



# UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO  
INGENIERÍA EN INFORMÁTICA

## MyPortfolio: Plataforma web de monitorización de activos

**Autor**  
Miguel Bravo Campos

**Directores**  
David Griol Barres



Escuela Técnica Superior de Ingenierías Informática y de  
Telecomunicación

—  
Granada, junio de 2025



# **MyPortfolio: Plataforma web de monitorización de activos**

Miguel Bravo Campos

**Palabras clave:** Inversión, Activos, Plataforma Web, Monitoreo, API

## **Resumen**

El propósito principal de este trabajo fin de grado es desarrollar una plataforma digital orientada al seguimiento y organización de activos de inversión de forma centralizada y accesible.

En las últimas dos décadas, el interés por el mundo de la inversión se ha ido extendiendo progresivamente a nivel global. Hoy en día, cualquier persona puede invertir a través de múltiples vías: desde la adquisición de acciones de empresas o productos financieros ofrecidos por los gobiernos, como las Letras del Tesoro, hasta la compra de criptomonedas, cuyo valor depende, en gran medida, de la confianza y percepción del mercado.

Antes de profundizar, conviene aclarar el concepto de inversión. Según la Real Academia Española, invertir es el “empleo, gasto o colocación de una cantidad de dinero en aplicaciones, negocios, valores, etc., para obtener ganancias”. Es decir, invertir implica destinar recursos habitualmente económicos con la expectativa de obtener un beneficio a cambio.

Dada la diversidad de activos y plataformas disponibles actualmente, surge la necesidad de poder organizar todas estas inversiones de manera unificada. En lugar de tener que consultar diversas aplicaciones o sitios web para conocer el estado de cada activo, resulta mucho más eficiente contar con una herramienta que centre toda esta información.

Con esta finalidad nace **MyPortfolio**, una aplicación web desarrollada en dos grandes módulos: un **back-end** y un **front-end**, los cuales se comunican entre sí mediante una API propia. Además, se apoya en una API externa (Alpha Vantage) para la obtención de datos de mercado en tiempo real.

En conclusión, **MyPortfolio** permite al usuario consultar, gestionar y visualizar todas sus inversiones, ya sean acciones, fondos indexados, ETFs o criptomonedas desde un único lugar. Esto evita la necesidad de acceder a múltiples plataformas, ahorrando tiempo y mejorando la experiencia de seguimiento financiero. Además, el diseño del sistema ha sido concebido de manera escalable, permitiendo futuras integraciones con brokers para automatizar aún más el proceso de monitoreo y gestión de inversiones. El código completo del proyecto está disponible en abierto en el siguiente enlace de GitHub: <https://github.com/donBravo1011/Trabajo-Final-de-Grado>



# **MyPortfolio: Web platform of assets monitoring**

Miguel Bravo Campos

**Keywords:** Investment, Assets, Web Platform, Monitoring, API

## **Abstract**

The main goal of this Bachelor's Final Degree Project is to develop a digital platform aimed at monitoring and organizing investment assets in a centralized and accessible way.

Over the past two decades, the concept of investing has gradually spread across the globe. Today, anyone can invest through a wide variety of channels: from purchasing shares of a company or government financial products such as Treasury bills, to acquiring cryptocurrencies whose value largely depends on market perception and individual trust.

Before delving deeper, it is useful to define what we mean by "investment." According to the Royal Spanish Academy, investment refers to the "use, expenditure or placement of a sum of money in applications, businesses, securities, etc., in order to obtain profits." In other words, to invest is to allocate resources, usually money with the expectation of generating a return.

Given the wide range of assets and platforms available today, the need arises for a tool that can help organize them all in one place. Rather than checking several different applications or websites to track how each asset is performing, it is far more efficient to centralize this information in a single, unified platform.

This is the idea behind **MyPortfolio**, a **web** application developed with two main components: a **back-end** and a **front-end**, which communicate through a custom API. Additionally, the application integrates with an external API (Alpha Vantage) to obtain real-time market data.

In conclusion, **MyPortfolio** allows users to access, manage, and visualize all their investments whether stocks, index funds, ETFs, or cryptocurrencies in one single place. This eliminates the need to log into multiple platforms, saving time and streamlining the investment tracking process. Moreover, the system has been designed with scalability in mind, making it possible to incorporate future integrations with brokers to further automate the monitoring and management of investments. The complete code of the project is available in open source at the following GitHub link:  
<https://github.com/donBravo1011/Trabajo-Final-de-Grado>



Yo, **Miguel Bravo Campos**, alumno de la titulacion TITULACION de la **Escuela Técnica Superior de Ingenierías Informática y de Telecommunicación de la Universidad de Granada**, con DNI 77147015R, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

La memoria y el código pueden consultarse en el siguiente repositorio de GitHub:  
<https://github.com/donBravo1011/Trabajo-Final-de-Grado>

Fdo:

*Miguel*

Granada a 22 de Mayo de 2025 .



D. **David Griol Barres**, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

**Informan:**

Que el presente trabajo, titulado **MyPortfolio, Monitorización de activos**, ha sido realizado bajo su supervisión por **Miguel Bravo Campos**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 22 de Mayo de 2025.

**Los directores:**

**David Griol Barres**



# **Agradecimientos**

A mi familia por apoyarme incondicionalmente

A todos los profesores que me han instruido durante toda esta etapa

A mi tutor por apoyarme y ayudarme siempre que lo he necesitado

# Índice general

<b>CAPÍTULO 1: Introducción</b>	<b>17</b>
1.1. Motivación	17
1.2. Objetivos	19
1.3. Objetivos específicos	19
1.4. Planificación	20
1.5. Presupuesto	24
1.6. Estructura de la memoria	26
<b>CAPÍTULO 2: Estado del arte</b>	<b>27</b>
2.1. Dominio del problema a resolver	27
2.2. Metodologías y tecnologías de base analizadas	28
2.2.1. Desarrollo del Back-End	28
2.2.2. Sistema de gestión de bases de datos (SGBD)	30
2.2.3. Desarrollo del Front-End	32
2.3. Aplicaciones similares	34
2.3.1. Análisis de CoinTracking	34
2.3.2. Análisis de The Dividend Tracker	35
2.3.3. Análisis de Sharesight	37
2.4. Contexto	38
2.4.1. Características principales de las criptomonedas	39
2.4.2. ¿Qué es la BlockChain?	39
2.4.4. ¿Cómo funcionan las redes de criptomonedas?	41
2.4.5. Mineros y validadores: el motor de las redes blockchain	43
2.4.6. ¿Qué es el halving y cómo afecta al valor de Bitcoin?	44
2.4.7. Formas de conseguir o comprar criptomonedas	45
2.4.8. Almacenamiento de criptomonedas: tipos de carteras digitales (wallets)	47
<b>CAPÍTULO 3: Análisis de requisitos</b>	<b>49</b>
3.1. Análisis de requisitos	49
3.1.1. Requisitos funcionales	49
3.1.2. Requisitos no funcionales	50
3.2. Casos de uso	51
3.2.1. Descripción de actores	52
3.2.2. Descripción casos de uso	53
3.3 Diagrama de Casos de Uso	67
<b>CAPÍTULO 4: Diseño e implementación</b>	<b>69</b>
4.1 Arquitectura del sistema	69
4.2. Tecnologías utilizadas	81
4.2.1 Back End	81
4.2.2 Base de datos	82
4.2.3 Front End	82
4.2.4 Herramientas Utilizadas	83
4.3. Diseño del sistema	84
4.3.1 Diseño del modelo de datos	84
4.3.2 Estructura de carpetas y ficheros del proyecto	86
4.3.3 Endpoints de la API REST	87

4.4. Implementación del frontend	90
4.4.1 Componentes principales	90
4.4.2 Consumo de la API	97
4.4.3 Gestión del estado de autenticación	100
4.5. Implementación del backend	100
4.5.1. Explicación de la lógica de negocio	101
4.5.2. Conexión con la base de datos	102
4.5.3. Controladores y servicios	102
4.6. Integración con APIs externas	104
4.6.1. Alpha Vantage	104
4.7. Integración del modelo de lenguaje	107
4.8. Pruebas y validación	108
4.9. Despliegue	108
4.10. Retos encontrados y soluciones aplicadas	109
<b>CAPÍTULO 5: Conclusiones y trabajos futuros</b>	<b>110</b>
5.1. Grado de consecución de los objetivos específicos	110
5.2. Trabajos futuros	111
<b>Bibliografía</b>	<b>113</b>

# Índice de tablas

Tabla 1.1 Resumen de Costes Estimados.....	25
Tabla 1.2 Tecnologías inicialmente consideradas.....	28
Tabla 1.3 Comparativa entre Spring Boot y Django.....	29
Tabla 1.4 Tecnologías consideradas.....	30
Tabla 1.5 Comparativa entre las principales opciones.....	31
Tabla 1.6 Tecnologías inicialmente consideradas.....	32
Tabla 1.7 Comparativa entre React y Angular.....	33
Tabla 1.8 CU-1. Registrar un nuevo usuario.....	53
Tabla 1.9 CU-2. Iniciar sesión en el sistema.....	54
Tabla 1.10 CU-3. Salir de la cuenta del usuario.....	54
Tabla 1.11 CU-4. Editar perfil del usuario.....	55
Tabla 1.12 CU-5. Eliminar cuenta de usuario.....	56
Tabla 1.13 CU-6. Ver datos del perfil.....	56
Tabla 1.14 CU-7. Consultar usuario por correo electrónico.....	57
Tabla 1.15 CU-8. Crear una nueva cartera.....	57
Tabla 1.16 CU-9. Ver todas las carteras del usuario.....	58
Tabla 1.17 CU-10. Ver los detalles de una cartera específica.....	58
Tabla 1.18 CU-11. Editar los datos de una cartera.....	59
Tabla 1.19 CU-12. Eliminar una cartera.....	59
Tabla 1.20 CU-13. Consultar carteras por ID de usuario o por email.....	60
Tabla 1.21 CU-14. Añadir una nueva transacción.....	61
Tabla 1.22 CU-15. Ver una transacción por ID.....	61
Tabla 1.23 CU-16. Eliminar una transacción.....	62
Tabla 1.24 CU-17. Consultar el precio actual de una acción.....	62
Tabla 1.25 CU-18. Consultar el precio actual de una criptomoneda.....	63
Tabla 1.26 CU-19. Consultar el precio actual de un ETF.....	63
Tabla 1.27 CU-20. Realizar una consulta al modelo de lenguaje.....	64

# Índice de figuras

Figura 1.1 Comparativa del rendimiento acumulado de diferentes activos durante el año 2023....	18
Figura 1.2 Planificación temporal inicial del proyecto.....	23
Figura 1.3 Planificación temporal final del proyecto.....	24
Figura 1.4 Diagrama de casos de uso.....	68
Figura 1.5 Diagrama de funcionamiento de Entities en el Back End.....	71
Figura 1.6 Diagrama de funcionamiento de Repositories en el Back End.....	72
Figura 1.7 Diagrama de funcionamiento de Services Interfaces en el Back End.....	74
Figura 1.8 Diagrama de funcionamiento de Services Implementation en el Back End.....	76
Figura 1.9 Diagrama de funcionamiento de Controllers en el Back End.....	78
Figura 1.10 Diagrama de funcionamiento del Back End.....	79
Figura 1.11 Diagrama de funcionamiento del Front End.....	80
Figura 1.12 Diagrama de funcionamiento del Sistema.....	81
Figura 1.13 Diseño del modelo de datos.....	85
Figura 1.14 Página Dashboard de la plataforma web.....	91
Figura 1.15 Página Portfolio de la plataforma web.....	91
Figura 1.16 Página Transaction de la plataforma web.....	92
Figura 1.17 Página Register de la plataforma web.....	92
Figura 1.18 Página Login de la plataforma web.....	93
Figura 1.19 Página Profile de la plataforma web.....	93
Figura 1.20 Página Editar Perfil de la plataforma web.....	94
Figura 1.21 Página Cambiar Contraseña de la plataforma web.....	94
Figura 1.22 Página Modelo de lenguaje de la plataforma web.....	95
Figura 1.23 Página Buscador de activos de la plataforma web.....	95
Figura 1.24 Página Crear Portfolio de la plataforma web.....	96
Figura 1.25 Página Editar Portfolio de la plataforma web.....	96
Figura 1.26 Página Comprar/Vender Activos de la plataforma web.....	97



# CAPÍTULO 1: Introducción

En este capítulo se expone la motivación que ha impulsado el desarrollo del proyecto, centrada en la necesidad de una herramienta unificada para el seguimiento de inversiones en acciones y criptomonedas. Se detallan los objetivos generales y específicos, así como la planificación del trabajo dividida en fases claramente estructuradas. También se incluye una estimación de costes y una planificación temporal orientativa. Finalmente, se describe la organización de la memoria, que permite comprender de forma progresiva el enfoque, desarrollo y resultados obtenidos a lo largo del proyecto.

## 1.1. Motivación

En el contexto histórico actual, los inversores particulares están adquiriendo una relevancia sin precedentes. Hoy en día, el comportamiento y el miedo de estos pequeños inversores pueden provocar fluctuaciones significativas en los mercados financieros, algo que anteriormente solo estaba al alcance de grandes fortunas o instituciones. Un claro ejemplo de este fenómeno se ha visto recientemente con los efectos de las políticas arancelarias y la guerra comercial impulsada por Donald Trump, donde los movimientos del mercado reflejaron de inmediato las reacciones del público inversor.[1]

Este creciente interés por parte del inversor minorista ha generado una necesidad evidente de organización y diversificación en sus activos. Actualmente, existen múltiples formas de inversión y una amplia gama de activos entre los cuales elegir.

Las vías tradicionales de inversión continúan siendo relevantes: acciones, fondos indexados, ETFs o incluso renta fija siendo los bonos del tesoro de Estados Unidos (Treasury Bonds) uno de los instrumentos más reconocidos a nivel global, ampliamente adquiridos por otros países como parte de sus reservas.

Asimismo, los bienes raíces han sido históricamente uno de los activos preferidos para invertir, ya sea en viviendas, edificios o naves industriales. Sin embargo, este sector tampoco ha estado exento de riesgos, como se evidenció en la crisis inmobiliaria de 2008, cuando se generó una burbuja que elevó los precios a niveles insostenibles.

No obstante, en los últimos años han surgido nuevas formas de inversión, entre las cuales destaca el creciente interés institucional por las criptomonedas. Un ejemplo notable es el del gobierno de Estados Unidos, que ha comenzado a conformar una reserva estratégica de Bitcoin, acumulando cerca de 200.000 unidades de esta criptomoneda. Bitcoin, la más reconocida del ecosistema, ha llegado a alcanzar valores cercanos a los 110.000 dólares, consolidándose como un activo escaso similar al oro y cada vez más considerado por diversos países como un posible valor refugio a largo plazo.[2]

La figura 1.1 muestra una comparativa del rendimiento acumulado entre distintos activos financieros a lo largo del año 2023. En ella se observa que Bitcoin ha sido el activo con mayor crecimiento, con una revalorización cercana al 80%, seguido de Ethereum con un aumento del 40,45%. Por otro lado, índices bursátiles tradicionales como el NASDAQ 100 y el S&P 500 presentan rendimientos más moderados, del 32,19% y 12,71% respectivamente. Esta comparación pone de manifiesto la alta

rentabilidad, pero también la volatilidad, que puede ofrecer el mercado de criptomonedas frente a los activos financieros convencionales.[3]

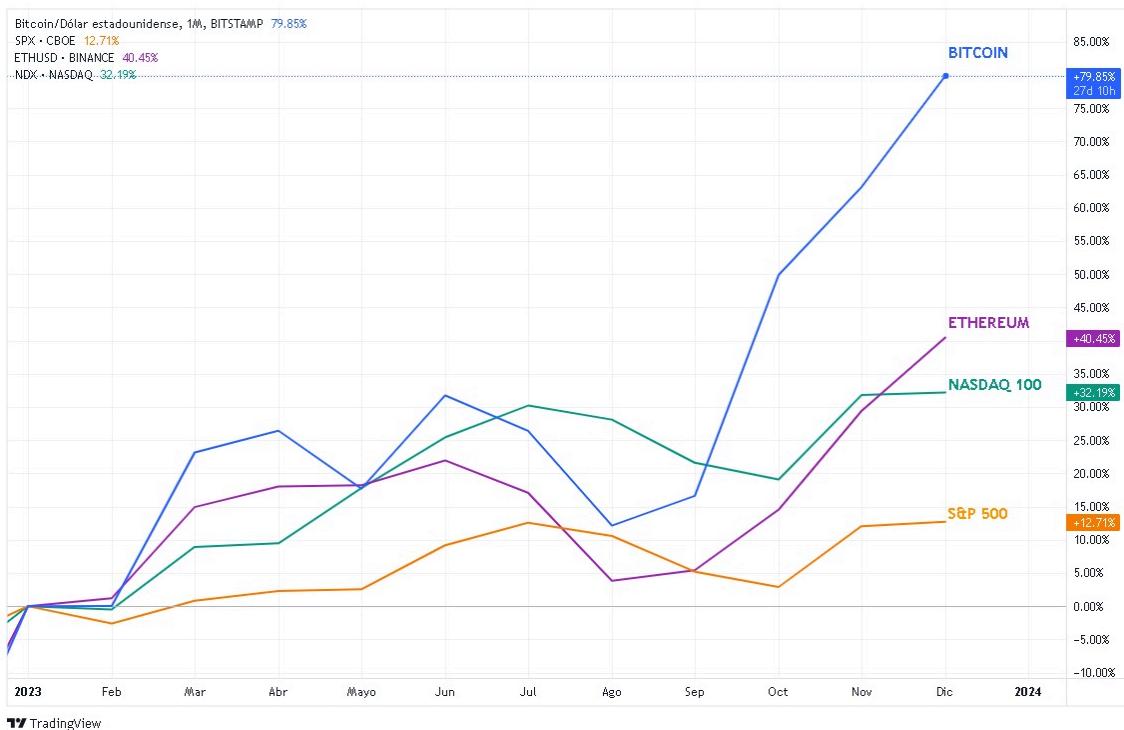


Figura 1.1 Comparativa del rendimiento acumulado de diferentes activos durante el año 2023.

Además de las formas tradicionales de inversión ya mencionadas, existen muchas otras alternativas que han ganado popularidad en los últimos años. Algunas de ellas incluyen la compraventa de relojes de lujo, el coleccionismo de artículos firmados por personalidades reconocidas, o incluso el uso de plataformas de *crowdlending* y *crowdfunding*[4]. En el caso del *crowdlending*, el inversor presta dinero a través de una plataforma que, a su vez, lo destina a particulares o empresas, quienes devuelven el préstamo con intereses. Por otro lado, el *crowdfunding* funciona de forma similar, pero está enfocado en el sector inmobiliario: el dinero del inversor se destina a una promotora para financiar la construcción de un edificio u otro bien raíz, quedando bloqueado durante un periodo que puede ir de seis meses a dos años.

También han surgido métodos de inversión menos convencionales, como el coleccionismo de cartas. Lo que en su momento fue visto como un simple juego, hoy puede convertirse en una inversión millonaria. Prueba de ello es la carta *Topps Mickey Mantle* de 1952, subastada por 12,6 millones de dólares en agosto de 2022, considerada una de las más valiosas del mundo.

A pesar de la variedad de métodos disponibles, en este proyecto nos enfocaremos en las inversiones tradicionales dentro del mercado bursátil. Actualmente, existen numerosos *brokers* que permiten invertir en los distintos tipos de activos presentes en bolsa. Cada uno de ellos suele estar especializado en un producto específico: acciones, fondos, criptomonedas, entre otros. Por esta razón, un inversor particular que busque una cartera diversificada suele acabar utilizando entre tres y cuatro plataformas diferentes, lo que a medio plazo puede generar confusión y desorganización. Revisar cada aplicación diariamente para seguir la evolución de los activos se vuelve una tarea tediosa y poco eficiente.

Frente a este escenario, surge la necesidad de una herramienta que permita centralizar la información financiera del inversor en un solo lugar. Aunque existen plataformas como *The Dividend Tracker*,<sup>[5]</sup> centrada en el seguimiento de ingresos por dividendos, o *CoinTracking*,<sup>[6]</sup> especializada en criptomonedas, ninguna ofrece una visión completa e integrada de todos los activos. Por tanto, este proyecto propone el desarrollo de una aplicación digital que permita al usuario visualizar de forma rápida e intuitiva toda su cartera, independientemente del tipo de activo o plataforma utilizada. El objetivo es facilitar la gestión financiera del inversor particular, permitiéndole tomar decisiones informadas sin perder tiempo ni eficiencia.

La elección de este proyecto nace de una experiencia personal. Hace algunos años descubrí mi interés por el mundo financiero, lo que me llevó a explorar e invertir en muchos de los activos que he mencionado. Con el tiempo, identifiqué de forma clara esta necesidad de organización, y al hablar con otros inversores, confirmé que muchos compartían el mismo problema. Por eso considero que una solución como la que planteo tiene un gran potencial y podría responder a una demanda real y creciente entre quienes, como yo, buscan mejorar su experiencia en el mundo de las inversiones.

## 1.2. Objetivos

El objetivo principal de este proyecto es facilitar la gestión de inversiones a los inversores particulares que disponen de carteras diversificadas distribuidas en distintas aplicaciones. Muchos de ellos pierden un tiempo valioso cada día consultando manualmente cada plataforma. Para resolver este problema, se propone desarrollar una plataforma digital simple, intuitiva y accesible para cualquier tipo de usuario, independientemente del tipo de activo que posea. Esta plataforma permitirá consultar en tiempo real el precio de cualquier activo que coticen en bolsa, lo que facilitará la toma de decisiones sobre posibles movimientos en el mercado. Además, incorporará un modelo de lenguaje que servirá de apoyo para aquellos usuarios que necesiten orientación sobre el contexto actual, información de activos específicos o resolver cualquier duda relacionada con sus inversiones.

## 1.3. Objetivos específicos

Con el objetivo principal de desarrollar una aplicación web que permita a los usuarios gestionar sus carteras de inversión de forma personalizada, sencilla y eficiente, se han definido una serie de objetivos específicos que guiarán la dirección del proyecto. Esta herramienta pretende cubrir una necesidad cada vez más evidente: el seguimiento en tiempo real de los activos financieros que posee un usuario, combinando una interfaz clara, funcionalidades útiles y tecnologías avanzadas como la inteligencia artificial.

Para lograr este objetivo general, se plantean los siguientes objetivos específicos:

- **Analizar plataformas existentes de seguimiento de carteras de inversión,** tales como The Dividend Tracker, Coin Tracking, Sharesight, con el fin de identificar sus funcionalidades, fortalezas y debilidades. Este análisis permite extraer ideas inspiradoras y detectar oportunidades de mejora que pueden

incorporarse al desarrollo del sistema, aportando así un valor diferencial.

- **Integrar fuentes de datos financieras en tiempo real**, utilizando APIs como Alpha Vantage o similares, que permitan a los usuarios consultar el precio actual de acciones, criptomonedas y ETFs directamente desde la aplicación, sin necesidad de recurrir a plataformas externas. Esta integración mejora la experiencia de usuario y centraliza la información relevante en un único lugar.
- **Permitir una gestión flexible y totalmente personalizable de las carteras**, de forma que cada usuario pueda crear, modificar y organizar sus carteras según sus criterios, intereses o estrategias de inversión. Este enfoque otorga al usuario el control total sobre la estructura de su cartera.
- **Ofrecer información útil sobre los activos mediante un modelo de lenguaje**, que permite al usuario hacer preguntas o solicitar explicaciones relacionadas con productos financieros, sin llegar a proporcionar recomendaciones de inversión específicas. Esto se traduce en una funcionalidad educativa e informativa, basada en IA, que complementa la toma de decisiones del usuario.

Con estos objetivos, el proyecto aspira a aportar una solución moderna, robusta y orientada al usuario, que combine datos en tiempo real, personalización y soporte inteligente, dentro de una plataforma accesible desde cualquier dispositivo.

## 1.4. Planificación

En esta sección se describen detalladamente las distintas fases que componen el desarrollo del proyecto, así como la estimación de tiempos asociada y la planificación temporal establecida, organizando la explicación de forma estructurada para ofrecer una visión clara del proceso completo.

El proyecto se divide en una serie de fases estructuradas que permiten organizar, planificar y evaluar tanto los recursos humanos como los costes temporales asociados. Dado que el desarrollo de **myPortfolio**, una plataforma digital para la organización de activos de inversión, implica el uso de diversas tecnologías como Java, Spring Boot y React, así como conocimientos básicos sobre el ámbito financiero, se consideró necesario dividir el trabajo en etapas bien definidas. Esta estructuración ha permitido abordar el proyecto de forma progresiva, facilitando la adquisición de competencias técnicas a lo largo del desarrollo.

- **Fase 0: Planteamiento del proyecto.**

En esta fase se identificó el problema a resolver: la falta de herramientas personalizables y accesibles para la gestión y seguimiento de carteras de inversión. Se establecieron los objetivos del proyecto y se analizó la viabilidad del mismo, tanto desde el punto de vista técnico como práctico.

- Identificación del problema: necesidad de una herramienta accesible y personalizable para la gestión de carteras de inversión.

- Establecimiento de objetivos funcionales y técnicos.
  - Análisis preliminar de posibles tecnologías a emplear (Java, Spring Boot, React).
  - Evaluación inicial de viabilidad técnica y temporal del proyecto.
  - Estimación: 25 horas.
- **Fase 1: Aprendizaje de tecnologías necesarias.**  
Al tratarse de un proyecto full-stack, fue necesario dedicar una parte inicial al aprendizaje de tecnologías fundamentales. Se repasaron conceptos de Java, se adquirieron conocimientos sobre el framework Spring Boot para el desarrollo del backend, y posteriormente se estudió React para la construcción del frontend.
- Revisión de fundamentos de Java y prácticas recomendadas en desarrollo backend.
  - Estudio y pruebas con el framework Spring Boot.
  - Aprendizaje de conceptos fundamentales de React y desarrollo de interfaces.
  - Exploración de herramientas de desarrollo (IDE, control de versiones, gestores de dependencias).
  - Estimación: 150 horas.
- **Fase 2: División y análisis del proyecto.**  
Una vez adquirido el conocimiento técnico básico, se procedió a dividir el sistema en módulos funcionales. Se definieron las funcionalidades principales de la plataforma (registro de usuarios, gestión de activos, visualización de carteras, etc.) y se diseñaron los esquemas de base de datos y la arquitectura general del sistema.
- Definición de las funcionalidades clave: gestión de usuarios, carteras, activos y visualización.
  - Diseño de la arquitectura general del sistema y modelo de datos.
  - Análisis de requisitos funcionales y no funcionales.
  - Preparación de diagramas y estructuras lógicas previas a la codificación.
  - Estimación: 50 horas.
- **Fase 3: Implementación.**  
Esta etapa consistió en el desarrollo del backend y frontend. Se construyeron los servicios REST en Spring Boot y la interfaz de usuario en React, asegurando la comunicación entre ambos mediante API. También se implementó la persistencia de datos mediante una base de datos relacional.

- Desarrollo del backend con Spring Boot: servicios REST, persistencia de datos y control de usuarios.
  - Construcción del frontend con React: formularios, vistas dinámicas, comunicación con API.
  - Conexión e integración entre frontend y backend.
  - Refactorización de código y mejora progresiva del diseño de interfaz.
  - Estimación: 140 horas.
- **Fase 4: Pruebas y tests.**  
Durante esta fase se realizaron pruebas funcionales y de integración para verificar el correcto funcionamiento del sistema. Se validaron casos de uso habituales y se corrigieron errores detectados durante el proceso de verificación.
- Verificación de funcionalidades mediante pruebas manuales e interactivas.
  - Corrección de errores de integración y problemas de validación de datos.
  - Simulación de flujos de uso reales.
  - Ajustes finales en la experiencia de usuario.
  - Estimación: 20 horas
- **Fase 5: Documentación.**  
Finalmente, se elaboró la memoria del proyecto, incluyendo la descripción del sistema, su arquitectura, tecnologías utilizadas, fases de desarrollo, pruebas realizadas y conclusiones. Esta fase también incluye la preparación de anexos técnicos y materiales complementarios.
- Redacción de la memoria del proyecto: introducción, análisis, diseño, desarrollo y conclusiones.
  - Inclusión de diagramas técnicos y capturas de pantalla de la aplicación.
  - Elaboración de anexos y guía técnica para futuros desarrolladores.
  - Revisión ortográfica, técnica y maquetación final del documento.
  - **Estimación:** 80 horas.

En la figura 1.2, se presenta el diagrama de Gantt con la planificación temporal inicial que fue planteado al comienzo del proyecto para planificar y distribuir las tareas a lo largo del tiempo. Durante el desarrollo del TFG, he mantenido reuniones semanales con mi tutor con el objetivo de definir los objetivos específicos de cada semana, revisar si los del periodo anterior se habían cumplido y ajustar la planificación en caso de ser necesario. Esta metodología de trabajo ha permitido un seguimiento constante del avance del proyecto, facilitando la toma de decisiones y la adaptación ante posibles imprevistos.



Figura 1.2 Planificación temporal inicial del proyecto

La figura 1.3 es el diagrama de Gantt final que refleja la planificación real del proyecto una vez concluido. A lo largo del desarrollo, ha sido necesario realizar diversos ajustes en los plazos y tareas inicialmente establecidos. Estos cambios se han debido, principalmente, a la imposibilidad de cumplir algunos objetivos en los tiempos previstos, así como a la redefinición de ciertos objetivos tras las reuniones semanales con el tutor. Gracias a esta flexibilidad, se ha podido adaptar el proyecto a las necesidades reales y optimizar los recursos disponibles, asegurando así la viabilidad y coherencia del trabajo realizado.

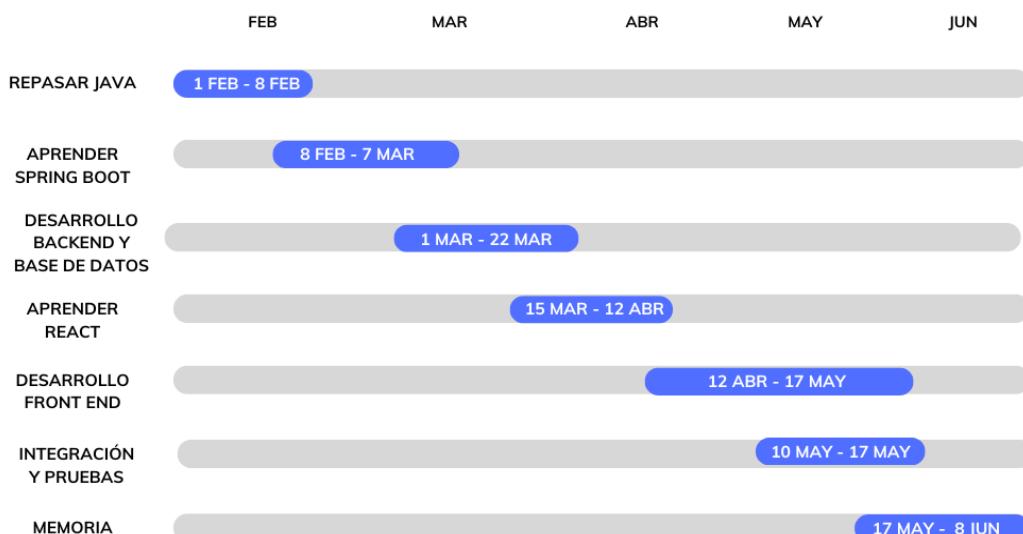


Figura 1.3 Planificación temporal final del proyecto

## 1.5. Presupuesto

En esta sección se presenta un desglose detallado de los costes asociados al proyecto, incluyendo los costes de personal y de equipamiento necesarios para su desarrollo. Además, se ofrece un resumen global de los costes estimados, proporcionando una visión general del presupuesto requerido.

### Costes de Personal y Equipamiento

#### 1. Costes de Personal

El trabajo en este proyecto comenzó en febrero de 2025, dedicando aproximadamente 4 horas diarias de lunes a viernes. Esto se traduce en un total estimado de **400 horas** a lo largo de **5 meses**, con una carga semanal de 20 horas. Esta dedicación ha sido esencial para avanzar en las diferentes fases del proyecto, desde la planificación hasta el desarrollo y pruebas.

#### 2. Costes de Hardware

Para el desarrollo de la plataforma, se ha requerido un **ordenador de sobremesa** de alto rendimiento, acompañado de sus periféricos y pantallas, cuyo coste total asciende a **1.500 €**. Este equipo es esencial para el desarrollo del software y para ejecutar las herramientas de programación y diseño de la plataforma.

#### 3. Costes de Software

El software necesario para el proyecto incluye diversas licencias y suscripciones:

- **SQL Server Management**: para gestionar y alojar la base de datos.
- **Licencia para IntelliJ IDEA**: un entorno de desarrollo integrado (IDE) esencial para el desarrollo del software.
- **Licencia de Figma**: herramienta de diseño para la creación de la interfaz de usuario.
- **Servicios en la nube**: para alojar y ejecutar los procesos del proyecto. Aunque inicialmente el proyecto podría ejecutarse localmente, se plantea la posibilidad de migrar a la nube si el tamaño de la aplicación aumenta a medida que se escala.

#### 4. Otros Costes

- **Marketing**: para promover la plataforma y atraer nuevos usuarios, se ha contemplado una inversión en campañas de publicidad digital. Esto incluye anuncios en plataformas como YouTube, Instagram, LinkedIn y X (anteriormente Twitter). La difusión en redes sociales será clave para aumentar la visibilidad de la plataforma.
- **Registro de la empresa y propiedad intelectual**: También se estima que será necesario registrar la **empresa asociada a la plataforma** y, si es necesario,

**patentar el software** para proteger la propiedad intelectual.

## Resumen de Costes Estimados

La tabla 1.1 muestra una estimación de los principales costes del proyecto, incluyendo personal, hardware, software, marketing y gastos legales. Algunos valores son aproximados debido a su naturaleza variable.

Concepto	Coste Estimado
Costes de Personal	400 horas a 20h/semana
Hardware (ordenador y periféricos)	1.500 €
Software (licencias y suscripciones)	Variable (aprox. 500 - 1.000 €)
Marketing (campañas publicitarias)	Variable (depende del alcance)
Registro de empresa y patente	Variable (aproximadamente 500 - 1.000 €)

Tabla 1.1 Resumen de Costes Estimados

## 1.6. Estructura de la memoria

El contenido de este trabajo se ha estructurado en cinco capítulos principales, cada uno con una función concreta y complementaria dentro del desarrollo del proyecto. A continuación, se describe brevemente el contenido de cada uno de ellos:

- **Capítulo 2:** Se presenta el dominio del problema, también el contexto del mundo de las criptomonedas y los activos financieros digitales. También se analizan distintas metodologías y tecnologías que podrían emplearse para abordar la problemática tratada, así como aplicaciones similares ya existentes en el mercado.
- **Capítulo 3:** Se realiza un análisis de los requisitos del sistema, tanto funcionales como no funcionales. Se identifican los posibles casos de uso que podrá realizar el usuario, y se acompaña el capítulo con su correspondiente diagrama de casos de uso.
- **Capítulo 4:** Se detalla el diseño e implementación del proyecto. Aquí se abordan aspectos como la arquitectura del sistema, las tecnologías utilizadas, el diseño de la solución, la implementación tanto del front-end como del back-end, la integración con APIs externas y modelo LLM, las pruebas realizadas, el proceso de despliegue y los retos encontrados durante el desarrollo.

- **Capítulo 5:** Se realiza una revisión de los objetivos específicos planteados inicialmente y se analiza el grado en que estos han sido alcanzados. Además, se proponen posibles líneas de trabajo futuro que podrían dar continuidad y mejora al proyecto.

Finalmente, se incluyen las referencias bibliográficas consultadas a lo largo del trabajo.

# CAPÍTULO 2: Estado del arte

En este capítulo se presenta el dominio del problema que se aborda en este proyecto. Se revisan las tecnologías y metodologías más apropiadas para dar solución al problema identificado. Además, se incluyen ejemplos de aplicaciones similares ya existentes en el mercado, con el fin de establecer una base comparativa y justificar la necesidad de la propuesta desarrollada en este proyecto. Por último se habla brevemente sobre la historia de las criptomonedas, el funcionamiento de sus redes y conceptos clave como la tecnología blockchain, entre otros aspectos fundamentales.

## 2.1. Dominio del problema a resolver

En la actualidad, la gestión y el seguimiento de activos financieros personales se ha vuelto una tarea cada vez más fragmentada y compleja. La popularización del acceso a distintos mercados financieros ha llevado a muchos usuarios a diversificar sus inversiones entre múltiples tipos de activos: acciones, fondos cotizados (ETFs), criptomonedas, e incluso otros bienes como inmuebles o metales preciosos. Sin embargo, esta diversificación conlleva una consecuencia clara: la necesidad de utilizar múltiples plataformas y aplicaciones para consultar, registrar o analizar el estado de cada clase de activo.

Desde esta perspectiva nace **MyPortfolio**, una plataforma web desarrollada con el objetivo de unificar en un solo lugar la organización, consulta y personalización de carteras de inversión diversificadas. Esta herramienta surge de una necesidad detectada tanto a nivel personal como compartida con muchos otros usuarios del entorno financiero, la falta de una aplicación centralizada y flexible que permita visualizar y gestionar múltiples tipos de activos sin tener que alternar entre varias interfaces. Actualmente, para tener una visión completa de su patrimonio, los usuarios deben consultar diferentes fuentes: aplicaciones bancarias, plataformas de inversión tradicionales, wallets de criptomonedas, hojas de cálculo personales, etc.

Existen soluciones parciales en el mercado que abordan aspectos específicos de esta problemática. Por ejemplo, The Dividend Tracker ofrece un enfoque centrado exclusivamente en el seguimiento de dividendos obtenidos por acciones, mientras que CoinTracking proporciona herramientas potentes para el análisis de carteras de criptomonedas. No obstante, ambos presentan limitaciones importantes: el primero excluye otros tipos de activos y carece de herramientas de personalización profunda, y el segundo se enfoca únicamente en el ecosistema cripto, sin integrar otros activos financieros tradicionales.

La propuesta que representa este Trabajo de Fin de Grado no sólo busca cubrir ese vacío existente, sino también ofrecer un enfoque modular y escalable, que permite a los usuarios organizar sus portafolios como deseen, adaptándose a sus necesidades particulares. Esta flexibilidad es un factor diferencial clave, ya que habilita al usuario para estructurar su información financiera con total libertad agrupando activos según criterios personalizados, o incluso comparando la evolución de distintos tipos de inversión en una sola vista.

A largo plazo, la plataforma plantea la posibilidad de expansión hacia funcionalidades aún más avanzadas, como la integración con brokers, bancos o APIs externas, de manera que los movimientos financieros del usuario (por ejemplo, la compra de una acción o el ingreso de dividendos) se reflejan automáticamente en su portafolio.

Asimismo, se contempla el soporte para nuevas categorías de activos, como bienes raíces, objetos de colección o activos alternativos, reforzando el objetivo de ofrecer una visión patrimonial global.

En definitiva, MyPortfolio se posiciona como una solución tecnológica innovadora en el ámbito de la gestión patrimonial digital, con un enfoque centrado en el usuario, la integración de fuentes y la personalización. Supone una evolución lógica frente a las herramientas actualmente disponibles, al reunir bajo una misma interfaz las funcionalidades que antes requerían el uso de múltiples aplicaciones, mejorando así la experiencia del usuario y optimizando su capacidad para tomar decisiones informadas.

## 2.2. Metodologías y tecnologías de base analizadas

A continuación, se expone el desarrollo de los principales componentes del sistema, abarcando tanto la implementación del back end y su integración con el sistema gestor de bases de datos, como la construcción del front end.

### 2.2.1. Desarrollo del Back-End

Durante la fase de planificación del desarrollo del backend de la plataforma **MyPortfolio**, se evaluaron diversas tecnologías que permiten construir aplicaciones web robustas y escalables. El backend tiene como responsabilidad principal la gestión de la lógica de negocio, las operaciones con la base de datos, la autenticación y el manejo de la API que comunica con el frontend, así como la API que obtiene datos en tiempo real.

La tabla 1.2 recoge un análisis de varias tecnologías consideradas al inicio del proyecto. Aunque todas ofrecen ventajas, fueron descartadas por motivos como complejidad, menor adopción o limitaciones en despliegue.

Lenguaje / Framework	Descripción breve
C++	Lenguaje compilado de alto rendimiento, pero poco usado para desarrollo web moderno. Su complejidad y falta de frameworks maduros lo descartaron rápidamente.[7]
C# con ASP.NET	Tecnología muy utilizada en entornos corporativos, especialmente en ecosistemas Microsoft. Potente, pero menos popular fuera de ese entorno y con menor flexibilidad en despliegues.[8]
PHP	Lenguaje históricamente dominante en el desarrollo web (WordPress, Laravel, etc.), pero con tendencia a la baja en nuevos proyectos debido a la evolución de otras tecnologías más modernas.[9]
Ruby on Rails	Framework muy ágil y productivo, con una curva de aprendizaje baja, pero con una comunidad decreciente y menor adopción actual.[10]

Tabla 1.2 Tecnologías inicialmente consideradas

## Tecnologías finalistas:

Después del análisis inicial, se acotaron las posibilidades a dos alternativas sólidas, modernas y ampliamente utilizadas:

- **Java con Spring Boot**
- **Python con Django**

Ambas permiten la creación de APIs RESTful, ofrecen gran soporte comunitario y poseen una arquitectura orientada a servicios, lo que encaja con los objetivos del proyecto.

La tabla 1.3 muestra una comparativa entre Spring Boot y Django según criterios clave como rendimiento, escalabilidad o soporte para APIs. Finalmente se optó por Spring Boot por su robustez, flexibilidad y enfoque a microservicios.

Criterio	Java + Spring Boot	Python + Django
Lenguaje	Java (estático, tipado, orientado a objetos)	Python (dinámico, legible, orientado a objetos)
Curva aprendizaje de	Moderada-Alta. Requiere más configuración inicial.	Baja. Gran simplicidad para proyectos iniciales.
Velocidad desarrollo de	Ligeramente más lento por su verbosidad.	Muy rápido: incluye ORM, panel de administración, y autenticación por defecto.
Popularidad en empresas	Muy alta. Ampliamente usado en grandes empresas y sistemas financieros complejos.	Alta. Muy usado en startups, prototipos, y productos de rápida salida al mercado.
Documentación y comunidad	Excelente, con abundantes recursos, pero más orientados a perfiles senior.	Excelente, ideal para quienes comienzan en desarrollo backend.
Soporte REST APIs	Potente, con Spring Web y Spring Security.	Muy potente con Django REST Framework.
Escalabilidad	Muy alta. Spring es ideal para microservicios y arquitecturas distribuidas.	Alta, aunque más orientado a monolitos al inicio (también se puede modularizar).
Flexibilidad	Alta, pero más compleja de gestionar al crecer el proyecto.	Muy buena, especialmente en proyectos medianos.

Tabla 1.3 Comparativa entre Spring Boot y Django

Ambas tecnologías son perfectamente válidas para la construcción del backend de una plataforma como **MyPortfolio**. La elección final dependerá del enfoque deseado:

- **Spring Boot** es ideal si se prioriza la escalabilidad futura, la integración con entornos empresariales complejos y se busca un backend altamente estructurado y modular.[11] [12]
- **Django** es óptimo si se busca una implementación rápida, productiva y con menor complejidad inicial, ideal para prototipado rápido y productos mínimos viables. [13]

### 2.2.2. Sistema de gestión de bases de datos (SGBD)

Uno de los pilares fundamentales del desarrollo de la plataforma **MyPortfolio** es el sistema de bases de datos. Este componente se encargará de almacenar de forma estructurada información clave como usuarios, carteras, activos financieros, historial de transacciones, preferencias, entre otros.

En el análisis inicial, se evaluaron tanto **bases de datos relacionales (SQL)** como **bases de datos no relacionales (NoSQL)**, considerando aspectos como flexibilidad, rendimiento, facilidad de integración y escalabilidad.

La tabla 1.4 recoge un conjunto de tecnologías de bases de datos evaluadas durante el desarrollo del proyecto, destacando su tipo, utilidad principal y entorno más adecuado. Se compararon tanto soluciones SQL tradicionales como alternativas NoSQL más flexibles.

Tecnología	Tipo	Descripción breve
<b>MongoDB</b>	NoSQL	Base de datos orientada a documentos. Muy útil para estructuras flexibles y dinámicas.[14]
<b>MariaDB</b>	SQL	Sistema derivado de MySQL, totalmente open source, ideal para desarrollos clásicos.[15]
<b>SQL Server Management Studio</b>	SQL	Herramienta de Microsoft para trabajar con SQL Server. Potente pero más común en entornos corporativos.
<b>SQLite</b>	SQL	Motor ligero, embebido, sin necesidad de servidor. Ideal para pruebas o prototipos.[16]
<b>MySQL Workbench</b>	SQL	Herramienta gráfica de administración para bases de datos MySQL. Permite modelar, consultar y administrar de forma visual. Muy popular.[17]

Tabla 1.4 Tecnologías consideradas

En la tabla 1.5 se comparan distintas opciones de bases de datos según criterios clave como escalabilidad, facilidad de uso o modelo de datos. El objetivo fue identificar la solución más adecuada para las necesidades específicas del proyecto.

Criterio	MySQL (con Workbench)	MariaDB	SQL Server (Management Studio)	MongoDB	SQLite
<b>Modelo de datos</b>	Relacional	Relacional	Relacional	Documental (basado en JSON)	Relacional
<b>Facilidad de uso</b>	Alta, especialmente con interfaz Workbench	Alta	Media (más técnico)	Alta, especialmente con herramientas como Compass o Mongoose	Muy alta (sin instalación compleja)
<b>Flexibilidad de esquema</b>	Baja (esquema rígido)	Baja	Baja	Alta (esquema dinámico)	Baja
<b>Escalabilidad</b>	Alta (dependiendo del diseño)	Alta	Muy alta	Muy alta (distribuido, horizontal)	Limitada
<b>Consistencia de datos (ACID)</b>	Alta	Alta	Muy alta	Media (eventual, configurable)	Alta
<b>Gestión visual</b>	Excelente con Workbench	Limitada (requiere herramientas externas)	Buena con SSMS	Media (herramientas visuales básicas)	Nula (modo consola)
<b>Soporte y comunidad</b>	Muy amplia	Amplia	Corporativa	Muy activa y en expansión	Limitada, pero suficiente
<b>Uso ideal</b>	Aplicaciones web generales	Proyectos open source y ligeros	Aplicaciones empresariales	Datos dinámicos, apps flexibles o en la nube	Prototipos, apps móviles, pruebas

Tabla 1.5 Comparativa entre las principales opciones

Tras este análisis, se observa que tanto **MySQL con Workbench** como **MariaDB** son excelentes opciones para una base de datos relacional en un proyecto como **MyPortfolio**, que requiere estructura, integridad de datos y facilidad de mantenimiento.

- **MySQL + Workbench** destaca por su amplia documentación, estabilidad y **herramientas visuales de gestión y modelado** que aceleran el desarrollo y depuración.

- **MongoDB** es ideal si se desea mayor flexibilidad en la estructura de los datos (por ejemplo, diferentes tipos de activos con atributos variables).
- **SQLite** puede ser útil para las primeras fases o versiones de escritorio del sistema.
- **SQL Server** es potente, pero más orientado a entornos corporativos con integración en ecosistemas Microsoft.

### 2.2.3. Desarrollo del Front-End

El frontend representa la capa visual e interactiva del proyecto **MyPortfolio**, es decir, la parte con la que el usuario final interactúa directamente. La elección de la tecnología adecuada afecta directamente a la experiencia de usuario (UX), el rendimiento de la interfaz, la facilidad de mantenimiento y la escalabilidad del producto.

La tabla 1.6 presenta las principales tecnologías de desarrollo frontend consideradas, destacando sus características y ventajas para crear interfaces web, desde soluciones básicas hasta frameworks modernos y completos.

Tecnología	Descripción
<b>HTML, CSS y JavaScript</b>	Tecnologías base del desarrollo web. Permiten total control, pero requieren más tiempo y esfuerzo para construir interfaces dinámicas y responsivas desde cero.
<b>Twig (motor de plantillas)</b>	Sistema de plantillas muy usado con Symfony y PHP. Facilita la separación entre lógica y presentación, pero menos flexible para aplicaciones SPA modernas.[18]
<b>React</b>	Biblioteca de JavaScript moderna y altamente usada en la industria para crear interfaces reactivas y componentes.[19]
<b>Angular</b>	Framework completo basado en TypeScript, diseñado para aplicaciones grandes y estructuradas, con herramientas integradas desde el inicio.[20]

Tabla 1.6 Tecnologías inicialmente consideradas

La tabla 1.7 compara React y Angular, dos de las tecnologías front-end más populares, analizando sus características clave, facilidad de aprendizaje, ecosistemas y adecuación según el tipo de proyecto y entorno laboral.

Criterio	React (Meta/Facebook)	Angular (Google)
<b>Tipo</b>	Biblioteca para interfaces (UI)	Framework completo (full-stack front-end)
<b>Lenguaje principal</b>	JavaScript (aunque se usa mucho con TypeScript también)	TypeScript (superset de JavaScript)
<b>Curva de aprendizaje</b>	Moderada. Más libertad, pero más decisiones arquitectónicas	Alta. Todo estructurado, pero con más conceptos obligatorios
<b>Comunidad y demanda laboral</b>	Muy alta. Dominante en startups, empresas tecnológicas	Alta, especialmente en entornos corporativos y grandes empresas
<b>Ecosistema</b>	Requiere combinar con otras librerías (router, gestión de estado)	Todo incluido (ruteo, formularios, servicios HTTP, etc.)
<b>Rendimiento</b>	Excelente. Virtual DOM muy eficiente	Muy bueno, aunque algo más pesado por defecto
<b>Productividad inicial</b>	Alta, pero depende de las decisiones de arquitectura	Muy alta si se sigue el estilo Angular (estructura predefinida)
<b>Reusabilidad de componentes</b>	Muy alta (basado en componentes funcionales y hooks)	Alta (componentes y servicios inyectables)
<b>Facilidad de testing</b>	Buena, pero depende de configuración	Muy buena. Herramientas integradas como Karma y Jasmine

Tabla 1.7 Comparativa entre React y Angular

Tanto **React** como **Angular** son opciones modernas, potentes y ampliamente utilizadas en el desarrollo frontend actual. La elección entre uno u otro depende principalmente del enfoque del equipo de desarrollo:

- **React** ofrece más libertad, mayor flexibilidad y un ecosistema muy activo. Es ideal si se desea construir una interfaz ligera, altamente personalizable y escalable en diferentes fases del producto.
- **Angular**, al ser un framework completo, guía más al desarrollador con una estructura clara desde el inicio, lo cual puede acelerar el desarrollo en equipos grandes o con menor experiencia.

## 2.3. Aplicaciones similares

En esta sección se realiza un análisis comparativo de tres herramientas destacadas para el seguimiento de inversiones: CoinTracking, The Dividend Tracker y Sharesight. Se examinan sus principales funcionalidades, ventajas y limitaciones, con el objetivo de evaluar su utilidad en la gestión y visualización de carteras de inversión.

### 2.3.1. Análisis de CoinTracking

**CoinTracking** es una de las plataformas más utilizadas a nivel global para el seguimiento y la gestión de carteras de criptomonedas. Se presenta como una herramienta orientada principalmente a usuarios que operan en mercados de activos digitales, ofreciendo funcionalidades de control de transacciones, generación de informes fiscales y análisis de rendimiento.[21]

CoinTracking permite a los usuarios importar automáticamente sus operaciones desde más de 300 exchanges mediante API, archivos CSV o entradas manuales. Entre sus funcionalidades más destacadas se encuentran:

- **Seguimiento detallado de cartera:** seguimiento de ganancias, pérdidas y valor de mercado de las criptomonedas.
- **Informes fiscales automatizados:** generación de informes compatibles con normativas fiscales de más de 100 países.
- **Soporte multi-activo dentro del mundo cripto:** permite registrar operaciones de trading, DeFi, NFTs, futuros y derivados.
- **Análisis e informes personalizados:** ofrece más de 25 tipos de informes que permiten analizar el rendimiento de las inversiones desde múltiples ángulos.
- **Medidas de seguridad:** cifrado de datos, autenticación en dos pasos y diseño sin custodia para mantener el control en manos del usuario.

Pese a centrarse en un solo tipo de activo, CoinTracking comparte ciertas funcionalidades con la aplicación propuesta en este TFG:

- Ambas plataformas ofrecen monitoreo de criptomonedas.
- CoinTracking y MyPortfolio ofrecen **herramientas de análisis de cartera** y generación de informes financieros.
- Se prioriza la **seguridad del usuario** mediante autenticación avanzada y medidas de protección de datos.

Las diferencias entre CoinTracking y la propuesta de *MyPortfolio* permiten visualizar el alcance más amplio de esta última:

- **Tipos de activos:** CoinTracking se limita exclusivamente a criptomonedas, mientras que *MyPortfolio* pretende integrar activos tradicionales (acciones, ETFs), digitales (cripto), e incluso físicos o alternativos (bienes raíces, cuentas bancarias).
- **Integración financiera:** CoinTracking no ofrece conectividad con brokers o instituciones bancarias tradicionales, lo cual limita la visión global del patrimonio.
- **Interfaz y usabilidad:** diversos usuarios han reportado que la interfaz es compleja y poco amigable para quienes no tienen experiencia técnica.
- **Personalización:** las opciones de visualización y organización de portafolios son limitadas frente a la libertad que plantea *MyPortfolio* para estructurar carteras de forma totalmente adaptable al usuario.

Algunos análisis y valoraciones en línea destacan las siguientes fortalezas y debilidades[22]:

- La generación de **informes fiscales** es una de las funciones más valoradas por los usuarios con alta actividad en criptomonedas.
- Existen críticas sobre la **precisión de las importaciones automáticas**, especialmente cuando se trata de exchanges menos conocidos o activos con estructura compleja.
- También se han identificado carencias en cuanto a la **rapidez y efectividad del soporte técnico**.

### 2.3.2. Análisis de The Dividend Tracker

The Dividend Tracker es una aplicación especializada en el seguimiento de carteras de inversión centradas en acciones que generan dividendos. Su objetivo principal es proporcionar a los inversores una herramienta sencilla y eficiente para monitorear y proyectar sus ingresos por dividendos[23].

Las características principales de esta herramienta son:

- **Visualización de ingresos por dividendos:** La aplicación permite ver los ingresos por dividendos de forma clara y desglosada por mes, año y otros períodos.
- **Integración con cuentas de corretaje:** Ofrece la posibilidad de vincular cuentas de corretaje como M1 Finance, Robinhood, Fidelity, Charles Schwab, Vanguard, entre otras, para actualizaciones automáticas de las posiciones.

- **Calendario de dividendos:** Permite filtrar y visualizar fechas de dividendos estimadas, declaradas y confirmadas, facilitando la planificación financiera.
- **Análisis de cartera:** Proporciona herramientas para verificar la diversificación de la cartera y proyectar el rendimiento futuro mediante calculadoras integradas.
- **Investigación de acciones:** Incluye funciones para investigar acciones potenciales mediante la característica "Dividend Grade", ayudando a los inversores a tomar decisiones informadas.

Las principales similitudes con la plataforma web MyPortfolio son:

- **Seguimiento de ingresos por dividendos:** Ambas aplicaciones permiten a los usuarios monitorizar sus ingresos por acciones con dividendos.
- **Proyecciones financieras:** Ofrecen herramientas para proyectar el rendimiento futuro de las inversiones, facilitando la planificación a largo plazo.
- **Análisis de cartera:** Proporcionan funcionalidades para analizar la diversificación de la cartera de inversiones.

Las diferencias y limitaciones principales entre las plataformas son:

- **Tipos de activos soportados:** The Dividend Tracker se centra exclusivamente en acciones que generan dividendos, mientras que *MyPortfolio* está diseñado para gestionar una gama más amplia de activos, incluyendo criptomonedas, bienes raíces y cuentas bancarias.
- **Integración con brokers y bancos:** Aunque The Dividend Tracker permite la vinculación con ciertos brokers, su integración es limitada en comparación con la propuesta de *MyPortfolio*, que busca una integración más amplia y profunda.
- **Personalización de portafolios:** La capacidad de personalizar y visualizar portafolios en The Dividend Tracker es limitada en comparación con la flexibilidad que ofrece *MyPortfolio*.
- **Interfaz de usuario:** Algunos usuarios han reportado que la interfaz de The Dividend Tracker puede ser compleja y poco intuitiva, especialmente para aquellos sin experiencia técnica.

Algunos análisis y valoraciones en línea destacan las siguientes fortalezas y debilidades[24]:

- Algunos usuarios han expresado frustración con la precisión de las importaciones y la complejidad de la interfaz.

- Otros han destacado la utilidad de la plataforma para la gestión fiscal de dividendos, especialmente para usuarios con un alto volumen de transacciones.

### 2.3.3. Análisis de Sharesight

**Sharesight** es una plataforma en línea diseñada para el seguimiento y análisis de carteras de inversión. Su objetivo principal es proporcionar a los inversores una herramienta integral para monitorizar el rendimiento de sus inversiones, incluyendo acciones, fondos y otros activos financieros[25].

Las características principales de esta herramienta son:

- **Seguimiento automático de inversiones:** Sharesight permite importar automáticamente datos de más de 200 corredores a nivel mundial, facilitando el seguimiento de precios, operaciones, dividendos y acciones corporativas.
- **Análisis de rendimiento:** La plataforma ofrece informes detallados sobre el rendimiento de la cartera, considerando factores como dividendos, fluctuaciones de divisas y reinversiones.
- **Informes fiscales:** Sharesight genera informes fiscales compatibles con las normativas de diversos países, ayudando a los inversores en la preparación de sus declaraciones fiscales.
- **Visualización de cartera:** Proporciona gráficos y tablas que permiten visualizar la diversificación y el rendimiento de la cartera en diferentes períodos.
- **Soporte para múltiples activos:** Además de acciones y fondos, Sharesight permite el seguimiento de criptomonedas, bienes raíces y otros activos no cotizados, ofreciendo una visión completa del patrimonio del inversor.
- **Seguimiento integral de inversiones:** Ambas plataformas permiten a los usuarios monitorizar una amplia gama de activos, proporcionando una visión holística de su cartera.
- **Análisis detallado del rendimiento:** Tanto Sharesight como *MyPortfolio* ofrecen herramientas para analizar el rendimiento de las inversiones, considerando diversos factores que afectan el valor de la cartera.

Las principales similitudes con la plataforma web *MyPortfolio* son:

- **Interfaz de usuario:** Algunos usuarios han señalado que la interfaz de Sharesight puede resultar compleja y poco intuitiva, especialmente para aquellos sin experiencia técnica.
- **Integración con brokers y bancos:** Aunque Sharesight permite la vinculación con ciertos brokers, su integración es limitada en comparación con la propuesta futura de *MyPortfolio*, que busca una integración más amplia y profunda.

Algunos análisis y valoraciones en línea destacan las siguientes fortalezas y debilidades[26][27]:

- Algunos usuarios han expresado frustración con la precisión de las importaciones y la complejidad de la interfaz.
- Otros han destacado la utilidad de la plataforma para la gestión fiscal de inversiones, especialmente para usuarios con un alto volumen de transacciones.

## 2.4. Contexto

El contexto en el que se sitúa este proyecto son los activos financieros desde acciones, fondos, ETFs hasta otras nuevas formas de inversión como criptomonedas, cartas de colección, relojes como hemos dicho anteriormente. También destacamos la que se ha hecho más conocida en los últimos que son las criptomonedas, sobre estas vamos a dar un contexto para todos aquellos que hayan oído hablar sobre ellas pero no sepan todo lo que existe por detrás de estas. Primero que todo cuando hablamos de criptomonedas nos referimos a monedas digitales que también son conocidas como cripto activos. El nacimiento de las criptomonedas se remonta muchos años en el pasado, son fruto del trabajo de matemáticos, científicos y otras personas con una gran perspectiva de futuro.

La historia de los bitcoins y criptomonedas comenzó de forma más oficial en 2008. Pero en realidad sus inicios datan de la década de los 80. Ya que existe un creador llamado David Chaum el cual creó en 1983 un sistema criptográfico que recibió el nombre de eCash, su objetivo era crear una red de dinero que fuera electrónico y criptográfico pero a la vez anónimo y efectivo. Finalmente el sistema que creó Chaum acabó convirtiéndose en el software de un banco estadounidense, cuya finalidad era almacenar dinero digitalmente a través de la firma criptográfica de la entidad bancaria.[28]

La criptomoneda más conocida en la actualidad y a través de la cual se dieron a conocer estas es el Bitcoin. Su origen viene por un artículo publicado en 2008, aunque sus inicios no fueron sencillos y los apasionados de las tecnologías consideraban que se trataba de una herramienta que apenas tenía valor y la tildaban de ser poco práctica. Hoy en día ha dado un giro completo y hasta los mayores gobiernos del mundo están considerando añadirla a sus reservas.[29]

A finales de 2008, una o varias personas anónimas que se presentaron como Satoshi Nakamoto decidieron presentar un documento técnico asociado al Bitcoin. Se publicó un sistema electrónico efectivo que revolucionó todo lo que se había visto. Fue así como Bitcoin salió a la luz y cómo nacieron las primeras criptomonedas.[30]

### 2.4.1. Características principales de las criptomonedas

Las criptomonedas han ganado popularidad en los últimos años no solo por su valor económico, sino por una serie de características que las diferencian de las monedas tradicionales. A continuación, se detallan las más relevantes:

La **descentralización** de las monedas emitidas por bancos centrales, las criptomonedas no dependen de ninguna institución financiera ni de un gobierno. Su funcionamiento se basa en redes distribuidas habitualmente construidas sobre tecnología blockchain que permiten a los usuarios realizar transacciones directamente entre sí, sin intermediarios.

En cuanto a la **seguridad** de las transacciones realizadas con criptomonedas están protegidas mediante técnicas criptográficas avanzadas. Una vez que se registra un movimiento en la red, este queda sellado en un bloque y no puede modificarse. Además, las wallets o carteras digitales que se utilizan para almacenar estos activos cuentan con claves privadas únicas que solo el propietario conoce.

El **anonimato** es una de las características más llamativas es que no es necesario proporcionar datos personales para realizar transacciones. Aunque no son completamente anónimas ya que las operaciones quedan registradas públicamente en la blockchain, sí permiten mantener la identidad del usuario en el anonimato, utilizando direcciones alfanuméricas en lugar de nombres.

La **transparencia** de todas las transacciones quedan registradas de forma pública y permanente en la cadena de bloques. Esto permite que cualquiera pueda consultar el historial completo de movimientos, lo cual aporta una gran transparencia al sistema.

La **accesibilidad global** para usar criptomonedas, no es necesario tener una cuenta bancaria ni vivir en un país con infraestructura financiera avanzada. Basta con tener acceso a internet y una wallet digital. Esto ha abierto nuevas posibilidades para personas que tradicionalmente han estado excluidas del sistema financiero.

La **volatilidad**, aunque no es una característica buscada, sí es una realidad del mercado cripto. El valor de las criptomonedas puede variar bruscamente en cuestión de horas, lo que representa tanto una oportunidad como un riesgo para los inversores.

#### 2.4.2. ¿Qué es la BlockChain?

La **blockchain**, o cadena de bloques, es una tecnología que permite registrar información de forma segura, transparente y sin depender de una entidad central que controle los datos. Su funcionamiento se basa en una red de ordenadores (nodos) que comparten una misma copia de un registro digital, lo que garantiza que todos tengan acceso a la misma información en todo momento.[31]

Cada vez que se realiza una acción dentro de la red, como una transacción económica, esta queda registrada en un bloque. Ese bloque contiene la información del movimiento y un identificador que lo enlaza con el bloque anterior, formando así una cadena. Esta estructura hace que los datos no puedan ser modificados sin alterar toda la cadena, lo cual es prácticamente imposible sin el consenso de la mayoría de los nodos. Gracias a este diseño, la blockchain se ha convertido en una solución fiable para guardar datos sin riesgo de manipulación o pérdida.

Las características principales de la BlockChain son:

La seguridad en blockchain está garantizada por el uso de **algoritmos criptográficos** que protegen la información de cada bloque. Cada transacción debe ser validada por la red antes de ser añadida, y una vez registrada, no puede ser modificada sin invalidar todos los bloques siguientes. Esto hace que los intentos de fraude sean fácilmente

detectables y rechazados. Además, la red completa actúa como sistema de verificación, lo que elimina los puntos únicos de fallo.

La descentralización, a diferencia de los sistemas tradicionales, en los que la información está gestionada por una autoridad central (como un banco o una empresa), la blockchain se apoya en miles de ordenadores distribuidos por todo el mundo. Cada nodo de la red tiene una copia del historial completo de la cadena, y las decisiones se toman por consenso. Esta **ausencia de control central** es lo que hace que el sistema sea más resistente a caídas, censuras o manipulaciones.

El pseudo anonimato en una red blockchain no es necesario usar nombres o datos personales. Cada usuario opera mediante una dirección digital (similar a un número de cuenta), lo que permite mantener su identidad fuera del registro público. Aunque todas las transacciones son visibles y transparentes, no están directamente asociadas a una persona, sino a una dirección encriptada. Por eso se habla de **seudonimato**, más que de anonimato completo.

La **transparencia**, todas las transacciones realizadas quedan registradas de forma pública y permanente en la cadena. Esto permite que cualquier persona pueda consultar los movimientos de la red en cualquier momento, lo que refuerza la confianza en el sistema.

#### 2.4.4. ¿Cómo funcionan las redes de criptomonedas?

Las criptomonedas como **Bitcoin**, **Ethereum** y muchas otras no solo existen como monedas digitales, sino que se basan en redes específicas diseñadas para gestionarlas de forma segura y descentralizada. Estas redes no son simples aplicaciones, sino ecosistemas completos basados en **tecnología blockchain**, donde cada nodo (ordenador conectado a la red) cumple una función clave para validar, registrar y mantener el sistema en funcionamiento.

Para poder hacer uso de estas redes de criptomonedas, tenemos que pagar una tarifa o **fee**. Esto sirve, por un lado, para incentivar a los mineros (personas o entidades que validan y registran las transacciones en la red, como veremos más adelante) y, por otro lado, para evitar el abuso del sistema. Si las transacciones fueran gratuitas, la red estaría casi siempre saturada por operaciones innecesarias. Además, estas tarifas permiten establecer un cierto nivel de prioridad: las transacciones con tarifas más altas suelen procesarse antes que las que ofrecen una comisión menor.

Esta tarifa, que se paga en la criptomoneda nativa de la red (por ejemplo, Bitcoin, Ethereum, Solana, etc.), puede tener diferentes destinos dependiendo del diseño de cada blockchain. Generalmente, una red blockchain puede optar por una de las siguientes opciones:

##### 1. La comisión se entrega íntegramente al validador o minero

Este es el caso más tradicional. La totalidad del **fee** que se paga por cada transacción va directamente al minero o validador como recompensa por incluirla en un bloque. Este es el modelo que usa, por ejemplo, **Bitcoin**, donde las comisiones no se eliminan ni se redistribuyen: se transfieren directamente al minero que haya resuelto el bloque.

## **2. La comisión se divide: una parte se quema y otra se entrega al validador**

Algunas redes han introducido un sistema en el que una parte de la tarifa se **quema** (es decir, se destruye o elimina del suministro total de monedas) y el resto se entrega como recompensa. Un ejemplo claro de esto es **Ethereum**,<sup>[32]</sup> que desde la actualización EIP-1559 implementó este sistema:

- **Base fee:** se quema automáticamente en cada transacción.
- **Propina o prioridad (tip):** va directamente al validador.

Este modelo permite no solo incentivar la participación en la red, sino también **reducir la inflación** de la criptomoneda, e incluso en algunos momentos hacerla deflacionaria, es decir, que disminuya su cantidad total en circulación.

## **3. La red implementa mecanismos de quema periódica o programada**

En redes como **Binance Smart Chain (BSC)**, no todas las tarifas se queman al instante. Sin embargo, Binance realiza **quemas periódicas** de su token BNB como parte de su modelo económico. Esto significa que, aunque no se quemen comisiones en cada transacción, la oferta total del token sí se reduce de forma programada, lo cual puede influir en su valor con el tiempo.

La quema de tokens tiene varios efectos positivos sobre la economía de una criptomoneda:

- **Reduce el suministro total**, lo que puede aumentar su escasez y, por tanto, su valor.
- **Ayuda a combatir la inflación**, especialmente en redes donde se emiten nuevas monedas regularmente.
- **Permite un modelo económico más equilibrado y sostenible a largo plazo.**

No todas las redes optan por esta estrategia. Algunas, como Bitcoin, apuestan por la simplicidad y transparencia de pagar a los mineros, mientras que otras, como Ethereum, buscan un equilibrio entre incentivo y control de la oferta.

A continuación, se describen las dos redes más representativas:

### **Red de Bitcoin**

La red de **Bitcoin** fue la primera en implementar el concepto de blockchain en 2009. Su principal objetivo es permitir transferencias de valor (bitcoins) entre usuarios, sin intermediarios. Funciona mediante un proceso conocido como **proof of work (PoW)**, en el que los nodos compiten por resolver complejos problemas matemáticos para validar un bloque de transacciones. A los nodos que resuelven estos problemas se les conoce como *mineros*, y reciben una recompensa en bitcoins por su trabajo.

Características principales:

- **Transacciones seguras y públicas**, registradas de forma inmutable.
- **Descentralización completa**: cualquier persona con un ordenador potente puede participar en la red.
- **Velocidad y escalabilidad limitadas**: suele procesar unas 7 transacciones por segundo.
- Es una red pensada exclusivamente para transacciones económicas simples.

### Red de Ethereum

Ethereum, lanzada en 2015, introdujo una evolución significativa al concepto original de blockchain. Además de permitir el envío de su propia criptomoneda (**Ether, ETH**), esta red permite crear **contratos inteligentes (smart contracts)** y aplicaciones descentralizadas (**dApps**) directamente sobre la blockchain.

A diferencia de Bitcoin, Ethereum ha ido evolucionando en su modelo de consenso. Originalmente usaba *proof of work*, pero desde 2022 funciona bajo un sistema más eficiente llamado **proof of stake (PoS)**, donde los validadores son seleccionados en función de la cantidad de Ethereum que tienen y "ponen en juego" como garantía.

Características destacadas:

- **Soporte para contratos inteligentes**, lo que permite crear apps financieras, juegos, NFTs, etc.
- **Red más flexible** y orientada a la programación.
- **Mayor velocidad de transacción**, aunque sigue teniendo costes asociados (conocidos como *gas*).
- Comunidad muy activa en desarrollo e innovación tecnológica.

### Otras redes destacadas

- **Binance Smart Chain (BSC)**: similar a Ethereum, pero con tarifas más bajas y alta compatibilidad.[33]
- **Solana**: conocida por su velocidad y bajo coste, aunque con menor descentralización.[34]
- **Polygon (MATIC)**: una red que mejora la escalabilidad de Ethereum funcionando como una capa secundaria.

#### **2.4.5. Mineros y validadores: el motor de las redes blockchain**

En cualquier red blockchain, el proceso de registrar y validar transacciones depende de actores clave: los **mineros** y los **validadores**. Aunque ambos cumplen un rol similar garantizar el buen funcionamiento de la red, lo hacen mediante mecanismos distintos, dependiendo del tipo de blockchain.

Los **mineros** son protagonistas en las redes que utilizan el sistema de *Proof of Work* (prueba de trabajo), como Bitcoin. Su labor consiste en resolver complejos problemas matemáticos utilizando equipos de alto rendimiento (generalmente ASICs o tarjetas gráficas potentes). El primero en resolver el problema y validar el bloque obtiene una **recompensa en criptomonedas**, además de las comisiones por transacción. Este sistema, aunque efectivo, requiere un consumo elevado de energía, lo que ha generado debates en torno a su sostenibilidad medioambiental.

Por otro lado, en las redes más recientes, muchas han migrado hacia un modelo conocido como *Proof of Stake* (prueba de participación), como es el caso de Ethereum desde 2022. En este sistema, los llamados **validadores** no minan, sino que **bloquean una cantidad de criptomonedas como garantía** para poder participar en la validación de bloques. Cuanto mayor sea la cantidad bloqueada, más probabilidades tienen de ser elegidos para validar transacciones y obtener recompensas.

El negocio de la minería y la validación de criptomonedas ha crecido de manera exponencial en los últimos años. Grandes empresas, conocidas como **mining farms**, han invertido millones en infraestructuras con miles de dispositivos trabajando en paralelo. Este crecimiento ha generado auténticas economías en torno a la minería, especialmente en países donde el coste energético es más bajo.

En el caso de Ethereum, la transición a *Proof of Stake* ha cambiado por completo el modelo. Ahora, cualquier usuario con una cantidad suficiente de ETH puede convertirse en validador, reduciendo así la barrera de entrada y el impacto ambiental. Este cambio ha democratizado en parte la participación, haciendo que el papel del validador esté al alcance de más personas, incluso mediante **pools de staking**, donde varios usuarios aportan sus fondos conjuntamente.

#### **2.4.6. ¿Qué es el halving y cómo afecta al valor de Bitcoin?**

Una de las características más singulares de Bitcoin es su política monetaria predefinida. A diferencia de las monedas tradicionales, que pueden ser emitidas en función de decisiones de bancos centrales, Bitcoin tiene un sistema automatizado y limitado: **solo existirán 21 millones de bitcoins**. Esta escasez está controlada por un mecanismo llamado **halving**, que juega un papel fundamental en la evolución del precio de esta criptomoneda.

## ¿En qué consiste el halving?

El *halving* es un evento programado que ocurre aproximadamente cada **210.000 bloques**, lo que equivale a unos **cuatro años**. En cada halving, la **recompensa que reciben los mineros por validar bloques se reduce a la mitad**.[35] Por ejemplo:

- En 2009, los mineros recibían 50 BTC por bloque.
- En 2012, se redujo a 25 BTC.
- En 2016, a 12.5 BTC.
- En 2020, a 6.25 BTC.
- En 2024,a 3.125 BTC.

Esta reducción progresiva limita la emisión de nuevos bitcoins, lo que refuerza su escasez a medida que se acerca al máximo de 21 millones.

## ¿Cómo afecta esto al valor de Bitcoin?

Desde un punto de vista económico, el halving tiene implicaciones importantes en la oferta y la demanda:

- **Disminución de la oferta nueva:** Al reducirse a la mitad la cantidad de nuevos bitcoins que entran en circulación, la presión de venta por parte de los mineros disminuye.
- **Mayor escasez:** Si la demanda se mantiene o crece, pero la oferta baja, el precio tiende a subir. Así ha ocurrido en ciclos anteriores: tras cada halving, Bitcoin ha experimentado subidas significativas meses después.
- **Efecto psicológico:** El halving también genera expectativas en los inversores y suele marcar el inicio de nuevos ciclos alcistas, lo cual contribuye a su revalorización en el mercado.

## Bitcoin como valor refugio

Además de su diseño deflacionario, Bitcoin destaca por ser **finito**, es decir, llegará un momento (alrededor del año 2140) en el que **ya no se podrá emitir ni un solo bitcoin más**. Esta propiedad lo convierte en un activo diferente a cualquier moneda fiduciaria, cuya emisión puede aumentar indefinidamente.

Por esta razón, cada vez más inversores e instituciones empiezan a considerar Bitcoin como una especie de "**valor refugio**", comparable al oro. En contextos de inflación, inestabilidad financiera o pérdida de valor de las monedas tradicionales, Bitcoin representa una reserva de valor que **no puede ser devaluada ni manipulada** por ningún banco central o gobierno.

#### **2.4.7. Formas de conseguir o comprar criptomonedas**

A medida que las criptomonedas han ido ganando protagonismo en el panorama financiero, también se han diversificado las formas en las que los usuarios pueden acceder a ellas. Actualmente, existen múltiples maneras de conseguir criptomonedas, que se pueden clasificar en dos grandes grupos: **mediante adquisición directa (compra)** y **mediante obtención activa o pasiva (ganancia o recompensa)**.

##### **1. Compra en plataformas de intercambio (exchanges)**

La forma más común y accesible de conseguir criptomonedas es a través de **plataformas de intercambio** o **exchanges**, como:

- **Binance**
- **Coinbase**
- **Kraken**
- **Bit2Me** (muy utilizada en España)

Estas plataformas permiten comprar criptomonedas con moneda fiat (euros, dólares, etc.) utilizando **tarjeta de crédito/débito, transferencia bancaria o incluso PayPal** en algunos casos. A cambio, se cobra una pequeña comisión por transacción. SueLEN ofrecer también monederos digitales integrados y funciones de trading más avanzadas.

##### **2. Cajeros automáticos de criptomonedas**

Aunque menos comunes, existen **cajeros físicos de criptomonedas** repartidos en algunas ciudades del mundo, incluida España. En estos cajeros, los usuarios pueden introducir dinero en efectivo y recibir criptomonedas en su cartera digital. También permiten el proceso inverso: vender criptomonedas y retirar euros. Aunque cómodos, suelen tener **comisiones más elevadas** que las plataformas online.

##### **3. Minería (*mining*)**

Otra manera de conseguir criptomonedas es mediante el proceso de **minería**, especialmente en redes como Bitcoin. Esto implica utilizar dispositivos informáticos para resolver cálculos complejos y validar transacciones. A cambio, el minero recibe **una recompensa en criptomonedas**. Sin embargo, hoy en día este método requiere **una inversión significativa en hardware** y electricidad, por lo que ha quedado principalmente en manos de empresas o grandes grupos organizados (*mining pools*).

##### **4. Participación en redes Proof of Stake (*staking*)**

En blockchainS que utilizan el mecanismo de *Proof of Stake*, como Ethereum, es posible obtener criptomonedas simplemente por **bloquear una cantidad determinada de fondos** en la red. Esto se conoce como *staking* y funciona como una especie de depósito que genera intereses. Es una opción más accesible que la minería y no requiere hardware especializado.[36]

## 5. Airdrops y recompensas

Algunos proyectos de criptomonedas, especialmente nuevos lanzamientos, ofrecen **recompensas gratuitas o airdrops** a usuarios que cumplan ciertas condiciones: registrarse en su plataforma, seguirlos en redes sociales, probar la aplicación, etc. Si bien los montos suelen ser bajos, es una forma interesante de iniciarse sin necesidad de invertir dinero.

## 6. Compra entre particulares (*peer-to-peer*)

También es posible adquirir criptomonedas mediante **intercambios directos entre personas**. Existen plataformas como **LocalBitcoins** o **Paxful** que facilitan este tipo de operaciones, actuando como intermediarios de confianza. Aunque este método ofrece más privacidad, también **requiere precauciones adicionales** para evitar estafas.

### 2.4.8. Almacenamiento de criptomonedas: tipos de carteras digitales (*wallets*)

A diferencia del dinero tradicional, las criptomonedas no se almacenan en cuentas bancarias físicas, sino en carteras digitales conocidas como *wallets*. Estas carteras no contienen directamente los activos, sino las claves privadas que permiten acceder a ellos en la red blockchain. La gestión adecuada de estas claves es crucial, ya que son el único medio para disponer de los fondos asociados. Por tanto, la elección del tipo de cartera influye directamente en la seguridad y accesibilidad de las criptomonedas.

En función de su conexión a Internet, se pueden distinguir dos grandes grupos: *hot wallets* y *cold wallets*. Las hot wallets, o carteras calientes, están permanentemente conectadas a la red, lo que las hace ideales para realizar transacciones frecuentes. Ejemplos comunes son los monederos proporcionados por exchanges como Binance o Coinbase, así como aplicaciones móviles o de escritorio como MetaMask, Trust Wallet o Exodus. Su principal ventaja es la accesibilidad, ya que suelen ser gratuitas y fáciles de usar. No obstante, su naturaleza conectada las expone a mayores riesgos de ciberataques o accesos no autorizados si el dispositivo o la conexión no están debidamente protegidos.

Por otro lado, las cold wallets, o carteras frías, no están conectadas a Internet, lo que proporciona un mayor nivel de seguridad. Son especialmente recomendables para almacenar grandes cantidades de criptomonedas a largo plazo. Entre ellas destacan las *hardware wallets*, dispositivos físicos similares a una memoria USB, como Ledger Nano o Trezor, y las *paper wallets*, que consisten en las claves privadas impresas o anotadas manualmente. Estas soluciones ofrecen una protección superior frente a amenazas digitales, aunque resultan menos prácticas para el uso diario y conllevan riesgos físicos, como la pérdida o el deterioro del soporte donde se almacenan.

Otra clasificación relevante distingue entre *wallets custodiales* y *no custodiales*, en función de quién posee las claves privadas. Las wallets custodiales delegan esa responsabilidad en un tercero, habitualmente un exchange o plataforma centralizada. Este modelo facilita la recuperación de acceso en caso de pérdida, pero reduce el control del usuario sobre sus propios fondos. En cambio, las wallets no custodiales otorgan al usuario el control absoluto sobre sus claves, lo que garantiza una mayor autonomía pero implica también una mayor responsabilidad en términos de seguridad y gestión.

En conclusión, el almacenamiento de criptomonedas no es una cuestión menor, ya que la seguridad de los activos depende directamente de cómo y dónde se guardan las claves privadas. Elegir el tipo de wallet adecuado dependerá del perfil del usuario, la frecuencia de uso, el volumen de activos y el nivel de experiencia con tecnologías de seguridad. Un uso combinado de carteras calientes para operaciones cotidianas y frías para almacenamiento seguro a largo plazo suele ser la opción más equilibrada para muchos inversores.

# CAPÍTULO 3: Análisis de requisitos

Este capítulo está dedicado al análisis y definición de los requisitos que debe cumplir el sistema, tanto funcionales como no funcionales, para garantizar su correcto funcionamiento y una buena experiencia de usuario. También se describen los distintos actores que interactuarán con la plataforma, así como los posibles escenarios de uso. Finalmente, se incluye un diagrama de casos de uso que permite visualizar de forma clara las principales funcionalidades del sistema y las relaciones entre los distintos elementos implicados.

## 3.1. Análisis de requisitos

El primer paso para desarrollar un proyecto de software consiste en definir claramente los **requisitos funcionales y no funcionales**. Estos requisitos representan las condiciones que debe cumplir el producto final para ser considerado válido, y se determinan a partir de los **objetivos del proyecto y la interacción con el cliente o usuario final**.

En este caso, se ha optado por una **metodología ágil**, basada en **iteraciones incrementales**, lo que permite una revisión continua de los requisitos a lo largo del desarrollo. Esto significa que dichos requisitos no son estáticos, sino que pueden **ajustarse y evolucionar** en función de las necesidades que vayan surgiendo en cada fase del proceso.

### 3.1.1. Requisitos funcionales

Los requisitos funcionales definen las operaciones que debe realizar el sistema y los comportamientos esperados según las funcionalidades implementadas. A continuación, se detallan los principales requisitos funcionales del proyecto:

#### 1. Gestión de usuarios

**RF-1.** Registrar nuevos usuarios con correo electrónico y contraseña.

**RF-2.** Iniciar sesión en el sistema con credenciales válidas.

**RF-3.** Visualizar los datos personales del usuario autenticado.

**RF-4.** Editar los datos del usuario, incluyendo nombre y contraseña.

**RF-5.** Salir de la cuenta de usuario.

**RF-6.** Obtener información del usuario por ID o correo electrónico.

#### 2. Gestión de carteras (portfolios)

**RF-7.** Crear nuevas carteras de inversión personalizadas.

**RF-8.** Consultar todas las carteras asociadas a un usuario.

**RF-9.** Editar los datos de una cartera.

**RF-10.** Eliminar una cartera existente.

**RF-11.** Obtener carteras por ID de usuario o por correo electrónico.

### **3. Gestión de transacciones**

**RF-12.** Registrar nuevas transacciones en una cartera.

**RF-13.** Consultar una transacción específica por ID.

**RF-14.** Listar todas las transacciones asociadas a una cartera.

**RF-15.** Modificar una transacción existente.

**RF-16.** Eliminar una transacción.

### **4. Consulta de precios en tiempo real**

**RF-17.** Consultar el precio en tiempo real de acciones mediante la API de Alpha Vantage.

**RF-18.** Consultar el precio en tiempo real de criptomonedas.

**RF-19.** Consultar el precio en tiempo real de los ETFs.

### **5. Interacción con el modelo de lenguaje**

**RF-20.** Enviar preguntas al modelo de lenguaje y obtener respuestas automatizadas.

#### **3.1.2. Requisitos no funcionales**

Los requisitos no funcionales describen las características técnicas, operativas o de calidad que debe cumplir el sistema:

##### **1. Seguridad**

**RNF-1.** Cifrar las contraseñas de los usuarios en la base de datos.

**RNF-2.** Utilizar el protocolo HTTPS para asegurar las comunicaciones.

**RNF-3.** Proteger los endpoints mediante autenticación y roles de usuario.

##### **2. Usabilidad**

**RNF-4.** Diseñar una interfaz clara, accesible y fácil de usar.

**RNF-5.** Garantizar que la interfaz sea responsive y funcione en distintos dispositivos.

##### **3. Rendimiento**

**RNF-6.** Asegurar tiempos de respuesta rápidos en las peticiones ( $\leq 1$ s en condiciones normales).

**RNF-7.** Soportar múltiples usuarios simultáneos sin pérdida de rendimiento.

#### **4. Escalabilidad**

**RNF-8.** Permitir la escalabilidad del sistema para futuras integraciones (brokers, bancos, otros activo)

#### **5. Mantenibilidad**

**RNF-9.** Mantener un código modular, estructurado y documentado.

#### **6. Disponibilidad**

**RNF-10.** Garantizar una alta disponibilidad del sistema (>99 % del tiempo).

### **3.2. Casos de uso**

Un caso de uso es una descripción de los pasos o actividades que deben realizarse para llevar a cabo un proceso concreto dentro del sistema. Estos casos de uso permiten modelar el comportamiento funcional del sistema desde el punto de vista del usuario y constituyen una parte esencial en el análisis del software.

Las entidades que interactúan con el sistema en un caso de uso se denominan **actores**.

#### **3.2.1. Descripción de actores**

A continuación se describen los actores que se pueden encontrar en el sistema.

##### **Ac-1. Usuario no registrado**

- **Descripción:** Persona que accede a la plataforma sin haber iniciado sesión. Tiene acceso limitado a las funcionalidades de la aplicación.
- **Características:** Puede registrarse, iniciar sesión y consultar información básica previa al acceso completo.
- **Relaciones:** Se transforma en “Usuario registrado” tras completar el proceso de autenticación (registro/login).
- **Atributos:** No posee identificador único dentro de la aplicación hasta que se registre.
- **Comentarios:** Este actor sirve como punto de entrada al sistema. No tiene acceso a datos financieros ni funciones de gestión de carteras o transacciones.

##### **Ac-2. Usuario registrado**

- **Descripción:** Persona que ha creado una cuenta en la aplicación y ha iniciado sesión correctamente. Es el principal usuario del sistema.
- **Características:** Puede crear y gestionar portafolios, realizar operaciones (altas, modificaciones y borrado de transacciones), consultar datos de mercado

en tiempo real e interactuar con el modelo de lenguaje.

- **Relaciones:** Puede tener múltiples portafolios, cada uno con diversas transacciones asociadas.
- **Atributos:** Identificador único, nombre de usuario, correo electrónico, contraseña cifrada, datos de configuración personal.
- **Comentarios:** Es el actor que aprovecha completamente las funcionalidades ofrecidas por la plataforma. Todo caso de uso principal depende de él.

### AC-3. Administrador

- **Descripción:** Persona encargada de la gestión interna del sistema y la supervisión general de la plataforma.
- **Características:** Tiene permisos ampliados respecto al usuario estándar. Puede consultar información de usuarios mediante su correo electrónico, realizar tareas de mantenimiento y auditoría del sistema, y comprobar el correcto funcionamiento de los servicios.
- **Relaciones:** Puede interactuar con los módulos de usuarios, carteras, transacciones y APIs de consulta de datos.
- **Atributos:** Acceso completo al backend y a herramientas administrativas.
- **Comentarios:** Este rol está limitado a perfiles con autorización especial y no forma parte de la experiencia de usuario común. Se asume un nivel técnico avanzado.

#### 3.2.2. Descripción casos de uso

En esta sección se describen detalladamente los casos de uso de la aplicación desarrollada. Se han identificado un total de 20 casos de uso que cubren las funcionalidades principales y complementarias del sistema. Cada caso de uso ha sido documentado siguiendo una estructura que consta de los siguientes campos:

- **Actores:** Define quién o qué interactúa con el sistema para llevar a cabo el caso de uso. Puede tratarse de un usuario, un administrador o incluso un sistema externo.
- **Tipo:** Indica la naturaleza del caso de uso, clasificándose como primario, esencial, secundario, opcional, entre otros.
- **Referencias:** Hace alusión al endpoint, pantalla o módulo asociado al caso de uso dentro de la aplicación.
- **Precondición:** Establece las condiciones que deben cumplirse antes de que el caso de uso pueda ejecutarse.

- **Postcondición:** Describe el estado del sistema una vez finalizado el caso de uso de forma satisfactoria.
- **Autor:** Persona responsable de la definición o documentación del caso de uso.
- **Versión:** Versión del caso de uso, útil para llevar un control de cambios a lo largo del desarrollo.
- **Propósito:** Resume brevemente la intención principal del caso de uso.
- **Resumen:** Describe de manera general qué ocurre durante la ejecución del caso de uso.
- **Curso normal:** Detalla paso a paso el flujo típico de interacción entre el actor y el sistema para lograr el objetivo del caso de uso. Cada paso se desglosa indicando claramente la acción realizada por el actor y la respuesta del sistema.

### **CU-1. Registrar un nuevo usuario**

**Actores:** Usuario no registrado.

**Tipo:** Primario, esencial.

**Referencias:** Endpoint [POST user/register](#)

**Precondición:** El usuario no debe tener una cuenta previamente registrada.

**Postcondición:** El usuario queda registrado y puede iniciar sesión.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir a nuevos usuarios crear una cuenta.

**Resumen:** El usuario accede al formulario de registro, introduce sus datos y, si son válidos, el sistema lo guarda y lo redirige al login.

#### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Pulsa el botón “Registrarse”.	
2		Muestra el formulario de registro.
3	Usuario: Rellena los campos requeridos y pulse “Registrar”.	
4		Validar los datos introducidos.
5		Si todo es correcto, crea el usuario y lo guarda en la base de datos.
6		Muestra mensaje de éxito y redirige al login.

Tabla 1.8 CU-1. Registrar un nuevo usuario

## **CU-2. Iniciar sesión en el sistema**

**Actores:** Usuario registrado.

**Tipo:** Primario, esencial.

**Referencias:** Endpoint `POST user/login?email=h@gmail.co&password=1234`.

**Precondición:** El usuario debe estar previamente registrado.

**Postcondición:** El sistema otorga acceso al usuario.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir que un usuario acceda a la aplicación.

**Resumen:** El usuario introduce sus credenciales, el sistema las válida y, si son correctas, le da acceso al sistema.

### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Pulsa “Iniciar sesión”.	
2		Muestra el formulario de login.
3	Usuario: Introduce email y contraseña, pulsa “Login”.	
4		Valida credenciales.
5		Si son correctas, otorga token de sesión y redirige al dashboard.
6		Muestra mensaje de bienvenida o error si las credenciales son inválidas.

Tabla 1.9 CU-2. Iniciar sesión en el sistema

## **CU-3. Salir de la cuenta del usuario**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Icon Log Out

**Precondición:** El usuario debe estar autenticado en el sistema.

**Postcondición:** El usuario queda desconectado y la sesión cerrada.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir que el usuario cierre sesión de forma segura.

**Resumen:** El usuario selecciona la opción de cerrar sesión y el sistema termina la sesión activa.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa en el icono “Cerrar sesión”.	
2		El sistema termina la sesión del usuario.
3		Muestra confirmación de cierre de sesión.
4	Usuario: Es redirigido a la pantalla de login o inicio.	

Tabla 1.10 CU-3. Salir de la cuenta del usuario

## CU-4. Editar perfil del usuario

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoints `PATCH user/updatePassword/:id`

`PATCH user/updateName/:id`

**Precondición:** El usuario debe haber iniciado sesión.

**Postcondición:** Los cambios en el perfil son guardados.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario modificar sus datos personales.

**Resumen:** El usuario puede cambiar su nombre o contraseña desde su perfil. Los cambios se actualizan al confirmar.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa en “Editar perfil”.	
2		Muestra el formulario con los datos actuales.
3	Usuario: Modifica los campos y pulsa “Guardar”.	
4		Valida la información modificada.
5		Actualiza los datos en la base de datos.
6		Muestra mensaje de éxito y recarga datos actualizados.

Tabla 1.11 CU-4. Editar perfil del usuario

## CU-5. Eliminar cuenta de usuario

**Actores:** Administrador

**Tipo:** Primario.

**Referencias:** Endpoint `DELETE user/:id`

**Precondición:** El usuario debe tener cuenta y tener privilegios especiales (admin/dev).

**Postcondición:** La cuenta del usuario es eliminada de forma permanente

**Autor:** Miguel Bravo Campos

**Versión:** 1.0

**Propósito:** Permitir al usuario eliminar su cuenta definitivamente del sistema.

**Resumen:** Desde la configuración, el administrador puede realizar la eliminación de su cuenta. El sistema solicita confirmación y, si se aprueba, borra la cuenta.

### Curso normal

Paso	Actor	Sistema
1	Administrador: Pulsa en la opción “Eliminar cuenta” desde la configuración.	
2		Muestra mensaje de advertencia solicitando confirmación.
3	Administrador: Confirma la eliminación de la cuenta.	
4		Elimina al usuario
5		Muestra un mensaje de éxito.

Tabla 1.12 CU-5. Eliminar cuenta de usuario

## CU-6. Ver datos del perfil

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /user/:id`.

**Precondición:** El usuario debe haber iniciado sesión.

**Postcondición:** Se muestran los datos actuales del perfil.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario visualizar sus datos personales almacenados.

**Resumen:** El usuario accede a su perfil, y el sistema muestra los datos actuales (nombre, correo, etc.) sin permitir modificación.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa en “Mi perfil”.	
2		Muestra los datos del perfil del usuario.
3	Usuario: Revisa la información mostrada.	

Tabla 1.13 CU-6. Ver datos del perfil

## CU-7. Consultar usuario por correo electrónico

**Actores:** Administrador / desarrollador.

**Tipo:** Secundario.

**Referencias:** Endpoint `GET /user/:email`.

**Precondición:** Tener privilegios especiales (admin/dev).

**Postcondición:** Se muestran los datos del usuario asociado al correo.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Consultar la existencia o información de un usuario concreto.

**Resumen:** Introduciendo un correo válido, el sistema busca y devuelve los datos del usuario correspondiente.

## Curso normal

Paso	Actor	Sistema
1	Admin/dev: Accede al buscador.	
2		Muestra campo para introducir correo electrónico.
3	Admin/dev: Introduce correo.	
4		Busca y muestra los datos del usuario asociado.

Tabla 1.14 CU-7. Consultar usuario por correo electrónico

## CU-8. Crear una nueva cartera

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `POST /portfolio`.

**Precondición:** El usuario debe haber iniciado sesión.

**Postcondición:** La cartera se almacena asociada al usuario.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario gestionar carteras de inversión.

**Resumen:** El usuario introduce los datos de la nueva cartera (nombre) y el sistema la guarda en su cuenta.

### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Pulsa “Crear nueva cartera”.	
2		Muestra formulario para crear cartera.
3	Usuario: Rellena los campos y confirma.	
4		Valida datos y guarda cartera en base de datos.
5		Muestra confirmación y lista actualizada de carteras.

Tabla 1.15 CU-8. Crear una nueva cartera

### **CU-9. Ver todas las carteras del usuario**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /portfolio/user/:id`.

**Precondición:** El usuario debe estar autenticado.

**Postcondición:** Se muestra el listado de carteras del usuario.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario visualizar todas sus carteras.

**Resumen:** Una vez autenticado, el usuario accede a la sección de carteras, donde se muestran todas las que ha creado.

### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Pulsa en “MyPortfolio”.	
2		Recupera de la base de datos todas sus carteras.
3		Muestra la lista con nombre, tipo y resumen.

Tabla 1.16 CU-9. Ver todas las carteras del usuario

## **CU-10. Ver los detalles de una cartera específica**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /portfolio/:id`.

**Precondición:** El usuario debe haber iniciado sesión y tener acceso a la cartera.

**Postcondición:** Se muestran los datos completos de la cartera seleccionada.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Ver el contenido detallado de una cartera concreta.

**Resumen:** Al seleccionar una cartera de la lista, el sistema muestra todos sus datos (activos, balance, transacciones, etc.).

### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Pulsa sobre una cartera de la lista.	
2		Recupera todos los datos asociados a la cartera por ID.
3		Muestra información detallada: nombre, tipo, activos, etc.

Tabla 1.17 CU-10. Ver los detalles de una cartera específica

## **CU-11. Editar los datos de una cartera**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `PATCH /portfolio/:id`.

**Precondición:** El usuario debe haber iniciado sesión y ser propietario de la cartera.

**Postcondición:** Se actualizan el nombre de la cartera en el sistema.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir modificar el nombre de una cartera.

**Resumen:** El usuario accede al detalle de una cartera, edita los campos permitidos (nombre) y guarda los cambios.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa “Editar” en una cartera.	
2		Muestra el formulario con los datos actuales.
3	Usuario: Modifica campos y pulsa “Guardar”.	
4		Valida y actualiza la información en la base de datos.
5		Muestra mensaje de confirmación.

Tabla 1.18 CU-11. Editar los datos de una cartera

## CU-12. Eliminar una cartera

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `DELETE /portfolio/:id`.

**Precondición:** El usuario debe estar autenticado y ser dueño de la cartera.

**Postcondición:** La cartera es eliminada del sistema.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario eliminar una de sus carteras.

**Resumen:** El usuario selecciona una cartera y la elimina de forma permanente.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa en “Eliminar” en una cartera.	
2		Solicita confirmación de la eliminación.
3	Usuario: Confirma la eliminación.	
4		Elimina la cartera.
5		Muestra mensaje de confirmación.

Tabla 1.19 CU-12. Eliminar una cartera

### **CU-13. Consultar carteras por ID de usuario o por email**

**Actores:** Administrador / desarrollador.

**Tipo:** Secundario.

**Referencias:** Endpoint `GET /portfolio/user/:id` o `GET /portfolio/email/:email`.

**Precondición:** Tener permisos administrativos o de desarrollo.

**Postcondición:** Se obtiene una lista de carteras asociadas al usuario indicado.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Ver las carteras de un usuario específico desde un rol con privilegios.

**Resumen:** El actor introduce un identificador (ID o email) y el sistema devuelve las carteras correspondientes.

#### **Curso normal**

Paso	Actor	Sistema
1	Admin/dev: Accede al buscador de carteras.	
2		Muestra campos para introducir ID o email.
3	Admin/dev: Introduce el dato.	
4		Consulta en base de datos y muestra resultados.

Tabla 1.20 CU-13. Consultar carteras por ID de usuario o por email

### **CU-14. Añadir una nueva transacción**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `POST /transaction`.

**Precondición:** El usuario debe estar autenticado y tener una cartera activa.

**Postcondición:** La transacción queda asociada a la cartera seleccionada.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Añadir compras o ventas de activos dentro de una cartera.

**Resumen:** El usuario completa el formulario con los datos de la transacción (tipo, cantidad, precio, activo) y esta se almacena.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Entra en una cartera y pulsa “Añadir transacción”.	
2		Muestra el formulario con los campos necesarios.
3	Usuario: Rellena el formulario y pulsa “Guardar”.	
4		Valida y guarda la transacción en base de datos.
5		Muestra la transacción en el historial.

Tabla 1.21 CU-14. Añadir una nueva transacción

## CU-15. Ver una transacción por ID

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /transaction/:id`.

**Precondición:** El usuario debe haber iniciado sesión y ser propietario de la transacción.

**Postcondición:** Se muestran los datos completos de la transacción.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Consultar la información específica de una operación.

**Resumen:** Al seleccionar una transacción concreta, el sistema muestra sus datos: tipo de activo, fecha, precio, cantidad, etc.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Accede a la sección de transacciones.	
2	Usuario: Selecciona una transacción por ID.	
3		Muestra los datos completos de esa transacción.

Tabla 1.22 CU-15. Ver una transacción por ID

## CU-16. Eliminar una transacción

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `DELETE /transaction/:id`.

**Precondición:** El usuario debe estar autenticado y ser propietario de la transacción.

**Postcondición:** La transacción es eliminada de la cartera.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir que el usuario elimine una transacción errónea o ya no deseada.

**Resumen:** El usuario selecciona una transacción y la elimina de forma permanente.

### Curso normal

Paso	Actor	Sistema
1	Usuario: Pulsa “Eliminar” en una transacción.	
2		Solicita confirmación para eliminar.
3	Usuario: Confirma la eliminación.	
4		Elimina la transacción de la base de datos.
5		Muestra mensaje de confirmación.

Tabla 1.23 CU-16. Eliminar una transacción

## CU-17. Consultar el precio actual de una acción

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /stock/:assets`.

**Precondición:** Usuario autenticado.

**Postcondición:** Se muestra el precio actualizado de la acción consultada.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario obtener el precio actual de una acción.

**Resumen:** El usuario introduce el símbolo de la acción y el sistema devuelve su precio en tiempo real.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Introduce símbolo de acción y solicita precio.	
2		Consulta la API Alpha Vantage y obtiene el precio.
3		Muestra el precio actualizado al usuario.

Tabla 1.24 CU-17. Consultar el precio actual de una acción

## CU-18. Consultar el precio actual de una criptomoneda

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /crypto/:cryptomoneda/eur`.

**Precondición:** Usuario autenticado.

**Postcondición:** Se muestra el precio actualizado de la criptomoneda consultada.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario obtener el precio actual de una criptomoneda.

**Resumen:** El usuario introduce el símbolo o nombre de la criptomoneda y el sistema devuelve su precio en tiempo real.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Introduce símbolo/nombre de criptomoneda y solicita precio.	
2		Consulta la API Alpha Vantage y obtiene el precio.
3		Muestra el precio actualizado al usuario.

Tabla 1.25 CU-18. Consultar el precio actual de una criptomoneda

## **CU-19. Consultar el precio actual de un ETF**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `GET /etf/:ETF`.

**Precondición:** Usuario autenticado.

**Postcondición:** Se muestra el precio actualizado del ETF consultado.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario obtener el precio actual de un ETF.

**Resumen:** El usuario introduce el símbolo del ETF y el sistema devuelve su precio en tiempo real.

### **Curso normal**

Paso	Actor	Sistema
1	Usuario: Introduce el símbolo del ETF y solicita precio.	
2		Consulta la API Alpha Vantage y obtiene el precio.
3		Muestra el precio actualizado al usuario.

Tabla 1.26 CU-19. Consultar el precio actual de un ETF

## **CU-20. Realizar una consulta al modelo de lenguaje**

**Actores:** Usuario registrado.

**Tipo:** Primario.

**Referencias:** Endpoint `POST /preguntar`.

**Precondición:** Usuario autenticado.

**Postcondición:** Se entrega una respuesta generada basada en la pregunta del usuario.

**Autor:** Miguel Bravo Campos

**Versión:** 1.0.

**Propósito:** Permitir al usuario realizar preguntas tipo chatbot para resolver dudas financieras.

**Resumen:** El usuario escribe una consulta y el sistema responde con información generada por el modelo de lenguaje.

## Curso normal

Paso	Actor	Sistema
1	Usuario: Introduce una pregunta en el chat.	
2		Envía la pregunta al modelo de lenguaje.
3		Recibe respuesta y la muestra al usuario.

Tabla 1.27 CU-20. Realizar una consulta al modelo de lenguaje

### 3.3 Diagrama de Casos de Uso

El diagrama de casos de uso representa de forma visual las distintas interacciones que los usuarios pueden realizar con el sistema **MyPortfolio**. En él se muestran los diferentes actores (usuarios registrados y no registrados) y los casos de uso asociados, es decir, las funcionalidades principales que ofrece la plataforma.

Este diagrama permite entender de manera clara y estructurada el comportamiento general del sistema desde el punto de vista del usuario, facilitando así el análisis y diseño del software. Gracias a esta representación, es posible identificar los requisitos funcionales del sistema y garantizar que todas las necesidades del usuario estén cubiertas.

La figura 1.4 es un diagrama de casos de uso, representa de forma global las funcionalidades principales del sistema desde la perspectiva de los diferentes actores que interactúan con él: usuario no registrado, usuario registrado y administrador. Cada actor tiene acceso a un conjunto específico de funcionalidades, siendo el usuario registrado quien dispone de la mayor parte de las operaciones relacionadas con la gestión de carteras, transacciones y consultas de mercado. Por su parte, el administrador cuenta con herramientas para la gestión de usuarios y carteras. Este esquema permite visualizar de manera clara los distintos escenarios de uso contemplados en la aplicación, facilitando así la comprensión general de su funcionamiento.

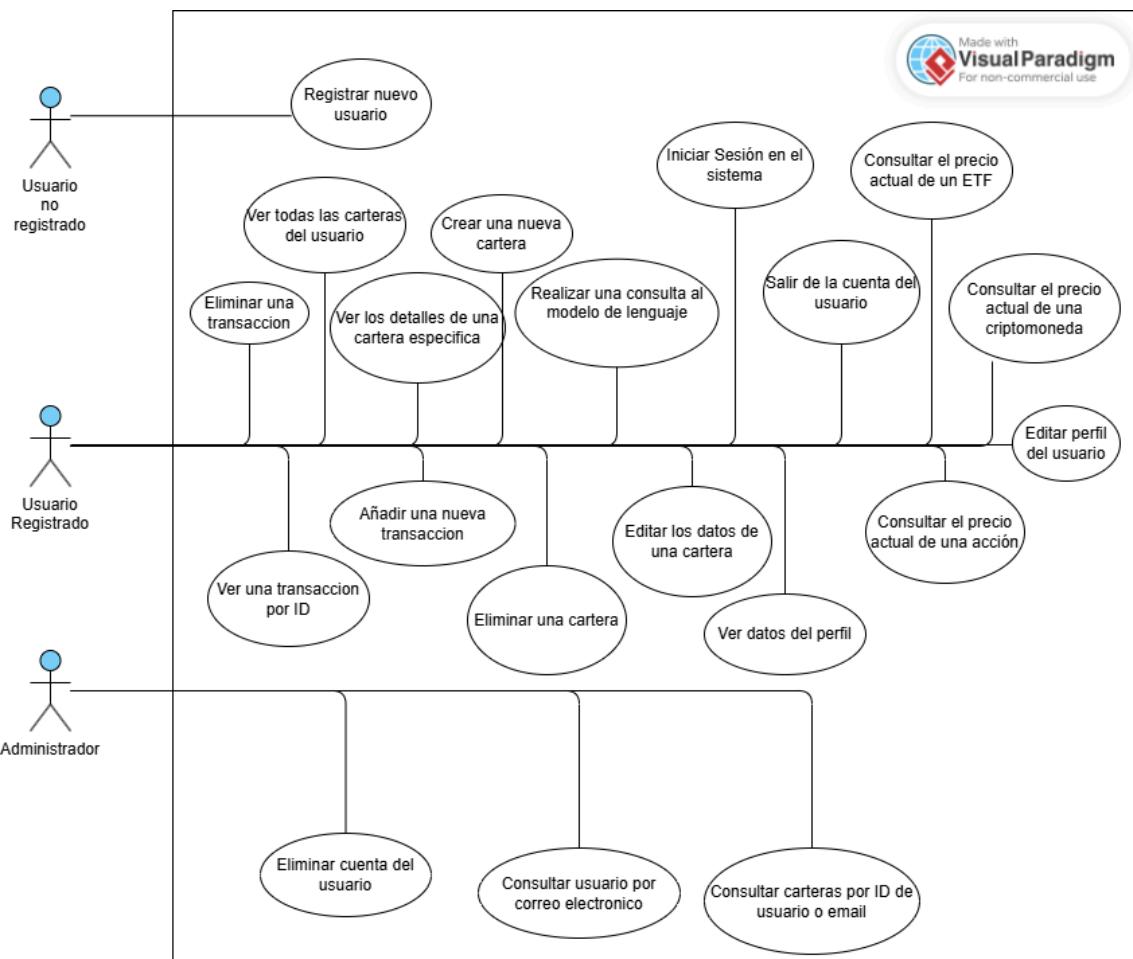


Figura 1.4 Diagrama de casos de uso

# CAPÍTULO 4: Diseño e implementación

En este capítulo se detalla el diseño e implementación técnica del proyecto, comenzando por la arquitectura del sistema basada en microservicios y las tecnologías seleccionadas para el desarrollo del frontend, backend y la base de datos. Se profundiza en el diseño del modelo de datos, la organización de carpetas y los endpoints. También se explica la implementación de la interfaz de usuario y los distintos componentes que la conforman, así como el desarrollo del backend, la conexión con la base de datos, la lógica de negocio y la estructura de controladores y servicios. Finalmente, se describe la integración con APIs externas y el modelo de lenguaje, las pruebas realizadas, el proceso de despliegue y los principales retos técnicos encontrados durante el desarrollo.

## 4.1 Arquitectura del sistema

Para el desarrollo de la plataforma **MyPortfolio**, se ha optado por una **arquitectura de microservicios** debido a las múltiples ventajas que esta ofrece, especialmente en términos de escalabilidad, mantenimiento y evolución del sistema. Esta decisión permite estructurar la aplicación como un conjunto de servicios independientes, cada uno encargado de una funcionalidad específica, que se comunican entre sí a través de interfaces bien definidas, generalmente mediante peticiones HTTP.

La arquitectura de microservicios destaca por su **desacoplamiento funcional**, lo que permite desarrollar, probar, desplegar y escalar cada módulo del sistema de manera independiente. Esta propiedad resulta especialmente útil en sistemas que, como el presente proyecto, podrían experimentar una evolución funcional progresiva o un crecimiento en número de usuarios.

En el caso concreto de MyPortfolio, se han definido distintos microservicios que responden a las principales áreas funcionales del sistema: **gestión de usuarios**, **gestión de carteras**, **transacciones**, **consulta de datos financieros en tiempo real** (a través de una API externa) y **procesamiento de lenguaje natural** para responder a preguntas de los usuarios. Estos servicios están organizados de forma modular, lo que facilita tanto su desarrollo como su mantenimiento a largo plazo.

Cada microservicio gestiona su lógica de negocio de forma autónoma, manteniendo la cohesión interna de sus operaciones. La comunicación entre los distintos módulos se lleva a cabo mediante un esquema de peticiones REST, en formato JSON, y gestionado por controladores para cada módulo que actúan como **API Gateway**, encargándose de enrutar las solicitudes del cliente hacia el microservicio correspondiente.

En conjunto, esta organización del sistema no solo ha permitido una implementación más clara y mantenible, sino que también sienta las bases para futuras ampliaciones del proyecto, como la integración con brokers externos o la incorporación de funcionalidades automatizadas en el seguimiento de inversiones.

La figura 1.5 que se presenta corresponde a la estructura de entidades del sistema, una parte fundamental del backend en el desarrollo de esta aplicación. Esta capa representa el modelo de datos que define la forma en la que la información se almacenará en la base de datos, así como su relación con el dominio del problema.

En este caso, se han definido tres entidades principales: User, Portfolio y Transaction. Cada una de estas clases ha sido diseñada siguiendo los principios de la programación orientada a objetos y empleando las anotaciones propias del framework Spring Boot, junto con JPA (Java Persistence API) para facilitar la persistencia de datos en una base de datos relacional.

- **User:** Esta entidad representa a los usuarios registrados en el sistema. Contiene campos básicos como identificador, nombre de usuario, correo electrónico y contraseña, entre otros. Además, mantiene una relación uno a muchos con la entidad Portfolio, ya que un mismo usuario puede gestionar múltiples carteras.
- **Portfolio:** Esta clase modela una cartera de inversión. Incluye atributos como un identificador único, nombre de la cartera y su fecha de creación. A su vez, se establece una relación uno a muchos con la entidad Transaction, ya que cada cartera puede contener múltiples transacciones.
- **Transaction:** Representa cada una de las operaciones realizadas por el usuario dentro de una cartera. Contiene información relevante como el activo implicado, la cantidad, el precio, la fecha de la operación, entre otros. Esta clase también incluye un atributo de tipo enumerado denominado TransactionType.
- **TransactionType:** Es una enumeración que permite distinguir el tipo de operación realizada, clasificándose como COMPRA o VENTA. Este enfoque proporciona una mayor claridad semántica y evita errores comunes al manejar cadenas de texto u otros valores menos controlados.

La utilización de clases entidad permite mapear directamente los objetos Java a tablas en la base de datos mediante ORM (Object-Relational Mapping), lo cual facilita las operaciones CRUD (crear, leer, actualizar y eliminar). Esta organización mejora la mantenibilidad del código, reduce la complejidad del acceso a datos y proporciona una estructura sólida sobre la cual construir el resto de la lógica del sistema.

Esta primera parte de la arquitectura del backend sienta las bases para el tratamiento eficiente y estructurado de la información que maneja la aplicación, garantizando integridad, coherencia y escalabilidad.

# Entities

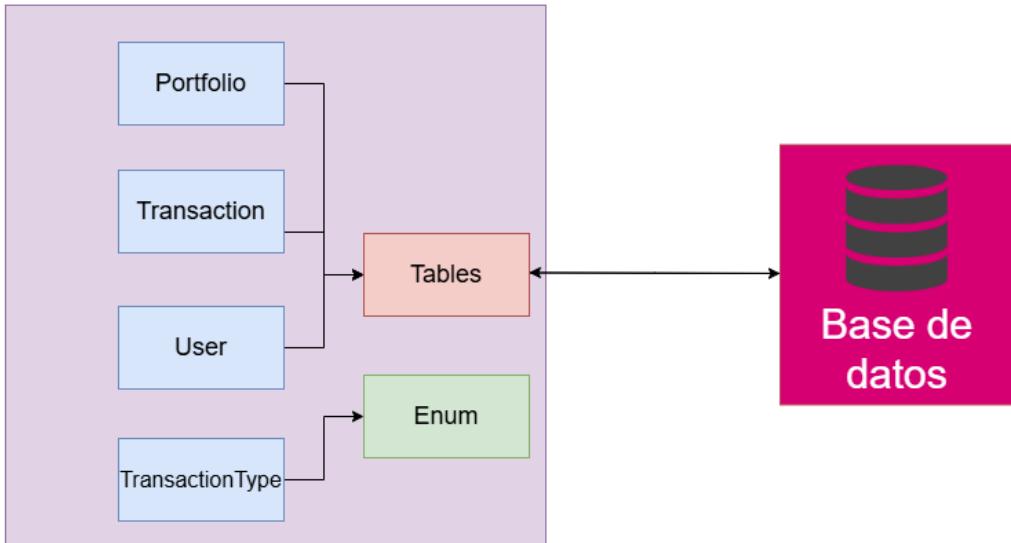


Figura 1.5 Diagrama de funcionamiento de Entities en el Back End

La figura 1.6 hace referencia a la capa de repositorio del backend, una parte clave dentro del patrón de diseño arquitectónico Modelo-Vista-Controlador (MVC) utilizado en esta aplicación. Esta capa actúa como intermediaria entre las entidades y la lógica de negocio, proporcionando una abstracción para las operaciones de acceso a la base de datos.

En este proyecto se han definido tres interfaces principales dentro del paquete de repositorios: UserRepository, PortfolioRepository y TransactionRepository. Cada una de estas interfaces extiende de JpaRepository, que forma parte del módulo Spring Data JPA de Spring Boot. Esta extensión permite heredar de manera automática una gran variedad de métodos para operaciones CRUD (Create, Read, Update, Delete), eliminando así la necesidad de implementarlos manualmente.

- **UserRepository:** Encargado de gestionar el acceso a los datos de los usuarios. A través de esta interfaz es posible, por ejemplo, buscar usuarios por nombre, correo electrónico o ID, así como guardar nuevos registros o eliminar los existentes.
- **PortfolioRepository:** Proporciona las funciones necesarias para interactuar con los datos de las carteras. Permite recuperar todas las carteras de un usuario concreto, acceder a una cartera por su identificador o crear nuevas carteras de inversión.
- **TransactionRepository:** Se encarga de las operaciones relacionadas con las transacciones dentro de las carteras. Esta interfaz permite consultar todas las transacciones asociadas a una cartera específica, filtrarlas por tipo (compra o venta) o realizar nuevas inserciones en la base de datos.

Gracias a la integración con JpaRepository, se logra una simplificación notable del código y una mejora significativa en la productividad del desarrollo, ya que Spring Boot genera automáticamente las implementaciones necesarias en tiempo de ejecución.

# Repositories

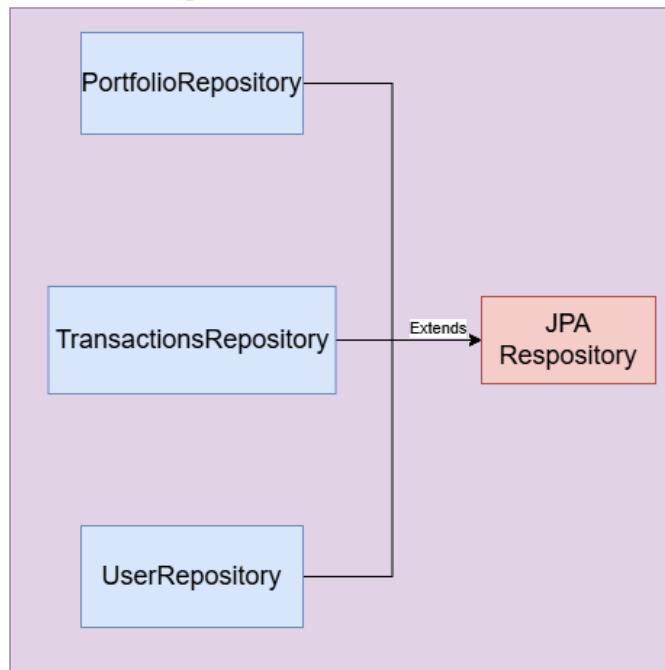


Figura 1.6 Diagrama de funcionamiento de Repositories en el Back End

La figura 1.7 se centra en la capa de servicios del backend, más concretamente en la definición de las interfaces UserService, PortfolioService y TransactionService. Esta capa representa un componente fundamental dentro de la arquitectura de la aplicación, ya que actúa como intermediaria entre los controladores (que gestionan las peticiones HTTP) y los repositorios (que acceden a la base de datos). Su principal objetivo es encapsular la lógica de negocio, asegurando una separación clara de responsabilidades y una mayor mantenibilidad del sistema.

Cada una de las interfaces define un conjunto de métodos que representan las operaciones clave que pueden realizarse sobre cada tipo de entidad del sistema:

- **UserService:** Define métodos relacionados con la gestión de usuarios, como el registro, autenticación, recuperación de información del perfil o actualización de datos. Estos métodos serán invocados desde los controladores encargados de gestionar las acciones del usuario final.
- **PortfolioService:** Contiene las operaciones específicas sobre las carteras de inversión, como la creación de nuevas carteras, la obtención de carteras por usuario, el cálculo de métricas de rendimiento o la eliminación de carteras existentes.

- **TransactionService:** Declara los métodos necesarios para gestionar las transacciones financieras dentro de las carteras, permitiendo registrar compras y ventas, consultar el historial de operaciones, o calcular estadísticas relacionadas con el comportamiento de los activos.

Estas interfaces permiten desacoplar la definición del servicio de su implementación concreta. Esto favorece la reutilización, facilita la inyección de dependencias a través de anotaciones como `@Autowired`, y promueve el principio de programación orientada a interfaces.

En fases posteriores del desarrollo, estas interfaces serán implementadas por clases que definen la lógica específica de cada operación, utilizando los métodos proporcionados por los repositorios.

## Services Interfaces

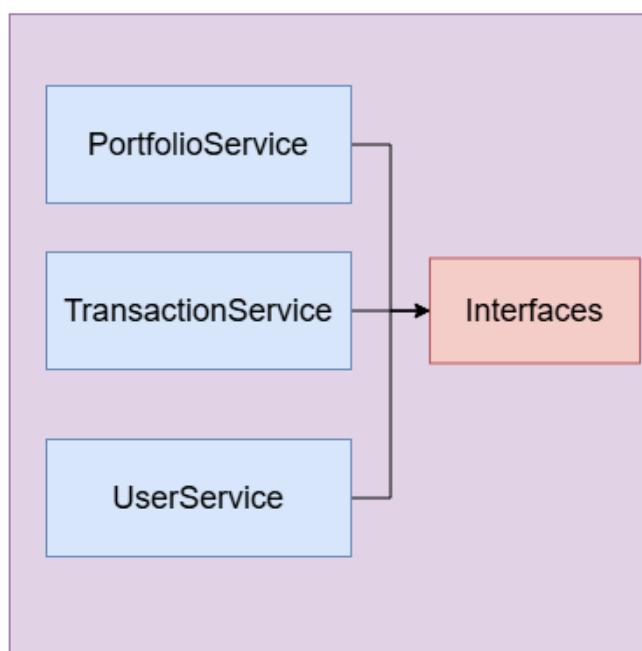


Figura 1.7 Diagrama de funcionamiento de Services Interfaces en el Back End

La figura 1.8 representa la capa de implementación de los servicios del backend. En ella se encuentran las clases `UserServiceImpl`, `PortfolioServiceImpl` y `TransactionServiceImpl`, encargadas de implementar la lógica de negocio definida previamente en las interfaces correspondientes: `UserService`, `PortfolioService` y `TransactionService`.

Cada clase de implementación está anotada con `@Service`, lo que permite que el framework de Spring las detecte como componentes de servicio y gestione su ciclo de vida. Además, en estas clases se realiza la **inyección de dependencias** mediante `@Autowired` o a través del constructor, incorporando así las instancias de los repositorios correspondientes (`UserRepository`, `PortfolioRepository`, `TransactionRepository`). Esto permite a los servicios interactuar directamente con la base de datos utilizando los métodos proporcionados por JPA Repository, como `save()`, `findById()`, `findAll()`, `deleteById()`, entre otros.

Estas clases constituyen el núcleo funcional de la aplicación. Por ejemplo:

- `UserServiceImpl` gestiona el registro de usuarios, autenticaciones, actualizaciones de perfil, entre otros.
- `PortfolioServiceImpl` permite crear, obtener o eliminar carteras de inversión asociadas a un usuario.
- `TransactionServiceImpl` maneja la lógica de las operaciones de compra y venta de activos, incluyendo validaciones específicas, persistencia y cálculos auxiliares.

Además de estas implementaciones estándar, se han desarrollado dos servicios adicionales que encapsulan la interacción con APIs externas:

- **OllamaService**: Este servicio encapsula la lógica necesaria para interactuar con una API externa basada en modelos de lenguaje (LLM). Su función principal es enviar peticiones al modelo alojado en Ollama para generar textos, respuestas o recomendaciones dentro de la aplicación, añadiendo así una capa de inteligencia artificial al sistema. Esto permite, por ejemplo, ofrecer asistencia automatizada.[37]
- **StockPriceService**: Este servicio se encarga de integrar la API de **Alpha Vantage**, un proveedor externo de datos financieros. Gracias a este componente, la aplicación es capaz de consultar precios actualizados de acciones en tiempo real o casi real, obtener históricos, y complementar así las funcionalidades del sistema con información precisa del mercado bursátil.[38]

Ambos servicios externos siguen el mismo principio de encapsulamiento que el resto de componentes: aislar la lógica de integración con terceros en clases específicas, lo que permite una mejor mantenibilidad, testeo y posibilidad de sustitución de proveedores en el futuro.

En conjunto, esta capa garantiza que toda la lógica de negocio, tanto interna como basada en servicios externos, esté correctamente organizada, modularizada y lista para ser utilizada por los controladores REST que componen la capa superior de la arquitectura.

# Services Implementation

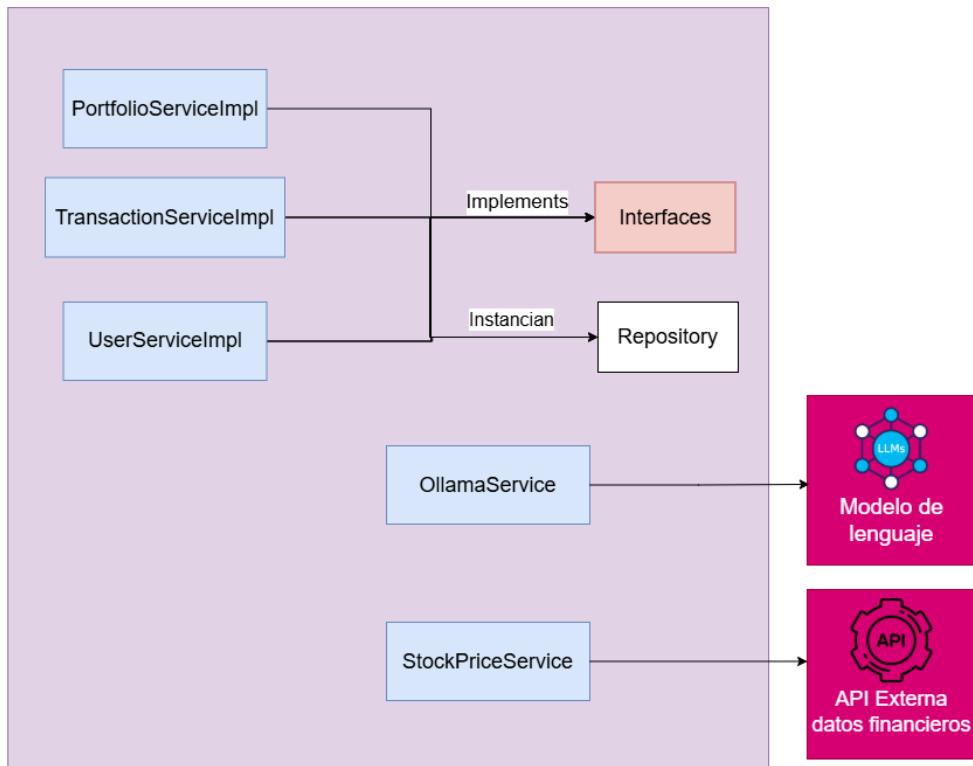


Figura 1.8 Diagrama de funcionamiento de Services Implementation en el Back End

La figura 1.9 representa la capa de controladores del sistema, responsables de exponer los servicios del backend a través de una interfaz RESTful. Esta capa constituye el punto de entrada de todas las peticiones HTTP provenientes del cliente (frontend, scripts o cualquier consumidor de la API). En el sistema desarrollado, se han definido cinco controladores principales: UserController, PortfolioController, TransactionController, OllamaController y StockPriceController.

Cada uno de estos controladores está anotado con `@RestController`, lo que permite a Spring Boot gestionarlos como controladores REST y serializar automáticamente las respuestas en formato JSON. Además, se utilizan anotaciones como `@RequestMapping`, `@GetMapping`, `@PostMapping`, `@PutMapping` o `@DeleteMapping` para definir claramente las rutas y métodos HTTP asociados a cada operación.

Los controladores actúan como intermediarios entre el cliente y la lógica de negocio. Cada uno contiene una instancia del servicio correspondiente (por ejemplo, `PortfolioServiceImpl` en `PortfolioController`), que es inyectada mediante el mecanismo de **inyección de dependencias** de Spring (`@Autowired` o constructor injection). Esto garantiza una separación clara entre la lógica de presentación (controlador) y la lógica de negocio (servicio), siguiendo el principio de responsabilidad única y fomentando una arquitectura limpia y mantenible.

## Controladores implementados:

- **UserController:** Gestiona endpoints relacionados con el registro, autenticación y gestión de usuarios. Ofrece funcionalidades como crear una nueva cuenta,

iniciar sesión y recuperar datos del perfil.

- **PortfolioController:** Permite al usuario crear, consultar y eliminar sus carteras de inversión. También ofrece funcionalidades adicionales para listar todos los portfolios de un usuario concreto.
- **TransactionController:** Expone endpoints para añadir, listar o eliminar transacciones (compras y ventas de activos financieros). Se encarga de gestionar las operaciones que afectan al contenido y rendimiento de los portfolios.
- **OllamaController:** Actúa como interfaz entre la aplicación y el modelo de lenguaje externo alojado en Ollama. A través de este controlador, se pueden enviar peticiones al modelo para obtener respuestas generadas por inteligencia artificial, por ejemplo, para obtener recomendaciones financieras o explicaciones sobre movimientos del mercado.
- **StockPriceController:** Ofrece acceso a los datos de precios bursátiles obtenidos mediante la API de Alpha Vantage. A través de este controlador, el usuario puede consultar el valor actual de un activo o recuperar su historial de precios.

## Controllers

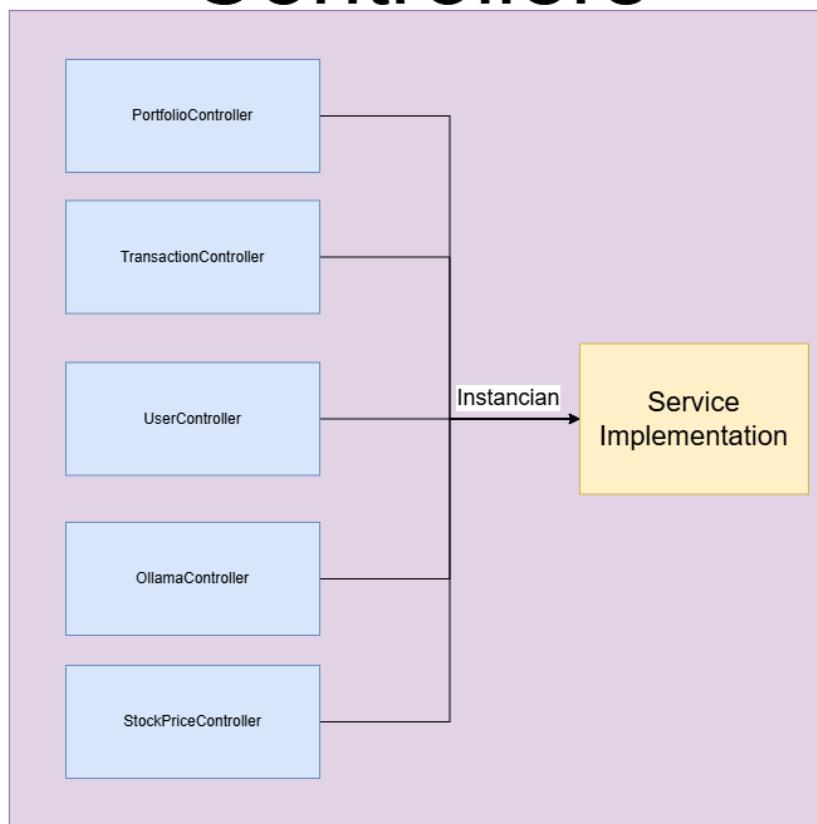


Figura 1.9 Diagrama de funcionamiento de Controllers en el Back End

La figura 1.10 ofrece una visión global de la arquitectura del backend del sistema desarrollado. En ella se visualizan claramente todos los módulos que conforman esta capa: entidades (entities), repositorios (repositories), interfaces de servicio (services), implementaciones de servicio (services implementation) y controladores (controllers).

Cada uno de estos componentes ha sido descrito detalladamente en secciones anteriores, y juntos constituyen la estructura modular del backend basada en el paradigma de arquitectura en capas, siguiendo las buenas prácticas del desarrollo con Spring Boot.

Tal y como se observa, el backend se conecta con el frontend a través de los controladores. Estos actúan como punto de entrada a la lógica de negocio, exponiendo una API REST mediante la cual el cliente puede interactuar con el sistema, ya sea para gestionar usuarios, carteras, transacciones, consultar precios bursátiles o comunicarse con el modelo de lenguaje externo.

Esta organización no solo favorece la mantenibilidad y escalabilidad de la aplicación, sino que también asegura una separación de responsabilidades clara y un flujo de datos controlado entre las diferentes capas del sistema.

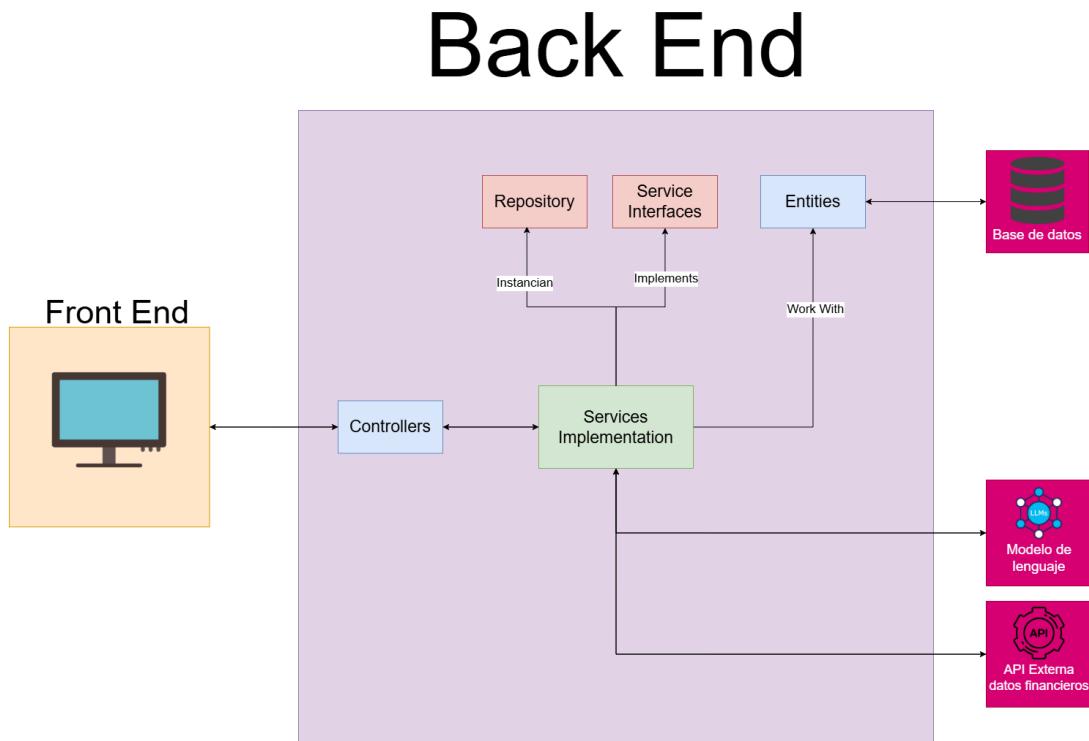


Figura 1.10 Diagrama de funcionamiento del Back End

La figura 1.11 muestra la arquitectura general del frontend de la aplicación, desarrollado utilizando React. En la parte izquierda del esquema se encuentra el usuario, quien interactúa directamente con la interfaz a través de un navegador. Esta interacción se canaliza mediante el componente principal App, que actúa como punto de entrada a toda la aplicación cliente y compone la estructura principal de navegación del sitio.

El componente App se conecta con varios módulos fundamentales:

- **Components:** contiene todos los componentes reutilizables que pueden ser utilizados en distintas partes de la aplicación, lo que favorece la modularidad y la reutilización del código.
- **Pages:** agrupa todas las páginas individuales de la aplicación. Cada una representa una vista distinta (como la página de inicio, login, dashboard, etc.).

El componente **Pages** se conecta a su vez con los siguientes módulos:

- **Hooks:** directorio en el que se encuentran los *custom hooks* desarrollados específicamente para la lógica reutilizable entre componentes o páginas. Permiten encapsular funcionalidades comunes y mejorar la organización del código.
- **Context:** contiene el archivo que proporciona el contexto global de la aplicación. A través de este, se gestiona el estado compartido entre componentes (como el estado de autenticación del usuario), facilitando la comunicación eficiente entre las distintas partes del frontend.
- **Styles:** incluye las hojas de estilo CSS utilizadas para dar formato y diseño a todos los elementos visuales de la interfaz, garantizando una experiencia de usuario coherente y estéticamente cuidada.
- **Services:** módulo responsable de realizar las peticiones HTTP al backend. En él se encapsulan las llamadas a las distintas API del servidor, centralizando la lógica de comunicación entre cliente y servidor. Este módulo representa el punto de conexión entre el frontend y el backend.

Esta organización del frontend permite una separación clara de responsabilidades, fomentando la escalabilidad, el mantenimiento y la legibilidad del código. Gracias a esta estructura modular, cada parte de la interfaz puede evolucionar de forma independiente, manteniendo al mismo tiempo una cohesión total con el sistema.

# Front End

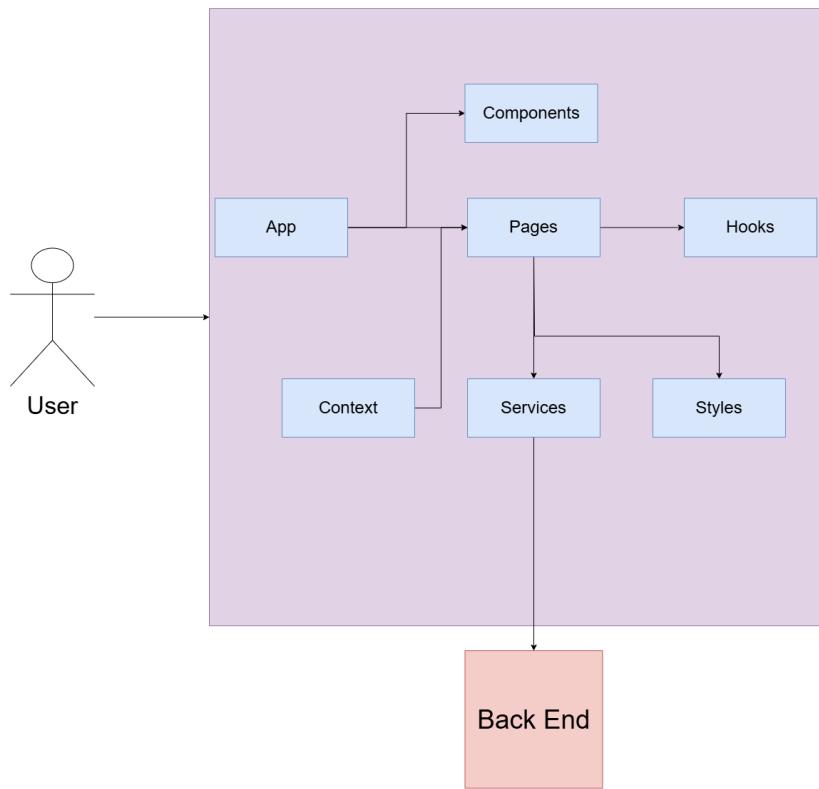


Figura 1.11 Diagrama de funcionamiento del Front End

La figura 1.12 proporciona una visión global de todo el sistema, mostrando la interacción entre sus distintos componentes. En primer lugar, se encuentra el **usuario**, que accede a la aplicación a través del **frontend**, desarrollado con React. Este frontend actúa como interfaz gráfica y punto de entrada a las funcionalidades del sistema.

El frontend se comunica directamente con el **backend**, construido con Spring Boot, mediante llamadas HTTP a los controladores (**controllers**), encargados de recibir las peticiones del cliente, procesarlas y enviar la respuesta correspondiente. A su vez, el backend gestiona la lógica de negocio a través de sus servicios internos y accede a los datos persistentes almacenados en la **base de datos**.

Además del acceso a la base de datos, el backend interactúa con dos **APIs externas**. Por un lado, se comunica con **Alpha Vantage**, una API que proporciona información actualizada sobre precios bursátiles, esencial para ofrecer datos de mercado en tiempo real. Por otro lado, también se integra con una API de un **modelo de lenguaje (LLM)**, utilizada para ofrecer respuestas generadas automáticamente mediante inteligencia artificial.

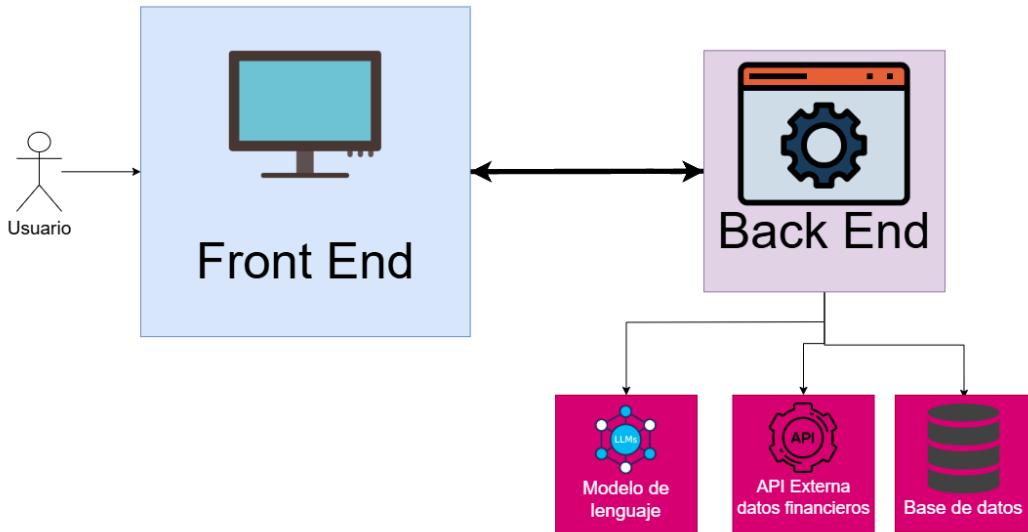


Figura 1.12 Diagrama de funcionamiento del Sistema

## 4.2. Tecnologías utilizadas

En esta sección se describen con detalle las tecnologías utilizadas para el desarrollo de la aplicación, estructurando la explicación en tecnologías que conforman el back end, la base de datos, el front end y las herramientas utilizadas.

### 4.2.1 Back End

Para el desarrollo del sistema, se valoraron distintas opciones de lenguajes de programación adecuados para la implementación del backend. Finalmente, se optó por **Java** como lenguaje principal para esta parte del proyecto.

La elección de Java se fundamenta en varios factores clave. En primer lugar, se trata de un lenguaje maduro, ampliamente utilizado en entornos profesionales y con una comunidad de desarrolladores muy activa, lo que garantiza un amplio soporte y documentación. Además, su tipado fuerte y estático permite detectar errores en tiempo de compilación, lo que contribuye a mejorar la fiabilidad y el mantenimiento del sistema a largo plazo.

Otra de las razones que motivaron su selección fue su buen rendimiento en ejecución y su capacidad para construir aplicaciones robustas y escalables, lo cual es especialmente relevante en un sistema que debe gestionar múltiples entidades (usuarios, carteras, transacciones) y mantenerse estable en escenarios de crecimiento. Asimismo, Java ofrece una estructura clara y modular, lo que facilita la organización del código y el trabajo en equipo en proyectos de cierta complejidad.

En resumen, Java ha sido elegido como lenguaje para el desarrollo del backend por su robustez, rendimiento, fiabilidad y adecuación a sistemas con requisitos de escalabilidad y mantenimiento.

Como medio para construir el backend se ha optado por utilizar **Spring Boot**, un framework que permite construir aplicaciones web robustas de manera eficiente. Su arquitectura modular, junto con una amplia comunidad y documentación, lo convierte en una opción sólida para proyectos de producción.

Spring Boot facilita la creación de APIs RESTful, la integración con bases de datos y la gestión de dependencias mediante su sistema de configuración simplificado. Además, permite un desarrollo rápido y estructurado gracias a sus convenciones predefinidas, lo que ha sido clave para garantizar la escalabilidad y el mantenimiento del sistema a lo

largo del tiempo.

#### 4.2.2 Base de datos

Para el almacenamiento de información, se ha optado por el uso de una base de datos relacional basada en **SQL**.

La elección de una base de datos SQL responde a la necesidad de estructurar los datos de forma clara y coherente, siguiendo un modelo relacional, en el que existen entidades bien definidas como usuarios, carteras o transacciones, y relaciones entre ellas. El uso de SQL permite aplicar integridad referencial mediante claves foráneas, así como definir restricciones que ayudan a garantizar la consistencia de los datos.

Además, este tipo de bases de datos cuenta con un lenguaje de consultas poderoso, estandarizado y ampliamente conocido, lo que facilita tanto la implementación como el mantenimiento y evolución futura del sistema. También proporciona mecanismos fiables de seguridad, respaldo y recuperación ante fallos.

#### 4.2.3 Front End

Por otro lado, al tratarse de un proyecto con una carga de trabajo predecible y centrado en operaciones CRUD (crear, leer, actualizar y eliminar), una base de datos SQL resulta idónea por su eficiencia en este tipo de escenarios y por su compatibilidad con herramientas y frameworks ampliamente utilizados en el desarrollo backend.

Para el desarrollo del frontend del sistema se ha optado por utilizar **React** como biblioteca principal, una de las herramientas más populares y consolidadas en el desarrollo de interfaces de usuario modernas. React permite construir aplicaciones web dinámicas, escalables y mantenibles de forma eficiente, gracias a su enfoque basado en componentes reutilizables y su modelo de renderizado reactivo.

En cuanto a los lenguajes empleados, se han utilizado los estándares habituales en entornos web:

- **HTML** se ha usado para definir la estructura básica de los distintos componentes de la aplicación, sirviendo como base sobre la que se organiza el contenido visible para el usuario.
- **CSS** ha sido el lenguaje encargado de definir el estilo visual y la disposición de los elementos en pantalla, contribuyendo a mejorar la experiencia de usuario y la estética general de la plataforma.
- **JavaScript**, por su parte, ha tenido un papel central, ya que es el lenguaje sobre el que se construye y ejecuta React. Gracias a su versatilidad y dinamismo, ha permitido implementar la lógica de interacción, gestionar el estado de la aplicación y realizar llamadas a la API de forma eficiente y reactiva.

La combinación de React con estos lenguajes ha facilitado el desarrollo de una interfaz moderna, modular y altamente interactiva, alineada con las expectativas actuales de los usuarios en cuanto a usabilidad y rendimiento.

#### 4.2.4 Herramientas utilizadas

En esta sección se describen las distintas herramientas empleadas a lo largo del desarrollo del proyecto. Se incluyen tanto entornos de desarrollo, sistemas de control de versiones y plataformas de gestión de proyectos, como otras utilidades que han facilitado la prueba, testeo y documentación del proyecto.

## **Postman**

Postman ha sido una de las herramientas fundamentales durante el desarrollo del proyecto. Su uso ha estado orientado principalmente a la prueba y verificación de los endpoints del backend, permitiendo comprobar que las peticiones HTTP (GET, POST, PUT, DELETE, etc.) funcionaban correctamente y que las respuestas del servidor eran las esperadas.[39]

Gracias a su interfaz intuitiva y a sus funcionalidades avanzadas (como la gestión de entornos, colecciones de peticiones o test automáticos), Postman ha facilitado considerablemente el proceso de depuración, validación de lógica de negocio y control de errores en la API, siendo una herramienta clave para asegurar la calidad del backend.

## **GitHub**

GitHub ha sido utilizado como herramienta de control de versiones y para mantener el código fuente del proyecto organizado y accesible. Su uso ha permitido llevar un seguimiento ordenado de los avances en el desarrollo, facilitando la gestión del proyecto y asegurando que todas las modificaciones realizadas quedan registradas de forma clara y estructurada. Además, ha servido como repositorio remoto, garantizando la disponibilidad y seguridad del código durante todo el proceso de desarrollo.

## **Visual Studio**

Visual Studio ha sido la herramienta principal utilizada para la edición y desarrollo del código de todo el front end fuente durante el proyecto. Este entorno de desarrollo integrado (IDE) ofrece múltiples funcionalidades que facilitan la escritura, depuración y mantenimiento del código, contribuyendo a un desarrollo más eficiente y organizado. Me ha facilitado mucho el trabajo con todas las extensiones relativas a React y los diferentes lenguajes de front end.

## **IntelliJ IDEA**

IntelliJ IDEA ha sido el entorno de desarrollo integrado (IDE) seleccionado para el desarrollo del backend con Spring Boot. Esta herramienta proporciona potentes funcionalidades específicas para Java, facilitando la escritura, depuración y gestión del código. Su integración con frameworks como Spring Boot permite un desarrollo ágil y eficiente, mejorando la productividad y asegurando la calidad del software a lo largo del proyecto.[40]

## **Vite**

Vite ha sido la herramienta escogida para el desarrollo del frontend en React debido a su rapidez y eficiencia. Este entorno de construcción moderno permite una carga instantánea del proyecto durante el desarrollo gracias a su servidor local optimizado y a la utilización de módulos ES(módulos de JavaScript según el estándar ECMAScript). Además, Vite facilita una configuración sencilla y tiempos de compilación reducidos en comparación con herramientas más tradicionales, lo que ha mejorado notablemente la productividad durante el desarrollo de la interfaz de usuario.[41]

## **NPM (Node Package Manager)**

NPM ha sido una herramienta esencial en el desarrollo del proyecto, utilizada principalmente para la gestión de dependencias y la ejecución de scripts. Gracias a comandos como npm run dev, ha sido posible lanzar el entorno de desarrollo local de forma rápida, permitiendo probar los cambios en tiempo real. Además, ha facilitado la integración de librerías necesarias para el correcto funcionamiento del front-end desarrollado con React y Vite.[42]

## **Alpha Vantage**

Alpha Vantage es una API externa que hemos utilizado para obtener datos financieros en tiempo real, como precios de acciones, criptomonedas y ETFs. Esta herramienta resulta fundamental para proporcionar a los usuarios información actualizada y precisa sobre el mercado, lo que permite un monitoreo efectivo de sus inversiones dentro de la plataforma. Su integración facilita la consulta directa desde el backend, mejorando la experiencia del usuario al evitar la necesidad de acceder a múltiples fuentes externas.

### **4.3. Diseño del sistema**

Esta sección está dedicada al diseño técnico del proyecto, incluyendo la definición del modelo de datos, la estructura de carpetas y ficheros empleada, así como la descripción de los endpoints desarrollados.

#### **4.3.1 Diseño del modelo de datos**

La base de datos se ha diseñado teniendo en cuenta la organización clara y eficiente de cada entidad principal que conforma el sistema, asegurando que cada una contenga únicamente los datos necesarios para su funcionamiento. En este sentido, las entidades fundamentales son:

- **Usuario (users):** contiene los campos id, email, nombre y password. El campo id es la clave primaria de la tabla, con un valor autoincremental que garantiza la unicidad de cada usuario.
- **Portafolio (portfolios):** incluye id, user\_id y nombre. El campo id es la clave primaria y autoincremental, mientras que user\_id funciona como clave foránea que establece la relación con la tabla users, vinculando cada portafolio a un usuario específico.
- **Transacciones (transactions):** comprende los campos id, cantidad, fecha, precio, asset\_id y tipo. El campo id es la clave primaria con autoincremento. El campo tipo es un enumerado (enum) que puede tomar los valores COMPRA o VENTA, indicando la naturaleza de la transacción. Además, portfolio\_id actúa como clave foránea que relaciona cada transacción con un portafolio específico.

Las relaciones entre tablas se establecen mediante claves foráneas, garantizando la integridad referencial y facilitando la consulta eficiente de datos relacionados. Por ejemplo, cada portafolio está asociado a un usuario mediante user\_id, y cada transacción está vinculada a un portafolio mediante portfolio\_id.

Este diseño sencillo y funcional proporciona una base sólida para la gestión de los datos, con la posibilidad de ser ampliado conforme el proyecto crezca y se incorporen nuevas funcionalidades o entidades.

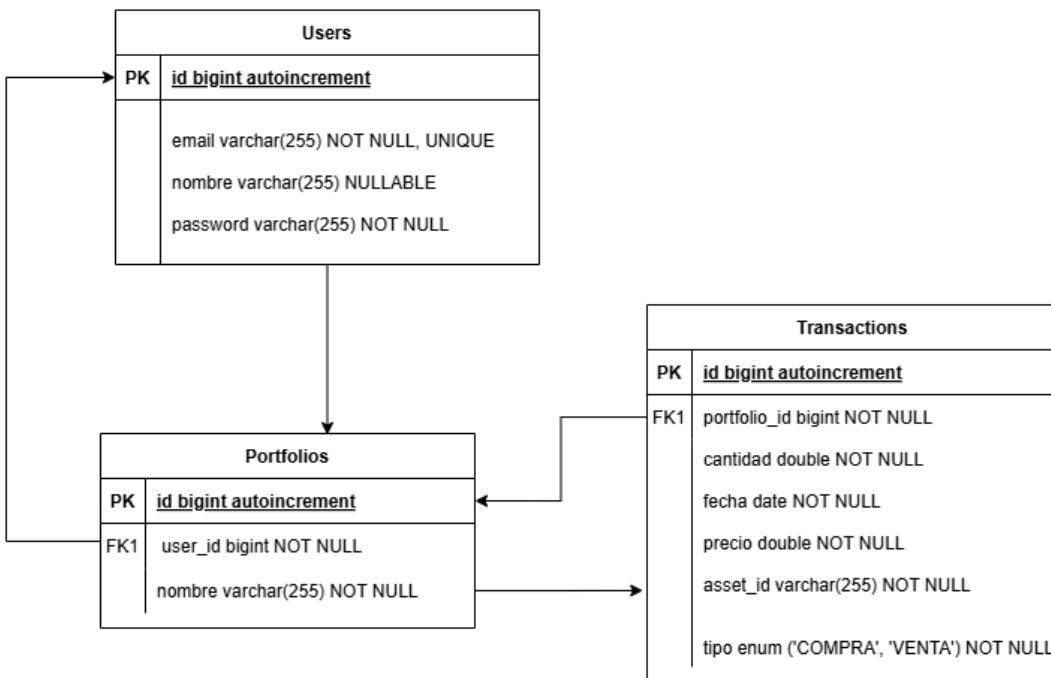


Figura 1.13 Diseño del modelo de datos

#### 4.3.2 Estructura de carpetas y ficheros del proyecto

En esta sección se describen como se han estructurado las carpetas y ficheros en las diferentes partes del proyecto, tanto el Back End como el Front End.

##### Back End

El backend del proyecto ha sido desarrollado utilizando Spring Boot y está organizado de forma modular para facilitar la escalabilidad y el mantenimiento. La estructura principal se encuentra dentro de la carpeta src, que contiene dos subdirectorios: main y test.

Dentro de main se encuentra la carpeta java, que incluye el paquete principal del proyecto y se organiza en diferentes carpetas según la responsabilidad de cada componente:

- MyPortfolioApplication.java: clase principal que arranca la aplicación.
- AppContext: contiene la configuración general del contexto de la aplicación.
- Config: clases de configuración del sistema, como seguridad o parámetros globales.
- Controllers: alberga los controladores que gestionan las peticiones HTTP.
- Entities: incluye las clases que representan las entidades de la base de datos (usuarios, carteras, transacciones...).
- Exceptions: define las excepciones personalizadas y su manejo.

- **Repositories:** contiene las interfaces de acceso a datos que extienden de JpaRepository.
- **Services:** implementa la lógica de negocio de cada entidad.

En la carpeta resources se encuentran los recursos del proyecto, incluyendo el archivo application.properties que define la configuración principal, como la conexión a la base de datos o los puertos utilizados por el servidor.

## Front End

La estructura del Front End se encuentra organizada principalmente dentro de la carpeta src y se ha diseñado para facilitar la modularidad y el mantenimiento del código.

La organización de carpetas y archivos es la siguiente:

- **components:** incluye componentes reutilizables en distintas páginas, como botones, cabeceras o formularios.
- **context:** gestiona el estado global de la aplicación mediante React Context.
- **hooks:** almacena hooks personalizados creados para encapsular lógica reutilizable.
- **pages:** contiene las diferentes vistas o pantallas de la aplicación, como inicio de sesión, registro, dashboard, entre otras.
- **services:** define las funciones encargadas de la comunicación con la API (peticiones HTTP).
- **styles:** agrupa los archivos CSS utilizados para el diseño de la interfaz.

En la raíz del proyecto también se encuentran los archivos principales de configuración y entrada:

- **index.html:** archivo HTML base que sirve como punto de entrada para la aplicación.
- **main.jsx:** punto de entrada de la aplicación React.
- **App.jsx:** componente principal donde se configuran las rutas y la estructura general de la interfaz.
- **vite.config.js:** archivo de configuración de Vite.

#### 4.3.3 Endpoints de la API REST

En esta sección se describen los endpoints de los que dispone la API en los diferentes módulos del proyecto.

##### Módulo de Usuarios

El módulo de usuarios de la API REST ofrece los endpoints necesarios para gestionar la autenticación y administración de cuentas dentro de la plataforma. Para el acceso a la aplicación, se disponen de dos endpoints principales: **POST /user/login&email=example@gmail.com&password=1234** para el inicio de sesión y **POST /user/register** para el registro de nuevos usuarios. Estos endpoints permiten la validación y creación de usuarios, respectivamente.

Además, el módulo incluye los endpoints típicos para la gestión de usuarios, tales como **GET /user** para obtener información del perfil, **PUT /user** para actualizar datos completos, y **PATCH /user/updatePassword/:id**, **PATCH /user/updateName/:id** para la modificación de la contraseña. También está disponible el endpoint **DELETE /user/:id** que permite eliminar la cuenta de forma definitiva.

Como funcionalidad adicional, se ha implementado un endpoint específico **GET /user/email/:email** que permite la consulta de un usuario a partir de su correo electrónico, función que puede ser utilizada por administradores o desarrolladores para facilitar la gestión y búsqueda de usuarios dentro del sistema.

##### Módulo de Transaction

El módulo de transacciones proporciona los endpoints necesarios para la gestión completa de las operaciones de compra y venta de activos dentro de las carteras de inversión. Se implementan los métodos CRUD habituales:

- **POST /transactions** para crear una nueva transacción.
- **GET /transactions/:id** para obtener los detalles de una transacción específica.
- **PUT /transactions** para actualizar una transacción existente.
- **DELETE /transactions/:id** para eliminar una transacción.

Además, se incluye un endpoint adicional **GET /transactions/portfolio/:portfolio\_id** que permite consultar todas las transacciones asociadas a una cartera determinada, facilitando así la visualización y seguimiento detallado de las operaciones realizadas por cartera.

##### Módulo de Portfolio

El módulo de portfolio proporciona los endpoints necesarios para la gestión completa de los portfolios de inversión de los usuarios. Se incluyen los métodos CRUD habituales:

- **POST /portfolios** para crear un nuevo portfolio.

- **GET /portfolios/:id** para obtener los detalles de un portfolio específico.
- **PUT /portfolios** para actualizar un portfolio de forma completa.
- **PATCH /portfolios/:id** para actualizar parcialmente un portfolio.
- **DELETE /portfolios/:id** para eliminar un portfolio.

Además, se han implementado dos endpoints adicionales para facilitar consultas específicas:

- **GET /portfolios/user/:user\_id** que permite obtener todos los portfolios asociados a un usuario por su identificador.
- **GET /portfolios/email/:email** que permite obtener los portfolios asociados a un usuario a partir de su correo electrónico.

Estos endpoints permiten una gestión eficiente y flexible de los portfolios, adaptándose a las necesidades tanto de usuarios finales como de procesos administrativos.

### **Módulo de Consulta de Precios en Tiempo Real (API Alpha Vantage)**

Este módulo integra la API externa de Alpha Vantage para obtener datos de mercado en tiempo real sobre diferentes tipos de activos financieros. Permite consultar el precio actual de acciones, criptomonedas y ETFs mediante los siguientes endpoints:

- **GET /stock/:stock**: Obtiene el precio actual de la acción especificada mediante su símbolo bursátil.
- **GET /crypto/:crypto/EUR**: Proporciona el precio actual de la criptomoneda indicada (por ejemplo, BTC) en euros.
- **GET /etf/:etf**: Recupera el precio actual del ETF especificado mediante su símbolo.

Estos endpoints facilitan la actualización dinámica de los valores de inversión en la plataforma, permitiendo a los usuarios tener acceso a información financiera precisa y en tiempo real para tomar decisiones informadas.

### **Módulo de modelo de lenguaje (IA)**

El sistema incluye un modelo de lenguaje que permite al usuario realizar preguntas relacionadas con el ámbito financiero. Este LLM cuenta con un único endpoint de tipo POST en la ruta /preguntar. El usuario envía una pregunta en formato JSON y el sistema responde con un texto generado por el modelo de lenguaje. Esta funcionalidad permite simular una interacción tipo chatbot para resolver dudas o recibir información financiera de forma sencilla.

### **Gestión de errores**

Para garantizar una correcta gestión de errores en el sistema, se ha implementado una excepción personalizada en el backend desarrollada con Spring Boot. Esta clase, denominada `ResourceNotFoundException`, permite controlar de forma específica los

casos en los que un recurso solicitado (como un usuario, una cartera o una transacción) no se encuentra en la base de datos.

Además, permite incluir un mensaje personalizado que proporciona información más clara sobre el error producido.

Esta excepción se utiliza principalmente en los métodos de tipo GET de los diferentes módulos (usuarios, portfolios y transacciones). En estos métodos, si no se encuentra el recurso solicitado, se lanza la excepción desde el servicio correspondiente, lo que permite que el controlador capture el error y devuelva una respuesta adecuada al cliente, con un mensaje descriptivo y el estado HTTP correspondiente.

De esta manera, el sistema mejora su robustez y experiencia de usuario al proporcionar respuestas claras y coherentes ante errores comunes, como la consulta de un recurso不存在.

La gestión de errores se ha llevado a cabo en todos los módulos principales, aplicándose a cada método que involucra la recuperación de datos por identificador o por criterios específicos como el correo electrónico, garantizando así una gestión de los errores correcta en toda la aplicación.

#### 4.4. Implementación del frontend

En esta sección se detalla la implementación del front end de la aplicación, centrándose en la construcción de los componentes principales de la interfaz, el consumo de la API para la obtención y envío de datos, y la gestión del estado de autenticación del usuario.

##### 4.4.1 Componentes principales

En este apartado se describen algunos de los componentes principales desarrollados para el frontend de la aplicación. Dentro de la carpeta **components** se han definido elementos reutilizables que están presentes en varias páginas, como el encabezado y el pie de página.

El **encabezado** incluye el título del proyecto, *MyPortfolio*. Si el usuario ha iniciado sesión, este título actúa como un enlace que redirige directamente al panel principal desde cualquier sección. Además, en ese estado se muestran dos iconos: uno para acceder al perfil del usuario y otro para cerrar sesión. En cambio, si no se ha iniciado sesión, estos iconos se sustituyen por los enlaces de registro e inicio de sesión, que llevan a sus respectivas páginas.

Por su parte, el **pie de página** es más simple e incluye una breve información sobre la autoría de la plataforma, los derechos de autor, y una nota indicando que el desarrollo del proyecto corresponde a un trabajo final de grado.

La carpeta **pages** contiene todas las vistas principales del proyecto, es decir, las páginas que componen la interfaz de usuario. A continuación, se describen brevemente las más relevantes:

**Dashboard** es la página principal a la que se accede tras iniciar sesión (ver figura 1.14). En ella se muestran todos los portafolios del usuario junto con su balance total, sin necesidad de acceder individualmente a cada uno. Esta vista incluye un botón para ocultar o mostrar los totales de los portafolios, así como la opción de crear uno nuevo.

Además, desde el dashboard se puede acceder directamente a otras funcionalidades como el buscador de activos o la página del modelo de lenguaje.



Figura 1.14 Página Dashboard de la plataforma web

**Portfolio** es una de las páginas principales del sistema, ya que centraliza la visualización y gestión de las transacciones asociadas a un portafolio concreto (ver figura 1.15). En esta vista, el usuario puede consultar de forma clara todas sus operaciones registradas, con la posibilidad de editar el nombre del portafolio o eliminarlo si lo desea. Además, se incluye un botón identificado como *Comprar/Vender activos*, que redirige a la página correspondiente para añadir nuevas transacciones. Por último, se facilita la navegación hacia el detalle de cada operación: con un simple clic sobre una transacción, el usuario accede a su información individualizada.

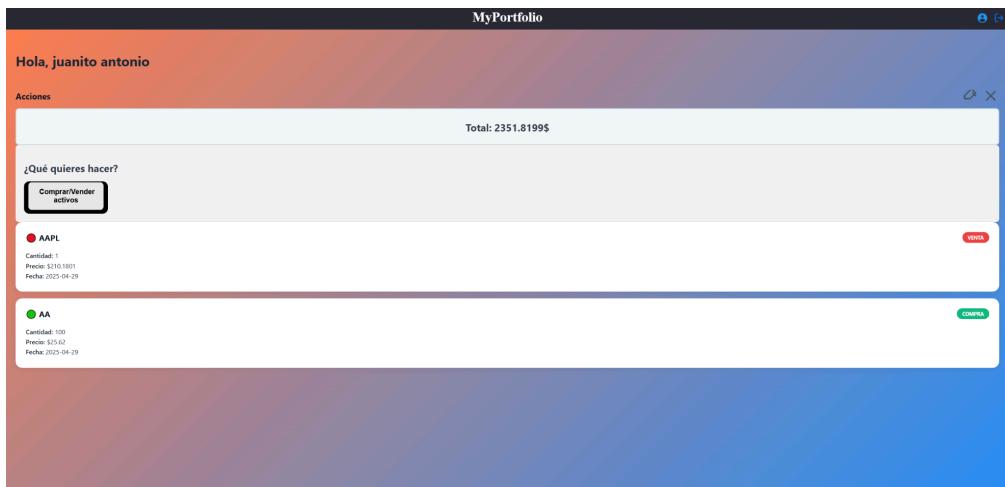


Figura 1.15 Página Portfolio de la plataforma web

**Transaction** muestra la información detallada de una transacción concreta (ver figura 1.16). Desde esta vista es posible regresar directamente al dashboard o eliminar la transacción actual. También incluye una sección con el resto de transacciones pertenecientes al mismo portafolio, facilitando así la navegación entre ellas sin tener que volver a la página de portafolio en cada ocasión.

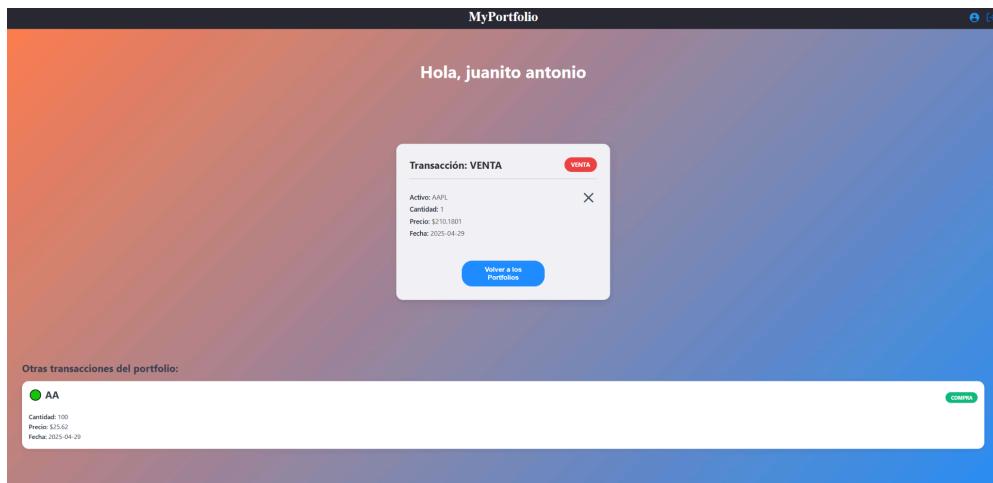


Figura 1.16 Página Transaction de la plataforma web

**Register** es la primera página que visualiza un usuario que aún no ha iniciado sesión (ver figura 1.17). En ella se presenta un formulario que, una vez completado, permite registrar un nuevo usuario en el sistema. Para garantizar la integridad de los datos, no se permite registrar más de un usuario con el mismo correo electrónico. Además, la página incluye un botón que redirige a la vista de inicio de sesión, pensado para aquellos usuarios que ya dispongan de una cuenta.

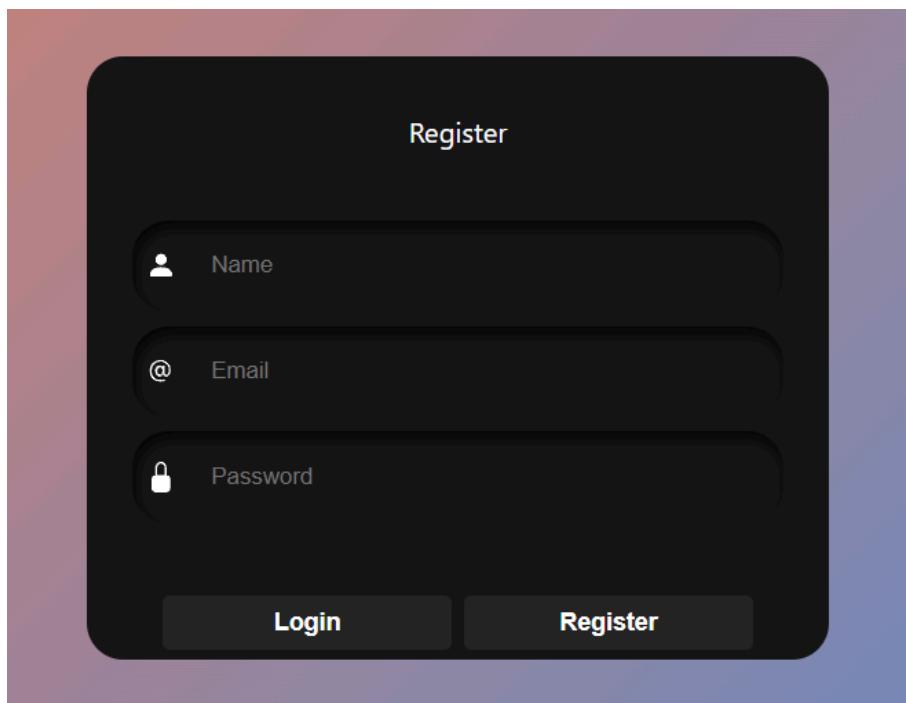


Figura 1.17 Página Register de la plataforma web

**Login** presenta un formulario similar al de registro, aunque en este caso únicamente solicita el correo electrónico y la contraseña para autenticar al usuario (ver figura 1.18). También incluye un botón para acceder rápidamente a la página de registro, en caso de que el usuario aún no esté dado de alta en el sistema.

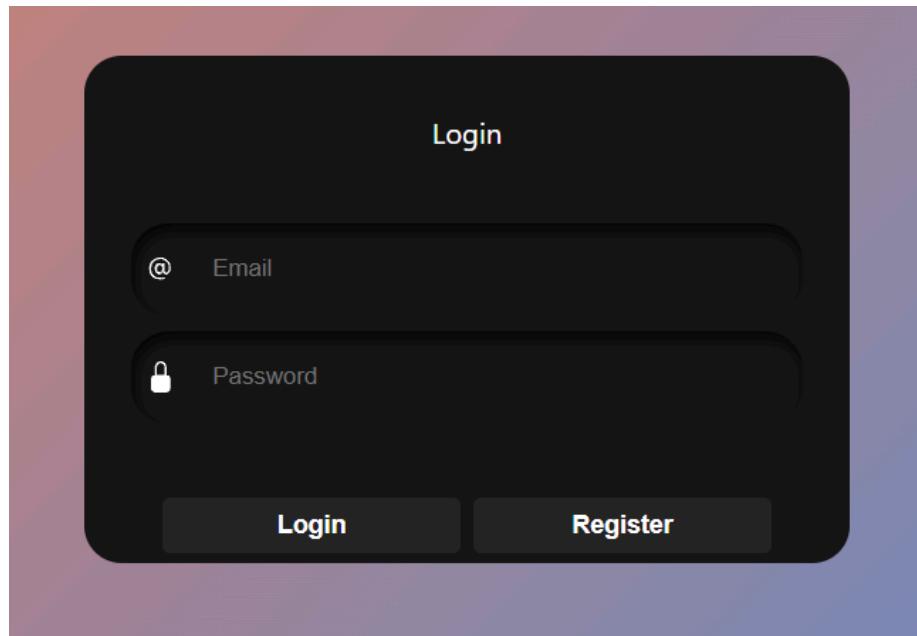


Figura 1.18 Página Login de la plataforma web

**Profile** es la sección destinada a la gestión de la cuenta del usuario. En esta página se muestran datos básicos como el nombre y el correo electrónico (ver figura 1.19). Además, desde aquí se proporciona acceso a distintas acciones: modificar el nombre de usuario (redirigiendo a la vista correspondiente), cambiar la contraseña o cerrar sesión de manera segura.



Figura 1.19 Página Profile de la plataforma web

**Editar perfil** es la página en la que el usuario puede actualizar su nombre (ver figura 1.20). Presenta un único campo para introducir el nuevo nombre deseado, junto con un botón para guardar los cambios y aplicar la modificación.

**Editar Perfil**

Nombre:

Guardar Cambios

Figura 1.20 Página Editar Perfil de la plataforma web

**Cambiar contraseña** permite al usuario actualizar su clave de acceso (ver figura 1.21). Por motivos de seguridad, esta página incluye dos campos para introducir la nueva contraseña, con el fin de confirmar que no se haya cometido ningún error al escribirla. También dispone de un botón para confirmar y guardar los cambios realizados.

**Cambiar Contraseña**

Nueva Contraseña:

Confirmar Nueva Contraseña:

Guardar Cambios

Figura 1.21 Página Cambiar Contraseña de la plataforma web

**Modelo** es la página que permite al usuario interactuar directamente con el modelo de lenguaje integrado en la plataforma (ver figura 1.22). En su diseño inicial, se presenta un único campo de entrada donde se introduce la consulta. Una vez enviada, y tras recibir la respuesta, esta se muestra de forma dinámica justo debajo del campo original.

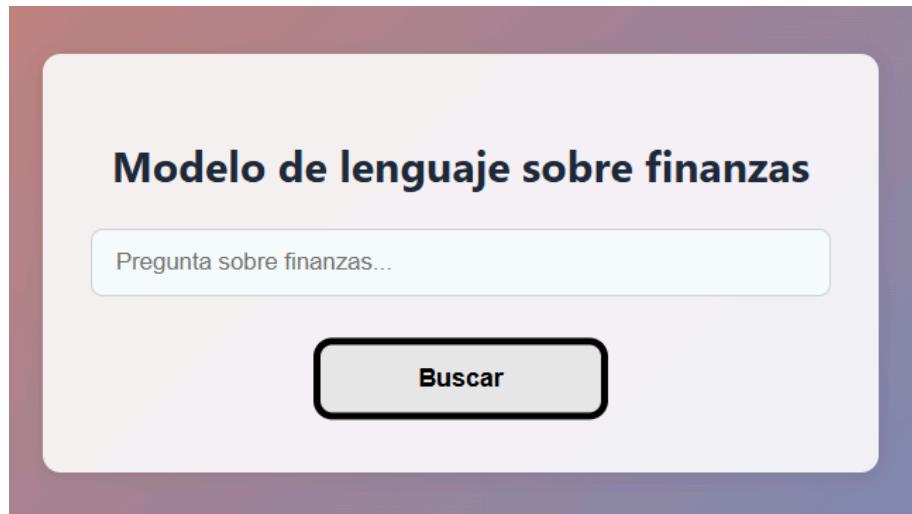


Figura 1.22 Página Modelo de lenguaje de la plataforma web

**Buscador** ofrece una funcionalidad de búsqueda de activos financieros (ver figura 1.23). Cuenta con un campo para introducir el valor a consultar y un desplegable que permite seleccionar el tipo de activo: acción (stock), criptomoneda o ETF. Según la opción elegida, se realiza una llamada diferente a la API externa para obtener los datos correspondientes. La información se presenta de forma inmediata debajo del buscador.

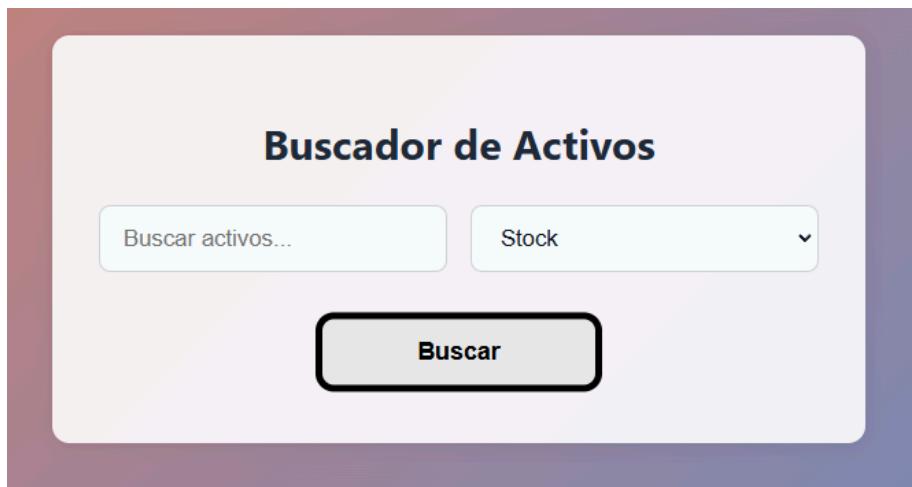


Figura 1.23 Página Buscador de activos de la plataforma web

**NewPortfolio** es la vista dedicada a la creación de un nuevo portfolio (ver figura 1.24). Su funcionamiento es sencillo: dispone de un único campo donde introducir el nombre del nuevo portfolio y un botón para confirmar. Una vez completado el proceso, el usuario es redirigido automáticamente al dashboard.

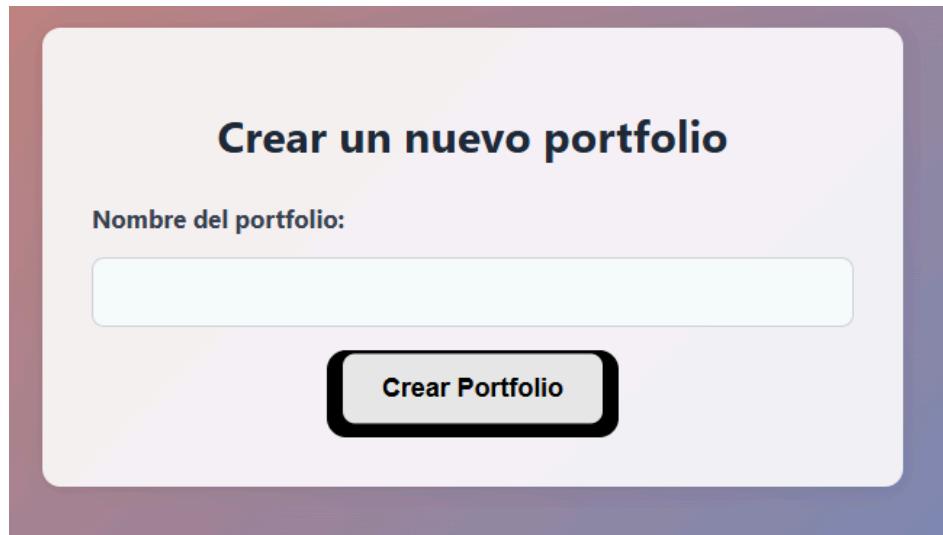


Figura 1.24 Página Crear Portfolio de la plataforma web

**EditPortfolio** permite modificar el nombre de un portfolio ya existente (ver figura 1.25). Su funcionamiento es similar al de la creación, con un campo para el nuevo nombre y un botón para guardar los cambios. Tras la edición, el sistema redirige de nuevo al dashboard.

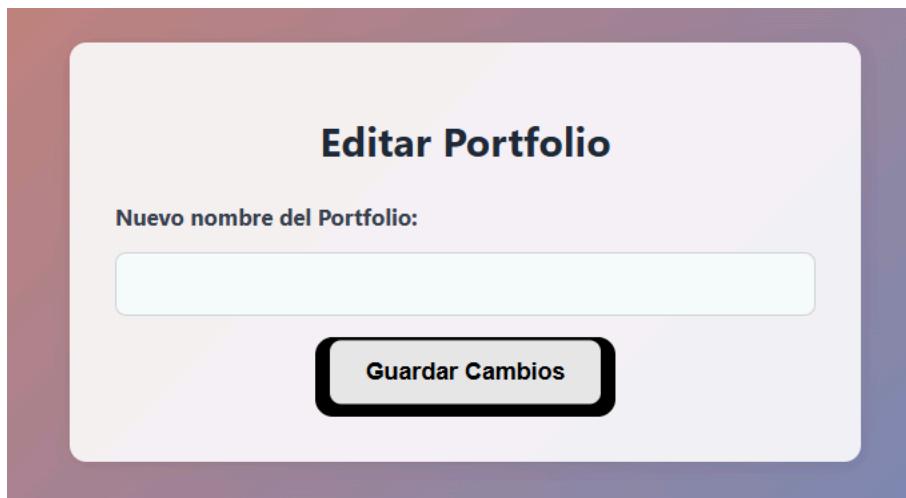


Figura 1.25 Página Editar Portfolio de la plataforma web

**Compra/Vender** es la página desde la cual el usuario puede registrar nuevas operaciones dentro de un portafolio existente (ver figura 1.26). En ella se incluye un campo donde se debe indicar el símbolo del activo (acción, criptomoneda o ETF), acompañado de un desplegable que permite seleccionar el tipo de activo correspondiente. Además, se ofrece otro desplegable para especificar si la operación será una compra o una venta. Finalmente, un botón de confirmación permite registrar la transacción en el sistema.

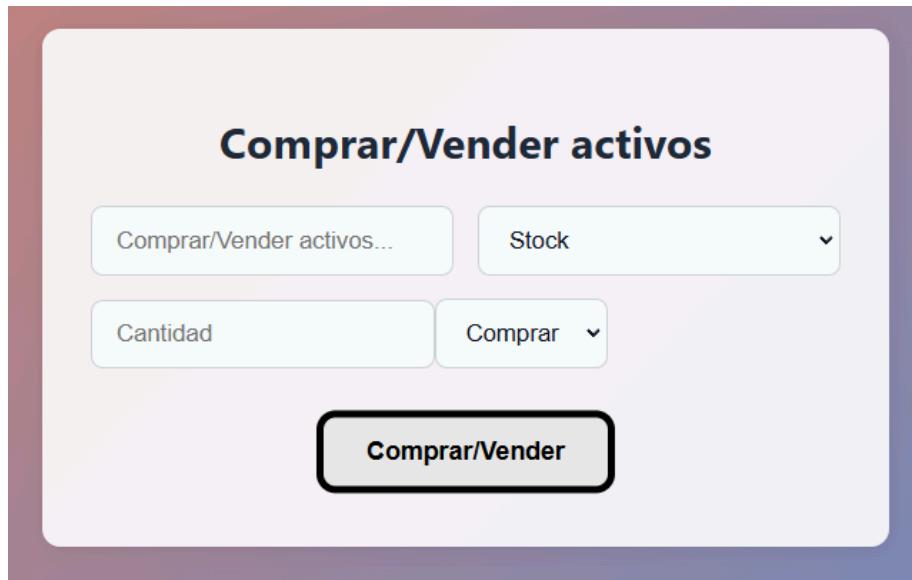


Figura 1.26 Página Comprar/Vender Activos de la plataforma web

## Flujo de navegación

La navegación del frontend se gestiona utilizando la librería React Router Dom. En el archivo App.jsx se define la estructura principal de rutas, que permite conectar cada dirección con su componente correspondiente. Gracias a esto, el usuario puede moverse entre páginas sin necesidad de recargar la aplicación, lo que mejora la experiencia de uso.

Cada ruta está pensada para representar una vista concreta de la aplicación. Por ejemplo, la página principal o dashboard se accede desde la ruta raíz "/", mientras que otras como "/portfolio/:id" o "/transaction/:id\_portfolio/:id\_transaction" permiten acceder a portafolios y transacciones concretas mediante parámetros en la URL.

También se incluyen rutas para el inicio de sesión, registro, gestión de portafolios, funcionalidades como buscador de activos o consulta al modelo de lenguaje, y otras relacionadas con el perfil del usuario. De esta manera, el flujo de navegación es claro, directo y bien organizado.

Además de la configuración general, existen funciones internas que permiten navegar entre páginas de forma programada. Un ejemplo es cuando el usuario hace clic sobre un portafolio desde el dashboard, lo que lanza una función que redirige a su vista específica con el identificador correspondiente.

### 4.4.2 Consumo de la API

Esta sección describe el consumo de la API desde el front end, detallando los distintos servicios implementados para interactuar con el back end. Entre ellos se incluyen los servicios de usuario, portafolio, transacciones, obtención de portafolios por ID, registro, inicio de sesión, edición de perfil, entre otros.

#### Servicio user

Este servicio permite obtener la información de un usuario a partir de su email. Utiliza fetch para realizar una solicitud HTTP al endpoint correspondiente del backend. Primero verifica que el campo de email no esté vacío antes de iniciar la petición. Si la

respuesta del servidor no es válida o hay algún problema durante la llamada, se lanza una excepción con un mensaje claro. En caso de éxito, se procesa la respuesta en formato JSON y se devuelve la información del usuario para su posterior uso.

### **Servicio portfolio**

Este servicio se encarga de obtener los portfolios asociados a un usuario a partir de su email. Realiza una petición HTTP al endpoint correspondiente del backend usando fetch. Si la respuesta no es satisfactoria, lanza un error con un mensaje específico para facilitar la depuración. En caso contrario, transforma la respuesta en formato JSON y devuelve la lista de portfolios obtenida. Esta información se puede utilizar posteriormente en el cliente para mostrar los portafolios del usuario.

### **Servicio transaction**

Este servicio permite obtener los detalles de una transacción específica a partir de su ID. Para ello, realiza una petición HTTP al endpoint correspondiente utilizando fetch. Si la respuesta del servidor no es correcta, lanza un error que facilita la identificación del problema. Si todo va bien, convierte la respuesta a formato JSON y devuelve la información de la transacción solicitada. Esta funcionalidad es clave para mostrar los datos detallados de una operación concreta dentro de un portafolio.

### **Servicio portfolioById**

Este servicio se encarga de obtener la información detallada de un portafolio a partir de su ID. Realiza una llamada HTTP al endpoint correspondiente y, si la respuesta es exitosa, transforma los datos recibidos en formato JSON para su posterior uso. En caso de error, lanza una excepción que puede ser manejada para informar al usuario. Esta funcionalidad es útil para cargar los datos de un portafolio específico al que se accede desde el dashboard o desde otra parte del sistema.

### **Servicio register**

Este servicio permite registrar a un nuevo usuario en el sistema. Envía una solicitud de tipo POST al backend incluyendo en el cuerpo los datos necesarios: correo electrónico, contraseña y nombre. Si la operación es exitosa y el servidor responde con un objeto que contiene un ID, se interpreta como que el registro fue completado correctamente. En caso contrario, se devuelve un valor falso o se lanza un error que puede ser tratado adecuadamente. Este servicio es fundamental para dar de alta nuevos usuarios desde la interfaz de registro.

### **Servicio login**

Este servicio se encarga de gestionar el inicio de sesión de los usuarios. Envía una solicitud POST al backend con el correo electrónico y la contraseña que introduce el usuario. Estos datos se mandan en formato JSON dentro del cuerpo de la petición. El servidor devuelve una respuesta indicando si las credenciales son válidas o no. Si todo es correcto, devuelve true. En caso de que haya algún error, ya sea por problemas en el servidor o porque los datos no coinciden, el servicio captura el fallo y devuelve false, indicando que el acceso no ha sido posible.

### **Servicio editar perfil**

Este servicio permite modificar el nombre de un usuario. Para ello, realiza una petición PATCH al backend enviando el nuevo nombre y el identificador del usuario correspondiente. La información se manda en formato JSON dentro del cuerpo de la solicitud. Si la operación es exitosa y el servidor devuelve una respuesta válida, se interpreta como un cambio correcto. En caso de error, ya sea por problemas en el servidor o por una respuesta inesperada, se captura la excepción y se informa de que no ha sido posible realizar la modificación.

### **Servicio editar contraseña**

Este servicio permite actualizar la contraseña de un usuario. Utiliza una solicitud PATCH que se envía al backend junto con el nuevo valor de la contraseña y el identificador del usuario correspondiente. El contenido se transmite en formato JSON y, si el servidor responde de forma correcta con un objeto válido, se da por hecha la modificación. Si ocurre algún error, ya sea por fallos en la comunicación o por una respuesta incorrecta, se captura y se informa del problema, devolviendo un resultado negativo.

### **Servicio crear portfolio**

Este servicio permite crear un nuevo portfolio para un usuario. Se envía una solicitud POST al servidor con el nombre del portfolio, el identificador del usuario y un array vacío de transacciones. Si la respuesta del servidor es correcta y devuelve un objeto con un identificador, se considera que la creación fue exitosa. En caso de error o respuesta incorrecta, se captura el fallo y se devuelve un resultado negativo.

### **Servicio editar portfolio**

Este servicio permite modificar el nombre de un portfolio existente. Se realiza una petición PATCH al servidor enviando el nuevo nombre y el identificador del portfolio que se desea actualizar. Si la respuesta es correcta y contiene un identificador, se considera que la actualización fue exitosa. En caso de que ocurra algún error o la respuesta no sea la esperada, se captura y se devuelve un resultado negativo.

### **Servicio eliminar portfolio**

Este servicio permite borrar un portfolio a partir de su identificador. Primero verifica que el id no esté vacío o sea nulo para evitar errores. Luego hace una petición DELETE al servidor para eliminar el portfolio indicado. Si la respuesta es correcta, devuelve un resultado positivo; en caso contrario, captura el error y devuelve falso indicando que la eliminación no se pudo realizar.

### **Servicio comprar vender**

Este servicio se utiliza para añadir una nueva transacción de compra o venta a un portfolio. Recibe varios datos como el tipo de transacción, el identificador del activo, la cantidad, el precio, la fecha y el id del portfolio. Estos datos se preparan y se envían al backend mediante una petición POST en formato JSON. Si la operación es exitosa, devuelve los datos de la transacción; si hay algún error, captura el problema y devuelve null.

### **Servicio eliminar transacción**

Este servicio permite eliminar una transacción específica usando su identificador. Realiza una petición DELETE al backend para borrar la transacción correspondiente. Si la operación se realiza correctamente, devuelve true y muestra un mensaje de éxito; en caso de error, captura la excepción y devuelve false.

### **Servicio buscar activo por categoría**

Este servicio permite buscar información sobre un activo financiero según su categoría, que puede ser criptomoneda, acciones o ETF. Construye la URL según el tipo de activo y realiza una llamada al backend para obtener los datos. Si la respuesta es correcta, devuelve la información recibida como texto. En caso de error o falta de parámetros, muestra un mensaje de error y retorna null.

### **Servicio modelo**

Este servicio envía una pregunta al modelo de lenguaje a través de una petición POST al backend. Envía la consulta en formato JSON y espera recibir la respuesta en texto plano. Si la respuesta es correcta, devuelve el resultado. En caso de error, muestra un mensaje en consola y retorna false.

#### **4.4.3 Gestión del estado de autenticación**

La gestión del estado de autenticación se implementa mediante un contexto de React llamado AuthContext. Este contexto almacena información sobre si el usuario está autenticado, su email y su ID. Cuando el usuario inicia sesión, se guarda su email y se marca como autenticado; esta información también se persiste en localStorage para mantener la sesión entre recargas de página. Al cerrar sesión, se limpia esta información y se redirige al usuario a la página de login. Además, el contexto ofrece métodos para iniciar y cerrar sesión, así como para actualizar el ID del usuario. Gracias a useContext, esta información está disponible en toda la aplicación para que cualquier componente pueda acceder a los datos de autenticación y reaccionar ante sus cambios.

### **4.5. Implementación del backend**

En esta sección se expone la implementación del back end de la aplicación, abordando la lógica de negocio que sustenta las funcionalidades principales, la conexión y gestión de la base de datos, así como la organización del código a través de controladores y servicios.

#### **4.5.1. Explicación de la lógica de negocio**

La estructura del backend sigue una arquitectura en capas bien definida, donde cada capa tiene una responsabilidad clara y delimitada. En primer lugar, se encuentran las entidades (entities), que representan el modelo de datos y definen la estructura de las tablas en la base de datos. A continuación, se dispone de los repositorios (repositories), que extienden de JpaRepository y proporcionan acceso directo a los datos persistentes.

Sobre estos repositorios se construyen interfaces de servicio, que definen los métodos que serán utilizados en la lógica de negocio. Estas interfaces son implementadas en la

capa de servicios (services), donde se desarrollan las operaciones concretas que interactúan con la base de datos y aplican la lógica del sistema.

Finalmente, los controladores (controllers) son los encargados de exponer las funcionalidades del backend mediante endpoints REST. Estos controladores acceden a los servicios a través de inyección de dependencias por constructor, lo que permite mantener una separación clara entre la lógica de presentación y la lógica de negocio. Todos los controladores están anotados con `@RestController`, lo que permite exponer endpoints accesibles vía HTTP, enviando y recibiendo datos en formato JSON. Estas rutas permiten a los clientes interactuar con el sistema realizando operaciones como registro de usuarios, creación y gestión de carteras, ejecución de transacciones de compra y venta, consultas a un modelo de lenguaje natural (LLM) y búsquedas de activos por categoría. Además, los usuarios pueden editar sus datos personales o eliminar elementos de su cuenta.

La estructura de desarrollo sigue un patrón estándar donde las **entidades** representan los modelos de datos que se almacenan en la base de datos. A continuación, se definen interfaces como **PortfolioService** o **UserService**, que agrupan las operaciones disponibles para cada entidad. Estas interfaces son implementadas por clases concretas de servicio que contienen la lógica de negocio, mientras que los **repositorios** (**UserRepository**, **TransactionRepository**, etc.) heredan de **JpaRepository** para facilitar el acceso a la base de datos sin necesidad de escribir SQL manual.

Los controladores acceden a los servicios mediante **inyección de dependencias por constructor**, una técnica que permite a Spring gestionar la creación de los objetos necesarios. Aunque se puede utilizar la anotación `@Autowired` para injectar directamente en campos, en este proyecto se ha optado por realizar la inyección a través del constructor del controlador, lo cual es una práctica recomendada por su mayor claridad, facilidad de testeo y compatibilidad con herramientas de análisis estático.

Además, se han aplicado mecanismos de **control de excepciones** para manejar situaciones como intentos de recuperar usuarios inexistentes, devolviendo mensajes de error adecuados en los casos correspondientes. De este modo, se mejora la robustez del sistema y se ofrecen respuestas claras a los clientes ante posibles errores.

En resumen, la lógica de negocio está diseñada para responder a los distintos flujos que puede seguir un usuario en la plataforma, asegurando una interacción fluida, coherente y segura entre la capa cliente y la base de datos, a través de endpoints RESTful bien definidos.

#### 4.5.2. Conexión con la base de datos

La conexión con la base de datos se ha implementado utilizando el soporte que ofrece Spring Boot junto con Spring Data JPA. La base de datos utilizada es MySQL, y la configuración se ha realizado a través del archivo **application.properties**, donde se han definido los parámetros de conexión como la URL del servidor, el nombre de usuario y la contraseña. También se han establecido propiedades de JPA como el modo **ddl-auto en valor "update"**, lo que permite que las tablas se creen y modifiquen automáticamente según la estructura de las entidades Java. Además, se ha activado la

opción para mostrar las consultas SQL que se ejecutan, facilitando así la depuración y el desarrollo.

Para que la aplicación pueda comunicarse con MySQL, se ha añadido la dependencia correspondiente en el fichero de configuración del proyecto (pom.xml), incluyendo el conector JDBC de MySQL en el entorno de ejecución.

El modelo de datos se ha estructurado mediante tres entidades principales: usuario, cartera (portfolio) y transacción. La relación entre estas entidades responde a la lógica de negocio del sistema. Un usuario puede tener varias carteras, pero cada cartera pertenece únicamente a un usuario. A su vez, una cartera puede contener múltiples transacciones, y cada transacción está asociada a una única cartera.

Estas relaciones se han definido mediante anotaciones JPA. Por ejemplo, la entidad "User" contiene una lista de objetos tipo "Portfolio", y dicha relación está configurada para que cualquier modificación en cascada (como la eliminación de un usuario) se propague también a sus carteras. A su vez, la entidad "Portfolio" mantiene una relación uno a muchos con "Transaction", que sigue el mismo principio de propagación y referencia única.

Para evitar problemas de serialización y referencias cíclicas en las respuestas JSON de la API, se han utilizado anotaciones de Jackson, como las que ignoran ciertas propiedades o identificar objetos por su ID cuando hay relaciones bidireccionales.

La persistencia se gestiona a través de interfaces que extienden de JpaRepository, como por ejemplo UserRepository, PortfolioRepository y TransactionRepository. Esto permite realizar operaciones básicas como guardar, buscar o eliminar registros sin necesidad de escribir consultas SQL manuales. Las operaciones de mayor complejidad o que implican lógica de negocio específica se han trasladado a la capa de servicios, separando así las responsabilidades y manteniendo el código más modular, claro y mantenible.

#### 4.5.3. Controladores y servicios

La aplicación implementa una arquitectura basada en controladores y servicios, siguiendo el patrón **MVC (Modelo-Vista-Controlador)**. Cada controlador es responsable de recibir las solicitudes HTTP entrantes, validar los datos, y delegar la lógica de negocio a su respectivo servicio. Esta estructura permite mantener una separación clara entre la lógica de presentación y la lógica de negocio, lo que mejora la mantenibilidad y escalabilidad del sistema.

A continuación, se describen los principales controladores implementados, junto con su relación con los servicios correspondientes.

##### UserController

El controlador UserController gestiona todas las operaciones relacionadas con los usuarios del sistema. Entre las rutas principales que maneja se encuentran el registro (/user/register), el inicio de sesión (/user/login), la obtención de información por ID o email, y la actualización de datos como nombre o contraseña.

Los datos se reciben en formato JSON o como parámetros en la URL, y las respuestas se devuelven en formato JSON, utilizando objetos ResponseEntity para controlar los

códigos de estado HTTP y los mensajes de error. La lógica de autenticación, validación de credenciales y actualizaciones se delega al servicio UserService, que interactúa con el repositorio UserRepository para persistir los datos.

### **PortfolioController**

El controlador PortfolioController está encargado de la gestión de carteras de inversión de los usuarios. Ofrece rutas para registrar nuevas carteras, obtener carteras por ID de usuario o email, y actualizar o eliminar portfolios existentes.

Los datos se transmiten en formato JSON o como parámetros en la URL, y las respuestas siguen el mismo formato estructurado. Este controlador trabaja conjuntamente con el servicio PortfolioService, que encapsula la lógica de negocio y se comunica con PortfolioRepository para el acceso a la base de datos.

### **TransactionController**

El controlador TransactionController permite registrar, consultar, modificar y eliminar transacciones financieras asociadas a las carteras. Las rutas disponibles permiten operar tanto sobre una transacción específica como sobre todas las asociadas a un portfolio determinado.

Al igual que los demás controladores, utiliza ResponseEntity para gestionar las respuestas y TransactionService para aplicar la lógica de negocio. Este servicio a su vez se apoya en TransactionRepository para realizar las operaciones CRUD sobre la base de datos.

### **OllamaController**

El controlador OllamaController expone un único endpoint (/preguntar) destinado a recibir preguntas en lenguaje natural y obtener respuestas generadas por un modelo de lenguaje (LLM) alojado localmente o externamente.

Este controlador recibe un objeto JSON con una propiedad pregunta y responde con un String plano que contiene la respuesta del modelo. La lógica de comunicación con el modelo se encuentra en el servicio OllamaService, que actúa como cliente HTTP y envía solicitudes POST al endpoint del modelo.

### **StockPriceController**

El controlador StockPriceController proporciona datos financieros en tiempo real, incluyendo precios de acciones, criptomonedas y ETFs, a través de la API externa de Alpha Vantage. Las rutas disponibles permiten obtener estos precios mediante parámetros incluidos en la URL.

Este controlador se comunica con el servicio StockPriceService, que encapsula las llamadas HTTP a la API externa y gestiona la transformación y entrega de los datos relevantes. Las respuestas se entregan como String o en formato JSON, según la necesidad.

## Estructura común y buenas prácticas

Todos los controladores comparten una estructura coherente basada en las anotaciones de Spring (@RestController, @RequestMapping, @Autowired), lo que garantiza un manejo uniforme de las solicitudes HTTP y la inyección de dependencias.

El uso de ResponseEntity permite personalizar los códigos de estado HTTP y los mensajes de error, facilitando la identificación de fallos y mejorando la experiencia del cliente.

La validación de datos y el manejo de errores se realizan preferentemente en la capa de servicios, lo que mantiene los controladores ligeros y enfocados únicamente en el enrutamiento y respuesta de las solicitudes. Las operaciones complejas o que implican reglas de negocio específicas se encapsulan en la lógica de los servicios, garantizando así un código más modular, reutilizable y fácil de mantener.

### 4.6. Integración con APIs externas

En esta sección se describen todos los aspectos que tengan relación con la API externa de Alpha Vantage desde que es, hasta qué servicios se utilizan.

#### 4.6.1. Alpha Vantage

Alpha Vantage[43] es una plataforma que proporciona acceso a datos financieros tanto en tiempo real como históricos, a través de una interfaz de programación de aplicaciones (API). Está orientada a desarrolladores, analistas, inversores y cualquier usuario que desee integrar información del mercado financiero en sus sistemas, aplicaciones o herramientas de análisis.

Es ampliamente utilizada por su facilidad de uso, su modelo de acceso gratuito con opciones premium, y la amplitud de los datos ofrecidos. Se ha convertido en una de las alternativas más populares entre quienes desarrollan software financiero, gracias a su estructura sencilla de consultas, su compatibilidad con múltiples formatos de salida como JSON y CSV, y la disponibilidad de bibliotecas creadas por la comunidad para diversos lenguajes de programación.

Alpha Vantage pone a disposición una serie de APIs organizadas en distintas categorías, que permiten acceder a información financiera de gran utilidad:

- **Datos de acciones (Stocks):** Incluye series temporales intradía, diarias, semanales y mensuales de valores bursátiles, útiles para análisis técnico y seguimiento de precios.
- **Datos de opciones financieras:** Ofrece información de contratos de opciones en el mercado estadounidense, con histórico completo y cobertura actualizada.
- **Monedas físicas y criptomonedas:** Proporciona tasas de cambio entre divisas tradicionales (como USD, EUR, JPY) y también entre criptomonedas como Bitcoin o Ethereum.
- **Indicadores económicos:** Incluye acceso a datos macroeconómicos clave como tasas de interés, PIB, desempleo, entre otros.

- **Datos fundamentales de empresas:** Acceso a información contable como balances, cuentas de resultados, flujos de caja, ratios financieros, etc.
- **Indicadores técnicos:** Dispone de más de 50 indicadores listos para su uso, como RSI, MACD, medias móviles, Bollinger Bands, entre otros.
- **Datos de materias primas (commodities):** Información de mercado relacionada con productos como petróleo, oro o gas natural.

Además, Alpha Vantage ofrece extensiones para hojas de cálculo como Excel y Google Sheets, permitiendo la integración directa de datos financieros en herramientas de análisis más accesibles para usuarios no técnicos.

Para utilizar la API de Alpha Vantage es necesario contar con una clave de acceso (API key). Esta clave se configura en el archivo application.properties de la siguiente manera:

**alpha.vantage.api.key=6MW8O2PIV5JHZ6PE**

De esta forma, Spring puede inyectar esta clave en los servicios mediante la anotación `@Value`.

Toda la lógica de comunicación con Alpha Vantage se encuentra encapsulada en la clase StockPriceService, ubicada en el paquete MyPortfolio.Services. Esta clase está anotada con `@Service` y se encarga de realizar las peticiones HTTP a los endpoints de Alpha Vantage, utilizando la clase RestTemplate. Además, se emplea la biblioteca Jackson para el procesamiento de las respuestas en formato JSON.

Este servicio ofrece tres métodos principales:

- `getStockPrice(String symbol)`: obtiene el precio actual de una acción usando la función TIME\_SERIES\_INTRADAY con un intervalo de 5 minutos. Extrae el último valor de cierre disponible.
- `getCryptoPrice(String fromCurrency, String toCurrency)`: consulta el tipo de cambio actual entre dos monedas, especialmente útil para ver el valor de criptomonedas como BTC, ETH, etc. respecto al USD u otra divisa.
- `getEtfPrice(String symbol)`: funciona igual que `getStockPrice` pero se utiliza para ETFs.

En todos los casos, se maneja el posible error mediante bloques try-catch para capturar tanto errores HTTP como problemas al procesar el JSON. Para obtener la última entrada de datos de la serie temporal, se utiliza el método privado `getLastTimestamp(JsonNode)`, que devuelve el primer campo del nodo JSON, correspondiente al timestamp más reciente.

El controlador que expone estos servicios es StockPriceController, ubicado en el paquete MyPortfolio.Controllers. Este controlador define tres endpoints públicos accesibles mediante peticiones HTTP GET:

- `/stock/{symbol}`: permite obtener el precio actual de una acción.
- `/crypto/{fromCurrency}/{toCurrency}`: devuelve el valor actual de una criptomoneda respecto a otra moneda.
- `/etf/{symbol}`: devuelve el valor actual de un ETF.

Estos endpoints son utilizados por el frontend o herramientas externas para obtener datos en tiempo real.

Para permitir que el frontend (por ejemplo, una aplicación desarrollada con React) acceda a estos datos, se ha configurado el sistema CORS dentro de la clase WebConfig. Esta configuración permite solicitudes desde <http://localhost:5173>, incluyendo métodos como GET, POST, PUT, DELETE y otros, así como encabezados personalizados y credenciales.

Por otro lado, el bean RestTemplate se declara en la clase ApplicationContext, permitiendo su inyección en los servicios de forma sencilla y reutilizable, lo cual es una práctica habitual en Spring Boot para centralizar la configuración de componentes.

Gracias a esta integración, la aplicación puede ofrecer a los usuarios información actualizada sobre sus activos financieros en tiempo real. Por ejemplo:

- Al visualizar una acción en la cartera del usuario, se realiza una petición al endpoint `/stock/AAPL` para obtener el valor actual de Apple.
- Al consultar una criptomoneda, como el Bitcoin, se accede al endpoint `/crypto/BTC/EUR` para mostrar su precio en euros.
- En el caso de fondos cotizados (ETFs), al consultar un activo como VOO, se utiliza `/etf/VOO` para mostrar su valor más reciente.

Esto permite a los usuarios tomar decisiones basadas en información de mercado actualizada directamente desde la aplicación.

#### **4.7. Integración del modelo de lenguaje**

En el desarrollo del sistema se ha integrado un modelo de lenguaje natural para permitir al usuario realizar preguntas y recibir respuestas automáticas, mejorando así la interactividad y utilidad de la aplicación. Para ello se ha utilizado **Ollama**, una plataforma local que permite ejecutar modelos grandes de lenguaje (LLMs) de manera eficiente en entornos controlados, sin necesidad de depender de servicios en la nube.

El LLM elegido ha sido **Llama3.2,[44]** que destaca por entre los grandes modelos de lenguaje de código abierto por su potencia y fiabilidad en la generación de texto. La integración se ha realizado a través de un servicio en el backend que se comunica con la API local de Ollama, la cual se encuentra disponible en <http://localhost:11434/api/generate>. El sistema envía una solicitud POST con el prompt introducido por el usuario y recibe como respuesta el texto generado por el modelo.

En el backend, esta funcionalidad se implementa mediante un servicio llamado OllamaService, el cual construye el cuerpo de la petición, establece los encabezados necesarios (como el tipo de contenido en formato JSON) y realiza la llamada HTTP usando RestTemplate. El resultado recibido se procesa y se devuelve al controlador, que lo expone a través del endpoint /preguntar.

En el frontend, se ha creado una función asíncrona que se encarga de consumir este endpoint. El usuario puede introducir una pregunta, que es enviada mediante una solicitud POST al backend. Este se encarga de enviarla al modelo de lenguaje y retornar la respuesta generada, que es mostrada en la interfaz de usuario.

Esta integración ha sido pensada para ejecutarse completamente en local, lo cual permite mantener la privacidad de los datos del usuario y garantiza un rendimiento rápido sin latencias asociadas a servicios externos. Además, la arquitectura modular del sistema permite reemplazar o actualizar el modelo fácilmente si en el futuro se desea utilizar otro motor de generación de texto.

Gracias a esta incorporación, se ha añadido una funcionalidad basada en inteligencia artificial al sistema que permite responder preguntas de los usuarios sobre los activos financieros, resolver dudas comunes y ofrecer asistencia en tiempo real.

#### 4.8. Pruebas y validación

Durante el desarrollo del sistema, se han realizado pruebas para verificar el correcto funcionamiento de los endpoints de la API. Para ello, se ha utilizado la herramienta **Postman**, una aplicación ampliamente utilizada en el ámbito del desarrollo de software para probar servicios web RESTful de manera rápida e interactiva.

Con Postman, ha sido posible enviar peticiones HTTP a los distintos endpoints definidos en los controladores de la aplicación y analizar las respuestas devueltas por el servidor. Esto ha permitido comprobar tanto la estructura como el contenido de las respuestas, así como validar los posibles errores que podrían surgir durante la comunicación entre el cliente y el backend.

Las pruebas realizadas incluyen:

- Consultas a endpoints como /stock/{symbol}, /crypto/{fromCurrency}/{toCurrency} y /etf/{symbol} para verificar que los datos devueltos por la API externa (Alpha Vantage) se integren correctamente y se devuelvan en el formato esperado.
- Comprobación de las rutas de usuario, carteras y transacciones, asegurando que las operaciones CRUD funcionen correctamente.
- Validación de los códigos de estado HTTP para confirmar que se devuelven respuestas adecuadas (por ejemplo, 200 OK en respuestas exitosas o 400/500 en casos de error).
- Verificación de la lógica de negocio relacionada con la creación, modificación o eliminación de datos.

#### **4.9. Despliegue**

El despliegue del sistema se ha realizado de forma local durante todo el proceso de desarrollo. Tanto el frontend como el backend han sido ejecutados en el entorno local del desarrollador.

El **frontend**, desarrollado con React, ha sido gestionado mediante **Vite**, una herramienta de desarrollo rápida que permite compilar y servir aplicaciones de forma eficiente. Para levantar la interfaz, se ha utilizado el comando `npm run dev`, el cual inicia el servidor de desarrollo de Vite y permite visualizar la aplicación en el navegador.

Por otro lado, el **backend** ha sido ejecutado desde **IntelliJ IDEA**, que ha permitido compilar, ejecutar y depurar la aplicación desarrollada con Spring Boot de manera sencilla. Al tratarse de un entorno local, no se ha hecho uso de servicios de despliegue en la nube como Render, Vercel, Netlify o Heroku.

Para la **gestión de versiones**, se ha utilizado **GitHub** como sistema de control de versiones. GitHub ha permitido mantener el código fuente organizado, llevar un historial de cambios y facilitar la recuperación de versiones anteriores en caso de errores o pérdida de información. Aunque el uso ha sido básico, ha servido también como mecanismo de respaldo, garantizando que el progreso del proyecto estuviera almacenado de forma segura y accesible.

Este entorno de trabajo local ha sido suficiente para el desarrollo y prueba del sistema, si bien, en una versión futura, se podría contemplar la posibilidad de desplegar la aplicación en la nube para facilitar el acceso desde distintos dispositivos o usuarios.

#### **4.10. Retos encontrados y soluciones aplicadas**

Durante el desarrollo del sistema se han presentado varios retos técnicos y de diseño que han requerido investigación, adaptación y mejora sobre el planteamiento inicial.

Uno de los primeros retos fue **cómo obtener información en tiempo real sobre los precios de los activos financieros**. Tras una extensa búsqueda y evaluación de múltiples APIs públicas y privadas, se llegó finalmente a utilizar **Alpha Vantage**, una API fiable, gratuita y suficientemente completa que permite obtener cotizaciones en tiempo real tanto de acciones como de criptomonedas y ETFs. Esta decisión permitió integrar correctamente la funcionalidad sin necesidad de recurrir a servicios de pago o integraciones más complejas.

Otro problema técnico importante fue **cómo integrar un modelo de lenguaje (LLM) en el sistema**. Este desafío fue abordado con la ayuda del tutor, quien recomendó el uso de **Ollama**, un modelo potente, ligero y de fácil integración para entornos locales. Gracias a esta solución, se pudo incorporar una funcionalidad basada en inteligencia artificial sin necesidad de recursos externos ni dependencias complejas.

Respecto a las **mejoras aplicadas sobre el diseño inicial**, una de las más destacadas es la incorporación de un **buscador para consultar precios de activos en tiempo real**. Esta característica no estaba contemplada en las fases tempranas del diseño, pero se añadió tras identificar su valor práctico para el usuario final y su viabilidad técnica dentro del sistema.

Otra mejora relevante fue el cambio en la estructura de la relación entre usuarios y carteras. Inicialmente, se había planteado que cada usuario solo pudiera tener un único portfolio. Sin embargo, tras avanzar en el desarrollo, se decidió **permitir a los usuarios crear y gestionar múltiples portfolios**, lo que ofrece una mayor flexibilidad y personalización dentro de la aplicación. Esta decisión también permitió una arquitectura más escalable y orientada a casos de uso reales.

# CAPÍTULO 5: Conclusiones y trabajos futuros

En este capítulo se realiza un repaso de los objetivos planteados al inicio del proyecto, evaluando en qué medida han sido alcanzados. Se incluye también una reflexión personal sobre el aprendizaje adquirido a lo largo del grado y cómo la realización del TFG ha contribuido a mi desarrollo académico y profesional. Finalmente, se plantean posibles líneas de trabajo futuro que podrían desarrollarse para ampliar o mejorar el sistema propuesto, y que no han podido abordarse por limitaciones de tiempo o recursos.

Este Trabajo de Fin de Grado ha tenido como objetivo general el desarrollo de una plataforma digital capaz de centralizar y monitorear todas las inversiones del usuario en un solo lugar, de manera rápida, sencilla e intuitiva. El proyecto surge como respuesta a la necesidad de simplificar el seguimiento de activos financieros, eliminando la tediosa tarea de acceder a múltiples aplicaciones o páginas web para obtener información actualizada. A lo largo del desarrollo del trabajo, se ha logrado cumplir satisfactoriamente este objetivo general, presentando una solución moderna, funcional y orientada al usuario, con una arquitectura técnica robusta y una documentación completa.

## 5.1. Grado de consecución de los objetivos específicos

En este apartado se realiza un repaso de los objetivos específicos del proyecto y el grado de avance alcanzado en cada uno. Se evalúa el desarrollo de una plataforma digital funcional, el análisis de aplicaciones similares para extraer buenas prácticas, el establecimiento de una estructura clara y escalable para el proyecto, y la implementación de la organización y el monitoreo de activos en tiempo real.

### Objetivo 1: Desarrollar una plataforma digital funcional

**Cumplimiento:** 100%

Se ha implementado una aplicación web que permite a los usuarios visualizar y gestionar sus inversiones desde una única interfaz, facilitando el acceso a información relevante de manera unificada. La plataforma presenta un diseño responsive y una experiencia de usuario intuitiva.

**Evidencia:** Capítulos 4, donde se describen tanto la arquitectura como las funcionalidades implementadas.

### Objetivo 2: Analizar aplicaciones similares para extraer buenas prácticas

**Cumplimiento:** 100%

Se ha realizado un estudio comparativo de cinco aplicaciones del sector financiero, identificando elementos comunes, aspectos positivos y limitaciones. Este análisis ha servido como base para diseñar una plataforma diferenciadora, mejor adaptada a las necesidades reales del usuario.

**Evidencia:** Sección 2.4 de la memoria.

### Objetivo 3: Establecer una estructura clara y escalable para el proyecto

**Cumplimiento:** 100%

Se ha definido una arquitectura técnica basada en Spring Boot para el backend y React para el frontend. El sistema está organizado modularmente, lo cual garantiza su escalabilidad, mantenimiento y posibilidad de evolución futura.

**Evidencia:** Capítulo 4 (Arquitectura y diseño del sistema).

#### **Objetivo 4: Implementar la organización y monitoreo de activos en tiempo real**

**Cumplimiento:** 60%

Se ha logrado implementar de forma completa la organización de activos financieros, permitiendo al usuario registrar, clasificar y gestionar sus inversiones. Sin embargo, no ha sido posible completar la funcionalidad de actualización automática del valor de los activos en tiempo real tras operaciones de compra o venta, debido a limitaciones técnicas y de tiempo.

**Evidencia:** Sección 4.6 Integración con APIs externas

La formación adquirida durante el grado ha sido fundamental para afrontar este proyecto, especialmente en programación, bases de datos y diseño orientado a objetos. Sin embargo, también ha sido necesario enfrentarse a nuevos retos, como aprender a usar tecnologías no vistas previamente en profundidad, como **Spring Boot** para el backend y **React** para el frontend. Estos aprendizajes han implicado comprender la arquitectura cliente-servidor, manejar APIs REST y diseñar una interfaz usable. Además, la redacción de una memoria técnica extensa ha supuesto un reto importante en cuanto a organización, claridad y documentación.

Este TFG me ha permitido participar por primera vez en un proyecto de meses de duración, lo que me ha enseñado a planificar y mantener un ritmo constante. Me siento orgulloso del resultado, tanto de la plataforma como de la memoria elaborada. A lo largo del proceso he mejorado habilidades clave como la organización, la resolución de problemas y la capacidad de adaptación. A pesar de las dificultades, ha sido una experiencia enriquecedora que ha reforzado mi formación técnica y personal.

#### **5.2. Trabajos futuros**

A pesar de que el desarrollo del proyecto ha permitido alcanzar los objetivos principales planteados, hay varias áreas de mejora que podrían abordarse en futuros trabajos. Estas propuestas surgen tanto de funcionalidades que no se han podido implementar por limitaciones de tiempo como de nuevas ideas detectadas durante el proceso de desarrollo.

##### **Integración con brokers reales**

Una funcionalidad clave pendiente es la integración directa con brokers financieros. Esta mejora permitiría que las operaciones de compra y venta se reflejan automáticamente en la plataforma, sin necesidad de introducción manual por parte del usuario. Para llevarla a cabo, sería necesario utilizar APIs ofrecidas por los propios brokers, garantizando una conexión segura mediante mecanismos de autenticación adecuados y sincronización de datos en tiempo real.

##### **Personalización de la interfaz de usuario**

Actualmente, la interfaz está diseñada de forma funcional y genérica, pero en el futuro podría ampliarse para permitir al usuario personalizar distintos aspectos visuales, como el tema de la aplicación (modo claro/oscuro), el orden de visualización de los activos o el diseño de los paneles principales. Esta mejora aumentaría la usabilidad y el grado de satisfacción del usuario final.

### **Implementación de pruebas automatizadas**

Una tarea importante que no ha podido abordarse por cuestiones de tiempo es la creación de pruebas automatizadas. La implementación de pruebas unitarias, de integración y de flujo contribuiría significativamente a la robustez y mantenibilidad del sistema.

### **Cálculo automático de rentabilidad por operación**

Otra mejora funcional interesante sería la inclusión del cálculo automático del porcentaje de ganancia o pérdida tras cada operación. Para ello, sería necesario registrar el precio de adquisición de los activos y comparar este valor con el precio actual del mercado. De este modo, se proporciona al usuario una visión más clara y directa del rendimiento de sus inversiones.

Estas líneas de trabajo suponen una base sólida para dar continuidad al proyecto, bien sea como mejora futura dentro de un entorno profesional, como parte de un TFG posterior o incluso como producto orientado a su comercialización.

# Bibliografía

- [1] BBC NEWS, La lista completa de los aranceles impuestos por Trump a cada país. Disponible en: <https://www.bbc.com/mundo/articles/c89gxwwj4k2o> (Último acceso: junio 2025)
- [2] Trading View, EEUU crea una Reserva Estratégica de Bitcoin: ¡El futuro de las criptomonedas ha llegado!, Disponible en:  
<https://es.tradingview.com/news/newsbtc:a39fc9a3c09cd:0/> (Último acceso: junio 2025)
- [3] Jesús Armando Castillo Torrealba. ¿Cuándo subirán las criptomonedas? -Análisis del mercado de criptomonedas 2023 y perspectivas 2024. Disponible en:  
<https://www.mitrade.com/es/articulo/cripto/analisis-de-cripto/criptomonedas-2023-y-2024>
- [4] MicroBank, ¿Qué es el crowdfunding y cómo funciona?, Disponible en:  
<https://www.microbank.com/es/blog/p/crowdfunding.html> (Último acceso: junio 2025)
- [5] CoinTracking. (s.f.). *CoinTracking - Your Crypto Tax Tool*. Disponible en:  
<https://cointracking.info/es/> (Último acceso: junio 2025)
- [6] The Dividend Tracker. (2025). *The Dividend Tracker - Sitio web oficial*. Disponible en: <https://thedividendtracker.com/> (Último acceso: junio 2025)
- [7] Microsoft, Documentación de C++. Disponible en:  
<https://learn.microsoft.com/es-es/cpp/cpp/?view=msvc-170>
- [8] Microsoft, Documentación del lenguaje C#, Disponible en:  
<https://learn.microsoft.com/es-es/dotnet/csharp/> (Último acceso: junio 2025)
- [9] Manual de PHP, Disponible en: <https://www.php.net/manual/es/index.php>
- [10] Ruby on Rails Guides, Disponible en: <https://guides.rubyonrails.org/>
- [11] Spring Framework. (s.f.). *RestTemplate en Spring*. Disponible en:  
<https://docs.spring.io/spring-framework/reference/integration/rest-clients.html> (Último acceso: mayo 2025)
- [12] Spring Boot. (s.f.). *Spring Boot Web MVC*. Disponible en:  
<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#web> (Último acceso: mayo 2025)
- [13] Documentación de Django, Disponible en: <https://docs.djangoproject.com/es/5.2/> (Último acceso: junio 2025)
- [14] MongoDB, Documentación MongoDB, Disponible en:  
<https://www.mongodb.com/docs/> (Último acceso: junio 2025)
- [15] MariaDB, Documentación MariaDB, Disponible en:  
<https://mariadb.com/kb/en/documentation/> (Último acceso: junio 2025)

- [16] SQLite, Documentación de SQLite, Disponible en: <https://www.sqlite.org/docs.html> (Último acceso: junio 2025)
- [17] MySQL Workbench, Documentación de MySQL Workbench, Disponible en: <https://dev.mysql.com/doc/workbench/en/> (Último acceso: junio 2025)
- [18] acens Technologies, Twig, el motor de plantillas para PHP que separa el código HTML, Disponible en:  
<https://www.acens.com/comunicacion/wp-content/images/2014/06/twig-plantillas-wp-acens.pdf> (Último acceso: junio 2025)
- [19] React, Documentación de React Disponible en: <https://es.react.dev/learn> (Último acceso: junio 2025)
- [20] Angular, Introducción a la Documentación de Angular, Disponible en:  
<https://docs.angular.lat/docs> (Último acceso: junio 2025)
- [21] BingX. (2023). *Best Crypto Portfolio Trackers in 2023*. Disponible en:  
<https://bingx.com/es-es/learn/crypto-portfolio-tracking> (Último acceso: mayo 2025)
- [22] Trustpilot. (s.f.). *Opiniones sobre CoinTracking*. Disponible en:  
<https://es.trustpilot.com/review/cointracking.info> (Último acceso: abril 2025)
- [23] SourceForge. (2025). *The Dividend Tracker Reviews*. Disponible en:  
<https://sourceforge.net/software/product/The-Dividend-Tracker/> (Último acceso: mayo 2025)
- [24] Reddit. (2024). *Does paying for Divtracker feel silly?*. Disponible en:  
[https://www.reddit.com/r/dividends/comments/1adfe4c/does\\_paying\\_for\\_divtracker\\_feel\\_silly/](https://www.reddit.com/r/dividends/comments/1adfe4c/does_paying_for_divtracker_feel_silly/) (Último acceso: mayo 2025)
- [25] Sharesight. (2025). *Sharesight - Stock Portfolio Tracker*. Disponible en:  
<https://www.sharesight.com/> (Último acceso: abril 2025)
- [26] Sharesight. (2020). *7 reasons why Sharesight is better than a spreadsheet*. Disponible en:  
<https://www.sharesight.com/blog/7-reasons-why-sharesight-is-better-than-a-spreadsheet> (Último acceso: mayo 2025)
- [27] Sharesight. (2025). *Sharesight vs. alternatives: Why investors choose our portfolio tracker*. Disponible en: <https://www.sharesight.com/blog/sharesight-vs-alternatives/> (Último acceso: mayo 2025)
- [28] Founderz. (s.f.). *¿Cómo nacieron las criptomonedas?*. Disponible en:  
<https://founderz.com/es/blog/como-nacieron-criptomonedas/> (Último acceso: mayo 2025)
- [29] Chaum, D. (1983). *Blind signatures for untraceable payments*. In *Advances in Cryptology* (pp. 199–203). Springer. Disponible en:  
[https://doi.org/10.1007/978-1-4757-0602-4\\_18](https://doi.org/10.1007/978-1-4757-0602-4_18) (Último acceso: junio 2025)
- [30] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Disponible en: <https://bitcoin.org/bitcoin.pdf> (Último acceso: junio 2025)

- [31] Tapscott, D., & Tapscott, A. (2016). *Blockchain revolution: How the technology behind bitcoin is changing money, business, and the world*. Penguin. (Último acceso: junio 2025)
- [32] Buterin, V. (2021). *EIP-1559: Fee market change for ETH 1.0 chain*. Ethereum Improvement Proposals. Disponible en: <https://eips.ethereum.org/EIPS/eip-1559> (Último acceso: junio 2025)
- [33] Binance. (s.f.). *BNB Burn*. Disponible en: <https://www.binance.com/en/bnb> (Último acceso: junio 2025)
- [34] Solana Foundation. (s.f.). *What is Solana?*. Disponible en: <https://solana.com/> (Último acceso: junio 2025)
- [35] Investopedia. (s.f.). *What is Bitcoin Halving?*. Disponible en: <https://www.investopedia.com/bitcoin-halving-4843769> (Último acceso: junio 2025)
- [36] Ethereum Foundation. (2022). *Ethereum Proof-of-Stake (PoS) Explained*. Disponible en: <https://ethereum.org/en/upgrades/merge/> (Último acceso: junio 2025)
- [37] Ollama, Documentacion de Ollama, Disponible en: <https://github.com/ollama/ollama/blob/main/README.md> (Último acceso: junio 2025)
- [38] Alpha Vantage. (s.f.). *Documentación de la API*. Disponible en: <https://www.alphavantage.co/documentation/> (Último acceso: mayo 2025)
- [39] Postman, Documentación de Postman, Disponible en: <https://learning.postman.com/> (Último acceso: junio 2025)
- [40] JetBrains, Documentación de IntelliJ, Disponible en: <https://www.jetbrains.com/help/idea/getting-started.html> (Último acceso: junio 2025)
- [41] Vite, Documentación de Vite, Disponible en: <https://vite.dev/guide/> (Último acceso: junio 2025)
- [42] Node Package Manager, Documentación de NPM, Disponible en: <https://www.npmjs.com/package/documentation> (Último acceso: junio 2025)
- [43] Alpha Vantage. (s.f.). *Sitio web oficial*. Disponible en: <https://www.alphavantage.co/> (Último acceso: abril 2025)
- [44] Meta, Llama 3.2: revolucionando la IA y la visión de vanguardia con modelos abiertos y personalizables, Disponible en: <https://about.fb.com/ltam/news/2024/09/llama-3-2-revolucionando-la-ia-y-la-vision-de-vanguardia-con-modelos-abiertos-y-personalizables/> (Último acceso: junio 2025)

