Facultad de Ciencias

DESARROLLO DE UNA APLICACIÓN WEB PARA LA GESTIÓN DE SUSCRIPCIONES A DISTINTOS SERVICIOS

Development of a web application for the management of subscriptions to different services

Trabajo de Fin de Grado para acceder al

GRADO EN INGENIERÍA INFORMÁTICA

Autor: Jorge Rosa Lago

Directora: Patricia López Martínez

Febrero - 2024

RESUMEN

El desarrollo de la tecnología en los últimos años ha hecho que se creen nuevas formas de entretenimiento y servicios que aportar a la gente. Hoy en día no hay prácticamente nadie que no se encuentre suscrito a algún tipo de servicio, ya sea de entretenimiento, trabajo, estudio, deporte o incluso para poder gestionar las necesidades básicas como son la luz, el agua o el gas. Este aumento en el número de posibilidades a las que suscribirse ha hecho que pueda ser difícil llegar a controlar y administrar bien todos los gastos que esto puede generar.

Por ello se propone el desarrollo de una aplicación web que ayude a llevar a cabo esta tarea, permitiendo consultar con claridad todos los servicios a los que estamos suscritos y los gastos que acarrean y de este modo, poder administrar mejor las suscripciones de cada persona y ver hasta qué punto merece la pena el pago de ciertas tecnologías y/o servicios que se están disfrutando.

La aplicación web estará desarrollada en React y se conectará a través de un servicio REST desarrollado en Spring a una base de datos gestionada por MySQL Server en la que se alojará la información de los usuarios y sus suscripciones.

Palabras clave: Servicio, Aplicación Web, React, servicio REST.

ABSTRACT

The development of technology in recent years has led to the creation of new forms of entertainment and services to provide to people. Nowadays there is virtually no one who is not subscribed to some kind of service, whether it is for entertainment, work, studies, sports or even for managing basic necessities such as electricity, water or gas. This increase in the number of possibilities to which be subscribed has made it difficult to control and manage well all the expenses that a person can have.

For this reason this work proposes the development of a web application that helps to carry out this task in which you can clearly see all the services to which you are subscribed and the costs they entail and thus to better manage the subscriptions of each person and see to what extent it is worth paying for certain technologies and / or services that are being enjoyed.

The web application will be developed in React and it will be connected through a REST service developed in Spring to a database managed by MySQL Server where the information of the users and their subscriptions will be stored.

Keywords: Service, Web application, React, REST service.

Índice

Índice de tablas

Índice de figuras

1. INTRODUCCIÓN Y OBJETIVOS DEL PROYECTO

Este capítulo se divide en tres secciones. Primero se expone el contexto de la aplicación, en concreto, los problemas que puede acarrear el estar suscrito a mútiples servicios. A continuación, se detallan las funcionalidades que debe aportar la aplicación para poder resolverlos y por último se proporciona una breve explicación de como se divide este documento.

1.1. Contexto

Hoy en día, los avances tecnológicos han cambiado profundamente la forma en que interactuamos con el mundo que nos rodea, tanto en el ámbito del entretenimiento como en nuestra vida profesional. La conectividad global y la digitalización de servicios han marcado el comienzo de una nueva era caracterizada por una abundancia de servicios de suscripción que satisfacen una variedad de necesidades, desde entretenimiento hasta productividad.

Los servicios de entretenimiento son la principal fuente de suscripciones que tienen las personas hoy en día, servicios como Netflix, Amazon Prime Video o Disney+ son muy comunes entre la población. También contamos con aplicaciones como Spotify o YouTube Music que nos ofrecen un gran repertorio de música y podcasts de calidad. Por otro lado, también contamos con servicios que nos aportan facilidades a la hora de realizar nuestro trabajo o para estudiar, como puede ser una licencia de Office u otros programas de pago. Por no hablar de las empresas de luz, agua y gas que pese a que no son tecnologías no dejan de ser suscripciones que debemos pagar y tener en cuenta a la hora de nuestros gastos anuales.

Todo ello va sumando y al final es posible que acabemos con una gran factura por servicios que no disfrutemos. O simplemente que, pese a disfrutar de ellos, el coste que supone su suscripción no nos merece la pena para el servicio que estamos obteniendo.

Otro problema que puede acarrear el estar suscrito a diversos servicios y que es comunmente recurrente en la población, sobretodo en personas de mayor edad, es el problema de que cada suscripción (o la mayoría de ellas) requiere de la creación de una cuenta para poder ser utilizada, lo que conlleva acabar con un número elevado de nombres de usuario y contraseñas que recordar.

1.2. Objetivos

Una vez expuesto su contexto se procede a enumerar una serie de objetivos que tiene que cumplir la aplicación a desarrollar, de forma que todos los problemas expuestos en el apartado anterior sean resueltos:

- Facilitar a los usuarios la obtención de información respecto a sus suscripciones (importe a pagar, fecha en la que se realizará el pago, frecuencia del pago, etc.)
- Facilitar la organización de las suscripciones de los usuarios.
- Proporcionar a los usuarios información acerca de sus gastos anuales y mensuales y de dónde proceden.
- Proporcionar a los usuarios la capacidad de poder consultar las credenciales de sus distintas suscripciones.

Para cumplir con estos objetivos se va a desarrollar una aplicación web, de forma que sea accesible y fácil de utilizar para todo el mundo.

1.3. Organización de la memoria

Para finalizar este apartado introductorio se explica brevemente cual es la organización de la memoria y los distintos puntos de los que consta:

- Se empieza la memoria, en el capítulo 2, exponiendo cómo se realiza el proyecto, en lo que a metodología de trabajo se refiere, es decir, la forma en la que se trabaja para desarrollar la aplicación. A continuación, se definen cuales son las tecnologías y herramientas utilizadas a lo largo del proceso de creación de la aplicación.
- Se continúa con el capítulo 3, en el que se explica como es el comportamiento de la aplicación, las utilidades que ofrece, los requisitos que debe cumplir, cada uno de los casos de uso que puede tener y un primer prototipo de la aplicación web.
- En el capítulo 4, se explica como es la arquitectura de la aplicación y de cada uno de sus componentes.
- En el capítulo 5, se comenta como es la implementación de la aplicación y de cada uno de sus componentes.
- El capítulo 6, explica cómo se realizan las distintas pruebas para verificar el correcto funcionamiento de la aplicación.
- En los capítulos 7 y 8, como es habitual, se finaliza con unas conclusiones sobre el proyecto y la bibliografía.
- Por último, la memoria contiene un anexo en el que se aporta información extra sobre alguno de los apartados de la memoria.

2. TECNOLOGÍAS Y HERRAMIENTAS

Este capítulo, al igual que el anterior, consta de tres secciones en las que se explica, primero, la metodología de desarrollo software que se ha seguido durante la creación de la aplicación, y a continuación, las tecnologías y herramientas empleadas para su creación.

2.1. Metodología de desarrollo

Las metodologías de desarrollo de software [12] son un conjunto de técnicas y métodos organizativos que se aplican para diseñar soluciones de software informático. El objetivo de las distintas metodologías es el de intentar organizar los equipos de trabajo para que estos desarrollen las funciones de un programa de la mejor manera posible.

Cuando se trata de desarrollar productos o soluciones para un cliente o mercado concreto, es necesario tener en cuenta factores como los costes, la planificación, la dificultad, el equipo de trabajo disponible, los lenguajes utilizados, etc. Todos ellos se engloban en una metodología de desarrollo que permite organizar el trabajo de la forma más ordenada posible. El trabajo con una metodología de desarrollo de software permite reducir el nivel de dificultad, organizar las tareas, agilizar el proceso y mejorar el resultado final de las aplicaciones a desarrollar.

Existen varias metodologías de desarrollo pero para este proyecto se ha decidido usar una mezcla entre la metodología llamada "Waterfall (En Cascada)" [12] y la metodología llamada "Incremental" [12]. En la metodología "Waterfall" el desarrollo se divide en varias etapas las cuales deben seguirse en riguroso orden, es decir, no se puede empezar una nueva etapa del desarrollo sin haber completado totalmente la anterior, de ahí su nombre. En este caso, se ha utilizado una versión más ligera de esta metodología, de forma que si durante la fase de Diseño se encontraba alguna mejora necesaria, se podría retroceder de nuevo a la fase de Análisis. Para el proyecto, esta metodología será utilizada para las fases de Análisis y Diseño.

Por otro lado, para la Creación de código y Pruebas se utilizará una metodología "Incremental". Con esta metodología se va construyendo la aplicación final de manera progresiva. En cada etapa incremental se agrega una nueva funcionalidad, lo que permite ver y testear resultados de una forma más rápida en comparación con el modelo en cascada. A continuación, en la Figura 1 se muestra la metodología de desarrollo utilizada.

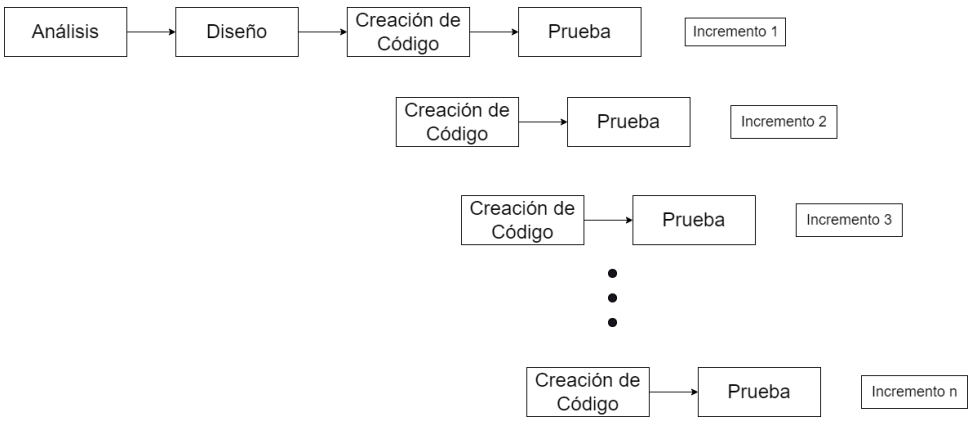

Figura 1: Metodología de desarrollo utilizada

2.2. Tecnologías

2.2.1. React

React [13] es un framework de desarrollo de aplicaciones web de código abierto, creado y mantenido por Facebook. Se centra en la construcción de interfaces de usuario interactivas y eficientes para aplicaciones web y móviles. React se basa en el concepto de componentes reutilizables y se utiliza ampliamente en el desarrollo de interfaces de usuario modernas. Las características clave de React incluyen:

- Componentes: React se basa en la idea de dividir la interfaz de usuario en componentes independientes y reutilizables. Cada componente encapsula una parte específica de la interfaz de usuario y puede contener su propia lógica y estado.
- Virtual DOM: React utiliza un Virtual DOM (Documento de Objeto Modelo) para optimizar la actualización de la interfaz de usuario. En lugar de actualizar el DOM real cada vez que cambia el estado de la aplicación, React compara el Virtual DOM con el DOM real y realiza actualizaciones mínimas, lo que mejora el rendimiento.
- JSX: React permite escribir componentes utilizando JSX (JavaScript XML), una extensión de JavaScript que facilita la definición de la estructura de la interfaz de usuario de manera similar al HTML.
- Unidireccionalidad de datos: React sigue el principio de unidireccionalidad de datos, lo que significa que los datos fluyen en una sola dirección, desde el componente principal a los componentes secundarios. Esto facilita la gestión del estado de la aplicación y la depuración.
- React Router: React proporciona una biblioteca llamada React Router que facilita la gestión de rutas y navegación en aplicaciones de una sola página (SPA).
- Comunidad y Ecosistema: React cuenta con una amplia comunidad de desarrolladores y una gran cantidad de bibliotecas y herramientas complementarias, lo que lo facilita en gran medida la creación de interfaces de calidad y las posibilidades que brinda.

En resumen, React es un framework de desarrollo de aplicaciones web que se destaca por su enfoque en componentes reutilizables, su uso del Virtual DOM para mejorar el rendimiento y su capacidad para crear aplicaciones web interactivas y eficientes. Es ampliamente utilizado en la industria y ha demostrado ser una elección popular para el desarrollo de aplicaciones modernas.

2.2.2. REST

Un servicio REST, o Representational State Transfer (Transferencia de Estado Representacional) [19], es un estilo arquitectónico utilizado en el diseño de sistemas de comunicación en redes, especialmente en el contexto de servicios web y aplicaciones distribuidas. Este estilo arquitectonico es de tipo cliente/servidor, en el cual el servidor publica la información en un formato, normalmente JSON, y esta es accedida por una aplicación cliente que se encargará de procesarla. Estos servicios no se encargan de mantener ningún tipo de estado entre peticiones y cada una de las peticiones es totalmente independiente de la siguiente, lo que significa que cada solicitud HTTP debe contener toda la información necesaria para entender y procesar la solicitud. Los elementos clave que definen un servicio REST son los siguientes:

- Recursos: En un servicio REST, los datos y funcionalidades se modelan como recursos, que son la unidad básica de información. Un recurso puede ser cualquier entidad identificable, como objetos, documentos, o incluso conceptos abstractos. Cada recurso se identifica mediante una URI (Uniform Resource Identifier).

- Verbos HTTP: Los servicios REST utilizan los verbos HTTP estándar, como GET, POST, PUT y DELETE, para realizar operaciones sobre los recursos. Por ejemplo, GET se utiliza para recuperar información del estado de un recurso en el servidor, POST y PUT se utilizan para la creación de nuevos recursos en el servidor, con la diferencia de que con POST el servidor es el que está encargado de decidir la URI del nuevo recurso y con PUT es el cliente el que debe proveer al servidor de la URI y DELETE para eliminar un recurso del servidor.
- Representaciones: Como se ha mencionado antes, el servidor publica la información solicitada por el cliente en diferentes representaciones , como texto, XML, JSON, HTML, entre otros. Los clientes pueden especificar el formato deseado al interactuar con el servicio a través de las cabeceras HTTP.

2.2.3. MySQL

MySQL [16] es un sistema de gestión de bases de datos relacional de código abierto ampliamente utilizado en aplicaciones web. Está basado en el modelo de bases de datos relacionales, lo que significa que almacena los datos en tablas que se relacionan entre sí mediante claves primarias y foráneas. Utiliza SQL como lenguaje para interactuar con la base de datos. Los desarrolladores pueden utilizar sentencias SQL para crear, leer, actualizar y eliminar datos, así como para realizar consultas complejas.

MySQL es compatible con varias plataformas, lo que significa que se puede ejecutar en sistemas operativos como Linux, Windows y macOS. Esto lo hace versátil y ampliamente utilizado en una variedad de entornos.

2.2.4. Git

Git [20] es un sistema de control de versiones distribuido ampliamente utilizado en el desarrollo de software. Es una tecnología que permite rastrear y gestionar cambios en el código fuente de proyectos de programación. Permite a los desarrolladores tener un seguimiento de los cambios en el código fuente a lo largo del tiempo. Cada cambio se registra en un historial lo que permite hacer un registro de la auditoría de los cambios en el código.

Git es un sistema de control de versiones distribuido, lo que significa que cada copia del repositorio Git contiene toda la información histórica y la capacidad de funcionar de forma autónoma. Permite la creación de ramas en las que los desarrolladores pueden implementar nuevas funcionalidades o arreglar errores sin que esto afecte a la rama principal y una vez hechas las comprobaciones de que lo realizado es correcto se puede unir a la rama principal haciendo que esta se actualice. También aporta herramientas para gestionar los conflictos cuando varios desarrolladores realicen cambios sobre las mismas partes del código de forma que puedan resolverse de manera efectiva.

2.3. Herramientas

2.3.1. Visual Studio Code

Visual Studio Code (VS Code) [14] es un editor de código fuente desarrollado por Microsoft. Es software libre y multiplataforma, está disponible para Windows, Linux y macOS. VS Code tiene una buena integración con Git, cuenta con soporte para depuración de código, y dispone de un gran número de extensiones, que básicamente te da la posibilidad de escribir y ejecutar código en cualquier lenguaje de programación. Es la herramienta utilizada para la creación de la interfaz en React.

2.3.2. IntelliJ IDEA

IntelliJ IDEA [9] es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains. Se enfoca principalmente en el desarrollo de aplicaciones Java, aunque también admite otros lenguajes de programación y tecnologías. IntelliJ IDEA es conocido por su potente conjunto de herramientas y características diseñadas para mejorar la productividad de los desarrolladores, anque es conocido por su compatibilidad con Java, tambien acepta otros lenguajes como Kotlin, Groovy, Scala, JavaScript, TypeScript, HTML, CSS y más. Es la herramienta utilizada para la creación del servicio REST.

2.3.3. XAMPP Control Panel

XAMPP [2] es un paquete de software de código abierto que facilita la configuración de un entorno de desarrollo web local en sistemas Windows, macOS y Linux. Es un paquete de software libre, que consiste principalmente en el sistema de gestión de bases de datos MySQL, el servidor web Apache y los intérpretes para lenguajes de script PHP y Perl. Es la herramienta utilizada para gestionar la base de datos MySQL.

2.3.4. draw.io Diagrams

draw.io Diagrams [10] es un software de dibujo gráfico multiplataforma desarrollado en HTML5 y JavaScript. Su interfaz se puede utilizar para crear diagramas como diagramas de flujo, estructuras alámbricas, diagramas UML, organigramas y diagramas de red. Es la herramienta utilizada para crear los diagramas de la memoria.

2.3.5. Postman

Postman [17] es una herramienta que permite a los desarrolladores probar, documentar y colaborar en la construcción de APIs. Se utiliza comúnmente en el desarrollo de aplicaciones web y servicios para simplificar y agilizar el proceso de pruebas y comunicación con APIs. Proporciona una interfaz de usuario intuitiva y fácil de usar que permite a los desarrolladores crear y enviar solicitudes HTTP a APIs, lo que facilita la prueba y la depuración.

2.3.6. Axure

Axure RP (Axure Rapid Prototyping) [3] es una poderosa herramienta de diseño y prototipado de interacción utilizada para crear prototipos de alta fidelidad y especificaciones de diseño de sitios web y aplicaciones móviles. Está diseñada para ayudar a los diseñadores y profesionales de la experiencia de usuario (UX) a visualizar y comunicar sus ideas de diseño de manera efectiva. Es la herramienta utilizada en la creación del prototipo de la aplicacion web.

2.3.7. Fork

Fork [6] es un cliente de Git que permite a los desarrolladores trabajar con Git de una manera mucho más visual y cómoda. No requiere escribir comandos manualmente, ofreciendo una cómoda interfaz gráfica con todas las opciones integradas. De esta forma, se pueden realizar las tareas de forma más rápida.

3. DESCRIPCIÓN DEL SISTEMA Y ANÁLISIS DE REQUISITOS

Siguiendo la metodología de desarrollo en cascada comenzamos el proyecto haciendo un análisis del mismo. En esta sección se exponen las distintas funcionalidades del sistema y su comportamiento. También se hace un análisis de sus requisitos, tanto funcionales como no funcionales, los distintos casos de uso que tiene la aplicación y también se ofrece un primer prototipo del aspecto final de la aplicación.

3.1. Descripción de la aplicación

En la aplicación habrá tres apartados claramente diferenciados, a través de los que se podrá acceder a distinta información sobre las subcripciones del usuario.

Por un lado, contaremos con una página principal, en la que se listarán todas las suscripciones actualmente vigentes, es decir, las que el usuario haya incluido en el sistema. Estas suscripciones se podrán ordenar tanto como por fecha de cobro como por importe a pagar, tanto en sentido ascendente como descendente. A su vez, en esta misma página contaremos con una estructura en forma de árbol, que permitirá al usuario la organización de las distintas suscripciones en distintas carpetas de forma que pueda ordenarlas y gestionarlas como más le guste.

A través de esta página principal, podremos acceder a los detalles de cada una de las suscripciones. Como se ha mencionado antes, uno de los objetivos de la aplicación es que el usuario pueda guardar de forma segura las credenciales de sus suscripciones por lo que mediante o bien la estructura en árbol o mediante la propia lista de suscripciones el usuario podrá acceder a una nueva página con todo tipo de detalles sobre la suscripción elegida.

Aparte de estas dos páginas, el sistema contará con una tercera en la que el usuario podrá consultar un resumen en forma de gráfico de barras de los gastos anuales divididos en meses provenientes de todas sus suscripciones. El gráfico contará con una leyenda de colores identificativos para cada suscripción y en él se podrá ver claramente la suma del importe total pagado o por pagar que supone el total de suscripciones así como una etiqueta con el importe específico de cada una de ellas. Este gráfico se podrá filtrar tanto por año como por cuenta bancaria ya que a la hora de añadir una suscripción se podrá especificar una cuenta bancaria, por si el usuario tiene sus gastos divididos. A su vez en esta misma página podremos alternar entre dos vistas del gráfico distintas, una vista será la ya comentada anteriormente, es decir, un gráfico de barras con la suma del importe total por mes y una nueva vista en la que se mostrará un gráfico circular con el porcentaje del importe pagado o a pagar por cada suscripción con respecto del total anual.

Todas estas páginas comparten una barra de navegación que nos permitirá movernos por la aplicación, así como cerrar sesión y acceder a los ajustes del usuario y la aplicación.

Para poder acceder a a la aplicación el usuario deberá tener dada de alta una cuenta en el sistema. Esto se podrá hacer mediante un formulario, en el que usuario facilitará una serie de datos necesarios para crear su usuario. Una vez creada esta cuenta, podrá acceder a la aplicación mediante un formulario de inicio de sesión.

3.2. Ánalisis de Requisitos

El análisis de requisitos [1] se define como “El proceso del estudio de las necesidades de los usuarios para llegar a una definición de los requisitos del sistema, de hardware o de software, así como el proceso de estudio y refinamiento de dichos requisitos”. Esta etapa del proyecto es de las más importantes ya que no hacer un buen análisis de requisitos puede acarrear que el coste del proyecto aumente significativamente.

3.2.1. Actores del Sistema

Para definir tanto los requisitos funcionales como no funcionales primero debemos especificar quienes son los tipos de usuarios o en este caso los roles que van a ejercer los usuarios dentro de la aplicación. Dado que este proyecto está pensado para que sea de utilidad para suscripciones personales tan solo contaremos con dos tipos de roles, que se exponen en la Tabla 1.

Tabla 1: Actores del Sistema

Rol	Descripción
Usuario Cliente	Este es el usuario al que va dedicada la aplicación y que disfruta de todas las funcionalidades que brinda, es decir, poder añadir suscripciones, visualizarlas, editarlas,etc. Tiene todo el poder en su espacio de trabajo.
Usuario Anónimo	Este es el tipo de usuario que todavía no tiene acceso al sistema ya sea porque no está dado de alta o porque todavía no ha iniciado sesión. Sus únicas opciones disponibles son registrarse, iniciar sesión en el sistema o en caso de que tenga un usuario pero no recuerde la contraseña, modificar su antigua contraseña.

3.2.2. Requisitos Funcionales

Unos vez identificados los roles del sistema, se procede a identificar cuales son sus requisitos funcionales. Los requisitos funcionales se definen en [4] como "descripciones explícitas del comportamiento que debe tener una solución de software y que información debe manejar.". En la Tabla 2 se recogen los requisitos funcionales del sistema de la aplicación.

Tabla 2: Requisitos Funcionales

Id	Requisito	Descripción
RF01	Acceso de usuario	El usuario podrá iniciar sesión en el sistema utilizando su email y contraseña.
RF02	Edición de perfil	El usuario podrá consultar y editar los datos de su cuenta.
RF03	Envío de resumenes	El usuario podrá recibir a principio de cada mes un correo con el resumen de gastos que se prevén.
RF04	Visualización de suscripciones	El usuario podrá visualizar un listado con las suscripciones.
RF05	Visualización de árbol de suscripciones	El usuario podrá visualizar una organización de suscripciones en carpetas.
RF06	Calculo de fechas de cobro	El usuario podrá visualizar la fecha de cobro de cada suscripción.

Tabla 2: Requisitos Funcionales

Id	Requisito	Descripción
RF07	Visualización del importe de la suscripción	El usuario podrá visualizar el importe a pagar de cada suscripción.
RF08	Orden de suscripciones	El usuario podrá ordenar las suscripciones en base al nombre de la aplicación, el importe que se cobrará o la fecha de cobro.
RF09	Visualización del detalle de suscripción	El usuario podrá visualizar detalladamente la información específica de cada suscripción, aparte de lo ya mostrado en la lista.
RF10	Visualización de los detalles del usuario de la suscripción	El usuario podrá visualizar en los detalles de la suscripción los datos de la cuenta asociada a esa suscripción, es decir, su correo, contraseña o nombre de usuario.
RF11	Copia de los detalles del usuario de la suscripción	El usuario podrá copiar al portapapeles los datos de la cuenta asociada a una suscripción, es decir, su correo, contraseña o nombre de usuario.
RF12	Edición de los detalles del usuario de la suscripción	El usuario podrá editar los datos de la cuenta asociada a una suscripción, es decir, su correo, contraseña o nombre de usuario.
RF13	Creación de carpetas	El usuario podrá crear carpetas en el arbol de suscripciones.
RF14	Eliminación de carpetas	El usuario podrá eliminar una carpeta del árbol de suscripciones.
RF15	Creación de suscripciones	El usuario podrá crear suscripciones.
RF16	Eliminación de suscripciones	El usuario podrá eliminar suscripciones.
RF17	Acceso a web de la suscripción	El usuario podrá acceder a la página web de la suscripción para poder darla de baja.
RF18	Navegación	El usuario podrá navegar por el sistema a través de un menú.
RF19	Visualización de gráfico resumen	El usuario podrá visualizar un gráfico de barras con el resumen de gastos anual de cada mes.
RF20	Visualización de gráfico porcentaje	El usuario podrá visualizar un gráfico circular con el importe total de cada suscripción y el porcentaje que supone del total.
RF21	Filtrado de datos	El usuario podrá filtrar los gráficos en base a unos parámetros seleccionados, como son, el año o la cuenta bancaria asociada a una suscripcion.
RF22	Visualización alterna de gráficos	El usuario podrá alternar la vista entre los dos gráficos.
RF23	Tutorial	El usuario podrá visualizar un tutorial que enseñe como usar la aplicación.

3.2.3. Requisitos No Funcionales

Los requisitos no funcionales [4] son las restricciones software que le imponemos al sistema de forma que podamos garantizar su calidad. Están divididos en distintos tipos como son: Fiabilidad, Seguridad, Rendimiento, Mantenimiento, Usabilidad y Accesibilidad. En este caso, los requisitos no funcionales definidos para la aplicación son los que se muestran en la Tabla 3.

Tabla 3: Requisitos No Funcionales

Id	Descripción	Categoría
RNF01	El sistema se adaptará al tamaño de la ventana y del navegador.	Accesibilidad
RNF02	El sistema encriptará la contraseña de cada usuario a la hora de registrarlo en la base de datos.	Seguridad
RNF03	El sistema encriptará los datos sensibles de cada suscripción alojados en los detalles.	Seguridad
RNF04	Solo se podrá acceder al sistema con un usuario y una contraseña alojados en la Base de Datos.	Seguridad
RNF05	El sistema usará menús facilmente reconocibles e intuitivos.	Usabilidad

3.2.4. Especificación de Casos de Uso

En esta sección se hablará de los distintos casos de uso que tendrá la aplicación, así como una serie de plantillas explicando el funcionamiento de cada uno de los casos con mayor detalle, para facilitar su comprensión.

3.2.4.1 Diagrama de Casos de Uso

A continuación, se detallarán los distintos casos de uso que pueden realizar los distintos actores en el sistema. En la Figura 2, se visualizan los casos de uso del usuario Anónimo y en las Figuras 3 y 4 se visualizan los casos de uso del usuario Cliente, separados en dos diagramas para mejorar su legibilidad.

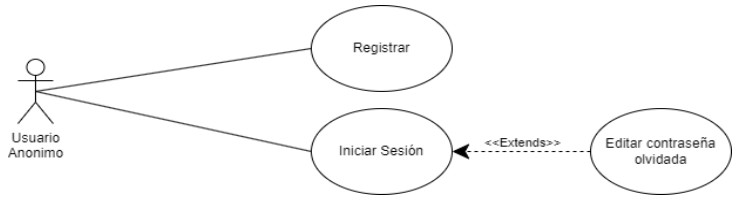

Figura 2: Casos de Uso para rol Usuario Anónimo

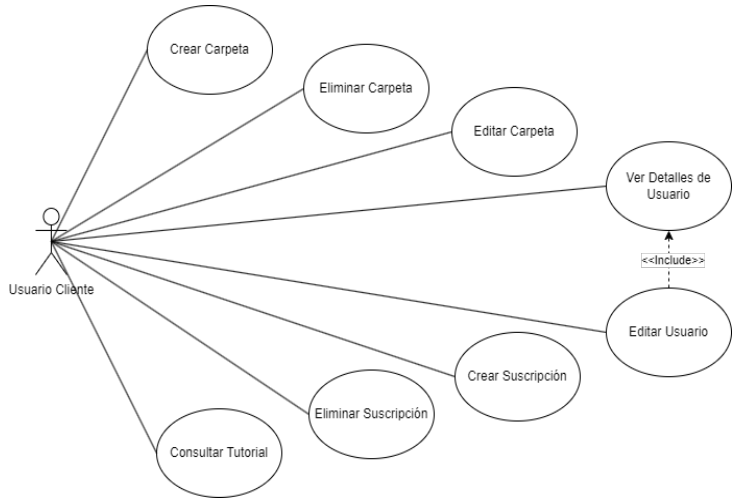

Figura 3: Casos de Uso para rol Usuario Cliente I

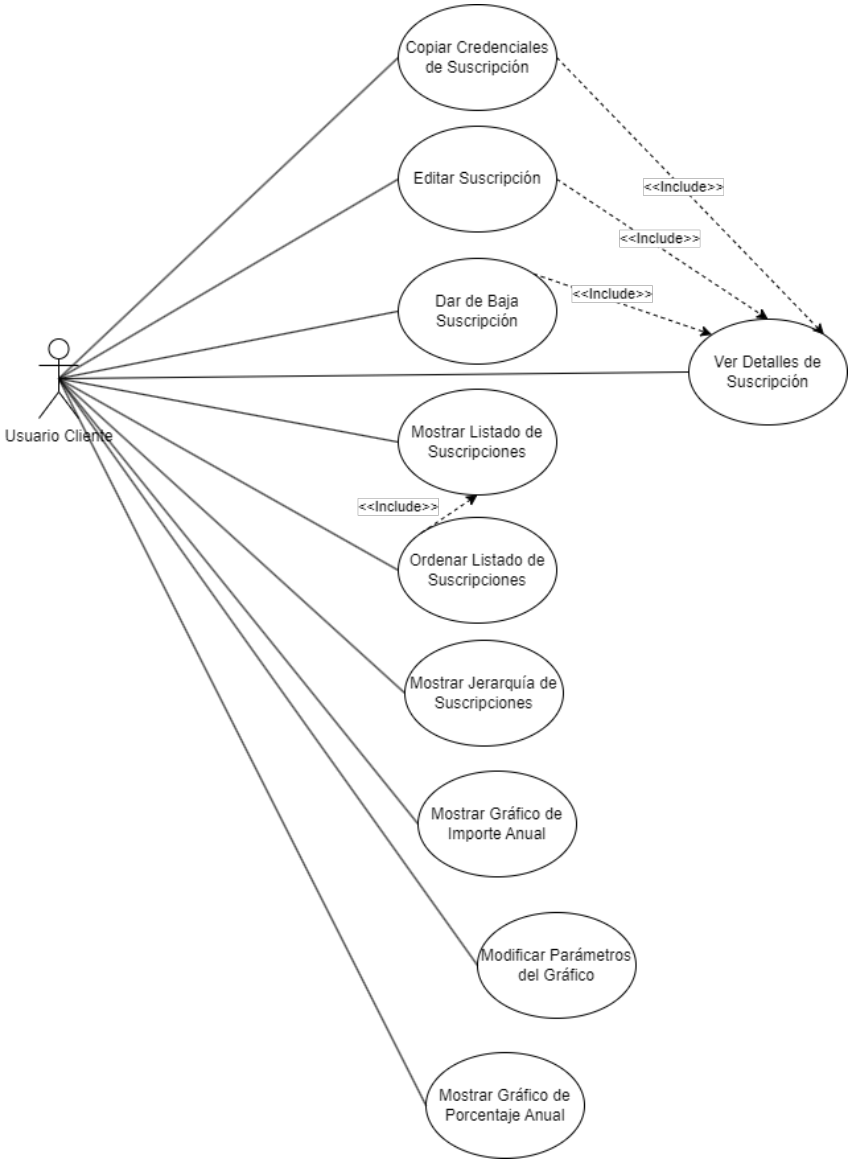

Figura 4: Casos de Uso para rol Usuario Cliente II

3.2.4.2 Plantillas de Casos de Uso

A continuación se describe más en detalle los distintos casos de uso mediante plantillas, para facilitar su comprensión. Dado el gran número de casos de uso, en este apartado se expondrán los que se consideran que son los más importantes, como son la creación o eliminación de una carpeta o suscripción, o ver los datos del usuario y editarlos. El resto de plantillas de casos de uso se complementarán en el anexo.

Tabla 4: Plantilla de Crear Carpeta

Nombre	Crear Carpeta
Id	PCU04
Descripción	El usuario crea una nueva carpeta.
Precondiciones	El usuario debe haber iniciado sesión en el sistema.

Flujo Principal	1. El usuario mantiene el puntero del ratón sobre en el boton "+". 2. El sistema desplegará una serie de opciones entre las que está "Añadir Carpeta". 3. El usuario pulsa en el botón "Añadir Carpeta". 4. El sistema mostrará un formulario con dos campos a rellenar (Nombre y Carpeta Padre). 5. El usuario introduce la información y pulsa en "Crear Carpeta". 6. El sistema crea la carpeta con los datos indicados. 7. Se carga la página principal del usuario (Listado de Suscripciones) con la nueva carpeta creada en el árbol de carpetas.
Flujos Alternativos	6.1 El sistema muestra un mensaje comunicando un error a la hora de crear la carpeta ya que ya existe una carpeta con ese mismo nombre.

Tabla 5: Plantilla de Eliminar Carpeta

Nombre	Eliminar Carpeta
Id	PCU05
Descripción	El usuario elimina una carpeta.
Precondiciones	El usuario debe haber creado la carpeta previamente y debe estar vacía.
Flujo Principal	1. El usuario pulsa en el boton "..." en la carpeta elegida en el árbol de suscripciones. 2. El sistema muestra una opción de eliminar carpeta. 3. El usuario pulsa en la opción "Eliminar Carpeta". 4. El sistema elimina la carpeta seleccionada. 5. Se carga la página principal del usuario (Listado de Suscripciones) con el árbol de carpetas actualizado.

Flujos Alternativos	4.1 El sistema muestra un mensaje comunicando un error a la hora de eliminar la carpeta porque no está vacía.

Tabla 6: Plantilla de Editar Usuario

Nombre	Editar Carpeta
Id	PCU07
Descripción	El usuario edita los detalles de su cuenta.
Precondiciones	El usuario debe haber iniciado sesión.
Flujo Principal	1. Include (Ver Detalles de Usuario). 2. El usuario pulsa en el botón "Editar" para editar los detalles del usuario. 3. El usuario edita sus detalles (nombre, foto de perfil, contraseña, opción de recibir correos mensuales con resúmenes). 4. El usuario guarda los cambios. 5. El sistema muestra los nuevos detalles de la cuenta.
Flujos Alternativos	5.1a El sistema muestra un mensaje comunicando que no se han podido editar los detalles del usuario debido a que ha dejado campos obligatorios vacíos como el nombre, el email o la contraseña. 5.1b El sistema muestra un mensaje comunicando que no se han podido cambiar los detalles del usuario debido a que el nombre de usuario introducido por el usuario ya existe.

Tabla 7: Plantilla de Crear Suscripción

Nombre	Crear Suscripción
Id	PCU08
Descripción	El usuario crea una nueva suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una carpeta creada.

Flujo Principal	1. El usuario mantiene el puntero del ratón sobre en el boton "+". 2. El sistema desplegará una serie de opciones entre las que está "Añadir Suscripción". 3. El usuario pulsa en el boton "Añadir Suscripción". 4. El sistema mostrará un formulario con varios campos a rellenar, como nombre, importe, fecha de contratación, la periodicidad de la suscripción, los datos del usuario de la suscripción y un desplegable con todas las carpetas creadas previamente. 5. El usuario introduce los datos solicitados en el formulario y pulsa en el botón "Crear Suscripción". 6. El sistema crea una nueva suscripción con los datos introducidos. 7. Se muestra la página principal (Listado de Suscripciones) con la nueva suscripción añadida.
Flujo Alternativo	6.1a El sistema muestra un mensaje comunicando un error a la hora de crear la suscripción ya que se han dejado campos obligatorios vacíos como el nombre, el importe, la fecha de contratación, la periodicidad o la carpeta en la que va introducida. 6.1b El sistema muestra un mensaje comunicando un error a la hora de crear la suscripción ya que hay errores en algunos campos, como, ya existe una suscripción con ese nombre, la fecha de contratación es posterior al día de hoy, el importe o la periodicidad son valores negativos, la periodicidad no es un valor entero o los ultimos 4 dígitos de la cuenta bancaria no son un número entero de 4 cifras.

Tabla 8: Plantilla de Eliminar Suscripción

Nombre	Eliminar Suscripción
Id	PCU09
Descripción	El usuario elimina una suscripción.
Precondiciones	El usuario debe tener la suscripción creada en el sistema.

Flujo Principal	1. El usuario pulsa en el boton "..." en la suscripción elegida en el árbol de suscripciones. 2. El sistema muestra una opción de eliminar suscripción. 3. El usuario pulsa en la opción "Eliminar Suscripción". 4. El sistema elimina la suscripción seleccionada. 5. Se carga la página principal del usuario (Listado de Suscripciones) con el árbol de carpetas y la lista de suscripciones actualizadas.

3.3. Primer prototipo de la web

Para mejorar la fase de captura y definición de requisitos, se decidió elaborar un prototipo de la aplicación, utilizando para ello el programa Axure [3], que permite la realización de este tipo de tareas de una forma sencilla, pero con gran capacidad.

Se comenzará mostrando directamente, con la Figura 5, la interfaz de la web una vez se haya iniciado sesión, ya que antes de eso simplemente serán formularios tanto de registro como de acceso.

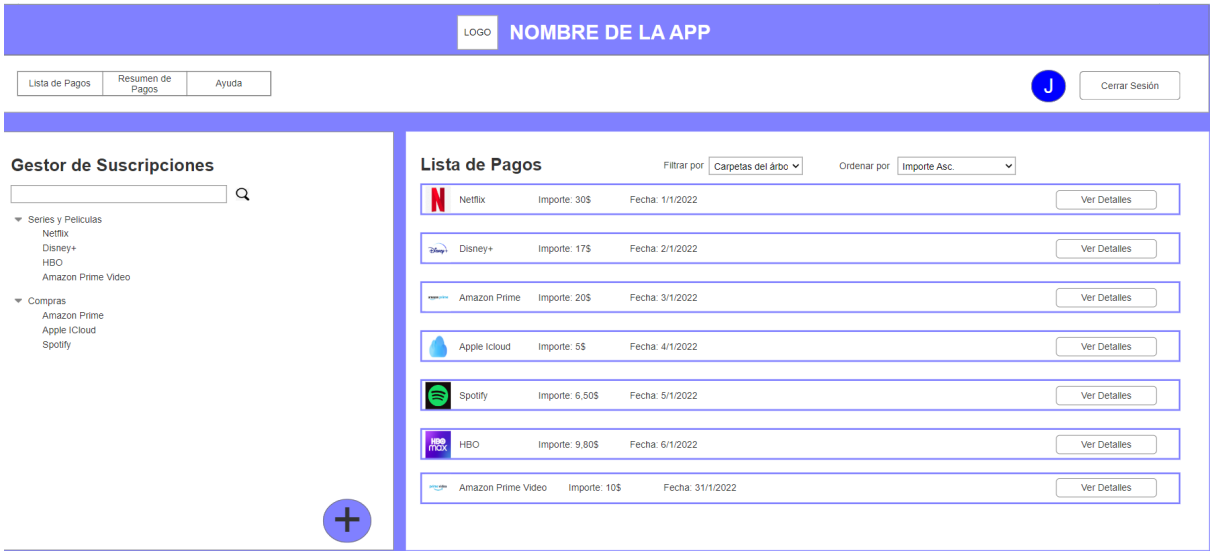

Figura 5: Prototipo Listado de Suscripciones y Jerarquía de Suscripciones

Esta es la idea de la página principal, en la que, como se puede apreciar, aparecen listadas las suscripciones, pudiendose ordenar como el usuario prefiera. A su izquierda, se visualiza la agrupación en carpetas en un esquema de árbol. Sobre estos dos visuales, la página contiene un menú de navegación para alternar entre las diferentes pestañas de la web, un botón con la imagen del usuario para poder acceder a sus ajustes y por último un botón para cerrar sesión.

Por otro lado, en la Figura 6 se puede apreciar como es el diseño de los detalles de cada suscripción, al que se puede acceder pulsando en el botón "Ver Detalles" en la lista de suscripciones o pulsando sobre la suscripción en el árbol de suscripciones. En esta página se recoge toda la información relacionada con la suscripción pudiendose editar en cualquier momento al hacer click sobre el botón "Editar".

Figura 6: Prototipo Detalle de Suscripción

Por último, tenemos la pestaña de los gráficos a la que se puede acceder desde el menú de navegación pulsando en el botón "Resumen de Pagos". Esta pestaña muestra por defecto el gráfico de barras (Figura 7) con unos parámetros preestablecidos y mediante el botón, que cambia de texto en función del gráfico que se está mostrando, se cambia la vista alternando el gráfico para visualizar la otra versión (Figura 8). En la parte de arriba habrá un par de campos en los que se podrán establecer los parámetros "Año" y "Últimos dígitos de la cuenta bancaria" para poder modificar el gráfico que se está visualizando.

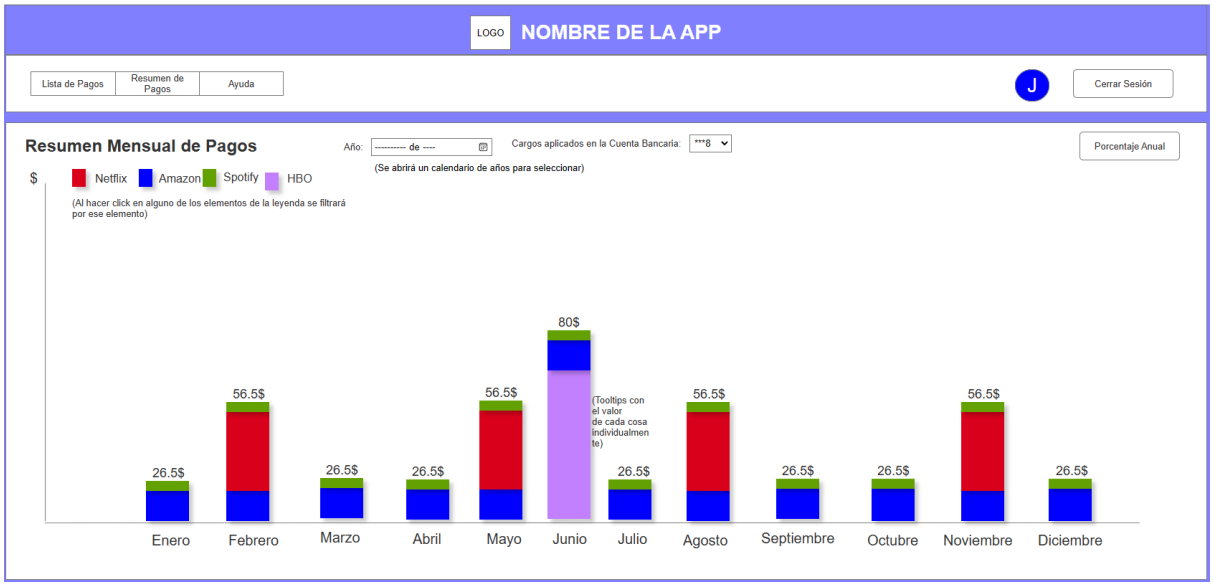

Figura 7: Prototipo Gráfico Importe Anual

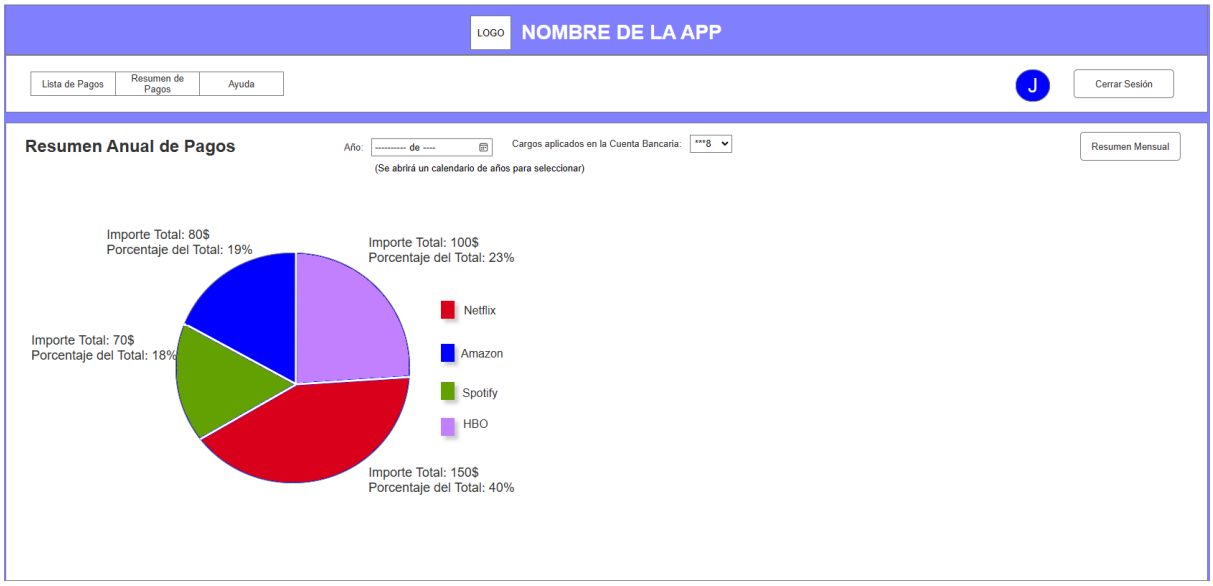

Figura 8: Prototipo Gráfico Porcentaje Anual

4. DISEÑO SOFTWARE

Continuando con nuestra metodología de trabajo llega el turno de crear el diseño de nuestra aplicación. En este apartado se expone como está diseñado el proyecto para cumplir con todos los requisitos encontrados en la fase de análisis. Se habla de su arquitectura, el diseño de la base de datos con la que se conecta, el servicio web que se usa y como está diseñada la aplicación web.

4.1. Diseño del sistema

El diseño de la arquitectura es una fase fundamental en el proceso de crear una aplicación, hay que tener en cuenta a la hora de tomar decisiones que el diseño escogido cumpla los requisitos funcionales y no funcionales establecidos. Existen varios modelos o patrones que se pueden aplicar y que ofrecen buenos resultados. De entre todos ellos, en este caso, se ha elegido una arquitectura en tres capas. Esta arquitectura permite separar la aplicación en tres capas independientes, tal y como se muestran en la Figura 9, asignando un rol específico a cada una:

- Capa de presentación: Esta capa permite la interacción con el usuario. Es la encargada de comunicarse con la capa de negocio para obtener la información que el usuario solicita y presentársela en una interfaz. Esta capa estará formada por la interfaz web.
- Capa de negocio: Esta capa es la que contendrá toda la lógica del negocio de la aplicación. Será la encargada de recibir las peticiones del usuario y devolver a la capa de presentación los resultados obtenidos y también comunicarse a su vez con la capa de datos en caso de que sea necesario para la lectura o el almacenamiento de algún dato. Esta capa estará formada por el servicio REST.
- Capa de datos: Esta capa será la encargada de almacenar todos los datos del sistema y se comunicará con la capa de negocio para leer o almacenar datos. Puede estar formada por uno o más gestores de bases de datos. Esta capa estará formada por una base de datos empleando el gestor MySQL.

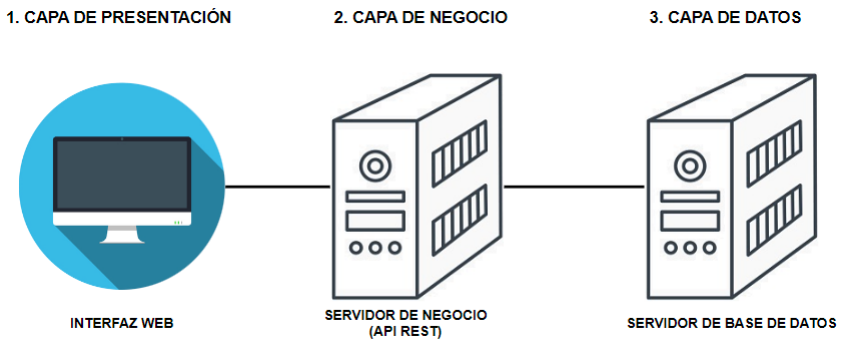

Figura 9: Arquitectura del sistema

4.2. Diseño de la API REST

La capa de negocio consiste en un componente intermedio entre lo que el usuario ve e interactúa y el almacenamiento de los datos. Es una capa invisible, que contiene toda la lógica de negocio, y que en este caso, se diseña en forma de servicio REST. Para el diseño de este tipo de servicios se han de cumplir una serie de pautas:

- El servicio se diseña en base al manejo de recursos. Los recursos son cada una de las entidades que representan los conceptos de negocio.

- El servicio debe estar basado en URIs. A cada recurso se le debe asignar un URI distinto que lo identifica unívocamente.
- El acceso a los recursos que proporciona el servicio debe seguir los métodos HTTP estándar:
  - GET: Es utilizado únicamente para consultar información de la base de datos, muy parecido a realizar un SELECT a la base de datos.
  - POST: Es utilizado para solicitar la creación de un nuevo recurso, es decir, algo que no existía previamente, es equivalente a realizar un INSERT en la base de datos.
  - PUT: Se utiliza para actualizar un recurso existente, es decir, es parecido a realizar un UPDATE a la base de datos.
  - DELETE: Este método se utiliza para eliminar un recurso existente, es similar a DELETE a la base de datos.

  Se deberá especificar, para cada URI, cuales son los métodos HTTP que soporta.
- El servicio debe transferir la información utilizando los tipos de formatos válidos en Internet, habitualmente JSON o XML.

A continuación, en la Figura 10, se expone la interfaz del servicio REST diseñado, esto es, se especifican los recursos que lo forman, sus URIs y los métodos HTTP que soportan.

Recurso	URI	Métodos
Lista de Usuarios	URIBase/users	GET
Usuario	URIBase/users/{userId}	GET PUT DELETE
Usuario	URIBase/users	POST
Lista de Carpetas	URIBase/users/{userId}/folders	GET
Carpeta	URIBase/users/{userId}/folders/{folderId}	GET PUT DELETE
Carpeta	URIBase/users/{userId}/folders	POST
Lista de Suscripciones	URIBase/users/{userId}/subscriptions	GET
Suscripción	URIBase/users/{userId}/subscriptions/{subscriptionId}	GET
Suscripción	URIBase/users/{userId}/folders/{folderId}/subscriptions	POST
Suscripción	URIBase/users/{userId}/folders/{folderId}/subscriptions/{subscriptionId}	PUT DELETE
Token	URIBase/token	POST

Figura 10: Recursos del servicio

En cuanto a la representación de la información en un formato válido, en la Figura 11 se puede contemplar, a modo de ejemplo, como sería la representación en formato JSON del recurso Usuario. Se ha elegido este recurso por ser el más complejo y englobar representaciones de los otros dos recursos del sistema.

Para la representación de la arquitectura interna del servicio REST se ha modelado el diagrama de componentes que se muestra en la Figura 12. El servicio cuenta con cuatro componentes de tipo Controller, uno para cada entidad y un controlador extra denominado “AuthController” para hacer login o registrarse en la aplicación.

```
{
    "userId": 2,
    "name": "Jorge",
    "surname": "",
    "email": "prueba@email.com",
    "messages": true,
    "folders": [
        {
            "folderId": 5,
            "name": "Carpeta 1",
            "subFolders": [],
            "subscriptions": [
                {
                    "subscriptionId": 5,
                    "name": "Netflix",
                    "price": 20.0,
                    "contractDate": "11/09/2000",
                    "subscriptionFrequency": 3,
                    "subscriptionUsername": "jorgerl",
                    "subscriptionPassword": "passprueba",
                    "subscriptionEmail": "admin@email.com",
                    "lastDigitsBank": "1234",
                    "subscriptionComments": "Texto de comentarios de prueba"
                }
            ]
        }
    ]
}
```


Figura 11: Representación JSON del recurso Usuario

Estos controladores son los encargados de recibir las distintas peticiones desde la capa de presentación y a través de las distintas interfaces interactuar con la capa de datos. Además, el servicio cuenta con un componente extra para gestionar aspectos relacionados con la seguridad, utilizando para ello tokens JWT [15]. El componente "AuthController" se encarga de gestionar los procesos de autenticación y registro en la aplicación. Para ello, hace uso del componente "UserAuthProvider", encargado tanto de la creación como de la validación de los tokens JWT. Utilizando este mecanismo, solo se pueden obtener datos una vez se haya validado que quien solicita la información es un usuario de la aplicación y se haya creado un token que deberá ser incluido en la cabecera de todas las peticiones HTTP que se realicen al servicio.

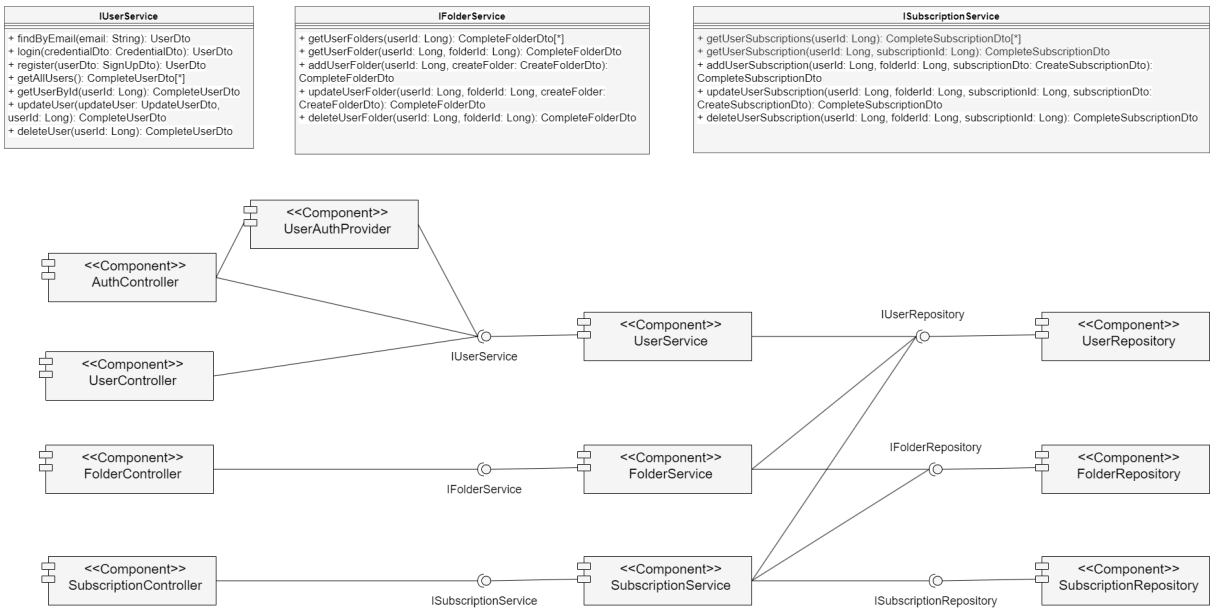

Figura 12: Diagrama de componentes del servicio REST y sus interfaces

En el diagrama de la Figura 12 solo se muestra en detalle la especificación de las interfaces de servicio, ya que las interfaces de los repositorios cuentan con los métodos característicos de una capa de acceso a datos.

4.2.1. Modelo de clases

Para complementar el diseño del servicio REST se ha creado el modelo de datos representado en la Figura 13 en la que se pueden observar como se relacionan las entidades del dominio de la aplicación.

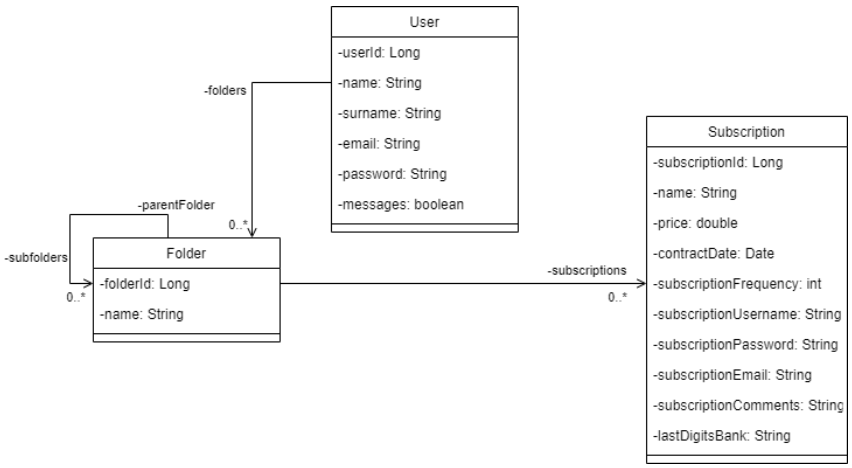

Figura 13: Diagrama de clases

Debido a que la tecnología usada en la creación del servicio REST lo permite, la base de datos se creará automáticamente al desplegar el servicio REST, a partir de este modelo de dominio, utilizando las anotaciones propias del framework JPA.

4.3. Diseño del Frontend

La capa de presentación, por otro lado, es la parte visible de nuestra aplicación, es decir, la interfaz web. Es lo que el usuario ve y con lo que usuario interacciona. Consiste en una interfaz gráfica que muestra la información de la base de datos y es la encargada de capturar las peticiones del usuario para comunicárselas a la capa de negocio. Esta capa de presentación se comunica únicamente con la capa de negocio, es decir, con el servicio REST.

Para el desarrollo de la aplicación web se ha decidido desarrollar una solución SPA. Una web SPA (Single Page Application) es una forma de desarrollo web en la que la página web está contenida en un único archivo HTML. Mientras navegamos por la web, se irán solicitando contenidos al servidor y se irán renderizando los contenidos dinámicamente dependiendo de las rutas, de esta forma se mejoran los tiempos de respuesta y por tanto la experiencia de usuario.

En cuanto al diseño de la aplicación web, ya que no vamos a realizar ningún tipo de lógica de negocio en ella y debido a la tecnología con la que va a ser implementada, se organizará como una estructura de componentes, en la que se dividirán los distintos componentes de la web en función de su naturaleza o utilidad.

Para la comunicación del frontend con el servicio REST se crea una capa denominada “service” en el frontend, la cual será la encargada de realizar las peticiones HTTP al servicio REST y de entregar los datos obtenidos a los componentes de la interfaz que soliciten la información. Para ver como se realiza la comunicación entre los componentes del frontend, el componente de servicio del frontend y el servicio REST se ha diseñado el diagrama mostrado en la Figura 14, en el que se puede ver qué componentes hacen uso del componente servicio y como este realiza las peticiones al servicio REST.

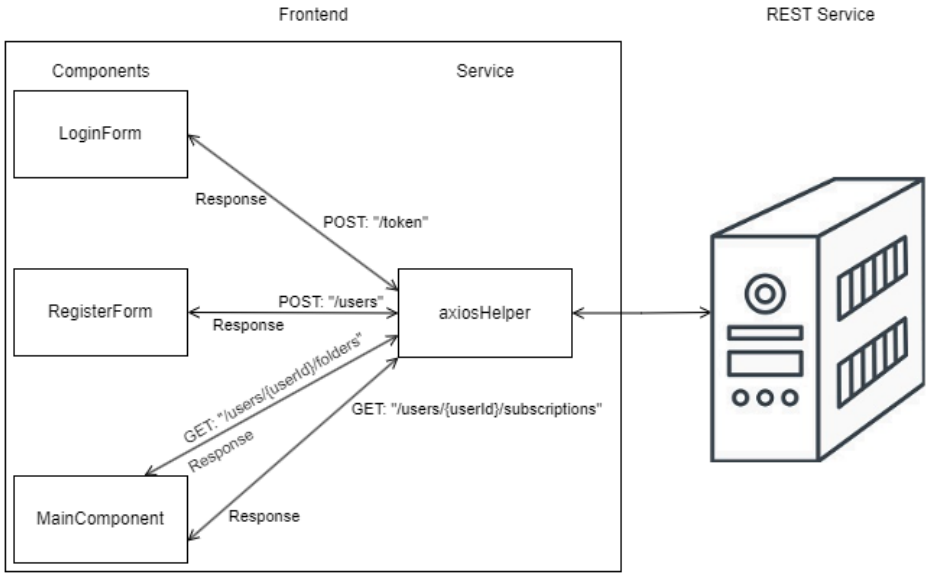

Figura 14: Diagrama de componentes del frontend

5. CREACIÓN DE CÓDIGO

Siguiendo la metodología de trabajo ha llegado el momento de comenzar con la creación de código. Como se menciona previamente esta fase del desarrollo se realiza de manera incremental comenzando con el desarrollo del servicio REST y finalizando con el desarrollo de la aplicación web. En este apartado se describe el proceso de codificación de la aplicación así como una pequeña explicación de las tecnologías utilizadas durante el mismo.

5.1. Backend

Como se menciona previamente el backend se creó con Spring Boot [22], que es un framework de Java que facilita la creación de servicios REST, entre otras cosas. Para su uso, solo es necesario un IDE y tener instalada alguna versión de JDK. En este caso se hizo uso el IDE "Intellij IDEA".

Una vez se instaló todo, se pudo comenzar con la creación del backend, para ello, se accedió a la web "Spring Initializr" donde se realizó una primera configuración del proyecto, así como la adición de las dependencias que tiene. En este caso, el proyecto utiliza Maven, que, entre otras cosas, facilita la gestión de las dependencias del proyecto, que principalmente son: Spring Data JPA, Lombok, MapStruct, Spring Web MVC, Spring Security y MySQL Driver, que permite la conectividad a la base de datos. Una vez terminada la configuración, la web permite descargar en un archivo zip el "esqueleto" del código fuente. Una vez descargado el código fuente se comenzó con el desarrollo del servicio REST. Siguiendo la estructura de componentes definida en el apartado de diseño, lo primero que se realizó fue la creación de una estructura en carpetas que facilitase la separación de los distintos tipos de componentes. Esta estructura de carpetas separa las clases según el siguiente criterio:

- Entities: Es la carpeta en la que se declaran las entidades principales del servicio, en nuestro caso los usuarios, carpetas y suscripciones.
- Repositories: Es la carpeta en la que se declaran las interfaces que extienden de la interfaz JPARepository y que permite realizar las distintas operaciones CRUD con las entidades.
- Services: Es la carpeta en la que se realiza la mayor parte del negocio. Obtiene los objetos de la capa Controller y con el uso inyectado del repositorio realiza las operaciones necesarias con los objetos en la base de datos.
- Controllers: La carpeta en la que se configuran los endpoints del servicio y en la que se usa la capa inyectada del servicio para realizar la lógica de negocio.

Aparte de estas carpetas, el servicio tambien cuenta con otras clases para la configuración, encriptación, manejo de excepciónes, DTOs, mapeadores, etc.

5.1.1. Entities

Como se ha indicado previamente, en esta carpeta se definen las entidades del servicio. Estas clases se apoyan en las anotaciones JPA para crear automáticamente la base de datos una vez se despliega el servicio. Por ejemplo, en la Figura 15, se muestra la configuración de la entidad Usuario.

```java
@AllArgsConstructor
@NoArgsConstructor
@Builder
@Data
@Entity
@Table(name = "users")
public class User {

    @Id
    @Column(name = "user_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name", nullable = false, length = 255)
    private String name;

    @Column(name = "user_surname", length = 255)
    private String surname;

    @Column(name = "user_email", nullable = false, length = 255)
    private String email;

    @Column(name = "user_password", nullable = false, length = 255)
    private String password;

    @Column(name = "user_messages", nullable = false)
    private boolean messages;

    @ToString.Exclude
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Folder> folders;
}
```

Figura 15: Entidad Usuario anotada con JPA y Lombok

Gracias a estas anotaciones se puede definir tanto el nombre de la tabla que se creará en la base de datos, a través de la anotación "@Table", como la configuración de las diferentes columnas de la tabla (anotación "@Column") o las relaciones con las otras tablas (anotaciones "@OneToMany"). Todo ello gracias a la anotación "@Entity" que identifica esta clase como una entidad JPA. Por otro lado, Lombok permite utilizar anotaciones como "@AllArgsConstructor" y "@NoArgsConstructor" que son capaces de generar un constructor con todos los atributos de la clase como argumentos y también crear otro sin argumentos sin necesidad de escribir su código. Las anotaciones "@Builder" y "@Data" permiten mayor facilidad a la hora de instanciar esta clase y generar los getters, setters y el método toString() de esta clase, respectivamente.

5.1.2. Repositories

Un repositorio, en el ámbito de un servicio REST implementado con Spring Boot, se trata típicamente de un componente encargado de manejar y acceder a los datos relacionados con una entidad específica, ya sea almacenada en una base de datos u otro sistema de almacenamiento. Es común emplear repositorios para dividir la lógica asociada con la recuperación y gestión de datos de la lógica de negocio de la aplicación. En esta carpeta, únicamente se declaran las interfaces de los repositorios de cada entidad. Estas interfaces extienden de la interfaz "JpaRepository", que permite la realización de todas las operaciones CRUD. En la figura 16, se puede observar como se declara la interfaz del repositorio de la entidad Usuario.

```java
@Repository
public interface IUserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
}
```

Figura 16: Interfaz del repositorio de la entidad Usuario

Como se puede observar, en este caso a la interfaz "IUserRepository" se le añade un método más, aparte de los heredados de "JpaRepository", como es el método para buscar un usuario por su email, ya que es necesario y será utilizado por otros componentes del servicio.

5.1.3. Services

En el ámbito de un servicio REST implementado con Spring Boot, las clases de servicio son componentes esenciales que encapsulan la lógica de negocio y funcionan como intermediarios que facilitan la comunicación entre las partes encargadas de la recepción de las peticiones HTTP y las encargadas de la persistencia de datos. Este enfoque de diseño contribuye a una estructura más modular y a una facilidad de mantenimiento al separar de manera clara las funciones y responsabilidades en capas definidas. En esta carpeta se declaran tanto las interfaces como las implementaciones de los distintos servicios que provee el servicio REST. A continuación, en la Figura 17 se puede ver como está declarada la implementación del servicio asociado a la entidad Usuario, así como la implementación de alguno de sus métodos.

```java
@RequiredArgsConstructor
@Service
public class UserService implements IUserService {

    private final IUserMapper userMapper;
    private final PasswordEncoder passwordEncoder;
    private final IUserRepository userRepository;


    public UserDto findByEmail(String email) {
        Optional<User> optionalUser = userRepository.findByEmail(email);

        if (optionalUser.isEmpty()) {
            return null;
        }
        return userMapper.toUserDto(optionalUser.get());
    }

    public UserDto register(SignUpDto userDto) {
        Optional<User> optionalUser = userRepository.findByEmail(userDto.getEmail());

        if (optionalUser.isPresent()) {
            return null;
        }

        User user = userMapper.signUpToUser(userDto);
        user.setPassword(passwordEncoder.encode(CharBuffer.wrap(userDto.getPassword())));

        User savedUser = userRepository.save(user);

        return userMapper.toUserDto(savedUser);
    }
```

Figura 17: Implementación del servicio de la entidad Usuario

Tanto estos métodos como todos los de esta clase que lo requieren, tienen como parámetro un objeto DTO. Un DTO (Data Transfer Object) [5] es un objeto que se utiliza para transferir datos entre las distintas capas de una aplicación, sin la necesidad de pasar la totalidad de la información de un objeto, sino solamente los datos que son necesarios o interesan. En este caso, la clase de servicio utiliza 4 tipos de DTOs: CredentialDto, que solo tiene el email y la contraseña, ya que son los únicos datos que necesito para verificar si un usuario existe en la base de datos o no, SignUpDto, que contiene toda la información necesaria para crear un usuario, CompleteUserDto, que devuelve toda la información de un usuario (menos la contraseña) y por último UpdateDto, que contiene todos los campos modificables de un usuario, para cuando se quieran modificar.

Como se puede observar en la Figura 17, en la clase en la que se desarrolla la implementación del servicio, se inyectan tanto el repositorio de usuarios IUserRepository como una clase denominada "PasswordEncoder" que nos servirá para codificar la información sensible del usuario antes de guardarla en la base de datos, de forma que la información sea almacenada de forma segura, cumpliendo así con uno de los requisitos no funcionales detectados en la fase de Análisis. Por otro lado, también se inyecta una interfaz denominada "IUserMapper" que, apoyándose en la librería MapStruct [8], permite hacer el mapeo entre la clase User y los distintos DTOs creados en el servicio REST de forma sencilla. En la Figura 18 se puede observar como es la especificación de esta interfaz.

```java
@Mapper(componentModel = "spring")
public interface IUserMapper {

    @Mapping(source="email", target = "email")
    @Mapping(source = "userId", target = "id")
    UserDto toUserDto(User user);

    @Mapping(target = "userId", ignore = true)
    @Mapping(target = "password", ignore = true)
    @Mapping(source = "name", target = "name")
    @Mapping(source = "surname", target = "surname")
    @Mapping(source = "messages", target = "messages")
    User signUpToUser(SignUpDto userDto);

    @Mapping(source = "name", target = "name")
    @Mapping(source = "surname", target = "surname")
    @Mapping(source = "email", target = "email")
    @Mapping(source = "messages", target = "messages")
    @Mapping(source = "folders", target = "folders")
    CompleteUserDto toCompleteUserDto(User user);

}
```

Figura 18: Especificación del mapper de la entidad Usuario

5.1.4. Controllers

En el ámbito de un servicio REST implementado con Spring Boot, los controladores sirven como primer punto de contacto para las solicitudes de los clientes. Se encargan de orquestar el camino que sigue la solicitud, dirigiéndola hacia las clases de servicio donde se encuentra la lógica de negocio. Posteriormente, se encargan de gestionar las respuestas que se envían de regreso al cliente. Este enfoque se traduce en una clara división de responsabilidades, lo que contribuye a un diseño estructurado y fácil de mantener. En esta carpeta se declaran los distintos controladores que posee el servicio REST. Como se ha comentado en la fase de diseño el servicio REST cuenta con 4 controladores, uno por entidad, y un controlador extra que contiene exclusivamente los endpoints encargados de la creación de un nuevo usuario y de

hacer login en la aplicación y así obtener un nuevo token. En las Figuras 19 y 20 se puede observar como es la implementación de este controlador extra, denominado "AuthController" y la implementación del controlador de la entidad Usuario.

```java
@RequiredArgsConstructor
@RestController
public class AuthController {

    private final UserService userService;
    private final UserAuthProvider userAuthProvider;

    @PostMapping("/token")
    public ResponseEntity<UserDto> getToken(@RequestBody @Valid CredentialDto credentialDto, BindingResult result){
        if (result.hasErrors()) {
            // Return a 400 Bad Request response if validation fails
            return ResponseEntity.badRequest().build();
        }
        UserDto user = userService.login(credentialDto);
        if (user == null) {
            return ResponseEntity.notFound().build();
        }
        user.setToken(userAuthProvider.createToken(user.getEmail()));
        return ResponseEntity.ok(user);
    }

    @PostMapping("/users")
    public ResponseEntity<UserDto> register(@RequestBody @Valid SignUpDto signUpDto, BindingResult result) {
        if (result.hasErrors()) {
            // Return a 400 Bad Request response if validation fails
            return ResponseEntity.badRequest().build();
        }
        UserDto user = userService.register(signUpDto);
        if (user == null) {
            return ResponseEntity.notFound().build();
        }
        user.setToken(userAuthProvider.createToken(user.getEmail()));
        return ResponseEntity.created(URI.create("/users/" + user.getId()))
                .body(user);
    }
}
```

Figura 19: Implementación del controlador AuthController

```java
@RequiredArgsConstructor
@RestController
public class UserController {

    private final UserService userService;

    @GetMapping(path = "/users")
    public ResponseEntity<List<CompleteUserDto>> getAllUsers(){return ResponseEntity.ok(userService.getAllUsers());}

    @GetMapping(path = "/users/{userId}")
    public ResponseEntity<CompleteUserDto> getUserById(@PathVariable Long userId){
        CompleteUserDto completeUserDto = userService.getUserById(userId);
        if (completeUserDto == null) {
            ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(completeUserDto);
    }

    @PutMapping(path = "/user/{userId}")
    public ResponseEntity<CompleteUserDto> updateUser(@RequestBody UpdateUserDto updateUser,
                                                      @PathVariable Long userId) {
        CompleteUserDto completeUserDto = userService.updateUser(updateUser,userId);
        if (completeUserDto == null) {
            ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(userService.updateUser(updateUser,userId));
    }

    @DeleteMapping(path = "user/{userId}")
    public ResponseEntity<CompleteUserDto> deleteUser(@PathVariable Long userId){
        CompleteUserDto completeUserDto = userService.deleteUser(userId);
        if (completeUserDto == null) {
            ResponseEntity.notFound().build();
        }
        return ResponseEntity.ok(userService.deleteUser(userId));
    }
}
```

Figura 20: Implementación del controlador UserController

La decisión de separar en dos este controlador es debido a que el "AuthController" hace uso de una clase denominada "UserAuthProvider" que será la encargada de tanto generar como validar un token creado con JWT para que solo puedan obtener datos del servicio REST los usuarios registrados en la aplicación.

Estos controladores, se apoyan también en las anotaciones de Spring Web MVC y Lombok para su implementación, por ejemplo, "@PostMapping", que indica que este método manejará las solicitudes HTTP de tipo POST en las rutas especificadas, "@PathVariable" que captura el valor del parámetro variable "userId" de la URL y lo asigna a la variable "userId" y "@RequestBody" que indica que se espera un cuerpo de solicitud (en formato JSON) que represente un objeto, por ejemplo Credential, que contiene la información necesaria para poder iniciar sesión en la aplicación.

5.1.5. JWT

Para finalizar el proceso de desarrollo de código del servicio REST se añade un nivel de seguridad extra en la configuración, para que los datos solo sean accesibles por usuarios de la aplicación. Para ello se ha decidido hacer uso de tokens JWT, que permiten autenticar si una petición al servicio REST la está realizando un usuario autorizado.

Para ello, se desarrolló primero la clase UserAuthProvider, en la que se configuran los métodos que generan y validan los tokens JWT. A continuación en la Figura 21 se puede ver la implementación de esta clase.

```
@RequiredArgsConstructor
@Component
public class UserAuthProvider {

    @Value("${security.jwt.token.secret-key}")
    private String secretKey;

    private final UserService userService;

    @PostConstruct
    protected void init() { secretKey = Base64.getEncoder().encodeToString(secretKey.getBytes()); }

    public String createToken(String email) {
        Date now = new Date();
        Date validity = new Date(now.getTime() + 3_600_000);
        return JWT.create()
                .withIssuer(email)
                .withIssuedAt(now)
                .withExpiresAt(validity)
                .sign(Algorithm.HMAC256(secretKey));
    }

    public Authentication validateToken(String token) {
        JWTVerifier verifier = JWT.require(Algorithm.HMAC256(secretKey))
                .build();
        DecodedJWT decoded = verifier.verify(token);
        UserDto user = userService.findByEmail(decoded.getIssuer());
        return new UsernamePasswordAuthenticationToken(user, credentials: null, Collections.emptyList());
    }
}
```

Figura 21: Implementación de la clase UserAuthProvider

Como se puede observar el token JWT se genera con el email del usuario que lo está generando, que es su identificador de usuario, la fecha actual y la fecha en la que expirará, que son 3600000 milisegundos despues de generarse, es decir, 1 hora.

Añadida a esta clase también se implementa un filtro previo, que se ejecuta cada vez que se llama a un recurso del servicio REST en el que primero se verifica que en la cabecera llamada "Authorization" de la petición HTTP hay datos, en concreto 2 y que el primero es el texto "Bearer", pues este es el formato que se sigue en la autenticación mediante tokens JWT. En la Figura 22 se puede ver su implementación.

```java
@RequiredArgsConstructor
public class JwtAuthFilter extends OncePerRequestFilter {

    private final UserAuthProvider userAuthProvider;
    @Override
    protected void doFilterInternal(
            HttpServletRequest request,
            HttpServletResponse response,
            FilterChain filterChain) throws ServletException, IOException {
        String header = request.getHeader(HttpHeaders.AUTHORIZATION);

        if (header != null) {
            String[] elements = header.split( regex: " ");

            if (elements.length == 2 && "Bearer".equals(elements[0])) {
                try {
                    SecurityContextHolder.getContext().setAuthentication(
                            userAuthProvider.validateToken(elements[1])
                    );
                } catch (RuntimeException e) {
                    SecurityContextHolder.clearContext();
                    throw  e;
                }
            }
        }
        filterChain.doFilter(request,response);
    }
}
```

Figura 22: Implementación de la clase JwtAuthFilter

Como se puede ver la clase extiende de la clase "OncePerRequestFilter" lo que hace que se ejecute siempre una vez por llamada. Por último, se crea una clase que permite filtrar cuales son los distintos recursos que pueden ser llamados sin autenticación previa y qué tipo de métodos se pueden utilizar en ellos. En este caso se define que solo se puedan utilizar métodos de tipo POST a las URIs "token" y "users", para poder tanto iniciar sesión como registrarse y así obtener un token que deberá ser incluido en la cabecera "Authorization" de futuras llamadas. En la Figura 23 se puede ver cómo se puede implementar esta lógica.

```java
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .exceptionHandling(configurer -> configurer.authenticationEntryPoint(userAuthenticationEntryPoint))
            .addFilterBefore(new JwtAuthFilter(userAuthProvider), BasicAuthenticationFilter.class)
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests((requests) -> requests
                    .requestMatchers(HttpMethod.POST, "/token", "/users").permitAll()
                    .anyRequest().authenticated()
            );
    return http.build();
}
```

Figura 23: Implementación del filtro de URIs

5.2. Aplicación Web

Continuando con la metodología de trabajo iterativa y una vez finalizado el desarrollo del servicio REST se comenzó con el desarrollo de la aplicación web. Como se menciona previamente, para el desarrollo de la aplicación web se ha decidido desarrollar una solución SPA.

Para comenzar con la creación del proyecto, es necesario instalar en nuestra máquina Node.js y npm. Node.js nos permite ejecutar programas escritos en JavaScript y npm es un sistema de gestión de paquetes que nos permite añadir módulos a nuestro Node.js. Una vez tengamos ambos instalados tan solo habrá que ejecutar el comando "npm create-react-app frontend" el cual nos creará un nuevo proyecto en React en un nuevo directorio con el nombre indicado en el comando, en este caso, "frontend". Para el desarrollo de esta parte web se ha utilizado Material-UI, que es una biblioteca de código abierto que permite añadir componentes a nuestra aplicación basados en componentes de Google. De esta forma se reduce el tiempo de trabajo y aumenta la calidad de los componentes de la aplicación.

Como se menciona anteriormente, esta aplicación web tiene una estructura en carpetas en función de la utilidad o naturaleza de sus componentes. Dentro de la carpeta "src" que es donde se sitúa el código de la aplicación se han creado distintas carpetas para diferenciar los distintos componentes que va a tener la web, las cuales son:

- Components
- Images
- Pages
- Service
- Styles
- Index.js

5.2.1. Components

Dentro de esta carpeta se sitúan los componentes que se reutilizan en la aplicación. Esta carpeta a su vez se divide en dos subcarpetas llamadas "container" y "pure". La carpeta "container" contiene los componentes que actúan de contenedores de los componentes más pequeños, por ejemplo, los contenedores que tienen la lista de suscripciones, el árbol de suscripciones, los gráficos o la modal que aparece al añadir una nueva carpeta o suscripción. Por otro lado, está la carpeta "pure", donde se alojan los componentes más sencillos, como pueden ser el árbol de suscripciones o el botón de añadir una nueva carpeta o suscripción. Esta carpeta a su vez está subdividida en "forms", que como su nombre indica contiene los distintos formularios de la aplicación, "graphics", que contiene los gráficos con los resúmenes de gastos y "toolbar", que contiene las distintas toolbars que aparecen en la aplicación. Con todo ello, en la Figura 24, se puede observar como queda la estructura de esta carpeta.

En React existen dos tipos de componentes: los componentes de tipo función y los componentes de tipo clase. La diferencia entre ellos, como su propio nombre indica es que los componentes de tipo clase son una clase como tal, con sus atributos, funciones e incluso estado. Sin embargo los componentes de tipo función no poseen atributos o estado si no que utilizan los llamados "hooks" para manejar el estado o el ciclo de vida de los componentes. En la aplicación se ha decidido usar únicamente componentes de tipo función debido a que de esta forma el código queda más limpio y entendible.

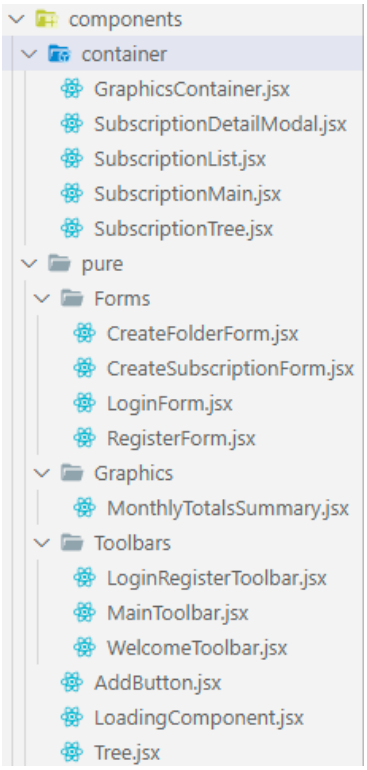

Figura 24: Carpeta Components

5.2.2. Images

La siguiente carpeta de nuestra aplicación web es "images" y como su propio nombre indica, nos sirve para guardar todas las imágenes que se utilizan en la web.

5.2.3. Pages

Esta carpeta contiene una subcarpeta con cada una de las distintas páginas que tiene la aplicación. Cada una de estas páginas tiene una URL distinta para poder navegar por la aplicación que se configurará en el archivo index.js.

Todas las páginas siguen el mismo diseño, primero se declara la toolbar que se crea previamente dentro de los componentes de tipo "pure" y después el componente de tipo "container" que contiene el resto de componentes de la página. A continuación en las Figuras 25 y 26 se puede observar como está organizada esta carpeta y como sería la implementación de la página principal respectivamente.

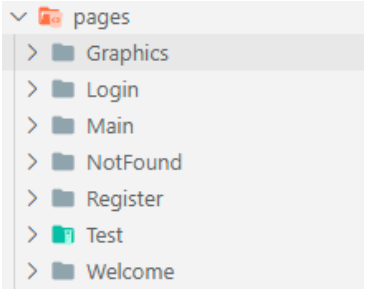

Figura 25: Carpeta Pages

src > pages > Main > MainPage.jsx > ...

```jsx
import React from 'react';
import MainToolbar from '../../components/pure/Toolbars/MainToolbar';
import SubscriptionMain from '../../components/container/SubscriptionMain';
import '../../styles/Main/MainPage.css'

const MainPage = () => {
    return (
        <div className="MainPage">
            <MainToolbar />
            <div className="MainContent">
                <SubscriptionMain />
            </div>/.MainContent
        </div>/.MainPage
    );
}

export default MainPage;

```

Figura 26: Implementación de la "Main Page"

5.2.4. Service

Dentro de esta carpeta se encuentra el componente llamado "axiosHelper". En él, se declaran los métodos que se encargarán de comunicarse con el servicio REST y de manejar los datos que devuelva. Para su implementación se hace uso de la librería llamada axios [21], que permite realizar llamadas HTTP al servicio REST de una forma sencilla. En la Figura 27 se puede ver como es la implementación de este componente.

Aparte de implementar el método que realiza las peticiones HTTP al servicio REST también se encarga de almacenar tanto el token, que se genera al iniciar sesión o registrarse, como de almacenar el userId, que servirá para utilizarlo en futuras llamadas HTTP.

```jsx
import axios from "axios";

axios.defaults.baseURL='http://localhost:8080'
axios.defaults.headers.post["Content-Type"] = 'application/json'

export const getAuthToken = () => {
    return window.localStorage.getItem("auth_token")
}

export const setAuthToken = (token) => {
    window.localStorage.setItem("auth_token",token)
}

export const getUserId = () => {
    return window.localStorage.getItem("user_id")
}

export const setUserId = (userId) => {
    return window.localStorage.setItem("user_id", userId)
}

export const request = (method,url,data) => {
    let headers = {}
    if(getAuthToken() !== null && getAuthToken() !== "null") {
        headers = {"Authorization": `Bearer ${getAuthToken()}`}
    }

    return axios({
        method : method,
        headers : headers,
        url : url,
        data : data
    })
}
```

Figura 27: Implementación del componente "AxiosHelper"

5.2.5. Styles

Para finalizar con las carpetas, tenemos la carpeta "styles" que también está subdividida en carpetas con los estilos CSS de cada una de las páginas y componentes que es necesario personalizar. En el caso de los componentes importados desde Material-UI la personalización de los mismos se realiza como parámetro a la hora de declararlo por lo que no es necesario un archivo de personalización.

5.2.6. Index.js

Por último la aplicación cuenta con el fichero index.js en el que se configura cada una de las rutas de la aplicación apoyándose en una librería llamada "react-router-dom" [18]. Gracias a esta librería se configura un componente de tipo "page" en cada una de las rutas de la aplicación, haciendo posible la navegación. La configuración se realiza de esta forma: primero se crea un router asignando a cada dirección un componente que renderizar y a continuación dentro de la función render del componente root se declara un objeto de tipo "RouterProvider" el cual tiene como parámetro el router que acabamos de crear. A continuación, en la Figura 28 se puede observar como es la implementación del router que se utiliza en la aplicación, así como su declaración dentro de la renderización del componente "root".

```
const router = createBrowserRouter([
    {
      path: "/",
      element: <WelcomePage />,
      errorElement: <NotFoundPage />
    },
    {
      path: "login",
      element: <LoginPage />,
      errorElement: <NotFoundPage />
    },
    {
      path: "register",
      element: <RegisterPage />,
      errorElement: <NotFoundPage />
    },
    {
      path: "main",
      element: <MainPage />,
      errorElement: <NotFoundPage />
    },
    {
      path: "graphics",
      element: <GraphicsPage />,
      errorElement: <NotFoundPage />
    },
    {
      path:"pruebas",
      element: <TestPage />,
      errorElement:<NotFoundPage />
    }
  ]);
```

(a) Router

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <RouterProvider router={router} />
);
```

(b) Router Provider

Figura 28: Inicialización de la aplicación

5.2.7. Formularios

Una vez se creó el esqueleto de la aplicación se comenzó con la implementación de los distintos componentes. Se comenzó con los componentes de tipo página ya que eran necesarios para la configuración

del router. Como se menciona previamente todos los componentes de tipo página siguen el mismo diseño, primero una toolbar diseñada con Material-UI y un componente de tipo container para contener los distintos componentes de la página. A continuación, se crearon los distintos formularios, tanto de registro como de inicio de sesión. Los formularios se construyeron utilizando como apoyo las librerías "Formik" y "yup". Estas librerías facilitan la configuración del formulario y proporcionan una ayuda a la hora de definir criterios y validaciones en los campos. Gracias a la librería "Formik" se pueden definir valores por defecto para los distintos campos y las acciones que tienen que realizarse una vez se haya pulsado el botón de tipo submit. Una de las ventajas que tienen estas librerías es su fácil integración entre ellas, primero se declaran las validaciones de los campos en un objeto apoyándonos en la librería "yup", como se puede apreciar en la Figura 29, y a continuación, se declara en una variable llamada "formik" tanto los valores iniciales de los campos, como el esquema de validación que se crea previamente con yup y las acciones que tiene que realizar el sistema una vez se ha pulsado el botón de tipo submit. A continuación en las figuras 29, 30 y 31 se expone como es la configuración de las validaciones para el formulario de registro y como se integra en el formulario usando formik en los distintos campos.

```
const validationSchema = yup.object({
    name: yup
        .string('Introduce tu nombre')
        .required('El nombre es obligatorio'),
    surname: yup
        .string('Introduce tus apellidos'),
    email: yup
        .string('Introduce tu correo')
        .email('Introduce un correo con un fromato valido')
        .required('El correo es obligatorio'),
    password: yup
        .string('Introduce tu contraseña')
        .required('La contraseña es obligatoria')
        .matches(
            /^(?=.*[a-z])(?=.*[A-Z])(?=.*[0-9])(?=.*[!@#$%^&*])(?=.{8,})/,
            "La contrasena debe tener minimo 8 caracteres, una letra mayuscula, una letra minuscula,un numero y un caracter especial"
        ),
    confirm: yup
        .string('Confirma tu contraseña')
        .oneOf([yup.ref('password'), null], 'Las contraseñas deben coincidir')
        .required('Este campo es obligatorio'),
    messages: yup.bool()
});
```

Figura 29: Yup

```
const formik = useFormik({
    initialValues: {
        name: '',
        surname: '',
        email: '',
        password: '',
        confirm: '',
        messages: false
    },
    validationSchema: validationSchema,
    onSubmit: async (values) => {
        try{
          request(
            "POST",
            "/users",
            {
              name:values.name,
              surname:values.surname,
              email:values.email,
              password:values.password,
              messages:values.messages
            }
          ).then((response) => {
            setUserId(response.data.id)
            setAuthToken(response.data.token)
            navigate('/main')
            toast.success("Usuario Registrado Correctamente");
          }).catch((error) => {
            console.log(error)
          })
        } catch(err) {
            console.log(err);
        }
    },
});
```

Figura 30: Configuración de formik en el formulario de Registro

```
<TextField
    sx={{ marginBottom: '20px' }}
    fullWidth
    id="name"
    name="name"
    label="Nombre*"
    value={formik.values.name}
    onChange={formik.handleChange}
    error={formik.touched.name && Boolean(formik.errors.name)}
    helperText={formik.touched.name && formik.errors.name}
/>
```

Figura 31: Campo nombre del formulario

Como se puede ver en la Figura 30, se utiliza la función declarada en el componente "axiosHelper" para poder hacer una petición HTTP de tipo POST al servicio REST y una vez se obtiene la respuesta se guarda tanto el id del usuario como el token generado, que se guarda en el localStorage del navegador para ser utilizado en futuras peticiones. En el caso del formulario de "Inicio de Sesión" el funcionamiento es el mismo.

5.2.8. Funcionamiento

Como se ha declarado en el router configurado en el archivo "Index.js", una vez lanzada la aplicación accedemos a la ruta "/" que presenta la página de bienvenida y que únicamente contiene una imagen de fondo y una toolbar en la que se nos presentan dos botones con las opciones de "Registrarse" o "Iniciar Sesión". Si hacemos click en cualquiera de los dos botones, esta acción nos redirigirá a una nueva ruta en la que, como se configuró en el router, se renderizará el componente de tipo página correspondiente que contendrá los distintos formularios.

Una vez hemos iniciado sesión o nos hemos registrado accederemos a la página principal de la aplicación, es decir, el listado de suscripciones. La acción de registrarse o iniciar sesión nos redirigirá a la ruta "/main" en la que se renderizará el componente de tipo página "MainPage.jsx". Este componente, aparte de renderizar la toolbar correspondiente, renderizará un componente de tipo container llamado "SubscriptionMain.jsx" que será el encargado de, haciendo uso de la funcion definida en el componente "axiosHelper", enviar la petición al servicio REST para cargar los datos del usuario.

Este componente hace uso del "hook" useEffect() que permite ejecutar una función una vez el componente desde el que se llama se ha renderizado, por ello dentro de este "hook" es desde donde se realiza la petición HTTP. A continuación, en la Figura 32, se puede observar como es la implementación de este "hook" en el que se realizan dos peticiones GET al servicio REST para obtener tanto las carpetas como las suscripciones del usuario.

Este "hook", aparte de realizar las peticiones HTTP al servicio REST primero, comprueba que estamos autenticados en el sistema, de forma que pese a que accedamos a esta ruta, si no estamos autenticados no pueda cargar ningún tipo de información. Posteriormente obtiene la información de las carpetas del usuario y las guarda en una lista llamada folders.

A continuación, realiza una petición de tipo GET para obtener la información de las suscripciones del usuario, pero, antes de almacenarlas en una lista, aplica una lógica para obtener cuando será la próxima fecha en la que se deberá abonar el importe de la suscripción, lo añade a los datos de la suscripción y guarda esta información en una lista. También, como información adicional, guarda en la suscripción el número de meses que el usuario lleva suscrito al servicio y que se presentará en la página de los detalles de la suscripción.

```
useEffect(() => {
    if (shouldLog.current) {
        shouldLog.current = false;
        if (userId == null) {
            alert('You need to log in first!')
            navigate('/login')
        }
        request( "GET", "/users/" + userId + "/folders", {}
        ).then((response) => {
            setfolders(response.data)
        }).catch((error) => {
            console.log(error)
        })
        request("GET", "/users/" + userId + "/subscriptions", {}
        ).then((response) => {
            const updatedSubscriptions = response.data.map(subscription => {
                const { contractDate, subscriptionFrequency } = subscription;

                const today = moment(); // Get the current date
                const contractMoment = moment(contractDate, 'DD/MM/YYYY');

                // Calculate the number of months between the current date and the contract date
                const monthsSinceContract = today.diff(contractMoment, 'months');

                // Calculate the number of payment periods that have passed
                const passedPaymentPeriods = Math.floor(monthsSinceContract / subscriptionFrequency);

                // Calculate the next payment date based on the current date and payment frequency
                const nextPaymentDate = contractMoment
                  .add(passedPaymentPeriods * subscriptionFrequency, 'months')
                  .add(subscriptionFrequency, 'months')
                  .format('DD/MM/YYYY');

                return {
                  ...subscription,
                  nextPaymentDate,
                  monthsSinceContract
                };
              });
            setSuscriptions(updatedSubscriptions)
        }).catch((error) => {
            console.log(error)
        })
    }
}, [navigate, userId])
```

Figura 32: Hook useEffect utilizado en el componente principal

Una vez obtenida la información del usuario esta información se propaga mediante las "props" de los componentes a los distintos componentes hijos para que hagan uso de la información obtenida. En la Figura 33 se puede apreciar como una vez terminada la ejecución del "hook" useEffect() se renderizan los componentes del árbol de suscripciones y el listado de suscripciones.

```
if (subscriptions && folders) {
    return (
    <div className='TreeListContainer'>
        <SubscriptionTree folders={folders} subscriptions={subscriptions} showSubModal={showSubModal} showFolderModal={showFolderModal} />
        <SubscriptionList subscriptions={subscriptions} showSubDetailModal={showSubscriptionDetailModal} />
        {showCreateSubModal ? (<CreateSubscriptionForm showModal={showSubModal} userId={userId} />) : null}
        {showCreateFolderModal ? (<CreateFolderForm showModal={showFolderModal} userId={userId} />) : null}
        {showSubDetailModal ? (<SubscriptionDetailModal showModal={showDetailModal} subscriptionId={subDetail} />) : null}
    </div>/.TreeListContainer)
} else {
    return (
        <div>
            <LoadingComponent />
        </div>

    )
}
```

Figura 33: Propagación de la información a los componentes hijos

La lógica que sigue el código mostrado en la Figura 33 es la siguiente. Primero, hasta que tanto la lista de suscripciones como la lista de carpetas dejen de estar vacías, se renderiza un componente "loading" que forma parte de los componentes de Material-UI. Una vez se ha terminado de cargar la información en las listas, se renderizan los componentes del árbol de suscripciones y la lista de suscripciones, teniendo como "props" los datos obtenidos del servicio REST y distintos booleanos para saber cuando mostrar ciertas modales. Esto es así porque los formularios de "Añadir carpeta" o "Añadir suscripción" o los detalles de las suscripciones se renderizan sobre una modal, que está oculta y que aparece sobre la página principal, por tanto, los componentes deben saber el estado de esta modal para saber cuando tienen que

renderizarse o no. A continuación en la figura 34 se muestra como es la página principal una vez se ha iniciado sesión y se han cargado los datos correctamente.

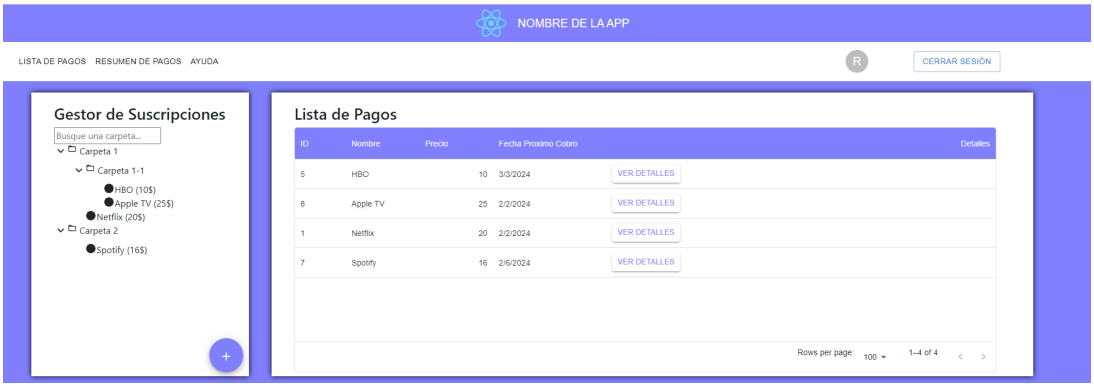

Figura 34: Página de Inicio

En las Figuras 35 y 36 se puede observar como se muestra finalmente la página de detalles de una suscripción y la página de gráficos.

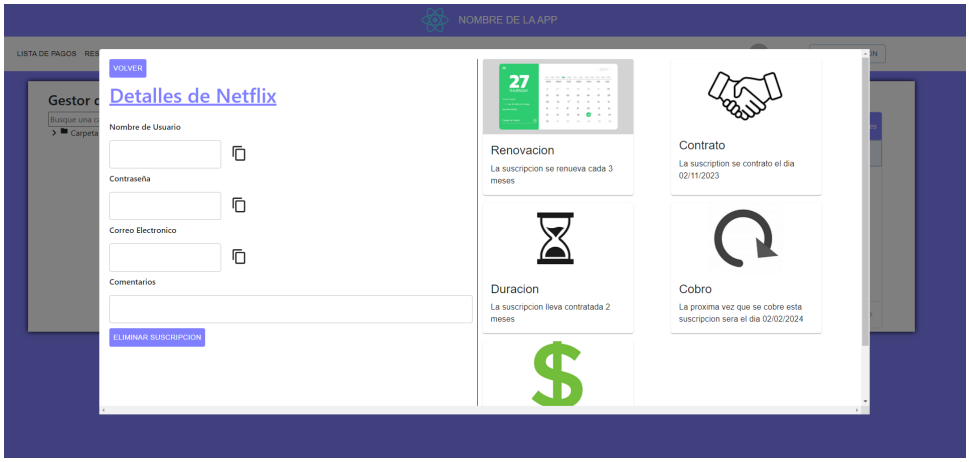

Figura 35: Página de Detalles de una Suscripción

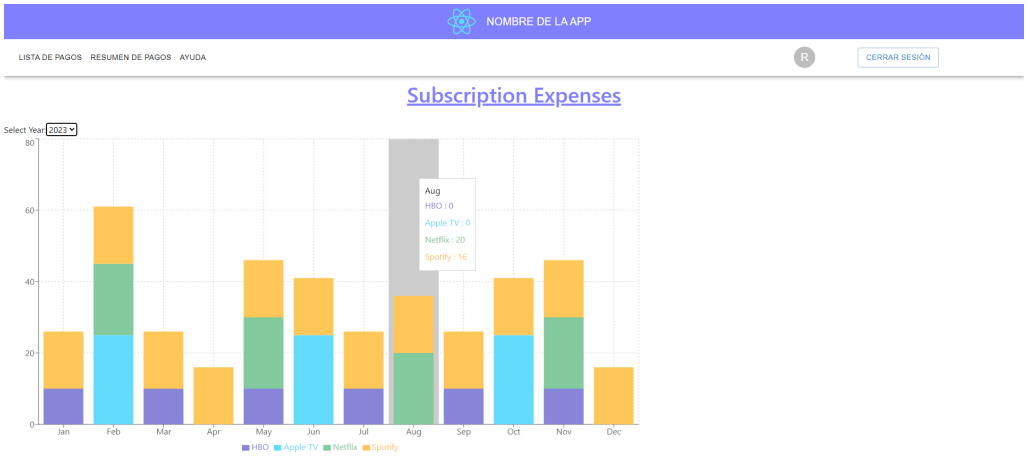

Figura 36: Página de Gráficos

6. PRUEBAS Y VALIDACIONES

El siguiente apartado en el desarrollo de la aplicación y que se realizó de forma iterativa junto con el desarrollo del código de la aplicación es la realización de pruebas y validaciones. Las pruebas permiten evaluar el correcto funcionamiento del software, permiten la detección de errores o comunmente llamados "Bugs" antes de que llegue a los usuarios finales. También garantizan la calidad del código escrito debido a que es una forma de asegurarse que el código cumple con todos los requisitos definidos durante la fase de análisis. Durante estas fases, se han realizado varios tipos de pruebas para garantizar la máxima calidad del software en sus distintos componentes.

6.1. Pruebas de integración del servicio REST

Las pruebas de integración permiten verificar el correcto funcionamiento entre los distintos componentes del software de la aplicación. Para la realización de estas pruebas se ha utilizado "Testcontainers" [11] que es una librería de Java compatible con JUnit que proporciona instancias ligeras y desechables de bases de datos comunes, navegadores web o cualquier otra cosa que pueda ejecutarse en un contenedor Docker. Docker [7] es una plataforma que facilita a los desarrolladores el empaquetado, la distribución y la ejecución consistente de aplicaciones mediante el uso de contenedores. Estos contenedores son unidades ligeras y portátiles que encapsulan tanto el software como sus dependencias, lo que simplifica la implementación y la ejecución en distintos entornos. Por ello, para la realización de estas pruebas ha sido necesaria la instalación de Docker y se ha tenido que añadir a las dependencias del servicio REST la librería "Testcontainers".

En este contexto lo ideal sería examinar los componentes del servicio REST incluyendo la base de datos MySQL. Una base de datos simulada en memoria no sería óptima para esto, ya que puede haber diferencias significativas con la base de datos real y, por lo tanto, puede que ciertas características no funcionen o que se pasen por alto errores. Para solucionar esto, con librería "Testcontainers" se puede arrancar una imagen Docker de exactamente la base de datos que es utilizada por el desarrollador y posteriormente en producción de forma que se puedan realizar pruebas sobre una base de datos "real". En base a esto, el proceso que se llevó a cabo para la prueba del servicio REST fue el siguiente. Primero, se crea la clase en la que se definirán los tests la cuál debe tener ciertas anotaciones. A continuación, en la Figura 37 se muestra un ejemplo de cómo se inicializaría una clase para probar la integración de los métodos de la entidad "User".

```
@Testcontainers
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
@AutoConfigureMockMvc
@ExtendWith(SpringExtension.class)
public class UserIntegrationTests {

    @Autowired
    private UserRepository userRepository;

    //Permite llamadas http
    @Autowired
    private MockMvc mockMvc;
```

Figura 37: Inicialización de la clase de pruebas

La anotación "@Testcontainers" indica que vamos a hacer uso de esta librería. Al utilizar la anotación "@SpringBootTest" en Spring Boot, se carga todo el contexto de la aplicación, lo que facilita la realización de pruebas de integración efectivas. Al establecer la variable "webEnvironment" en "RANDOM_PORT", la aplicación se inicia en un puerto aleatorio disponible, evitando conflictos durante las

pruebas. Además, la configuración de MockMvc se realiza automáticamente mediante la anotación "AutoConfigureMockMvc", lo que proporciona utilidades prácticas para llamar a los endpoints de manera sencilla durante las pruebas. Una vez hecho esto, se configuró la imagen de la base de datos MySQL que se utiliza durante los test. Para ello, en la Figura 38 se visualiza como se inicializa el contenedor de docker con la imagen de la base de datos y el método para que comience su ejecución.

```
@ServiceConnection
private static final MySQLContainer mySQLContainer = new MySQLContainer( dockerImageName: "mysql:8.2");

static {
    mySQLContainer.withUrlParam( paramName: "serverTimezone", paramValue: "UTC") JdbcDatabaseContainer
            .withReuse( reusable: true) GenericContainer
            .start();
}
```

Figura 38: Inicialización del contenedor

A continuación, se creó un método para preconfigurar la base de datos de forma que sabemos en qué estado se encuentra antes de ejecutar los tests. Por último, se realizaron los test de integración, a continuación en la Figura 39 se muestra un ejemplo de un test para verificar que si no se ha introducido un valor obligatorio, como es el nombre del usuario, no se cree ningún usuario en la base de datos.

```
public void setUpCreate(){
    userRepository.deleteAll();
    userRepository.resetIdentityColumn();
}

//CREATE TESTS
jorgebm04
@Test
public void shouldNotCreateWithoutName() throws Exception {
    setUpCreate();
    String url = "/user";
    Assertions.assertThat(userRepository.findAll()).isEmpty();

    mockMvc.perform(post(url)
            .contentType(MediaType.APPLICATION_JSON)
            .content("""
                    {"name": null,
                    "surname":"root",
                    "email":"admin@email.com",
                    "password":"1q2w3E**",
                    "messages":false
                    }"""))
            .andExpect(status().isBadRequest());
    Assertions.assertThat(userRepository.findAll().stream().toList().size()).isEqualTo( expected: 0);
}
```

Figura 39: Inicialización del contenedor

Primero, creamos un método para vaciar la tabla de usuarios con el fin de verificar el correcto funcionamiento del test y a continuación definimos el Test. En el test, primero verificamos que la tabla se ha vaciado correctamente y a continuación realizamos una petición HTTP de tipo "POST" con el objeto que representa a la entidad "Usuario" pero con el campo de nombre vacío. Una vez realizada la petición, verificamos que el método devuelve un codigo de tipo "Bad Request" o "400". Por último, se vuelve a verificar que la tabla de la base de datos de usuarios sigue vacía y no ha añadido al usuario.

Alternativamente a la ejecución de estos tests se realizaron pruebas con la aplicación Postman, para poder observar visualmente cómo son los objetos que se devuelven una vez se hacen las peticiones HTTP al servicio REST. Primero, es necesario hacer una llamada al endpoint "/token" para poder obtener un token que incluir en la cabecera "Authorization". A continuación en la Figura 40, se puede ver cómo se realiza la peticion HTTP de tipo POST con los datos de un usuario y el token que devuelve el servicio.

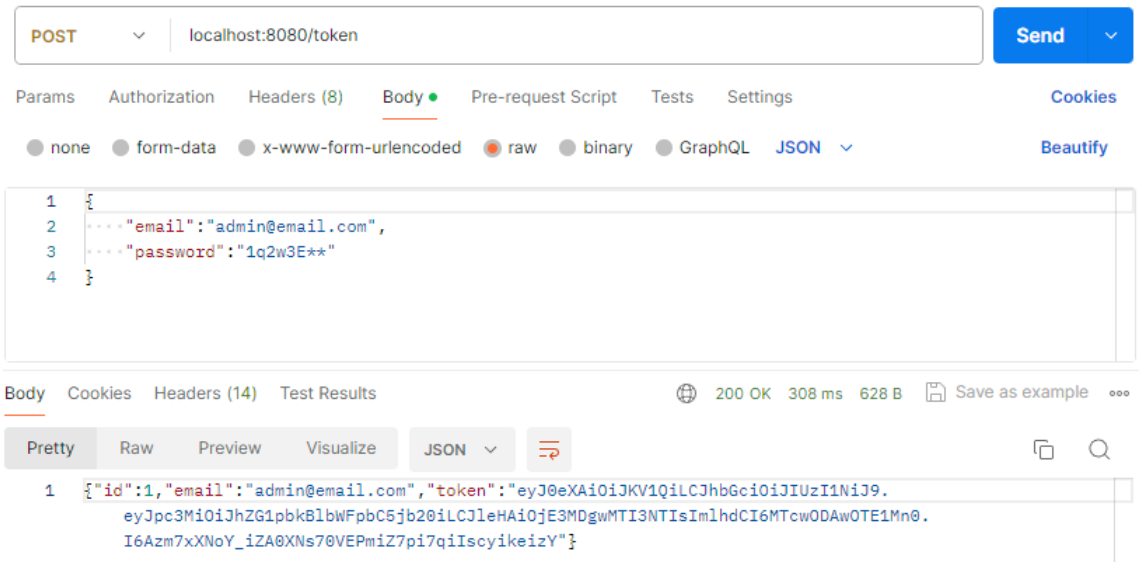

Figura 40: Petición HTTP de tipo POST a la dirección "/token"

Una vez obtenido el token se hace una llamada al endpoint "/users/1/folders" con el token creado en la cabecera "Authorization".

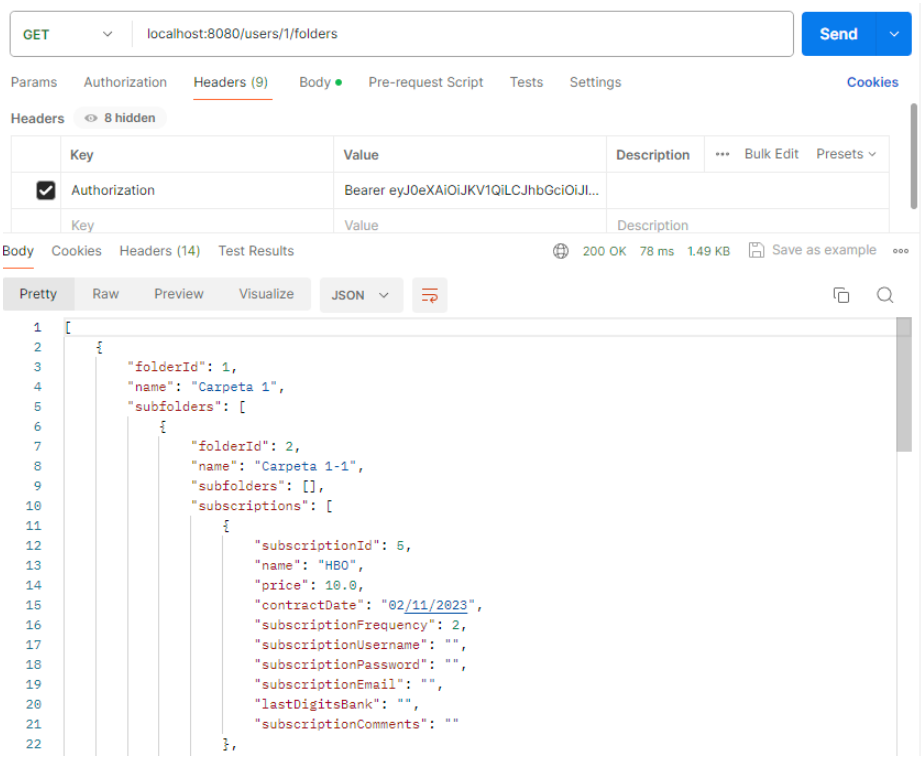

Figura 41: Petición HTTP de tipo GET

Como se puede observar, el servicio devuelve las carpetas del usuario 1 correctamente. A continuación se prueba a crear una nueva carpeta enviando una petición de tipo POST a la misma dirección que la petición de tipo GET y con los datos necesarios para su creación. A continuación en las Figuras 42 y 43 se puede ver como se ha creado correctamente la carpeta y como se visualiza.

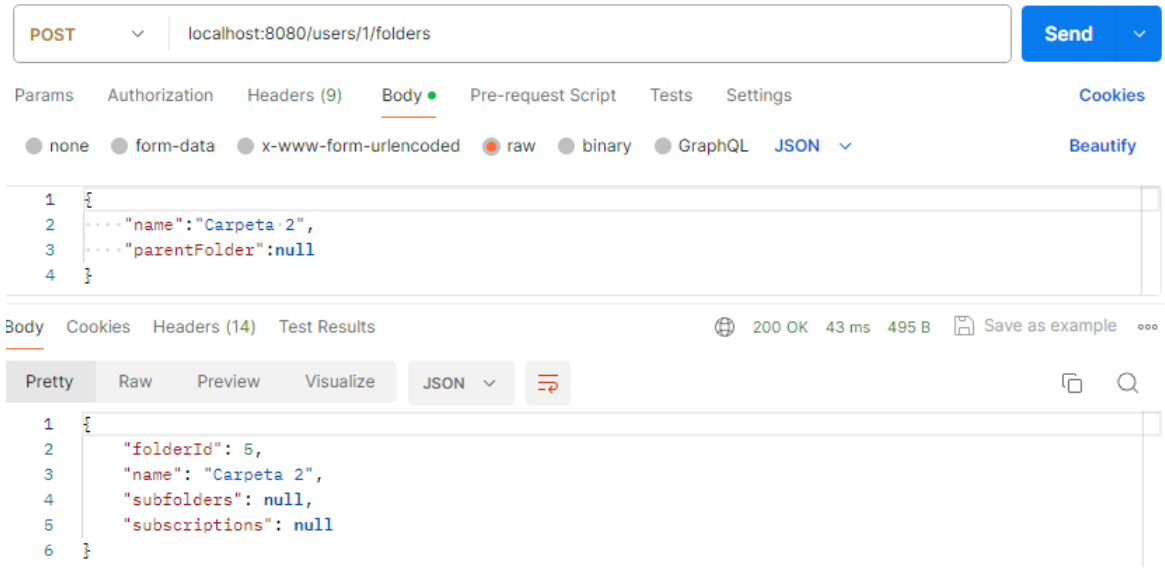

Figura 42: Petición HTTP de tipo POST

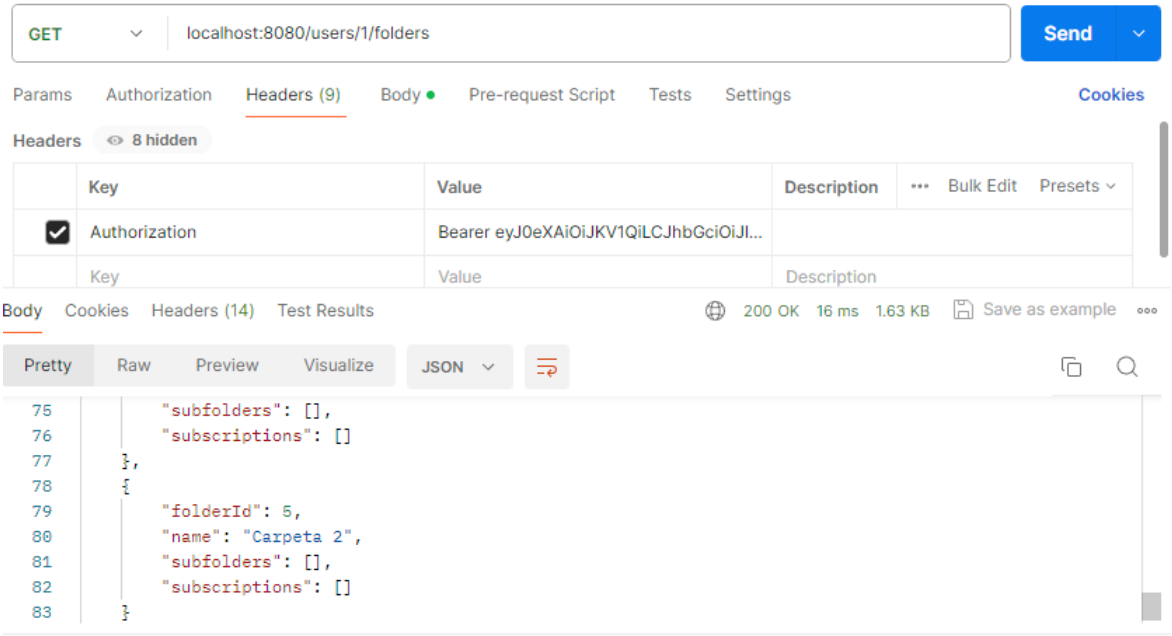

Figura 43: Petición HTTP de tipo GET después de crear la Carpeta

6.2. Pruebas de Integración de la Aplicación Web

En cuanto a las pruebas de integración realizadas a la aplicación web, simplemente se realizaron de manera manual, por falta de tiempo, verificando que los datos obtenidos desde el servicio REST eran los esperados y que la navegación entre las páginas era la correcta.

7. CONCLUSIONES

Para finalizar, en este capítulo se expone un análisis del trabajo realizado, así como unas conclusiones personales y futuros desarrollos que podría incluir la aplicación.

7.1. Conclusiones

Este proyecto tenía como objetivo la realización de una aplicación web que permitiese a los usuarios tanto gestionar sus suscripciones, como tener una visualización de sus gastos y a su vez, un lugar seguro en el que poder almacenar las credenciales de sus suscripciones.

Para ello, se ha intentado seguir la metodología de trabajo definida en capítulos previos de esta memoria, de forma que, pese a ser una sola persona, el desarrollo se haya podido dar con cierta fluidez. De esta forma, siguiendo cada uno de los pasos de la metodología de trabajo, se ha podido desarrollar una aplicación que cumple con casi todos los requisitos definidos durante la fase de análisis. Sin embargo, algunos de ellos, como el envío de resumenes mensuales al correo, no han sido llevados a cabo, por falta de tiempo.

Desde un punto de vista personal, este trabajo ha supuesto todo un reto, ya que se partía sin ningún tipo de conocimiento tanto en servicios REST como en el desarrollo de aplicaciones web, por lo que ha servido para aprender sobre nuevas tecnologías, y como sería el desarrollo de una aplicación web desde cero. La mayor dificultad en el desarrollo del proyecto ha sido la implementación del servicio REST junto con la base de datos gestionada por MySQL ya que pese a poder descargar el "esqueleto" de la aplicación desde la web Spring Initialzr, el servicio REST requiere de una alta configuración para su correcto funcionamiento. Por otro lado, la configuración de la seguridad con los tokens JWT también ha resultado bastante compleja en un primer momento.

7.2. Futuros Trabajos

La aplicación web cuenta con las funcionalidades básicas a realizar sobre las suscripciones y carpetas. Para futuros desarrollos se podría implementar:

- Distintos gráficos para visualizar resúmenes de gastos.
- Cambios en el componente del árbol de suscripciones para hacerlo de mejor calidad en cuanto a funcionalidades nuevas.
- Mejora en el sistema de búsqueda de las suscripciones.
- Inclusión de nuevos idiomas.
- Mejorar la seguridad del servicio REST.
- Implementación del sistema de resúmenes mensuales por correo.
- Habilitar la posibilidad de iniciar sesión con una cuenta de Google.

8. BIBLIOGRAFÍA

[1] IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. *IEEE Std 610*, pages 1–217, 1991.

[2] Apache Friends. Xampp. `https://www.apachefriends.org/es/index.html`. [Último acceso: 12-02-2024].

[3] Axure Software Solutions, Inc. Axure. `https://docs.axure.com/axure-rp/reference/getting-started-video/`. [Último acceso: 12-02-2024].

[4] Kevin Brennan et al. *A Guide to the Business Analysis Body of Knowledger*. Iiba, 2009.

[5] Cecilio Álvarez Caules. Dto. `https://www.arquitecturajava.com/dto-y-inmutabilidad/`. [Último acceso: 12-02-2024].

[6] Danil Pristupov. Fork. `https://fork.dev/`. [Último acceso: 12-02-2024].

[7] Docker Inc. Docker. `https://docs.docker.com/`. [Último acceso: 12-02-2024].

[8] Gunnar Morling, Andreas Gudian, Sjaak Derksen, Filip Hrisafov and the MapStruct community. Mapstruct. `https://mapstruct.org/documentation/stable/reference/html/`. [Último acceso: 12-02-2024].

[9] JetBrains s.r.o. Intellij idea. `https://www.jetbrains.com/help/idea/discover-intellij-idea.html`. [Último acceso: 12-02-2024].

[10] JGraph Ltd. Draw.io. `https://www.drawio.com/blog`. [Último acceso: 12-02-2024].

[11] Kasra Madadipouya. Test Containers. `https://www.geekyhacker.com/spring-boot-mysql-integration-tests-with-testcontainers/`. [Último acceso: 12-02-2024].

[12] Maida, EG, Pacienzia, J. Metodologías de desarrollo de software. `https://repositorio.uca.edu.ar/handle/123456789/522`. [Último acceso: 12-02-2024].

[13] Meta Open Source. React. `https://es.react.dev/`. [Último acceso: 12-02-2024].

[14] Microsoft. Visual studio code. `https://code.visualstudio.com/docs`. [Último acceso: 12-02-2024].

[15] Okta, Inc. Jwt. `https://jwt.io/introduction`. [Último acceso: 12-02-2024].

[16] ORACLE. Mysql. `https://dev.mysql.com/doc/refman/8.0/en/introduction.html`. [Último acceso: 12-02-2024].

[17] Postman, Inc. Postman. `https://www.postman.com/`. [Último acceso: 12-02-2024].

[18] Remix Software, Inc. React router. `https://reactrouter.com/en/main/start/overview`. [Último acceso: 12-02-2024].

[19] Leonard Richardson and Sam Ruby. *RESTful web services*. .ºReilly Media, Inc.", 2008.

[20] Software Freedom Conservancy. Git. `https://git-scm.com/docs`. [Último acceso: 12-02-2024].

[21] The Axios Project. Axios. `https://axios-http.com/es/docs/intro`. [Último acceso: 12-02-2024].

[22] VMware Tanzu. Spring. `https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/`. [Último acceso: 12-02-2024].

9. ANEXO

9.1. Plantillas de Casos de Uso

Como se menciona en el capitulo de Análisis de Requisitos, a continuación, en este apartado se recogen el resto de plantillas de los casos de uso.

Tabla 9: Plantilla de Registrar

Nombre	Registar
Id	PCU01
Descripción	El usuario se registra en la aplicación.
Precondiciones	Ninguna.
Flujo Principal	1. El usuario pulsa en el boton Registrarse". 2. El sistema mostrará un formulario con varios campos a rellenar. 3. El usuario introduce sus datos. 4. Se carga la página principal del usuario (Listado de Suscripciones).
Flujos Alternativos	**3.1** El sistema muestra un mensaje comunicando un error a la hora de registrarse debido a fallos a la hora de introducir datos en el formulario.

Tabla 10: Plantilla de Iniciar Sesión

Nombre	Iniciar Sesión
Id	PCU02
Descripción	El usuario inicia sesión en la aplicación.
Precondiciones	El usuario debe tener creada previamente una cuenta.
Flujo Principal	1. El usuario pulsa en el boton “Iniciar Sesión”. 2. El sistema mostrará un formulario con dos campos a rellenar (Correo electrónico y contraseña). 3. El usuario introduce sus credenciales. 4. Se carga la página principal del usuario (Listado de Suscripciones).

Flujos Alternativos	**3.1** El sistema muestra un mensaje comunicando un error a la hora de inicar sesión, ya sea porque no existe ningún usuario con ese correo electrónico o porque la contraseña no es correcta.

Tabla 11: Plantilla de Editar Contraseña Olvidada de Usuario

Nombre	Editar contraseña
Id	PCU03
Descripción	El usuario cambiará su contraseña de usuario en el sistema.
Precondiciones	El usuario debe tener una cuenta registrada en el sistema.
Flujo Principal	1. El usuario pulsa en el boton "He olvidado mi contraseña.[en] el formulario de inicio de sesión. 2. El sistema enviará un correo electrónico a la dirección del usuario con un código. 3. El sistema mostrará una nueva ventana para que el usuario introduzca el código enviado. 4. El usuario introducirá el código. 5. El sistema mostrará una nueva ventana para que el usuario pueda introducir su nueva contraseña dos veces. 6. El usuario introduce ambas contraseñas e inicia sesión en el sistema
Flujo Alternativo	**4.1** El sistema muestra un mensaje de error diciendo que el código introducido no es correcto. **6.1** El sistema muestra un mensaje de error diciendo que las nuevas contrseñas no coinciden.

Tabla 12: Plantilla de Ver Detalles de Usuario

Nombre	Ver detalles de usuario
Id	PCU06
Descripción	El usuario visualiza los detalles de su cuenta.
Precondiciones	El usuario debe haber iniciado sesión.

Flujo Principal	1. El usuario pulsa en el boton con su foto de perfil. 2. El sistema muestra una opción para ir a sus detalles. 3. El usuario pulsa en la opción para acceder a los detalles del usuario. 4. El sistema muestra los detalles del usuario.
Flujos Alternativos	

Tabla 13: Plantilla de Listar Suscripciones

Nombre	Mostrar listado de suscripciones
Id	PCU10
Descripción	El usuario visualiza un listado con sus suscripciones.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton “Lista de Pagos”. 2. El sistema mostrará un listado con todas las suscripciones del usuario con información diversa.
Flujo Alternativo	**2.1** El sistema muestra un mensaje comunicando que no hay ninguna suscripción registrada en el sistema.

Tabla 14: Plantilla de Ordenar Suscripciones

Nombre	Ordenar listado de suscripciones
Id	PCU11
Descripción	El usuario ordena el listado de suscripciones.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción en el sistema.
Flujo Principal	1. El usuario pulsa sobre la columna sobre la que quiere ordenar la lista, una vez para orden ascendente, dos veces para orden descendente y tres veces para reestablecer la lista. 2. El sistema mostrará un listado con todas las opciones ordenadas en función de la columna y de la forma seleccionada.

Flujo Alternativo	**2.1** El sistema no ordena la lista porque no suscripciones en el sistema.

Tabla 15: Plantilla de Mostrar Jeraraquía de Suscripciones

Nombre	Mostrar jerarquía de suscripciones
Id	PCU12
Descripción	El usuario visualiza un esquema en árbol organizado en carpetas con las suscripciones.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton "Lista de Pagos". 2. El sistema mostrará un esquema en árbol organizado en carpetas con las suscripciones.
Flujo Alternativo	**2.1** El sistema muestra un mensaje comunicando que no hay ninguna suscripción registrada en el sistema.

Tabla 16: Plantilla de Ver Detalles Suscripción

Nombre	Ver detalles de suscripción
Id	PCU13
Descripción	El usuario visualiza los detalles de una suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción registrada.
Flujo Principal	1. El usuario pulsa en el boton "Ver detalles" en la suscripción deseada en el listado de suscripciones o en la suscripción en el esquema en árbol. 2. El sistema mostrará los detalles de la suscripción seleccionada.
Flujo Alternativo	

Tabla 17: Plantilla de Ver Credenciales Suscripción

Nombre	Visualización de los detalles del usuario de la suscripción

Id	PCU14
Descripción	El usuario visualiza las credenciales de una suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción registrada.
Flujo Principal	1. El usuario pulsa en el boton “Ver detalles” en el listado de suscripciones o en la suscripción en el esquema en árbol. 2. El sistema mostrará las credenciales (ocultas) de la suscripción seleccionada. 3. El usuario pulsa en el botón con un “ojo” para visualizar el texto.
Flujo Alternativo	**1.1** El sistema no muestra las credenciales ya que es un campo opcional y puede estar vacío.

Tabla 18: Plantilla de Copiar Credenciales Suscripción

Nombre	Copiar credenciales de suscripción
Id	PCU15
Descripción	El usuario copia las credenciales de una suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción registrada.
Flujo Principal	1. El usuario pulsa en el boton “Ver detalles” en el listado de suscripciones o en la suscripción en el esquema en árbol. 2. El sistema mostrará las credenciales (ocultas) de la suscripción seleccionada. 3. El usuario pulsa en el botón “Copiar” para copiar el texto en el portapapeles del ordenador.
Flujo Alternativo	**1.1** El sistema no muestra las credenciales ya que es un campo opcional y puede estar vacío.

Tabla 19: Plantilla de Editar Suscripción

Nombre	Editar suscripción

Id	PCU16
Descripción	El usuario edita las credenciales de una suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción registrada.
Flujo Principal	1. El usuario pulsa en el boton "Ver detalles" en el listado de suscripciones o en la suscripción en el esquema en árbol. 2. El sistema mostrará los detalles de la suscripción seleccionada. 3. El usuario pulsará en el botón de "Editar Suscripción" 4. El sistema mostrará un formulario para editar los datos de la suscripción. 5. El usuario introducirá sus nuevos datos. 6. El sistema actualizará estos nuevos datos en la Base de Datos.
Flujo Alternativo	**5.1** El sistema muestra un mensaje comunicando que no se ha podido actualizar los datos por error en los datos introducidos.

Tabla 20: Plantilla de Dar de Baja Suscripción

Nombre	Dar de baja suscripción
Id	PCU17
Descripción	El usuario será redireccionado a la página web de la suscripción.
Precondiciones	El usuario debe tener una sesión iniciada y al menos una suscripción registrada.

Flujo Principal	1. El usuario pulsa en el boton “Ver detalles” en el listado de suscripciones o en la suscripción en el esquema en árbol. 2. El sistema mostrará los detalles de la suscripción. 3. El usuario pulsa en el botón "Dar de baja Suscripción" 4. El sistema redireccionará al usuario a la página web de la suscripción para poder darla de baja. 5. A su regreso el sistema mostrará una pregunta para confirmar si ha realizado la baja de la suscripción. 6. El sistema borra de la Base de Datos la suscripción.
Flujo Alternativo	**4.1** El usuario pulsa no confirma que se haya dado de baja de la suscripción y el sistema muestra el listado de suscripciones intacto.

Tabla 21: Plantilla de Ver Gráfico de Importe Anual

Nombre	Ver gráfico de importe anual
Id	PCU18
Descripción	El sistema mostrará un gráfico de barras con un resumen económico anual.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton “Resumen de Pagos” del menú de navegación. 2. El sistema mostrará un gráfico con un resumen económico anual.
Flujo Alternativo	**2.1** El sistema muestra un mensaje comunicando que no hay ninguna suscripción registrada en el sistema.

Tabla 22: Plantilla de Ver Gráfico de Porcentaje Anual

Nombre	Ver gráfico de porcentaje anual
Id	PCU19

Descripción	El sistema mostrará un gráfico circular con un resumen económico anual detallado por suscripción.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton “Resumen de Pagos” del menú de navegación. 2. El usuario pulsa en el botón “Porcentaje Anual”. 3. El sistema mostrará un gráfico circular con un resumen económico anual detallado por suscripción.
Flujo Alternativo	**2.1** El sistema muestra un mensaje comunicando que no hay ninguna suscripción registrada en el sistema.

Tabla 23: Modificar Parámetros Gráfico

Nombre	Modificar parámetros del gráfico
Id	PCU20
Descripción	El usuario modifica los parámetros del gráfico y este cambia.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton “Resumen de Pagos” del menú de navegación. 2. El sistema mostrará un gráfico con un resumen económico anual y sobre él unos campos para elegir “Año” y “Últimos digitos de la cuenta bancaria” que servirán para filtrar el gráfico". 3. El usuario elige los parámetros 4. El sistema modifica el gráfico para adaptarse a los parámetros.
Flujo Alternativo	**2.1** El sistema muestra un mensaje comunicando que no hay ninguna suscripción registrada en el sistema.

Tabla 24: Plantilla de Consultar Tutorial

Nombre	Mostrar Tutorial

Id	PCU21
Descripción	El sistema mostrará un tutorial sobre como usar la aplicación.
Precondiciones	El usuario debe tener una sesión iniciada.
Flujo Principal	1. El usuario pulsa en el boton “Ayuda”. 2. El sistema mostrará un mensaje indicando si se quiere visualizar un tutorial de como se usa la aplicación. 3. El usuario pulsa en “Sí”. 4. El sistema mostrará un tutorial paso a paso sobre como usar la aplicación.
Flujo Alternativo	**2.1** El usuario pulsa en “No” y se vuelve a la página del listado de Suscripciones.