Trading App:
Development of a tool to help investment in the Stock Market

Trading App:

Desarrollo de una herramienta de ayuda a la inversión en Bolsa

UNIVERSIDAD COMPLUTENSE MADRID

Trabajo de fin de grado del Grado en Ingeniería Informática

Facultad de Informática

Autor

Marlon Campoverde Méndez

Director

Antonio Sarasa Cabezuelo

Curso 2022–2023

Trading App

Development of a tool to help investment in the Stock Market

Final Degree Project Report

Marlon Campoverde Méndez

Directed by Antonio Sarasa Cabezuelo

Faculty of Computer Science

Complutense University of Madrid

Madrid, 2023

Sobre TEFLONX

Teflon X(cc0 1.0(documentación) MIT(código))es una plantilla de LaTeX creada por David Pacios Izquierdo con fecha de Enero de 2018. Con atribuciones de uso CC0.

Esta plantilla fue desarrollada para facilitar la creación de documentación profesional para Trabajos de Fin de Grado, Trabajos de Fin de Máster o Doctorados. La versión usada es la X

V:X Overleaf V2 with XeLaTeX, margin 1in, bib

Contacto

Autor: David Pacios Izquiero

Correo: dpacios@ucm.es

ASCII: ascii@ucm.es

Despacho 110 - Facultad de Informática

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La inversión en el mercado de valores es una actividad que ha atraído la atención de individuos y entidades por igual. La búsqueda de oportunidades para maximizar el rendimiento del capital ha llevado al desarrollo de diversas estrategias y enfoques. Entre estos enfoques, el análisis técnico [6] se ha establecido como una herramienta esencial en la toma de decisiones de inversión, basándose en el estudio de gráficas de precios, patrones y volumen para prever futuros movimientos del mercado.

1.1. Motivación

La naturaleza volátil y dinámica de los mercados financieros proporciona a los inversores un entorno desafiante y en constante cambio. La búsqueda de patrones y tendencias en los movimientos de precios se ha convertido en una búsqueda constante para aquellos que desean aprovechar las oportunidades de inversión. Sin embargo, el análisis exhaustivo de datos históricos y la interpretación de indicadores técnicos pueden ser laboriosos. También es necesario una profunda prueba de la estrategia que se pretende utilizar (backtesting), de esta forma se determina en que escenarios es buena y en cuales no lo es.

Motivado por la necesidad de facilitar el proceso de toma de decisiones para inversores de todos los niveles, se planteó la idea de desarrollar una aplicación que simplificara

y automatizara parte de este proceso. La aplicación se basa en la premisa de utilizar indicadores técnicos como señales para identificar posibles puntos de compra y venta en el mercado de valores. Permitiendo acceder a datos históricos y así analizarlos de forma sencilla y rápido.

1.2. Objetivos

El objetivo principal es desarrollar una aplicación básica para la ayuda en la toma de decisiones de inversión en bolsa, utilizando indicadores técnicos como base para las recomendaciones de compra. Para determinar los puntos de venta se empelan principios sobre la gestión del riesgo.

En base a este objetivo se plantean subobjetivos:

- Establecer una idea general sobre los objetivos del análisis técnico y su aplicación en la toma de decisiones de inversión.
- Seleccionar un conjunto adecuado de indicadores técnicos que serán utilizados como base para las señales de compra.
- Proporcionar un precio objeto para establecer los puntos de venta
- Diseñar e implementar la aplicación, asegurando su funcionalidad y la capacidad de generar recomendaciones de inversión.
- Evaluar la efectividad de la aplicación mediante pruebas que hagan uso de datos históricos del mercado y ejecutando las recomendaciones.
- Proporcionar un análisis de los resultados obtenidos, discutiendo las limitaciones de la aplicación, sus posibles mejoras y su potencial utilidad en el entorno financiero real.

1.3. Metodología

La implantación del proyecto se ha llevado a cabo de acuerdo a un plan de proyecto y una organización del trabajo la cual se describe en las secciones que siguen.

1.3.1. Fases del proyecto

Para el desarrollo del proyecto, se ha realizado una primera etapa para la toma de requisitos, de esta forma se establece el alcance de la aplicación determinando los módulos que se incluirá. En la siguientes etapas se desarrolla y se prueba la funcionalidad del programa, en la última etapa se realiza la memoria. En la Tabla 1.1 se puede ver la duración de cada uno de las etapas del proyecto, por otro lado, se puede ver de forma visual mediante un diagrama de Gantt en la Figura 1.1.

Etapa	**Fecha de duración**
Investigación y toma de requisitos	19/09/2022 - 19/10/2022
Obtención y prueba de API	20/10/2022 - 18/11/2022
Desarrollo de la aplicación	19/10/2022 - 30/05/2023
Pruebas y solución de errores	01/06/2023 - 24/06/2023
Presentación al tutor	25/06/2023
Desarrollo de la memoria	26/06/2023 - 03/09/2023

Cuadro 1.1: Fases en el desarrollo del proyecto

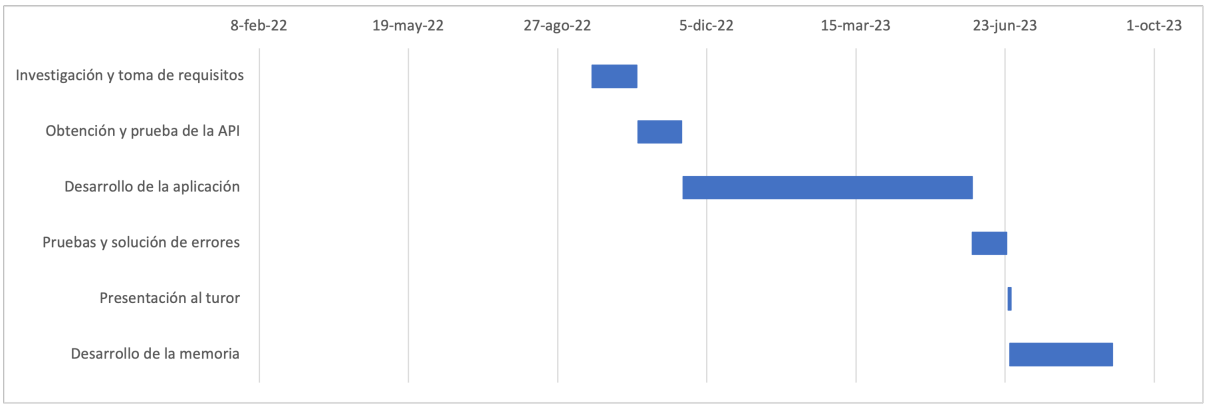


Figura 1.1: Diagrama de Gantt

A continuación se explicará con más detalle cada una de las etapas en las que ha dividido el proyecto.

Fase 1: Toma de requisitos

En la primera etapa, se analizó cuales son los módulos príncipes que una aplicación debe tener, junto con los que debería tener un software dedicado a las inversiones. Posteriormente se detalló cuales son las funcionalidades de cada módulo.

Fase 2: Obtención y prueba de la API

La segunda fase, consta de dos procesos. Un primer proceso de investigación, donde se analizó una serie de empresas que proporcionaban una API para la obtención de los datos financieros. Una segunda de pruebas, en la cual se comprobaba su correcto funcionamiento así como las limitaciones de cada una. Para finalmente elegir la que mejor se adaptase al proyecto.

Fase 3: Desarrollo de la aplicación

En la tercera etapa, se llevó a cabo el desarrollo de la aplicación la cual se dividió en 4 iteraciones. Se realizó la implantación de los módulos gestión de cuentas de usuario, administración de los datos, área de trading e historial de trading.

Fase 4: Pruebas y solución de errores

En la fase cuatro, se realizaron pruebas de usuario de la aplicación, probando todas las funcionalidades para corroborar su correcto funcionamiento y se solucionaron aquellos errores que iban apareciendo.

Fase 5: Presentación al tutor

En la quinta fase se hizo una presentación del funcionamiento la aplicación al tutor, modulo por modulo.

1.3.2. Organización del trabajo

Para realizar el proyecto se ha establecido una metodología de trabajo iterativa incremental. Es decir, se dice que es incremental ya que en cada entrega se añaden nuevas funcionalidades, e iterativa debido a que sobre las funcionalidades existentes se realizan mejoras.

Cada iteración se finalizaba con una reunión en la cual se verificaba el trabajo realizado, pasando a marcar los objetivos de la siguiente.

Por otro lado, la necesidad de llevar un seguimiento del desarrollo del software hizo que fuese necesario el uso de GitHub como sistema de control de versiones.

Capítulo 1

Introduction

Investing in the stock market is an activity that has captured the attention of individuals and entities alike. The pursuit of opportunities to maximize capital returns has led to the development of various strategies and approaches. Among these approaches, technical analysis has established itself as an essential tool in investment decision-making, relying on the study of price charts, patterns, and volume to predict future market movements.

1.1. Motivation

The volatile and dynamic nature of financial markets presents investors with a challenging and ever-changing environment. The quest for patterns and trends in price movements has become an ongoing pursuit for those looking to seize investment opportunities. However, comprehensive analysis of historical data and the interpretation of technical indicators can be labor-intensive. Additionally, a thorough strategy testing (backtesting) is necessary to determine in which scenarios it performs well and in which it doesn't.

Motivated by the need to streamline the decision-making process for investors of all levels, the idea of developing an application that simplifies and automates part of this process was conceived. The application is based on the premise of using technical indicators as signals to identify potential buying and selling points in the stock market, providing easy

and fast access to historical data for analysis

1.2. Objective

The main objective is to develop a basic application to assist in stock investment decision-making, using technical indicators as the basis for purchase recommendations. To determine selling points, risk management principles are use.

Based on this objective, the following subobjectives are proposed:

- Establish a general understanding of the objectives of technical analysis and its application in investment decision-making.
- Select an appropriate set of technical indicators that will be used as the basis for purchase signals.
- Provide a target price to establish selling points.
- Design and implement the application, ensuring its functionality and the ability to generate investment recommendations.
- Evaluate the effectiveness of the application through tests utilizing historical market data and executing the recommendations.
- Provide an analysis of the obtained results, discussing the limitations of the application, potential improvements, and its possible utility in the real financial environment."

1.3. Methodology

The project implementation has been carried out in accordance with a project plan and a work organization, both of which are described in the following sections.

1.3.1. Fases del proyecto

For the project development, a preliminary phase was conducted to gather requirements. This helped define the scope of the application, determining the modules that would be included. In the subsequent phases, the program functionality was developed and tested, and in the final phase, the project report was prepared. The table 1.1 shows the duration of each project phase. Additionally, a Gantt chart depicting this information visually is presented in Figure 1.1.

Phase	**Duration date**
Research and Requirement Gathering	19/09/2022 - 19/10/2022
API Acquisition and Testing	20/10/2022 - 18/11/2022
Application Development	19/10/2022 - 30/05/2023
Testing and Bug Resolution	01/06/2023 - 24/06/2023
Presentation to Supervisor	25/06/2023
Report Development	26/06/2023 - 03/09/2023

Cuadro 1.1: Phases in Project Development

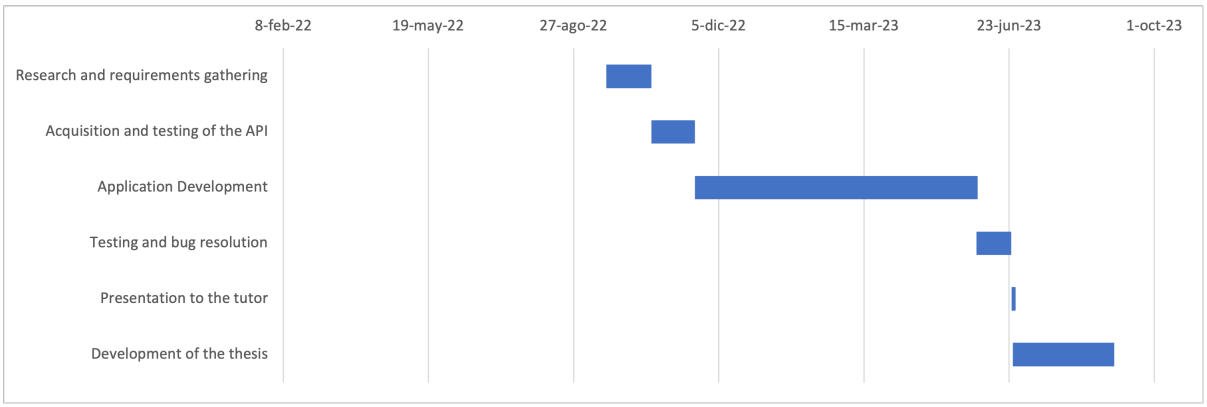


Figura 1.1: Gantt diagram

1.3.2. Work organization

Phase 1: Requirement Gathering

In the initial stage, an analysis was conducted to determine the principal modules that an application should have, in addition to those that a dedicated investment software should include. Subsequently, a detailed breakdown of the functionalities of each module was provided.

Phase 2: API Acquisition and Testing

The second phase consists of two processes. The first process involves research, where a series of companies providing an API for obtaining financial data were analyzed. The second process is testing, in which their proper functionality as well as the limitations of each were verified. This was done to ultimately choose the one that best suited the project.

Phase 3: Application Development

In the third stage, the development of the application was executed, which was divided into 4 iterations. The implementation of the user account management modules, data administration, trading area, and trading history was carried out.

Phase 4: Testing and Bug Resolution

In phase four, user testing of the application was conducted, testing all functionalities to verify their proper operation. Any errors that arose were addressed and resolved during this testing phase.

Phase 5: Presentation to Supervisor

In the fifth phase, a presentation of the application's functionality was given to the supervisor, module by module.

Capítulo 2

Estado del arte

En esta capítulo se exponen algunas aplicaciones que comparten funcionalidades similares con el sistema creado en este proyecto.

2.1. MetaTrader 4

Esta plataforma de trading permite la creación y ejecución de asesores expertos (EAs). Son programas que funcionan en la plataforma de MetaTrader 4 [5], se emplean para operar los mercados financieros mediante algoritmos, es decir, trading automático diseñado con los gustos y preferencias de cada usuario.

Se encargan de encontrar oportunidades en función de parámetros establecido por el usuario, notificando o abriendo la posición de forma automática. Una vez que la posición se ha abierto se establecen las condiciones de cierre.

2.2. Thinkorswim

Broker que permite operar diferentes mercados. Ofrece datos en tiempo real, gráficos, estudios técnicos y mas utilidades. Dispone de una sección ”demo”que permite testear en un entorno simulado diferentes técnicas y estrategias.[10]

2.3. TradingView

En esta plataforma se pueden realizar análisis de todos los mercados financieros del mundo, ofreciendo a los usuarios gráficos y herramientas. [11].

A través de su función Pine Script el cual es un lenguaje de programación desarrollado por la misma empresa. Mediante un editor incorporado en la plataforma los usuarios pueden escribir indicadores y estrategias personalizadas que pueden ser automatizadas para general señales de compra y de venta.

2.4. eToro

Empresa que permite acceder a varios mercados, es decir, es un broker. Cuenta con una sección denominada **trading social**, los usuarios forman una comunidad en donde exponen sus movimientos permitiendo que otros usuarios puedan copiar automáticamente dichos movimientos de otros inversores.

La empresa denomina esta acción **CopyTrader**, la cual permite imitar operaciones en tiempo real. [3]

2.5. QuantConnect

Es un entorno de código abierto, de trading algorítmico diseñado en la nube. Permite a los usuarios crear y probar algoritmos de trading mediante librerías escritas en Python y C#. Ya sea para hacer backtesting u operar en vivo. [9]

Además dispone de un entorno de aprendizaje, tanto para los conocimientos de programación como de funcionamiento de la plataforma.

Capítulo 3

Tecnología empleada

3.1. Lenguaje de programación

El lenguaje de programación que se ha seleccionado para la creación de esta aplicación ha sido Python.
El motivo principal para la elección de este lenguaje sobre los otros existentes, ha sido que en Python la manipulación de datos es mediante el uso de librerías como Numpy, la cual hace que sea sencillo. Además de las numerosas APIs y librerías que existen.

3.2. Qt Designer

Es una herramienta gráfica para el diseño de interfaces de usuario (UI). Qt Designer se caracteriza por facilitar la creación y el diseño de las interfaces de usuario de manera visual.

Permite a los programadores diseñar ventanas, botones, menús, etc. Sin tener que escribir el código manualmente.

Qt Designer proporciona una interfaz gráfica en la cual se puede arrastrar y soltar elementos de la interfaz a crear, esto reduce de manera muy significativa la necesidad de codificar.

Permite configurar las propiedades de los elementos de nuestra aplicación como el tamaño, la ubicación, los colores. Directamente desde la interfaz de la aplicación.
Una vez que se haya terminado de diseñar la interfaz se puede generar de forma automática el código en C++ o Python que corresponde con la interfaz que se ha creado. [4].

3.3. PySide2

Es un conjunto de bibliotecas la cual proporciona una interfaz de Python para el uso de Qt. Es decir, PySide2 permite a los desarrolladores utilizar Qt desde Python, esto permite la creación de aplicaciones con interfaz gráficas complejas desde un entorno de alto nivel como lo es Python.[8].

3.4. Alpha Vantage

Proveedor de datos financieros el cual ofrece gran cantidad de información y servicios relacionados con el mercado de valores y las finanzas. Los datos que ofrece Alpha Vantage se realiza mediante su API (Interfaz de Programación de Aplicaciones). [1].

Esta API ofrece datos históricos de activos, permitiendo a los usuarios analizar el rendimiento pasado además de poder realizar análisis técnico.
Además incorpora una amplia gama de indicadores técnicos los cuales se pueden usar para evaluar las tendencias de los mercados.

3.5. TA Libreria

Libreria de análisis técnico hecha por **Bukosabino** cuya licencia es MIT. La cual permite el uso de DataFrames de la librería Panda de Python para la gestión de series temporales de datos. De esta forma la gestión de varios datos que contienen los precios de un activo no resulta complejo. [2].

3.6. MySQL

Es un sistema de gestión de bases de datos relacionales, el cual se utiliza para almacenar, recuperar y administrar datos.
Los datos se organizan en tablas, con filas y columnas. Que sea relacional significa que las tablas se relacionan entre otras a través de claves primarias y claves foráneas.

El lenguaje que se utiliza es SQL, el cual permite realizar consultas, inserciones, actualizaciones y eliminaciones en las bases de datos.

3.7. PyCharm

PyCharm es un IDE (Entorno de desarrollo integrado) para el desarrollo de programas en Python. Este entorno de desarrollo incorpora diversas características como la posibilidad de la gestión de las bases de datos, la integración del control de presiones, la posibilidad de la depuración del código, entre otras funcionalidades. [7]

3.8. GitHub

Plataforma que permite la colaboración y el almacenamiento en el desarrollo de software la cual incorpora un control de versiones Git.
El control de versiones permite trazar los cambios que se han hecho en una aplicación a la hora de ser desarrollada permitiendo de esta manera el trabajo en equipo.
La funcionalidad que cabe destacar en este proyecto ha sido la de los repositorios, donde se almacena y se organizan el código fuente así como los archivos necesarios para el correcto funcionamiento del proyecto.

Capítulo 4

Especificación de requisitos

En esta sección se describe las características, restricciones y funcionalidades de la aplicación que se va a diseñar. La especificación del sistema que se va a diseñar es importante ya que sirve como base para definir que debe hacer el software y cómo debe comportarse. A continuación se van a describir los actores y los casos de uso asociados a la aplicación.

4.1. Actores

En esta sección se definen los actores que intervienen en la aplicación:

- **Usuarios registrados:** estos usuarios podrán hacer uso de todas las funcionalidades de la aplicación.
- **Usuarios no registrados:** estos usuarios no podrán acceder a las funcionalidades de la aplicación, y solo podrán crear una cuenta para poder usar las funciones.
- **Administrador:** es el usuario encargado de realizar tareas de mantenimiento en la aplicación

4.2. Módulos

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) Gestión de cuentas de usuario
    - Registrar usuario
    - Login
    - Logout
    - Modificar perfil
    - Darse de baja
2) Área de trading
    - Establecer saldo cuenta
    - Selección de activo
    - Realizar compras/ventas
    - Visualización de datos técnicos
3) Historial de trading
    - Visualizar historial
    - Filtrar historia
4) Administrar datos de la aplicación
    - Añadir activos
    - Añadir elementos técnicos
    - Gestión de cuenta de usuario

4.2.1. Gestión de cuentas de usuario

Este módulo es el encargado de administrar quien puede hacer uso de la aplicación, podrán todos aquellos usuarios que se hayan creado una cuenta. Una vez estén registrados se les permitirá realizar las funciones de inicio de sesión, cerrar sesión así como modificar los datos personales de cada uno.

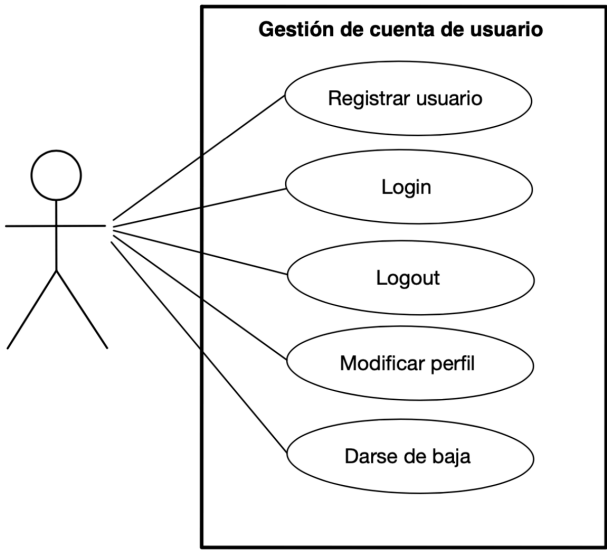


Figura 4.1: Diagrama de casos de uso módulo gestión cuentas de usuario

A continuación se describirá cada caso de uso de este módulo

Requisitos	**Registrar usuario**	
Identificador	3.1.1	
Prioridad	Alta	
Precondición	No estar registrado	
Descripción	El usuario puede darse de alta al sistema.	
Entrada	Nombre de usuario, nombre y apellidos, correo electrónico, contraseña, preguntas sobre experiencia y preguntas psicológicas.	
Salida	Se informa que se ha creado bien la cuenta ademas del tipo de usuario que es junto con una breve descripción.	
Procedimientos	Pasos	Acciones
	1	En la pagina principal aparece la opción registrar, el usuario accede a esta sección con un click
	2	Se muestra un formulario con la información requerida.
	3	Una vez rellenado por el usuario se valida
	4	Se verifica la información y se procede al registro.
	5	Muestra información que se ha creado con éxito además del tipo de cuenta
Postcondición	Cuenta de usuario creada con éxito	
Excepciones	Pasos	Acciones
	1	Cuenta de usuario existente o por correo electrónico repetido o nombre de usuario
	2	Información incorrecta, contraseña no cumple con los criterios o correo electrónico
Actores	Administrador, usuario no registrado	

Figura 4.2: Caso de uso registrar usuario

Requisitos	**Login**	
Identificador	3.1.2	
Prioridad	Alta	
Precondición	Ha de estar registrado el usuario que se intenta loggear	
Descripción	Se introduce correo electrónico y contraseña para hacer el inicio de sesión	
Entrada	Correo electrónico y contraseña	
Salida	Mensaje de éxito	
Procedimientos	Pasos	Acciones
	1	Se muestra un formulario con los datos de inicio de sesión
	2	Se rellena el formulario y se hace click a iniciar sesión
	3	Se verifica si la información es correcta
Postcondición	Datos coinciden con usuario registrado	
Excepciones	Pasos	Acciones
	1	Datos incorrectos
Actores	Admin, usuario registrado	

Figura 4.3: Caso de uso Login

Requisitos	**Logout**	
Identificador	3.1.3	
Prioridad	Alta	
Precondición	Estar con la sesión iniciada	
Descripción	Se cierra la sesión	
Entrada	Solicitud de cerrar sesión	
Salida	Mensaje de confirmación	
Procedimientos	Pasos	Acciones
	1	Se hace click en cerrar sesión
	2	El sistema comprueba la petición y procede a cerrar sesión
Postcondición	Sesión cerrada	
Excepciones	N/A	
Actores	Admin, usuario registrado	

Figura 4.4: Caso de uso Logout

Requisitos	**Modificar perfil**	
Identificador	3.1.4	
Prioridad	Media	
Precondición	Usuario con la sesión iniciada	
Descripción	Se puede cambiar los datos personales así como revaluar los datos psicológicos	
Entrada	Nuevos datos	
Salida	Mensaje de éxito	
Procedimientos	Pasos	Acción
	1	Seleccionar lo que se quiera modificar
	2	Rellenar con los nuevos datos
	3	Confirmar los cambios
Postcondición	Datos modificados con éxito	
Excepciones	Pasos	Acción
	1	Los datos nuevos no cumplen con los requerimientos solicitados
Actores	Usuario Registrado	

Figura 4.5: Caso de uso modificar perfil

4.2.2. Área de trading

El usuario puede simular el funcionamiento de la aplicación con dinero ficticio, el saldo se verá incrementado o disminuido dependiendo del éxito de las operaciones. También se podrán obtener parámetros técnicos del activo para conocer los valores según temporalidad.

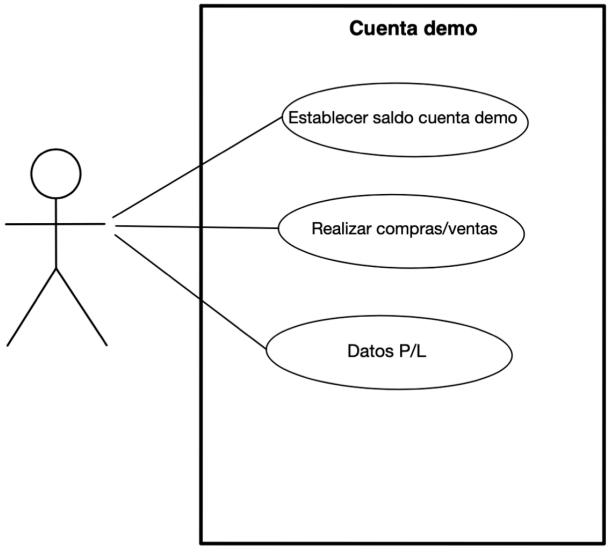


Figura 4.6: Diagrama de caso de uso cuenta demo

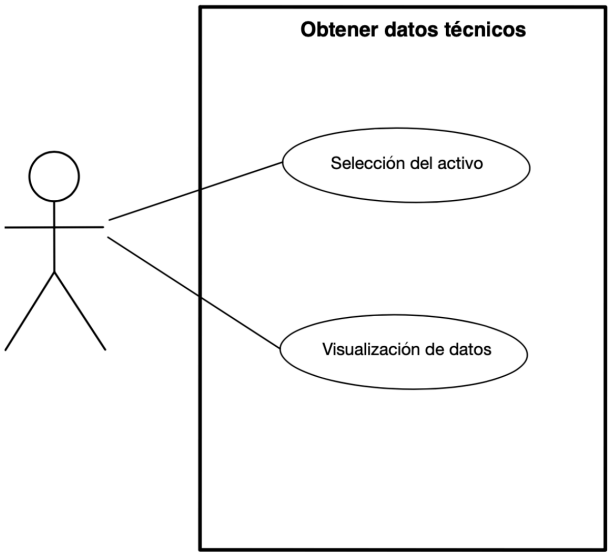


Figura 4.7: Diagrama de caso de uso obtener datos

Vamos a describir los casos de uso de este módulo.

Requisitos	**Establecer saldo cuenta demo**	
Identificador	3.2.1	
Prioridad	Media	
Precondición	Estar registrado y en la sección demo	
Descripción	Se puede establecer el saldo para la cuenta	
Entrada	Cifra numerica	
Salida	N/A	
Procedimientos	Pasos	Acción
	1	Seleccionar cambiar saldo
	2	Establecer nuevo saldo y confirmar
Postcondición	Saldo modificado	
Excepciones	Pasos	Acción
	1	Datos de entrada no son números
Actores	Usuario registrado	

Figura 4.8: Caso de establecer saldo

Requisitos	**Selección de activo**	
Identificador	3.2.2	
Prioridad	Media	
Precondición	Estar con la sesión iniciada	
Descripción	Dado un activo se puede visualizar datos de interés a la hora de operar	
Entrada	Click seleccionando el activo elegido	
Salida	Datos sobre ese activo	
Procedimientos	Pasos	Acción
	1	Seleccionar activo
	2	Hacer click para continuar
Postcondición	N/A	
Excepciones	Pasos	
	1	Componentes técnicas no disponibles
Actores	Usuario registrado	

Figura 4.9: Caso de selección de activo

Requisitos	**Realizar compra/ventas**	
Identificador	3.2.3	
Prioridad	Media	
Precondición	Estar en sección demo	
Descripción	Se puede simular compra y venta junto con la modificación del saldo	
Entrada	Datos de compra y venta a un precio	
Salida	N/A	
Procedimientos	Paso	Acción
	1	Número de acciones y precio de venta o compra
	2	Ejecutar compra o venta
Postcondición	Mensaje de éxito	
Excepciones	Paso	Acción
	1	Datos incorrectos, balance insuficiente
Actores	Usuario registrado	

Figura 4.10: Caso de uso realizar compra/venta

Requisitos	**Visualización de datos técnicos**
Identificador	3.2.4
Prioridad	Media
Precondición	Haber elegido un activo
Descripción	Muestra todos los datos referentes al análisis técnico según temporalidad seleccionada
Entrada	El componente en concreto que queremos que se muestre
Salida	Datos
Procedimientos	N/A
Postcondición	N/A
Excepciones	N/A
Actores	Usuarios registrados

Figura 4.11: Caso de uso visualizar datos técnicos

4.2.3. Historial de trading

En esta sección el usuario podrá ver el listado de operaciones realizadas, se podrán clasificar por tipo, orden de compra u orden de ventas.

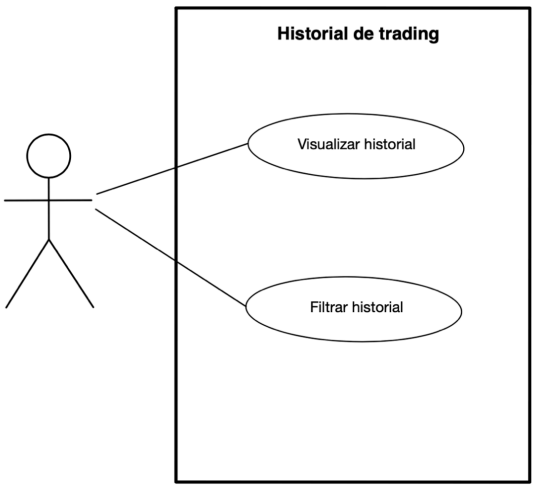


Figura 4.12: Diagrama de caso de uso historial

Requisitos	**Mostrar historia y filtrar datos**	
Identificador	3.3.1	
Prioridad	Media	
Precondición	Tener operaciones realizadas	
Descripción	Muestra una lista con todas las operaciones que el usuario ha realizado	
Entrada	Click para ir a la sección del historial	
Salida	Lista con los datos referentes al historial	
Procedimientos	Pasos	Acción
	1	Click para ir a la sección de historial
	2	Seleccionar tipo de datos que queremos que nos salga (Compra, venta o todos)
Postcondición	N/A	
Excepciones	Pasos	Acción
	1	No existan datos
Actores	Usuarios registrados	

Figura 4.13: Caso de uso historial

4.2.4. Administrar datos de la aplicación

La persona encargada de administrar la aplicación tendría la capacidad de añadir nuevos activos, nuevos componentes para el análisis técnico. También se encargará de la base de conocimiento por lo que podrá eliminar cuentas de usuarios que estén registrados en la base de datos.

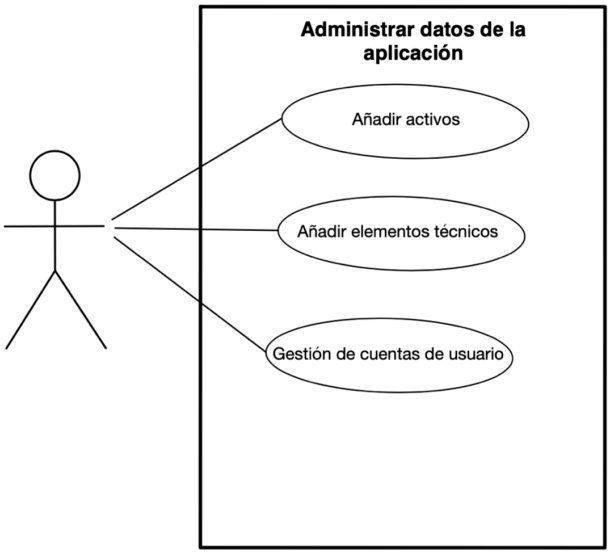


Figura 4.14: Diagrama de caso de uso administración de la aplicación

Requisitos	**Administrar datos**	
Identificador	3.5.1	
Prioridad	Alta	
Precondición	Ser usuario admin	
Descripción	La persona encargada del mantenimiento de la aplicación se encargara del buen funcionamiento del sistema.	
Entrada	Una selección	
Salida	N/A	
Procedimientos	Paso	Acción
	1	Selecciona la tarea que desea realizar.
Postcondición	N/A	
Excepciones	N/A	
Actores	Administrador	

Figura 4.15: Caso de uso administración de la aplicación

Capítulo 5

Arquitectura de la aplicación

En este capítulo se explicará la estructura de la aplicación y el modelo de datos que se ha utilizado.

5.1. Arquitectura de la aplicación

La aplicación implementa una arquitectura denominada ”sin servidor”(Serveless)

La Arquitectura sin servidor proporciona una visión de diseño de aplicaciones en la que los desarrolladores no necesitan preocuparse por la gestión de servidores o infraestructuras en su lugar deben enfocarse fundamentalmente en escribir código que impelente la lógica de la aplicación.
Los desarrolladores diseñan segmentos de código para que se puedan ejecutar de manera automática en respuesta a eventos específicos. Estas funcionalidades se ejecutan en infraestructuras de un proveedor de servicios en la nube.
De esta forma los clientes delegan en MySQL los datos de autenticación y los datos referentes a su cuenta de usuario.

Los clientes también obtienen información adicional a través de dos API REST de terceros.

Un API (application programming interface), son un conjunto de reglas que definen como los dispositivos y aplicaciones se pueden conectar y comunicarse entre ellos.
Un API REST es una API que se ajusta a los principios del REST (representational state transfer), el cual es un estilo arquitectónico.
Se hace referencia a interfaz de programación de aplicaciones (API) de terceros ya que quien la proporciona es una entidad externa que no tienen nada que ver con los que están desarrollando la aplicación.

En la figura 5.1 se muestra un esquema de la arquitectura empleado, como interactúa el cliente con el sistema.

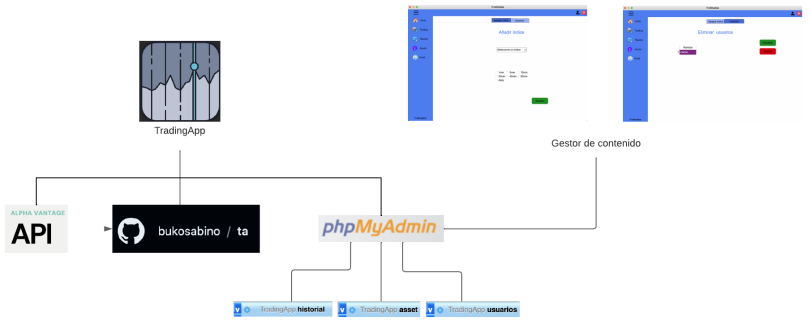


Figura 5.1: Esquema de la arquitectura

5.2. Modelo de datos

La información que se guarda en la base de datos es: información personal de los usuarios, historial de cada uno de ellos en una tabla diferente la cual se relaciona por el nombre de usuario. La tabla asset es gestionada por el admin donde se guarda que activo se ha añadido y la temporalidad.

Se ha elegido la base de datos *phpMyAdmin* como almacenamiento persistente ya que es un sistema sencillo de utilizar. Este sistema utiliza un sistema de base de datos SQL, lo cual permite guardar la información en tablas permitiendo que se relacionen entre si mediante un atributo en común.

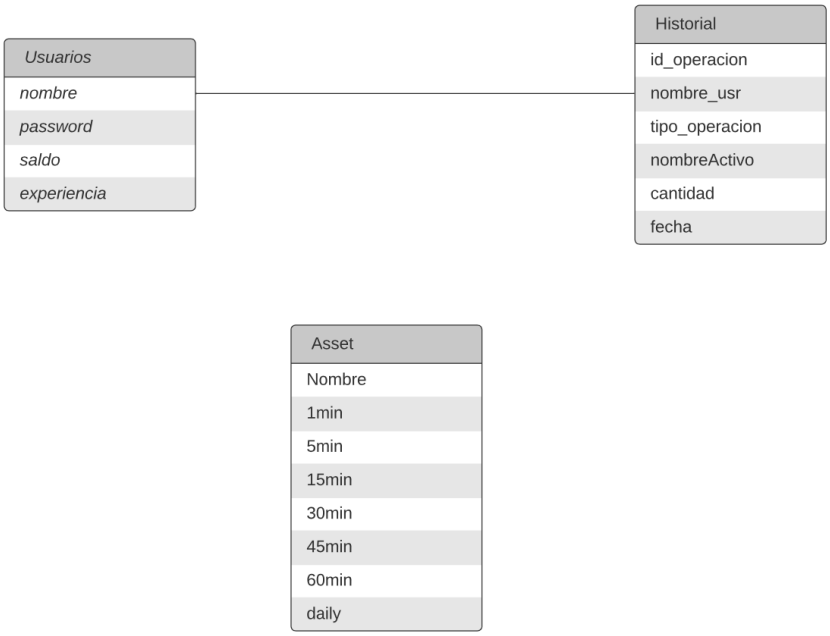


Figura 5.2: Diagrama UML del modelo de datos

Para poder establecer el modelo de datos que se va a utilizar, primero se ha de decidir el tipo de base de datos que mejor se adapte. Para posteriormente hacer una selección de la información que es necesaria almacenar. De esta forma se obtiene un modelo de datos como el de la Figura 5.2.

Las tablas usadas son:

5.2.1. Usuarios

En esta tabla se guarda la información con la cual se registra un nuevo usuario, siendo posible posteriormente modificarla desde el perfil. La clave se encripta para proporcionar mayor seguridad. El ”nombre”ha de ser único, es clave primaria. Este campo se utiliza para identificar de manera única cada fila de la tabla garantizando la integridad de los datos asegurando que no habrán datos duplicados.

5.2.2. Historial

Dado un usuario se almacena las operaciones que se realizan, la información que se guarda en esta tabla es toda la necesaria para saber identificar la operación, el activo, la cantidad, la fecha, si es de compra o de venta. De esta forma se puede llevar un registro del proceso de inversión.

5.2.3. Asset

Esta tabla es gestionada por el admin, almacena que activo y en que temporalidad está disponible en la aplicación. Ya que al agregar un activo previamente se descarga el histórico y se almacena en formato csv. De esta forma se evita la latencia que existe a la hora de descargar la información teniendo que esperar una única vez.

Capítulo 6

Implementación y diseño de la aplicación

En esta sección se explicaran los detalles de la implantación y el diseño de la aplicación, agrupando las funcionalidades según el módulo al que pertenecen:

- Gestión de cuentas de usuario
- Área de trading
- Historial de trading
- Administrar datos de la aplicación

6.1. Modulo gestión de cuentas de usuario

6.1.1. Inicio de sesión

Lo que se ve en la figura 6.1 es la primera pantalla que el usuario ve al iniciar la aplicación. Esta pantalla contiene dos campos donde se introduce el nombre de usuario y la contraseña respectivamente. En caso de que los datos ofrecidos coincidan con los almacenado en la base de datos se procede a acceder, en caso contrario muestra mensajes de error indicando que datos son los incorrectos.
Por otro lado, en caso de no estar registrado permite la opción de crear una cuenta, para ello se pulsa en el botón Registrar". Se direcciona a un formulario para crear una cuenta.

Por otro lado en la figura 6.2 se muestran funciones que interactúan entre la vista y la base de datos. Por un lado, la función *verificar* la cual extrae los datos del formulario de login y verifica que los datos sean correctos en caso contrario muestra mensajes de error. En la figura 6.3 se pueden ver la función de la base de datos, donde se le pasa como argumento un nombre de usuario a la función *getUsuario*, esta función realiza una consulta de tipo *SELECT* la cual devuelve todas las columnas para el nombre pasado por argumento. En caso negativo no devuelve nada lo cual significa que no está dado de alta. Las demás funciones de la figura 6.3 son auxiliares devuelve un dato en concreto del usuario.

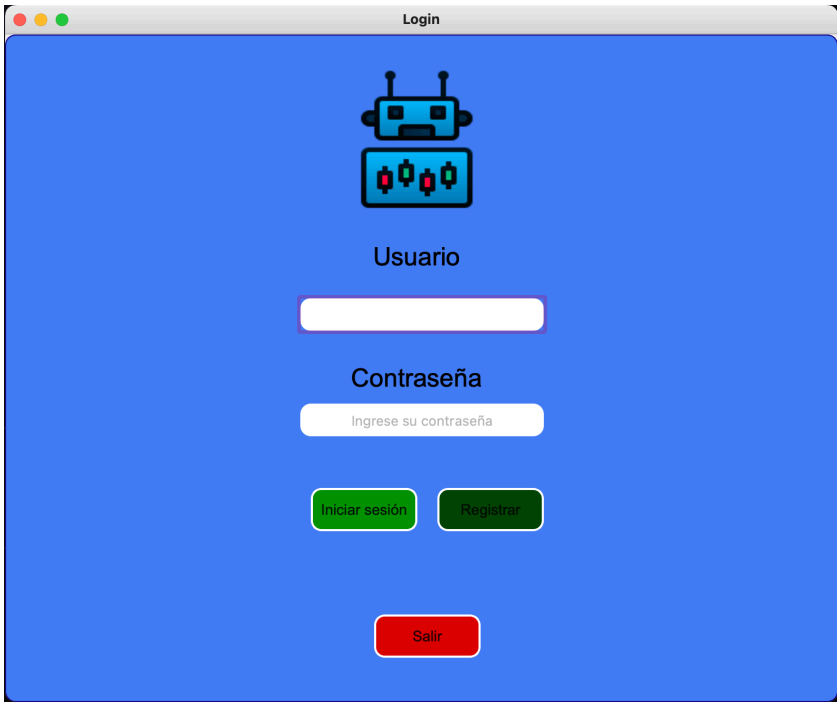


Figura 6.1: Pantalla de login

```
Marlon
def verificar(self):
    self.ui.contrasena_incorrecta.setText('')
    self.ui.usuario_incorrecto.setText('')
    user_entry = self.ui.usuario.text()
    passw_entry = self.ui.password.text()

    if self.correctUser(user_entry) and self.correctPassword(passw_entry, user_entry):
        self.panel = MainPanel(user_entry)
        self.hide()
        self.panel.show()
    else:
        self.ui.contrasena_incorrecta.setText('Contraseña incorrecta')
        self.ui.usuario_incorrecto.setText('Usuario incorrecto')

Marlon
def correctUser(self, usuario):
    estado = True

    if not self.dao.existeUsuario(usuario):
        estado = False
    return estado

Marlon
def correctPassword(self, password, userName):
    same = False
    encrypted = self.dao.get_encryptedPassword(userName)

    if check_password_hash(encrypted, password):
        same = True

    return same
```

Figura 6.2: Fragmento de código Login

```
Marlon
def getUsuario(self, name):
    if self.conexion.is_connected():
        try:
            cursor = self.conexion.cursor()
            sql = "SELECT * FROM usuarios where nombre = %s"
            cursor.execute(sql, [name])
            resultado = cursor.fetchone()
            cursor.close()
            return resultado
        except Error as ex:
            print("Error al intentar la conexión: {0}".format(ex))

new *
def existeUsuario(self, userName):
    usuario_bd = self.getUsuario(userName)

    if usuario_bd is None:
        return False

    return True

new *
def correct_password(self, password, userName):
    same = False
    encrypted = self.get_encryptedPassword(userName)
    if check_password_hash(encrypted, password):
        same = True

    return same

new *
def get_encryptedPassword(self, name):
    password = self.getUsuario(name)
    return password[1]
```

Figura 6.3: Fragmento de código Login en consultas SQL

6.1.2. Registrar usuario

A esta pantalla se accede desde la de login, como se puede ver en la figura 6.4 consta de tres campos, por un lado el usuario el cual debe ser único, se comprueba que no existe. Si el nombre de usuario existe se muestra un mensaje de error informando que ese nombre ya está registrado y no deja avanzar. Por otro lado, se puede ver dos campos para la contraseña de esta manera corroboramos que el usuario introduce la misma contraseña. Si ambos contraseñas no coinciden muestra un menaje de error informando del error.
Una vez se han introducido todos los datos correctos se puede proceder al registro. Cuando se pulsa el botón de registrar se redirige a la página principal, donde se encuentra el Login figura 6.1.

En la figura 6.5 se pueden ver 3 funciones, la principal es *registrar* la cual es la que está entre el usuario y la base de datos. Primero se extrae los datos de los formularios y se les da formato. Posteriormente se procede a realizar dos comprobaciones, por un lado que el usuario no esté dado de alta y por otro que la clave coincida en ambos campos. En caso de algún error se notifica al usuario. Finalmente, en la figura 6.6 se puede ver la función que se comunica con la base de datos y realiza la consulta para registrar los datos del nuevo usuario. En este caso se realiza una consulta de tipo *INSERT*.

Figura 6.4: Pantalla de registrar

```
Marlon
def register(self):
    if self.isConfirmationPassword():
        userName_entry = self.ventana.usuario_rgs.text().strip()
        password_entry = self.ventana.password_rgs.text().strip()
        encrypted_password = self.encriptarPassword(password_entry)

        if not self.dao.existeUsuario(userName_entry):
            self.dao.registrarUsuario([userName_entry, encrypted_password])
            print("Se ha registrado el usuario")
            self.ventana.registroOK_label.setText('Se ha registrado con éxito')
            self.ventana.usuario_rgs.clear()
            self.ventana.password_rgs.clear()
            self.ventana.passwordConfir_rgs.clear()
            self.hide()
    self.ventana.registroOK_label.setText(' ')

Marlon
def isConfirmationPassword(self):
    status: bool = True
    self.ventana.passDiferente_incorrecto.setText('')
    password_entry = self.ventana.password_rgs.text()
    passwordConfir_entry = self.ventana.passwordConfir_rgs.text()

    if password_entry != passwordConfir_entry:
        status = False
        self.ventana.passDiferente_incorrecto.setText('Las contraseñas no coinci

    return status

Marlon
def encriptarPassword(self, password_text):
    encrypted_pass = generate_password_hash(password_text, 'sha256')
    return encrypted_pass
```

Figura 6.5: Fragmento de código de registrar

```
Marlon
def registrarUsuario(self, usuario):
    if self.conexion.is_connected():
        try:
            cursor = self.conexion.cursor()
            sql = "INSERT INTO usuarios (nombre, password) VALUES ('{0}', '{1}')"
            cursor.execute(sql.format(str(usuario[0]), usuario[1]))
            self.conexion.commit()
            cursor.close()
            print("¡Usuario Registrado!\n")
        except Error as ex:
            print("Error al intentar la conexión: {0}".format(ex))
```

Figura 6.6: Fragmento de código de la consulta SQL de registrar

6.1.3. Logout

En la parte superior de la pantalla se encuentra el botón para poder hacer el logout, por un lado desde el botón izquierdo de cerrar la ventana y por otro desde el botón rojo de la derecha ambos cierran sesión y la aplicación simultáneamente.
Estas acciones se configuran automáticamente, uno el sistema operativo y otro mediante Qt Designer respectivamente.

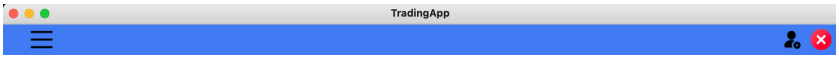


Figura 6.7: Sección para hacer Logout

6.1.4. Modificar perfil

Una vez se ha iniciado sesión, desde el menú izquierdo se puede acceder a la sección ”Perfil”. Desde aquí se puede acceder para poder cambiar los datos de la cuenta. En la figura 6.8 se puede cambiar el nombre de usuario, al igual que en el registro se comprueba que el nombre es único y no existe para poder permitir hacer el cambio. Por otro lado se debe hacer una comprobación para ello se solicita la contraseña si es correcta se procede a hacer el cambio, en caso contrario se muestra un mensaje de error.
En la imagen 6.9 se puede ver la la función la cual recoge los datos vistos en la figura anterior. Esta función llama a la función que realiza la consulta en la base de datos realizando un consulta del tipo *UPDATE* 6.10. De la misma forma se hace con el cambio de contraseña y modificación del saldo.

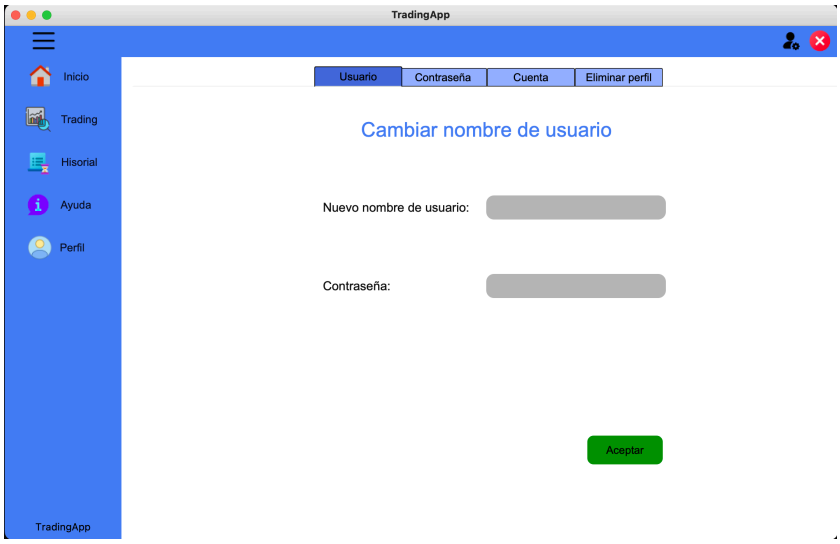


Figura 6.8: Pantalla de modificar datos

```
Marlon
def cambiarNombreUsuario(self):

    nuevo_user = self.mainWindow.usuarioNuevo_tab.text()
    confirm_pass = self.mainWindow.passwordConfirm_tab.text()

    if not self.dao.existeUsuario(nuevo_user):
        if self.dao.correct_password(confirm_pass, self.user_id):
            self.dao.cambiarNombre(nuevo_user, self.user_id)
            self.user_id = nuevo_user
            print("Nombre de usuario cambiado")
            self.mainWindow.usuarioNuevo_tab.clear()
            self.mainWindow.passwordConfirm_tab.clear()
        else:
            print("contraseña incorrecta")
    else:
        print("UserName en uso, elija otro")
```

Figura 6.9: Fragmento de código cambiar nombre

```
Marlon
def cambiarNombre(self, newName, odlName):
    if self.conexion.is_connected():
        try:
            cursor = self.conexion.cursor()
            sql = "UPDATE usuarios SET nombre = %s WHERE nombre = %s "
            cursor.execute(sql, [newName, odlName])
            self.conexion.commit()
            cursor.close()
        except Error as ex:
            print("Error al intentar la conexión: {0}".format(ex))
```

Figura 6.10: Fragmento de código de la consulta SQL cambiar nombre

```
Marlon
def cambiarpassword(self):

    old_pass = self.mainWindow.oldPasword_perfil_tab.text()
    new_pass = self.mainWindow.newPassword_perfil_tab.text()
    new_pass_confirm = self.mainWindow.newPasswordConfirm_perfil_tab.text()

    if new_pass == new_pass_confirm:
        if self.dao.correct_password(old_pass, self.user_id):
            encrypted_pass = generate_password_hash(new_pass, 'sha256')
            self.dao.cambiarPasword(encrypted_pass, self.user_id)
            print("contraseña cambiada correctamente")
            self.mainWindow.oldPasword_perfil_tab.clear()
            self.mainWindow.newPassword_perfil_tab.clear()
            self.mainWindow.newPasswordConfirm_perfil_tab.clear()
        else:
            print("Contraseña incorrecta")
    else:
        print("Nueva contraseña no coincide con confirmación")
```

Figura 6.11: Fragmento de código cambiar contraseña

```
Marlon
def cambiarPasword(self, newPass, userName):
    if self.conexion.is_connected():
        try:
            cursor = self.conexion.cursor()
            sql = "UPDATE usuarios SET password = %s WHERE nombre = %s '
            cursor.execute(sql, [newPass, userName])
            self.conexion.commit()
            cursor.close()
        except Error as ex:
            print("Error al intentar la conexión: {0}".format(ex))
```

Figura 6.12: Fragmento de código de la consulta SQL eliminar cuenta

6.1.5. Darse de baja

Desde esta pantalla 6.13 se permite al usuario eliminar su cuenta. Previamente se ha de indicar con el tick que se quiere borrar la cuenta además de introducir la contraseña para confirmar el proceso. Posteriormente se cierra la aplicación.

La función de la figura 6.14 se encarga de comprobar que se cumplen los criterios mostrados en la interfaz para proceder a la eliminación de la cuenta. Por otro lado, esta función llama a la función 6.15 la cual hace una consulta del tipo *DELETE* en la base de datos.

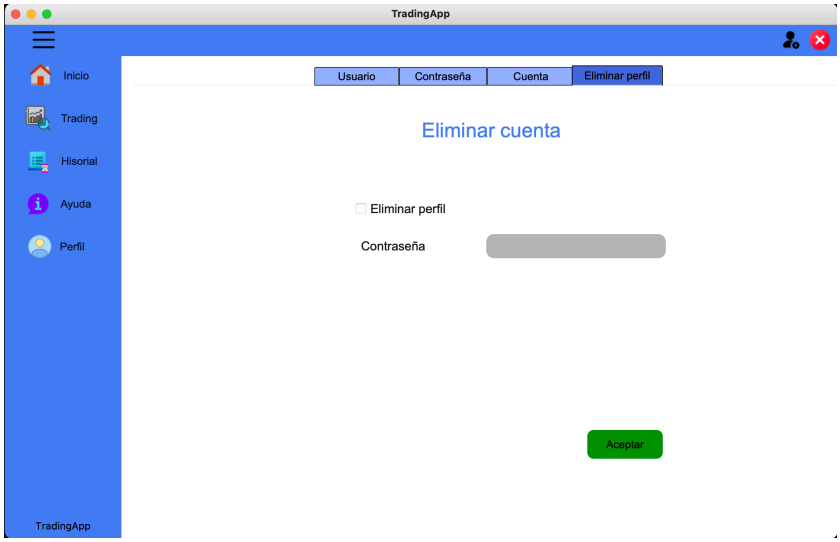


Figura 6.13: Pantalla de darse de baja

```
Marlon *
def eliminarCuenta(self):

    confirm_pass = self.mainWindow.passwordConfirmDelete_tab.text()

    if self.dao.correct_password(confirm_pass, self.user_id):
        if self.mainWindow.delete_box.isChecked():
            self.dao.deleteUser(self.user_id)
            print("Se ha eliminado la cuenta con éxito")
            self.mainWindow.passwordConfirmDelete_tab.clear()
        else:
            print("No se ha marcado la casilla")
    else:
        print("Contraseña incorrecta")
```

Figura 6.14: Fragmento de código de eliminar cuenta

Figura 6.15: Fragmento de código eliminar cuenta consulta SQL

6.2. Área de trading

En esta pantalla el usuario interactúa con la aplicación.

6.2.1. Realizar compras/ventas

La vista pasa toda la información que se recoge de la pantalla 6.16 a las funciones internas, encargadas de usar dichos datos. En la porción de código de la figura 6.17 se encarga de recopilar los datos necesarios para comenzar con el análisis, esta función es la intermediaria entre el usuario y la lógica de negocio. Por un lado recoge los datos enviados por el usuario mediante la interfaz y por otro abre para leer los datos históricos. Los datos que se devuelven están en formato de dataFrame el cual se pasa por parámetro para extraer los datos técnicos. Toda esta información es pasado como argumento de la función *tradingResult* 6.18, la cual se encarga de hacer la operativa, la compra y la venta respectivamente. Cada función devuelve lo referente al proceso de compra y venta.

La función que se ve en la figura 6.19, recibe un dataFrame con las posibles entradas así como un porcentaje para promediar, el nombre del activo y el monto total de la posición. Un dato importe a tener en cuenta es que se descarta todas aquellas operaciones que se hagan después de las 16h por bajada de la volatilidad del mercado y evitar un exceso de exposición. Cabe destacar que en cada compra que se realiza se registra en la base de datos para poder tener un historial. Los datos se devuelven en un array para su posterior uso y además para que puedan ser visualizados por el usuario.

Seguidamente está la función *ventaDia* 6.20, la cual determina el precio de venta en función de los datos devueltos por *comporasDia*. Dado que se sigue una operativa conservadora se establece en un 1,5 % el precio de venta en función del precio medio de compra. Al igual que en la compra, en esta también se registra en la base de datos. En la figura 6.21, esta la función encargada de conectar con la base de datos y realizar una consulta SQL del tipo *insert* para añadir los datos de la operación en la tabla historial.

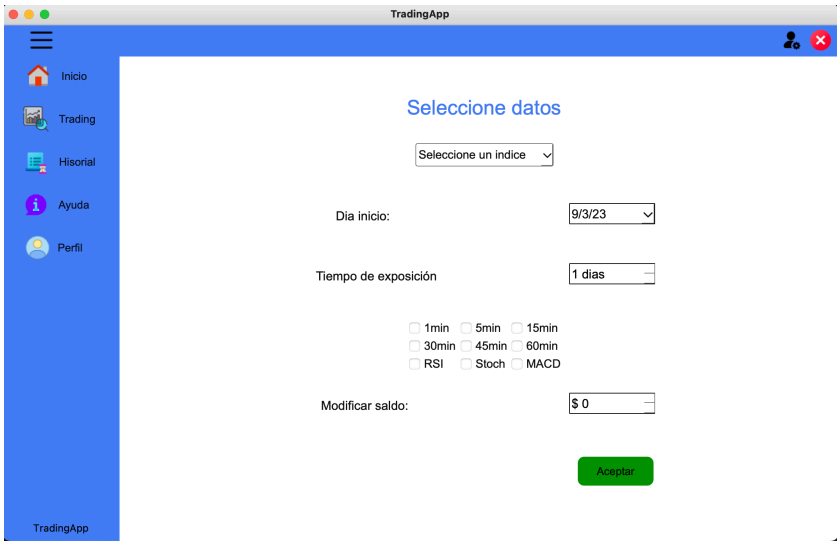
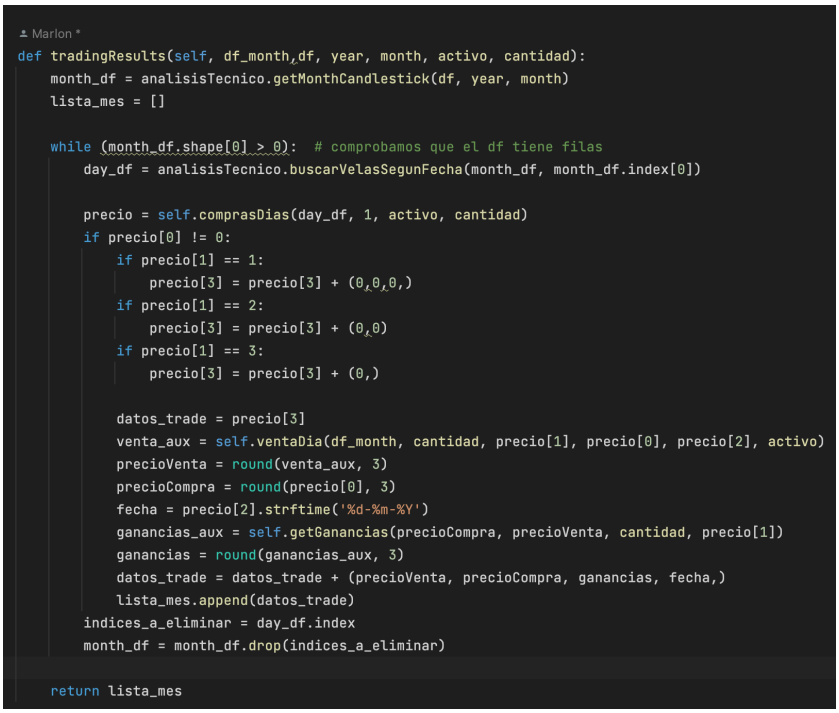


Figura 6.16: Pantalla selección de datos

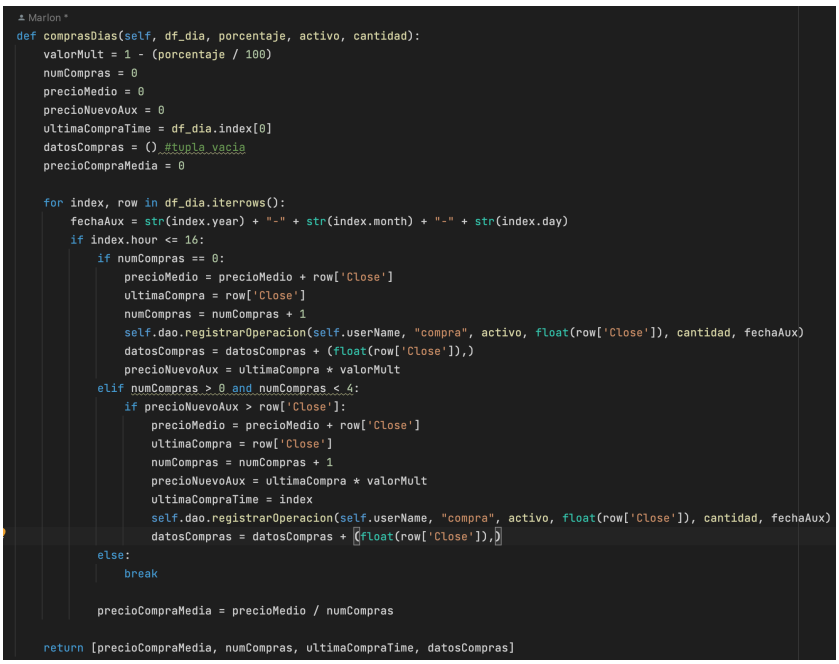
```
Marlon *
def accion(self):
    self.mainWindow.ventanas_trading.setCurrentWidget(self.mainWindow.page_resultados)
    asset = self.getAsset()
    timeFrame = self.getTimeFrame()
    fecha = self.getFecha()
    year = fecha.year
    month = fecha.month
    cantidad = self.getSbxCantidadPosicion()

    df = self.getAssetDataIntraday(timeFrame)
    df_technicalData = analisisTecnico.getDatosTecnicos(df)
    df_month = analisisTecnico.getMonthCandlestick(df, year, month)
    entradas = analisisTecnico.posiblesEntradas(df_technicalData)
    lista_datos = self.trading.tradingResults(df_month,entradas, year, month, asset, cantidad)
    self.mainWindow.tableWidget_3.setRowCount(len(lista_datos))
    filaTabla = 0

    for tupla in lista_datos:

        self.mainWindow.tableWidget_3.setItem(filaTabla, 0, QtWidgets.QTableWidgetItem(str(tupla[0])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 1, QtWidgets.QTableWidgetItem(str(tupla[1])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 2, QtWidgets.QTableWidgetItem(str(tupla[2])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 3, QtWidgets.QTableWidgetItem(str(tupla[3])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 4, QtWidgets.QTableWidgetItem(str(tupla[4])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 5, QtWidgets.QTableWidgetItem(str(tupla[5])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 6, QtWidgets.QTableWidgetItem(str(tupla[6])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 7, QtWidgets.QTableWidgetItem(tupla[7]))
        filaTabla += 1
```

Figura 6.17: Fracción de código donde se leen los datos



```
Marlon *
def tradingResults(self, df_month,df, year, month, activo, cantidad):
    month_df = analisisTecnico.getMonthCandlestick(df, year, month)
    lista_mes = []

    while (month_df.shape[0] > 0):  # comprobamos que el df tiene filas
        day_df = analisisTecnico.buscarVelasSegunFecha(month_df, month_df.index[0])

        precio = self.comprasDias(day_df, 1, activo, cantidad)
        if precio[0] != 0:
            if precio[1] == 1:
                precio[3] = precio[3] + (0,0,0,)
            if precio[1] == 2:
                precio[3] = precio[3] + (0,0)
            if precio[1] == 3:
                precio[3] = precio[3] + (0,)

            datos_trade = precio[3]
            venta_aux = self.ventaDia(df_month, cantidad, precio[1], precio[0], precio[2], activo)
            precioVenta = round(venta_aux, 3)
            precioCompra = round(precio[0], 3)
            fecha = precio[2].strftime('%d-%m-%Y')
            ganancias_aux = self.getGanancias(precioCompra, precioVenta, cantidad, precio[1])
            ganancias = round(ganancias_aux, 3)
            datos_trade = datos_trade + (precioVenta, precioCompra, ganancias, fecha,)
            lista_mes.append(datos_trade)
        indices_a_eliminar = day_df.index
        month_df = month_df.drop(indices_a_eliminar)

    return lista_mes
```

Figura 6.18: Fragmento de código de realizar compra/venta



```
Marlon *
def comprasDias(self, df_dia, porcentaje, activo, cantidad):
    valorMult = 1 - (porcentaje / 100)
    numCompras = 0
    precioMedio = 0
    precioNuevoAux = 0
    ultimaCompraTime = df_dia.index[0]
    datosCompras = () #tupla vacia
    precioCompraMedia = 0

    for index, row in df_dia.iterrows():
        fechaAux = str(index.year) + "-" + str(index.month) + "-" + str(index.day)
        if index.hour <= 16:
            if numCompras == 0:
                precioMedio = precioMedio + row['Close']
                ultimaCompra = row['Close']
                numCompras = numCompras + 1
                self.dao.registrarOperacion(self.userName, "compra", activo, float(row['Close']), cantidad, fechaAux)
                datosCompras = datosCompras + (float(row['Close']),)
                precioNuevoAux = ultimaCompra * valorMult
            elif numCompras > 0 and numCompras < 4:
                if precioNuevoAux > row['Close']:
                    precioMedio = precioMedio + row['Close']
                    ultimaCompra = row['Close']
                    numCompras = numCompras + 1
                    precioNuevoAux = ultimaCompra * valorMult
                    ultimaCompraTime = index
                    self.dao.registrarOperacion(self.userName, "compra", activo, float(row['Close']), cantidad, fechaAux)
                    datosCompras = datosCompras + (float(row['Close']),)
            else:
                break

            precioCompraMedia = precioMedio / numCompras

    return [precioCompraMedia, numCompras, ultimaCompraTime, datosCompras]
```

Figura 6.19: Fragmento de código de realizar compra

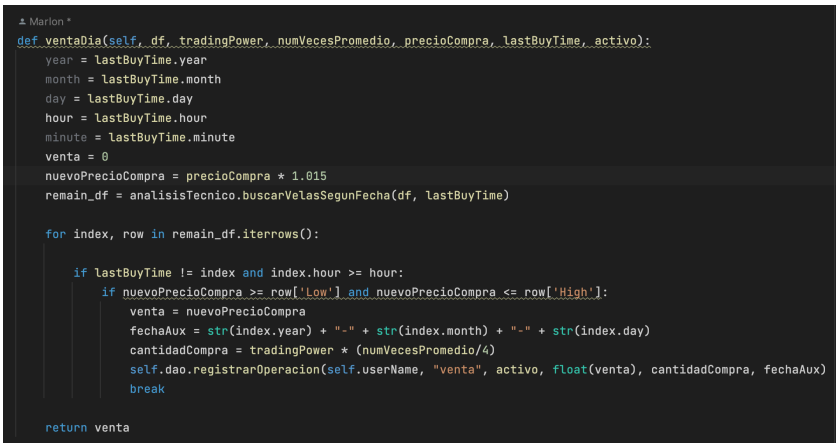


Figura 6.20: Fragmento de código de realizar las ventas

Figura 6.21: Fragmento de código de consulta SQL para registrar en historial

6.2.2. Visualización de datos técnicos

La figura 6.22 es lo que el usuario ve finalmente después del proceso descrito en el apartado anterior. Como se puede ver, en esta pantalla se visualizan los datos en formato de tabla la cual consta de ocho columnas. En las columnas de compras cuando aparece un cero es que no se ha realizado ninguna compra, del mismo modo en la columna de ventas si aparece un cero significa que no se ha alcanzado el precio de venta. Cuando no se alcanza el precio de venta se asume como perdida, el cual está en un 2 % (valor predeterminado) del precio de compra medio.

Por otro lado, en la figura 6.23 se puede ver el código que hace posible que se visualicen los datos. La función *TradingResult* descrita anteriormente es la que proporciona la información necesaria para poder mostrar la información al usuario. Esta información se almacena en forma de array de filas el cual se va recorriendo una a una.

Figura 6.22: Pantalla de los resultados

```
Marlon *
def accion(self):
    self.mainWindow.ventanas_trading.setCurrentWidget(self.mainWindow.page_resultados)
    asset = self.getAsset()
    timeFrame = self.getTimeFrame()
    fecha = self.getFecha()
    year = fecha.year
    month = fecha.month
    cantidad = self.getSbxCantidadPosicion()

    df = self.getAssetDataIntraday(timeFrame)
    df_technicalData = analisisTecnico.getDatosTecnicos(df)
    df_month = analisisTecnico.getMonthCandlestick(df, year, month)
    entradas = analisisTecnico.posiblesEntradas(df_technicalData)
    lista_datos = self.trading.tradingResults(df_month,entradas, year, month, asset, cantidad)
    self.mainWindow.tableWidget_3.setRowCount(len(lista_datos))
    filaTabla = 0

    for tupla in lista_datos:

        self.mainWindow.tableWidget_3.setItem(filaTabla, 0, QtWidgets.QTableWidgetItem(str(tupla[0])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 1, QtWidgets.QTableWidgetItem(str(tupla[1])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 2, QtWidgets.QTableWidgetItem(str(tupla[2])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 3, QtWidgets.QTableWidgetItem(str(tupla[3])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 4, QtWidgets.QTableWidgetItem(str(tupla[4])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 5, QtWidgets.QTableWidgetItem(str(tupla[5])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 6, QtWidgets.QTableWidgetItem(str(tupla[6])))
        self.mainWindow.tableWidget_3.setItem(filaTabla, 7, QtWidgets.QTableWidgetItem(tupla[7]))
        filaTabla += 1
```

Figura 6.23: Fragmento de código para la visualización de los datos

6.3. Historial de trading

El usuario puede ver las compras/ventas que se ha realizado a lo largo del tiempo, se almacena el histórico 6.24. En la figura 6.25 se encuentra la función encargada de mostrar el historial a través de la interfaz gráfica. Esta función a su vez llama a otra 6.26 que es la encargada de hacer una consulta en la base de datos para recuperar los datos del usuarios que tiene la sesión iniciada.

6.3.1. Visualizar historial

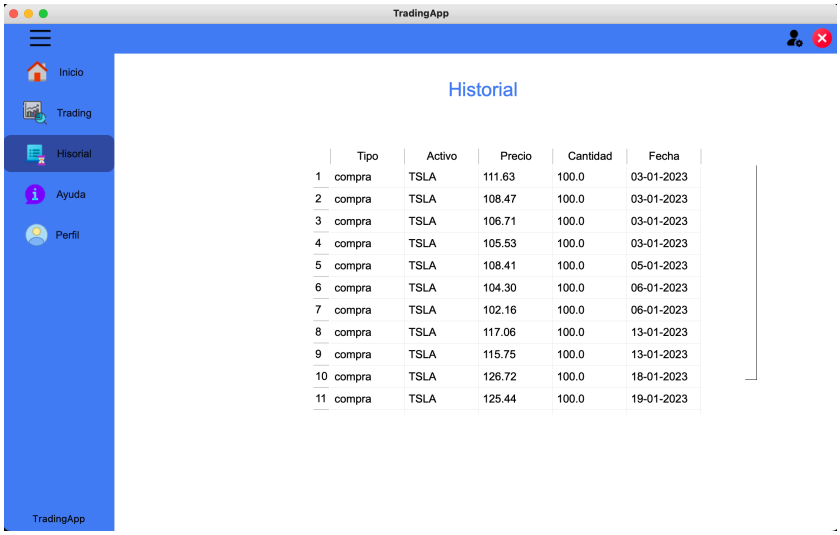


Figura 6.24: Pantalla historial

```
Marlon *
def mostrarHistorial(self):
    datos = self.dao.getHistorial(self.usrName)
    totalHistorial = len(datos)
    self.mainWindow.tableWidget_2.setRowCount(totalHistorial)
    filaTabla = 0

    for fila in datos:
        precio = str(fila[4])
        cantidad = str(fila[5])
        fecha = fila[6].strftime('%d-%m-%Y')
        self.mainWindow.tableWidget_2.setItem(filaTabla, 0, QtWidgets.QTableWidgetItem(fila[2]))
        self.mainWindow.tableWidget_2.setItem(filaTabla, 1, QtWidgets.QTableWidgetItem(fila[3]))
        self.mainWindow.tableWidget_2.setItem(filaTabla, 2, QtWidgets.QTableWidgetItem(precio))
        self.mainWindow.tableWidget_2.setItem(filaTabla, 3, QtWidgets.QTableWidgetItem(cantidad))
        self.mainWindow.tableWidget_2.setItem(filaTabla, 4, QtWidgets.QTableWidgetItem(fecha))
        filaTabla += 1
```

Figura 6.25: Fragmento de código de mostrar historial

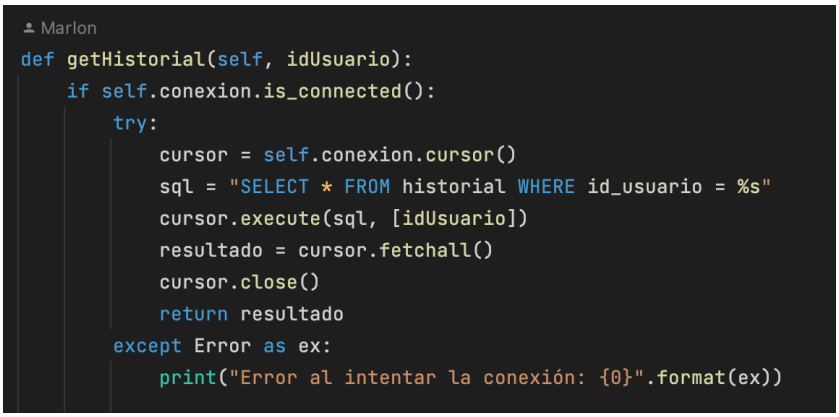


Figura 6.26: Fragmento de código de la consulta SQL para historial

6.4. Administrar datos de la aplicación

Es donde el administrador gestiona un uso correcto de las funcionalidades de la aplicación.

6.4.1. Añadir activos

Desde aquí 6.27 el administrador puede añadir nuevos datos de activos, de esta manera evitamos una descarga continua de todo el historia lo cual supone una espera de varios minutos para el usuario. El administrador selecciona en el desplegable el nombre del activo que quiere descargar, posteriormente elige la temporalidad en la que se quiere descargar los datos.

La figura 6.28 se encarga de la comunicación entre el administrador y la aplicación que a su vez se comunica con la API, para la obtención del histórico mediante la función que se ve en la figura 6.29 para una temporalidad en minutos y 6.30 para temporalidad diaria.

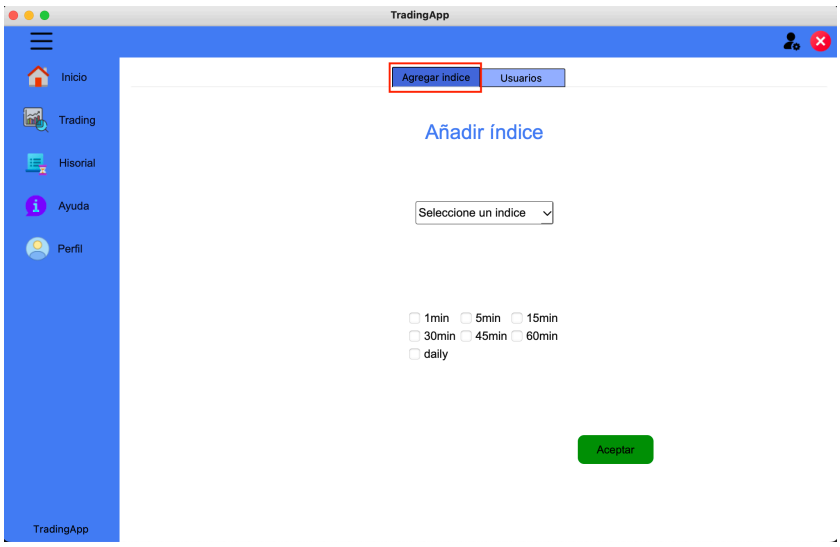


Figura 6.27: Pantalla añadir activo

Figura 6.28: Fragmento de código de añadir activo

Figura 6.29: Fragmento de código guardar datos

Figura 6.30: Fragmento de código guardar datos diarios

6.4.2. Gestión de cuenta de usuario

Desde esta pantalla 6.31 el administrador puede visualizar los usuarios que están dados de alta en el sistema. Desde ahí selecciona cual quiere dar de baja para eliminarlo de la base de datos.

Primero se muestra los usuarios existentes con la función 6.32, la cual hace una llamada a la base de datos para realizar una consulta que devuelva todos los usuarios registrados.

Posteriormente, la función 6.33 se encarga de realizar dicha acción, le da formato valido al string con el nombre de usuario, para pasárselo a la función encargada de hacer la consulta de tipo *DELETE* 6.34. Esta función ejecuta en la base de datos la eliminación del usuario elegido.

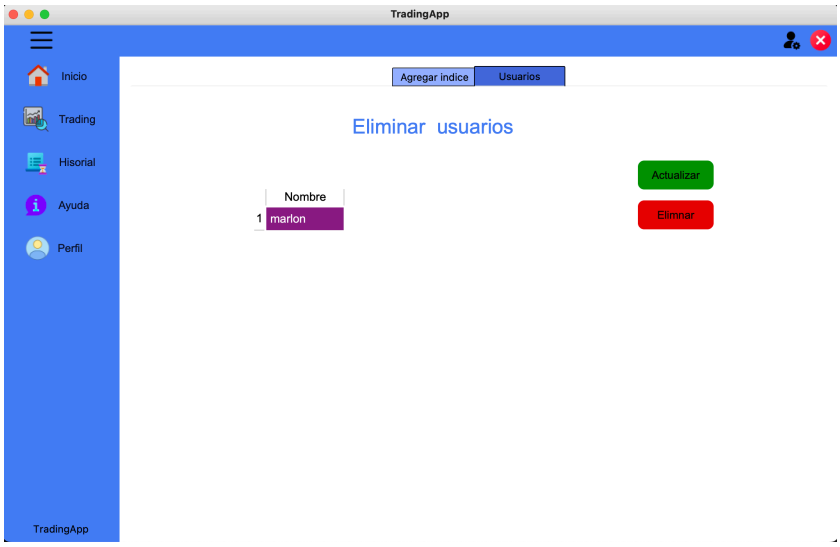


Figura 6.31: Pantalla gestión de cuentas de usuario

```
Marlon
def mostrarUsuarios(self):
    datos = self.dao.mostrarUsuarios()
    totalusers = len(datos)
    self.mainWindow.tableWidget.setRowCount(totalusers)
    filaTabla = 0

    for columna in datos:
        self.mainWindow.tableWidget.setItem(filaTabla, 0, QtWidgets.QTableWidgetItem(columna[0]))
        filaTabla += 1
```

Figura 6.32: Fragmento de código mostrar usuarios

```
def deleteUser(self):

    self.fila_seleccionada = self.mainWindow.tableWidget.currentRow()

    if self.fila_seleccionada != -1:
        nombre_usr = self.mainWindow.tableWidget.item(self.fila_seleccionada, 0).text()
        self.mainWindow.tableWidget.removeRow(self.fila_seleccionada)
        self.dao.deleteUser(nombre_usr)
```

Figura 6.33: Fragmento de código eliminar usuario

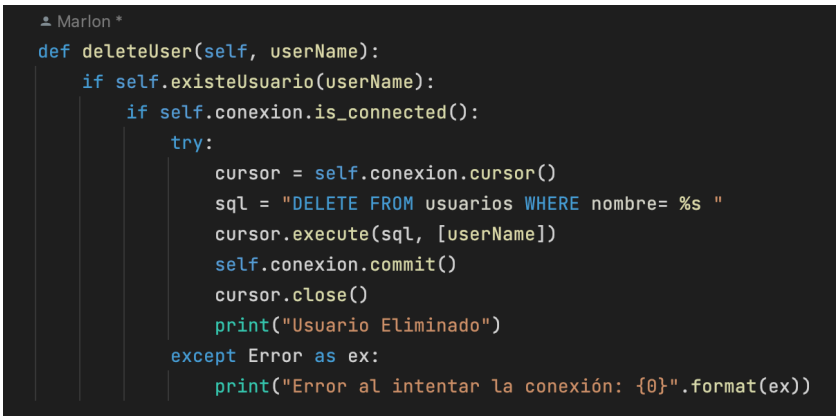


Figura 6.34: Fragmento de código de consulta SQL eliminar usuario

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Se ha desarrollado una aplicación de trading funcional que incorpora indicadores técnicos como el RSI y el Estocástico, así como reglas de compra y venta basadas en promedios y porcentajes de ganancias. Lo que se pretende con esta aplicación es realizar un proceso de automatización, permitiendo a los inversores a tomar decisiones basadas en reglas predefinidas en lugar de las emociones. Este tipo de operativa suele traer mejores resultados ya que es necesario seguir un planificación marcada y no salirse de ahí. Al establecer un porcentaje para el punto de venta permite establecer unos criterios para la gestión del riesgo. Esto es fundamental para proteger el capital y minimizar las perdidas.

Por otro lado, el administrador juega un papel importante ya que es el encargado de añadir nuevos índices, esta acción hace que se comunique con la API y descargue los datos históricos para que los usuarios puedan acceder en forma de lectura de un documento, de esta forma se reduce en gran cantidad el tiempo de acceso a los datos.

Los ficheros del proyecto se encuentran almacenados en el siguiente repositorio de GitHub: https://github.com/naadiee/TFG_qt

7.2. Trabajo futuro

Aunque se han podido alcanzar y desarrollar la mayor parte de los objetivos planteados al principio del proyecto en la fase de toma de requisitos. Se han encontrado nuevas características que se proponen a continuación como trabajo futuro:

- **Más datos técnicos.** Permitir a los usuarios incorporar en los resultados mas datos técnicos según sus preferencias, para que en base a eso muestro los resultados. Esto permite hacer simulaciones de una operativa.
- **Filtros de búsqueda en el historial.** Dado que los resultados del historial son bastantes y muy variados. Hacer que el usuario pueda obtener unos resultados en concreto mediante unos filtros de búsqueda.
- **Temporalidad de resultados.** Hacer una selección más amplia te la temporalidad de análisis, actualmente es de un mes entero.
- **Dar de alta a admin.** Permitir que mediante la interfaz se pueda dar de alta un administrador para que sea una acceso más fácil, actualmente se hace de forma interna mediante una consulta en phpMyAdmin.
- **Operativa con datos en tiempo real**. Poder hacer un análisis a tiempo real, para ello es necesario una plataforma de pago que proporcione los datos.

Capítulo 7

Conclusions and future work

7.1. Conclusions

A functional trading application has been developed that incorporates technical indicators such as RSI and Stochastic, as well as buying and selling rules based on averages and profit percentages. The goal of this application is to automate the trading process, allowing investors to make decisions based on predefined rules rather than emotions. This type of trading often yields better results since it requires following a predetermined plan and sticking to it. Setting a percentage for the selling point allows for risk management criteria to be established, which is crucial for protecting capital and minimizing losses.

On the other hand, the administrator plays an important role as they are responsible for adding new indices. This action involves communicating with the API and downloading historical data so that users can access it in the form of a document, significantly reducing data access time.

The project files are stored in the following GitHub repository: https://github.com/naadiee/TFG_qt

7.2. Future work

While most of the initially set project objectives have been achieved and developed during the requirements gathering phase, new features have been identified for future work:

- **More technical data:** Allow users to incorporate additional technical data into the results based on their preferences, enabling simulations of trading operations.
- **Search filters in the history:** Given the abundance and diversity of historical results, enable users to obtain specific results by implementing search filters.
- **Result timeframes:** Expand the selection of analysis timeframes, currently limited to one entire month.
- **Admin registration:** Enable user-friendly admin registration through the interface, as it currently requires internal setup via phpMyAdmin queries.
- **Real-time data trading:** Enable real-time analysis, requiring access to a paid platform that provides real-time data.

Bibliografía

[1] AlphaVantage. «API Documentation.» (), dirección: https://www.alphavantage.co/documentation/.

[2] Bukosabino. «Documentation.» (), dirección: https://technical-analysis-library-in-python.readthedocs.io/en/latest/ta.html#trend-indicators.

[3] eToro. (), dirección: https://www.etoro.com/es/copytrader/.

[4] Q. Group. «Documentation.» (), dirección: https://doc.qt.io/qt-6/qtwidgets-index.html.

[5] MetaTrader4. (), dirección: https://www.metatrader4.com/es/automated-trading.

[6] J. J. Murphy, *Análisis técnico de los mercados financieros*. 1986.

[7] PyCharm. «Documentation.» (), dirección: https://www.jetbrains.com/help/pycharm/getting-started.html.

[8] Pyside. «Documentation.» (), dirección: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/index.html.

[9] QuantConnect. (), dirección: https://www.quantconnect.com/docs/v2.

[10] Thinkorswim. (), dirección: https://tlc.thinkorswim.com/center.

[11] TradingView. (), dirección: https://www.tradingview.com.

Apéndice A

Guía de uso

A.1. Vista principal del administrador

A.1.1. Añadir activos

En esta pantalla A.1 el administrador puede añadir activos, es decir, se descarga los datos de los precios de un determinado activo en una temporalidad en concreto. De esta manera se genera una vez un csv con la información permitiendo a los usuarios leer este documento lo cual hace que no se tarde nada en la obtención de la información.

Primero, el admin de la lista de activos disponibles selecciona cual quiere añadir A.2, posteriormente selecciona que temporalidad quiere que se generen los datos, como se ve en la figura A.3. Finalmente, le da al botón aceptar par que se genere. Dependiendo de la temporalidad se tardará mas o menos.

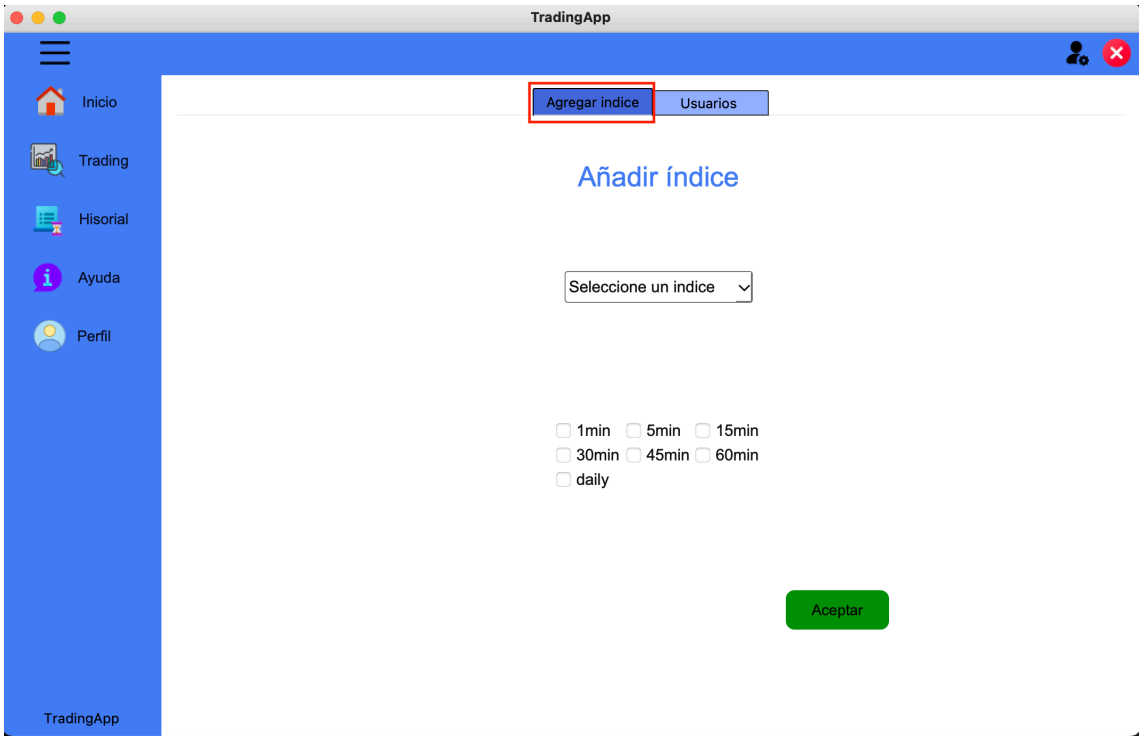


Figura A.1: Pantalla añadir activo

Figura A.2: Pantalla añadir activo selección

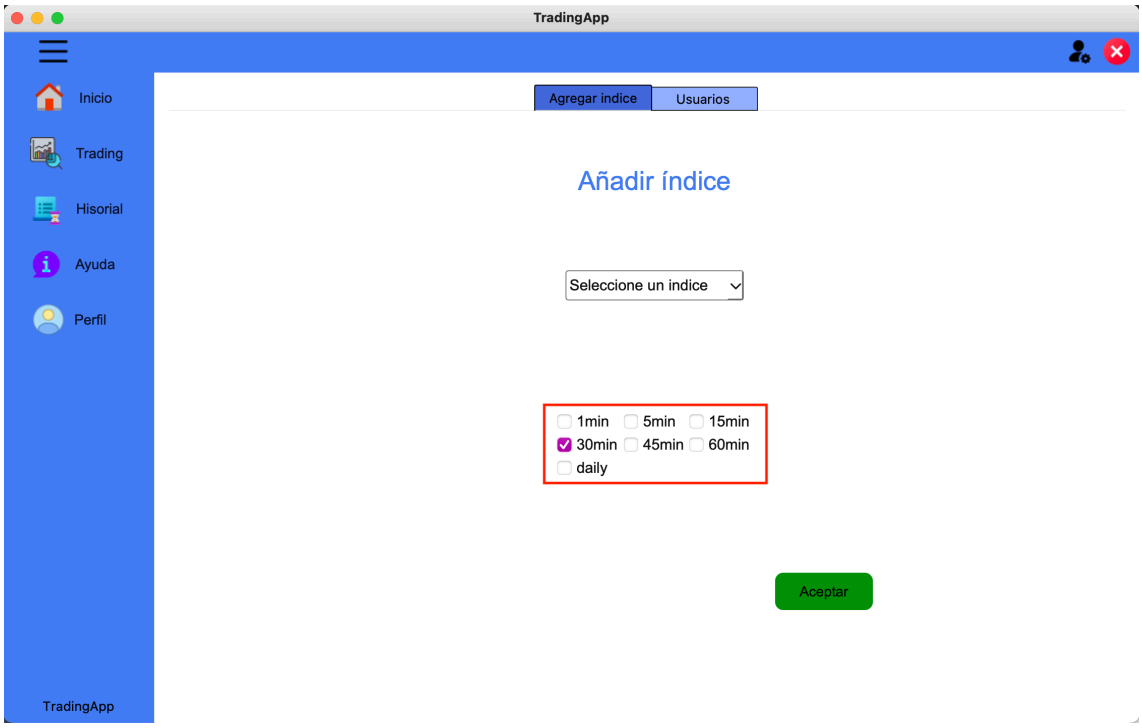


Figura A.3: Pantalla añadir activo timeframe

A.2. Área de trading

En esta pantalla los usuarios eligen las opciones para poder obtener los resultados. Primero elige un activo en concreto de la lista de disponibles A.4. Posteriormente, elige la fecha del análisis que se realizará mediante un calendario desplegable A.5. La tercera opción es elegir la temporalidad mediante un menú desplegable A.6. Finalmente se elige el monto total de las operaciones que se harán, como se puede ver en la figura A.7. Para proceder a obtener los resultados se pulsa el botón aceptar.

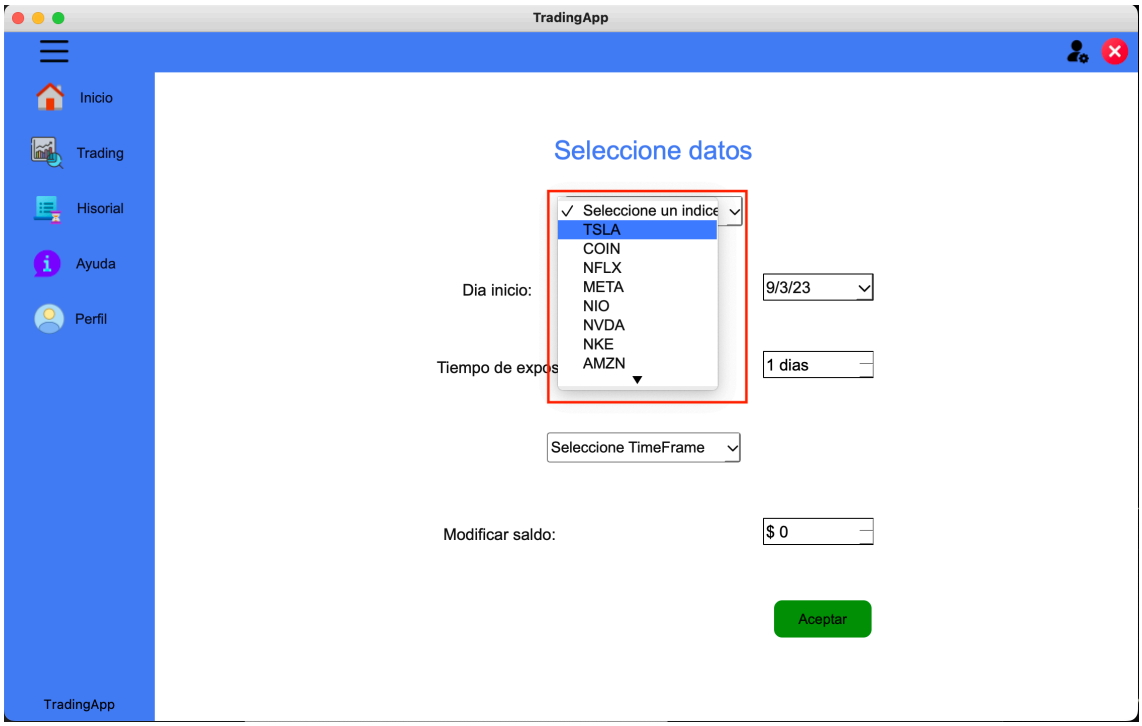


Figura A.4: Pantalla trading selección de activo

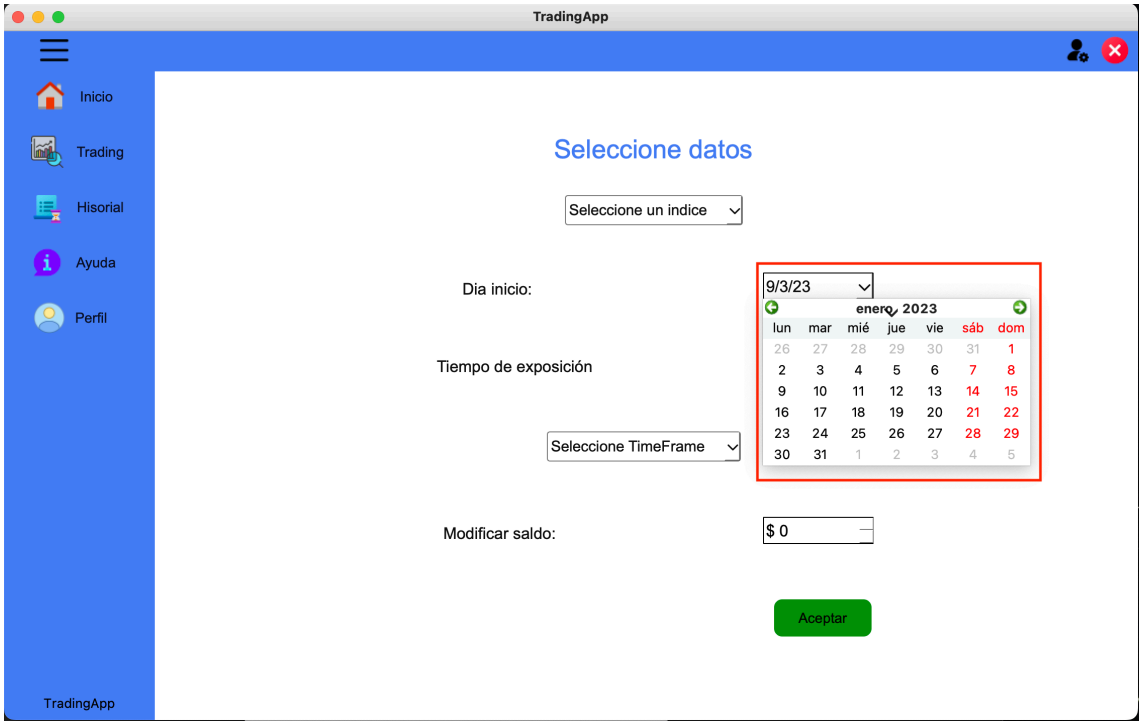


Figura A.5: Pantalla trading elegir fecha

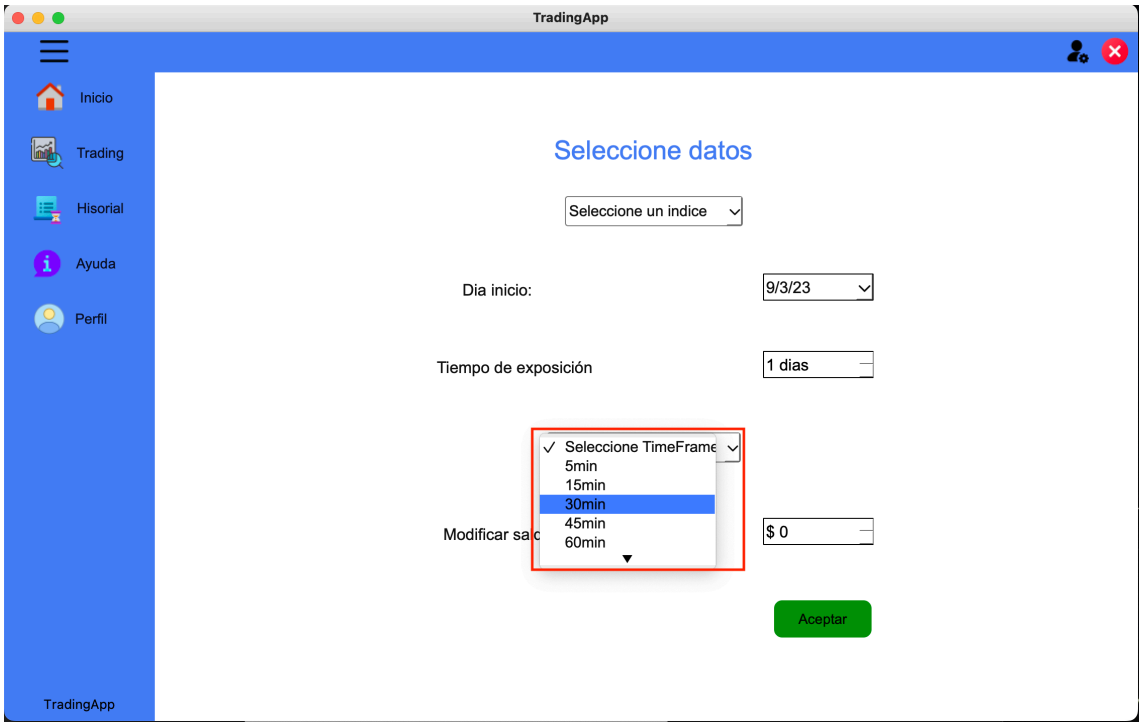


Figura A.6: Pantalla trading seleccionar temporalidad

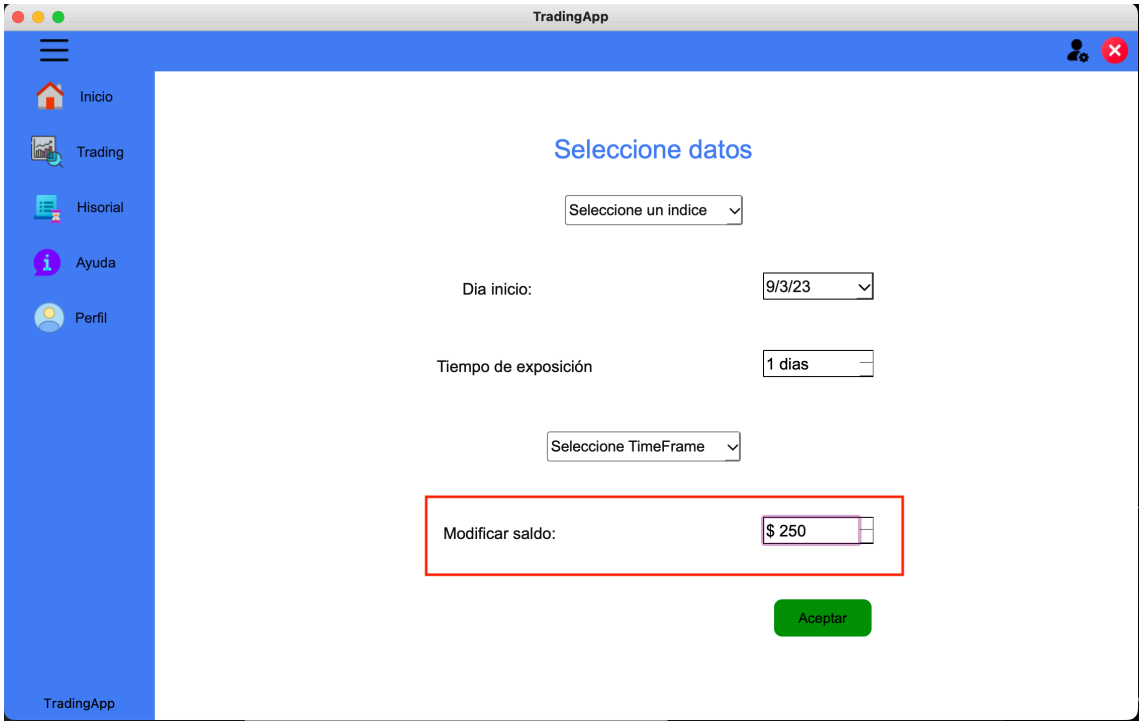


Figura A.7: Pantalla trading seleccionar cantidad

A.3. Resultados

Los usuarios obtendrán los resultados del análisis en forma de tabla como se muestra en la figura A.8. Las cuatro primeras columnas son las compras que se deben de realizar. El monto de la operación se divide entre 4, en función de que si es necesario se hace cuatro compras de esta forma permite bajar la media en caso de que vaya en dirección contraria el precio además de que la exposición es menor. La quinta columna es el precio de venta. La columna siguiente es el precio medio de las compras y la ultima columna la fecha en la que se ha realizado.

	Compra 1	Compra 2	Compra 3	Compra 4	Precio venta	Precio promedio	Saldo final	Fecha
1	111.63	108.4724	106.7071	105.53	109.706	108.085	14.997	03-01-2023
2	108.41	0	0	0	110.036	108.41	3.75	05-01-2023
3	104.3	102.16	0	0	104.778	103.23	7.498	06-01-2023
4	117.06	115.75	0	0	118.151	116.405	7.5	13-01-2023
5	125.44	0	0	0	127.322	125.44	3.751	19-01-2023
6	166.7	0	0	0	169.2	166.7	3.749	30-01-2023
7	164.2	0	0	0	166.663	164.2	3.75	31-01-2023

Figura A.8: Pantalla resultados

Marlon Campoverde Méndez

2023

Last Update: 15 de septiembre de 2023

LaTeX lic. LPPL & powered by TeFLoN CC-ZERO

Esta obra está bajo una licencia Creative Commons «CC0 1.0 Universal».