DESARROLLO DE UNA APLICACIÓN WEB PARA EL BACKTESTING DE ESTRATEGIAS DE TRADING

DEVELOPMENT OF A WEB APPLICATION FOR THE BACKTESTING OF TRADING STRATEGIES

TRABAJO DE FIN DE GRADO

CURSO 2021-2022

AUTOR

SALVADOR FIGUEROS SENTANA

DIRECTOR

ANTONIO SARASA CABEZUELO

DOBLE GRADO ADE – INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

DESARROLLO DE UNA APLICACIÓN WEB PARA EL BACKTESTING DE ESTRATEGIAS DE TRADING

DEVELOPMENT OF A WEB APPLICATION FOR THE BACKTESTING OF TRADING STRATEGIES

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTOR

SALVADOR FIGUEROS SENTANA

DIRECTOR

ANTONIO SARASA CABEZUELO

CONVOCATORIA: JUNIO 2022

DOBLE GRADO ADE – INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

RESUMEN

El trabajo aquí presentado consiste en la especificación, diseño e implementación de una aplicación web que permita la ejecución del "backtesting" de estrategias de trading. El usuario va a poder ejecutar, a través de esta aplicación, el "backtesting" de una estrategia basada en la compra o en la venta de un activo financiero en el máximo o en el mínimo de su precio para un período temporal dado. Los datos de los precios de los activos financieros son extraídos de Yahoo! Finance y se ha seguido un diseño guiado por eventos para la implementación del simulador.

La aplicación también proporciona la infraestructura necesaria para llevar a cabo la gestión de carteras. En concreto, permite la creación de carteras financieras y la optimización de estas en base al método del análisis de la varianza media. Además, también se han desarrollado otra serie de servicios con una gran aplicación práctica en la operativa del trader. Finalmente, la aplicación permite la compartición de información entre los usuarios registrados, favoreciendo el intercambio de ideas y la creación de una comunidad de gente interesada en las finanzas cuantitativas.

Palabras clave

backtesting, estrategia, trading, trader, carteras, Python, finanzas cuantitativas

ABSTRACT

The work here presented consists in the specification, design and implementation of a web application that allows the execution of the backtesting of trading strategies. The user of this web application is going to be able to execute the "backtesting" of a strategy based on buying or selling an asset at its highs or lows of a given time frame. The price data for the assets is pulled out from Yahoo! Finance and the backtester has been built by programming an Event-Driven system.

The application also provides the necessary infrastructure for portfolio management. Specifically, it allows the creation of a portfolio and its optimization based on the mean-variance analysis. Moreover, several other trading services have also been developed which could prove to be of great use for the trader. Finally, the application allows the users to share information with each other and thus presents an opportunity for the creation of a community of people interested in quantitative finance.

Keywords

backtesting, strategy, trading, trader, portfolios, Python, quantitative finance

ÍNDICE DE CONTENIDOS

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1 Motivación

Una estrategia de trading es el plan de acción que se sigue con el fin de obtener un beneficio operando en los mercados financieros. Por lo tanto, una estrategia de trading es un sistema formado por un conjunto de reglas definidas previamente que indican qué activos financieros comprar o vender, cuándo comprar o vender dichos activos financieros, qué porcentaje del capital destinar a la correspondiente compra o venta y cómo gestionar el riesgo tanto de una operación como de la cuenta en total.

Antes de ejecutar una estrategia de trading en vivo y en real, conviene primero valorar la viabilidad de esta. Ahí es donde entra el juego el "backtesting". El "backtesting" es el método por el cual se evalúa cómo habría funcionado una estrategia de trading sobre un conjunto de datos históricos. Por lo tanto, ejecutar el "backtesting" de una estrategia de trading consiste en realizar una simulación de dicha estrategia utilizando datos pasados.

Ahora, muchos son los inversores que no tienen conocimientos de programación. Si bien la ejecución de un "backtesting" sobre un período de tiempo corto es algo que puede hacer cualquier persona de forma manual, esta implementación resulta costosa, por no decir imposible, cuando se trata de un período temporal más largo. Estos inversores quedan privados entonces del estudio pasado de las estrategias que quieren implementar.

Para dar solución a tal problema, este proyecto plantea el desarrollo de una aplicación web que ofrezca las funcionalidades necesarias para la ejecución del "backtesting" de una forma cómoda e intuitiva. Para ello, se pretende implementar una estrategia de trading basada en comprar o vender un activo financiero en sus máximos o mínimos para un determinado marco temporal.

1.2 Objetivos

El objetivo de este trabajo consiste en el desarrollo de una aplicación web que permita al usuario realizar el "backtesting" de una estrategia de trading sobre un conjunto de datos históricos.

Este objetivo se puede dividir en los siguientes objetivos específicos:

- Desarrollar una aplicación web que permita al usuario configurar la cartera, elegir los activos financieros y parametrizar la estrategia de trading sobre los que correr el "backtesting".
- Implementar una funcionalidad que permita al usuario gestionar carteras financieras.
- Implementar una funcionalidad que permita al usuario subir al sistema los datos históricos de los precios de los activos financieros que este quiera.
- Desarrollar una aplicación web que permita la gestión, colaboración y compartición de información ("backtestings" ejecutados y carteras financieras) entre los usuarios suscritos.
    - Implementar una funcionalidad que permita la gestión de cuentas de usuario.
    - Implementar una funcionalidad que permita a un usuario registrado guardar y compartir los "backtesting" ejecutados.
    - Implementar una funcionalidad que permita a un usuario registrado compartir una cartera financiera.
- Implementar otras funcionalidades que sirvan de ayuda a un trader.
    - Implementar una funcionalidad que permita al usuario llevar un diario de trading.
    - Implementar una funcionalidad que permita al usuario realizar un "backtesting" de forma manual.
- Diseñar una API que sea intuitiva, minimalista y al mismo tiempo contenga una cantidad de información lo suficientemente exhaustiva.

1.3 Plan de trabajo

Para el cumplimiento de los objetivos específicos enumerados en el anterior apartado, se han seguido una serie de etapas. A continuación, se explican cuáles han sido estas etapas y se muestra el Diagrama de Gantt correspondiente a cada una de ellas.

Etapa 1: Especificación de Requisitos

Esta etapa inicial ha consistido en la discusión y especificación de los casos de uso que iba a tener la aplicación web a desarrollar.

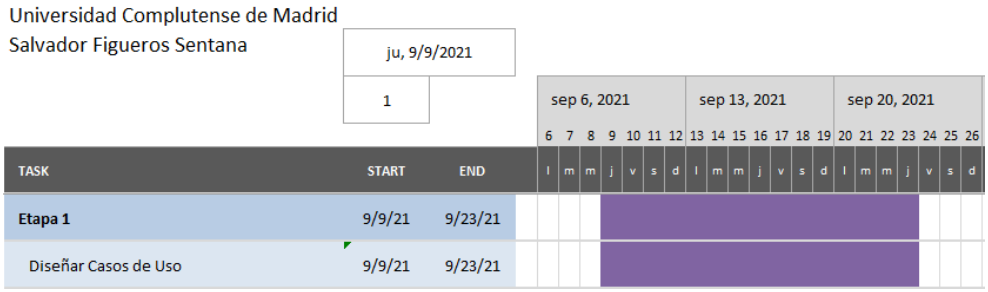


Ilustración 1. Diagrama de Gantt Etapa 1

Etapa 2: Creación de la Aplicación Web e Implementación de las Cuentas de Usuario

En esta segunda etapa se ha desarrollado el código necesario para la creación de la aplicación web. Una vez hecho esto, se ha implementado la gestión de las cuentas de usuario y, después, la funcionalidad relacionada con la gestión de los datos históricos de los precios de los activos.

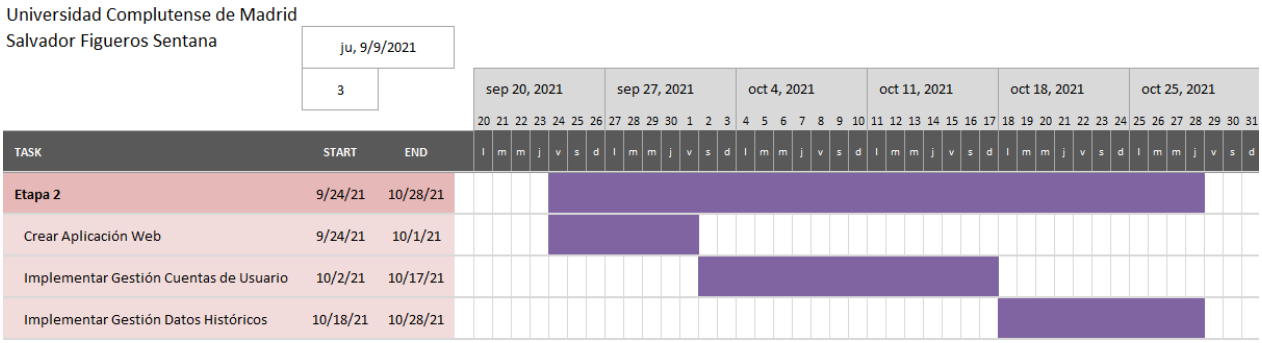


Ilustración 2. Diagrama de Gantt Etapa 2

Etapa 3: Implementación de la Gestión de Carteras

En esta etapa se ha realizado la implementación de la gestión de carteras. En concreto, se han desarrollado todas las funcionalidades necesarias para ello: crear, ver, modificar, publicar y eliminar una cartera.

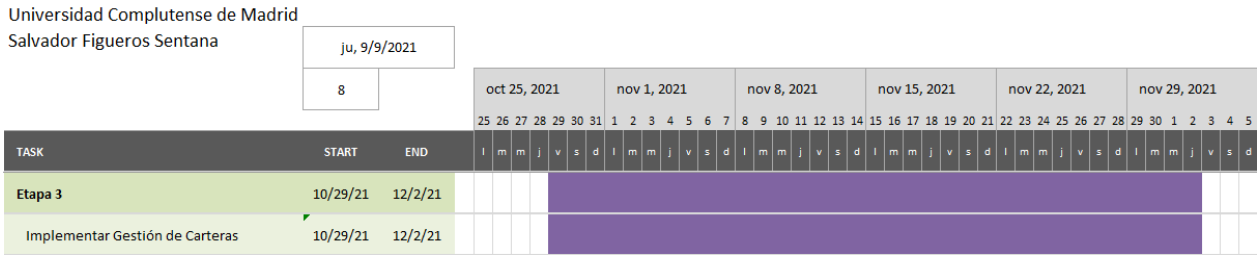


Ilustración 3. Diagrama de Gantt Etapa 3

Etapa 4: Implementación del "Backtesting"

En esta etapa se ha realizado la implementación de la funcionalidad principal de la aplicación: "Backtesting". Para ello, en primer lugar, se ha realizado el desarrollo del código necesario para le ejecución del "backtesting" de la estrategia de trading aquí implementada. Una vez hecho esto, se ha desarrollado la infraestructura necesaria para su gestión. Aquí entra todo lo que tiene que ver con guardar, ver, publicar y eliminar un "backtesting" creado.

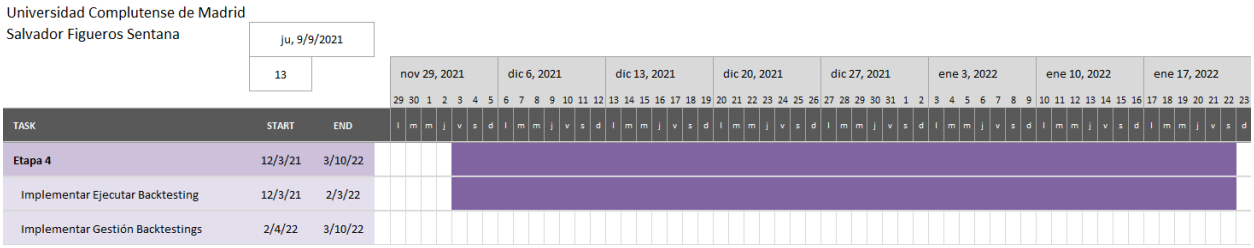


Ilustración 4. Diagrama de Gantt Etapa 4.1

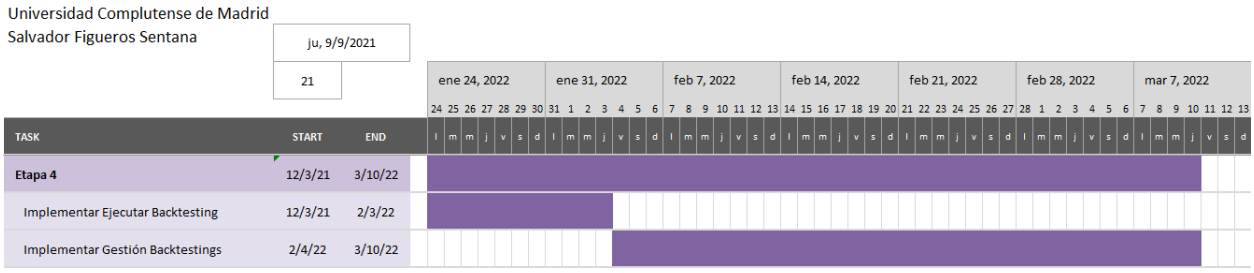


Ilustración 5. Diagrama de Gantt Etapa 4.2

Etapa 5: Implementación de Otros Servicios de Trading

En primer lugar, en esta etapa se ha implementado el Diario de Trading. Después, se ha desarrollado la funcionalidad relacionada con el "Backtesting" Manual.

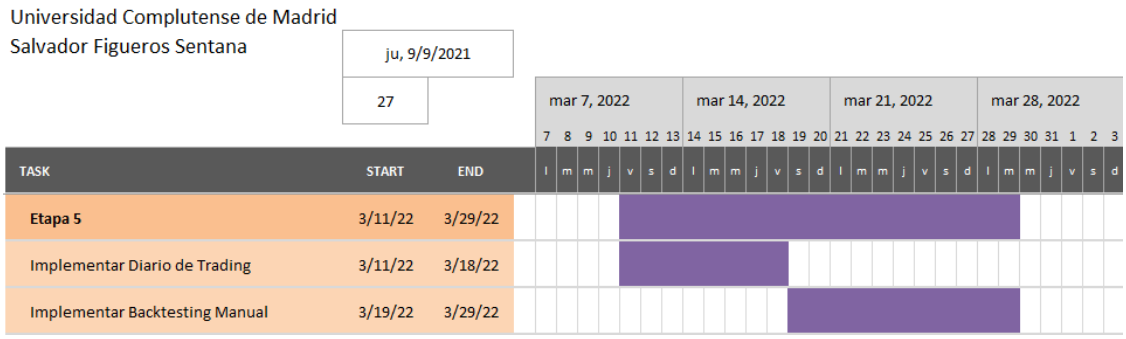


Ilustración 6. Diagrama de Gantt Etapa 5

Etapa 6: Desarrollo de la Interfaz Visual

Esta etapa ha consistido en el desarrollo de la API de la aplicación web.

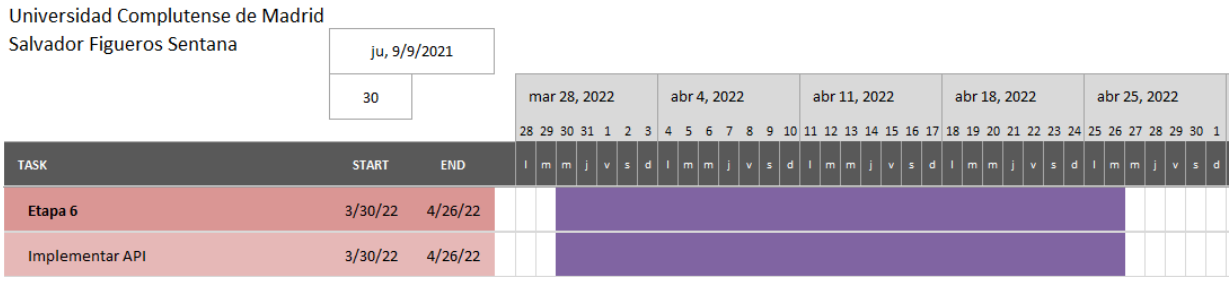


Ilustración 7. Diagrama de Gantt Etapa 6

1.4 Estructura de la memoria

A continuación, se describe de forma breve los principales capítulos en los que se divide la memoria:

- Capítulo 1: Este capítulo está dedicado a la motivación del trabajo, los objetivos, el plan de trabajo seguido y la estructura de la memoria.

- Capítulo 2: Este capítulo está dedicado a la explicación de la estrategia de trading implementada.
- Capítulo 3: Este capítulo está dedicado a la descripción de aplicaciones que ofrecen funcionalidades similares a las implementadas en este proyecto.
- Capítulo 4: Este capítulo está dedicado a presentar las tecnologías que se han utilizado en la realización del proyecto.
- Capítulo 5: Este capítulo está dedicado a la descripción de los actores y la especificación de requisitos realizada.
- Capítulo 6: En este capítulo se hace una descripción de la arquitectura que se ha seguido para el desarrollo e implementación de la aplicación web.
- Capítulo 7: En este capítulo se explica el modelo de datos que se ha definido para realizar la persistencia de la información gestionada por la aplicación web.
- Capítulo 8: En este capítulo se va a realizar una descripción de las principales funcionalidades implementadas y el diseño que se ha seguido para cada una de ellas.
- Capítulo 9: Este capítulo está dedicado a las conclusiones del trabajo y las líneas de trabajo planteadas de cara al futuro.
- Bibliografía: Esta sección contiene un listado con todas las fuentes bibliográficas utilizadas en la realización del trabajo.
- Apéndices: Manual de usuario

Chapter - Introduction

Motivation

A trading strategy is the action plan one follows with the intention of being profitable in trading the financial markets. Therefore, a trading strategy is a system made up of a group of rules previously defined which signal which assets to buy or sell, when to buy or sell those assets, which percentage of the capital to allocate to those buys or sells, and how to manage the risk associated with one trade and with the portfolio overall.

Before applying a trading strategy in a live environment, it is necessary to assess the profitability of the strategy. That is where the backtesting comes into play. Backtesting a trading strategy is the method by which it is possible to evaluate how a trading strategy would have behaved over a set of historical data. Therefore, running the backtesting of a trading strategy is like running a simulation using past data.

Now, there are a lot of investors who do not know how to program a backtesting. Although executing a backtesting by hand over a short period of time is something anybody can do, this task becomes difficult, if not impossible, when the length of the period is increased.

To address this problem, this project focuses on the development of a web application that provides necessary features to run a backtesting. To that end, the application is going to use a trading strategy based on buying or selling at its highs or lows of a given time frame.

Objectives

The main objective of this project is the development of a web application that allows a user to run the backtesting of a trading strategy over a set of historical data.

This objective can be divided among the following specific objectives:

- Develop a web application that allows a user to design the portfolio, choose the assets, and change the parameters of the strategy on which the backtesting is going to be executed.
- Develop a functionality for portfolio management.
- Develop a functionality for uploading historical data to the system.
- Develop a web application where users can collaborate and share information (backtesting information and portfolios) with each other.
    - Develop a functionality for managing user accounts.

    - Develop a functionality for saving and sharing backtesting information.
    - Develop a functionality for sharing portfolios.
- Develop other functionalities for helping a user to trade the financial markets.
    - Create a trading journal.
    - Develop a functionality for backtesting by hand.
- Design an intuitive, minimalist, and at the same time thorough visual interface.

Work plan

In order to satisfy all the specific objectives laid out in the previous section, the work plan has been divided into a series of phases. Next is a description of all these phases with their corresponding Gantt Chart.

Phase 1: Specification of Requirements

This initial phase has consisted in the discussion and specification of the use cases which have been implemented during the development of the web application.

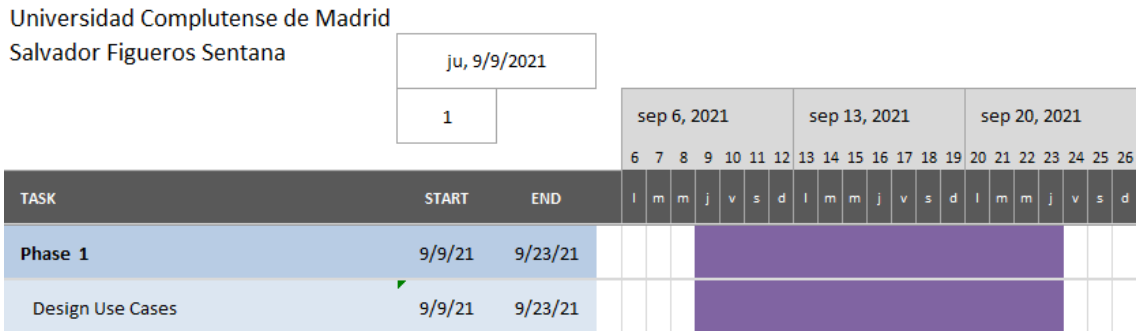


Ilustración 8. Gantt Chart Phase 1

Phase 2: Creation of the Web App and Implementation of User Accounts

This second phase has been devoted to the development of the necessary code for the creation of the web application. The next steps have been the implementation of the user accounts management and then, the functionality related to the handling of the historical price data.

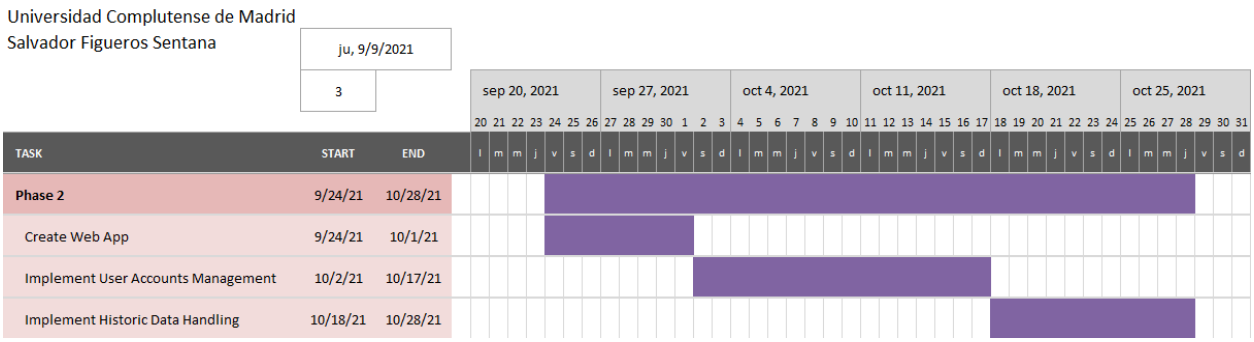


Ilustración 9. Gantt Chart Phase 2

Phase 3: Implementation of Portfolio Management

This phase has been devoted to the implementation of portfolio management. In particular, the following functionalities have been developed: create, view, modify, share and delete a portfolio.

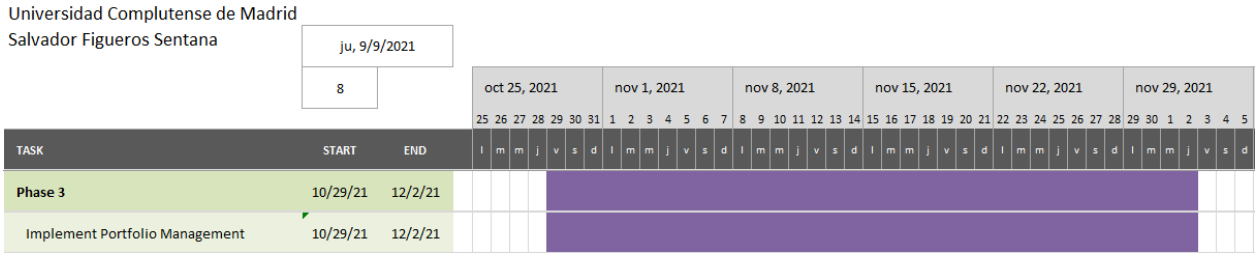


Ilustración 10. Gantt Chart Phase 3

Phase 4: Implementation of Backtesting

This phase has been focused on the implementation of the main functionality of the application: the backtesting. For that, the first thing that has been done has been the development of the code necessary for the execution of the backtesting of the trading strategy here implemented. The next step has been the development of the infrastructure necessary for the management of the backtestings. This has included saving, sharing, and deleting a previously created backtesting.

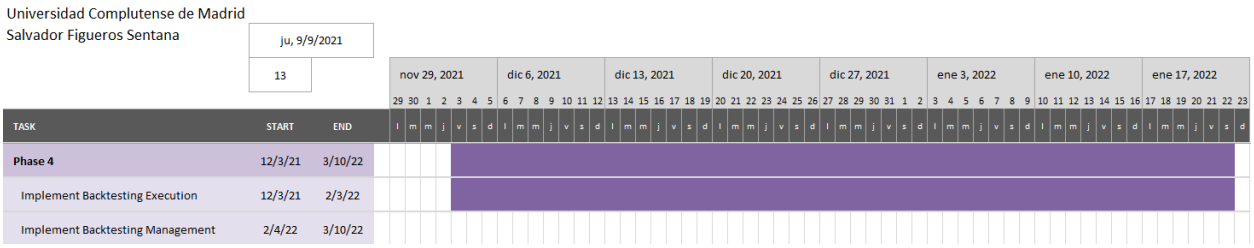


Ilustración 11. Gantt Chart Phase 4.1

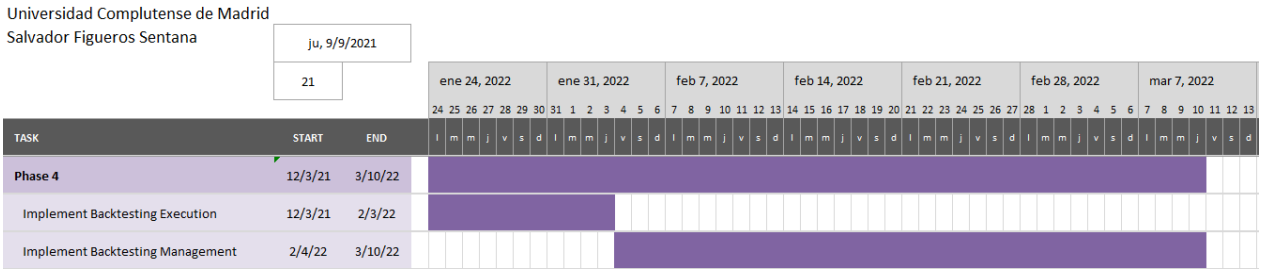


Ilustración 12. Gantt Chart Phase 4.2

Phase 5: Implementation of Other Trading Services

First, this phase has been devoted to the implementation of the Trading Journal. Once this was finished, the next step has been the implementation of the backtesting by hand functionality.

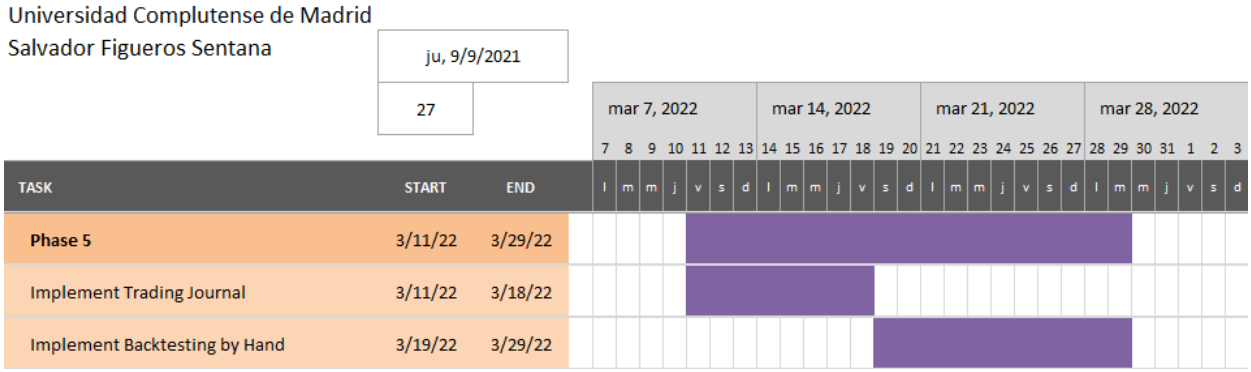


Ilustración 13. Gantt Chart Phase 5

Phase 6: Development of the Visual Interface

This phase has consisted in the development of the API for the web application.

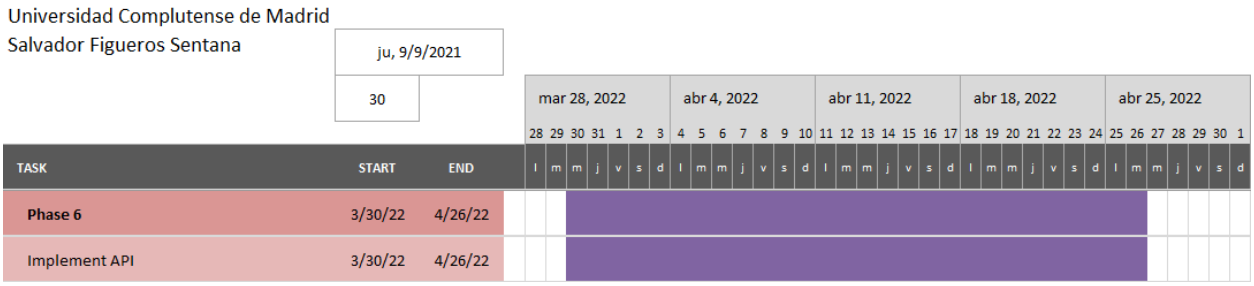


Ilustración 14. Gantt Chart Phase 6

Structure of the memory

Th structure of the memory is briefly described below:

- Chapter 1: This chapter is devoted to the motivation behind this work, the objectives, the work plan, and the structure of the memory.
- Chapter 2: This chapter is devoted to the explanation of the trading strategy here implemented.
- Chapter 3: This chapter describes the applications that offer similar functionalities to the ones here implemented.
- Chapter 4: This chapter describes the technology that has been used throughout the development of the project.
- Chapter 5: This chapter describes the actors of the system and their use cases.
- Chapter 6: This chapter is devoted to describing the architecture which has been followed to build the web application.
- Chapter 7: This chapter explains the data model which has been defined to store the data handled by the web application.
- Chapter 8: This chapter describes the main functionalities which have been implemented and the design which has been followed to do so.

- Chapter 9: This chapter is devoted to the conclusions of the project and the lines of work laid out for the future.
- Bibliography: This section includes a list with all the sources used throughout the development of the project.
- Appendix: User manual

Capítulo 2 - Estrategia de Trading

Este capítulo está dedicado a la explicación de la estrategia de trading que se pretende implementar en esta aplicación web.

2.1 Idea general

La estrategia de trading aquí planteada es una estrategia basada en comprar o vender un activo financiero en sus máximos o mínimos para un determinado marco temporal. En realidad, se trata de 4 casos particulares de una estrategia general basada en comprar o vender en momentos extremos de precio dentro de un marco temporal determinado. Cada uno de estos 4 casos particulares viene dado por los 4 tipos de entradas distintas.

2.2 Entradas

A continuación, se definen los 4 tipos de entradas. Cada una de ellas conformaría una sub-estrategia en sí misma.

- Ponerse **largo** (comprar) en el **máximo** de la cotización de un activo financiero en el marco temporal de referencia.
- Ponerse **corto** (vender) en el **máximo** de la cotización de un activo financiero en el marco temporal de referencia.
- Ponerse **largo** (comprar) en el **mínimo** de la cotización de un activo financiero en el marco temporal de referencia.
- Ponerse **corto** (vender) en el **mínimo** de la cotización de un activo financiero en el marco temporal de referencia.

Para cada una de estas entradas, se va a permitir realizar el correspondiente "backtesting" con los máximos y los mínimos de los siguientes marcos temporales de referencia: 5 días, 10 días, 15 días, 1 mes, 3 meses, 6 meses y 1 año.

2.3 Salidas

El usuario va a poder elegir dos tipos de salidas distintas:

- **Salidas con stop temporal** (salir una vez transcurrido un cierto período de tiempo desde que se realizó la entrada). Se va a permitir la implementación de esta salida de tres formas distintas:

- Primer Stop Temporal: 50% de los días del marco temporal de referencia.
- Segundo Stop Temporal: 100% de los días del marco temporal de referencia.
- Tercer Stop Temporal: 150% de los días del marco temporal de referencia.

- **Salidas con trailing stop** (actualizar el stop conforme el precio se va moviendo). Se va a permitir realizar el backtesting de esta estrategia con una salida basada en un trailing stop definido por el indicador ATR (Average True Range: indicador de volatilidad) [1]. Y se pueden utilizar hasta tres multiplicadores del ATR diferentes: 1, 2 y 3.

  Por ejemplo, para una cotización de un activo financiero de 100 y un ATR de ese activo en los últimos 10 días de 3, es posible ejecutar la estrategia utilizando un trailing stop de 97 (100 - (3 x 1)), de 94 (100 - (3 x 2)) o de 91 (100 - (3 x 3)).

2.4 Activos financieros

El "backtesting" de la estrategia de trading comentada en los puntos anteriores se podrá realizar con acciones y futuros.

Capítulo 3 - Estado del arte

Este capítulo está dedicado a la descripción de aplicaciones que ofrecen funcionalidades similares a las implementadas en este proyecto. Concretamente se van a revisar algunas de las principales aplicaciones que implementan "backtesting":

- **MetaTrader 5** [2]**:** MetaTrader 5 es la plataforma de trading en Fórex más popular, si bien permite en general la realización de operaciones comerciales en Fórex, la bolsa de valores y futuros. MetaTrader's Trading Strategy Tester es el nombre que tiene la funcionalidad que permite la ejecución del "backtesting" de una estrategia de trading antes de su implementación en trading en vivo.

  Entre sus características más relevantes, destacan las siguientes:

  - Reporte de los resultados detallado.
  - Resultados gráficos de la simulación.
  - Modo de simulación visual que permite el seguimiento en tiempo real de la ejecución de la estrategia.
  - No proporciona la posibilidad de incluir el análisis fundamental en la configuración de la estrategia.
  - Optimización de los parámetros de la estrategia.

- **MetaStock** [3]**:** MetaStock es uno de los softwares de "backtesting" más adecuados para aquellos usuarios principiantes o intermedios. Cuando se ejecuta la aplicación de MetaStock, desde la consola que aparece se puede seleccionar "System Test". Esta opción permite ejecutar el "backtesting" de hasta 58 sistemas de trading distintos.

  Entre sus características más relevantes, destacan las siguientes:

  - Reporte de los resultados detallado.
  - Resultados gráficos de la simulación.
  - Ejecución del "backtesting" a un solo click.
  - Funcionalidad de predicción.
  - No permite la ejecución automática de operaciones de trading en vivo.
  - Escaneo y filtración de activos financieros basado en fundamentales.

- **Interactive Brokers** [4]**:** Esta empresa multinacional es uno de los bróker más grandes e importantes de todo el mundo y el más importante a nivel particular. Cuenta con una potente plataforma conocida como Trader Workstation (TWS). De entre todas las funcionalidades ofrecidas por este software, la herramienta con el nombre de "Porfolio Manager" permite a los usuarios la gestión de carteras. De esta forma, los clientes de esta compañía pueden configurar una cartera sobre la que correr un "backtesting" basado en los principales fundamentales. Esto le hace un software de "backtesting" distinto al resto del mercado.

  Entre las características más relevantes, destacan las siguientes:

  - "Backtesting" basado en al análisis fundamental.
  - "Backtesting" sobre indicadores técnicos limitado.
  - Acceso directo al mercado.
  - Datos de todos los mercados y activos financieros.

- **TrendSpider** [5]**:** TrendSpider es una plataforma de trading programada enteramente en HTML 5. A diferencia del resto de aplicaciones, esta está programada para detectar automáticamente líneas de tendencia y patrones de Fibonacci, dibujando aquellas que sean más probables. Además, TrendSpider también cuenta con un testador de estrategias que permite a los usuarios que la utilizan la posibilidad de escribir el código de la estrategia y su posterior ejecución.

  Entre sus características más relevantes, destacan las siguientes:

  - Detección automática de líneas de tendencia y de Fibonacci.
  - Acceso a la plataforma únicamente desde la aplicación web.
  - Ejecución del "backtesting" a un solo click.
  - No permite la ejecución automática de operaciones de trading en vivo.

- **TradingView** [6]**:** Esta aplicación web (TradingView también cuenta con una aplicación de escritorio) es una de las plataformas de trading más populares a nivel mundial. De hecho, TradingView cuenta con una de las comunidades sociales de trading más importantes en todo el mundo. Lo que significa que, aparte de permitir a sus usuarios ejecutar el "backtesting" de una estrategia de trading, también permite

la colaboración entre todos ellos mediante la distribución y publicación de ideas de trading.

Entre sus características más relevantes, destacan las siguientes:

- Opción de pago que permite la ejecución de estrategias de trading más sofisticadas.
- Reporte de los resultados detallado.
- Resultados gráficos.
- "Backtesting" Manual.
- Acceso al "Backtesting" desde la plataforma web.
- No proporciona la posibilidad de incluir el análisis fundamental en la configuración de la estrategia.

- **NinjaTrader** [7]**:** Este software de operaciones y plataforma de corretaje ha sido galardonada como una de las mejores plataformas de trading. Así que, como no podía ser de otra forma, Ninja Trader 8 también ofrece una de las mejores funcionalidades de "backtesting" de todo el mercado. El Analizador de estrategias de Ninja Trader está basado en C# y proporciona una solución muy sólida a la hora de simular, optimizar y analizar el rendimiento de las estrategias.

  Entre sus características más relevantes, destacan las siguientes:

  - Reporte de los resultados detallado.
  - Resultados gráficos de la simulación.
  - Número muy grande de indicadores técnicos que utilizar en la definición de la estrategia.

Capítulo 4 - Tecnología empleada

Este capítulo está dedicado a presentar las tecnologías que se han utilizado para la implementación de la aplicación del proyecto.

4.1 Herramientas Frontend

4.1.1 Bootstrap 5

Bootstrap 5 [8] es un framework totalmente gratuito que se usa para el desarrollo de aplicaciones web. Incluye HTML, CSS y Javascript y hoy en día es el framework de front-end más popular a nivel mundial. Permite la creación de aplicaciones web de forma flexible y reactiva sin muchas complicaciones ni esfuerzos.

En este proyecto, Bootstrap 5 se ha utilizado para el desarrollo de la interfaz visual de la aplicación web, haciendo uso de los componentes ofrecidos por este framework tales como formularios, botones, flashcards, barras de navegación o acordeones.

4.1.2 Jinja2

Jinja2 [9] es un motor de templates para Python incorporado en el framework Flask. Un template es un archivo de texto que contiene una serie de variables y expresiones que JInja2 reemplaza por las variables pasadas a través de los métodos incorporados en el framework Flask.

En este proyecto, Jinja2 se ha utilizado para la separación de la lógica del sistema de la presentación de este. Las posibilidades que ofrece esta herramienta son muchas y ello ha permitido la implementación de una aplicación web de una forma más limpia y ordenada.

4.1.3 jQuery

jQuery [10] es una librería de JavaScript que permite la manipulación de elementos de HTML, la gestión de eventos, la animación y el uso de Ajax de una forma muy sencilla.

En este proyecto, el principal uso de jQuery ha sido el de poder utilizar JavaScript de una forma mucho más fácil. Además, se ha utilizado también para aquellos aspectos de la interfaz visual a los que el uso de Bootstrap 5 no alcanzaba.

4.2 Herramientas Backend

4.2.1 Python

Python [11] es un lenguaje de programación interpretado y de alto nivel cuya principal filosofía radica en su fácil legibilidad. Es un lenguaje multiparadigma, dinámico y multiplataforma. Actualmente, cuenta con una comunidad muy grande y un número muy alto de librerías que hacen de su uso una elección muy deseable.

En este proyecto, Python se ha utilizado como lenguaje de programación de la lógica del programa. En concreto, se elegido el uso de Python como lenguaje de programación para la lógica del sistema por la posibilidad que ofrece este de utilizar las librerías de análisis de datos Pandas y Numpy.

4.2.2 Flask

Flask [12] es un microframework para Python que permite la creación de aplicaciones web de una forma bastante rápida. El concepto de microframework hace referencia a marcos de aplicaciones minimalistas que, a diferencia de los marcos de aplicación completos ("macroframework"), suelen enfocarse exclusivamente en el manejo de solicitudes HTTP y dejan de lado otros aspectos más complejos.

En este proyecto, Flask se ha utilizado para el desarrollo de la aplicación web bajo el patrón Modelo-Vista-Controlador.

4.2.3 MySQL

MySQL [13] es un sistema de gestión de bases de datos relacional de código abierto. A diferencia de otros proyectos con funciones parecidas, MySQL está desarrollada por una empresa privada, Oracle Corporation, la cual tiene los derechos de la mayor parte del código. Además, MySQL utiliza también el modelo cliente-servidor. Los datos residen en la parte del servidor y el cliente envía peticiones a este para aquellos datos que el cliente necesita.

En este proyecto, MySQL se ha utilizado para la gestión de la base de datos de la aplicación web. Por ejemplo, los datos de los usuarios o los precios de los activos financieros han sido guardados en el servidor de este modelo para su posterior utilización y consulta durante la ejecución de la aplicación.

4.3 Otras herramientas

4.3.1 Visual Studio Code

Visual Studio Code [14] es un editor de código desarrollado para Microsoft, Linux, macOS y Web. Incluye soporte para la depuración, ejecución de tareas, resaltado de sintaxis, finalización inteligente de código y control de versiones.

El desarrollo de los archivos de texto que componen la aplicación web se ha hecho en su totalidad a través de esta herramienta. En concreto, este IDE se ha utilizado para el desarrollo de archivos con extensión ".py", ".html", ".css" y ".js".

4.3.2 XAMPP

XAMPP [15] es una distribución de Apache que consiste básicamente en el sistema de gestión de base de datos MySQL, el servidor Apache y los intérpretes para lenguajes de script PHP y Perl. Es gratuita y fácil de instalar. De hecho, una de sus principales virtudes reside en la facilidad de instalación y uso que brinda dicho paquete. Está disponible en Windows, Linux y macOS.

En este proyecto, XAMPP se ha utilizado para la gestión de la base de base de datos a través de MySQL. Se ha hecho uso de esta herramienta para diseñar, configurar, crear y modificar las varias tablas y relaciones que componen la base de datos del sistema.

4.3.3 Git

Git [16] es un software gratuito y de código abierto que sirve para el control de versiones en el desarrollo de proyectos tanto grandes como pequeños. Está pensado para el mantenimiento de versiones de una forma rápida y eficiente. Hoy en día, este software de control de versiones es el más utilizado del mundo.

Por lo tanto, esta herramienta ha sido utilizada con la finalidad de llevar a cabo una coordinación y mantenimiento de las versiones de una forma confiable, eficiente y rápida a lo largo de todo el proyecto.

Capítulo 5 - Casos de uso

Este capítulo está dedicado a la descripción de los actores y la especificación de requisitos realizada.

5.1 Actores

A continuación, se enumeran cuáles son los principales actores del sistema y se realiza la descripción correspondiente de cada uno de ellos:

- **Usuario registrado:** Aquellos usuarios que se registren en la aplicación podrán disponer de una serie de privilegios frente aquellos usuarios que no lo hagan. Podrán realizar todos los tipos de "backtesting" y simulaciones, disponer de carteras de activos financieros y disfrutar de otras funcionalidades como el diario de trading y el "backtesting" manual. Además, podrán compartir contenido ("backtesting" y carteras) con el resto de los usuarios y ver el contenido compartido por el resto de los usuarios igualmente.
- **Usuario no registrado:** este tipo de usuario podrá navegar por la aplicación y utilizar alguna de sus funcionalidades, pero no podrá disfrutar de todo el repertorio ofrecido por la aplicación. En concreto, este tipo de usuario podrá realizar "backtesting" y búsquedas, pero no podrá compartir contenido, configurar carteras de activos financieros y disfrutar de otras funcionalidades como el diario de trading.
- **Administrador:** este usuario se dedicará a la gestión de las cuentas de usuario (dar de baja cuentas de usuario y modificar las cuentas de usuario). Además, podrá administrar los datos almacenados en la aplicación (5.2.5 Administrar datos).

5.2 Especificación de requisitos

En esta sección se van a explicar los casos de uso definidos para la aplicación. Para ello los casos de uso se han agrupado modularmente de la siguiente forma:

1. Cuentas de usuario
2. Backtesting
3. Gestión de carteras
4. Otros servicios de trading
5. Administración de datos históricos

5.2.1 Cuentas de usuario

Requisito	**Registrar usuario**	
Identificador	1.1	
Prioridad	Alta	
Precondición	NA	
Descripción	Los usuarios pueden crearse una cuenta en esta aplicación.	
Entrada	Nombre, Apellidos, Nombre de Usuario, Contraseña, Correo Electrónico	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace click en "Registrarse".
	2	El sistema muestra una pantalla en la que se le pide al usuario toda la información necesaria para registrarse.
	3	El usuario rellena todos los campos y hace click en "Crear Cuenta".
	4	El sistema valida la información.
	5	El sistema muestra de nuevo la pantalla principal.
Postcondición	Se ha creado la cuenta de usuario.	
Excepciones	Paso	Acción
	3	Se muestra el mensaje correspondiente si la contraseña no cumple con los requisitos.
	3	Se muestra el mensaje correspondiente si el correo electrónico no es correcto.
	3	Se muestra el mensaje correspondiente si el correo electrónico o el nombre de usuario ya existen.
Comentarios	NA	
Actores	Usuario no registrado, Admin	

Tabla 1. Registrar usuario

Requisito	**Login**	
Identificador	1.2	
Prioridad	Alta	
Precondición	El usuario ha de estar registrado.	
Descripción	Los usuarios registrados pueden iniciar sesión.	
Entrada	Nombre de Usuario, Correo Electrónico, Contraseña	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace click en "Login".
	2	El sistema muestra una pantalla en la que se le pide al usuario toda la información necesaria para iniciar sesión.
	3	El usuario rellena todos los campos y hace click en "Login".
	4	El sistema valida la información.
	5	El sistema muestra de nuevo la pantalla principal.
Postcondición	El usuario ha iniciado sesión.	
Excepciones	Paso	Acción
	4	No se ha podido realizar el inicio de sesión.
Comentarios	NA	
Actores	Usuario no registrado, Admin	

Tabla 2. Login

Requisito	**Logout**
Identificador	1.3
Prioridad	Alta
Precondición	El usuario ha de haber iniciado sesión.
Descripción	Los usuarios pueden salir de la sesión iniciada para iniciar una nueva sesión o navegar sin estar registrados.
Entrada	Nombre de usuario, Id de usuario

Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace click en "Logout".
	2	El sistema cierra la sesión y vuelve a la pantalla principal.
Postcondición	Se ha cerrado la sesión.	
Excepciones	Paso	Acción
	2	No se puede cerrar sesión.
Comentarios	NA	
Actores	Usuario registrado, Admin	

Tabla 3. Logout

Requisito	**Dar de baja usuario**	
Identificador	1.4	
Prioridad	Media	
Precondición	El admin tiene que haberse registrado y tener una sesión iniciada.	
Descripción	El admin puede eliminar cuentas de usuario.	
Entrada	ID usuario, Nombre de Usuario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace click en "Eliminar Usuario".
	2	El sistema muestra una pantalla en la que se le pide al usuario que introduzca el nombre del usuario que quiere eliminar.
	3	El usuario introduce el nombre del usuario.
	4	El sistema muestra una lista con los nombres de usuario que contengan la palabra introducida antes.
	5	El usuario selecciona el usuario que quiere eliminar y hace click en "Eliminar".
	6	El sistema muestra un mensaje preguntando por la confirmación.
	7	El usuario hace click en "Continuar".

	8	El sistema muestra la pantalla principal.
Postcondición	Se ha eliminado la cuenta de usuario.	
Excepciones	Paso	Acción
	4	No se ha encontrado ningún usuario.
	7	El usuario hace click en "Cancelar".
Comentarios	NA	
Actores	Admin	

Tabla 4. Dar de baja usuario

Requisito	**Modificar datos de usuario**	
Identificador	1.5	
Prioridad	Media	
Precondición	El admin tiene que haberse registrado y tener una sesión iniciada.	
Descripción	El admin puede modificar los datos de otras cuentas de usuario.	
Entrada	ID usuario, Nombre de Usuario, Nombre, Apellidos, Correo Electrónico	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y hace click en "Modificar Usuario".
	2	El sistema muestra una pantalla en la que se le pide al usuario que introduzca el nombre del usuario que quiere modificar.
	3	El usuario introduce el nombre del usuario.
	4	El sistema muestra una lista con los nombres de usuario que contengan la palabra introducida antes.
	5	El usuario selecciona el usuario que quiere modificar y hace click en "Modificar".
	6	El sistema muestra todos los datos del usuario.
	7	El usuario modifica los datos y hace click en "Modificar".
	8	El sistema muestra un mensaje preguntando por la confirmación.
	9	El usuario hace click en "Continuar".

	10	El sistema muestra los datos del usuario que se ha modificado.
Postcondición	Se han modificado los datos del usuario.	
Excepciones	Paso	Acción
	4	No se ha encontrado ningún usuario.
	9	El usuario hace click en "Cancelar".
	10	No se han podido modificar los datos del usuario.
Comentarios	NA	
Actores	Admin	

Tabla 5. Modificar datos de usuario

5.2.2 Backtesting estrategia de trading

Requisito	**Hacer el backtesting de un activo financiero concreto**	
Identificador	2.1	
Prioridad	Alta	
Precondición	El sistema tiene que tener datos históricos del activo financiero sobre el que se quiere hacer el backtesting.	
Descripción	Realizar el backtesting de la estrategia de posiciones largas y cortas de máximos y mínimos sobre el activo financiero seleccionado por el usuario.	
Entrada	Nombre del activo financiero, Capital inicial, Marco temporal, Tipo de salida	
Salida	Resultado del backtesting	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Realizar Backtesting"
	2	El sistema muestra una nueva pantalla con todas las opciones de configuración necesarias para llevar a cabo el backtesting.
	3	El usuario selecciona las opciones de configuración con las que hacer el backtesting y hace click en el botón "Backtesting".
	4	El sistema muestra los resultados del backtesting llevado a cabo.
Postcondición	NA	

Excepciones	Paso	Acción
Comentarios	Las opciones de configuración para llevar a cabo el backtesting que el usuario necesita seleccionar son las siguientes: • Seleccionar el activo financiero sobre el que hacer el backtesting y el capital inicial. • Seleccionar el marco temporal de referencia a utilizar para calcular los máximos y los mínimos. • Seleccionar el tipo de salida que quiere efectuar: salida temporal o salida con trailing stop. • En caso de elegir la salida con trailing stop, seleccionar el multiplicador a utilizar.	
Actores	Usuario registrado, Usuario no registrado	

Tabla 6. Hacer el backtesting de un activo financiero concreto

Requisito	**Hacer el backtesting de una cartera de activos financieros**	
Identificador	2.2	
Prioridad	Alta	
Precondición	El sistema debe tener datos históricos de los activos financieros que van a componer la cartera sobre la que se quiere hacer el backtesting.	
Descripción	Realizar el backtesting de la estrategia de posiciones largas y cortas de máximos y mínimos sobre una cartera formada por una serie de activos financieros seleccionados por el usuario.	
Entrada	Nombres de los activos financieros, Nº de acciones/contratos, Capital inicial, Marco temporal, Tipo de salida	
Salida	Resultado del backtesting	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Realizar Backtesting de una Cartera"
	2	El sistema muestra una nueva pantalla con todas las opciones de configuración necesarias para llevar a cabo el backtesting de una cartera.
	3	El usuario selecciona las opciones de configuración con las que hacer el backtesting y hace click en el botón "Backtesting".
	4	El sistema muestra los resultados del backtesting llevado a cabo.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	Las opciones de configuración para llevar a cabo el backtesting que el usuario necesita seleccionar son las siguientes:	

	• Seleccionar los activos financieros que van a componer la cartera, el número de acciones/contratos de cada uno y el capital inicial. • Seleccionar el marco temporal de referencia a utilizar para calcular los máximos y los mínimos. • Seleccionar el tipo de salida que quiere efectuar: salida temporal o salida con trailing stop. • En caso de elegir la salida con trailing stop, seleccionar el multiplicador a utilizar.
Actores	Usuario registrado, Usuario no registrado

Tabla 7. Hacer el backtesting de una cartera de activos financieros

Requisito	**Realizar backtesting de activos financieros agrupados en función de una métrica**	
Identificador	2.3	
Prioridad	Alta	
Precondición	Tiene que haber datos de la métrica a utilizar para cada uno de los activos financieros que van a conformar los distintos grupos sobre los que se va a realizar el backtesting. Tiene que haber datos históricos de los precios de los activos financieros sobre los que se va a realizar el backtesting.	
Descripción	Se puede agrupar a los activos financieros en distintos conjuntos en función de alguna métrica con la intención de realizar un backtesting sobre cada uno de esos grupos y observar si existen diferencias importantes en los rendimientos realizados por cada uno de los grupos.	
Entrada	Métrica, Capital inicial, Marco temporal, Tipo de salida	
Salida	Resultados del backtesting para cada uno de los grupos	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción “Realizar Backtesting de Grupos de Activos”
	2	El sistema muestra una nueva pantalla con todas las opciones de configuración necesarias para llevar a cabo el backtesting de los grupos.
	3	El usuario selecciona las opciones de configuración con las que hacer el backtesting y hace click en el botón “Backtesting”.
	4	El sistema muestra los resultados del backtesting llevado a cabo.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	Las opciones de configuración para llevar a cabo el backtesting que el usuario necesita seleccionar son las siguientes:	

	• Seleccionar la métrica que se va a utilizar para agrupar a los activos financieros en distintos conjuntos. • Seleccionar el capital inicial. • Seleccionar el marco temporal de referencia a utilizar para calcular los máximos y los mínimos. • Seleccionar el tipo de salida que quiere efectuar: salida temporal o salida con trailing stop. • En caso de elegir la salida con trailing stop, seleccionar el multiplicador a utilizar.
Actores	Usuario registrado, Usuario no registrado

Tabla 8. Realizar backtesting de activos financieros agrupados en función de una métrica

Requisito	**Obtener el número de acciones/contratos de una cartera a partir de un rendimiento esperado**	
Identificador	2.4	
Prioridad	Media	
Precondición	El sistema tiene que tener datos históricos de los activos financieros que van a componer la cartera y tiene que existir una combinación de acciones/contratos que pueda arrojar un rendimiento igual o parecido al deseado por el usuario.	
Descripción	Dada una cartera formada por un conjunto de activos financieros y un rendimiento esperado por el usuario, se obtiene el número de acciones/contratos que habría que tener de cada uno de esos activos financieros que es necesario para alcanzar dicho rendimiento. La estrategia de inversión utilizada es la basada en posiciones largas y cortas de máximos y mínimos.	
Entrada	Nombres de los activos financieros, Capital inicial, Marco temporal, Tipo de salida, Rendimiento	
Salida	Número de acciones/contratos de cada uno de los activos financieros que componen la cartera	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Determinar Nº de Acciones/Contratos de una Cartera"
	2	El sistema muestra una nueva pantalla en la que se pide al usuario seleccionar los activos financieros que van a componer la cartera y el rendimiento deseado por este.
	3	El usuario selecciona los activos financieros que van a componer su cartera, introduce el rendimiento deseado y hace click en el botón "Siguiente".
	4	El sistema muestra una nueva pantalla en la que se pide al usuario seleccionar las opciones de configuración relativas a la estrategia de inversión utilizada.

	5	El usuario selecciona las opciones de configuración relativas a la estrategia de inversión utilizada y hace click en el botón "Determinar Composición de Cartera".
	6	El sistema muestra el número de acciones/contratos necesarios para alcanzar el rendimiento deseado por este.
Postcondición	El resultado arrojado (salida) tiene que permitir alcanzar el rendimiento introducido anteriormente por el usuario en el período histórico al que pertenecen los datos de los precios utilizados.	
Excepciones	Paso	Acción
	4	Se muestra una pantalla indicando que el rendimiento introducido tiene que ser positivo.
	6	Se muestra un mensaje indicando que no es posible determinar una combinación de los activos financieros que componen la cartera que permita alcanzar el rendimiento deseado.
Comentarios	Las opciones de configuración relativas a la estrategia de inversión utilizada que el usuario necesita seleccionar son las siguientes: • Seleccionar el capital inicial. • Seleccionar el marco temporal de referencia a utilizar para calcular los máximos y los mínimos. • Seleccionar el tipo de salida que quiere efectuar: salida temporal o salida con trailing stop. • En caso de elegir la salida con trailing stop, seleccionar el multiplicador a utilizar.	
Actores	Usuario registrado, Usuario no registrado	

Tabla 9. Obtener el número de acciones/contratos de una cartera a partir de un rendimiento esperado

Requisito	**Obtener la composición de una cartera a partir de un rendimiento esperado**
Identificador	2.5
Prioridad	Media
Precondición	El sistema tiene que tener datos históricos de los activos financieros que van a componer la cartera y tiene que existir una combinación de acciones/contratos de una serie de activos financieros que pueda arrojar un rendimiento igual o parecido al deseado por el usuario.
Descripción	Dado un rendimiento esperado por el usuario, se obtienen tanto los activos financieros como el número de acciones/contratos de cada uno de ellos que es necesario para alcanzar dicho rendimiento. La estrategia de inversión utilizada es la basada en posiciones largas y cortas de máximos y mínimos.
Entrada	Capital inicial, Marco temporal, Tipo de salida, Rendimiento
Salida	Activos financieros y número de acciones/contratos de cada uno de esos activos financieros.

Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Determinar Composición de una Cartera"
	2	El sistema muestra una nueva pantalla en la que se pide al usuario seleccionar el rendimiento deseado por este.
	3	El usuario introduce el rendimiento deseado y hace click en el botón "Siguiente".
	4	El sistema muestra una nueva pantalla en la que se pide al usuario seleccionar las opciones de configuración relativas a la estrategia de inversión utilizada.
	5	El usuario selecciona las opciones de configuración relativas a la estrategia de inversión utilizada y hace click en el botón "Determinar Composición de Cartera".
	6	El sistema muestra los activos financieros y el número de acciones/contratos de cada uno de ellos necesarios para alcanzar el rendimiento deseado por este.
Postcondición	El resultado arrojado (salida) tiene que permitir alcanzar el rendimiento introducido anteriormente por el usuario en el período histórico al que pertenecen los datos de los precios utilizados.	
Excepciones	Paso	Acción
	4	Se muestra una pantalla indicando que el rendimiento introducido tiene que ser positivo.
	6	Se muestra un mensaje indicando que no es posible determinar una posible composición de cartera que permita alcanzar el rendimiento deseado.
Comentarios	Las opciones de configuración relativas a la estrategia de inversión utilizada que el usuario necesita seleccionar son las siguientes: • Seleccionar el capital inicial. • Seleccionar el marco temporal de referencia a utilizar para calcular los máximos y los mínimos. • Seleccionar el tipo de salida que quiere efectuar: salida temporal o salida con trailing stop. • En caso de elegir la salida con trailing stop, seleccionar el multiplicador a utilizar.	
Actores	Usuario registrado, Usuario no registrado	

Tabla 10. Obtener la composición de una cartera a partir de un rendimiento esperado

Requisito	**Simular estrategia de trading en tiempo real**
Identificador	2.6
Prioridad	Media

Precondición	El usuario tiene que estar registrado y haber iniciado sesión. El usuario tiene que haber realizado a través de la aplicación el backtesting de una acción o de una cartera. Es decir, el usuario se encuentra en la página que contiene los resultados del backtesting que ha realizado. El sistema tiene que tener datos en tiempo real de los activos financieros sobre los que se va a realizar la simulación.	
Descripción	Realizar la simulación de la estrategia de posiciones largas y cortas de máximos y mínimos sobre un activo financiero o una cartera con datos en tiempo real.	
Entrada	Nombres de los activos financieros, Nº de acciones/contratos, Capital inicial, Marco temporal, Tipo de salida	
Salida	Resultado de la simulación en tiempo real	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla que contiene los resultados del backtesting y hace click en "Realizar Simulación en Tiempo Real"
	2	El sistema muestra una nueva pantalla con los resultados de la simulación en tiempo real.
	3	El usuario, si así lo desea, puede hacer click en "Finalizar Simulación".
Postcondición	NA	
Excepciones	Paso	Acción
	2	El sistema dice que no se disponen de los datos de esos activos financieros en tiempo real.
Comentarios	La simulación realizada utiliza la estrategia de trading basada en posiciones largas y cortas de los máximos y mínimos de distintos marcos temporales de referencia. Y, en concreto, el marco temporal utilizado es el elegido previamente en el backtesting realizado con anterioridad a la simulación en tiempo real.	
Actores	Usuario registrado	

Tabla 11. Simular estrategia de trading en tiempo real

Requisito	**Compartir backtesting con otros usuarios**
Identificador	2.7
Prioridad	Alta
Precondición	El usuario tiene que estar registrado y haber iniciado sesión. El usuario tiene que encontrarse en la pantalla que contiene los datos del backtesting realizado anteriormente y que ahora quiere compartir.
Descripción	Compartir con otros usuarios la información relativa al backtesting realizado.
Entrada	ID del backtesting, Id usuario

Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla que contiene los resultados del backtesting y hace click en "Compartir".
	2	El sistema muestra un mensaje preguntando al usuario si quiere seguir adelante.
	3	El usuario hace click en "Sí".
Postcondición	La información relativa al backtesting realizado es visible por el resto de usuarios.	
Excepciones	Paso	Acción
	3	El usuario hace click en "No".
Comentarios	Para entender cómo buscar el backtesting publicado por otro usuario, mirar Requisito "Buscar Backtestings por el nombre del activo financiero".	
Actores	Usuario registrado	

Tabla 12. Compartir backtesting con otros usuarios

Requisito	**Buscar Backtesting por el nombre del activo financiero**	
Identificador	2.8	
Prioridad	Alta	
Precondición	NA	
Descripción	El usuario podrá realizar la búsqueda por el nombre del activo financiero de los backtestings compartidos por otros usuarios.	
Entrada	Nombre del activo financiero	
Salida	Lista de los backtestings realizados sobre el activo financiero buscado y lista de los backtestings realizados sobre una cartera que contiene el activo financiero buscado.	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla principal, hace click en la barra de búsqueda e introduce el nombre del activo financiero.
	2	El sistema muestra los resultados de la búsqueda.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado, Usuario no registrado	

Tabla 13. Buscar Backtesting por el nombre del activo financiero

Requisito	**Guardar backtesting**	
Identificador	2.9	
Prioridad	Alta	
Precondición	El usuario tiene que estar registrado y haber iniciado sesión. El usuario tiene que encontrarse en la pantalla que contiene los datos del backtesting realizado anteriormente y que ahora quiere guardar.	
Descripción	Guardar la información de un backtesting realizado anteriormente con la intención de poder verlo después.	
Entrada	ID del backtesting, Id usuario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla que contiene los resultados del backtesting y hace click en "Guardar".
	2	El sistema muestra un mensaje preguntando al usuario si quiere seguir adelante.
	3	El usuario hace click en "Sí".
Postcondición	Los datos del backtesting se han guardado.	
Excepciones	Paso	Acción
	3	El usuario hace click en "No".
Comentarios		
Actores	Usuario registrado	

Tabla 14. Guardar Backtesting

Requisito	**Ver información de un backtesting**	
Identificador	2.10	
Prioridad	Alta	
Precondición	El backtesting debe existir.	
Descripción	El usuario podrá ver la información relativa a un backtesting publicado por otro usuario.	
Entrada	ID del backtesting	
Salida	Información del backtesting	
	Paso	Acción

Secuencia normal	1	El usuario accede a una lista de backtestings.
	2	El usuario selecciona un backtesting.
	3	El sistema muestra la información del backtesting.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado, Usuario no registrado	

Tabla 15. Ver información de un backtesting

Requisito	**Realizar Análisis Descriptivo del Backtesting**	
Identificador	2.11	
Prioridad	Baja	
Precondición	El usuario tiene que estar registrado y tiene que existir el backtesting sobre el que se quiere realizar el análisis descriptivo.	
Descripción	Se puede mostrar más información de carácter estadístico acerca del backtesting realizado previamente por el usuario.	
Entrada	ID backtesting	
Salida	Datos de carácter estadístico	
Secuencia normal	Paso	Acción
	1	El usuario accede a una lista de backtestings.
	2	El usuario selecciona un backtesting.
	3	El sistema muestra la información del backtesting.
	4	El usuario selecciona la opción "Análisis Descriptivo".
	5	El sistema muestra los resultados del análisis descriptivo.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado	

Tabla 16. Realizar Análisis Descriptivo del Backtesting

5.2.3 Carteras de activos financieros de los usuarios

Requisito	**Crear cartera de activos financieros**	
Identificador	3.1	
Prioridad	Media	
Precondición	El usuario ha de estar registrado y con la sesión iniciada.	
Descripción	Se puede configurar una cartera de activos financieros para observar cómo evoluciona el rendimiento producido por ella en el tiempo.	
Entrada	Nombres de los activos financieros, Nº de acciones/contratos, ID usuario	
Salida	Cartera de activos financieros y rendimiento	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Crear cartera"
	2	El sistema muestra una pantalla en la que se le pide al usuario introducir los activos financieros que van a componer la cartera.
	3	El usuario introduce el nombre de los activos financieros y el número de acciones/contratos de cada uno de ellos.
	4	El usuario hace click en el botón "Crear".
Postcondición	NA	
Excepciones	Paso	Acción
	4	Se muestra un mensaje de error en caso de que se haya introducido un nombre que no existe o un valor negativo.
Comentarios	NA	
Actores	Usuario registrado	

Tabla 17. Crear cartera de activos financieros

Requisito	**Modificar cartera de activos financieros**
Identificador	3.2
Prioridad	Media
Precondición	El usuario ha de estar registrado y con la sesión iniciada. La cartera tiene que existir.

Descripción	Un usuario puede modificar una cartera suya comprando o vendiendo activos financieros.	
Entrada	ID cartera, Nombres de los activos financieros, Nº de acciones/contratos, ID usuario	
Salida	Cartera de activos financieros y rendimiento	
Secuencia normal	Paso	Acción
	1	El usuario accede a la lista de sus carteras.
	2	El usuario selecciona una cartera.
	3	El sistema muestra una pantalla en la que se observan los datos de la cartera.
	4	El usuario hace click en el botón "Modificar"
	5	El sistema muestra una pantalla que permite comprar y vender activos financieros.
	6	El usuario compra o vende activos financieros y hace click en "Finalizar".
	7	El sistema muestra de nuevo una pantalla en la que se observan los datos de la cartera.
Postcondición	NA	
Excepciones	Paso	Acción
	6	Se muestra un mensaje de error en caso de que se haya introducido el nombre de un activo financiero que no existe o un número negativo.
Comentarios	NA	
Actores	Usuario registrado	

Tabla 18. Modificar cartera de activos financieros

Requisito	**Finalizar cartera de activos financieros**	
Identificador	3.3	
Prioridad	Media	
Precondición	El usuario ha de estar registrado y con la sesión iniciada. La cartera tiene que existir.	
Descripción	Un usuario puede eliminar una cartera suya.	
Entrada	ID Cartera, ID usuario	
Salida	NA	
	Paso	Acción

Secuencia normal	1	El usuario accede a la lista de sus carteras.
	2	El usuario selecciona una cartera.
	3	El sistema muestra una pantalla en la que se observan los datos de la cartera.
	4	El usuario hace click en el botón "Eliminar"
	5	El sistema muestra un mensaje de confirmación.
	6	El usuario hace click en "Continuar".
	7	El sistema vuelve a la lista de las carteras.
Postcondición	La cartera eliminada ya no existe.	
Excepciones	Paso	Acción
	6	El usuario hace click en "Cancelar".
Comentarios	NA	
Actores	Usuario registrado	

Tabla 19. Finalizar cartera de activos financieros

Requisito	**Compartir cartera de activos financieros**	
Identificador	3.4	
Prioridad	Media	
Precondición	El usuario ha de estar registrado y con la sesión iniciada. La cartera tiene que existir.	
Descripción	Un usuario puede compartir una cartera suya para que el resto de usuarios pueda verla.	
Entrada	ID cartera, ID usuario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la lista de sus carteras.
	2	El usuario selecciona una cartera.
	3	El sistema muestra una pantalla en la que se observan los datos de la cartera.
	4	El usuario hace click en el botón "Compartir"
	5	El sistema muestra un mensaje de confirmación.

	6	El usuario hace click en "Continuar".
	7	El sistema vuelve a mostrar los datos de la cartera.
Postcondición	La cartera es visible por el resto de los usuarios.	
Excepciones	Paso	Acción
	6	El usuario hace click en "Cancelar".
Comentarios	NA	
Actores	Usuario registrado	

Tabla 20. Compartir cartera de activos financieros

Requisito	**Buscar cartera de activos financieros**	
Identificador	3.5	
Prioridad	Media	
Precondición	NA	
Descripción	Los usuarios pueden buscar las carteras compartidas por otros usuarios por el nombre de alguno de los activos financieros que componen dicha cartera.	
Entrada	Nombre del activo financiero	
Salida	Lista de carteras	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla principal, hace click en la barra de búsqueda e introduce el nombre del activo financiero.
	2	El sistema muestra los resultados de la búsqueda.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado, Usuario no registrado	

Tabla 21. Buscar cartera de activos financieros

Requisito	Ver información de cartera de activos financieros	
Identificador	3.6	
Prioridad	Media	
Precondición	La cartera debe existir.	
Descripción	Se podrán ver los detalles que componen una cartera, en concreto, los activos financieros que la componen y el resultado arrojado por esta.	
Entrada	ID cartera	
Salida	Nombres de los activos financieros, Nº de acciones/contratos, Capital, Rendimiento	
Secuencia normal	Paso	Acción
	1	El usuario accede a una lista de carteras.
	2	El usuario selecciona una cartera.
	3	El sistema muestra la información de la cartera.
Postcondición	NA	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado, Usuario no registrado	

Tabla 22. Ver información de una cartera de activos financieros

5.2.4 Otros servicios de trading

Requisito	Realizar backtesting de forma manual
Identificador	4.1
Prioridad	Baja
Precondición	NA
Descripción	El usuario puede introducir de forma manual las ganancias y las pérdidas que obtiene siguiendo una determinada estrategia y, así, observar el rendimiento obtenido con esa estrategia.
Entrada	Capital inicial, Ganancias, Pérdidas
Salida	Rendimiento obtenido

Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Realizar backtesting de forma manual"
	2	El sistema muestra una pantalla con las opciones necesarias para realizar el backtesting de forma manual.
	3	El usuario introduce el capital inicial.
	4	El usuario introduce la cantidad de cada ganancia y la cantidad de cada pérdida y hace click en el botón correspondiente para añadir una ganancia o una pérdida. Y así para cada ganancia y pérdida realizada.
	5	El sistema muestra los resultados y una gráfica actualizada con el rendimiento realizado.
Postcondición	El rendimiento mostrado por pantalla es el correcto después de aplicar las ganancias y pérdidas realizadas por el usuario.	
Excepciones	Paso	Acción
	3	Si el usuario introduce un valor negativo, se muestra un mensaje de error.
	4	Si el usuario introduce un valor negativo tanto en la ganancia como en la pérdida, se muestra un mensaje de error.
Comentarios	Hay un botón verde para añadir una ganancia y un botón rojo para añadir una pérdida.	
Actores	Usuario registrado, Usuario no registrado	

Tabla 23. Realizar backtesting de forma manual

Requisito	**Escribir diario de trading**	
Identificador	4.2	
Prioridad	Baja	
Precondición	El usuario tiene que estar registrado y haber iniciado sesión.	
Descripción	Los usuarios pueden escribir como si de un diario se tratara sus pensamientos y sus reflexiones acerca de sus operaciones en bolsa y, también, los backtesting realizados.	
Entrada	ID usuario, Texto del diario	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario accede a la aplicación, se le muestra la pantalla principal y selecciona la opción "Diario de Trading"

	2	El sistema muestra un calendario.
	3	El usuario selecciona el día del calendario sobre el que quiere escribir.
	4	El sistema permite al usuario introducir texto.
	5	El usuario introduce texto y hace click en "Finalizar".
	6	El sistema vuelve al calendario.
Postcondición	El calendario contiene el texto introducido por el usuario en el día seleccionado por este.	
Excepciones	Paso	Acción
Comentarios	NA	
Actores	Usuario registrado	

Tabla 24. Escribir diario de trading

5.2.5 Administrar datos

Requisito	**Subir datos históricos**	
Identificador	5.1	
Prioridad	Baja	
Precondición	El admin tiene que haber iniciado sesión como tal.	
Descripción	Se pueden subir datos sobre activos financieros a la base de datos del sistema.	
Entrada	Datos que se quieren añadir.	
Salida	Datos recién añadidos.	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla principal y selecciona la opción "Administrar Datos".
	2	El sistema muestra una pantalla en la que se observan dos opciones a elegir por el usuario: "Subir Datos" y "Eliminar Datos".
	3	El usuario hace click en "Eliminar Datos".

	4	El sistema muestra una pantalla que pide introducir los datos a añadir.
	5	El usuario introduce los datos.
	6	El usuario hace click en "Añadir".
	7	El sistema muestra un mensaje preguntando por la confirmación.
	8	El usuario hace click en "Continuar".
	9	El sistema muestra por pantalla los datos recién añadidos.
Postcondición	Los datos han sido añadidos al sistema.	
Excepciones	Paso	Acción
	8	El usuario hace click en "Cancelar".
	9	No se pueden añadir los datos.
Comentarios	Se irá detallando cuáles son los tipos de datos que se va a poder subir conforme se va diseñando la aplicación con mayor profundidad.	
Actores	Admin	

Tabla 25. Subir datos históricos

Requisito	**Eliminar datos históricos**	
Identificador	5.2	
Prioridad	Baja	
Precondición	Tienen que existir los datos que se quieren eliminar y el admin tiene que haber iniciado sesión como tal.	
Descripción	Se pueden eliminar datos de activos financieros de la base de datos del sistema.	
Entrada	ID de los datos que se quieren eliminar.	
Salida	NA	
Secuencia normal	Paso	Acción
	1	El usuario se encuentra en la pantalla principal y selecciona la opción "Administrar Datos".
	2	El sistema muestra una pantalla en la que se observan dos opciones a elegir por el usuario: "Subir Datos" y "Eliminar Datos".
	3	El usuario hace click en "Eliminar Datos".
	4	El sistema muestra una pantalla que pide introducir los datos a eliminar.

	5	El usuario selecciona los datos a eliminar.
	6	El usuario hace click en “Eliminar”.
	7	El sistema muestra un mensaje preguntando por la confirmación.
	8	El usuario hace click en “Continuar”.
	9	El sistema vuelve a la pantalla que muestra las dos opciones al usuario: “Subir Datos” y “Eliminar Datos”.
Postcondición	Ya no existen los datos eliminados.	
Excepciones	Paso	Acción
	8	El usuario hace click en “Cancelar”.
	9	No se pueden eliminar los datos.
Comentarios	Se irá detallando cuáles son los tipos de datos que se va a poder eliminar conforme se va diseñando la aplicación con mayor profundidad.	
Actores	Admin	

Tabla 26. Eliminar datos históricos

Capítulo 6 - Arquitectura

En este capítulo se hace una descripción de la arquitectura que se ha seguido para el desarrollo e implementación de la aplicación web.

6.1 Arquitectura del sistema

La arquitectura del sistema sigue un modelo cliente-servidor tal como se muestra en la Ilustración 15.

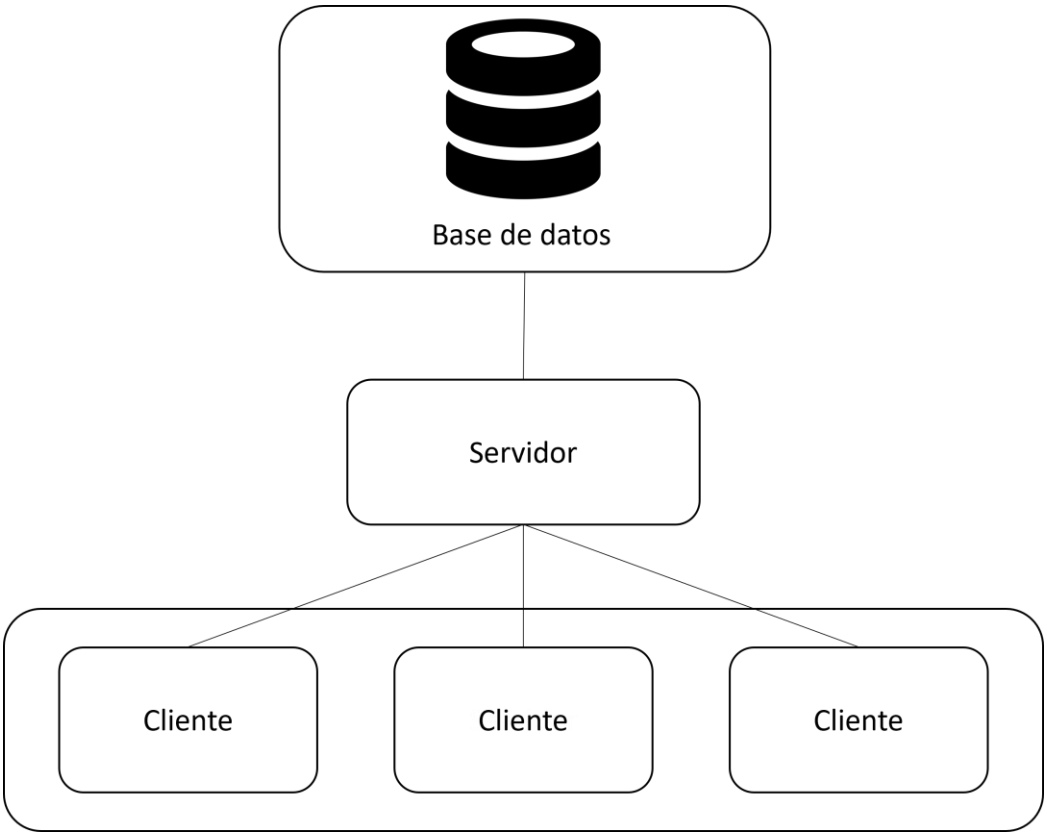


Ilustración 15. Arquitectura del sistema

Esta es una arquitectura en la que uno o más clientes realizan peticiones y reciben servicio de un servidor central. Para ello, el cliente necesita disponer de una interfaz visual a través de la cual realizar las peticiones y recibir las respuestas del servidor. En este caso, el cliente tiene que comunicarse con el servidor central a través de un navegador web. El servidor central, por su parte, al iniciarse espera a recibir las solicitudes de los clientes. Cuando estas llegan, el servidor

las procesa y envía la respuesta correspondiente al cliente que había hecho la petición. Y toda esta comunicación se realiza a través del protocolo de la capa de aplicación HTTP [17].

6.2 Patrones arquitectónicos

6.2.1 MVC (Modelo-Vista-Controlador)

El patrón arquitectónico MVC [18] es un patrón que separa la lógica y los datos del sistema de la forma en la que esos datos son presentados a través de la interfaz visual al usuario. Para ello, el modelo divide la aplicación en tres componentes interconectados:

- Modelo: Es la representación de la información y los datos del sistema.
- Vista: Hace referencia a la presentación de los datos a través de la interfaz visual al usuario.
- Controlador: Es el intermediario entre el "modelo" y la "vista". Responde a eventos del usuario y realiza peticiones al modelo. Y, en el otro sentido, también puede mandar comandos a la "vista".

El uso de Flask en esta aplicación web es totalmente compatible con el patrón MVC. En Flask, se escriben métodos que se asignan a rutas específicas. Además, Flask brinda la posibilidad de utilizar Jinja2. A continuación, se muestra en la Ilustración 16 una simplificación del código que sirve de ejemplo a lo comentado:

```
@views.route('/')
def home():
    return render_template("home.html")
```

Ilustración 16. Ejemplo de la implementación MVC en Flask

El método "home", que, además, tiene asignado la ruta específica "/", actúa como el "controlador" del patrón MVC. Y el template "home.html" hace de la "vista" de dicho patrón. En cuanto al "modelo", Flask da libertad a la hora de su implementación [18].

6.2.2 DAO

El patrón "Data Access Object" (DAO) es un patrón arquitectónico que permite separar la lógica del negocio de la fuente de datos. En la aplicación web aquí desarrollada, este patrón ha sido implementado concretamente para separar los datos de las estrategias a nivel de la lógica de negocio de los almacenados en la base de datos. La Ilustración 17 contiene el correspondiente diagrama de clases.

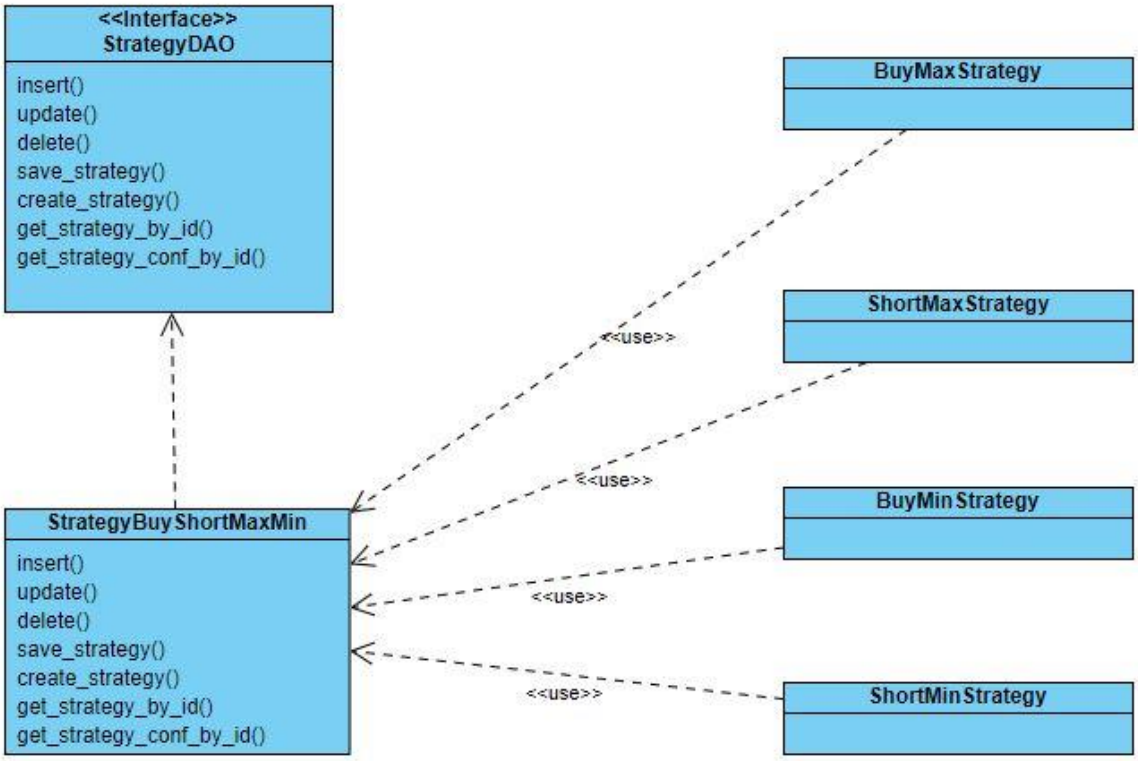


Ilustración 17. Diagrama de clases del patrón DAO

6.3 Patrones de diseño

6.3.1 Patrón Factoría

Este patrón de diseño permite la creación de objetos a través de una interfaz común [20]. En este caso, este patrón ha sido utilizado para la creación de objetos de distintas estrategias.

Cada estrategia de trading implementada en la aplicación web tiene su propia clase. La clase de la estrategia "Comprar en Máximos" es "BuyMaxStrategy", la clase de la estrategia "Vender en Mínimos" es "ShortMaxStrategy", la clase de la estrategia "Comprar en Mínimos" es "BuyMinStrategy" y la clase de la estrategia "Vender en Mínimos" es "ShortMinStrategy". Por lo tanto, cada vez que se ejecuta un "backtesting", se crea una instancia de una de las cuatro clases recién comentadas. Para ello, en vez de utilizar una estructura condicional if/elif/else para determinar cuál es la clase de la estrategia que se va a instanciar, la aplicación delega tal decisión a un componente separado que se encarga de crear dicho objeto. A continuación, la Ilustración 18 incluye un diagrama de clases que relaciona los distintos componentes que conforman la implementación de este patrón de diseño:

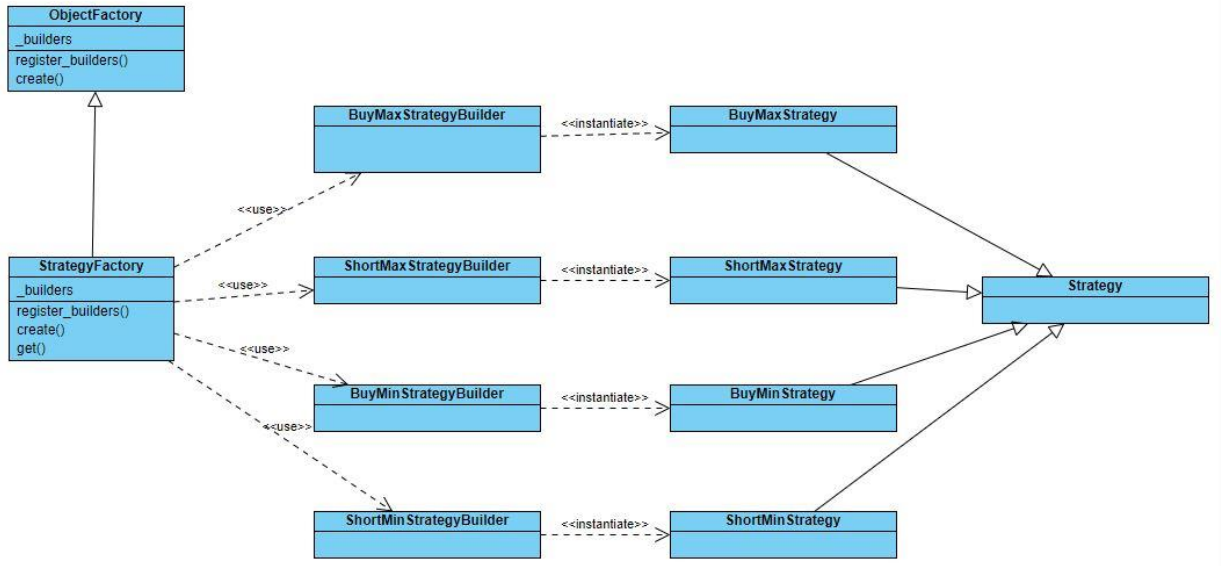


Ilustración 18. Diagrama de clase del patrón Factoría

6.3.2 Patrón Singleton

Singleton es un patrón de diseño creacional que permite crear tan solo un objeto de su tipo. En este trabajo, se ha utilizado este patrón para la clase "StrategyFactory". Por lo que así se garantiza que durante la ejecución de la aplicación tan solo existe un objeto de este tipo. La Ilustración 19 contiene el correspondiente diagrama de clases.

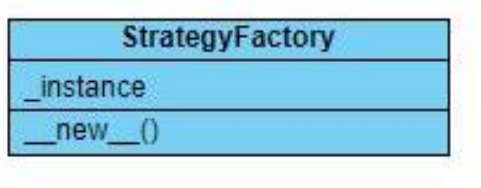


Ilustración 19. Diagrama de clases del patrón Singleton

Capítulo 7 - Modelo de datos

En este capítulo se explica el modelo de datos que se ha definido para realizar la persistencia de la información gestionada por la aplicación web.

7.1 Modelo Entidad-Relación

Para el diseño de la base de datos del sistema, se ha descrito un modelo entidad-relación que refleje las entidades de información que necesitan ser gestionadas por la aplicación.

Los datos que guarda el sistema pueden ser de dos tipos distintos: datos de los activos financieros y datos del usuario o generados por el usuario. Los datos de los activos financieros son también de dos tipos. En primer lugar, están los datos referentes a las especificaciones de los contratos de los activos financieros. Aquí estarían, por ejemplo, el nombre del activo, el mercado en el que cotiza el activo financiero en cuestión o la moneda en la que está denominado dicho contrato. Por otro lado, estarían los datos históricos de los precios diarios de los activos financieros. Estos datos incluyen la fecha, el precio de apertura, el precio de cierre, el máximo del día, el mínimo del día, el precio de cierre ajustado y el volumen.

En cuanto a los datos del usuario o generados por el usuario, en primer lugar, estarían los que hacen referencia a las cuentas de usuario. Para configurar una cuenta de usuario, es necesario tener un nombre de usuario, un nombre completo, una contraseña y un rol ("Administrador" o "Usuario").

Los datos generados por el usuario pueden ser de cuatro tipos distintos (en función de la funcionalidad a la que estén asociados): "backtesting", gestión de carteras, "backtesting" manual y diario de trading.

Para la ejecución de un "backtesting", es necesario almacenar el usuario que ejecuta dicho "backtesting", el nombre que este usuario le pone, la información de la cartera que se ha configurado para la ejecución de este, las fechas de inicio y de fin, si el usuario lo ha guardado o no, si el usuario lo ha hecho público o no, la parametrización de la estrategia a testear y los activos financieros sobre los que correr el "backtesting". El usuario, la parametrización de la estrategia y los activos financieros sobre los que correr el "backtesting" van a conformar entidades independientes y el resto de los atributos mencionados van a pertenecer a la entidad "Backtesting".

La gestión de carteras genera igualmente una serie de datos a tener en cuenta. En primer lugar, se generan los datos de la configuración de la cartera. Los atributos de la entidad

"Cartera" son la fecha de inicio y de fin, el capital, la moneda en la que está denominada la cartera y si el usuario la ha compartido o no. En segundo lugar, la gestión de carteras también incluye transacciones y posiciones. Por lo tanto, también tienen que existir dichas entidades.

Para cada "backtesting" manual que un usuario crea y ejecuta, se necesita conocer el usuario que lo ha creado, el nombre que le ha puesto, la configuración de la cartera de este y si el usuario lo ha guardado o no para su posterior recuperación. Además, cada "backtesting" manual está compuesto por una serie de operaciones añadidas a través de la aplicación por el usuario. Estas últimas van a conformar igualmente una entidad.

Finalmente, el sistema también guarda los datos generados por el usuario en la funcionalidad "Diario de trading". Los atributos de la entidad "Diario de trading" van a ser el usuario, la fecha de creación del texto, la fecha a la que hace referencia el texto y el texto mismo.

La Ilustración 20 contiene el diagrama Entidad-Relación referido al modelo que se acaba de describir.

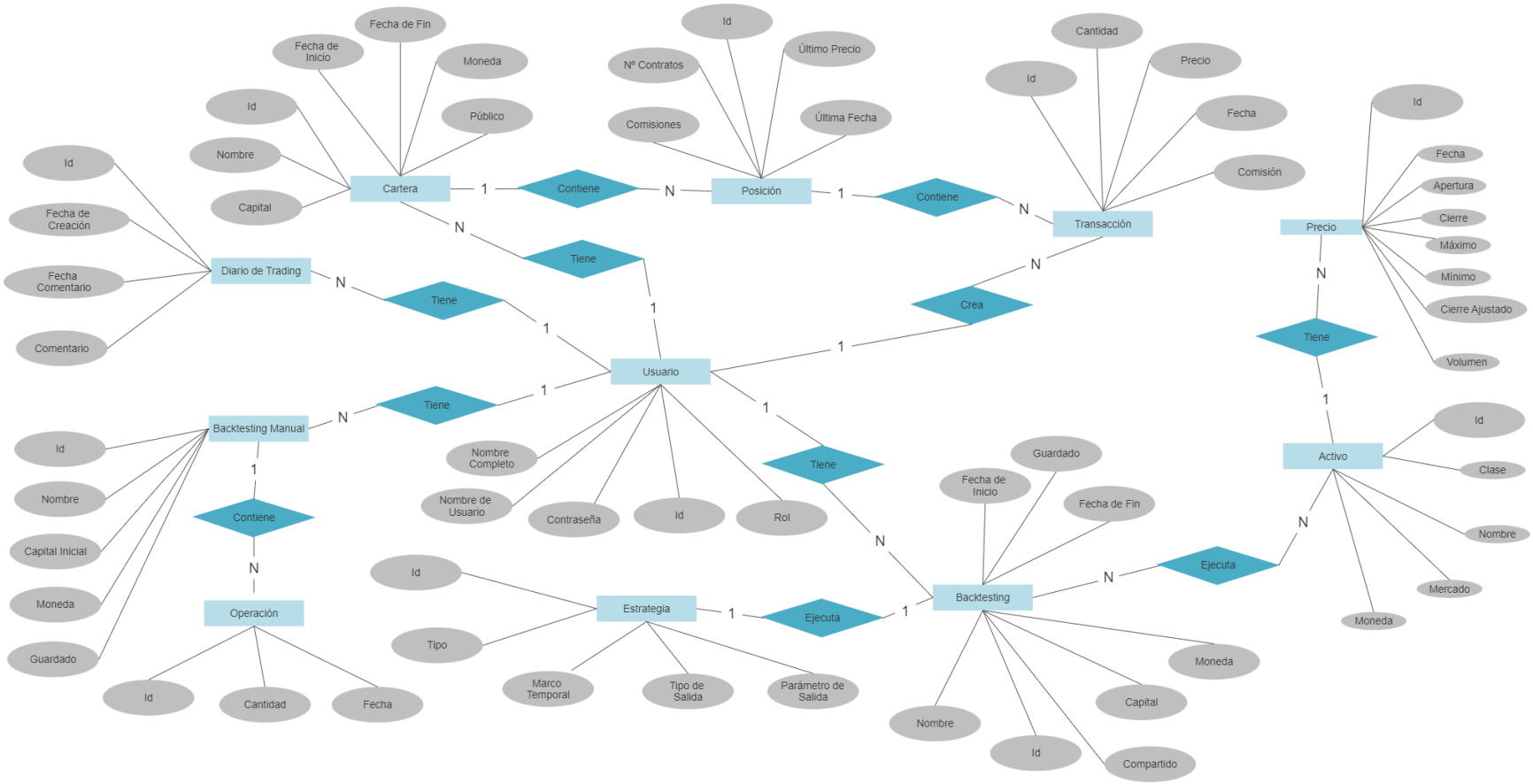


Ilustración 20. Diagrama Entidad-Relación

7.2 Implementación de la base de datos

La implementación del modelo E-R descrito en el anterior apartado se ha hecho a través de una base relacional de tipo MySQL.

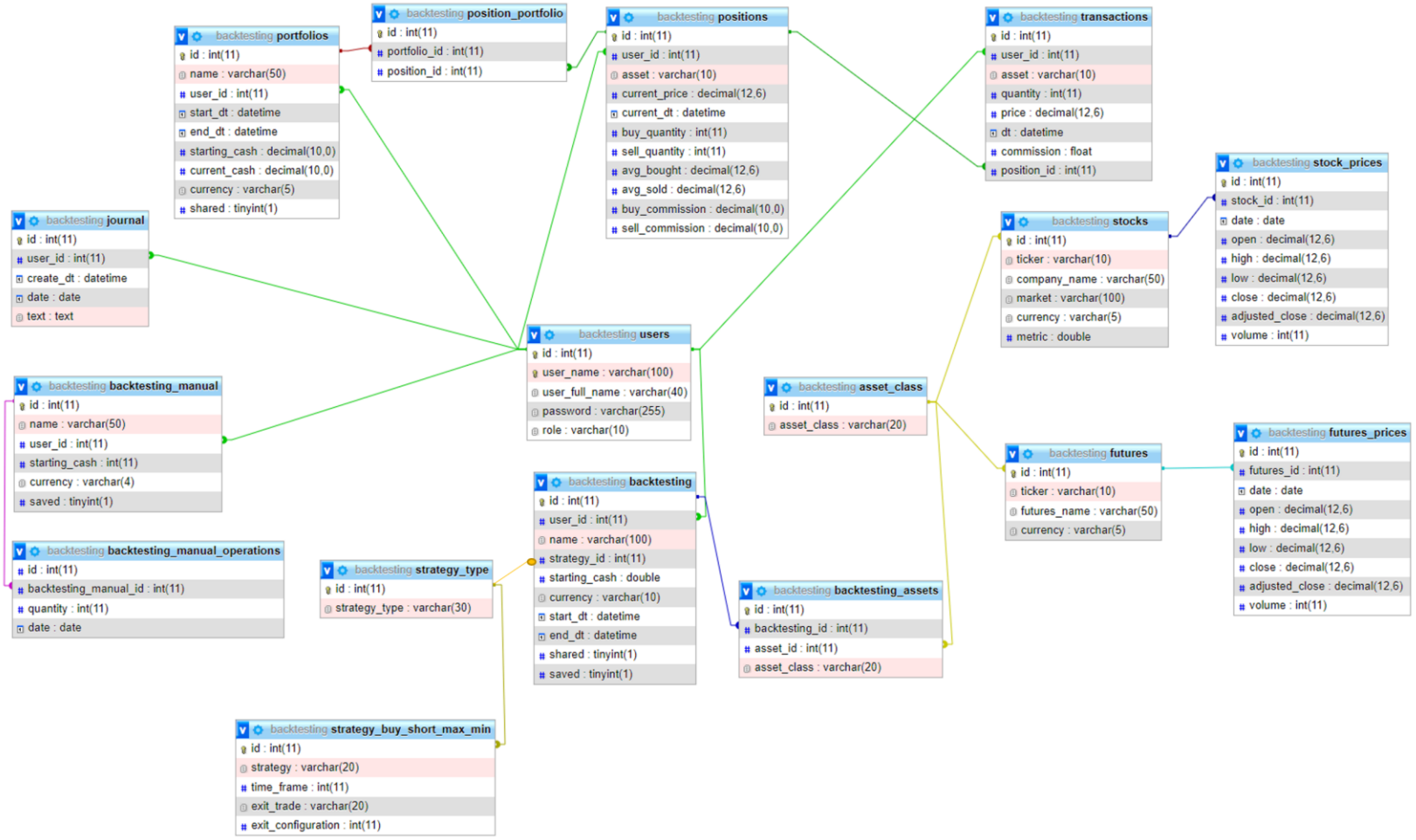


Ilustración 21. Implementación de la base de datos

A continuación, se describen cada una de las tablas de la base de datos.

7.2.1 Tablas sobre las cuentas de usuario

- Tabla "users": incluye la información necesaria que ha de tener toda cuenta de usuario creada en la aplicación web. Estos son sus campos de información:
    - "id": identificador del usuario.
    - "user_name": nombre de usuario.
    - "user_full_name": nombre y apellidos del usuario.
    - "password": contraseña de la cuenta.
    - "role": rol que tiene el usuario. Puede ser "Admin" (Administrador) o "User" (Usuario registrado corriente).

7.2.2 Tablas sobre "Backtesting"

- Tabla "backtesting": incluye todos los datos que se necesitan para la ejecución de un "backtesting". Estos son sus campos de información:
    - "id": identificador del "backtesting".
    - "user_id": identificador del usuario al que pertenece el "backtesting".
    - "name": nombre que le ha puesto el usuario al "backtesting".
    - "strategy_id": identificador de la estrategia del "backtesting".
    - "starting_cash": capital inicial de la cartera sobre la que se ejecuta el "backtesting".
    - "currency": moneda en la que están denominados los datos del campo "starting_cash".
    - "start_dt": fecha de inicio del "backtesting".
    - "end_dt": fecha de fin del "backtesting".
    - "shared": si el usuario ha compartido el "backtesting" o no.
    - "saved": si el usuario ha guardado el "backtesting" o no.
- Tabla "strategy_type": tabla que relaciona la estrategia del "backtesting" (tabla anterior) con el tipo de estrategia que es esta. Estos son sus campos de información:
    - "id": identificador de la estrategia ("strategy_id" en tabla "backtesting").

    - "strategy_type": tipo de la estrategia.
- Tabla "strategy_buy_short_max_min": incluye la parametrización de la estrategia implementada en esta aplicación web (recordar que la estrategia aquí implementada es una estrategia global que se puede dividir en cuatro tipos de sub-estrategias más concretas: comprar en máximos, vender en máximos, comprar en mínimos y vender en mínimos). Estos son sus campos de información:
    - "id": identificador de la estrategia.
    - "strategy": nombre de la sub-estrategia. Puede ser "buymax" (comprar en máximos), "shortmax" (vender en máximos), "buymin" (comprar en mínimos) y "shortmin" (vender en mínimos).
    - "time_frame": número de días sobre los que calcular el máximo o el mínimo de la estrategia.
    - "exit_trade": tipo de salida de la estrategia. Puede ser "exit_time" (salida por tiempo) o "trailing_stop" (salida cuando se ejecuta un stop).
    - "exit_configuration": parámetro del tipo de salida. Su significado depende del valor de "exit_trade".
- Tabla "backtesting_assets": tabla que guarda cuál es la lista de activos financieros sobre los que se ejecuta un "backtesting". Estos son sus campos de información:
    - "id": identificador de la fila.
    - "backtesting_id": identificador del "backtesting".
    - "asset_id": identificador del activo financiero.
    - "asset_class": clase del activo financiero. Puede ser "Stock" (el activo financiero es entonces una acción) o "Future" (el activo financiero es entonces un futuro).

7.2.3 Tablas sobre gestión de carteras

- Tabla "portfolios": incluye la información de configuración de cada cartera financiera creada por un usuario. Estos son sus campos de información:
    - "id": identificador de la cartera financiera.
    - "name": nombre puesto por el usuario propietario.
    - "user_id": identificador del usuario propietario.

    - “start_dt”: fecha de creación de la cartera.
    - “end_dt”: fecha de eliminación/fin de la cartera.
    - “starting_cash”: capital inicial de la cartera.
    - “current_cash”: capital actual de la cartera.
    - “currency”: moneda en la que está denominada la cartera.
    - “shared”: si el usuario ha compartido la cartera o no.
- Tabla “positions”: contiene la información de cada posición existente. Estos son sus campos de información:
    - “id”: identificador de la posición.
    - “user_id”: identificador del usuario.
    - “asset”: ticker del activo financiero de la posición.
    - “current_price”: último precio del activo financiero.
    - “current_dt”: fecha del último precio del activo financiero.
    - “buy_quantity”: número de contratos comprados.
    - “sell_quantity”: número de contratos vendidos.
    - “avg_bought”: precio medio de compra de los contratos.
    - “avg_sold”: precio medio de venta de los contratos.
    - “buy_commission”: cantidad en comisiones pagadas en operaciones de compra.
    - “sell_commission”: cantidad en comisiones pagadas en operaciones de venta.
- Tabla “position_portfolio”: tabla que relaciona cada posición de la tabla “positions” con su correspondiente cartera financiera de la tabla “portfolio”. Estos son sus campos de información:
    - “id”: identificador de la fila.
    - “portfolio_id”: identificador de la cartera financiera.
    - “position_id”: identificador de la posición.
- Tabla “transactions”: incluye toda la información de todas las operaciones de compra y de venta efectuada durante la gestión de carteras.
    - “id”: identificador de la transacción.

- "user_id": identificador del usuario que ha realizado dicha transacción.
- "asset": ticker del activo financiero sobre el que se ha realizado la transacción.
- "quantity": cantidad de contratos comprados o vendidos en la transacción.
- "price": precio del activo financiero al que se ha completado la transacción.
- "dt": fecha de la transacción.
- "commission": comisión de la transacción.
- "position_id": identificador de la posición al que está asociada la transacción.

7.2.4 Tablas sobre otros servicios de trading

- Tabla "backtesting_manual": incluye la información de los "backtesting" manuales creados y ejecutados por los usuarios. Estos son sus campos de información:
  - "id": identificador del "backtesting" manual.
  - "name": nombre puesto por el usuario propietario.
  - "user_id": identificador del usuario propietario.
  - "starting_cash": capital inicial de la cartera del "backtesting" manual.
  - "currency": moneda en la que están denominados los datos del campo de información "starting_cash".
  - "saved": si el usuario lo ha guardado o no para su posterior recuperación.
- Tabla "backtesting_manual_operations": incluye las operaciones que ha añadido cada usuario durante la ejecución del "backtesting" manual. Estos son sus campos de información:
  - "id": identificador de la operación.
  - "backtesting_manual_id": identificador del "backtesting" manual.
  - "quantity": beneficio o pérdida de la operación.
  - "date": fecha de la operación.
- Tabla "journal": incluye toda la información del diario de trading de cada usuario. Estos son sus campos de información:
  - "id": identificador del comentario.
  - "user_id": identificador del usuario que ha realizado el comentario.

- "create_dt": fecha en la que el usuario ha realizado el comentario.
- "date": fecha a la que hace referencia el comentario.
- "text": texto del comentario.

7.2.5 Tablas sobre administrar datos

- Tabla "asset_class": contiene todos los activos financieros del sistema sean de la clase que sean. Estos son sus campos de información:
  - "id": identificador del activo financiero.
  - "asset_class": clase del active financiero. Puede ser "Stock" (el activo financiero es entonces una acción) o "Future" (el activo financiero es entonces un futuro).
- Tabla "stocks": incluye los datos de las especificaciones de los contratos de las acciones. Estos son sus campos de información:
  - "id": identificado del activo financiero.
  - "ticker": ticker del activo financiero.
  - "company_name": nombre de la compañía.
  - "market": mercado en el que cotiza la acción.
  - "currency": moneda en la que está denominada la acción.
  - "metric": valor de la métrica que se utiliza en la funcionalidad "Backtesting" por grupos.
- Tabla "stock_prices": incluye los datos diarios de los precios de las acciones. Estos son sus campos de información:
  - "id": identificador de la fila.
  - "stock_id": identificador del activo financiero.
  - "date": fecha del dato.
  - "open": precio de apertura del día.
  - "high": máximo precio del día.
  - "low": mínimo precio del día.
  - "close": precio de cierre del día.

    - "adjusted_close": precio de cierre del día ajustado.
    - "volumen": volumen de operaciones realizadas en el día.
- Tabla "futures": incluye los datos de las especificaciones de los contratos de los futuros. Estos son sus campos de información:
    - "id": identificador del activo financiero.
    - "ticker": ticker del activo financiero.
    - "futures_name": nombre del contrato.
    - "currency": moneda en la que está denominado el futuro.
- Tabla "futures_prices": incluye los datos diarios de los precios de los futuros. Estos son sus campos de información:
    - "id": identificador de la fila.
    - "futures_id": identificador del activo financiero.
    - "date": fecha del dato.
    - "open": precio de apertura del día.
    - "high": máximo precio del día.
    - "low": mínimo precio del día.
    - "close": precio de cierre del día.
    - "adjusted_close": precio de cierre del día ajustado.
    - "volumen": volumen de operaciones realizadas en el día.

Capítulo 8 - Diseño

En este capítulo se va a realizar una descripción de las principales funcionalidades implementadas y el diseño que se ha seguido para cada una de ellas.

8.1 Diseño de la interfaz visual

Para la interfaz visual, se ha buscado un diseño que fuera elegante y minimalista al mismo tiempo. Con tal objetivo en mente, se ha hecho uso de la plantilla "Portal" [21]. Esta es una plantilla gratuita de Bootstrap 5 para el desarrollo web de paneles centrales, si bien cuenta con más de 10 diseños de páginas distintos. Es completamente "responsive", utiliza HTML5 y CSS3, y está construida sobre Bootstrap 5. No utiliza JQuery, lo que, además, la hace una plantilla más ligera ("lightweight"). A continuación, la Ilustración 22 contiene una imagen con una de las páginas de la demo:

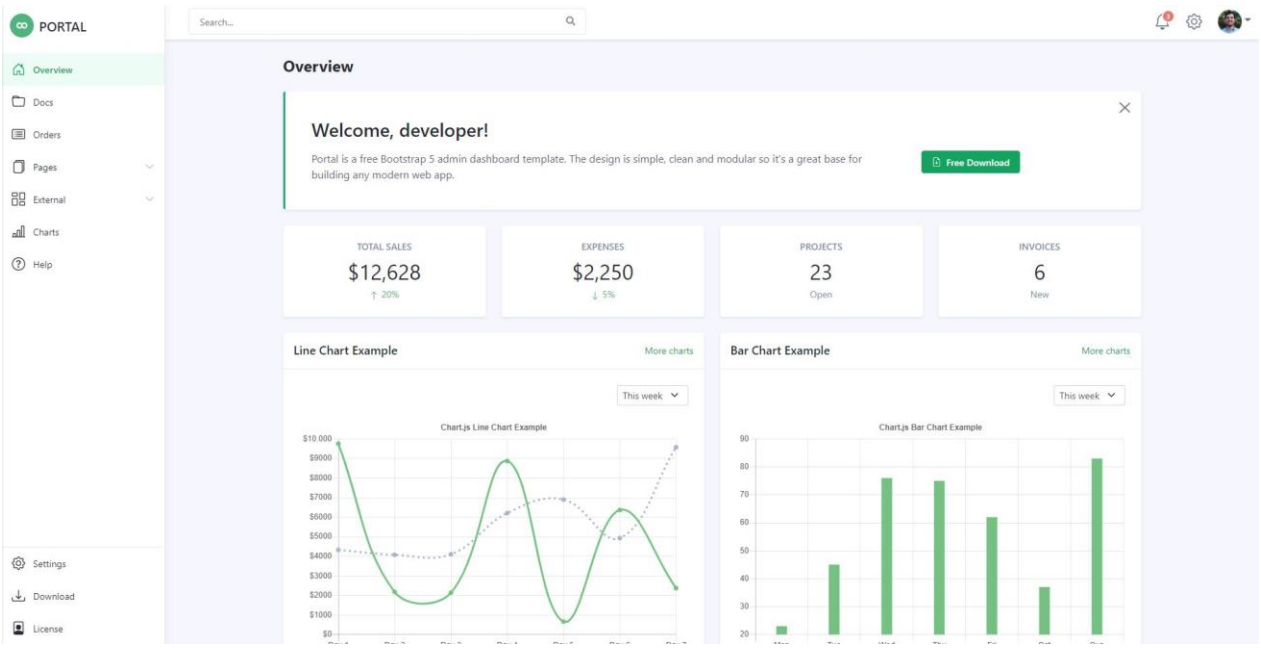


Ilustración 22. Plantilla "Portal"

Como se puede observar, el uso de esta plantilla permite el diseño de una interfaz visual minimalista en la que el principal color del tema es el verde. La forma en la que se ha trabajado en este proyecto a la hora del diseño de la interfaz visual ha sido modificando, a raíz de esta

plantilla, los distintos componentes de esta y ajustándola a las necesidades de las funcionalidades de la aplicación aquí desarrollada.

Aunque esta plantilla permite la creación de gráficos utilizando la librería de JavaScript conocida como Chart.js, también se ha hecho uso de otras herramientas para la creación, configuración y diseño de estos. En concreto, se ha utilizado la librería de visualización de Python llamada Matplotlib para la creación de algunos gráficos y, en algunas ocasiones, para su presentación a través de la interfaz visual, se ha hecho uso de la librería mpld3. Esta última librería de Python permite la creación de gráficos para el desarrollo web.

8.2 Funcionalidad de la aplicación

Este apartado está dedicado a la descripción del diseño de las funcionalidades implementadas durante el desarrollo de esta aplicación web.

8.2.1 Dashboard

Esta es la pantalla principal del usuario registrado. Aquí, el usuario puede observar cuatro conjuntos de objetos distintos. En primer lugar, el usuario tiene listados los tres últimos "backtesting" creados y guardados. En segundo lugar, el usuario puede observar las tres carteras creadas más recientemente. Finalmente, el usuario tiene acceso directamente a sus tres últimos "backtesting" manuales guardados con anterioridad y los tres últimos comentarios añadidos al diario de trading. En la Ilustración 23 se puede observar un ejemplo de esta vista.

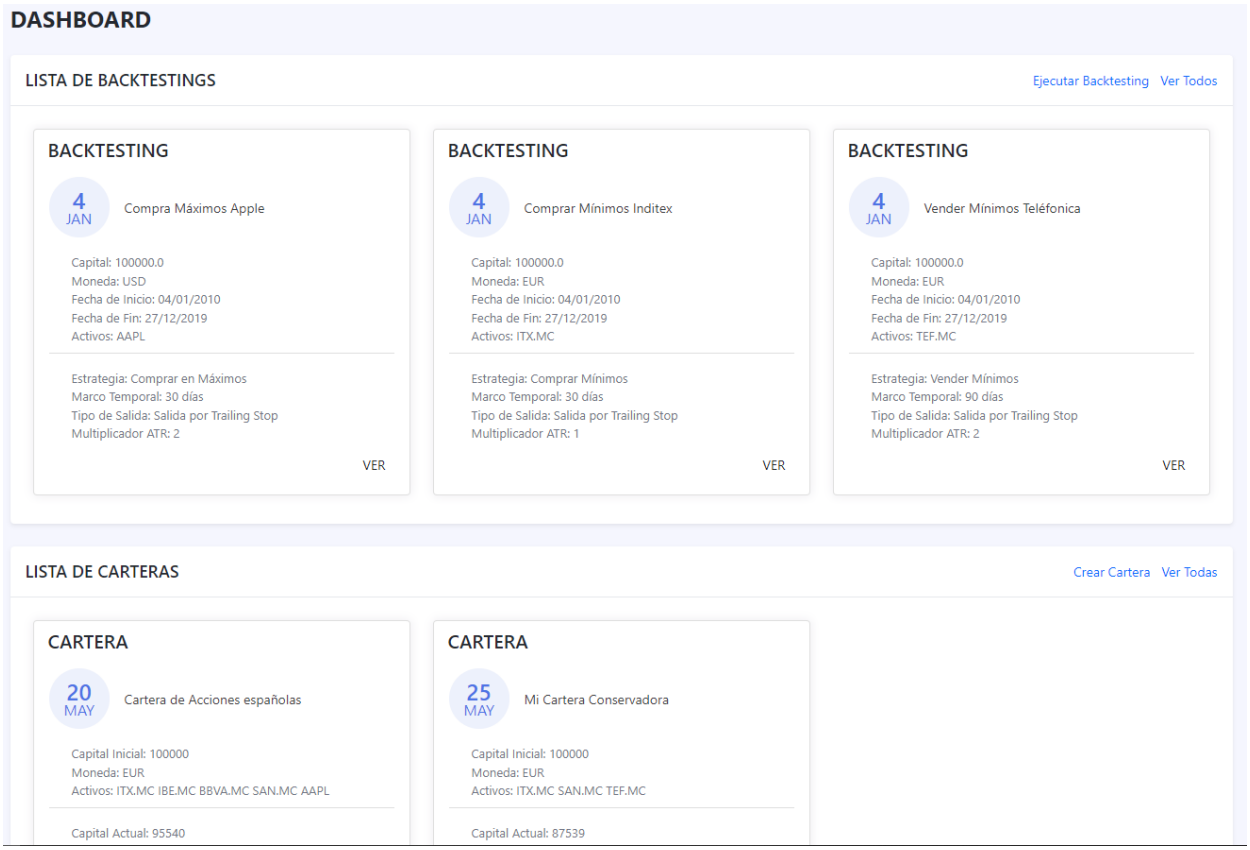


Ilustración 23. Vista Dashboard

Además, el usuario cuenta con una barra de navegación a través de la cual puede realizar la búsqueda de "backtesting" y carteras financieras cuyos usuarios propietarios hayan publicado previamente. Si el usuario selecciona "Backtesting" e introduce el ticker de un activo financiero, el sistema devuelve todos los "backtesting" públicos que se hayan realizado sobre dicho activo financiero. Si no encuentra ningún "backtesting" o el activo financiero no existe, muestra una búsqueda vacía. De la misma forma, si el usuario selecciona "Cartera" e introduce el ticker de un activo financiero, el sistema devuelve todas las carteras públicas que contengan dicho activo. Además, el usuario también puede buscar a otros usuarios. Esta funcionalidad resulta de mucha utilidad para el administrador, quien tiene permisos para modificar la información del resto de usuarios. Finalmente, el usuario también puede buscar activos financieros introduciendo el ticker correspondiente. En este caso, se mostrará una gráfica y una tabla de datos con los precios de dicho activo financiero.

8.2.2 Funcionalidad del "Backtesting"

La funcionalidad principal de la aplicación web aquí desarrollada es la de ejecutar el "backtesting" de una estrategia de trading basada en la compra o venta en el máximo o el mínimo del precio en un marco temporal determinado. Por lo tanto, para ejecutar dicho "backtesting", el usuario necesita primero crearlo con los datos necesarios y correctos. Una vez creado el "backtesting" sobre una estrategia de trading, y antes de su ejecución, la información de este quedaría conforme viene en la Ilustración 24.

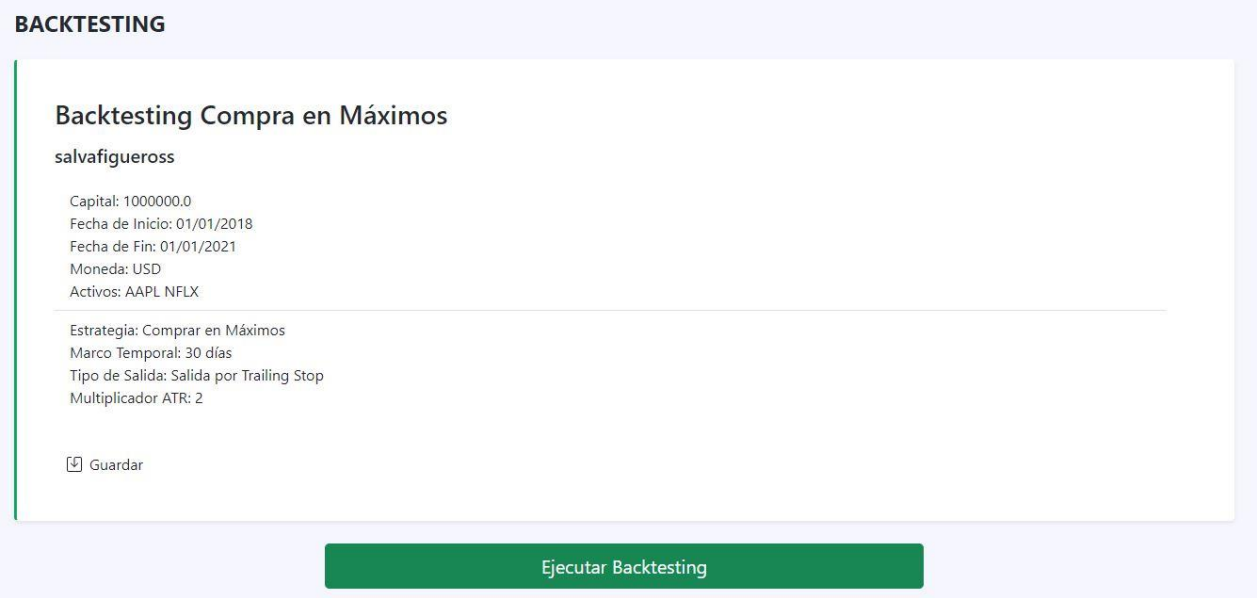


Ilustración 24. Vista Backtesting configurado

Una vez creado el "backtesting", el usuario puede guardarlo para su posterior recuperación, publicarlo o eliminarlo en caso de haberlo guardado previamente, y, para lo que está hecho principalmente, puede también ejecutarlo. El guardado y la publicación del "backtesting" creado se hacen a través de peticiones Ajax y la destrucción de este elimina cualquier registro suyo del sistema.

Ahora, el diseño y la codificación de la ejecución del "backtesting" requiere de una explicación en detalle.

En primer lugar, para la implementación del "backtesting", se ha seguido un diseño guiado por eventos [22]. Este tipo de diseño consiste en la implementación de un algoritmo en bucle que recibe eventos y responde a cada uno de estos eventos conforme corresponde. De

esta forma, es posible alcanzar la ilusión de que se están manejando respuestas en tiempo real, aunque el "backtesting" se esté ejecutando sobre datos históricos.

Esta forma de diseñar el "backtesting" tiene asociadas una serie de ventajas:

- En primer lugar, evita el **sesgo de anticipación**. Este consiste en la utilización de datos durante la ejecución del "backtesting" que no se habrían tenido durante el período simulado. Por lo tanto, un diseño guiado por eventos garantiza que la simulación ejecutada, en el momento de tomar las decisiones de compra y de venta, tan sólo cuenta con los datos que verdaderamente tendría disponibles.
- Un diseño guiado por eventos permite la **reusabilidad del código**. En concreto, el mismo código se puede utilizar tanto para la ejecución de un "backtesting" sobre datos históricos como para la implementación de una estrategia en tiempo real. Tan sólo es necesaria la realización de unos pocos cambios.

Para la implementación del diseño guiado por eventos justamente comentado, se han seguido las indicaciones encontradas en la serie de artículos de Quantstart dedicados al desarrollo de un "backtesting" guiado por eventos con Python [22]. Los componentes que lo conforman son los siguientes:

Evento [23]: Como su nombre indica, estas son las unidades sobre las que va a actuar el algoritmo cuando este los reciba. En esta aplicación web, se ha definido la clase "Event" como clase abstracta, de la cual heredan los distintos tipos de eventos. El tipo del evento que recibe el algoritmo es lo que determina cómo este actúa en consecuencia. A continuación, se definen los distintos tipos de eventos y sus correspondientes clases:

- "MarketEvent": Tipo de evento que se dispara cuando se reciben datos de mercado.
- "SignalEvent": Tipo de evento que se dispara cuando la estrategia de trading marca una señal de compra o de venta.

```
class SignalEvent(Event):

    def __init__(self, ticker, datetime, signal_type):
        self.type = 'SIGNAL'
        self.ticker = ticker
        self.datetime = datetime
        self.signal_type = signal_type
```

Ilustración 25. Clase SignalEvent

- "OrderEvent": Tipo de evento que se dispara después de recibir una señal de compra o de venta y para enviar una orden.

```
class OrderEvent(Event):

    def __init__(self, ticker, order_type, quantity, direction):
        self.type = 'ORDER'
        self.ticker = ticker
        self.order_type = order_type
        self.quantity = quantity
        self.direction = direction
```

Ilustración 26. Clase OrderEvent

- "FIllEvent": Tipo de evento que indica a qué precio y con qué coste se ejecutó una orden de compra o de venta.

```
class FillEvent(Event):

    def __init__(self, timeindex, ticker, exchange, quantity, direction, fill_cost, commission=None):
        self.type = 'FILL'
        self.timeindex = timeindex
        self.ticker = ticker
        self.exchange = exchange
        self.quantity = quantity
        self.direction = direction
        self.fill_cost = fill_cost

        if commission is None:
            self.commission = 0
        else:
            self.commission = commission
```

Ilustración 27. Clase FillEvent

Gestor de Datos [24]: La clase "HistoricDBDataHandler" recupera de la base de datos los datos históricos de los precios que se necesiten para la ejecución del "backtesting" y los prepara para su posterior procesamiento. En concreto, esta clase es la encargada de generar eventos de mercado. En la Ilustración 28 se puede ver un extracto del código escrito para la definición de esta clase.

```python
class HistoricDBDataHandler(DataHandler):

    def __init__(self, events, ticker_list, start_dt, end_dt):
        self.events = events
        self.ticker_list = ticker_list
        self.start_dt = start_dt
        self.end_dt = end_dt

        self.ticker_data = {}
        self.latest_ticker_data = {}
        self.continue_backtest = True

        self._get_data_db()



    def _get_data_db(self):
        comb_index = None
        for t in self.ticker_list:
            stock = Stock.get_stock_by_ticker(t)
            future = Futures.get_futures_by_ticker(t)

            if stock:
                self.ticker_data[t] = stock.get_stock_prices_dates(self.start_dt, self.end_dt)
            elif future:
                self.ticker_data[t] = future.get_futures_prices_dates(self.start_dt, self.end_dt)

            if stock or future:

                if comb_index is None:
                    comb_index = self.ticker_data[t].index
                else:
                    comb_index.union(self.ticker_data[t].index)

                self.latest_ticker_data[t] = []

        for t in self.ticker_list:
            self.ticker_data[t] = self.ticker_data[t].reindex(index=comb_index, method='pad').iterrows()

        self.start_date = comb_index[0]
```

Ilustración 28. Clase HistoricDBDatahandler

El método siguiente, "update_bars" (de la clase "HistoricDataDBHandler"), es el encargado de generar los eventos de mercado y para ello, ejecuta el método "_get_new_bar" (de la clase "HistoricDataDBHandler"). Es a través de la ejecución e interacción de estos dos métodos como se consigue evitar el sesgo de anticipación. La Ilustración 29 incluye la definición del método "_get_new_bar()" y la Ilustración 30 del método "update_bar".

```
def _get_new_bar(self, ticker):
    for b in self.ticker_data[ticker]:
        yield tuple([ticker, datetime.datetime.strptime(b[0].strftime('%Y-%m-%d %H:%M:%S'), '%Y-%m-%d %H:%M:%S'),
                     b[1][0], b[1][1], b[1][2], b[1][3], b[1][4], b[1][5]])
```

Ilustración 29. Método _get_new_bar()

```
def update_bars(self):
    for t in self.ticker_list:
        try:
            bar = self._get_new_bar(t).__next__()
        except StopIteration:
            self.continue_backtest = False
        else:
            if bar is not None:
                self.latest_ticker_data[t].append(bar)
    self.events.put(MarketEvent())
```

Ilustración 30. Método update_bars()

Estrategia [25]**:** Este componente contiene la lógica de la estrategia de trading a implementar y, por lo tanto, es el encargado de generar las señales de compra y de venta ("SignalEvent"). Como se ha comentado al inicio de la memoria, esta aplicación web ha tratado de implementar una estrategia de trading que se podía dividir conceptualmente en cuatro sub-estrategias. Por lo tanto, se ha creado una clase por cada una de las sub-estrategias a implementar. En la Ilustración 31 se observa el código del constructor de la clase "BuyMaxStrategy" (clase referente a la sub-estrategia basada en la compra en máximos):

```
class BuyMaxStrategy(Strategy):

    def __init__(self, time_frame, exit_trade, exit_configuration, strategy_id=None, bars=None, events=None):
        self.id = strategy_id
        self.bars = bars
        self.events = events
        self.time_frame = time_frame
        self.exit_trade = exit_trade
        self.exit_configuration = exit_configuration

        self.started = False

        if self.bars:
            self.set_bars()
```

Ilustración 31. Clase BuyMaxStrategy

El siguiente método, "calculate_signals" (de la clase "BuyMaxStrategy"), en la Ilustración 32, se ejecuta cada vez que el algoritmo recibe un evento de mercado y su lógica consiste en tratar de determinar las señales de compra y de venta generadas por las reglas de la estrategia de trading. Esta estrategia consiste en la compra en los máximos del precio. Por lo tanto, esta estrategia genera una señal de compra cuando el máximo del día evaluado para un activo financiero supera al máximo de un período temporal dado por el usuario. Esta es la condición "if bars[0][3] > self.maximum[t]". "bars[0][3]" contiene el valor del máximo del activo financiero en el día evaluado y "self.maximum[t]" contiene el máximo del activo financiero en el período temporal previo.

En cuanto a las señales de salida, estas dependen del tipo de salida elegido por el usuario. Cuando se haya elegido una salida temporal, se evalúa si han transcurrido o no el número de días elegidos por el usuario. Esta es la condición "if self.days_strategy[t] == int(self.exit_configuration * 0.5 * self.time_frame)". "self.days_strategy[t]" contiene el valor del número de días que han transcurrido desde que se compró dicho activo financiero y "self.exit_configuration * 0.5 * self.time_frame" es el número de días elegido por el usuario (self.exit_configuration puede ser "1", "2" o "3").

Por otro lado, si el usuario ha elegido una salida basada en un trailing stop, entonces se evalúa si el mínimo del precio del activo financiero en el día evaluado es menor o igual al precio determinado por el trailing stop. Esta es la condición "if bars[0][4] <= self.trailing_stop[t]. "bars[0][4]" contiene el valor del mínimo del activo financiero en el día evaluado y "self.trailing_stop[t]" contiene el valor del trailing stop del activo en cuestión.

```
def calculate_signals(self, event):

    if event.type == "MARKET":
        for t in self.ticker_list:
            bars = self.bars.get_latest_bars(t, N=1)
            if bars is not None and bars != []:
                self.num_bars[t] += 1

                if(self.num_bars[t] > self.time_frame):
                    self.started = True
                    if self.bought[t] == False:
                        if bars[0][3] > self.maximum[t]:
                            signal = SignalEvent(bars[0][0], bars[0][1], 'LONG')
                            self.events.put(signal)
                            self.bought[t] = True
                            self._update_trailing_stop(t, self.bars.get_latest_bars(t, N=2))
                    else:
                        self.days_strategy[t] += 1
                        if self.exit_trade == "exit_time":
                            if self.days_strategy[t] == int(self.exit_configuration * 0.5 * self.time_frame):
                                signal = SignalEvent(bars[0][0], bars[0][1], 'EXIT')
                                self.events.put(signal)
                                self.bought[t] = False
                                self.days_strategy[t] = 0
                        else:
                            if bars[0][4] <= self.trailing_stop[t]:
                                signal = SignalEvent(bars[0][0], bars[0][1], 'EXIT')
                                self.events.put(signal)
                                self.bought[t] = False

                            self._update_trailing_stop(t, self.bars.get_latest_bars(t, N=2))


                self._update_maximum(t, bars)
```

Ilustración 32. Método calculate_signals()

Cartera [26]**:** Este componente es el encargado de recibir las señales de trading ("SignalEvent") y generar órdenes en consecuencia ("OrderEvent"). Es en este componente donde cabría codificar la gestión del riesgo. Además, una vez generadas las órdenes de compra o de venta, este componente recibe eventos del tipo "FillEvent" y actualiza las posiciones mantenidas en la cartera de acuerdo. La implementación de este componente se ha realizado a través de la clase "PortfolioBacktesting". Esta clase lleva un seguimiento de todas las posiciones y de la evolución de estas durante todo el "backtesting". La Ilustración 33 contiene un extracto de código con la definición de esta clase.

```python
class PortfolioBacktesting(object):

    def __init__(self, bars, events, start_date, starting_cash=100000.0, currency="USD"):
        self.bars = bars
        self.events = events
        self.ticker_list = self.bars.ticker_list
        self.start_date = start_date
        self.starting_cash = starting_cash
        self.currency = currency

        #Positions
        self.all_positions = self.construct_all_positions()
        self.current_positions = dict( (k,v) for k, v in [(t, 0) for t in self.ticker_list] )

        #Holdings
        self.all_holdings = self.construct_all_holdings()
        self.current_holdings = self.construct_current_holdings()

        self.all_returns = self.construct_all_returns()
        self.latest_positions = dict( (k,v) for k, v in [(t, 0) for t in self.ticker_list] )

        #Operations
        self.all_operations = pd.DataFrame(columns=["timeindex", "asset", "direction", "price", "quantity"])
```

Ilustración 33. Clase PortfolioBacktesting

Para la generación de las órdenes una vez obtenida la señal de compra o venta, se asigna el capital de la cartera de forma equitativa entre todos los activos financieros tal como se puede observar en la Ilustración 34.

```python
def generate_order(self, signal):
    order = None

    ticker = signal.ticker
    direction = signal.signal_type

    no_position = sum(1 for v in self.current_positions.values() if v == 0)
    cur_quantity = self.current_positions[ticker]
    quantity = 0
    order_type = 'MKT'

    if direction == 'LONG' and cur_quantity == 0:
        allocation = (self.current_holdings["cash"]/(no_position))
        mkt_quantity = int(allocation / self.bars.get_latest_bars(ticker)[0][5])  # Close price
        quantity = mkt_quantity
        order = OrderEvent(ticker, order_type, mkt_quantity, 'BUY')
    if direction == 'SHORT' and cur_quantity == 0:
        allocation = (self.current_holdings["cash"]/(no_position))
        mkt_quantity = int(allocation / self.bars.get_latest_bars(ticker)[0][5])  # Close price
        quantity = mkt_quantity
        order = OrderEvent(ticker, order_type, mkt_quantity, 'SELL')

    if direction == 'EXIT' and cur_quantity > 0:
        quantity = abs(cur_quantity)
        order = OrderEvent(ticker, order_type, abs(cur_quantity), 'SELL')
    if direction == 'EXIT' and cur_quantity < 0:
        quantity = abs(cur_quantity)
        order = OrderEvent(ticker, order_type, abs(cur_quantity), 'BUY')

    self.all_operations["quantity"].loc[self.all_operations["timeindex"] == signal.datetime] = quantity

    return order
```

Ilustración 34. Método generate_order()

Ejecutor de Órdenes [27]**:** Este componente recibe órdenes ("OrderEvent") y simula una conexión a un bróker. Si se realizara la implementación de la ejecución de una estrategia en vivo y en real, entonces este componente sería el encargado de conectarse al bróker para realizar las transacciones.

```
class SimulatedExecutionHandler(ExecutionHandler):

    def __init__(self, events):
        self.events = events

    def execute_order(self, event):

        if event.type == 'ORDER':
            fill_event = FillEvent(datetime.datetime.utcnow(), event.ticker,
                                   '', event.quantity, event.direction, None)
            self.events.put(fill_event)
```

Ilustración 35. Clase SimulatedExecutionHandler

Por último, todos estos componentes forman parte de la clase "Backtesting". Esta clase contiene un método, "execute_backtesting", que se encarga de ejecutar el algoritmo que recibe y gestiona los eventos en bucle [22]. En la Ilustración 36 se puede observar la lógica de dicho algoritmo.

```
def execute_backtesting(self):
    #Prepare Backtesting
    self.events = queue.Queue()
    self.bars = HistoricDBDataHandler(self.events, self.backtesting_assets.ticker_list, self.start_dt, self.end_dt)
    self.port = PortfolioBacktesting(self.bars, self.events, self.bars.get_start_date(), self.starting_cash, self.currency)
    self.simulator = SimulatedExecutionHandler(self.events)
    self.strategy.events=self.events
    self.strategy.set_bars(self.bars, self.backtesting_assets.ticker_list)

    #Run Backtesting
    while True:
        if self.bars.continue_backtest == True:
            self.bars.update_bars()
        else:
            break

        while True:
            try:
                event = self.events.get(False)
            except queue.Empty:
                break
            else:
                if event is not None:
                    if event.type == 'MARKET':
                        self.strategy.calculate_signals(event)
                        self.port.update_timeindex(event)

                    elif event.type == 'SIGNAL':
                        self.port.update_signal(event)

                    elif event.type == 'ORDER':
                        self.simulator.execute_order(event)

                    elif event.type == 'FILL':
                        self.port.update_fill(event)
```

Ilustración 36. Algoritmo del backtesting

Una vez ejecutado el "backtesting", el usuario ya puede ver los resultados arrojados por este. Además, el usuario también tiene la opción de ver, para cada uno de los activos financieros incluidos en el "backtesting", las correspondientes señales de entrada y de venta generadas por la estrategia. La Ilustración 37 contiene un ejemplo de este tipo de gráfico. Para la generación de este, además, se ha hecho uso de la librería mpld3. Esta ha permitido la generación del código HTML necesario para la creación de gráficos interactivos. Esto resulta muy conveniente, pues en función del período temporal escogido, puede llegar a ser necesario hacer zoom en el gráfico para observar con mayor claridad los momentos exactos de compra y de venta del activo en cuestión.

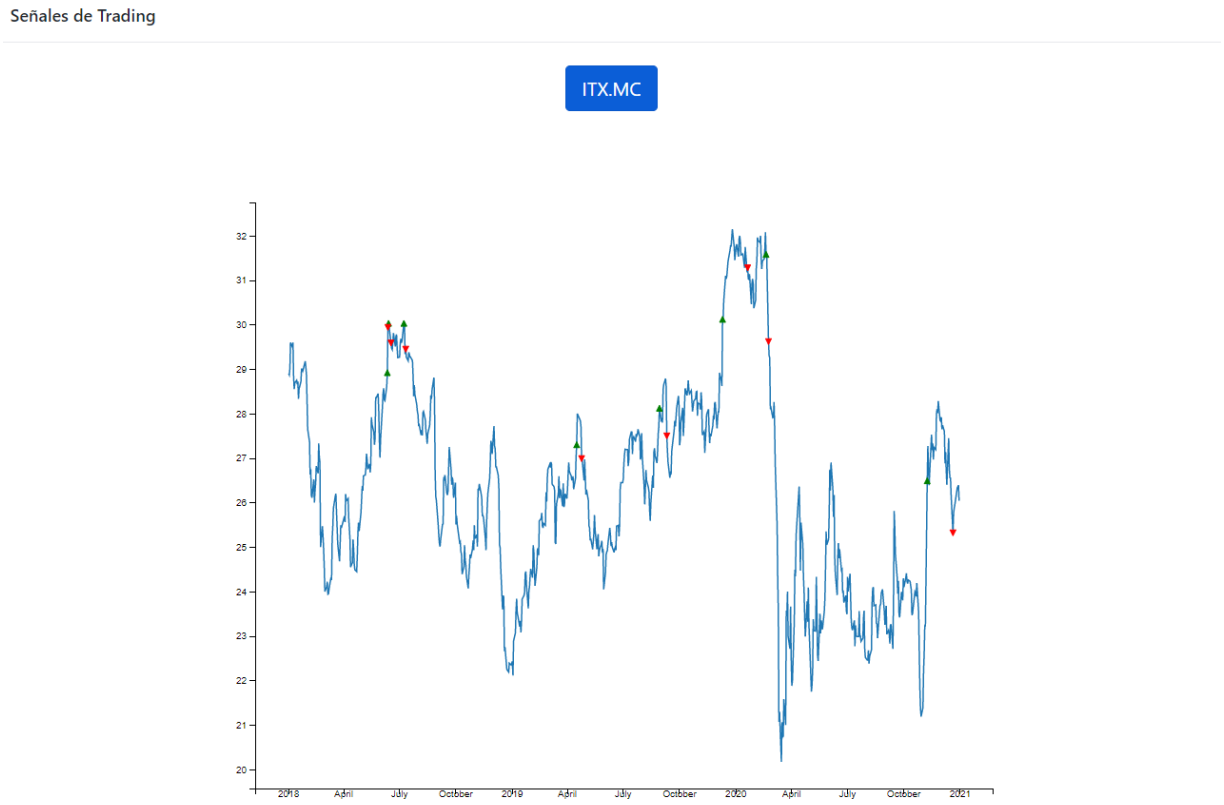


Ilustración 37. Gráfico con las señales de trading

La funcionalidad del "backtesting" por grupos trata igualmente de ejecutar el "backtesting" de una estrategia de trading de la misma forma y siguiendo el mismo diseño que se acaba de comentar. La diferencia reside en que, previamente, se divide el conjunto de activos financieros en dos grupos en función de la métrica seleccionada por el usuario. Esta

métrica puede ser la capitalización del mercado o el volumen. De tal forma, que, si el usuario ha elegido la primera, se incluyen los activos financieros con baja capitalización de mercado en un conjunto y a los que tiene una capitalización de mercado grande se les incluye en el otro conjunto. Y se sigue la misma lógica en el caso de que el usuario hubiera elegido el volumen como métrica. Una vez separados los activos financieros en dos grupos, se crea y ejecuta un "backtesting" por cada uno de los dos conjuntos existentes.

8.2.3 Funcionalidad de la gestión de carteras

La Ilustración 38 incluye la vista de la principal funcionalidad de la gestión de carteras.

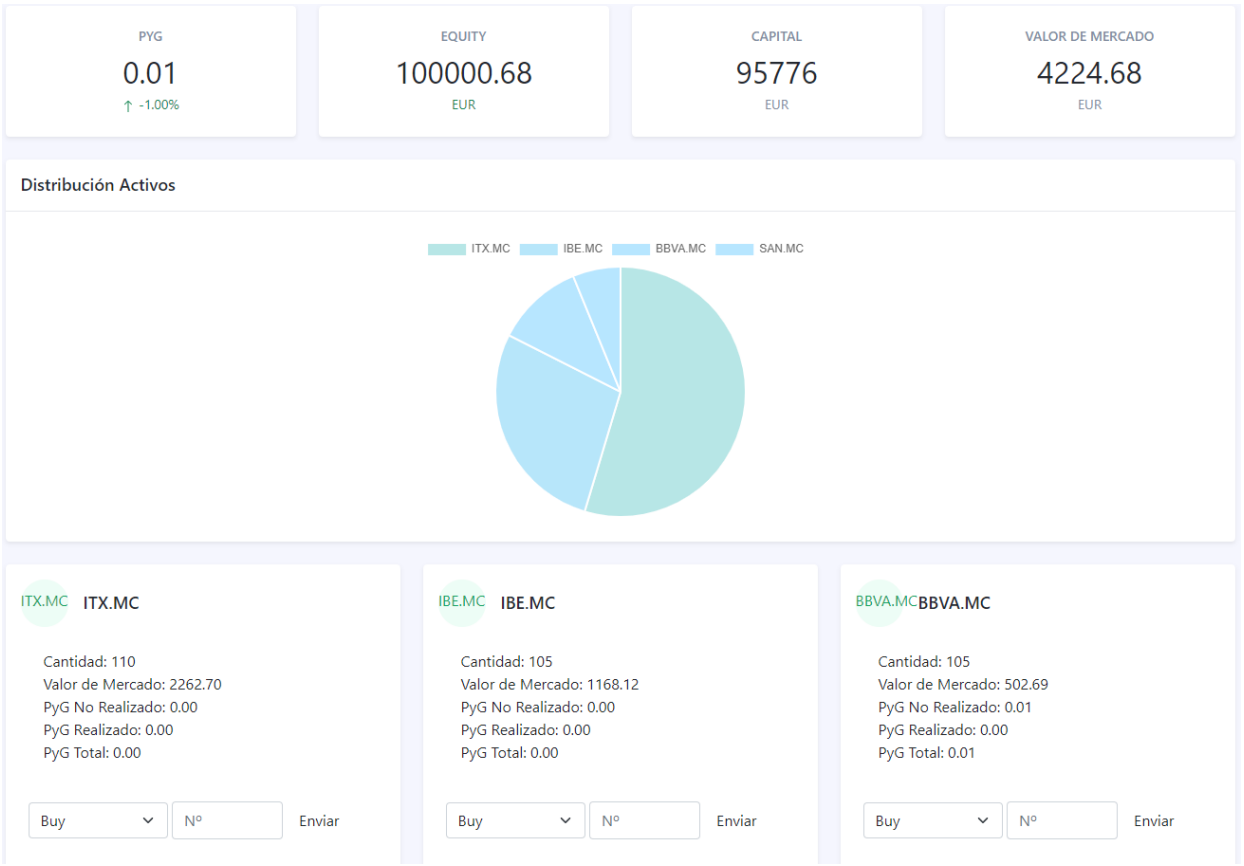


Ilustración 38. Vista gestión de carteras

La principal funcionalidad de la gestión de carteras es la creación y mantenimiento en el tiempo de carteras financieras. La estructura que se ha seguido para la implementación de esta funcionalidad concreta está compuesta por los siguientes componentes:

- Transacción: Una transacción es una operación de compra o de venta de un activo financiero, a un precio determinado y en una fecha concreta.
- Posición: Una posición hace referencia al número de contratos que mantiene un usuario sobre un activo financiero concreto. Cuando un usuario no ha comprado ni vendido un activo financiero, se dice que ese usuario no mantiene ninguna posición en dicho activo. Ahora, si ese mismo usuario realiza por primera vez una transacción sobre ese activo financiero, entonces se crea una posición en dicho activo financiero. De la misma forma, si el usuario ejecuta una nueva transacción que le hace volver a tener cero contratos de dicho activo, entonces se elimina dicha posición.
- Cartera: Una cartera financiera está compuesta por un conjunto de posiciones en varios activos financieros. De tal forma, que los resultados de la cartera serán consecuencia de los resultados que arrojen todas sus posiciones a nivel individual.

Una vez definida la estructura a nivel conceptual, y siguiendo las indicaciones encontradas en la serie de artículos de QuantStart dedicados a la gestión de carteras con Python [28], se definen las siguientes clases para su implementación:

"Transaction": Clase encargada de englobar la lógica en lo que respecta a las transacciones. La transacción se ejecuta al último precio del activo financiero en cuestión. Para ello, se ejecuta el método "get_last_quote", que obtiene dicho precio a través de técnicas de web scraping y el uso de la librería "pandas_datareader". La Ilustración 39 incluye un extracto del código para la definición de la clase "Transaction" y la Ilustración 40 indica cómo se ha hecho para crear una transacción.

```
class Transaction(object):

    def __init__(self, user_id, asset, quantity, price, dt, commmission=0.0, position_id = None):
        self.user_id = user_id
        self.asset = asset
        self.quantity = quantity
        self.price = price
        self.dt = dt
        self.commission = Decimal(commmission)
        self.position_id =  position_id
```

Ilustración 39. Clase Transaction

```python
@staticmethod
def create_transaction(user_id, asset, quantity):
    # 1. Get Asset's Actual Price
    price = Stock.get_last_quote(asset)
    # 2. Get dt of Asset's Actual Price
    dt = datetime.datetime.now()
    # 3. Get commission
    # 4. Create Transaction Object
    transaction = Transaction(user_id, asset, quantity, price, dt)
    # 5. Insert Transaction Object into db
    transaction = Transaction.insert(transaction)
    # 6. Returns Transaction Object
    return transaction
```

Ilustración 40. Código para crear una transacción

"Position" [28]**:** Clase encargada de englobar la lógica en lo que respecta a una posición. La Ilustración 41, la Ilustración 42 y la Ilustración 43 contienen métodos muy importantes de esta clase.

```python
@staticmethod
def open_from_transaction(transaction):
    asset = transaction.asset
    current_price = transaction.price
    current_dt = transaction.dt

    if transaction.quantity > 0:
        buy_quantity = transaction.quantity
        sell_quantity = 0
        avg_bought = current_price
        avg_sold = 0.0
        buy_commission = transaction.commission
        sell_commission = 0.0
    else:
        buy_quantity = 0
        sell_quantity = -1.0 * transaction.quantity
        avg_bought = 0.0
        avg_sold = current_price
        buy_commission = 0.0
        sell_commission = transaction.commission

    return Position(
        transaction.user_id,
        asset,
        current_price,
        current_dt,
        buy_quantity,
        sell_quantity,
        avg_bought,
        avg_sold,
        buy_commission,
        sell_commission
    )
```

Ilustración 41. Código para abrir una posición

```python
@staticmethod
def create_position(transaction):
    # 1. open_from_transaction: create Position Object from Transaction Object
    position = Position.open_from_transaction(transaction)
    # 2. insert Position Object into db
    position = Position.insert(position)
    # 3. return Position Object
    return position
```

Ilustración 42. Código para crear una posición

```python
def transact(self, transaction):
    if int(floor(transaction.quantity)) == 0:
        return


    if transaction.quantity > 0:
        quantity = transaction.quantity
        self.avg_bought = ((self.avg_bought * self.buy_quantity) + Decimal((quantity * transaction.price))) / (self.buy_quantity + quantity)
        self.buy_quantity += Decimal(quantity)
        self.buy_commission += Decimal(transaction.commission)

    else:
        quantity = Decimal(-1.0)*transaction.quantity
        self.avg_sold = ((self.avg_sold * self.sell_quantity) + (quantity * transaction.price)) / (self.sell_quantity + quantity)
        self.sell_quantity += quantity
        self.sell_commission += transaction.commission

    self.update_current_price(transaction.price, transaction.dt)
    self.current_dt = transaction.dt
```

Ilustración 43. Código para ejecutar una transacción

"Portfolio" [29]**:** Clase encargada de englobar la lógica en lo que respecta a una cartera financiera. En la Ilustración 44 se puede observar el código pensado para la creación de una cartera.

```python
@staticmethod
def create_portfolio(name, user_id, start_dt, starting_cash, currency):
    portfolio = Portfolio(name, user_id, start_dt, starting_cash = starting_cash, currency = currency)
    portfolio.current_cash = portfolio.starting_cash

    return Portfolio.insert(portfolio)
```

Ilustración 44. Código para crear una cartera

"PositionHandler" [30]**:** Clase encargada de gestionar las posiciones de una cartera. Guarda las posiciones en un diccionario ordenado y hace uso de la librería "forex_python" para

realizar las conversiones de moneda que sean necesarias (clase "CurrencyRates"). En la Ilustración 45 se puede observar la inicialización de todos estos atributos.

```
class PositionHandler(object):

    def __init__(self, currency="USD"):
        self.currency = currency
        self.positions = OrderedDict()
        self.c = CurrencyRates()
```

Ilustración 45. Clase PositionHandler

Como se puede observar en la Ilustración 46, el método "transact_position()" se encarga de modificar el diccionario de posiciones de acuerdo con la información de la transacción recibida.

```
def transact_position(self, portfolio_id, transaction):
    asset = transaction.asset
    if asset in self.positions:
        self.positions[asset].transact(transaction)
        Position.update(self.positions[asset])
        position_portfolio = PositionPortfolio(portfolio_id, self.positions[asset].id)
    else:
        position = Position.create_position(transaction)
        self.positions[asset] = position
        position_portfolio = PositionPortfolio(portfolio_id, position.id)
        position_portfolio = PositionPortfolio.insert(position_portfolio)

    transaction.position_id = self.positions[asset].id
    transaction = Transaction.update(transaction)

    if self.positions[asset].net_quantity == 0:
        del self.positions[asset]
```

Ilustración 46. Código para modificar una posición

Luego, se puede observar en la Ilustración 47 como las estadísticas de la cartera (en este caso el valor de mercado de la cartera) son, en última instancia, resultado de las estadísticas individuales de cada posición.

```python
def total_market_value(self):
    return sum(
        self.c.convert(PositionHandler.get_asset_currency(asset), self.currency, pos.market_value)
        for asset, pos in self.positions.items()
    )
```

Ilustración 47. Código para obtener el valor de mercado de la cartera

"PositionPortfolio": Clase encargada de acceder a la base de datos y extraer las relaciones entre las carteras y las posiciones.

Por lo tanto, para la creación de una cartera, es necesario relacionar todos estos componentes de la forma adecuada. Una vez se obtienen los datos del formulario enviado por el usuario a través de una petición POST, se realiza la validación de estos. Si la validación es exitosa, en primer lugar, se crea la cartera. Luego, para cada uno de los activos financieros incluidos en el formulario de envío, se crea una transacción y se realizan las correspondientes modificaciones en la cartera creada. En el extracto de código incluido en la Ilustración 48, se puede observar de una forma simplificada la implementación que se ha llevado a cabo:

```python
@views.route('/create-portfolio', methods=['GET', 'POST'])
def create_portfolio():
    if request.method == 'POST':
        name = request.form.get('name')
        starting_cash = Decimal(request.form.get('starting_cash'))
        currency = request.form.get('currency')
        assets = request.form.getlist('asset[]')
        quantities = request.form.getlist('quantity[]')

        if name and starting_cash and currency:
            user = User.search_user(session["user_name"])
            portfolio = Portfolio.create_portfolio(name, user.id, datetime.datetime.now(), starting_cash, currency)

            for asset, quantity in zip(assets, quantities):
                asset_object = Stock.get_stock_by_ticker(asset)
                if asset_object == False:
                    asset_object = Futures.get_futures_by_ticker(asset)

                if asset_object:
                    transaction = Transaction.create_transaction(user.id, asset, int(quantity))
                    portfolio.transact_asset(transaction)

            portfolio = Portfolio.update(portfolio)
            return redirect(url_for('views.view_portfolio', id = portfolio.id))

        else:
            flash('Los campos Nombre, Capital y Moneda son obligatorios.', category='error')

    return render_template("create_portfolio.html")
```

Ilustración 48. Código para la gestión de la petición del usuario de creación de carteras

Como se puede observar en la Ilustración 49, cada vez que un usuario visite una cartera, se actualizará el valor de mercado de cada activo que se mantenga y, por lo tanto, también se actualizarán el valor de mercado y otras estadísticas de la cartera a nivel global.

```
@views.route('/view-portfolio', methods=['GET'])
def view_portfolio():
    portfolio_id = request.args.get('id')
    if portfolio_id:
            portfolio = Portfolio.get_portfolio_by_id(portfolio_id)

            if portfolio:

                if ((portfolio.user_id == session["user_id"]) or portfolio.shared == True) and (portfolio.end_dt is None ):
                    portfolio.update_market_value_of_assets()

                    return render_template("portfolio2.html", portfolio=portfolio, user=User.get_user_by_id(portfolio.user_id))

    return redirect(url_for('views.home'))
```

Ilustración 49. Código para la gestión de la petición del usuario de ver una cartera

Para la modificación de la cartera, tan sólo es necesario crear, a raíz de los datos enviados de nuevo por el usuario a través de una petición POST, una transacción y modificar la cartera de acuerdo con dicha transacción. La Ilustración 50 sirve como ejemplo de ello.

```
@views.route('/modify-portfolio', methods=['GET', 'POST'])
def modify_portfolio():
    if request.method == 'POST':
        position_id = request.form.get('position_id')
        portfolio_id = request.form.get('portfolio_id')
        direction = int(request.form.get('direction'))
        quantity = Decimal(request.form.get('quantity'))
        asset = request.form.get('asset')

        user = User.search_user(session["user_name"])

        asset_object = Stock.get_stock_by_ticker(asset)
        if asset_object == False:
            asset_object = Futures.get_futures_by_ticker(asset)

        if asset_object:
            portfolio = Portfolio.get_portfolio_by_id(portfolio_id)
            transaction = Transaction.create_transaction(user.id, asset, quantity*direction)
            portfolio.transact_asset(transaction)
            portfolio = Portfolio.update(portfolio)
        else:
            flash('No existe el ticker del activo.', category='error')

        return redirect(url_for('views.view_portfolio', id = portfolio.id))


    return redirect(url_for('views.home'))
```

Ilustración 50. Código para la gestión de la petición del usuario de modificación de carteras

Finalmente, las posibilidades de compartir y eliminar la cartera han sido implementadas igualmente mediante peticiones POST.

Aparte de la funcionalidad concreta de creación y mantenimiento de carteras, también se ha implementado de forma complementaria una funcionalidad de optimización de carteras. En esta funcionalidad, un usuario introduce un capital y una lista de activos financieros y el sistema realiza una asignación óptima de ese capital entre los distintos activos financieros que muestra finalmente al usuario como se puede observar en la Ilustración 51.

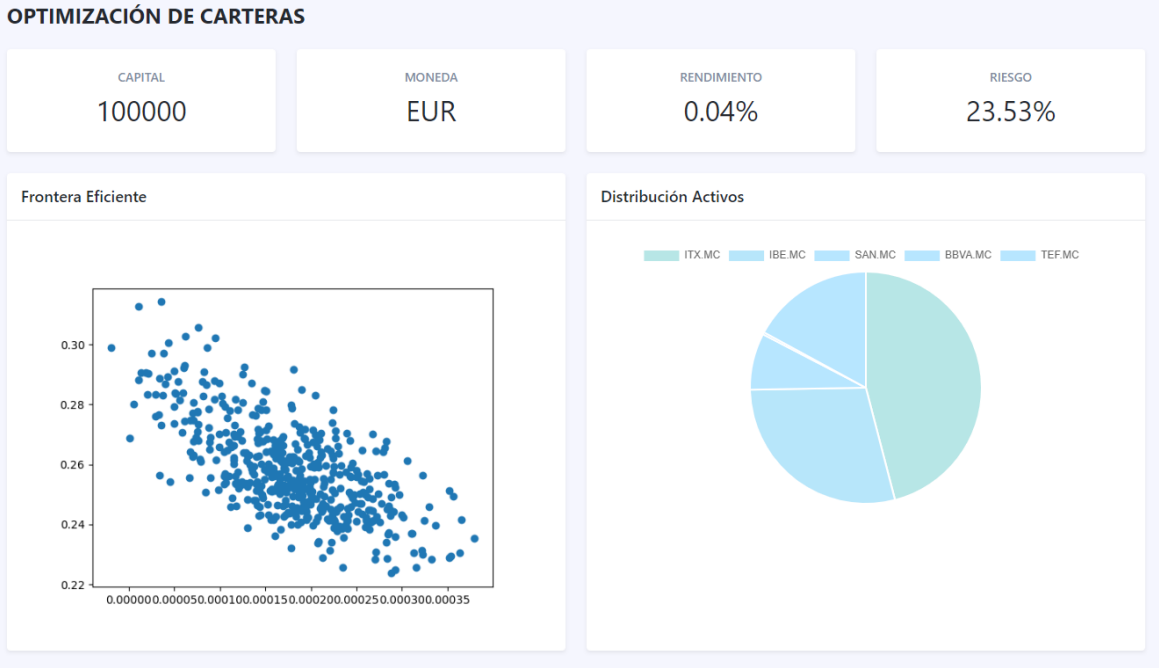


Ilustración 51. Vista Optimización de Carteras

Para la implementación de tal optimización, se ha seguido el método del análisis de varianza media, comúnmente conocido como teoría de porftolios moderna. Este consiste básicamente en la ejecución de un algoritmo por fuerza bruta en el que se crean tantas carteras como se quiera. Cada cartera se creará con una asignación de pesos entre los activos distinta. De tal forma, que aquella cartera que ofrezca mejor rendimiento será la devuelta al usuario y la frontera eficiente estará formada por aquel conjunto de carteras que ofrezca mejor relación rendimiento-riesgo (en el gráfico se pueden observar el rendimiento y el riesgo para todas y cada una de las carteras creadas durante la ejecución del algoritmo) [31]. En esta aplicación, el algoritmo por fuerza bruta ha sido implementado para la creación de hasta 500 carteras distintas.

La definición de este algoritmo puede verse en el extracto de código incluido en la Ilustración 52.

```python
@views_backtesting.route('/backtesting/optimization-portfolio', methods=['GET', 'POST'])
def optimization_portfolio():
    form = OptimizationBacktestingForm()

    if form.validate_on_submit():
        starting_capital = int(form.starting_capital.data)
        currency = form.currency.data
        allAssets = form.allAssets.data

        ticker_list = []
        if allAssets is True:
            list_stock = Stock.get_all_stocks()
            for stock in list_stock:
                ticker_list.append(stock.ticker)
        else:
            ticker_list = request.form.getlist('asset[]')

        df = Stock.get_df_prices(ticker_list)
        df = np.log(1 + df.pct_change())

        returns = []
        stds = []
        w = []

        for i in range(500):
            weights = weights_creator(df)
            returns.append(portfolio_return(weights, df))
            stds.append(portfolio_std(weights, df))
            w.append(weights)

        return render_template("backtesting_optimization_results.html",
                               ticker_list=ticker_list,
                               efficient_frontier=efficient_frontier_html(returns, stds),
                               starting_capital=starting_capital, currency=currency,
                               weight=w[returns.index(max(returns))],
                               return_user=max(returns),
                               std_user=stds[returns.index(max(returns))],
                               color_list=color_list(ticker_list))

    return render_template("backtesting_optimization_assets.html", form=form)
```

Ilustración 52. Código para la optimización de carteras

8.2.4 Funcionalidad de otros servicios de trading

Dos son los servicios de trading que se han implementado de forma adicional. En primer lugar, existe la posibilidad para el usuario de ejecutar un "backtesting" manual. La idea es que el usuario pueda añadir de forma manual los beneficios y las pérdidas que este obtiene en su operativa como trader. Cada vez que el usuario añade una ganancia o pérdida, el sistema calcula de nuevo las estadísticas más relevantes y las vuelve a mostrar a través de la interfaz visual incluida en la Ilustración 53. Y para poder obtener una experiencia de uso más suave, se ha realizado la implementación de esta funcionalidad a través de peticiones Ajax.

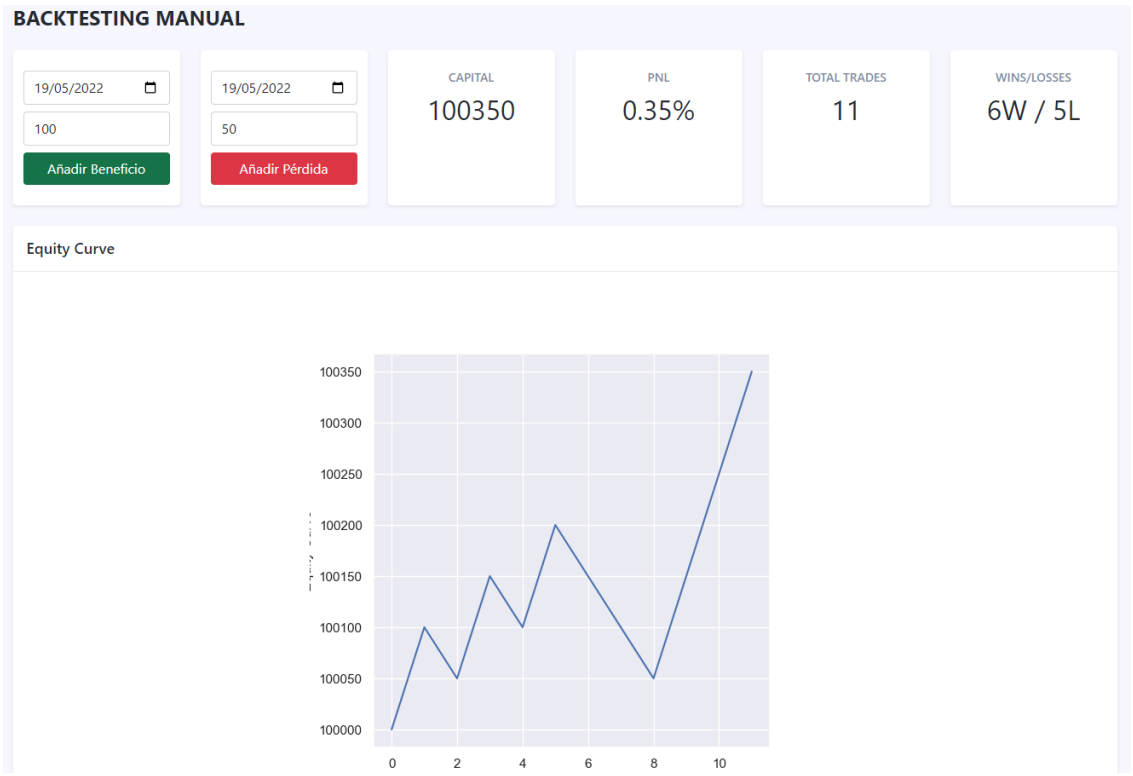


Ilustración 53. Vista Backtesting Manual

El gráfico representa la evolución del capital de la cartera tras cada beneficio o pérdida añadidos y se ha hecho uso de la librería matplotlib para su creación. Además, conviene comentar que, una vez completada con éxito la petición Ajax, es necesario cambiar la url asociada al gráfico para que este se actualice en el momento. Para ello, ha habido que asociar un timestamp a dicha url, pues refrescar únicamente la url sin cambiar el timestamp asociado no permitía actualizar el gráfico en vivo. La Ilustración 54 contiene el código utilizado para la gestión de la petición Ajax.

```
$(document).ready(function() {
    $("#form_profit").submit(function (e) {
        var url = "{{ url_for('views_other.backtesting_manual_process_profit') }}"; // send the form data here.
        $.ajax({
            type: "POST",
            url: url,
            data: $('#form_profit').serializeArray(), // serializes the form's elements.
            success: function (data) {
                console.log(data)  // display the returned data in the console.
                $('#capital_now').html(data.data.capital )
                $('#pnl').html(data.data.pnl + "%")
                $('#total_trades').html(data.data.total_trades)
                $('#wins_losses').html(data.data.total_wins + "W / " + data.data.total_losses + "L")


                var timestamp = new Date().getTime();
                var el = document.getElementById("chart_equity_curve");
                el.src = "/other/backtesting-manual-chart?backtesting_manual_id={{backtesting_manual.id}}&t=" + timestamp;
            }
        });
        e.preventDefault(); // block the traditional submission of the form.

        // Inject our CSRF token into our AJAX request.
        $.ajaxSetup({
            beforeSend: function(xhr, settings) {
                if (!/^(GET|HEAD|OPTIONS|TRACE)$/i.test(settings.type) && !this.crossDomain) {
                    xhr.setRequestHeader("X-CSRFToken", "{{ form_profit.csrf_token.value }}")
                }
            }

        })
    });
```

Ilustración 54. Código para la ejecución de peticiones AJAX

El otro servicio de trading implementado ha sido el diario de trading. Si el usuario no ha escrito ningún comentario en su diario, en el dashboard aparecerá como tal. De todas formas, siempre que este quiera puede crear un nuevo comentario accediendo al formulario correspondiente desde el panel central. En la Ilustración 55 se puede observar un ejemplo de esta funcionalidad.

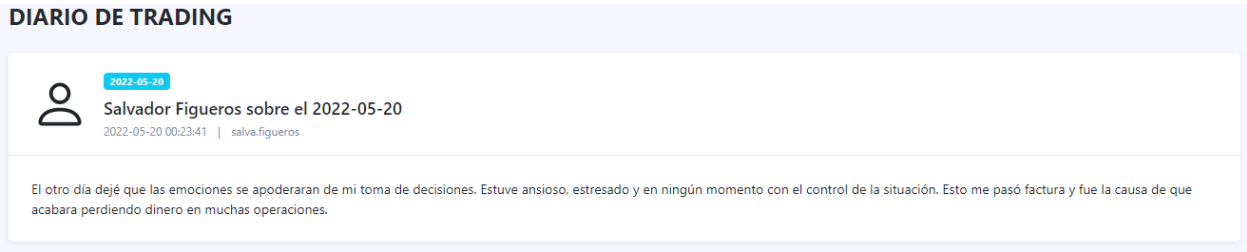


Ilustración 55. Vista Diario de Trading

8.2.5 Funcionalidad de la administración de los datos históricos

El administrador del sistema puede subir datos históricos de los precios de los activos financieros a la base de datos. Para la obtención de estos datos, se han utilizado técnicas de web scraping. El administrador tan sólo tiene que completar un formulario indicando los datos que quiere subir al sistema. Una vez enviado, el sistema obtiene los datos de los precios de las acciones de Yahoo! Finance. Para ello, se hace uso del método "read_csv" de la librería pandas tal como se puede observar en la Ilustración 56. A este método hay que pasar como parámetro la url de Yahoo! Finance que permita la descarga de un archivo "csv". Este archivo contiene los datos con los precios de las acciones que el administrador quiere subir al sistema. Finalmente, en caso de introducir una url incorrecta, ya sea por el envío de un formulario con datos erróneos por parte del administrador o por cualquier otro motivo, el sistema no permite la realización de la tarea e informa al usuario de la imposibilidad de completar esta con éxito.

```
query_string = f'https://query1.finance.yahoo.com/v7/finance/download/
                    {self.ticker}?
                    period1={period1}&
                    period2={period2}&
                    interval=1d&
                    events=history&
                    includeAdjustedClose=true'
df = pd.read_csv(query_string)
```

Ilustración 56. Código para la obtención de datos de Yahoo! Finance

Además, el sistema detecta si los datos que el administrador quiere subir ya existen o no, y solo realiza la inserción en la base de datos de aquellos que no existan.

Capítulo 9 - Conclusiones y trabajo futuro

9.1 Conclusiones

En este proyecto, se ha realizado el desarrollo de una aplicación web que permite la ejecución del "backtesting" de una estrategia basada en la compra o venta en máximos o mínimos del precio de un activo financiero. La implementación de esta funcionalidad se ha hecho siguiendo un diseño guiado por eventos. Ello hace que el usuario de la aplicación pueda correr su "backtesting" con las garantías suficientes y sin caer en el sesgo de anticipación.

Además, su implementación permite una gran escalabilidad en términos del número de estrategias de trading que se pueden añadir al sistema. Tan sólo haría falta crear una nueva clase con el nombre de la estrategia que se quiere añadir e implementar el método encargado de generar las señales de esta.

El objetivo principal de esta aplicación era el desarrollo del "backtesting", sin embargo se han implementado otras funcionalidades que permiten al usuario realizar un seguimiento de su operativa como trader. Estas son la gestión de carteras, el "backtesting" manual, el diario de trading y la administración de los datos históricos.

Por lo tanto, la aplicación web aquí desarrollada conforma un conjunto de herramientas muy útiles para invertir en los mercados financieros.

9.2 Trabajo futuro

La aplicación puede ser mejorada de diversas formas. En este sentido, se plantean las siguientes líneas de trabajo futuro:

- **Añadir más estrategias**: El diseño guiado por eventos permite fácilmente codificar nuevas estrategias y añadirlas para que el usuario pueda ejecutarlas. Añadiendo nuevas estrategias al sistema, se conseguiría incrementar la riqueza y el valor añadido aportados por la aplicación.
- **Trading en vivo**: El diseño guiado por eventos permite reutilizar gran parte de su estructura para el trading en vivo. La dificultado adicional tan solo estaría en la obtención y gestión en vivo de los datos de los precios de los activos financieros. Por lo tanto, una funcionalidad tan interesante como esta podría implementarse con no mucha dificultad.
- **Eficiencia del backtesting:** En función del periodo temporal escogido y el número de activos financieros sobre el que correr el "backtesting", este puede llegar a tardar una

suma considerable de tiempo. La ejecución de esta tarea podría verse reducida en una cantidad no poco importante si se realizara una implementación multiproceso o multihilo de esta.

- **Optimización de carteras:** Aparte del método de análisis de varianza media que se ha utilizado para implementar la optimización de carteras, también existen otros métodos de optimización distintos (HRP, mCVAR…). La utilidad de la aplicación incrementaría mucho si se pudieran implementar otros métodos de optimización y el usuario pudiera elegir cuál quiere utilizar para llevar a cabo la tarea.
- **Añadir más clases de activos financieros:** Actualmente, esta aplicación cuenta sólo con la posibilidad de gestionar acciones y futuros. Nuevas clases de activos, como bonos, criptomonedas o monedas (fórex), podrían ser añadidas al sistema. Esto daría lugar a una aplicación más amplia y extensa.
- **Aplicación del Aprendizaje Automático:** La aplicación de técnicas de Aprendizaje Automático al mundo financiero es cada vez mayor. Tanto para una gestión adecuada del riesgo como para la configuración eficiente de una estrategia de trading, estas técnicas podrían aportar una ventaja competitiva muy importante frente al resto de herramientas similares.

Chapter - Conclusions and future work

Conclusions

This project has developed a web application that allows the users to run the backtesting of a trading strategy based on the buying or selling at the highs or lows of an asset's price. The backtester has been built following an Event-Driven system. This allows the user to avoid the look-ahead bias when running the simulation.

Its implementation also allows for a rapid scalability in terms of the number of trading strategies than can be added to the system. It would only be necessary to create a new class with the name of the strategy to be added and to implement the method responsible for generating the trading signals.

The main goal of this web application was the development of the backesting. Nevertheless, other functionalities which allow the user to keep track of his trading have also been implemented. These are the functionalities related to portfolio management, backtesting by hand, the trading journal, and the handling of historical data.

Therefore, the web application here developed is made up of a set of tools which are very useful for investing in financial markets.

Future work

It is possible to improve the application in several ways. In that sense, these are the following lines of work to focus in the future:

- **Add more strategies:** The Event-Driven system makes it very easy to code new strategies and add them to the system so that the user can run them. By doing this, the application would be of more use for a trader and would, therefore, gain in added value.
- **Live trading:** It is also possible to reuse most of the code of the Event-Drive backtester for the implementation of live trading. It would only be necessary to handle the feed of live data.
- **Backtesting efficiency:** Depending on the period and the number of assets on which to run the backtesting, this could take a vast amount of time to be completed. The execution of this task could be greatly improved if multiprocessing or multithreading were included in the development of such.

- **Portfolio optimization:** Apart from the mean variance analysis method that has been used in the web application for portfolio optimization, there are other different optimization methods that can be used (HRP, mCVAR…). The functionality of the application would be enrichened if the user could choose which method he wants to use to complete this task.
- **Add more asset classes:** This project has only considered the possibility of investing in equities and futures, but other asset classes, such as bonds, cryptocurrencies, and currencies (forex), could be added to the system. This would create a more ample and thorough application.
- **Machine Learning:** The application of Machine Learning techniques to financial markets is greatly increasing in popularity. These techniques, which could be used for both risk management and the development of successful strategies, could provide the application with a competitive advantage over the rest of similar tools in the market.

BIBLIOGRAFÍA

[1] Fidelity, «Fidelity,» [En línea]. Available: https://www.fidelity.com/learning-center/trading-investing/technical-analysis/technical-indicator-guide/atr.

[2] MetaTrader 5, «MetaTrader,» [En línea]. Available: https://www.metatrader5.com/es.

[3] MetaStock, «MetaStock,» [En línea]. Available: https://www.metastock.com/.

[4] Interactive Brokers, «Interactive Brokers,» [En línea]. Available: https://www.interactivebrokers.co.uk/es/home.php.

[5] TrendSpider, «TrendSpider,» [En línea]. Available: https://trendspider.com/.

[6] TradingView, «TradingView,» [En línea]. Available: https://es.tradingview.com/.

[7] NinjaTrader, «NinjaTrader,» [En línea]. Available: https://ninjatrader.com/es/.

[8] Bootstrap , «Bootstrap 5,» [En línea]. Available: https://getbootstrap.com/docs/5.0/getting-started/introduction/.

[9] A. Navarro, «Junco TIC,» 7 Mayo 2019. [En línea]. Available: https://juncotic.com/jinja2-en-flask-introduccion/.

[10] jQuery, «jQuery,» [En línea]. Available: https://jquery.com/.

[11] Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Python.

[1 J. J. L. Gómez, «j2logo,» [En línea]. Available: https://j2logo.com/tutorial-flask-
2] espanol/.

[1 Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/MySQL.
3]

[1 Visual Studio Code, «Visual Studio Code,» [En línea]. Available:
4] https://code.visualstudio.com/.

[1 XAMPP, «XAMPP,» [En línea]. Available:
5] https://www.apachefriends.org/es/index.html.

[1 Wikipedia, «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Git.
6]

[1 CIO Wiki, «Cliente Server Architecture,» [En línea]. Available: https://cio-
7] wiki.org/wiki/Client_Server_Architecture#:~:text=Client%20Server%20Architecture%2
0is%20a,a%20network%20or%20internet%20connection..

[1 PLEDIN 3.0, «Patrón modelo-vista-controlador,» [En línea]. Available:
8] https://plataforma.josedomingo.org/pledin/cursos/flask/curso/u03/.

[1 Z. Vojkovic, «Flask-framework: MVC pattern,» [En línea]. Available:
9] https://stackoverflow.com/questions/12547206/flask-framework-mvc-pattern.

[2 I. Rodríguez, «The Factory Method Pattern and Its Implementation in Python,» 11
0] Febrero 2019. [En línea]. Available: https://realpython.com/factory-method-python/.

[2 UX & Bootstrap, «Portal: Free Bootstrap 5 Admin Dashboard Template For Developers,»
1] [En línea]. Available: https://themes.3rdwavemedia.com/bootstrap-
templates/startup/portal-free-bootstrap-admin-dashboard-template-for-
developers/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part I,» [En línea]. Available:
2] https://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-I/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part II,» [En línea]. Available:
3] http://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-II/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part III,» [En línea]. Available:
4] http://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-III/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part IV,» [En línea]. Available:
5] https://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-IV/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part V,» [En línea]. Available:
6] https://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-V/.

[2 QuantStart, «Event-Driven Backtesting with Python - Part VI,» [En línea]. Available:
7] https://www.quantstart.com/articles/Event-Driven-Backtesting-with-Python-Part-VI/.

[2 QuantStart, «Advanced Trading Infrastructure - Position Class,» [En línea]. Available:
8] https://www.quantstart.com/articles/Advanced-Trading-Infrastructure-Position-Class/.

[2 QuantStart, «Advanced Trading Infrastructure - Portfolio Class,» [En línea]. Available:
9] https://www.quantstart.com/articles/Advanced-Trading-Infrastructure-Portfolio-Class/.

[3 QuantStart, «Advanced Trading Infrastructure - Portfolio Handler Class,» [En línea].
0] Available: https://www.quantstart.com/articles/Advanced-Trading-Infrastructure-Portfolio-Handler-Class/.

[3 Algovibes, «Modern portfolio theory in Python: Efficient Frontier and minimum-
1] variance portfolio,» 2020. [En línea]. Available: https://www.youtube.com/watch?v=mJTrQfzr0R4.

[3 L. A. Bucki, Word 2013 Bible, John Wiley & Sons, 2013.
2]

[3 CFI, «Cursos de Formación en Informática,» [En línea]. Available:
3] http://cursosinformatica.ucm.es. [Último acceso: 01 06 2019].

[3 Fidelity, «Fidelity,» [En línea]. Available: https://www.fidelity.com/learning-
4] center/trading-investing/technical-analysis/technical-indicator-guide/atr.

Apéndice A - Manual del usuario

Un usuario no registrado tiene acceso a tan sólo unas pocas funcionalidades del sistema. Por lo tanto, si se quiere acceder a un mayor número de funcionalidades, conviene iniciar sesión. Para ello, el usuario puede hacer click en "Registarse" o en "Login" en los botones que aparecen arriba a la derecha en la barra de navegación. La Ilustración 57 contiene las vistas correspondientes para iniciar sesión y crear una cuenta en la aplicación.

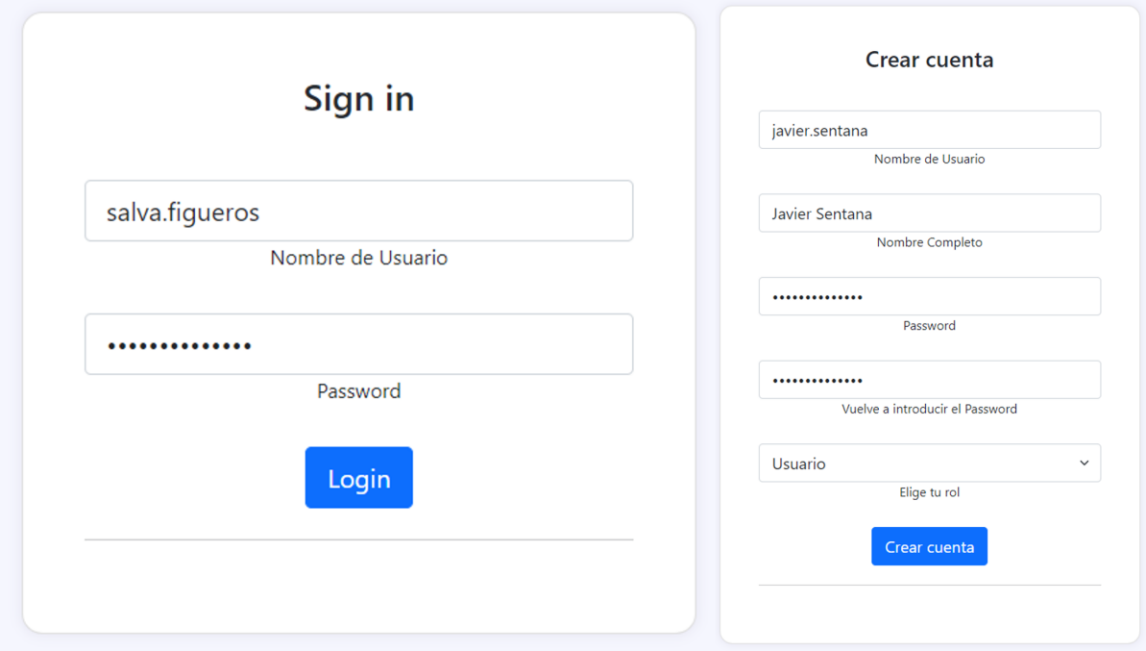


Ilustración 57. Vista Login y Vista Crear Cuenta

Una vez registrado, el usuario accede directamente al dashboard. Aquí, este puede ver la lista de "backtesting" guardados, la lista de carteras financieras creadas, la lista de "backtesting" manuales guardados y el diario de trading. La Ilustración 58 contiene una imagen de esta vista.

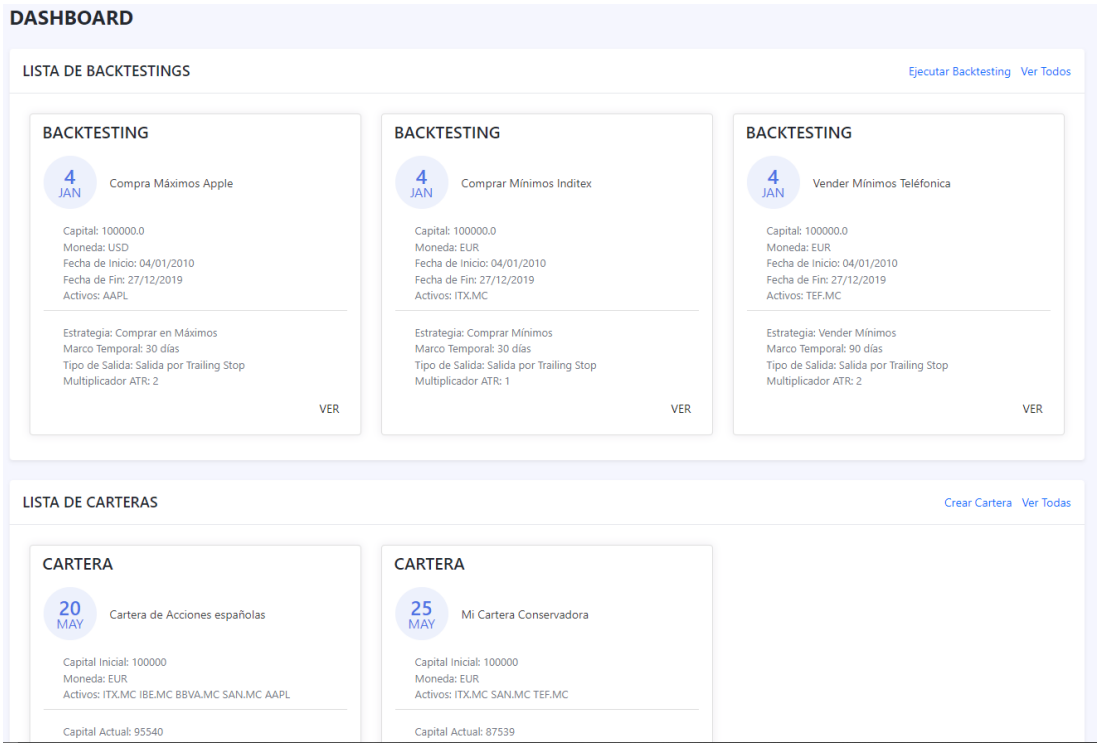


Ilustración 58. Vista Dashboard

Además, en la barra de navegación superior hay un buscador. Este buscador tiene dos campos. En el primero, el usuario selecciona la opción que quiere buscar: "Backtesting", "Cartera", "Usuario" o "Activo". En el segundo campo, el usuario introduce el texto que quiere encontrar. En caso de haber seleccionado "Backtesting", "Cartera" o "Activo", el usuario ha de introducir el ticker de un activo financiero válido. El ticker tiene que ser el mismo con el que se identifica a dicho activo financiero en la página desde donde se realiza la extracción de datos: "Yahoo! Finance". Y es que el administrador añade al sistema los datos de los activos financieros a través de dicha página. Por lo tanto, el ticker del activo financiero en el sistema es el mismo que el de Yahoo! Finance. En la Ilustración 59 se muestra un ejemplo de la búsqueda de "Backtesting" a través del ticker "AAPL" (ticker de Apple).

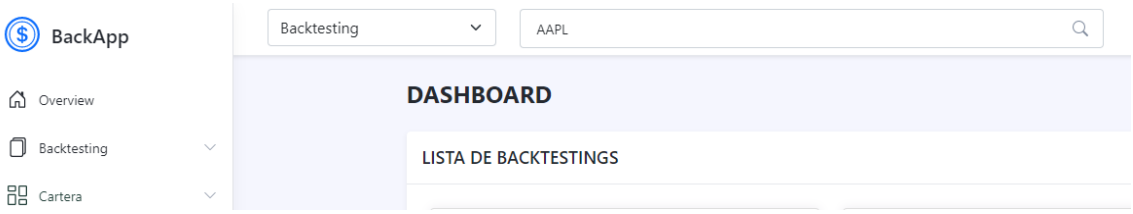


Ilustración 59. Vista Barra de Navegación Superior

Si el usuario introduce un ticker válido, entonces se muestran por pantalla los "backtesting" publicados por el resto de los usuarios que se hayan ejecutado sobre el activo financiero cuyo ticker fue introducido. En la Ilustración 60 se observa el resultado de la búsqueda de "Backtesting" a través del ticker "AAPL".

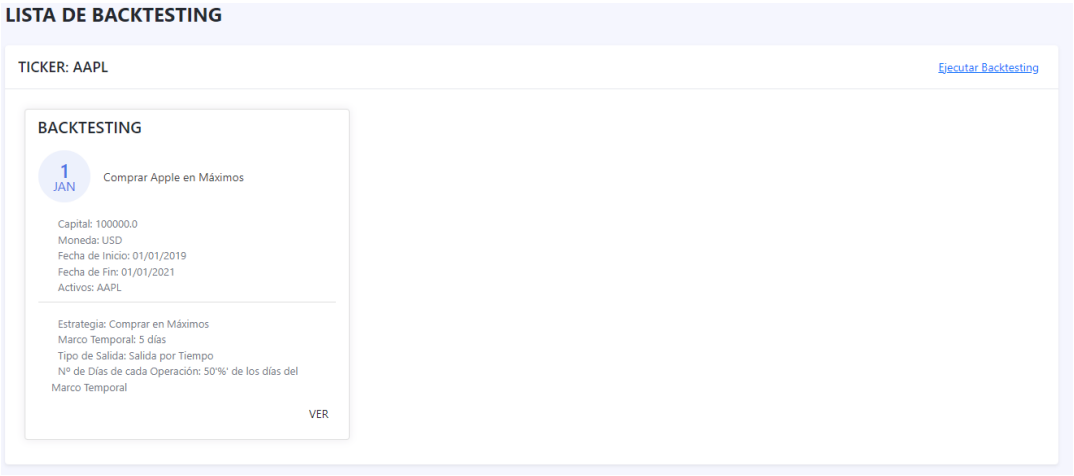


Ilustración 60. Vista Búsqueda Backtesting

La misma lógica se sigue para la búsqueda de carteras. El usuario selecciona en la barra de navegación superior la opción "Cartera" e introduce a continuación el ticker del activo financiero sobre el que quiere realizar la búsqueda. Se muestran entonces todas las carteras publicadas que contengan el activo cuyo ticker fue introducido. La Ilustración 61 contiene un ejemplo de búsqueda de carteras a través del ticker "NFLX" (ticker de Netflix).

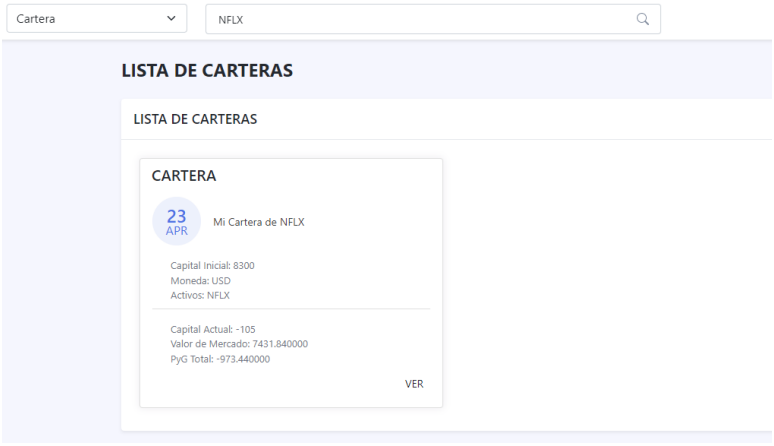


Ilustración 61. Vista Búsqueda de Carteras

Por otra parte, el usuario tiene también la opción de buscar otros usuarios. Para ello, ha de seleccionar en la barra de navegación superior la opción "Usuario" e introducir a continuación el nombre de usuario del usuario que quiere encontrar. La Ilustración 62 contiene un ejemplo de búsqueda de usuarios a través del nombre de usuario "javier.sentana".

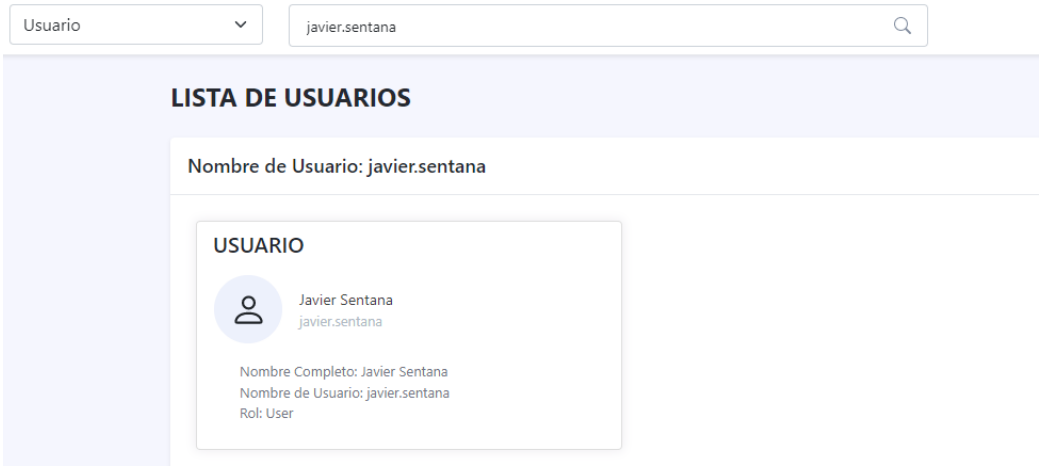


Ilustración 62. Vista Búsqueda de Usuarios

Finalmente, el usuario también puede buscar activos financieros con la intención de ver qué datos diarios del precio hay disponibles en el sistema. Para ello, el usuario ha de seleccionar en la barra de navegación superior la opción "Activo" e introducir a continuación el ticker del activo financiero que quiere buscar. Si el usuario introduce un ticker válido, entonces se muestra un gráfico con la evolución del precio del activo y, más abajo, una tabla con todos los datos diarios del precio tal como se puede observar en la Ilustración 63.

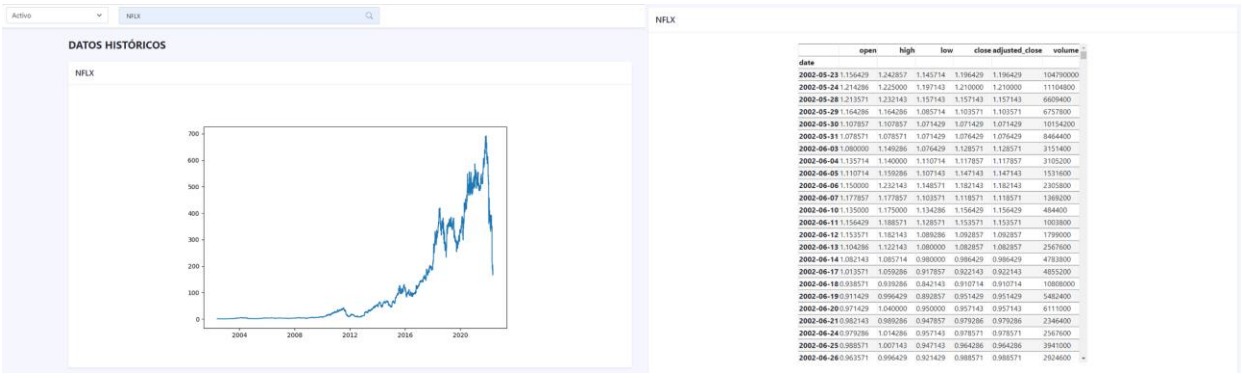


Ilustración 63. Vista Búsqueda de Activos

Backtesting

Para ejecutar un "backtesting", el usuario puede acceder a dicha página tanto a través del dashboard como de la barra de navegación. A través del dashboard, el usuario tiene que hacer click en el botón "Ejecutar Backtesting" dentro de la sección "Lista de Backtestings". Y para hacerlo a través de la barra de navegación, tiene que hacer primero click en el botón "Backtesting" de la barra de navegación lateral. Este abrirá un menú desplegable y el usuario deberá hacer click en la opción "Ejecutar Backtesting". En la Ilustración 64 se puede observar cómo seleccionar la opción de "Ejecutar Backtesting" a través de las dos vías descritas.

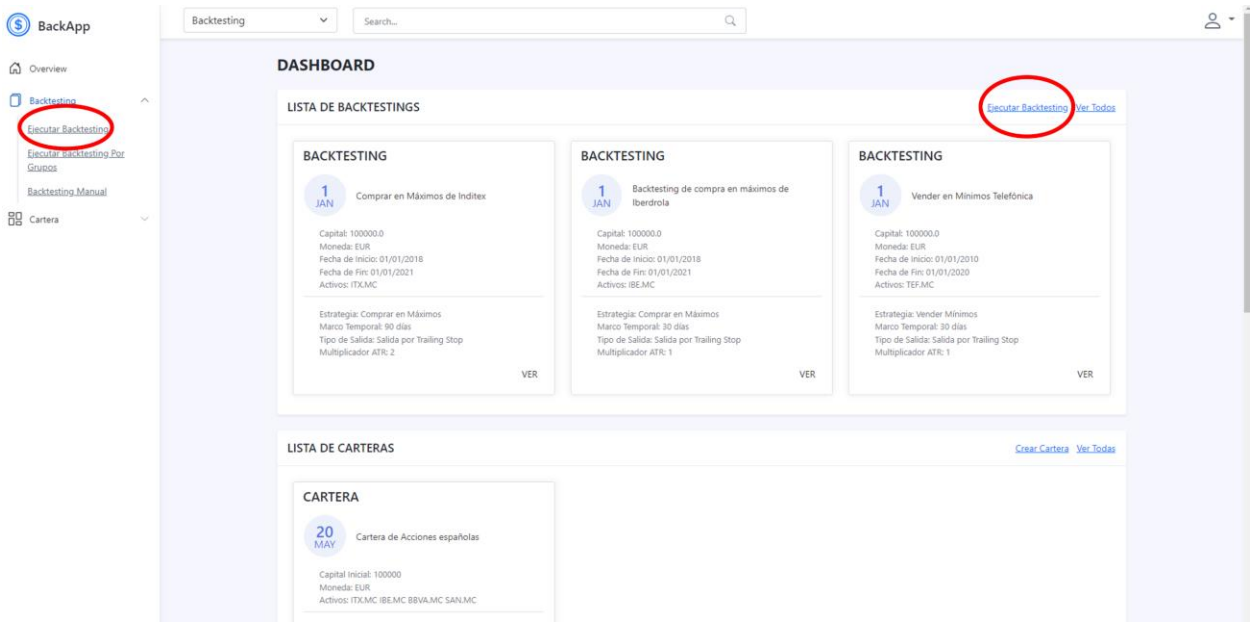


Ilustración 64. Seleccionar "Ejecutar Backtesting"

A continuación, el usuario ha de completar el formulario que aparece en pantalla para la creación del "backtesting". Este es un formulario dividido en 5 pasos.

El primer paso sirve para que el usuario le dé un nombre al "backtesting" y configure la cartera sobre la que va a correr el "backtesting". En el campo "Moneda", este ha de introducir el símbolo de la moneda ("EUR", "USD" …).

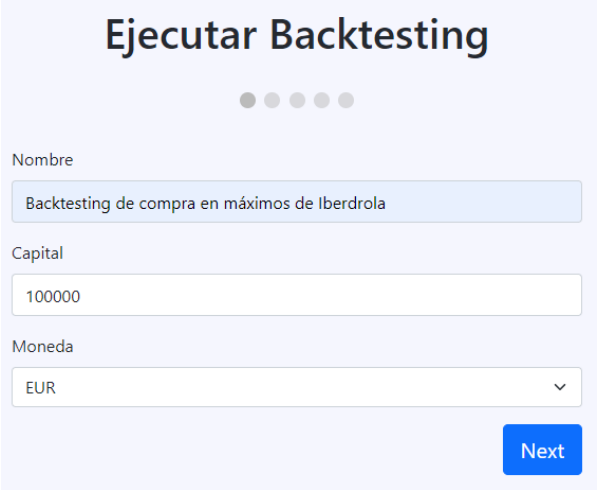


Ilustración 65. Formulario Backtesting Paso 1

En el segundo paso, el usuario ha de introducir la lista de tickers de los activos financieros sobre los que quiere correr el "backtesting".

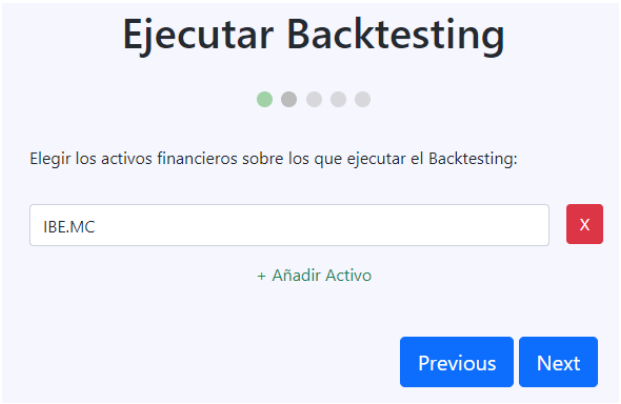


Ilustración 66. Formulario Backtesting Paso 2

En el tercer paso, el usuario ha de elegir las fechas entre las que quiere correr el "backtesting". Si elige unas fechas sobre las que no existan datos para los activos financieros escogidos, la aplicación informará al usuario después de que este haya enviado el formulario.

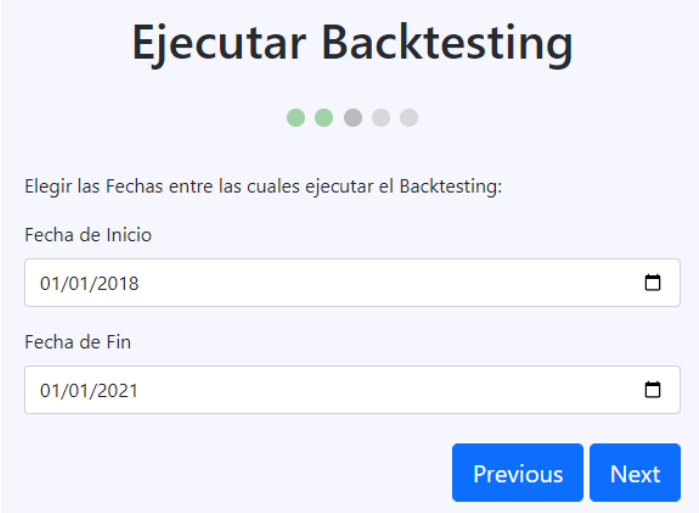


Ilustración 67. Formulario Backtesting Paso 3

En el paso 4, el usuario elige la sub-estrategia que quiere evaluar. Aquí puede elegir las cuatro sub-estrategias implementadas: "Comprar Máximos", "Vender Máximos", "Comprar Mínimos" y "Vender Mínimos".

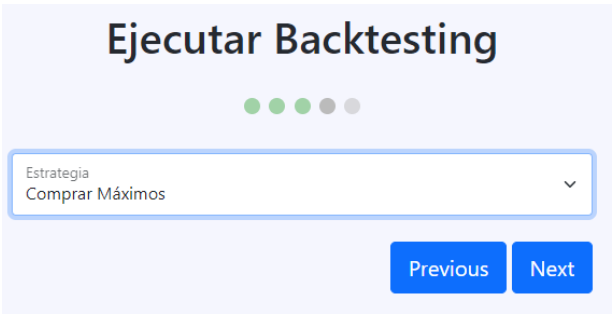


Ilustración 68. Formulario Backtesting Paso 4

Finalmente, en el último paso el usuario tiene que elegir los parámetros de la sub-estrategia escogida. Aquí, tiene que introducir el marco temporal y el tipo de salida que quiere implementar.

Ilustración 69. Formulario Backtesting Paso 5

Una vez completados todos los campos del formulario, el usuario ya puede hacer click en "Ejecutar". Entonces, si todos los datos son correctos, se crea el "backtesting" y se muestra por pantalla su configuración. Para ejecutarlo finalmente, el usuario ha de hacer click en el botón verde "Ejecutar Backtesting".

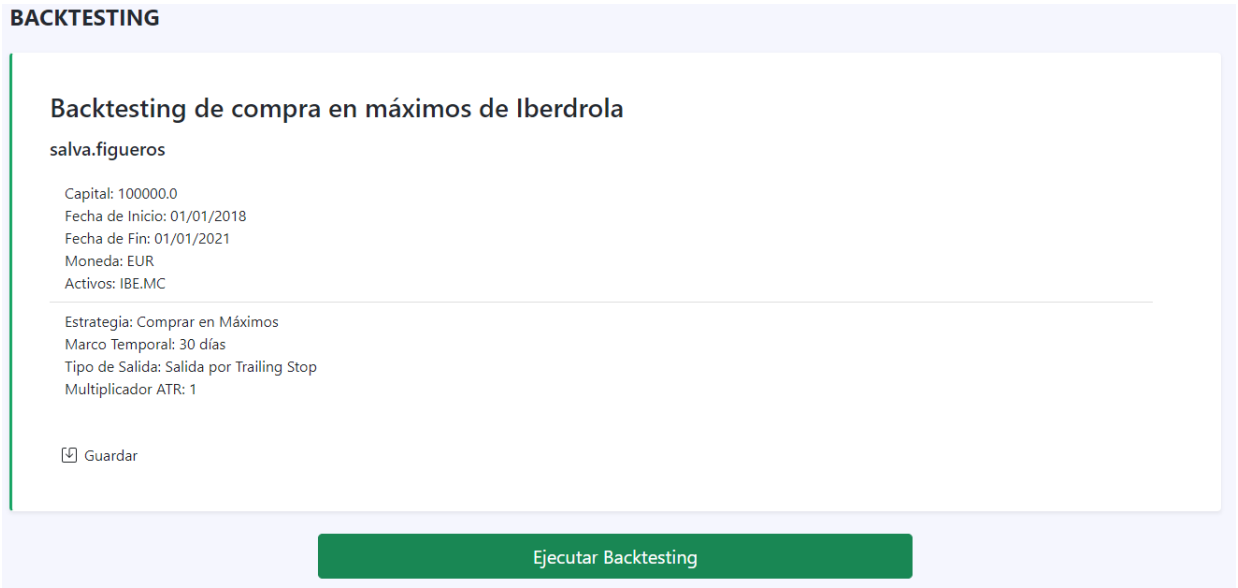


Ilustración 70. Vista Configuración Backtesting

Una vez ejecutado, se muestran los resultados del "backtesting". En primer lugar, el sistema muestra una alerta acerca de si se recomienda o no el uso de esta estrategia tal como se puede ver en la Ilustración 71.

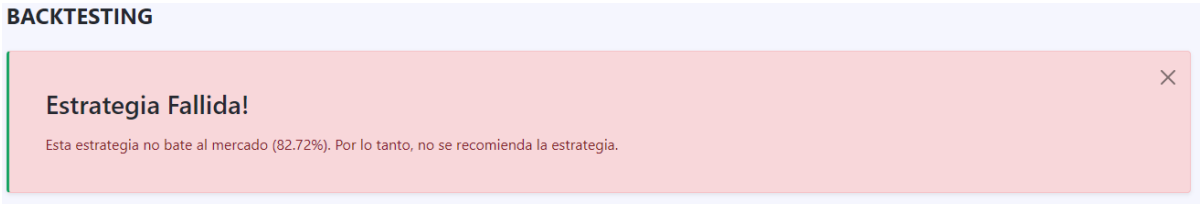


Ilustración 71. Alerta Resultado Backtesting

Lo siguiente ya son los resultados estadísticos del "backtesting" ejecutado. Como se puede observar en la Ilustración 72, se muestra el rendimiento acumulado, el Ratio de Sharpe (que mide la relación entre rendimiento y riesgo), el mayor Drawdown (máxima caída) y la duración de este Drawdown. A esto, además, hay que añadir un gráfico con la evolución de la curva de capital de la estrategia.

Ilustración 72. Vista Resultados Backtesting

Si el usuario hace scroll hacia abajo tiene también la opción de ver un gráfico, para cada uno de los activos financieros sobre los que se ha ejecutado el "backtesting", en el que se muestra la evolución del precio del activo financiero en cuestión junto a las señales de compra

y de venta marcadas por la estrategia. En la Ilustración 73 se puede observar un ejemplo de este tipo de gráfico con la evolución del precio para Iberdrola y las señales de compra (verde) y de venta (rojo) marcadas por la estrategia.

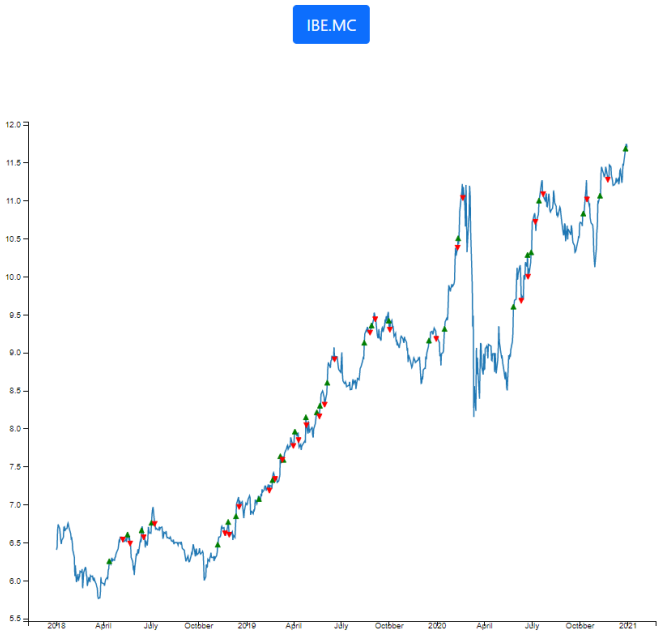


Ilustración 73. Vista Gráfico de las Señales de Trading

Para guardar el "backtesting" para su posterior recuperación, el usuario tiene que hacer click en el botón "Guardar" que aparece en la Ilustración 70. Una vez guardado, aparecerán, como se puede observar en la Ilustración 74, los botones para compartir y eliminar el "backtesting".

Ilustración 74. Botones "Compartir" y "Eliminar" Backtesting

La ejecución del "backtesting" por grupos se hace de una forma muy parecida. Para acceder a esta página, hay que hacer click en el elemento "Backtesting" de la barra lateral de navegación. Una vez abierto el menú desplegable, el usuario tiene que hacer click en "Ejecutar Backtesting Por Grupos".

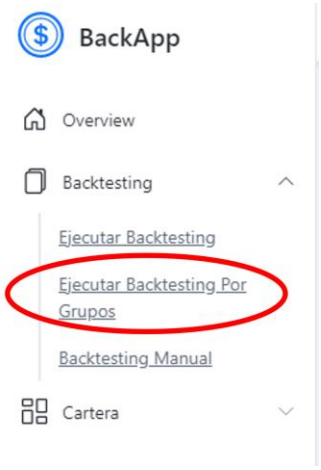


Ilustración 75. Seleccionar "Ejecutar Backtesting Por Grupos"

Ahora, el usuario ha de completar un formulario casi idéntico al de "Ejecutar Backtesting". La única diferencia reside en completar un paso previo en el que hay que elegir la métrica que se quiere utilizar para dividir a los activos financieros en dos grupos. Para ello, el usuario tiene dos opciones: "Capitalización del Mercado" y "Volumen". En la Ilustración 76 se puede ver la vista para este primer paso del formulario. El resto de los pasos son los mismos que hay que completar en el formulario de "Ejecutar Backtesting".

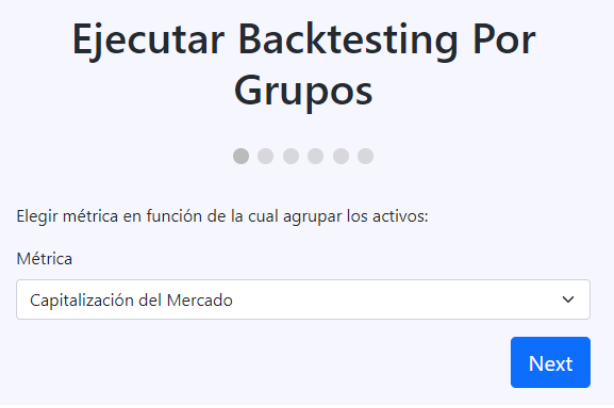


Ilustración 76. Formulario "Ejecutar Backtesting Por Grupos"

Una vez completado el formulario, el usuario ya puede ver por pantalla los "backtesting" creados por cada uno de los dos conjuntos de activos financieros formados tal como se muestra en la Ilustración 77. A partir de ahí, este puede gestionarlos (verlos, ejecutarlos, guardarlos, publicarlos o eliminarlos) siguiendo los pasos que se han comentado hasta aquí.

Ilustración 77. Vista Backtesting Por Grupos

Gestión de carteras

Al igual que la opción "Ejecutar Backtesting", el usuario puede acceder a la página "Crear Cartera" tanto a través del dashboard como de la barra de navegación. A través del dashboard, el usuario tiene que hacer click en el botón "Crear Cartera" dentro de la sección "Lista de Carteras". Y para hacerlo a través de la barra de navegación, tiene que hacer primero click en el botón "Cartera" de la barra de navegación lateral. Este abrirá un menú desplegable y el usuario deberá hacer click en la opción "Crear Cartera". En la Ilustración 78 se puede observar cómo seleccionar la opción de "Crear Cartera" a través de las dos vías descritas.

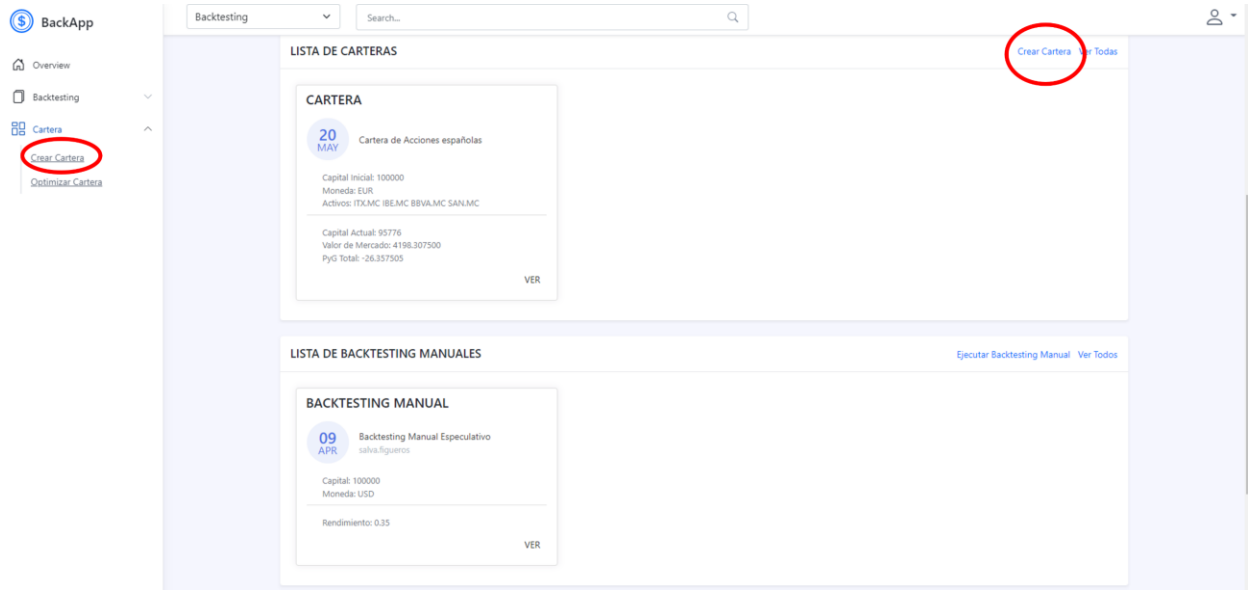


Ilustración 78. Seleccionar "Crear Cartera"

La creación de una cartera se hace también a través de un formulario. Este formulario está compuesto por los dos pasos incluidos en la Ilustración 79.

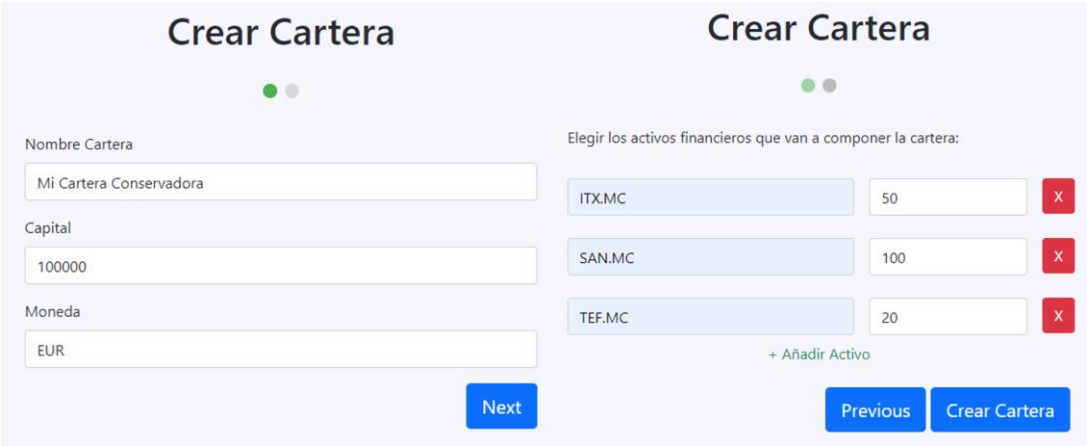


Ilustración 79. Vista Formulario Crear Cartera

Una vez creada la cartera, el usuario puede acceder a las estadísticas más relevantes de esta, como se puede observar en la Ilustración 80. Ahí el usuario puede consultar visualmente las pérdidas o ganancias, el valor de mercado de la cartera, el capital que tiene en su cuenta y el

"Equity" (valor de mercado + capital). Además, también cuenta con un gráfico que representa la distribución de los activos de la cartera.

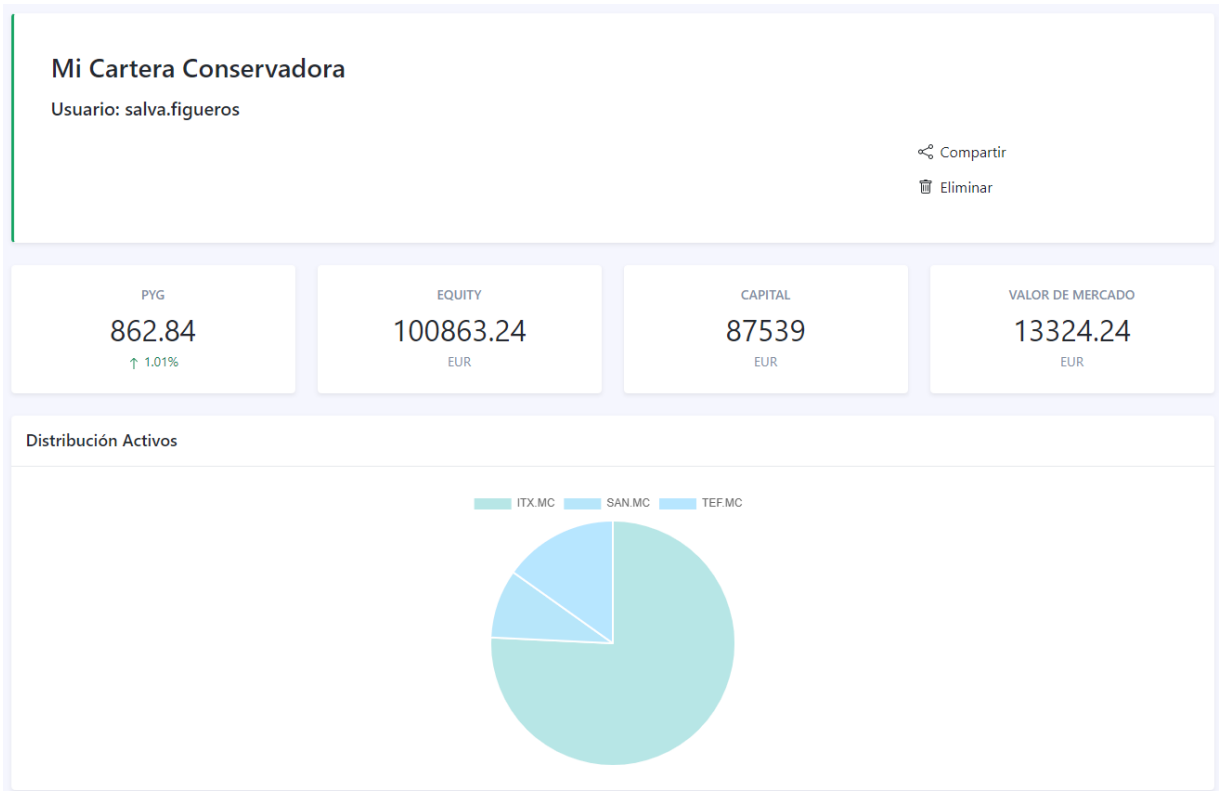
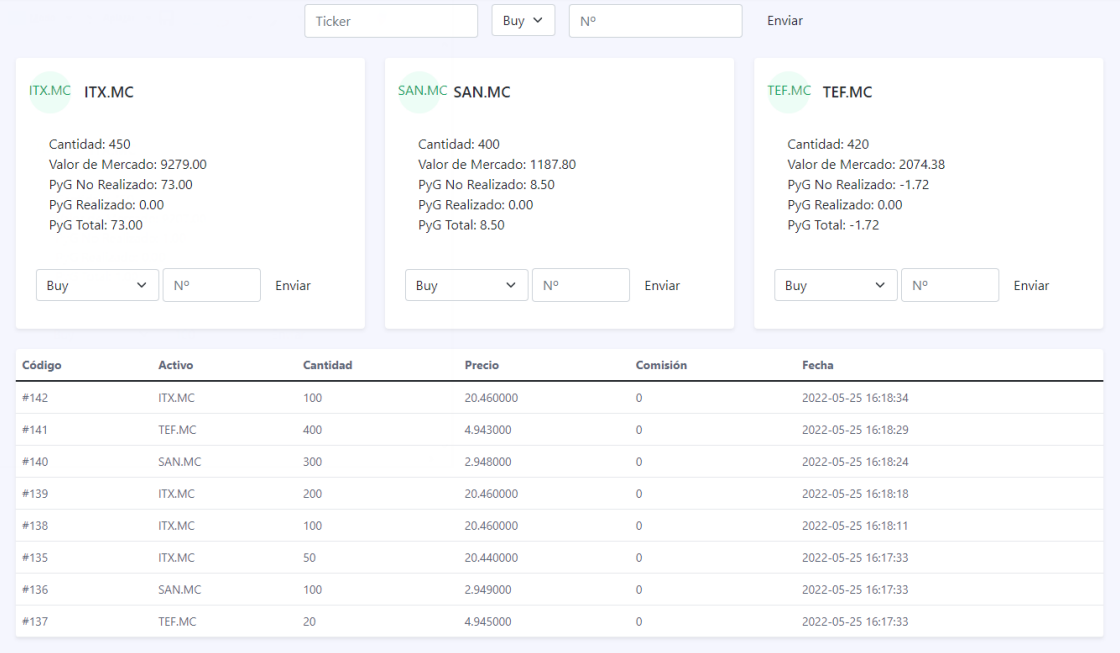


Ilustración 80. Vista Cartera

Para compartir o eliminar la cartera, el usuario tiene que hacer click en los correspondientes botones que aparecen en el encabezado superior a la derecha en la Ilustración 80.

Además, desde esta misma vista y tan solo haciendo scroll hacia abajo, el usuario también tiene acceso al conjunto de estadísticas relacionadas con las posiciones y las transacciones de la cartera, tal como se muestra en la Ilustración 81.



Código	Activo	Cantidad	Precio	Comisión	Fecha
#142	ITX.MC	100	20.460000	0	2022-05-25 16:18:34
#141	TEF.MC	400	4.943000	0	2022-05-25 16:18:29
#140	SAN.MC	300	2.948000	0	2022-05-25 16:18:24
#139	ITX.MC	200	20.460000	0	2022-05-25 16:18:18
#138	ITX.MC	100	20.460000	0	2022-05-25 16:18:11
#135	ITX.MC	50	20.440000	0	2022-05-25 16:17:33
#136	SAN.MC	100	2.949000	0	2022-05-25 16:17:33
#137	TEF.MC	20	4.945000	0	2022-05-25 16:17:33

Ilustración 81. Vista Posiciones y Transacciones de la Cartera

La modificación de la cartera consiste en la compra o venta de contratos de alguno de los activos financieros que componen dicha cartera. Para ello, y conforme se puede observar en la Ilustración 81, hay un formulario en la parte superior en donde el usuario ha de introducir el ticker, indicar si quiere comprar o vender ("Buy" o "Sell") y la cantidad de contratos que quiere comprar o vender. Además, debajo de cada activo financiero hay otro formulario en el que el usuario indica también si quiere comprar o vender y la cantidad de contratos que quiere comprar o vender del activo financiero en cuestión. Por ejemplo, si se quieren comprar 100 acciones más de Inditex habría que seleccionar la opción "Buy", introducir la cifra de 100 en el campo siguiente y hacer click en el botón "Enviar" tal como se muestra en la Ilustración 82.

Ilustración 82. Vista Modificar Cartera

En cuanto a la optimización de carteras, el proceso es muy similar al de creación. Se accede desde la barra de navegación lateral, haciendo click en "Optimizar Cartera" una vez desplegado el menú asociado a la opción "Cartera".

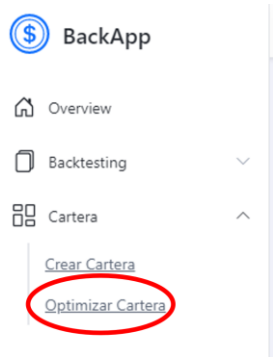


Ilustración 83. Seleccionar "Optimizar Cartera"

Una vez el usuario accede a la página, este ha de completar un formulario muy parecido al que hay que completar para la creación de carteras. En el primer paso, el usuario introduce el capital de la cartera a optimizar y la moneda en la que está denominada. En el segundo paso, el usuario añade los activos financieros que van a componer dicha cartera. Además, aquí el usuario tiene la opción de añadir todos los activos financieros del sistema marcando la casilla "Todos los activos". De esta forma, el formulario quedaría como se indica en la Ilustración 84.

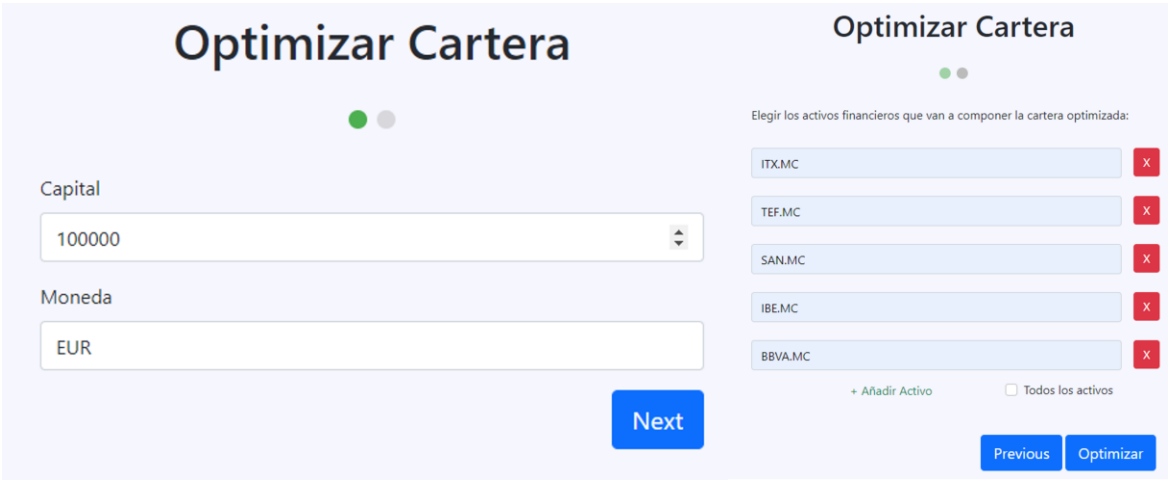


Ilustración 84. Vista Formulario Optimizar Cartera

Finalmente, y una vez enviado el formulario, el usuario puede observar los resultados obtenidos de dicha optimización tal como se muestra en la Ilustración 85. Lo que puede ver por pantalla el usuario son el rendimiento, la volatilidad y la distribución de pesos de la cartera optimizada. Además, se muestra también el gráfico de la frontera eficiente, el cual incluye la representación de todas y cada una de las carteras que se han generado durante la ejecución del algoritmo. El eje de abscisas representa la desviación típica de las carteras (volatilidad) y el eje de ordenadas el rendimiento diario de estas.

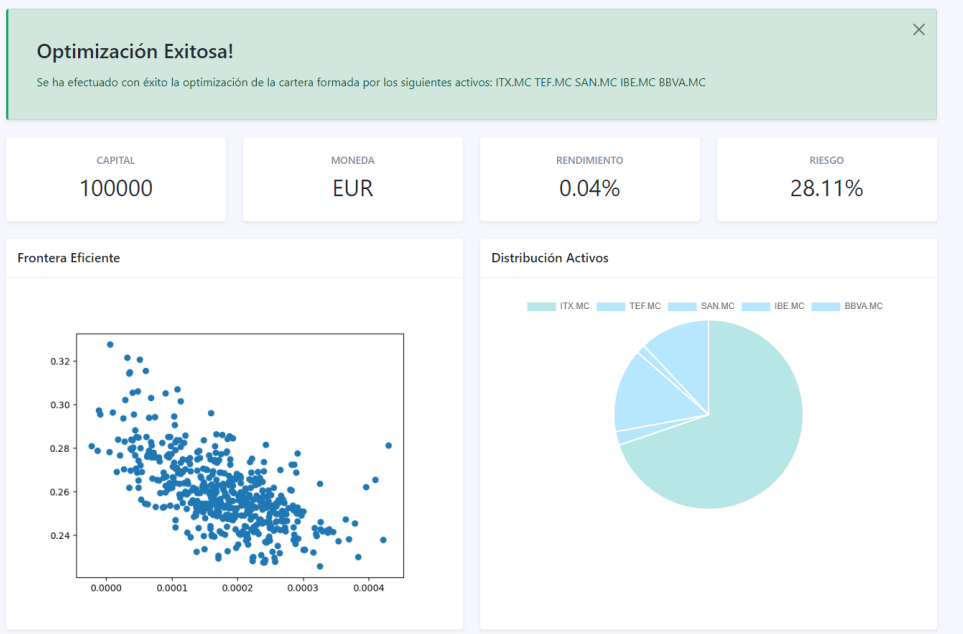


Ilustración 85. Vista Optimización de Cartera

Otros servicios de trading

Para la ejecución de un "backtesting" de forma manual, se puede acceder a la página correspondiente de dos formas distintas. A través del dashboard, el usuario tiene que hacer click en el botón "Ejecutar Backtesting Manual" dentro de la sección "Lista de Backtesting Manuales". Y para hacerlo a través de la barra de navegación, tiene que hacer primero click en el botón "Backtesting" de la barra de navegación lateral. Este abrirá un menú desplegable y el usuario deberá hacer click en la opción "Backtesting Manual". En la Ilustración 86 se puede observar cómo seleccionar la opción de "Backtesting Manual" a través de las dos vías descritas.

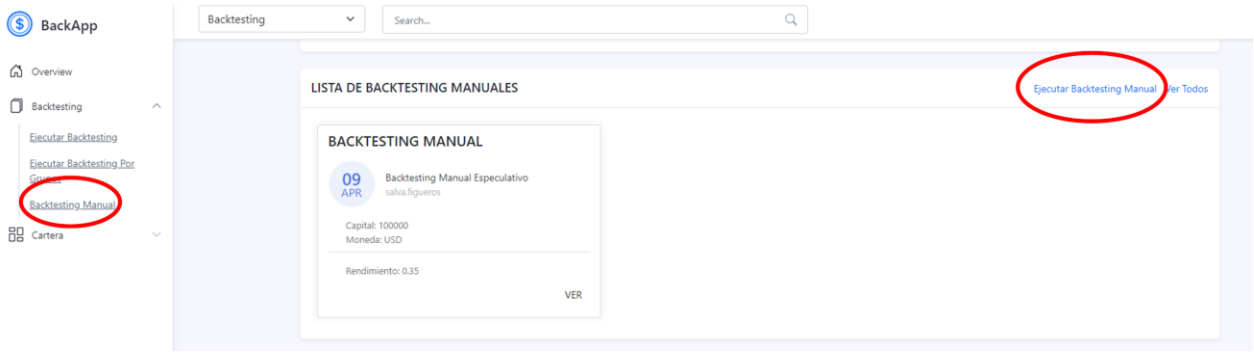


Ilustración 86. Seleccionar "Backtesting Manual"

Una vez en la página, el usuario completa un formulario cuyos campos hacen referencia al capital de la cartera que quiere configurar y la moneda en la que está denominada dicha cartera. En la Ilustración 87 se puede observar un ejemplo de este formulario.

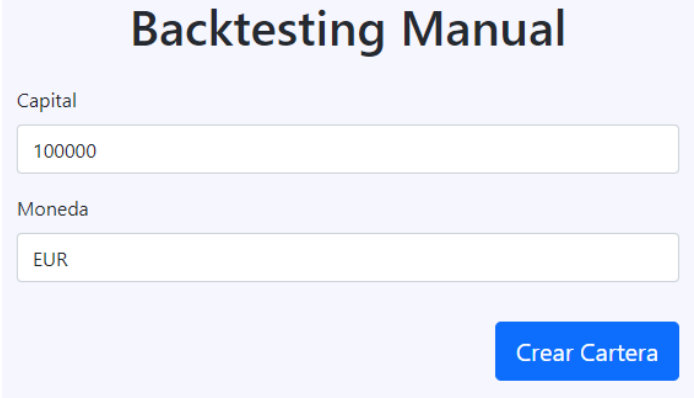


Ilustración 87. Vista Formulario "Backtesting Manual"

Una vez creado el "backtesting" manual, el usuario accede a este de forma automática. Inicialmente, el usuario no ha podido añadir aún beneficios o pérdidas asociados a operaciones. Por lo tanto, el "backtesting" manual se mostraría con su configuración inicial tal cual viene indicado en la Ilustración 88.

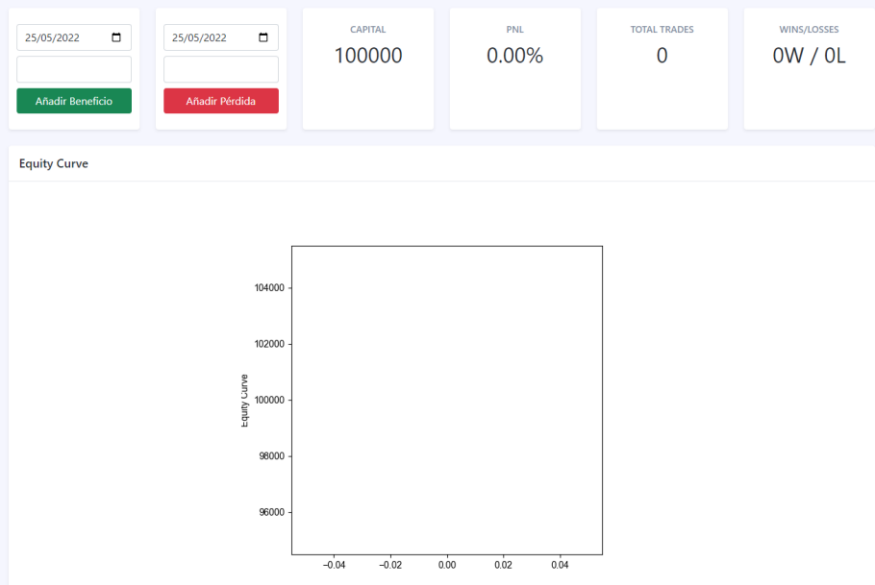


Ilustración 88. Vista Backtesting Manual Creado

Ahora el usuario ya puede introducir los beneficios y las pérdidas que quiere registrar. Para ello, ha de introducir la cantidad del beneficio o pérdida que ha sufrido, la fecha en la que ha sufrido dicho beneficio o pérdida y hacer click en "Añadir Beneficio" o "Añadir Pérdida". En la Ilustración 89 se puede observar un ejemplo de dónde y cómo realizar estas acciones.

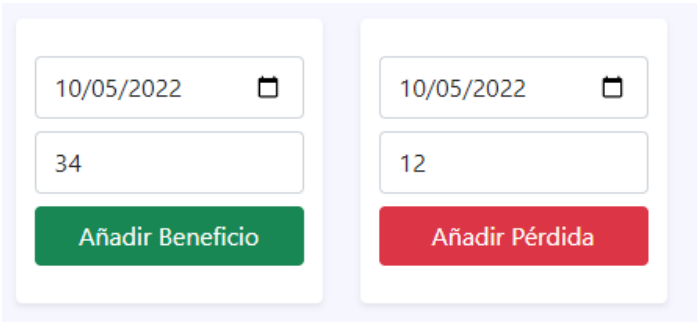


Ilustración 89. Ganancias y Pérdidas Backtesting Manual

Conforme el usuario vaya añadiendo los beneficios y pérdidas sufridos por este en su operativa como trader, el sistema va actualizando todas las estadísticas asociadas y muestra dichas actualizaciones automáticamente. De tal forma que la vista asociada a un "backtesting" manual quedaría tal cual se puede observar en la Ilustración 90.

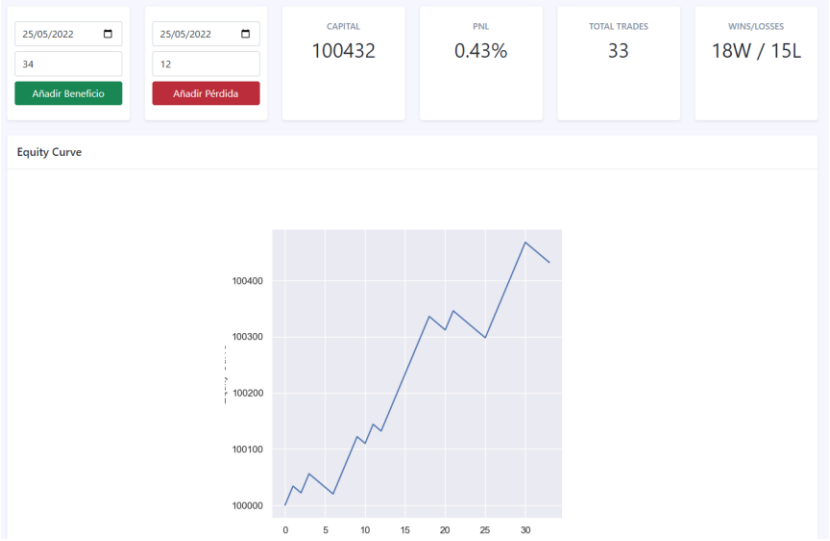


Ilustración 90. Vista Backtesting Manual

Finalmente, si el usuario quiere guardar dicho "backtesting" manual para su posterior recuperación, este tiene que hacer click en el botón verde "Guardar" que aparece abajo del todo y, acto seguido, la aplicación pedirá a este que introduzca el nombre que le quiere dar al "backtesting" manual. En la Ilustración 91 se puede ver un ejemplo de dónde se encuentra dicho botón y, al mismo tiempo, dónde aparece la ventana en la que el usuario ha de introducir el nombre que le quiere poner al "backtesting" manual.

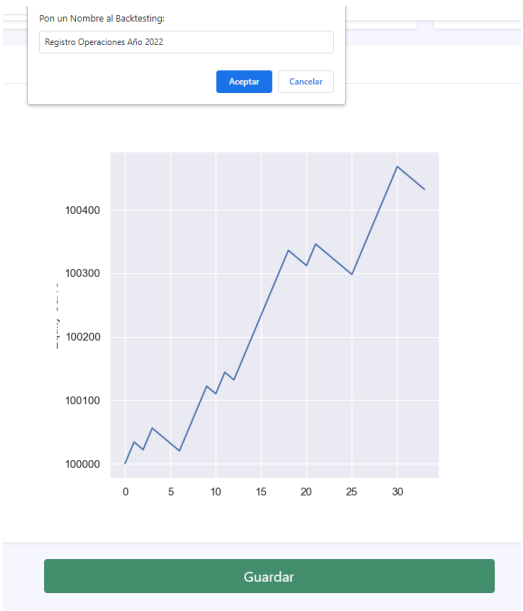


Ilustración 91. Vista Guardar Backtesting

Aparte del "Backtesting Manual", el usuario también tiene la oportunidad de escribir comentarios en su diario de trading. Para escribir en el diario de trading, el usuario tiene que acceder a él desde el dashboard, haciendo click en el botón "Escribir" que hay en la sección "Diario de Trading" tal como se indica en la Ilustración 92.

Ilustración 92. Seleccionar "Escribir" en Diario de Trading

Una vez ahí, el usuario ya puede escribir el texto que quiera sobre la fecha que quiera. Entonces, se muestra automáticamente una vista con el texto escrito por el usuario. Esto se puede ver claramente en la Ilustración 93.

Ilustración 93. Vista Diario de Trading

Administrar datos

Sólo el administrador tiene acceso a esta funcionalidad. Por lo tanto, una vez iniciada la sesión como administrador, el usuario puede hacer click en el botón "Subir Datos Históricos" que aparece en la barra lateral de navegación tal como se indica en la Ilustración 94.

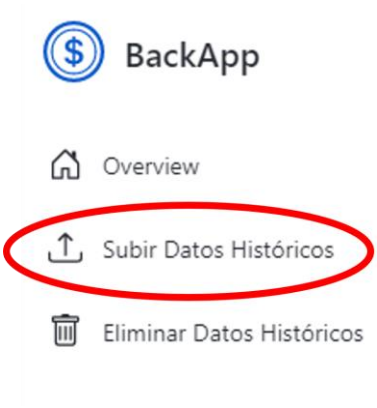


Ilustración 94. Seleccionar "Subir Datos Históricos"

Entonces, tal cual aparece en la Ilustración 95, el usuario tiene la opción de subir datos sobre acciones o futuros.

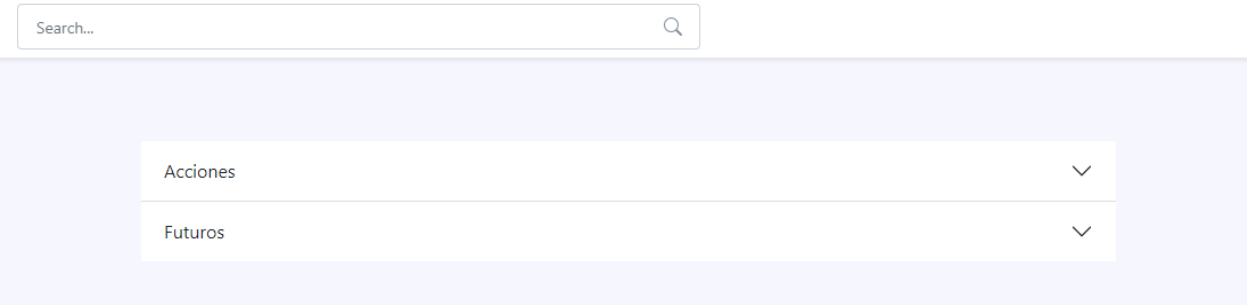


Ilustración 95, Vista Acciones o Futuros

Si por ejemplo, el usuario elige la primera opción, entonces tendrá que rellenar un formulario con los datos correctos. Para ello, tiene que introducir el ticker del activo financiero sobre el que quiere subir los datos, el nombre de la compañía, el mercado en el que cotiza la acción, la moneda en la que está denominada la acción y las fechas de los datos. El nombre, el

mercado y la moneda tan sólo son necesarios rellenarlos para aquellas acciones sobre las que no se hayan subido datos históricos previamente.

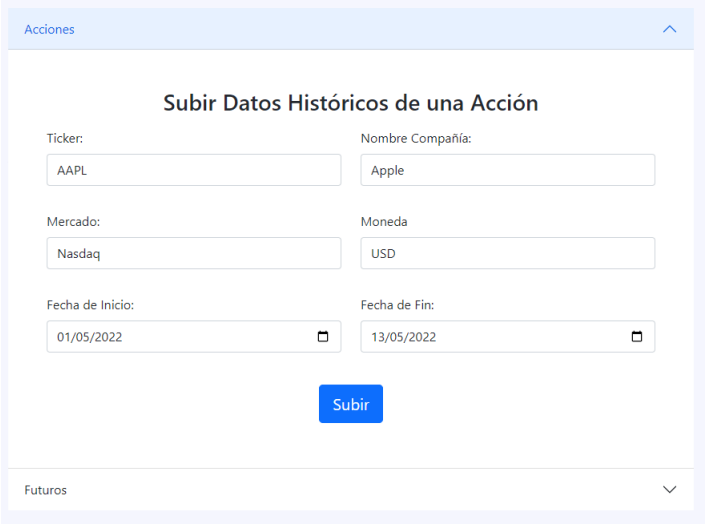


Ilustración 96, Vista Formulario para subir datos históricos

Una vez enviado dicho formulario, se muestra una vista con el gráfico de los precios a modo de confirmación.

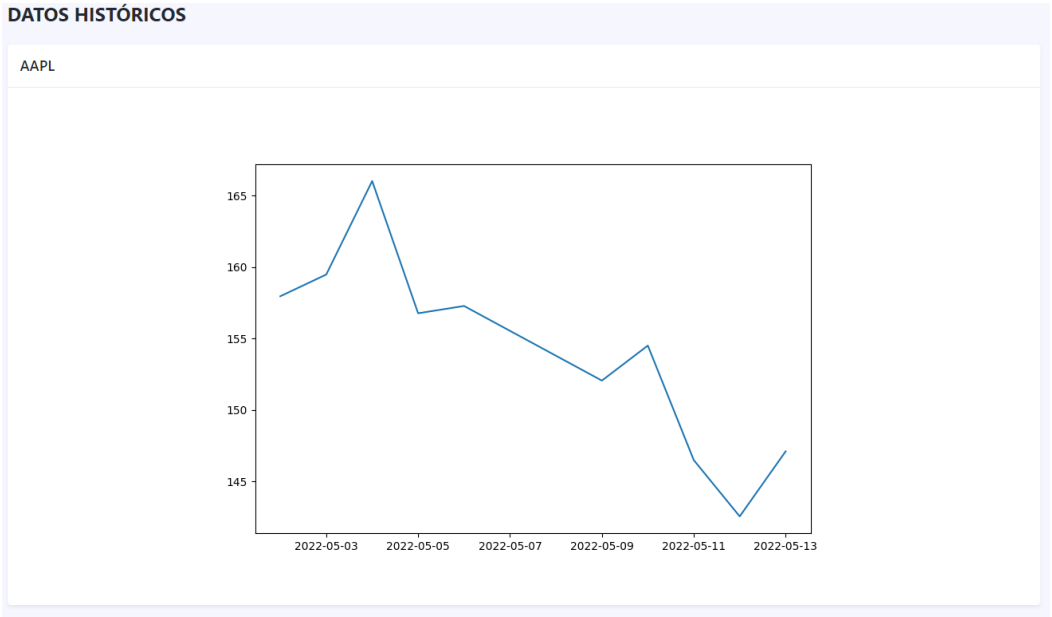


Ilustración 97. Vista Confirmación después de subir datos históricos

Finalmente, para eliminar datos históricos que se hayan subido previamente, el administrador ha de hacer click en el botón "Eliminar Datos Históricos" de la barra de navegación lateral.

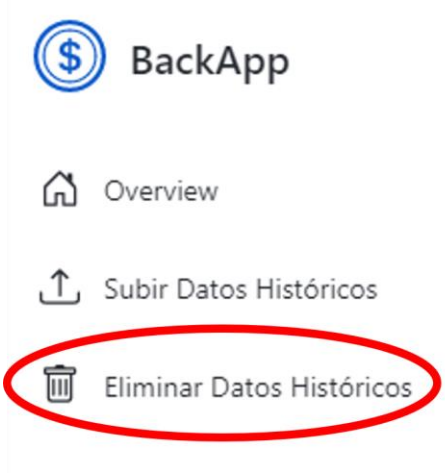


Ilustración 98. Seleccionar "Eliminar Datos Históricos"

Una vez hecho esto, el administrador debe completar el formulario que aparece por pantalla. Aquí, ha de introducir el ticker del activo financiero sobre el que quiere eliminar los datos y, después, las fechas de los datos que quiere eliminar. Tiene también la posibilidad de eliminar todos los datos del ticker introducido marcando la casilla de "Seleccionar Todo". La Ilustración 99 contiene una imagen con los dos pasos que tiene que completar el administrador.

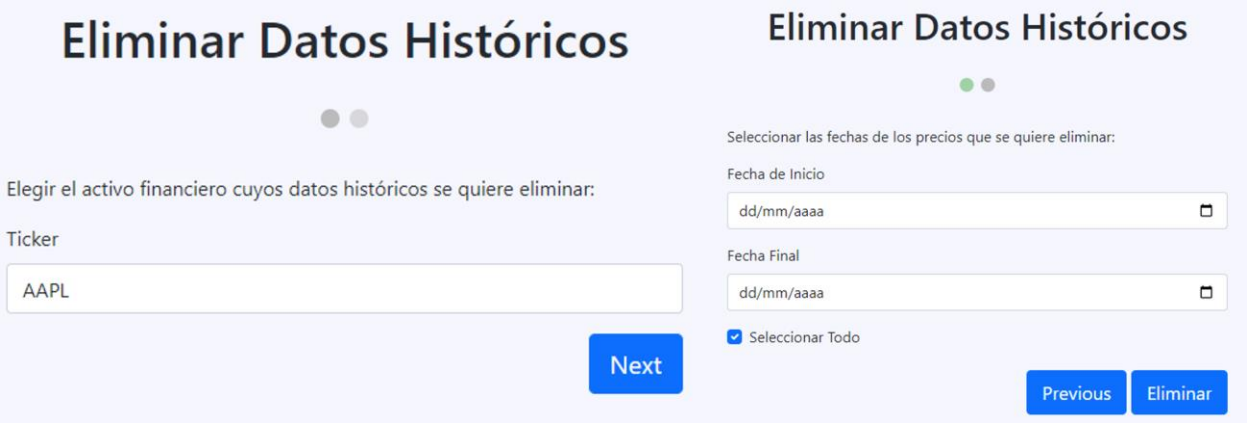


Ilustración 99. Vista Formulario Eliminar Datos Históricos

Si los datos introducidos por el usuario son correctos, el sistema procederá con la eliminación de los datos indicados por el administrador y, posteriormente, dirigirá a este al menú principal.

Gestión de las Cuentas de Usuario

Por último, cada usuario puede modificar la información asociada a su cuenta. Para ello, el usuario ha de hacer click en el botón "Cuenta de Usuario" que aparece en el menú desplegable asociado al símbolo de usuario situado arriba a la derecha. Se puede ver dónde se encuentra dicho botón en la Ilustración 100.

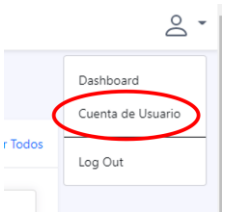


Ilustración 100. Vista Seleccionar "Cuenta de Usuario"

Una vez hecho esto, el usuario accede a la página en donde puede modificar toda su información. En concreto, desde aquí puede modificar su nombre de usuario, su nombre completo y su contraseña y, además, puede darse de baja también en la aplicación haciendo click en el botón rojo "Eliminar Cuenta" que aparece en Ilustración 101.

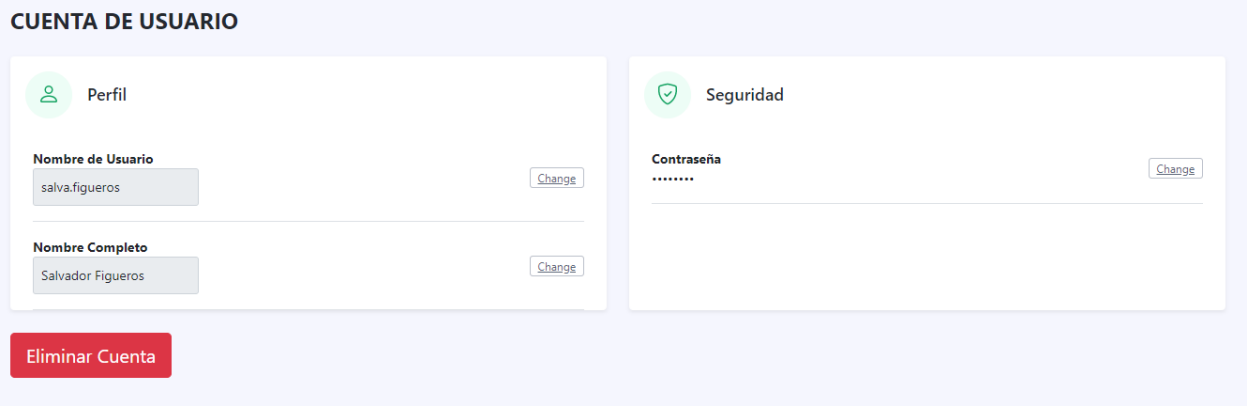


Ilustración 101. Vista Cuenta de Usuario

Cada usuario tendrá acceso únicamente a la información de su propia cuenta y podrá modificar únicamente la información de su propia cuenta excepto el administrador, que podrá realizar sendas acciones para todas las cuentas de usuario del sistema.