

Es un error arquitectónico muy común acoplar la lógica de negocio con la lógica de datos o de presentación.

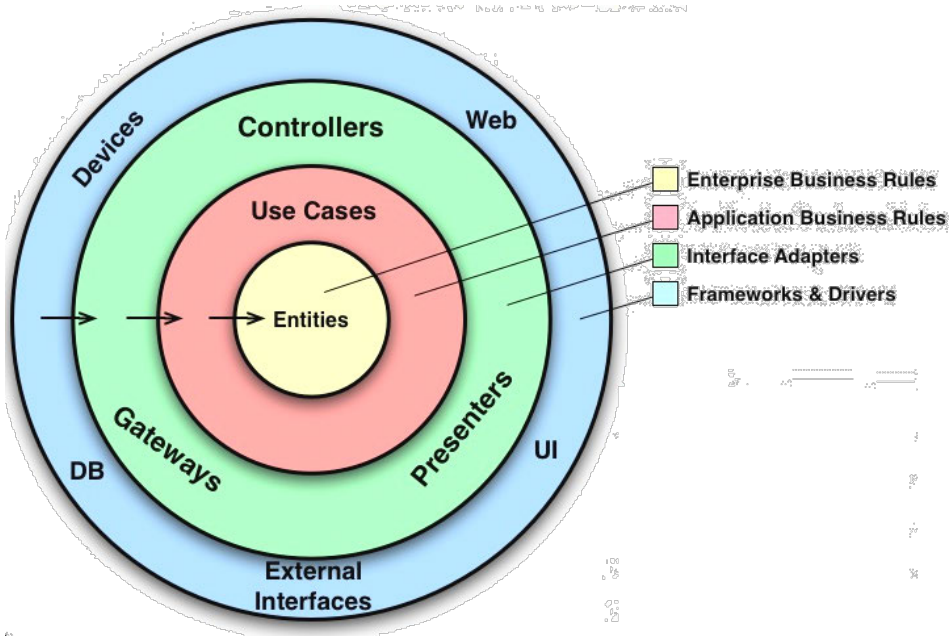


Figura 4.1: Estructura general de una posible arquitectura clean

Arquitectura Hexagonal

El objetivo es encapsular la funcionalidad para ser usada por cualquier elemento externo a través de adaptadores. Esta arquitectura se puede ver mediante el uso de los módulos que ofrece NodeJS. La idea de esta arquitectura es que cada una de las capas se encargue de sus propias responsabilidades. Con esta idea se pueden disponer unos límites en ciertas lógicas o funcionalidades y saber donde se debe colocar cada una de las partes y de como deben interactuar entre ellas. Aunque la idea de esta arquitectura se representa mediante un hexágono, lo más importante no es la forma, si no el concepto de lo que representa cada lado. Cada lado del polígono estaría asociado a un puerto de entrada o de salida a la aplicación. Por ejemplo, un puerto podría corresponder con las peticiones HTTP, e incluso otro puerto nos podría conectar con otro puerto de una aplicación. Con esta arquitectura lo que se consigue es que las dependencias se vayan creando de fuera hacia dentro, un ejemplo sería, que los Frameworks dependan de la capa de aplicación, y así sucesivamente [Luc16].

En el siguiente ejemplo se explica bien este tipo de arquitectura. Para ello se usan

las llamadas HTTP. Al realizar una llamada (ya sea de tipo GET, POST , PUT...) a nuestra aplicación, es nuestra capa Framework quien interpreta las rutas y las envía al controlador (estando aún en la capa Framework). Una vez procesada la información es la capa Aplicación la que se encarga de procesar la lógica de la petición, creando así un *servicio*. Por tanto ya tenemos la comunicación entre nuestra capa Framework y nuestra capa Aplicación, la cual, es nuestro servicio. Por último, nuestro servicio necesita de clases y entidades para poder obtener adecuadamente los datos necesarios, y esto se realiza en la capa de Dominio. De esta forma, la capa de Dominio le proporciona los datos a la aplicación, y esta los retorna a la capa Framework.

Con esta arquitectura, el mayor de los beneficios es la mantenibilidad que otorgamos a nuestro código. En caso de crear errores en el código o de tener malas decisiones, vamos a poder abordarlas de manera rápida y localizada en una sola capa. Otra característica es que conseguimos independizar nuestro código de librerías externas, pudiendo usar las librerías como herramientas y no como dependencias.

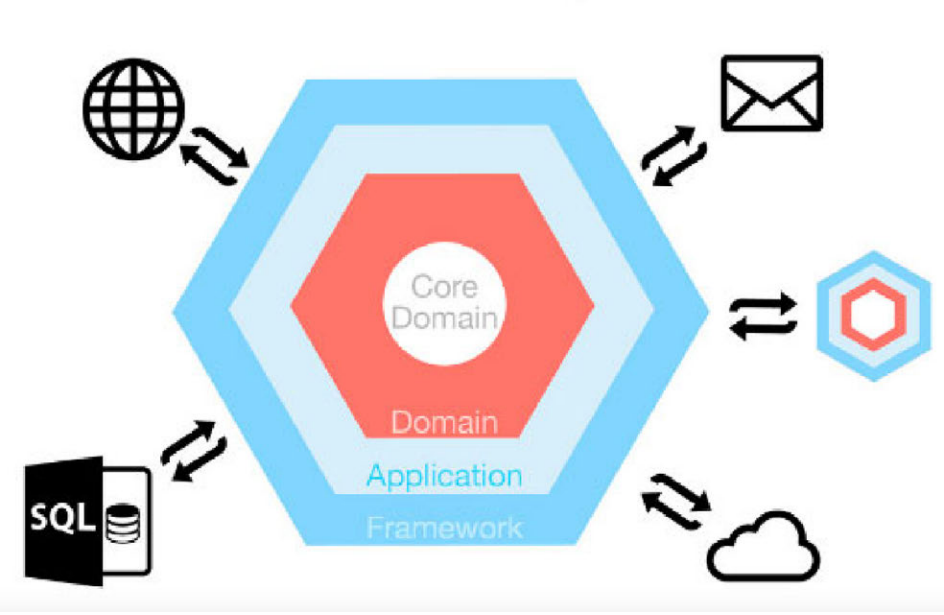


Figura 4.2: Diagrama de la arquitectura hexagonal

4.2.2. Cliente

Esta sección cubre la arquitectura de la plataforma iOS. Uno de los objetivos de la plataforma es soportar varios dispositivos (iOS, OS X y watchOS) reutilizando la mayor parte del código posible. Puesto que todas estas plataformas tienen diferentes interfaces, soportarlas requiere una buena planificación de diseño.

La aplicación de iOS esta dividida en una serie de frameworks con entidad propia. A pesar de que estos frameworks compartes elementos comunes, cada uno tiene una arquitectura que se ajusta a sus necesidades.

Frameworks

A la hora de desarrollar para la plataforma de Apple es importante contar con una serie de herramientas que nos faciliten algunas tareas. Este es el caso de los framework que complementan las APIs que trae el SDK de iOS.

Un framework es un set de código reutilizable y módular que sirve para construir componentes software de mayor nivel. La principal razón para utilizar frameworks es que se pueden compilar y ejecutar tests por separado. Además, gracias a la compilación incremental, permiten un mayor rendimiento en el desarrollo.

La aplicación esta formada por un total de 4 framework internos y 2 externos. Los frameworks creados internamente son:

- **ReaderKit.** Este fremework contiene todo lo necesario para poder renderizar un archivo en formado PDF dado un path.
- **Logger.** Este es un framework muy sencillo que contiene todas las clases necesarias para poder mostrar logs en consola.
- **Crypto.** Crypto es un framework que actúa de wrapper. Se utiliza para comprimir y descomprimir, así como para encriptar y desencriptar datos en batches.
- **App bundle.** El main bundle es el paquete que contiene las clases de la aplicación. Las pantallas de login, el manejo del ciclo de vida de la app, la capa de storage y red etc.

Los frameworks externos utilizados son:

- **ReactiveCocoa.** Framework que expone una interface basada en streamings de datos. Necesario para utilizar el paradigma de programación reactiva. Es el framework más utilizado y soportado por la comunidad, por encima de otros como RXSwift.
- **SQLite.** Framework basado SQL. Utilizado para el guardado del contenido generado por ReaderKit. Se ha optado por SQLite frente a otros componentes de persistencia como pueden ser CoreData o UserDefaults ya que la generación de tablas es muy sencilla utilizando su interface.

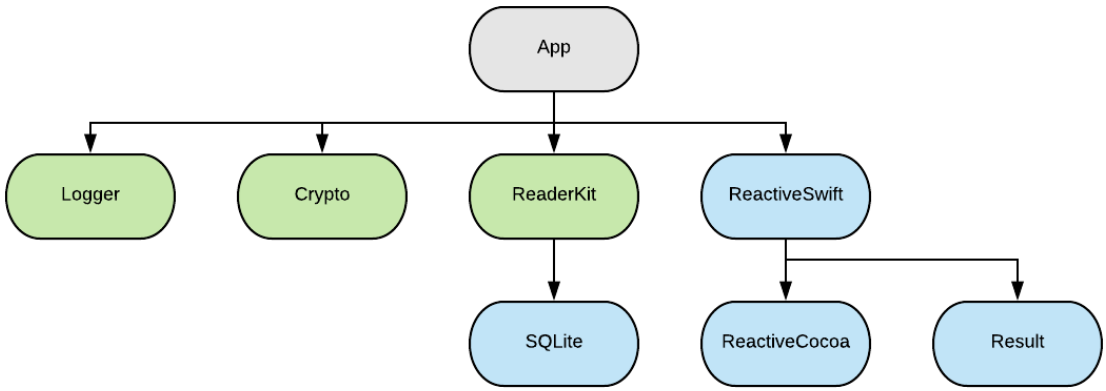


Figura 4.3: Jerarquía de framework utilizados del cliente iOS

La principal de esta subdivisión es la velocidad de desarrollo, el desacoplamiento y la separación estructurada de código.

Por un lado la compilación incremental que ofrece Xcode, consigue que una vez compilados ReaderKit y Logger, no haga falta volver a compilarlos si solo modificas clases del app bundle. Esto es muy útil si el proyecto evoluciona de forma iterativa, construyendo primero una versión estable de estos frameworks, testearlos por separado y después construyendo la aplicación principal. En este caso, el framework de ReaderKit y Logger se desarrollaron en primera instancia como frameworks separados.

VIPER

La arquitectura seleccionada para la aplicación es VIPER (View, Interactor, Presenter, Entity y Router). Esta arquitectura sigue los principios SOLID anteriormente mencionados. Define un total de cinco componentes:

- **Vista:** Las vistas muestran el contenido que se ha originado en el presenter. Estas vistas tienen que ser lo más simple posible. En general, no se pueden reutilizar entre plataformas, por eso toda la lógica tiene que estar contenida en el presenter.
- **Interactor:** Un interactor define un caso de uso. Contiene la lógica de negocio asociada con el caso de uso. En general actual como intermediario entre la capa de red, la capa de datos y el presenter.
- **Presenter:** El presenter se centra en conseguir que la respuesta que nos devuelve el interactor (que ejecuta un caso de uso) se pueda mostrar. Transforma el modelo de negocio en un modelo de vista que la capa de renderizado puede manejar. El presenter, por tanto, media entre la capa de negocio y la capa de vista. Es una pieza importante ya que es el punto a partir del cual las capas se pueden reutilizar entre diferentes plataformas (a diferencia de las vistas que son específicas por plataforma).
- **Entity:** Las entidades son los modelos manipulados por los interactors. Estos objetos nunca llegan a la capa de presentación.
- **Router:** Se centra en el flujo entre vistas. Cuando se quiere presentar una vista, el router es el responsable de hacer que aparezca en pantalla.

Se puede concluir, que las vistas y los router son específicas de cada plataforma. Esto significa que para poder conseguir desacoplamiento y reutilización del código, es necesario que estas dos capas tengan sus propias interfaces (de forma que los presenters dependan de esta abstracción en lugar de depender en la implementación concreta). Por tanto, si se quisiese reutilizar un proyecto de OS X en iOS, solo sería necesario implementar las interfaces de vistas y wireframes que la plataforma requiera. Los presenters dependerán de las interfaces de vista y router.

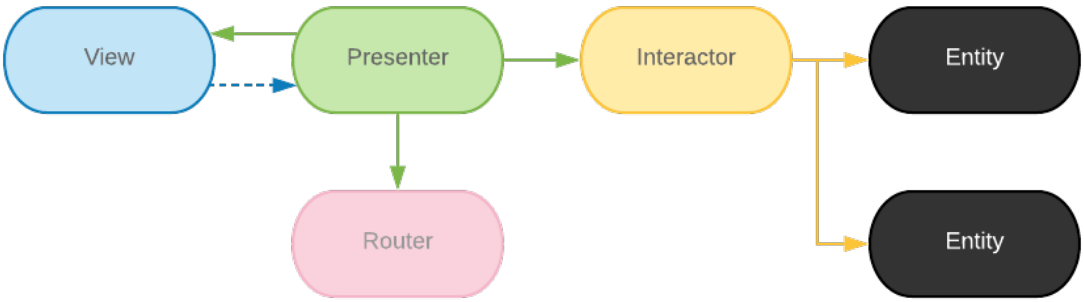


Figura 4.4: Componentes principales de la arquitectura VIPER

Plugins

A pesar de utilizar la arquitectura VIPER, la complejidad de la comunicación y carga del visor PDF ha hecho necesario utilizar un sistema de plugins. Este sistema permite que, concretamente el visor PDF, sea más modular y potente, consiguiendo bastante extensibilidad de cara al desarrollador.

Para cargar un PDF sencillamente es necesario un path hasta el archivo y un array de plugins. Estos plugins encapsulan funcionalidades dentro del visor, de forma que si se introdujese un array vacío el visor renderizaría el PDF en una vista plana. Todos los componentes que hay por encima del PDF son plugins: la barra de progreso, la label de páginas, las barras superior e inferior e incluso el manejo de los gestos que realiza el usuario es un plugin.

Estos plugins se cargan de forma síncrona con la renderización del PDF. Además, un plugin puede tener una serie de plugins hijos que se cargan (en serie) antes de la carga del plugin padre.

La principal ventaja es que, en primera instancia, el visor se convierte en un objeto muy modularizable. Existe al capacidad de deshabitar o habilitar features de forma muy sencilla. Por otro lado, el código queda mucho más encapsulado en componentes más sencillos con contexto único.

Finalmente, expone una mayor potencia de cara al exterior, donde los desarrolladores que hacen uso de ReaderKit tienen la habilidad de añadir o eliminar features del visor,

incluso creando sus propios plugins.

El caso del visor es un caso excepcional, que debido a su complejidad requería de un trato especial de sus componentes.

En la figura 4.5 se puede ver una pequeña jerarquía de la estructura de plugins que existe en el proyecto. Más concretamente, es interesate comprobar como el plugin que gestiona las barras del visor tiene a su vez plugins hijos. Estos se cargan primero y son opcionales. De esta forma, si al crear el visor no le proporcionas el ToCPlugin, no se renderizará la tabla de contenidos. De igual forma, si no se inyecta el plugin de gestos, la aplicación no sabrá como gestionarlos. Es una arquitectura basada en capabilities, de una forma que es muy fácil desde el exterior añadir nuevas funcionalidades de manera muy sencilla.

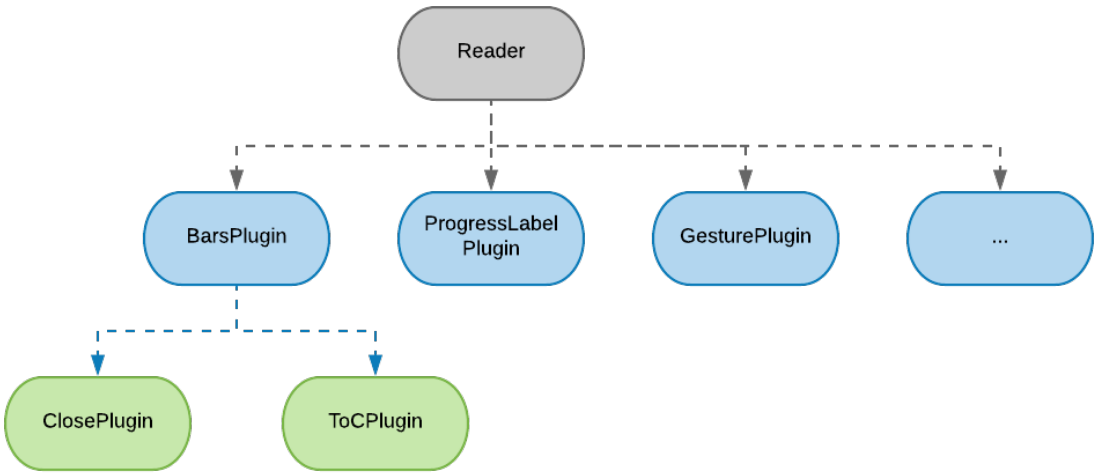


Figura 4.5: Jerarquía de plugins

Estos plugins encapsulan todo lo necesario para poder gestionar todas sus funcionalidades. Todas las dependencias les vienen dadas en su método de carga. Del mismo modo que para VIPER, todas aquellas capas o dependencias que dependan de una implementación concreta por cliente, se encuentran detrás de una interface.

Programación funcional Reactiva

Existen muchas explicaciones y definiciones sobre la programación reactiva en internet, aunque no todas ellas son correctas. Algunas son muy genéricas y teóricas. Otras son demasiado canónicas y no aptas para principiantes. La terminología acaba siendo pesada y confusa.

La programación reactiva es programación asíncrona basada en streams de datos.

Esto no es nada nuevo. En 2010 java introduce en sus librerías el patrón Observer sobre el que se basa este tipo de programación. Los eventos como pueden ser los clicks o los taps son en realidad un stream de eventos asíncronos, al que te puedes suscribir y aplicar 'side effects'. La programación reactiva parte de este concepto añadiendo mucho más. Es posible crear eventos de stream de cualquier cosa, no solo de taps o clicks. Variables, eventos del usuario, propiedades, caches, requests e incluso estructuras de datos. Puedes crear un evento de cualquier cosa y ser capar de escuchar los cambios que se producen sobre ellos, reaccionando a corde.

Sobre el concepto principal, se sublevan una serie de ideas y herramientas que te permiten trabajar con estos streams. De forma que se pueden combinar, filtrar o mapear para crear nuevos. Es aquí donde entra la magia funcional de la que hemos hablado antes. Un stream se puede utilizar como input para crear otro. Incluso múltiples streams se pueden combinar como inputs de otro. Se pueden mergear streams. Se pueden filtrar streams para reducir la cantidad de eventos que recibe, y quedarte solo con los que te interesan. Las posibilidades son muy amplias.

Teniendo en cuenta lo central que es el concepto de stream, es importante centrarse un poco en ellos.

En la programación funcional, los cálculos se ven como funciones que transforman una serie de entradas en una serie de salidas. Estas funciones tienen lo que se llama transparencia referencial. Esto significa que siempre devuelven las mismas salidas para las mismas entradas (no pueden depender de forma interna de un estado externo). Su origen fue motivado por las necesidades de la inteligencia artificial, los cálculos simbólicos y los procesadores de lenguaje.

Este paradigma esta ganando cada vez mas adeptos en las aplicaciones móviles. La principal razón es que actualmente las aplicaciones son mucho más interactivas. Se ha

avanzado bastante con respecto a las webs de hace unos años donde en la mayoría de los casos solo había formularios y elementos visuales simples. Actualmente las apps tienen una gran cantidad de eventos en "tiempo real" de todo tipo que abren un abanico muy amplio de interacción para el usuario.

La programación reactiva trata sobre programar de forma asíncrona con streams de datos. Estos streams se puede crear de casi cualquier cosa: estructuras de datos, eventos del usuario, propiedades, notificaciones, etc. Al rededor de todos estos streams hay una gran cantidad de funciones que se pueden encadenar, para transformar, mapear, filtrar o encapsular datos. De forma que incluso los streams se pueden usar como imputs para otros streams.

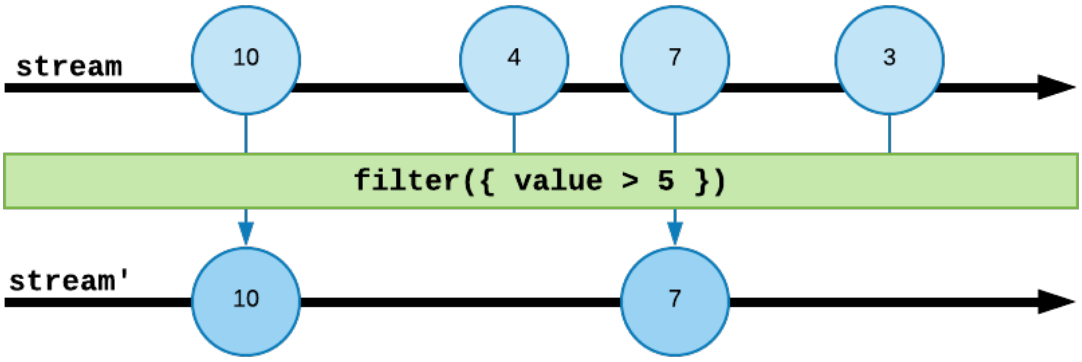


Figura 4.6: Ejemplo de filtrado de un stream

Quartz

CoreGraphics también conocido como Quartz 2D, es un motor avanzado de dibujo bidimensional disponible para el desarrollo de aplicaciones iOS, macOS y watchOS. Quartz 2D proporciona un renderizado 2D ligero y de bajo nivel con una fidelidad de salida que no se ve determinada por la dimensión de la pantalla. Es independiente de la resolución y el dispositivo.

La API de Quartz es fácil de usar y proporciona acceso a funciones muy potentes como pintado de transparencias, pintado basado en trazas, renderizado offscreen, manejo de colores y sobretodo creación, parseo y pintado de PDFs.

Los documentos PDF almacenan gráficos vectoriales independientes de la resolución, texto e imágenes como una serie de comandos escritos en un lenguaje de programación compacto. Quartz crea, para todas las aplicaciones documentos de alta fidelidad que conservan las operaciones de dibujado.

El PDF resultante puede optimizarse para un uso específico (como una impresora en particular o para la web) por otras partes del sistema o por productos de terceros. Los documentos PDF generados por Quartz se ven correctamente en Vista previa y Acrobat, dos programas de visualización PDF.

4.3. Justificación de herramientas y lenguajes elegidos

En esta sección se explica las herramientas software utilizadas para el desarrollo además de los lenguajes de programación elegidos, argumentando la elección y las características más importantes

4.3.1. Herramientas software

Git [1]

Git es un software de control de versiones. Es un sistema descentralizado pero que se puede comparar con sistemas centralizados como Subversion (SVN) o Team Foundation Server (TFS).

En comparación con otros sistemas, Git permite el trabajo en ramas. Esto permite un desarrollo en paralelo entre varios desarrolladores. Es una poderosa herramienta que permite que varios integrantes del equipo trabajen al mismo tiempo en distintas partes del proyecto.

Git tiene flujos de trabajo que lo hacen particularmente flexible y que puede ser usado desde un desarrollador independiente hasta equipos ágiles.

Es descentralizado, siendo mas robusto a fallos y a ataques poseyendo varios repositorios. Internamente, el código se encripta para que no pueda ser modificado por

[1] https://git-scm.com/doc

terceros.

A continuación se explica el uso de git y de sus sentencias.

Primero se debe de crear un repositorio con git init o se clona (en caso de ya existir) con git clone. Git puede ser usado con diferentes protocolos como pueden ser HTTPS, SSH, etc. Dentro del repositorio, los ficheros pueden tener dos estados: rastreado o no rastreado. Rastreado son aquellos ficheros que pueden ser modificados o alterados. Los no rastreados son el resto de ficheros.

Una vez que los ficheros han sido modificados, se debe realizar un commit (git commit -m "Mensaje del commit") con todos los ficheros alterados que se quieran incluir. El commit implica un guardado momentáneo de la situación en la que están los archivos incluidos en ese paquete. Los commit se realizan de manera local y se almacenan en el repositorio local.

En el momento de poseer uno o varios commit, el usuario puede realizar un push al repositorio (git push). Esta sentencia confirma los cambios y sube el código al repositorio remoto.

El uso de GIT de manera remota, hace que el trabajo pueda ser realizado en distintos puntos geográficos. Además permite comprobar el historial de los archivos, ayudando a no perder código desechado por error.

Git-Flow

Es un modelo Git que establece una serie de reglas a seguir para un desarrollo limpio y organizado. Este concepto se basa en tener dos ramas principales : Master y develop.

- **Develop:** Rama que posee el código para la siguiente versión.
- **Master:** Rama la cual posee un código preparado para subir a producción

Además de estas ramas, se tienen ramas auxiliares:

- **Feature:** Se originan a partir de develop y son usadas para nuevas funcionalidades.

- **Release:** Se originan a partir de develop y son usadas para preparar el siguiente código en producción.
- **Hotfix:** Se originan a partir de master y son usadas para corregir errores en producción.

Se decide seguir este modelo para tener una estructura de ramas y un control de versiones.

GitHub

Es un repositorio GIT en la nube en el cual se pueden ejecutar todos los comandos GIT necesarios. La elección de tener dicho repositorio es la de facilitar el trabajo y la comunicación del equipo, sin necesidad de tener el repositorio alojado de manera local, si no accesible desde cualquier punto.

Visual Studio Code [2]

Editor de texto ofrecido por Microsoft. La elección de este editor es que es multi-plataforma y gratuito. Es Open Source, lo que permite que el usuario pueda adquirir plugins que ayuden al desarrollo. Uno de los plugins utilizados en este proyecto, por ejemplo es GIT: plugin que permite visualizar en el editor de texto toda la información relacionada con GIT, como quien editó esa línea de código, cuando, etc.

Xcode [3]

Xcode es un entorno de desarrollo integrado (IDE) desarrollado para macOS que incluye todas las herramientas necesarias para crear aplicaciones de Mac, iPhone, iPad, Apple TV y WatchOS. Xcode prevé a los desarrolladores del ecosistema de Apple un flujo unificado de interfaz de diseño, testing y debugging. Xcode incluye el IDE de Xcode, compiladores de Swift y Objective-C, Instruments (herramienta de análisis), simuladores, el último SDK y algunas otras herramientas.

A pesar de que hay otras opciones gratuitas para el desarrollo de aplicaciones iOS como puede ser AppCode, o algunos plugins para otros IDEs, Xcode lleva la ventaja en

[2] https://code.visualstudio.com/
[3] https://developer.apple.com/xcode/ide/

estabilidad y features soportadas.

Postman [4]

Es una extensión de Chrome (aunque actualmente posee aplicación nativa), que permite el envió de peticiones HTTP REST. Para probar las peticiones ofrecidas por el servidor, se utiliza este cliente que permite visualizar si la respuesta es la adecuada, realizar pruebas de error, organizar las peticiones en carpetas y poder tener distintos entornos, pudiendo cambiar las peticiones entre Desarrollo, Preproducción y Producción.

Swagger [5]

Es un framework para generar documentación sobre una API REST. En el caso de que los integrantes del equipo no compartan el mismo idioma, o en ocasiones no trabajen en la misma localización geográfica, este framework ayuda a crear una documentación con un diseño fácil de entender y que tiene integración real con el servidor, lo que permite poder realizar las peticiones REST de manera verídica.

Trello [6]

Es una plataforma online que ofrece tableros. Con estos tableros se puede realizar una estrategia Kanban y tener un tablero visual online.

LaTeX

Es un sistema de composición tipográfico creado para la edición de los textos. Ofrece una serie de comandos que tras compilar el documento, dan directamente formato, crea relaciones entre distintas partes del documento, permite crear links y referencias dentro del documento al igual que links externos, etc.

Sharelatex[7]

[4] https://www.getpostman.com/
[5] https://swagger.io/
[6] https://trello.com/
[7] https://es.sharelatex.com/

Es un editor online de LaTeX para crear documentos. Dispone de una buena cantidad de plantillas para ahorrar bastante tiempo en la creación de diferentes tipos de documentos. La elección de esta herramienta contra otros editores online como pueden ser Google Drive, fue el uso del sistema LaTeX.

4.3.2. Lenguaje de programación

JavaScript

JavaScript es un lenguaje de programación que surgió por la necesidad de ejecutar código en el propio navegador. El lenguaje nace de la mano del programador Brendan Eich, el cual trabajaba en Netscape y en sus orígenes lo llamó LiveScript. Netscape firmó una alianza con Sun Microsystems y antes de lanzar este lenguaje al publicó decidió cambiar el nombre a JavaScript ya que Java era una palabra en el mundo de la programación mucho más conocida. Microsoft copió la idea y lanzó su propio lenguaje llamándolo JScript. Esto hizo que Netscape decidiese estandarizar el lenguaje en 1997. La primera estandarización se definió como ECMAScript [Egu15].

En el transcurso de los años ECMAScript ha ido evolucionando y creando nuevas estandarizaciones. La última de ella es ECMAScript 6 o ES6 lanzada en 2015.

Tras la creación del estándar, el siguiente gran avance llegó en los navegadores. El navegador era quien debía hacer correr las aplicaciones y sin un motor adecuado, se seguiría perdiendo velocidad en la carga. Fue entonces cuando se empezó una guerra entre compañías para sacar el mejor navegador con el mejor motor. Microsoft invirtió mucho dinero y tiempo para volver a ser pionero e intentar desbancar a Netscape creando su propio motor llamado Trident.

Después de estos sucesos, Netscape empezó a quedar en un segundo plano, y decidió liberar el código fuente de su motor (denominado Gekko) para toda la comunidad software pudiese colaborar. De esta acción surge el actual navegador Firefox, basado en dicho motor junto con muchos otros.

Otra gran empresa, como Apple, también creó su propio motor WebKit y llamó a su navegador Opera.

Sin embargo, hubo una gran compañía que finalmente se alzó y consiguió ponerse líder en el sector. Hacia el año 2008 y con la guerra de los navegadores aún abierta,

google era el buscador preferido por la mayoría de los usuarios. Google pensó que era consecuente ofrecer un navegador propio y quitarse dicha dependencia en el sector. Fue entonces cuando se creó Google Chrome con un motor creado a partir de WebKit. Es aquí cuando JavaScript ya no era un script que venía en un HTML, si no que ahora el navegador lo compilaba al cargarse y se ejecutaba cuando se interactuaba con la página Web. Dicho motor fue tan rápido que se le dominó como V8, haciendo referencia al motor de 8 cilindros en V de los coches.

Tras este paso, se creó NodeJS, que requiere del motor V8 y permite ejecutar JavaScript no solo en un navegador, si no también en la parte servidor, en PC o en cualquier sitio que se soporte el V8 [Egu15].

El uso en este proyecto en la parte servidor de NodeJS hace que el lenguaje que se requiere sea JavaScript.

La sintaxis de JavaScript es muy simple y se asemeja con otros lenguajes de programación como Java o Python.

```
function myFunction (parameters) {
        // code to be executed
}
```

Lo más destacado en ES6 que se utiliza en este proyecto, es por ejemplo el uso de la variable let. Esta palabra reservada nos permite declarar variables locales a un bloque, es decir, podemos declarar únicamente variables definidas dentro de una sentencia if o una expresión. También pasa a soportar constantes usando la palabra reservada const. Otra característica es el valor por defecto en los parámetros de una función.

```
function addNum(x = 1, y = 1) {
        return x + y ;
}
```

En el anterior ejemplo podemos ver como la función addNum en caso de no pasar valores en los parámetros siempre devolverá 2 ya que por defecto los valores de los parámetros serán 1 y 1 para x e y.

Swift

Actualmente, Apple soporta varios lenguajes de programación en sus plataformas: C, C++, Objective-C o Swift. Sin embargo, los dos lenguajes principalmente usados para el desarrollo de aplicaciones son Objective-C y Swift. El resto de lenguajes sirven de apoyo para proporcionar soporte de librerías (con algunas excepciones).

Swift y Objective-C son lenguajes muy diferentes.

- **Objective-C** es un superconjunto de C, orientado a objetos, con mucho rodaje (creado en 1980) y basado en mensajes. Uno de los puntos fuertes de este lenguaje es su potencia de ejecución de código en runtime.

- **Swift** es un lenguaje más moderno, presentado por Apple en 2014 en la WWDC. Chris Lattner comenzó su desarrollo en el 2010, eventualmente obtuvo colaboración de otros programadores. Swift se benefició de la experiencia de muchos lenguajes tomando ideas de Objective-C, Haskell, Rust, Ruby, Python, C#, CLU entre otros. De ahí que sea considerado un lenguaje multiparadigma: Orientado a protocolos, objetos, funcional, programación imperativa.

A pesar de que Objective-C ha sido la base del desarrollo de iOS hasta la fecha, no se puede comparar a Swift en términos de rendimiento, seguridad o mantenibilidad del código.

Las principales razones por las que se ha utilizado Swift son las siguientes:

- **Más fácil de leer.** Swift tiene un gran parecido con el inglés más básico, caracteriza común con los lenguajes de programación más modernos. Además, reduce la complejidad eliminando los puntos y comas para terminar las lineas, o los paréntesis para las expresiones condicionales en las instrucciones if / else. Finalmente, tiene una lectura más liviana al eliminar los corchetes tan característicos de Objective-C.

- **Incrementa la productividad del desarrollador.** Swift ayuda al desarrollador a conseguir un código limpio. La fuerza de Objective-C radica en la habilidad de exponer interfaces, pero esto a su vez se convierte en una desventaja cuando se desea escribir código.

- **Compatible con Objective-C y Cocoa touch.** Apple ha dedicado mucho tiempo y esfuerzo a conseguir una compatibilidad casi total entre Swift y Objective-C. Algo razonable, teniendo en cuenta la cantidad de trabajo realizado por la comunidad alrededor de Objective-C.

- **Compilador más rápido e inteligente.** Swift utiliza LLVM (Low Level Virtual Machine). Un mejor compilador supone une mejor experiencia para el desarrollador.

- **Más seguro.** Con la ausencia de punteros, Swift es mucho más seguro de usar para los programadores. El problema con los punteros es que crean vulnerabilidad en la seguridad del código. También actúan como la barrera para corregir errores. Sin embargo, en Swift es más estricto obligándote a definir cuando una propiedad puede ser o no nula. De esta forma, si accedes a una propiedad con valor nulo que no debería tenerlo, el programa crashea.

- **Mejor manejo de memoria.** Los leaks de memoria reducen la memoria disponible para ejecutar una cierta aplicación. Swift soporte ARC (Automatic Reference Counting) para todas las APIs. Sin embargo, Cocoa Touch falla en algunos contextos como core graphics, donde ARC no se soporta.

4.4. Requisitos funcionales y no funcionales

En esta sección se listan los requisitos funcionales del proyecto. La sintaxis usada para describir los requisitos son R para requisito, B para describir backend y M para describir app móvil con el uso de dígitos numéricos para hacer único cada requisito.

4.4.1. Requisitos no funcionales

Servidor

Código	**RB01**
Título	Escalabilidad
Descripción	El backend debe tener la posibilidad de evolucionar en un futuro sin perder calidad y pudiendo mantener.

Tabla 4.1: Requisito RB01

Código	**RB02**
Título	Mantenibilidad
Descripción	El backend debe poder mantenerse (Sistema organizado en módulos)

Tabla 4.2: Requisito RB02

Código	**RB03**
Título	Rapidez
Descripción	El backend debe retornar la información lo más rápido posible

Tabla 4.3: Requisito RB03

Código	**RB04**
Título	Seguridad
Descripción	Los datos personales deben estar encintados en la base de datos

Tabla 4.4: Requisito RB04

Código	**RB05**
Título	Seguridad
Descripción	Todas las peticiones de la API deben tener seguridad mediante api key

Tabla 4.5: Requisito RB05

Código	**RB06**
Título	Testeabilidad
Descripción	El código debe poder ser testeado

Tabla 4.6: Requisito RB06

Cliente

Código	**RM01**
Título	Seguridad
Descripción	No se podrá acceder a una cuenta que no sea la tuya.

Tabla 4.7: Requisito RM01

Código	**RM02**
Título	Testeabilidad
Descripción	El código debe poder ser testeado

Tabla 4.8: Requisito RM02

Código	**RB03**
Título	Mantenibilidad
Descripción	Se podrán realizar evolutivos sobre el producto base de una forma sencilla

Tabla 4.9: Requisito RB03

Código	**RM04**
Título	Portabilidad
Descripción	El código podrá ser usado, en la medida de lo posible, en otras plataformas

Tabla 4.10: Requisito RM04

Código	**RM05**
Título	Usabilidad
Descripción	Será sencillo para los usuarios manejar la aplicación

Tabla 4.11: Requisito RM05

4.4.2. Requisitos funcionales

Servidor

Código	**RB07**
Título	Registro de usuario
Descripción	Si un usuario no posee nick y password en la base de datos se debe de poder registrar en el sistema

Tabla 4.12: Requisito RB07

Código	**RB08**
Título	Login de usuario
Descripción	Si el usuario está registrado previamente en el sistema, debe poder hacer login y acceder

Tabla 4.13: Requisito RB08

Código	**RB09**
Título	Devolver token de seguridad tras hacer login
Descripción	Para la seguridad de las llamadas, tras hacer login la petición debe devolver un token de seguridad

Tabla 4.14: Requisito RB09

Código	**RB10**
Título	Listado de PDF's para un usuario
Descripción	Petición HTTP que devuelve el listado de PDF's que tiene el usuario asociado

Tabla 4.15: Requisito RB10

Código	**RB11**
Título	Detalle de un PDF
Descripción	Petición HTTP que devuelve el detalle de un PDF

Tabla 4.16: Requisito RB11

Código	**RB12**
Título	Crear un PDF
Descripción	El usuario debe poder guardar PDF's. El sistema debe guardarlo en la base de datos y crear la relación con el usuario

Tabla 4.17: Requisito RB12

Código	**RB13**
Título	Borrar un PDF
Descripción	El usuario puede borrar PDF's. El sistema debe borrar la relación y eliminarlo de base de datos

Tabla 4.18: Requisito RB13

Código	**RB14**
Título	Actualizar un PDF
Descripción	El usuario puede editar la información del PDF. El sistema debe actualizarlo en base de datos

Tabla 4.19: Requisito RB14

Código	**RM07**
Título	Zoom
Descripción	Se podrá hacer zoom sobre las páginas.

Tabla 4.20: Requisito RM07

Cliente

Código	**RM08**
Título	TOC
Descripción	Se podrá acceder a la tabla de contenidos del documento PDF

Tabla 4.21: Requisito RM08

Código	**RM09**
Título	Links
Descripción	Se podrán tapear los links para navegar tanto de forma interna como externa

Tabla 4.22: Requisito RM09

Código	**RM10**
Título	Carga asíncrona
Descripción	La carga de páginas será asíncrona para que sea más liviana

Tabla 4.23: Requisito RM10

Código	**RM11**
Título	Volver atrás
Descripción	Se podrá volver a la página anterior si saltas de página a través de de los links o de la tabla de contenidos

Tabla 4.24: Requisito RM11

Código	**RM12**
Título	Visualización progreso
Descripción	Se podrá visualizar el progreso actual de lectura

Tabla 4.25: Requisito RM12

Código	**RM13**
Título	Salto de página
Descripción	Se podrá saltar de página a través de la barra de progreso

Tabla 4.26: Requisito RM13

4.5. Metodología de desarrollo

En esta sección se describe el tipo de metodología que se ha elegido para el desarrollo como información y explicación de la misma.

4.5.1. Metodologías Ágiles

Las metodologías ágiles son un método de trabajo que permite adaptar la forma de trabajar a las condiciones del proyecto. Esto ofrece flexibilidad e inmediata solución a los problemas que surgen sobre el desarrollo [Mar14].

Las metodologías ágiles se caracterizan por la implicación del equipo a lo largo de todo el proceso del proyecto. Todos los miembros del equipo deben poder conocer el estado del proyecto en cualquier momento. De esta manera el compromiso es más grande ya que todos los integrantes conocen los tiempos de desarrollo y los problemas que van surgiendo, pudiendo ayudarse unos a otros.

En los procesos ágiles, la dinámica de trabajo es ofrecer al cliente versiones funcionales del producto tras pequeños periodos de tiempo denominados *sprint*. Este periodo de tiempo es acordado entre cliente y equipo antes de comenzar el desarrollo.

Al ofrecer al cliente versiones funcionales, esto permite conocer la satisfacción del cliente con el producto, y en caso de errores o cambios, abordarlos lo más rápidamente posible evitando cambios una vez está creado todo el producto, y mejorando la calidad del servicio.

4.5.2. Tipos de Metodologías Ágiles

SCRUM

La estrategia en este tipo de metodología está orientada a gestionar los procesos antes del desarrollo para no tener que eliminarlos a posteriori. Esto se hace mediante una serie de eventos a lo largo del proceso.

Eventos

En la metodología Scrum hay una serie de eventos que ocurren a lo largo del desarrollo del producto [Jef16]

- **Sprint.** Es el bloque de tiempo definido con el cliente para ofrecer las entregas funcionales del producto. El tiempo está determinado en puntos de esfuerzo y suele comprender entre dos semanas y un mes. Cuando el tiempo es mayor a un mes, se debe separar en funcionalidades más pequeñas y asequibles. En un sprint se encuentran todas las fases y reuniones del desarrollo (Daily, Planning, Review, Retrospective). Durante el sprint no se realizan cambios que afecten a la entrega del producto.

- **Daily.** Son reuniones diarias en las que todos los integrantes del equipo explican brevemente cual fue su trabajo en el día anterior y cual es el propósito de trabajo en el día actual. También se menciona si hay procesos que pueden bloquear a algún integrante o si se han encontrado problemas. El objetivo principal de estas reuniones es de conocer la situación actual del proyecto, comprobando que los tiempos son acertados y pudiendo anticiparse a los problemas o bloqueos que han ido surgiendo.

- **Planning.** Es la planificación del sprint y se lleva acabo con la participación de todo el equipo. En esta reunión se muestran los requisitos que se van a desarrollar durante el sprint, además de definir quién es el encargado de desarrollar dichos requisitos.

- **Review.** Una vez finalizado el desarrollo del sprint, se realiza una Review. Es una reunión de equipo para inspeccionar el trabajo realizado y en caso de que algún punto no haya podido terminarse, explicar que ha ocurrido. El equipo comenta lo que se hizo a lo largo del sprint por parte de cada integrante y que cosas se podrían optimizar para los sprint's futuros. También se hace una pequeña demostración del trabajo donde se ofrece una visión de la evolución del producto.

- **Retrospective.** Es una reunión donde el equipo se inspecciona a si mismo. Se realiza después de la Review y antes de la planificación del siguiente sprint. En

esta reunión el principal proceso es identificar que elementos salieron bien y cuales fallaron en el sprint anterior, y si es posible mejorarlo y como. Cualquier integrante del equipo puede aportar ideas y soluciones. Con esta reunión se intenta mejorar el ambiente del equipo, ofreciendo más conformidad y calidad al ámbito de trabajo.

Figuras en el equipo

En el equipo de trabajo hay una serie de figuras.

- **Equipo de desarrollo** El equipo de desarrollo son los profesionales que van a realizar el trabajo. El equipo debe tener una serie de características:

  - Autoorganización. Nadie indica al equipo de desarrollo como convertir los elementos de la lista en incrementos funcionales.
  - Multifuncionales. Como equipo se debe poseer todas las habilidades necesarias para crear un incremento en el producto.
  - Igualdad. Todos los integrantes son desarrolladores, independientemente del trabajo que realiza cada persona.

  Los tamaños de los equipos varían entre cuatro y diez personas, dependiendo de la necesidad del proyecto. Son equipos pequeños para favorecer la comunicación, y lo suficientemente grandes para asegurarse de que el producto se realiza adecuadamente y se pueden cubrir todas las necesidades de desarrollo.

- **Scrum Master.** Responsable de asegurar que Scrum se entienda y se adopte. Es un líder que está al servicio del equipo para ayudar. Además es el encargado de las relaciones entre las personas externas y el equipo.

- **Product Owner.** Es el dueño del producto. Debe ser una única persona. Es el responsable de maximizar el valor del producto y el trabajo del equipo. Debe ser la única persona encargada de organizar las listas de trabajo y de gestionarlas. En la planificación, es el encargado de explicar cada elemento que se va a realizar en el siguiente sprint, asegurándose que es entendido por todos los integrantes del equipo.

KANBAN

Es otro tipo de metodología ágil cuyo objetivo es gestionar de manera general cómo se van realizando las tareas. La principal característica es el fácil uso que tiene. Es una técnica muy visual permitiendo ver rápidamente el estado del proyecto y de las tareas. Se diferencia de otras metodologías en varios puntos [Gil13]:

- Calidad. Las tareas deben ser realizadas a la perfección. No se basa en la rapidez, si no en que la calidad sea la adecuada.
- YAGNI. Se debe realizar lo justo y lo necesario, pero de manera correcta.
- Mejora continua. Se pueden ir añadiendo tareas de mejora que se irán abordando al terminar otra tarea.
- Flexibilidad. El desarrollador elige que tarea ir cumplimentando, eligiéndola de la lista de tareas.

Para seguir una estrategia Kanban, lo primero que se debe realizar es un tablero. Este tablero debe ser visible y accesible para todos los participantes. Se definen una serie de columnas que indican el estado de cada tarea. Las columnas habituales suelen ser TO DO (tarea que se va a realizar), DOING (tarea en proceso de desarrollo) y DONE (tarea realizada). Hay casos en los que se suele añadir más columnas. Al ser una metodología que no posee unas reglas específicas, cada equipo puede crear las columnas que considere oportunas siempre y cuando el flujo de trabajo quede claro.

Kanban se basa en el principio de desarrollo incremental, dividiendo el trabajo en distintas partes. Cada una de las partes o tareas, se escribe en un post-it, el cual contiene la información básica de la tarea, y se pone en el tablero. Con la evolución de las tecnologías y el desarrollo de apps, hoy en día podemos encontrar varios tableros virtuales que nos ofrecen todas las funciones necesarias para llevar a cabo Kanban.

El fundamento básico de Kanban es no empezar una tarea hasta que otra no esté acabada. De esta manera se prioriza el trabajo en curso antes de empezar tareas nuevas. Este principio puede suponer problemas si las tareas no están bien priorizadas o definidas, ya que una mala especificación puede suponer dependencias o bloqueos en el tablero.

4.5.3. Metodología Ágil: Kanban en el proyecto

Para este proyecto se ha elegido el uso de metodologías ágiles. La elección del uso en contra posición al uso de metodologías pesadas, es sobre todo por la facilidad que ofrecen de trabajar de manera más libre, pudiendo elegir que tareas realizar y cuando, pudiendo mover las fechas de entrega a tiempos en los que el equipo consideraba más oportunas, y todo ello teniendo a la vez una organización interna controlada.

El equipo consta de dos integrantes. En las definiciones para metodologías ágiles se podría decir que es un equipo pequeño, que incluso no llega a tener el número mínimo de integrantes para un equipo, el cual se suele poner siempre en tres. Sin embargo, la ausencia de algunos roles no ha ocasionado ningún problema para efectuar dicha metodología.

El proyecto es el desarrollo de una app móvil, debiendo desarrollar la implementación desde el inicio. Esto supone un análisis previo de requisitos, diagramas necesarios para entender el funcionamiento de la app, un modelo del diseño, y por último el desarrollo de la parte frontend y backend.

Este desarrollo puede llevar a cabos cambios en la implementación de la app, lo que supondría una reestructuración de todos los modelos y diagramas, por ello se ha elegido usar Kanban, para poder elegir que tareas realizar primero y llegar todas de manera organizada y secuencial, acabando una tarea antes de empezar la siguiente. Se han establecido unos requisitos mínimos, y en caso de acabarlos se tienen planeados futuros evolutivos, con estimaciones de tiempo de las tareas, para que todas se puedan realizar dentro del plazo acordado.

A continuación, se muestra el tablero en distintas fases del proyecto.

Figura 4.7: Tablero inicio Kanban

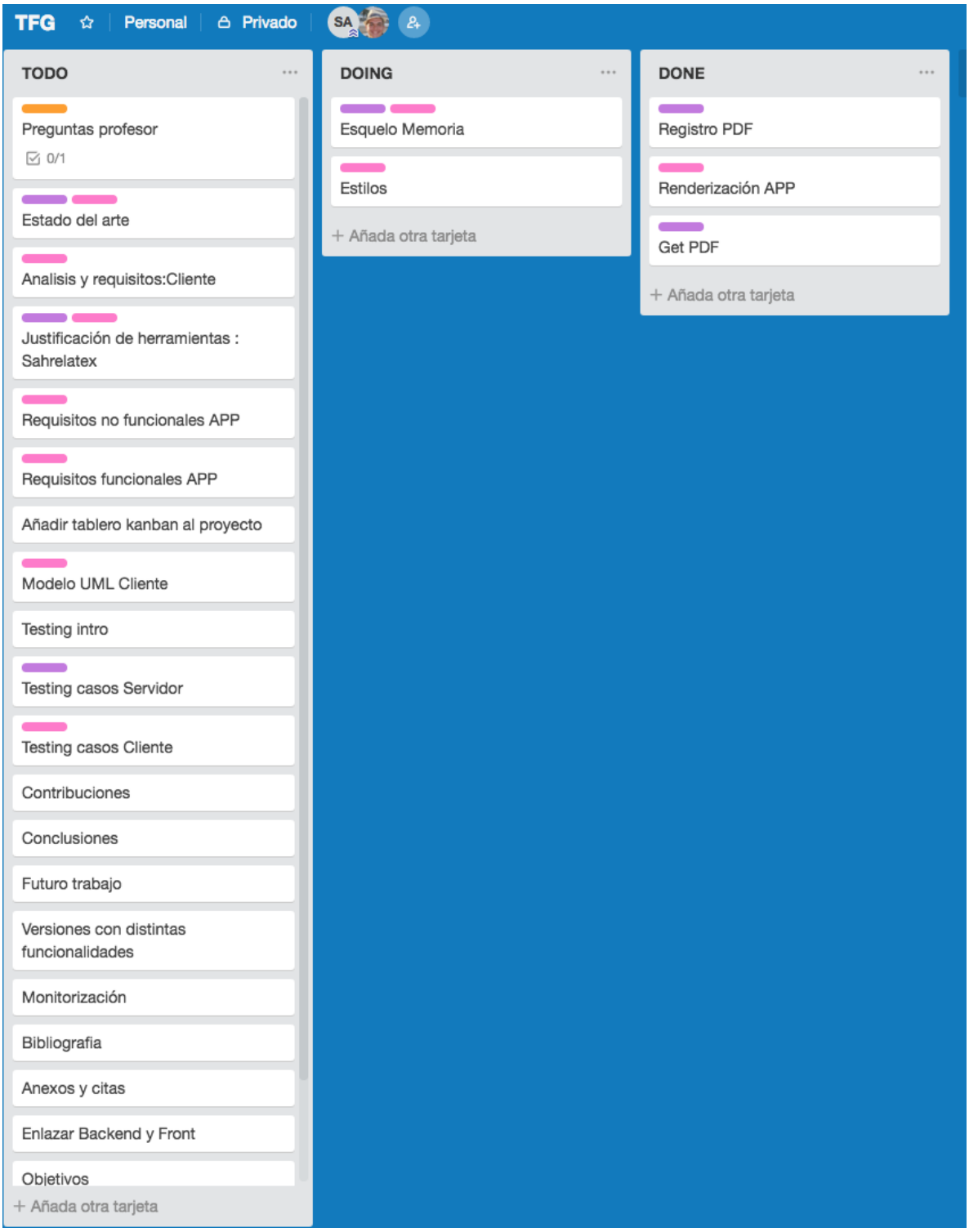


Figura 4.8: Tablero después de prototipo app

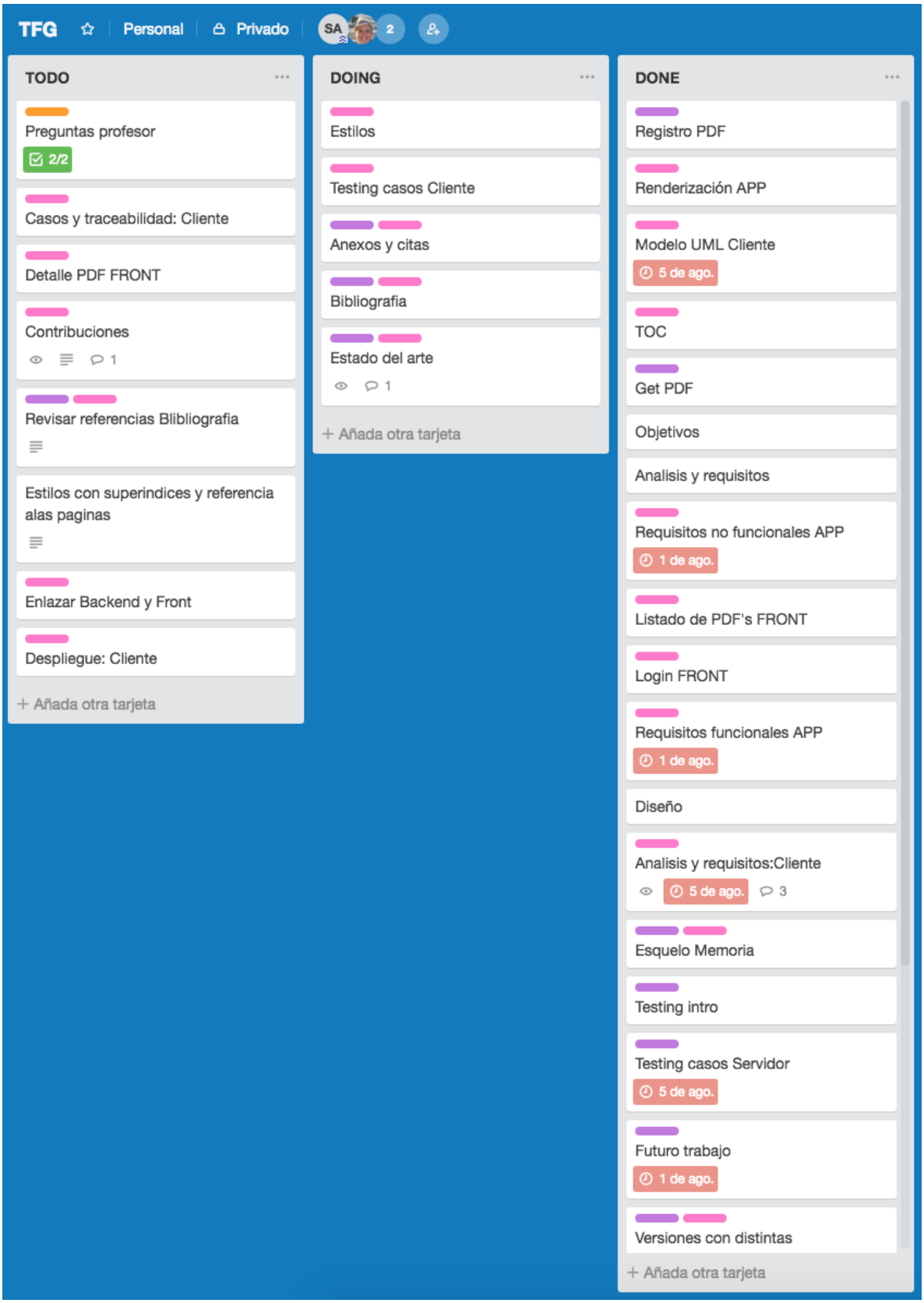


Figura 4.9: Tablero con memoria en desarrollo

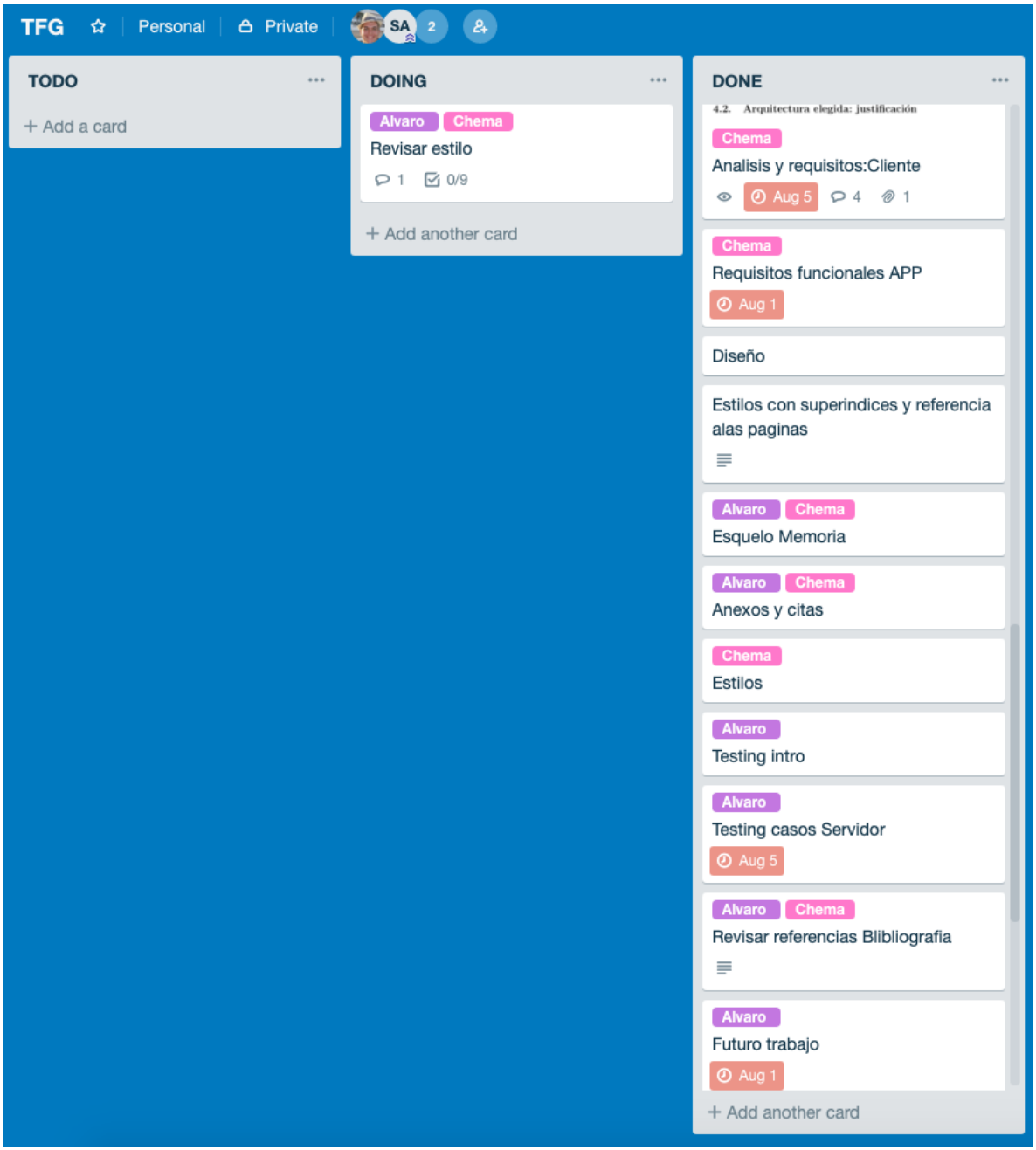


Figura 4.10: Tablero con memoria final

Capítulo 5

Diseño e implementación

5.1. Modelo UML

En esta sección se muestras los diagramas UML más relevantes para entender la construcción del sistema y como funcionan los servicios ofrecidos. Está dividida en dos secciones: servidor y cliente

5.1.1. Servidor

En esta sección se detallan los diagramas necesarios para comprender la composición del backend y de los procesos de Registro, Login, obtención del listado de PDF para el usuario, Detalle de un PDF y Guardado de un nuevo PDF, mediante un modelo de diagrama de secuencias.

Diagrama

Como ya se ha explicado en el apartado de arquitectura, para el backend se sigue un modelo hexagonal, eso quiere decir que la parte del Servidor de datos y la parte cliente quedan separadas de la parte servidor. La comunicación entre las tres partes se hace mediante connotación tipo JSON.

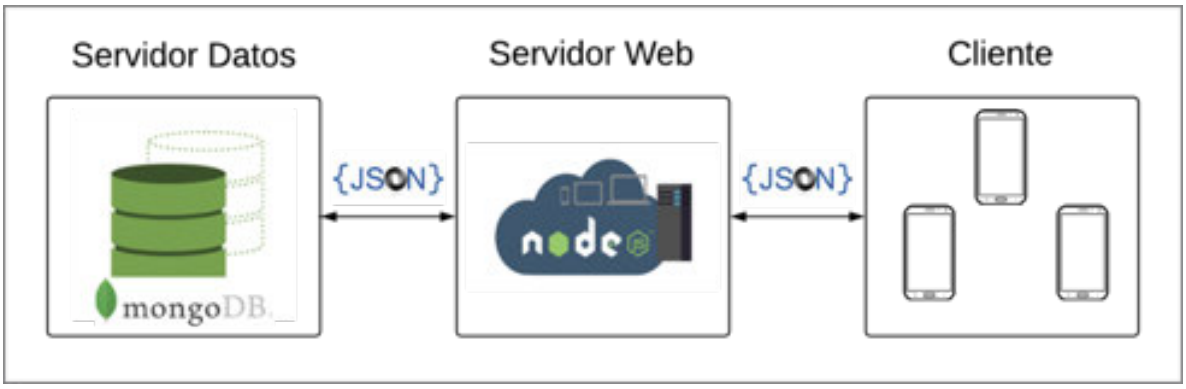


Figura 5.1: Diagrama aplicación

Registro

Para el registro, el cliente solicita al sistema el registro introduciendo un nick y un password. El sistema solicita información a la base de datos sobre ese nick, y en caso de ya estar usado retorna un error. Este proceso se realiza hasta que el cliente introduce un nick y un password correcto, en ese caso, el sistema solicita el guardado del usuario y retorna una respuesta satisfactoria.

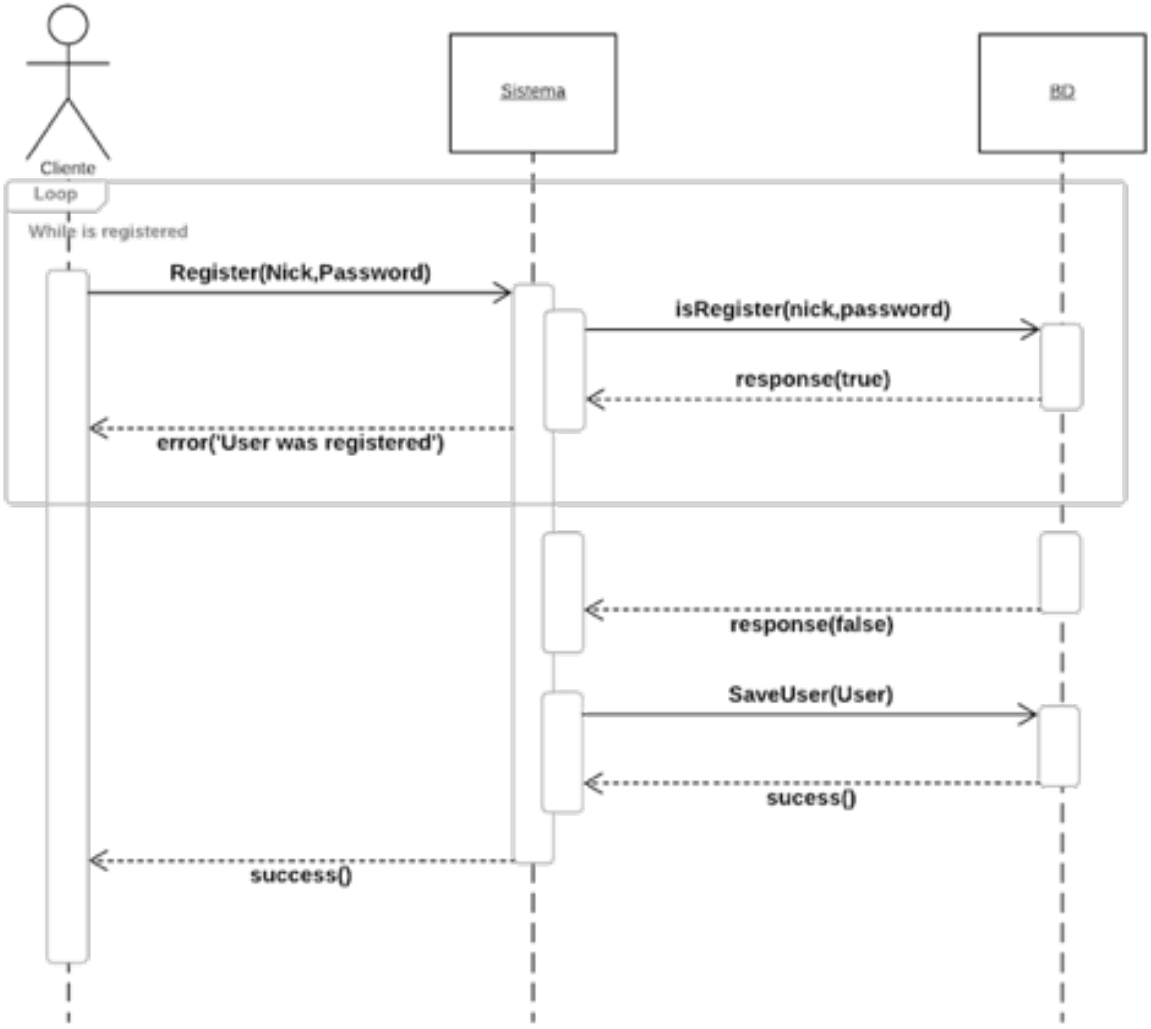


Figura 5.2: Diagrama de secuencia del registro

Login

Para el login, el Cliente introduce su nick y su password. El sistema solicita la información del cliente a la base de datos, y genera un token de sesión con esa información y una clave secreta. Response con una respuesta satisfactoria y un token de sesión.

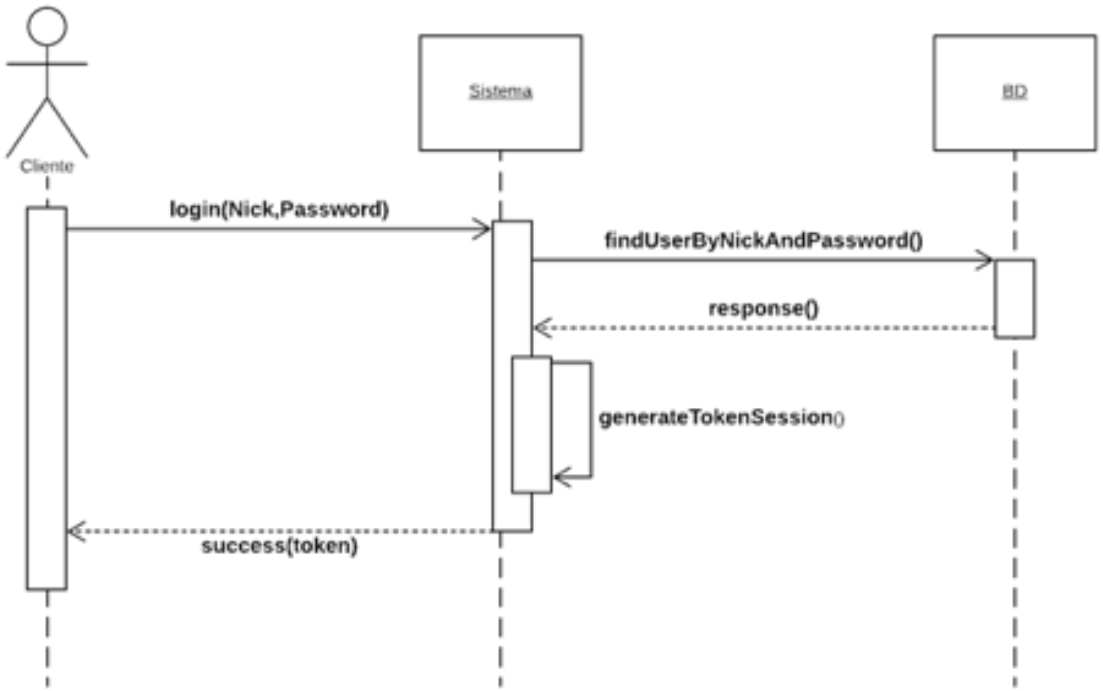


Figura 5.3: Diagrama de secuencia del login

Obtener listado PDF

Para obtener el listado del PDF el cliente solicita al sistema la información enviando por parámetro un token de sesión. Mientras el token no sea correcto, el sistema devolverá siempre un error. Cuando el token es correcto, el sistema solicita a la base de datos el listado de PDF asociados a un usuario y retorna la información.

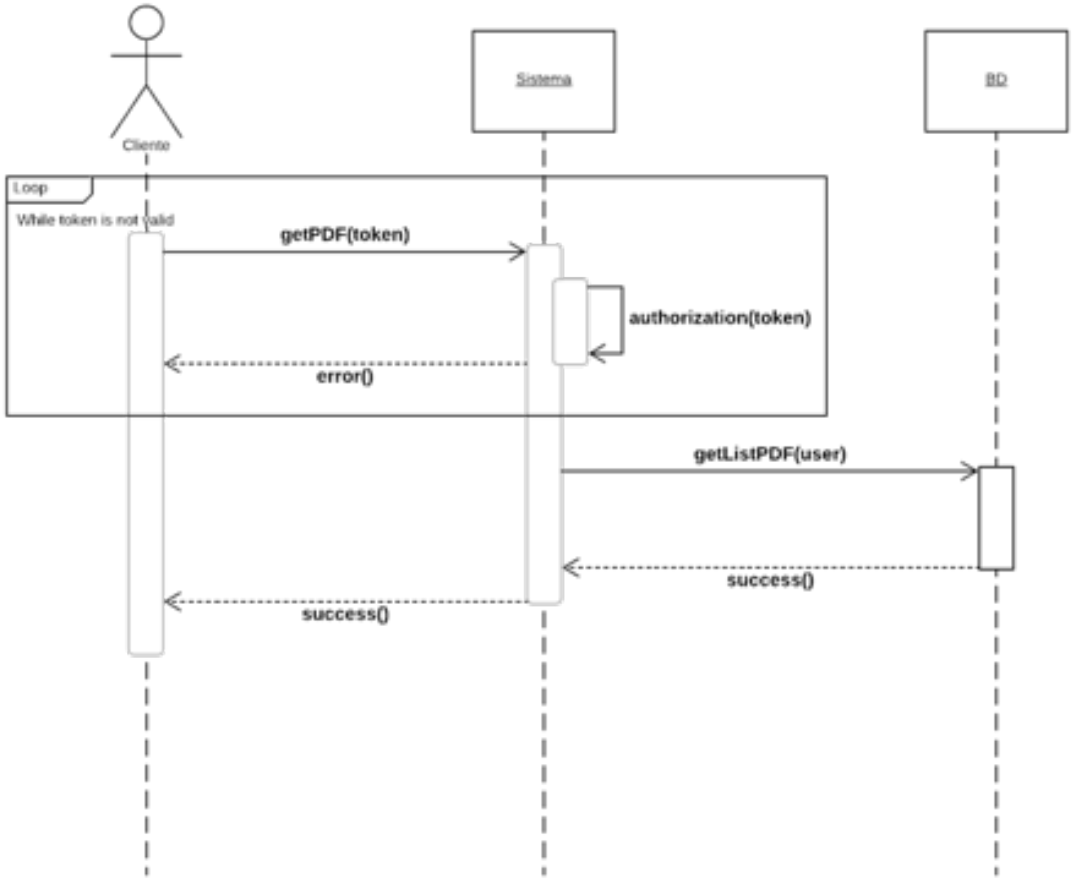


Figura 5.4: Diagrama de secuencia del listado de PDFs

Obtener detalle de PDF

Para obtener el detalle del PDF el cliente solicita al sistema la información enviando por parámetro un token de sesión. Mientras el token no sea correcto, el sistema devolverá siempre un error. Cuando el token es correcto, el sistema solicita a la base de datos el listado de PDF asociados a un usuario y correspondiente a un nombre de fichero y retorna la información en la respuesta de la solicitud.

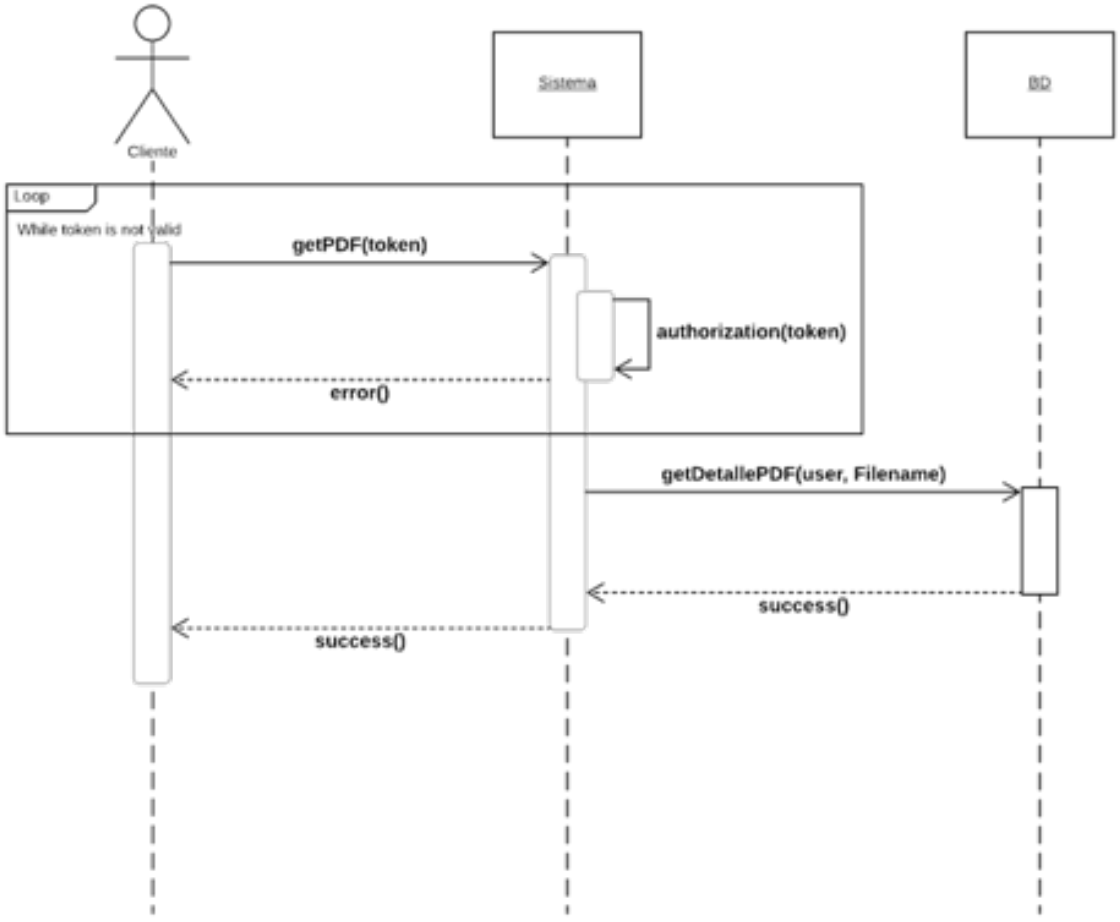


Figura 5.5: Diagrama de secuencia del detalle de PDFs

Guardar PDF

Para guardar el PDF el cliente solicita al sistema la información enviando por parámetro un token de sesión. Mientras el token no sea correcto, el sistema devolverá siempre un error. Cuando el token es correcto, el sistema envía la información del PDF a la base de datos y se guarda el PDF asociado a un usuario.

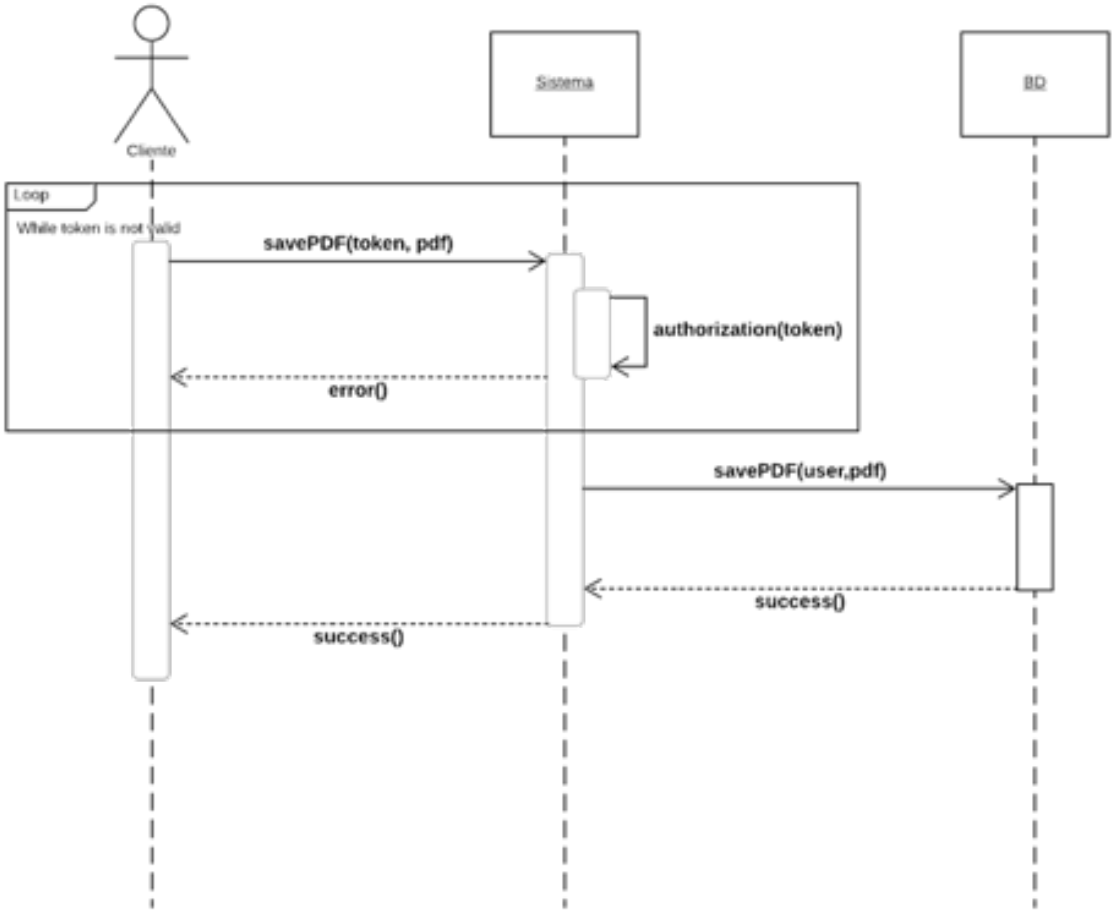


Figura 5.6: Diagrama de secuencia del guardado de PDFs

5.1.2. Cliente

En esta sección se detallan los problemas más importantes que se han tenido que solucionar en la parte de cliente, así como diagramas UML y explicativos para apoyar las decisiones.

Login

El proceso de login en el cliente es bastante sencillo. El usuario abre la aplicación y si no tiene sesión se redirecciona a la pantalla de login. En esta pantalla se introduce el nick y el password para iniciar sesión. La vista notifica a su presenter de que el usuario quiere iniciar sesión. Este llama al interactor que contiene el caso de uso de hacer login. A su vez, el interactor llama al repositorio del usuario para iniciar sesión. Este repositorio encapsula la capa de red y de datos bajo una misma interface. Se realiza la llamada a la API para iniciar sesión y obtener el token. En caso de que la llamada falle, se notifica en cascada hasta la vista, que muestra una alerta. Si la petición no falla, estaremos autenticados y obtendremos el token. Este token tiene que ser guardado para futuras peticiones.

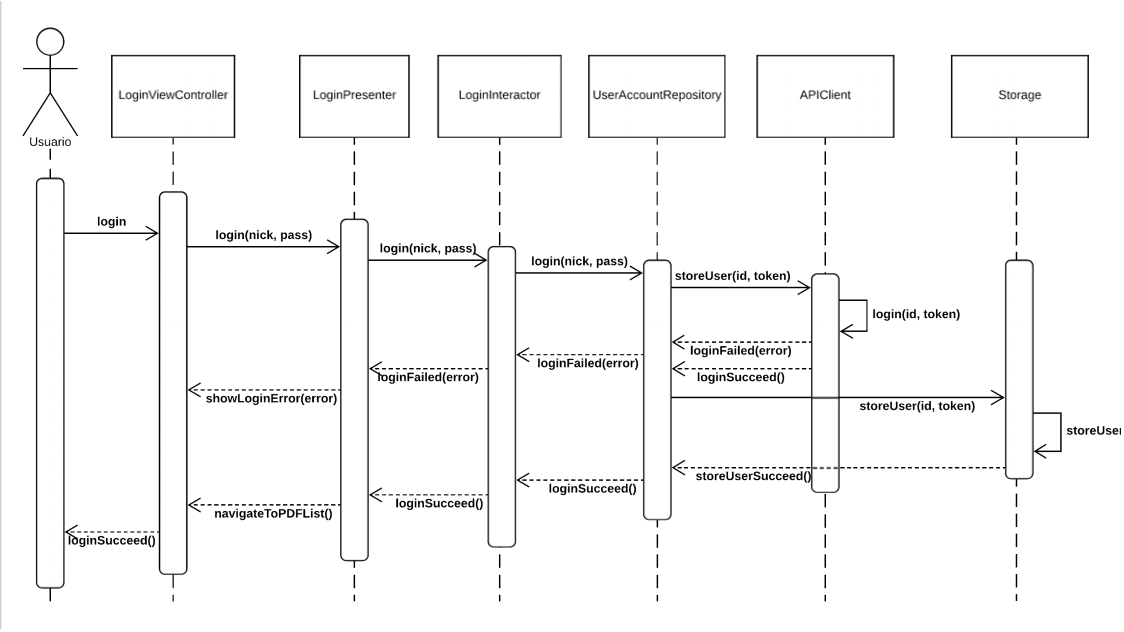


Figura 5.7: Diagrama de secuencia del login del cliente

Abrir PDF

Al abrir documentos PDF nos vamos a encontrar con dos posibles situaciones. Que el PDF está ya descargado en el dispositivo o que por el contrario no lo esté. Si el documento está descargado, el usuario puede hacer tap sobre él en la lista. En tal caso, el PDFListViewController lanza el evento de que se ha hecho tap sobre una cierta celda de índice N. PDFListViewController avisa a PDFListPresenter y esté accede al modelo situado en la posición N. En ese modelo se encuentra, entre otras cosas, el path donde el documento está descargado. Ese path es el parámetro necesario que necesita el interactor OpenPDF. Este interactor le pide al wireframe de ReaderKit que lance el visor dado en el path anterior.

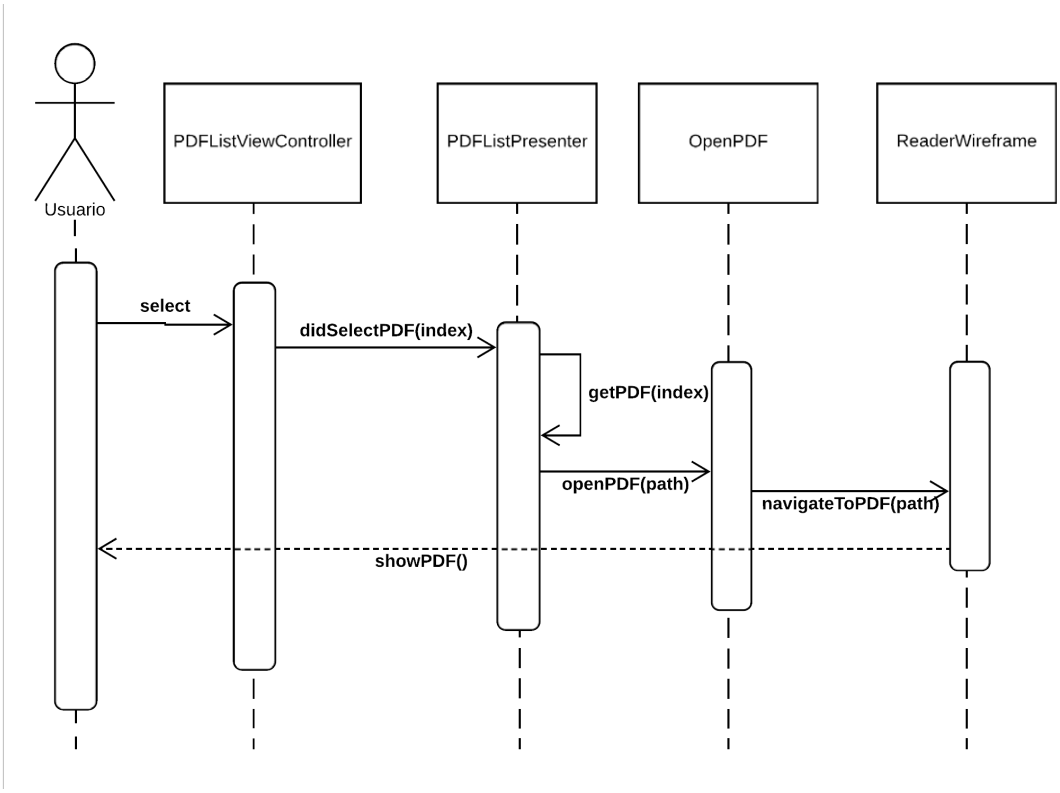


Figura 5.8: Diagrama de secuencia para abrir un documento PDF

Si por el contrario, el PDF no está descargado, hay que descargarlo primero antes de realizar esta secuencia.

Listar PDFs

Listar los documentos PDF disponibles también sigue la jerarquía de llamadas basada en VIPER donde la vista notifica al presenter y este ejecuta la lógica de negocio asociada.

Concretamente, cuando el usuario abre la aplicación, se lanza el evento viewDidLoad dentro del PDFListViewController. El PDFListPresenter es notificado y llama al interactor LoadPDFList para comenzar con la carga de datos. Este interactor habla con PDFRepository que maneja la capa de red y base de datos abstrayendo la fuente de datos. Se realiza la llamada API para obtener el array de documentos disponibles y se guarda en disco para evitar tener que lanzar la petición cuando no disponemos de conexión. Si la petición falla, se notifica en cascada a la vista que mostrará un empty case con un botón para recargar. Cuando el presenter recibe la notificación de que los datos están listos, avisa a la vista para que se recargue la tabla.

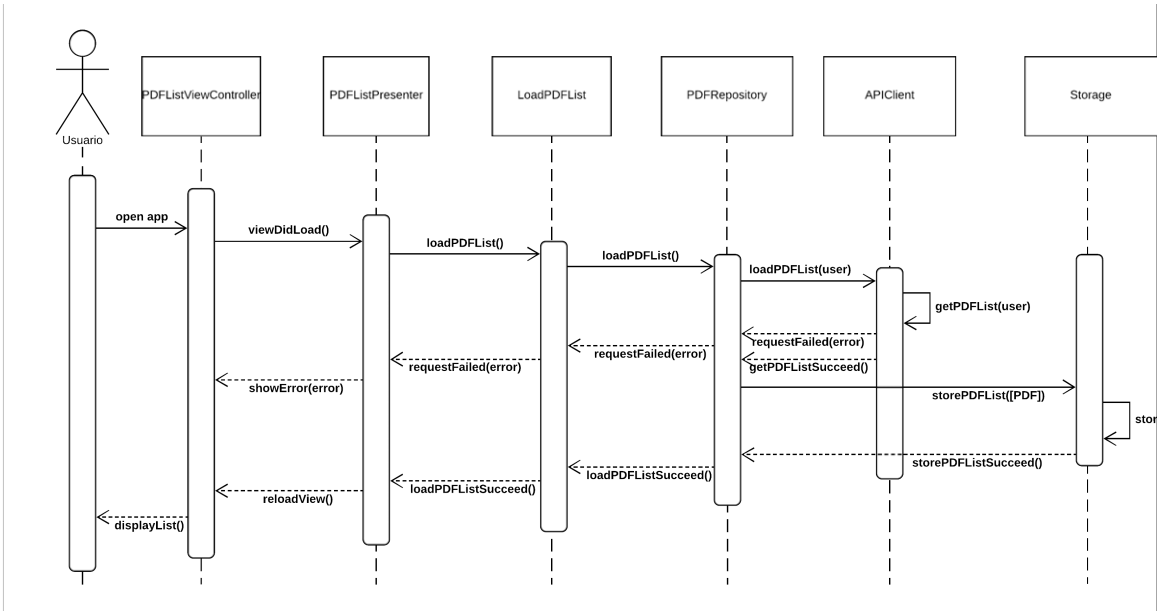


Figura 5.9: Diagrama de secuencia para cargar la lista de PDFs

Algoritmo de carga de páginas

Un libro tarda en abrirse el tiempo que tarda en renderizarse la primera página. Una vez abierta la primera página, el visor sigue renderizando páginas en segundo plano offscreen para que estén listas cuando se requiera su previsualización. Sin embargo, elegir el orden de cual es la siguiente página a renderizar no es trivial. Si una página no está disponible se bloquean el hilo principal para evitar que el usuario pueda llegar a una página en blanco. Es importante entonces, intentar que todas las páginas estén listas cuando el usuario quiera acceder a ellas.

Si renderizamos las N siguientes páginas a la actual, podríamos tener una mala experiencia de usuario si se intenta retroceder, ya que esta página no estaría disponible. Por otro lado, si renderizamos N páginas hacia atrás, es posible que el usuario nunca las llegue a ver, ya que es poco probable que el usuario retroceda. Se estarían gastando recursos en vano.

Es por eso que es necesario conseguir una secuenciación de renderizado que consiga la mejor calidad de experiencia para el usuario. Para ello se tendrá en cuenta la dirección sobre la que está avanzando el usuario, la página actual, la capacidad de memoria del dispositivo y el tamaño de pantalla.

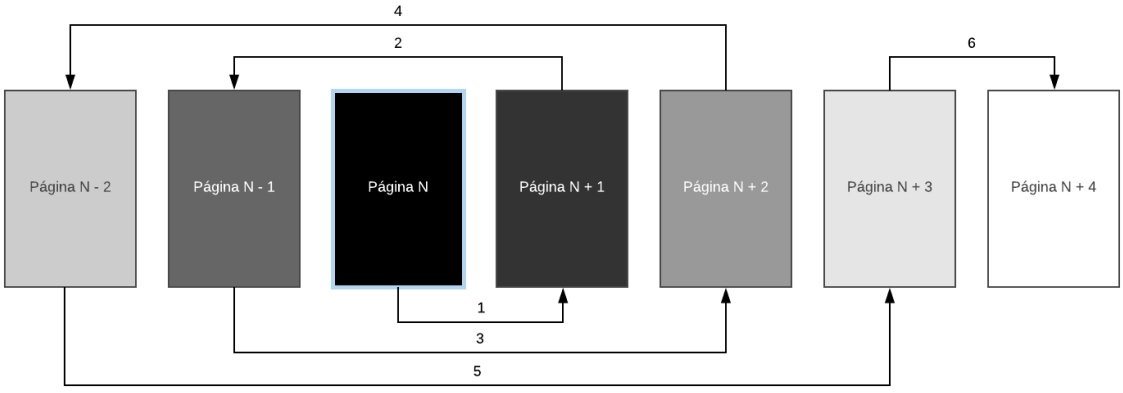


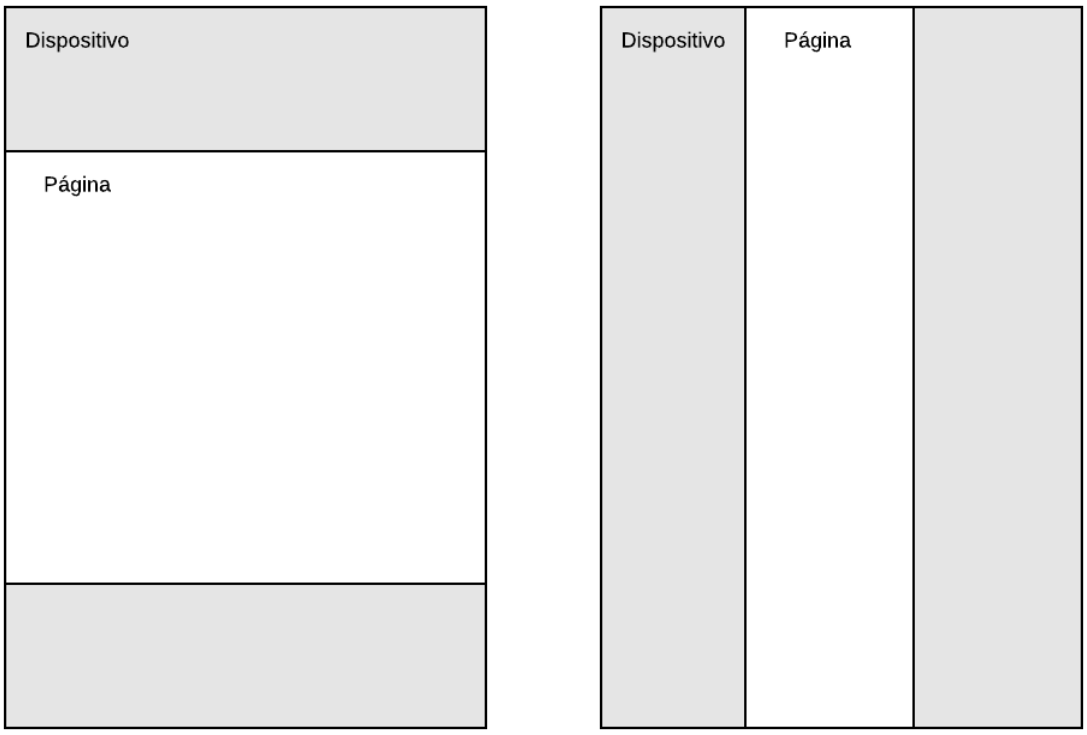
Figura 5.10: Algoritmo de renderizado de uso normal

En la figura anterior, se observa la secuenciación de cargas de página que se realiza al abrir el libro o avanzar de página. Todas las páginas se cachean, y esa caché se

comprueba antes de realizar otra renderización.

Estas imágenes se renderizan con el tamaño más grande que pueden tener en base a la pantalla del dispositivo que se está usando. El tamaño de la renderización viene determinada por el ancho y alto de la página en relación con el tamaño de la pantalla, manteniendo siempre el aspect ratio de la página original. Es decir, que si una página es más ancha que alta, se usara el ancho de la pantalla como ancho de página y se calculará el alto en base al aspect ratio de la página.

Del mismo modo, si una página es más alta que ancha, se usará el alto de la pantalla como alto de página y se calculará el ancho en base al aspect ratio.



(a) Página con más ancho (b) Página con más alto

Figura 5.11: Posibles tamaños de la página en relación con la pantalla

Las imágenes se cachean en memoria, es por eso que nuestro algoritmo también tiene que tener en cuenta la memoria y el tamaño de pantalla del dispositivo actual.

A mayor pantalla, mayor peso de las imágenes, y cuantas más imágenes más espacio ocupado en memoria.

Zoom

Poder realizar zoom dentro de una página de PDF es una tarea que supone sincronización de imágenes, concurrencia y renderización basada en capas, además del manejo de gestos por parte del usuario.

Cuando se abre un archivo PDF las páginas se cargan en forma de imágenes en base a un algoritmo anteriormente explicado. Estas imágenes se renderizan con el tamaño más grande que pueden tener la pantalla del dispositivo que se está usando. En general, será el tamaño que encajaría en el dispositivo en posición landscape. Sin embargo, cuando un usuario hace zoom, está de alguna forma cambiando virtualmente el tamaño de la pantalla del dispositivo. Es por esto, que si no se utilizase ningún sistema alternativo, las páginas se verían con poca calidad (puesto que se han renderizado en primera instancia para otro tamaño de pantalla). La ventaja de hacer el renderizado inicial de las páginas con el tamaño mínimo necesario para que se vean bien en el dispositivo es la rapidez de carga. Si se intentasen cargar imágenes más grandes de lo necesario al principio, la experiencia de usuario sería peor porque tardaría más tiempo en abrir el libro. Además, en la mayoría de los casos, los usuarios no hacen zoom, sería desperdiciar memoria, CPU y tiempo del usuario.

Se hace necesario entonces encontrar una solución alternativa.

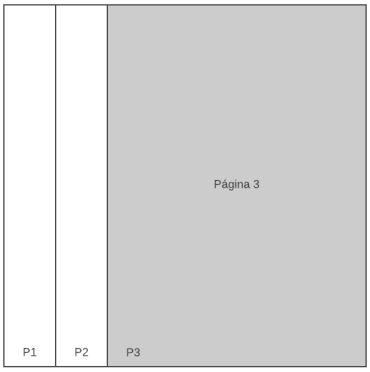


Figura 5.12: Carga de páginas PDF

Para mantener esa rápida carga inicial (manteniendo el renderizado único y conseguir el zoom en las páginas), se ha utilizado un sistema de renderización por capas. Esta renderización por capas divide la página de PDF en N cuadrados de igual tamaño. Después se encola el renderizado de estás capas en base a la zona de la pantalla que se quiere visualizar. Estas capas cargan de forma asíncrona, es por eso que mientras están cargando se va a ver la página con menor calidad de la esperada.

Cuando van cargando, se muestran encima de la página original. Hay hay que recalcar la importancia de que estas capas encajen de forma exacta en la página original, ya que no tienen por qué cargar de forma secuencial.

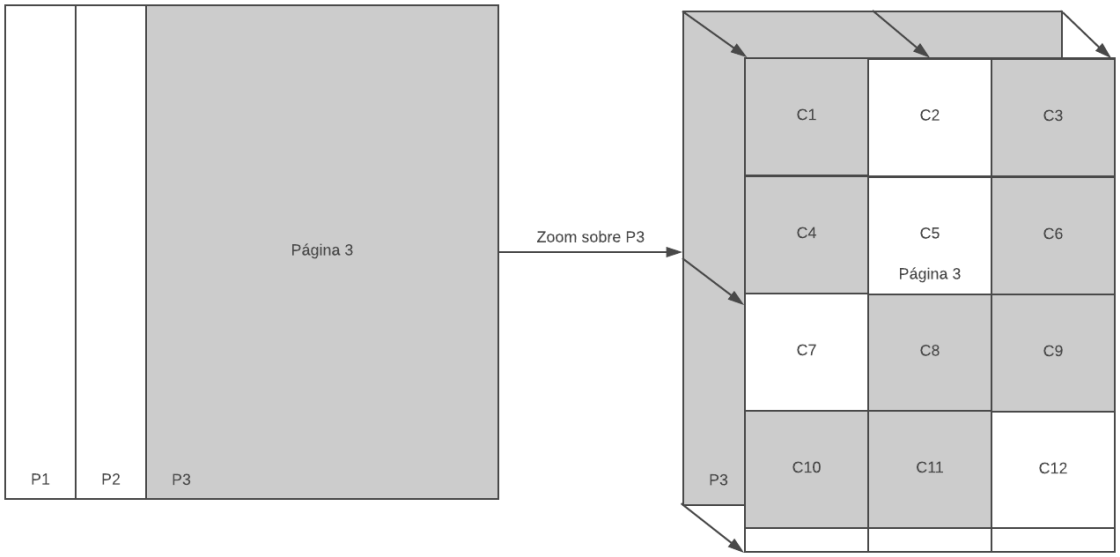


Figura 5.13: Zoom sobre una página PDF

A medidas que se hace o deshace zoom o se cambia la sección de la pantalla, las capas van reajustando sus prioridades para conseguir una carga más rápida.

Links

Permitir el uso de links dentro ReaderKit es un proceso manual. La información de estos links se encuentran dentro de una estructura de datos basada en clave:valor, un

diccionario. La información de los links se consigue accediendo a la clave *Annots*, que posee el documento.

El parseo de este diccionario es bastante complejo y puede ser una tarea con cierta carga de procesador. Es importante evitar realizar este parseo en el hilo principal.

Hay tres claves importantes dentro de *Annots*:

- **Subtype.** Contiene el tipo de anotación. En nuestro caso solo tendremos que tener en cuenta aquellos de tipo *Link*.
- **Rect.** Contiene el recuadro en el que se encuentra el link. Como se puede comprobar, esto denota la manualidad del proceso de uso de links dentro de un documento PDF. Es el desarrollador el que se tiene que encargar de manejar este recuadro, añadiendo algún tipo de sistema que permita que cuando el usuario haga tap sobre el link, se produzca la redirección.
- **Dest.** Contiene el destino del link. Puede ser de dos tipos: GoTo para redireccionamiento interno y URI para redireccionamiento externo.

Hay algunas otras anotaciones, sin embargo, para este proyecto solo hemos tenido en cuenta los links. Puede encontrar más información sobre los links y los diferentes tipos de anotaciones en la guía de referencia de PDF [Ado18].

ToC

Permitir al usuario acceder a la tabla de contenidos del documento es un proceso muy parecido al realizado con los links. Parecido en el sentido de manual. Toda la tabla de contenidos está localizada en un diccionario clave:valor con varias anidaciones. Por otro lado, hay varias formas de extraer estos datos, ya que diferentes archivos pueden implementar diferentes jerarquías de tabla de contenidos.

Concretamente para nuestro proyecto se han tenido en cuenta cuatro jerarquías. Cada una de estas jerarquías está formada por N ToCItem. Cada uno de estos items solo podrá realizar una navegación absoluta a una página concreta del documento.

Todos los elementos del ToC se encuentran dentro de la clave *Outlines* del catalog.

- **Por acción:** El item contiene un diccionario en la clave «A», el cual contiene la acción a ejecutar en la clave «D».
- **Por página:** El item contiene un diccionario en la clave «A», el cual contiene un array en la clave «D» cuyo primer elemento es la página.
- **Por acción en una lista:** El item contiene un valor en la clave «Dest», que indica la acción.
- **Por página en una lista:** El item contiene un array en la clave «Dest», cuyo primer elemento es la página.

Estos items pueden tener hermanos al mismo nivel que sigan la misma estructura. Esta jerarquía se repite de forma recursiva para conseguir los niveles de una tabla de contenidos.

El proceso de parseo es pesado, por ello, dicho proceso se ejecuta en una cola asíncrona intentando evitar hacerlo en la apertura del libro, ya que existe mucha carga en este punto.

Capítulo 6

Testing

6.1. Casos y traceabilidad

Esta sección corresponde a explicar los distintos casos de testing que se han realizado y a la traceabilidad de la aplicación. Se han realizado tanto test unitarios como funcionales, ambos en la parte servidor y en la parte cliente. Para hacer diferencia entre los test, se usa un código alfanumérico: TC para indicar que es un test; seguido de B o C para indicar backend o cliente; U o F para indicar si es unitario o funcional. Por último para hacer único el caso de test, se añade un código numérico.

Código	**TCBU01**
Título	Comprobar funcionamiento método isNullOrEmptyOrUndefined con valor null
Descripción	Para un objeto, se comprueba si el elemento tiene valor null
Diseño	Pasar como atributo la palabra reservada null
Condición	El valor debe ser null
Resultado	La función retorna el valor true

Tabla 6.1: Test TCBU01

Código	**TCBU02**
Título	Comprobar funcionamiento método isNullOrEmptyOrUndefined con valor undefined
Descripción	Para un objeto, se comprueba si el elemento tiene valor undefined
Diseño	Pasar como atributo la palabra reservada undefined
Condición	El valor debe ser undefined
Resultado	La función retorna el valor true

Tabla 6.2: Test TCBU02

Código	**TCBU03**
Título	Comprobar funcionamiento método isNullOrEmptyOrUndefined con una lista vacía
Descripción	Para un objeto, se comprueba si el elemento tiene longitud 0
Diseño	Pasar como atributo una lista sin elementos
Condición	El valor debe ser una lista sin elementos
Resultado	La función retorna el valor true

Tabla 6.3: Test TCBU03

Código	**TCBU04**
Título	Comprobar funcionamiento método isNullOrEmptyOrUndefined con cualquier valor
Descripción	Para un objeto, se comprueba si el elemento tiene algún valor
Diseño	Pasar como atributo un string
Condición	El atributo tiene que tener algún valor
Resultado	La función retorna el valor false

Tabla 6.4: Test TCBU04

Código	**TCBU05**
Título	Comprobar funcionamiento método createToken
Descripción	Comprobar la creación del token de usuario
Diseño	Pasar como atributo un usuario
Condición	El atributo tiene que ser la información del usuario necesaria para crear el token
Resultado	Un objeto con los atributos accesToken y tokenType
Requisitos involucrados	RB09

Tabla 6.5: Test TCBU05

Código	**TCBU06**
Título	Comprobar funcionamiento método getTokenFromHeader
Descripción	Comprobar que la cabecera de la petición tiene el token de usuario
Diseño	Pasar como atributo una petición con sus correspondientes cabeceras
Condición	La petición debe poseer al menos la cabecera Authorization y debe tener un valor válido
Resultado	La función introduce en la petición la información relativa al usuario

Tabla 6.6: Test TCBU06

Código	**TCBU07**
Título	Comprobar funcionamiento método getTokenFromHeader
Descripción	Comprobar que la cabecera de la petición no tiene el token de usuario
Diseño	Pasar como atributo una petición con sus correspondientes cabeceras
Condición	La petición no debe poseer la cabecera Authorization
Resultado	La función retorna la petición con estado 404 y un mensaje de error
Requisitos involucrados	RB09

Tabla 6.7: Test TCBU07

Código	**TCBU08**
Título	Comprobar funcionamiento método findByNickAndPass
Descripción	Comprobar que el método encuentra el usuario
Diseño	Pasar como atributos un nick y una password
Condición	El nick y el password deben pertenecer a un usuario registrado en la base de datos
Resultado	La función retorna la información del usuario

Tabla 6.8: Test TCBU08

Código	**TCBU09**
Título	Comprobar funcionamiento método findByNickAndPass
Descripción	Comprobar que la función retorna error en caso de no encontrar el usuario
Diseño	Pasar como atributo una password o un nick que no concuerdan con la información del usuario en base de datos
Condición	La información no está relacionada con ningún usuario en base de datos
Resultado	La función retorna la petición con estado 401 y un mensaje de error

Tabla 6.9: Test TCBU09

Código	**TCBU10**
Título	Comprobar funcionamiento método isRegister
Descripción	Comprobar que la función retorna error en caso de que el usuario ya esté registrado
Diseño	Pasar como atributo un usuario registrado
Condición	El usuario debe estar en base de datos
Resultado	La función retorna la petición con estado 500 y un mensaje de error

Tabla 6.10: Test TCBU10

Código	**TCBU11**
Título	Comprobar funcionamiento método isRegister
Descripción	Comprobar que la función no retorna error en caso de que el usuario no esté registrado
Diseño	Pasar como atributo un usuario válido
Condición	El usuario no debe estar en base de datos
Resultado	La función permite continuar el flujo de trabajo

Tabla 6.11: Test TCBU11

Código	**TCBU12**
Título	Comprobar funcionamiento método findByFilename
Descripción	Comprobar que la función busca un PDF por nombre de archivo
Diseño	Pasar como atributo un nombre válido
Condición	El PDF debe existir en base de datos con el nombre del atributo
Resultado	La función retorna la información del PDF

Tabla 6.12: Test TCBU12

Código	**TCBU13**
Título	Comprobar funcionamiento método findByFilename
Descripción	Comprobar que la función busca un PDF por nombre de archivo y retorna error si no lo encuentra
Diseño	Pasar como atributo un nombre no válido
Condición	El PDF no debe existir en base de datos
Resultado	La función retorna un error con estado 404 y un mensaje de error

Tabla 6.13: Test TCBU13

Código	**TCBU14**
Título	Comprobar funcionamiento método saveUser
Descripción	Comprobar que la función guarda el usuario con información encriptada
Diseño	Pasar como atributo la información adecuada
Condición	El usuario no debe estár ya guardado
Resultado	La función permite continuar con el flujo
Requisitos involucrados	RB05

Tabla 6.14: Test TCBU14

Código	**TCBF01**
Título	Login
Descripción	El usuario realizar un login
Diseño	Petición con información del usuario
Condición	El usuario debe existir en base de datos
Resultado	La petición devuelve un estado 200 y la información del token de sesión del usuario
Requisitos involucrados	RB05, RB08

Tabla 6.15: Test TCBF01

Código	**TCBF02**
Título	Login
Descripción	El usuario realizar un login erróneo
Diseño	Petición con información del usuario errónea
Condición	El usuario debe existir en base de datos y la información en la petición debe no corresponder con la información almacenada
Resultado	La petición devuelve un estado 401 y la información del token de sesión del usuario
Requisitos involucrados	RB05, RB08

Tabla 6.16: Test TCBF02

Código	**TCBF03**
Título	Registro
Descripción	El usuario se registra
Diseño	Petición con información del usuario
Condición	El usuario no debe existir en base de datos
Resultado	La petición devuelve un estado 201
Requisitos involucrados	RB05, RB07

Tabla 6.17: Test TCBF03

Código	**TCBF04**
Título	Registro
Descripción	El usuario no se registra
Diseño	Petición con información del usuario
Condición	El usuario debe existir en base de datos
Resultado	La petición devuelve un estado 500 y un mensaje de error
Requisitos involucrados	RB04, RB07

Tabla 6.18: Test TCBF04

Código	**TCBF05**
Título	Obtener lista de PDF
Descripción	Obtener el listado de PDF asociados al usuario
Diseño	Petición con token de sesión del usuario
Condición	El token de sesión debe ser válido
Resultado	La petición devuelve un estado 200 con un listado de nombres de los archivos asociados al usuario
Requisitos involucrados	RB10

Tabla 6.19: Test TCBF05

Código	**TCBF06**
Título	Obtener detalle de PDF
Descripción	Obtener el detalle de un PDF asociado al usuario
Diseño	Petición con token de sesión del usuario
Condición	El token de sesión debe ser válido
Resultado	La petición devuelve un estado 200 con el detalle del archivo
Requisitos involucrados	RB11

Tabla 6.20: Test TCBF06

Código	**TCBF07**
Título	Guardar un PDF
Descripción	Usuario guarda un nuevo PDF
Diseño	Petición con token de sesión del usuario he información de un PDF
Condición	El token de sesión debe ser válido y el PDF debe ser nuevo
Resultado	La petición devuelve un estado 201
Requisitos involucrados	RB12

Tabla 6.21: Test TCBF07

6.1.1. Despliegue

En esta sección se explica cómo poder realizar los despliegues para la aplicación tanto en la parte servidor como en el cliente iOS.

Servidor

Uno de los puntos por lo que se eligió NodeJS fue la manera en que se podía recrear un servidor de manera rápida. Para poder desplegar la aplicación se requiere tener instalado NodeJS y MongoDB en el sistema. Además se debe de haber configurado la base de datos anteriormente.

Si ya los tenemos instalados, debemos arrancar MongoDB en la terminal con permisos de administrador. Para ello, con la terminal nos vamos a la ruta donde tenemos MongoDB instalado, accedemos a la carpeta bin y ejecutamos el comando «./mongod». Con otra ventana de la terminal, en la misma ruta ejecutamos el comando «./mongo» para crear una instancia a nuestras bases de datos. Para crear la base de datos y collecciones necesarias realizamos los siguientes comandos en la terminal con mongo arrancado: *use tfg*, *db.createCollection(users)*, *db.createCollection(pdfs)*.

Con estas secuencias tenemos los pasos mínimos para arrancar nuestra base de datos con las colecciones que vamos a usar.

A continuación nos dirigimos con una nueva ventana de la terminal a la ruta donde tenemos el proyecto guardado y ejecutamos *npm install*. Esto instala los paquetes necesarios que se utilizan en la aplicación.

Tras instalar los paquetes, el servidor se arranca mediante el comando *npm start*. Este comando arranca un servidor local en el puerto 3080. Este puerto está definido por defecto en el archivo /constants/constants.js

Para visualizar la documentación de la API rest se hace mediante la ruta /api-docs.

Como podemos observar, la mayor dificultad para arrancar nuestro servidor se basa en tener preinstalado las tecnologías necesarias. Pero en el caso de tenerlas, lo único necesario es tener previamente arrancada la base de datos para poder realizar la conexión y ejecutar el comando npm start, y con estos dos pasos tenemos un servidor arrancado y operativo.

En conclusión, NodeJS nos ofrece de manera rápida y sencilla un servidor operativo, sin necesidad de configuraciones complicadas o necesidad de herramientas externas.

Cliente

En esta sección se explica todo lo necesario para subir una aplicación a la App Store de Apple. El proceso es distinto de otras stores porque Apple hace reviews individuales para poder verificar que se siguen sus directrices. Lo primero que se necesita es inscribirse en el programa de desarrollo de Apple (Apple Developer Program) como una persona individual o como una organización. Las organizaciones tienen que estar legalmente registradas como empresas ya que es necesario introducir el CIF para inscribirse. Ambos programas tienen un precio de 99 euros.

La cuenta brinda acceso tanto al centro de miembros (Member Center) como a la plataforma iTunes Connect que se usa para subir aplicaciones en la App Store. El centro para miembros contiene un panel donde los desarrolladores pueden administrar certificados, identificadores, dispositivos y perfiles de aprovisionamiento. Para probar las aplicaciones localmente, se necesita un certificado de desarrollo. Se necesita un certificado de producción para subir aplicaciones en la App Store. Los identificadores de aplicaciones identifican las aplicaciones inequívocamente en la App Store. La sección de dispositivos permite a los desarrolladores registrar sus dispositivos, lo que les permite probar y depurar aplicaciones antes de subirlas en la App Store. Los perfiles de aprovisionamiento se utilizan para preparar y configurar una aplicación para que se inicie en dispositivos y use los servicios de la aplicación. Se componen de un identificador de aplicación, un certificado y una lista de dispositivos asociados. Los perfiles de aprovisionamiento permiten que las aplicaciones utilicen los servicios de Apple, como las notificaciones push. Una vez que el perfil de aprovisionamiento esté listo, la aplicación necesita usar un SDK estable; Apple no permite subir aplicaciones usando versiones beta de sus herramientas. Antes de subir la aplicación, es necesario registrarla en el portal de iTunes Connect. El registro de una aplicación implica cargar un icono, hacer unas de capturas de pantalla, escribir una descripción, etc. Cuando se sube la aplicación, generalmente demora alrededor de dos semanas en aprobarse o rechazarse. En caso de que se rechace, el revisor deja una nota con el motivo, generalmente la directriz

de incumplimiento. Por otro lado, si la aplicación es aceptada, debería estar disponible para los clientes en cuestión de minutos.

Capítulo 7

Conclusiones y trabajo futuro

El proyecto se estableció para explorar y aprovechar los últimos avances tecnológicos en computación ubicua.

En el aspecto técnico, los objetivos principales eran construir un backend altamente escalable que fuera testeable, y desarrollar una aplicación de iOS altamente interactiva usando Swift y el Paradigma Reactivo Funcional.

7.1. Contribuciones

Esta sección se revisa los objetivos principales que se establecieron inicialmente para este proyecto (ver el Capítulo 4) junto con una discusión sobre cómo se implementó cada uno de ellos, junto con su estado al momento de finalizar el proyecto.

- **Análisis de las posibles tecnologías que se pueden usar para el backend y la base de datos.** Para la base de datos se tienen dos posibilidades: SQL o noSQL. Una base de datos SQL nos ofrece una robustez en cuanto a las relaciones, pero para nuestro caso se ha elegido una base de datos no relacional ya que solo poseemos dos entidades, Usuarios y PDF, y una base de datos no relacional nos permite trabajar con más tipos de atributos sin tener que definirlos previamente. Para el backend, existen varias tecnologías que son altamente mantenibles además de ofrecer calidad y diversos frameworks de trabajos. En este proyecto han sido valoradas Java y PHP, sin embargo se ha optado por el uso de NodeJS que usa

JavaScript como lenguaje de programación. Esta elección viene sobre todo por la facilidad en que NodeJS ofrece una arquitectura hexagonal con el uso de express, a la vez de que nos permite tener un servidor configurable en tan solo varios minutos.

- **Implementación de una API Rest**. Cómo obtener los datos es un factor importante que no siempre se tiene en cuenta. Para ello se deben de seguir una serie de reglas a la hora de formalizar las url y asegurarse de que todas siguen el mismo formato. Seguir la arquitectura planificada y crear una buena relación entre las capas es difícil, ya que muchas veces se tiende a realizar las funcionalidades en el lugar donde se necesitan. Sin embargo, para la implementación de la API se ha llevado a cabo un seguimiento en cada una de las capas intentando cumplir con exactitud la arquitectura planteada. Para la API rest se tiene un CRUD de los PDF del usuario.

- **Diseño y desarrollo del backend.** Para el proyecto se ha propuesto un diseño básico, para poder minimizar los errores y las dependencias en el desarrollo. El diseño proviene de una arquitectura compuesta entre Clean Architecture y Arquitectura Hexagonal. De ambas arquitectura la idea principal seleccionada para este caso es la separación completa de las responsabilidades en capas. Un desarrollo limpio y sin errores ofrece mayor posibilidad de escalado en cuanto a introducir nuevos requisitos se refiere.

- **Modelar un renderizador de PDFs.** Sin lugar a dudas la parte más apasionante del proyecto. Se ha conseguido implementar un renderizador de PDF que soporta zoom, orientaciones, links y tabla de contenidos. Todo ello manteniendo una velocidad rápida de renderizado y una interfaz fluida para el usuario. Utilizar las APIs que proporciona CoreGraphics han ayudado bastante a centrarse en la calidad aportada para el usuario final. Apple ha hecho un gran esfuerzo en la implementación de su API para PDFs.

- **Diseñar y desarrollar un cliente iOS.** El proyecto se construyó utilizando la arquitectura VIPER para facilitar el trabajo en otras plataformas como watchOs y

macOS. Sería relativamente fácil adaptarlo a otros sistemas operativos mientras se reutiliza la mayor parte del código. El Paradigma Reactivo Funcional junto con el lenguaje de programación Swift también fueron adoptados para crear una experiencia realmente interactiva y minimizar errores.

- **Explorar el lenguaje de programación Swift y justificar su elección sobre Objective-C.** El lenguaje de programación Swift es un lenguaje nuevo presentado por Apple hace tres años. Al desarrollar una aplicación de iOS, los desarrolladores pueden elegir usar Swift u Objective-C. El cliente de iOS se creó usando Swift. Swift es un lenguaje más rápido, más seguro, más poderoso y más moderno que Objective-C. Su elección para este proyecto definitivamente ha ayudado al no permitir que se introduzcan errores por ser más estrictos que el lenguaje anterior. Gracias a su sintaxis moderna, también ha ayudado a reducir el tiempo de desarrollo.

- **Diseño e implementación de los test.** Los tests son una parte crítica de cada aplicación de software. El trabajo presentado tiene docenas de casos de prueba que fueron desarrollados usando pruebas unitarias y pruebas funcionales. Algunas técnicas recientes, como las pruebas HTTP, han sido probadas con el uso de Postman. Estas se usaron para cubrir partes de la aplicación que antes no se podían haber cubierto.

- **Trabajar con metodologías ágiles.** Una buena metodología condiciona las pautas principales del equipo y finalmente el resultado. La elección de las tecnologías ágiles es principalmente el dar a los integrantes del proyecto la libertad de organizar y realizar las tareas cuando uno mismo quiera, pero siempre teniendo control sobre ellas. Para este proyecto, el cual no es complejo, se ha elegido KANBAN, que se adapta perfectamente a las preferencias de los desarrolladores y al tiempo de ejecución del proyecto.

7.2. Conclusiones

- Para este proyecto han sido utilizadas varias de las tecnologías que actualmente son las más usadas en el sector. La adquisición de conocimientos sobre tecnologías que los desarrolladores van a utilizar en el futuro, hace que los integrantes del equipo entiendan lo difícil que es mantenerse actualizado debiendo prestar mucha atención, de manera continuada, a las nuevas versiones. Debiendo estudiar, si dichas versiones se adaptan a los requisitos, si son viables para adaptarlas, o si por otro lado, es mejor despreciarlas y seguir trabajando con una versión antigua.

- La toma de decisiones ha sido el factor más importante. No solo en cuanto a stack tecnológico se refiere, si no al orden y la organización llevados para poder sacar de manera satisfactoria el proyecto. El buen uso de las tecnologías ágiles ha hecho posible que los errores producidos se hayan ido abordando de manera eficaz sin prolongar el tiempo de desarrollo.

- El trabajo en equipo es algo fundamental para cualquier desarrollo. Siempre se trabaja de manera más sencilla, cuando los requisitos van saliendo y ambas partes trabajan por igual. Gracias a la buena organización, y a la responsabilidad de cada uno, los plazos acordados se han ido cumpliendo, y esto a favorecido al éxito del proyecto. Esta actitud es una gran base, ya que en el sector de la informática, la mayoría de los trabajos de desarrollo son en equipo.

- El desarrollo individual siempre es un gran problema. En la mayoría de los casos, la parte backend se desarrolla de manera independiente de la parte frontend acordando mediante el uso de alguna interfaz como se van a realizar las comunicaciones entre ambas partes. Sin embargo, en el momento de realizar un cambio, al no tener conocimiento sobre la otra mitad, el mínimo cambio puede ocasionar un gran impacto o incluso un nuevo desarrollo. Por ello, para este caso se decidió que ambos integrantes debían tener conocimientos de ambos sectores, para que en el momento de realizar la integración se redujese en el mayor número posible los problemas y los errores. Otra medida que se tomó para este caso, fue el tener siempre a disposición una interfaz de los servicios mediante el uso de swagger.

- En algunas ocasiones ocurre que se desarrolla un algoritmo o una función, y acto seguido se optimiza el código de desarrollo. Sin embargo, tras esta optimización ocurres fallos no controlados o incluso hay que descartar el cambio. El uso de GIT y de la metodología GIT-Flow ha ayudado a no perder funciones realizadas de manera correcta con inserciones de código no probado, teniendo siempre una versión estable en la que los desarrolladores pudiesen trabajar sin fallos.

- Para los desarrolladores realizar buenos casos de test ayuda a encontrar fallos no controlados. Con este trabajo, se ha visto la importancia de los test, y el esfuerzo que supone no desarrollar test a la par que se va desarrollando las funciones, ya que para el futuro, reduce el tiempo de búsqueda de errores a la vez que proporciona calidad a nuestro sistema.

7.3. Futuro trabajo

El proyecto tiene un gran potencial para ampliar sus horizontes. Como ya se ha comentado, uno de los requisitos para este proyecto era desarrollar una aplicación que fuera escalable. En esta sección se presenta algunos temas que están siendo estudiados y planeados para el futuro tanto a corto como a largo plazo.

7.3.1. Multiplataforma

A corto plazo, se espera que el proyecto tenga un ecosistema multiplataforma. La aplicación de Android se creará aprovechando la estabilidad del backend y de forma nativa. La aplicación de iOS también se trasladará a watchOS cuando llegue a una versión estable (actualmente es una versión beta). Por otro lado, se plantea la posibilidad de una versión web con algunas implementaciones nuevas, pero este caso a largo plazo.

7.3.2. Compresión y encriptado

A pesar de que se ha intentado hacer un POC (Proof of concept), ni el cliente ni el servidor están aún capacitados para permitir el encriptado y compresión de los archivos

PDF. Sin embargo, se han sentado bases para poder completar la implementación en futuras versiones.

7.3.3. Versiones con distintas funcionalidades

Otro punto que se tuvo en cuenta al principio del desarrollo, fue el poder disponer de distintas versiones (basic y pro) las cuales tuviesen distintas funcionalidades. La versión actual no tiene limitaciones, pero se quiere separar las versiones de manera que haya dos tipos de usuarios, los cuales verán limitadas sus opciones de guardado, con tamaños máximos de archivos, etc. Por otro lado, la versión completa, no solo dispone de todo lo desarrollado actualmente si no que también será la versión en la cual se irán implantando los nuevos requisitos.

7.3.4. Backend en AWS

El backend ha sido desarrollado en NodeJS. Actualmente es un servidor que se ejecuta de manera local en este proyecto, sin embargo, se está estudiando como implantar este servidor en Amazon Web Services (AWS).

AWS es una plataforma cloud computing que permite obtener servidores por todo el mundo. Algunas de las herramientas que soportan son NodeJS, JavaScript , iOS y Android. Son las plataformas que actualmente se están utilizando para el proyecto, por ello se quiere trasladar el proyecto a esta plataforma con la idea de poder llegar al máximo de usuarios posibles.

Esta idea está pensada a corto plazo, ya que la integración con el servicio conlleva una serie de integraciones y configuraciones que están implementadas en el código, pero no están probadas.

7.3.5. Monitorización

Como ya se ha explicado en el apartado anterior, la plataforma AWS nos ofrece una herramienta de monitorización llamada CloudWatch. Con esta herramienta podemos ver todos los informes de nuestra aplicación, además de poder configurar una serie de alertas y alarmas en el caso de que el sistema falleo tenga errores. También ofrece un

sistema de monitorización en cuanto al rendimiento y a los tiempos de las llamadas, ofreciendo un servicio bastante completo y sencillo de usar, ya que proporciona una consola e incluso ayuda online

Otra característica, es el poder configurar los servidores aumentando el tamaño de memoria, de disco, las capacidades para las bases de datos... Sin embargo, muchos de estos servicios son ofrecidos en versiones de pago, igual que la instalación en varios servidores distribuidos de manera mundial. Es por ello, que la implantación de esta plataforma esté estudiada para versiones futuras.

7.3.6. Implementación de test en backend con Mocha

Actualmente la parte backend ha sido provada mediante test unitarios que los propios desarrolladores han ido probando según iban desarrollando las funciones. Sin embargo, para la siguiente versión, se quiere implementar un buen sistema de test. Se ha optado por Mocha, que es un framework JavaScript para pruebas unitarias sobre NodeJS. La elección de este framework corresponde en mayor parte a que el backend está desarrollado en NodeJS.

Capítulo 8

Identificación de impactos y aspectos éticos, sociales y ambientales relacionados con el proyecto

8.1. Contexto general del proyecto

El proyecto es una aplicación móvil ambientada en el sector tecnológico de los Smartphones. Sector que año tras años evoluciona en paralelo con la evolución de las tecnologías y el software.

El ciclo de vida en el que se enmarca el proyecto, son todas las fases del ciclo de vida de un proyecto software (requisitos, análisis, diseño, implementación, pruebas, mantenimiento).

El contexto socio-económico, geográfico y cultural en el que se enmarca el proyecto puede ser cualquiera.

Los grupos de interés relativos al proyecto son en principal medida grupo de personas que utilicen dispositivos móviles en su día a día. Trabajadores que requieran llevar documentos de manera portátil y necesiten de almacenamiento en la nube. Estudiantes que prefieran usar documentos digitales, etc.

8.2. Impacto del proyecto:

En los últimos años el uso de teléfonos móviles se ha hecho fundamental en nuestros sistemas de vida. Esto supone que las compañías necesiten mejorar la calidad y los materiales de sus productos cada año para poder actualizarse y que sus productos se vendan en el mercado. Esto hace que las fábricas de teléfonos móviles estén por encima en los rankings. Por ejemplo, Apple, está denominada la empresa más contaminante en estados unidos según artículos de green peace. [Gar11]

Por otro lado, mucho de los productos utilizados en los Smartphones son reutilizables, lo que hace que muchas compañías faciliten y promuevan la renovación de sus productos siempre y cuando entregues el antiguo. También el mayor uso de formato digital afecta a las impresiones en papel, que han sido reducidas considerablemente tanto en estudiantes como en empresas.

Otro de los puntos clave de este proyecto, es el manejo de la propiedad intelectual del contenido que nuestros usuarios quieren hacer uso. Las grandes plataformas de lectura son capaces de ofrecer contenido bajo demanda, así como ventas de contenido. Esto les obliga a llevar un control más minucioso de todo el contenido que ofrecen, para garantizar el derecho de autor de los creadores. Sin embargo, esto no ocurre con nuestro proyecto, puesto que el objetivo de este, es que el usuario pueda compartir sus libros ya comprados entre varios de sus dispositivos. Es por esto, y porque no tenemos el concepto de compartir libros entre usuarios, por lo que nuestro proyecto queda de algún modo eximido de tener que manejar estos comprometidos y complejos conceptos legales.

La lectura es uno de los pilares culturales para el ser humano. Poder hacerlo a través de un dispositivo móvil supone un gran apoyo para el crecimiento cultura de la población. Añadir facilidades a través del uso de las nuevas tecnologías como pueden ser el almacenado en la nube y el uso de multi-dispositivos, hace que el usuario final se sienta más cómodo. Con nuestra app, hemos intentado conseguir una gran calidad para el usuario final, de forma que la lectura desde cualquier parte sea lo más liviana y satisfactoria posible.

Capítulo 9

Bibliografía

[Mar08] Robert C. Margin. *Clean Architecture*. Jul. de 2008.

[Gar11] Jodie Van horn Gary Cook. *How dirty is your data?* 2011. URL: https://www.greenpeace.org/archive-international/Global/international/publications/climate/2011/CoolIT/dirty-data-report-greenpeace.pdf.

[Gil13] Laia Gilibets. *Que es la metodología Kanban y cómo utilizarla*. 2013. URL: https://www.iebschool.com/blog/metodologia-kanban-agile-scrum/ (visitado 15-07-2018).

[Mar14] Eduardo Martínez. *Las 8 grandes ventajas de las metodologías ágiles*. 2014. URL: https://www.iebschool.com/blog/que-son-metodologias-agiles-agile-scrum/ (visitado 01-08-2018).

[Egu15] Javier Eguiluz. *Introducción a JavaScript*. 2015. URL: http://librosweb.es/libro/javascript/ (visitado 15-07-2018).

[Jef16] Ken Schwager y Jeff Sutherland. *La guía definitiva de Scrum: Las reglas del juego*. 2016. URL: http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-Spanish.pdf#zoom=100 (visitado 15-07-2018).

[Luc16] Luckys. *Introducción Arquitectura Hexagonal*. 2016. URL: https://laraveles.com/introduccion-arquitectura-hexagonal (visitado 04-08-2018).

[Ado18] Adobe. *Document management - Portable document format*. 2018. URL: https://www.adobe.com/content/dam/acom/en/devnet/pdf/pdfs/PDF32000_2008.pdf (visitado 15-07-2018).

Capítulo 10

Anexo

10.1. Apéndice 1: Ejemplo de diccionario de links y ToC

10.1.1. Link


```
{
        "Border": [
                0,
                0,
                0
        ],
        "Subtype": "Link",
        "A": {
        },
        "Type": "Annot",
        "Rect": [
                201,
                20.68,
                302,
                6.6799
```

```
        ]
}
```

10.1.2. ToC

```
{
        "Type": "Catalog",
        "Metadata": "Stream",
        "ViewerPreferences": "Dictionary",
        "PageLayout": "SinglePage",
        "Outlines": "Dictionary",
        "AcroForm": "Dictionary",
        "Names": "Dictionary",
        "PageMode": "UseOutlines",
        "OpenAction": "Dictionary",
        "Pages": "Dictionary",
        "PageLabels": "Dictionary"
}
```

Glosario

AppStore Aplicación donde se descargan las apps pertenecientes al sistema iOS.. 17

BDR Abreviatura para referirse a una Base de datos relacional.. 24

capabilities Conjunto de capacidades.. 36

catalog Catalogo de elementos.. 77

crashea Término usado cuando una aplicación finaliza el flujo debido a un fallo.. 46

ePub Formato de código abierto para leer texto e imágenes.. 18

fixed-layout Distribución de elementos de forma estática.. 16

framework Es una estructura definida, con módulos concretos, que ayuda a la definición y desarrollo software.. 19

layers En software, referencia a una capa de acceso de datos. 20

Open Source Término que se usa en la informática para referirse a un modelo de código abierto donde cualquiera puede colaborar.. 19

PDF Siglas que significan Portable Document Format.. 16

PlayStore Aplicación de descarga de aplicaciones correspondiente al sistema Android.. 17

PSPDFKit Kit de desarrollo software para la visualización de PDFs.. 19

smartphone Término por el que se conoce a un dispositivo móvil, que permite conectarse a internet, y ofrece una serie de servicios como correo electrónico, mensajería, llamadas, etc.. 17

stack tecnológico Conjunto de subsistemas o componentes necesarios para crear una plataforma completa.. 24

VIPER Arquitectura limpia basada en la distribución de Vista, Interactor, Presenter, Entidad y Router.. 23

wireframes Guía visual que representa el esqueleto de un sitio web.. 34