UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Backtesting de algoritmos de inversión en el mercado valores

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Vicente Císcar, Adrià

Tutor/a: Busquets Mataix, José Vicente

CURSO ACADÉMICO: 2023/2024

Resumen

El mundo del *trading* es muy amplio y apasionante. Una forma de realizar *trading* es utilizando estrategias, que deben ser probadas y evaluadas antes de utilizarse en el mundo real. En este punto cobran sentido las herramientas de *backtesting*, ya que sirven para realizar esta misma tarea sin arriesgar capital.

Estas herramientas, sin embargo, no están al alcance de todas las personas ya que no suelen ser gratuitas, por lo que existe un hueco en la situación actual de las tecnologías de *backtesting*. El objetivo de este trabajo es el de crear una solución software de código abierto que permita a los usuarios realizar *backtesting* sobre el mercado de valores y que aporte valor a la comunidad.

A lo largo de este trabajo se estudia la situación de las tecnologías de *backtesting* y se descubre que todas las plataformas que permiten realizar *backtesting* sin tener que programar un sistema de *backtesting*, ya sea desde cero o utilizando una librería que facilite su creación, son de pago y de código cerrado, por lo que se propone crear una herramienta de código abierto para realizar *backtesting.*

Después de un análisis de requisitos y de otros aspectos sobre el proyecto, se decide que esta solución será una aplicación de escritorio multiplataforma denominada Strategist, y se desarrolla un plan de trabajo para su creación. Además, se generan actividades y un diseño detallado de cómo será la aplicación antes de su desarrollo.

Por último, se desarrolla la aplicación, consiguiendo unos resultados aceptables a nivel de los requisitos planteados. Se prueba la aplicación y se propone un plan de implantación.

En conclusión, se consigue crear una solución de software de código abierto que permite a los usuarios realizar *backtesting* sobre el mercado de valores y que aporte valor a la comunidad. Esta aplicación, al ser de código abierto, tiene la ventaja de ser extensible por la comunidad, por lo que también se plantean una serie de trabajos futuros aplicables al proyecto y a la aplicación.

Palabras clave: *backtesting*; Python; código abierto; mercado de valores; *trading*.

Resum

El món del *trading* és molt ampli i apassionant. Una forma de fer *trading* és utilitzant estratègies, que han de ser provades i avaluades abans d'utilitzar-se al món real. En aquest punt cobren sentit les eines de *backtesting*, ja que serveixen per fer aquesta mateixa tasca sense arriscar capital.

No obstant això, aquestes eines no estan a l'abast de totes les persones ja que no solen ser gratuïtes, per la qual cosa hi ha un buit en la situació actual de les tecnologies de *backtesting*. L'objectiu d'aquest treball és crear una solució software de codi obert que permeta als usuaris realitzar *backtesting* sobre el mercat de valors i que aporte valor a la comunitat**.**

Al llarg d'aquest treball s'estudia la situació de les tecnologies de backtesting i es descobreix que totes les plataformes que permeten fer *backtesting* sense haver de programar un sistema de *backtesting*, ja siga des de zero o utilitzant una llibreria que facilite la creació, són de pagament i de codi tancat, per la qual cosa es proposa crear una eina de codi obert per fer *back**t**esting*.

Després d'una anàlisi de requisits i altres aspectes sobre el projecte, es decideix que aquesta solució serà una aplicació d'escriptori multiplataforma anomenada Strategist, i es desenvolupa un pla de treball per a la seua creació. A més, es generen activitats i un disseny detallat de com serà l'aplicació abans del desenvolupament**.**

Finalment, es desenvolupa l'aplicació, aconseguint uns resultats acceptables respecte als requisits plantejats. Es prova l'aplicació i es proposa un pla d'implantació.

En conclusió, s'aconsegueix crear una solució de programari de codi obert que permet als usuaris fer *backtesting* sobre el mercat de valors i que aporte valor a la comunitat. Aquesta aplicació, com que és de codi obert, té l'avantatge de ser extensible per la comunitat, per la qual cosa també es plantegen una sèrie de treballs futurs aplicables al projecte i a l'aplicació.

Paraules clau: *backtesting*; Python; codi obert; mercat de valors; *trading*

Abstract

The world of trading is very ample and exciting. One way of trading is to use strategies, which must be tested and evaluated before being used in the real world. This is where *backtesting* tools make sense, as they are used to perform this very task without risking capital.

These tools, however, are not available to everyone as they are usually not free, so there is a gap in the current state of the art of *backtesting* technologies. The objective of this work is to create an open-source software solution that allows users to perform *backtesting* on the stock market and that brings value to the community.

Throughout this work the situation of *backtesting* technologies is studied and it is discovered that all platforms that allow *backtesting* without having to program a *backtesting* system, either from scratch or using a library that facilitates its creation, are paid and closed source, so it is proposed to create an open-source tool for *backtesting*.

After an analysis of requirements and other aspects of the project, it is decided that this solution will be a multiplatform desktop application called Strategist, and a work plan for its creation is developed. In addition, activities and a detailed design of how the application will look like are generated before its development.

Finally, the application is developed, achieving acceptable results with regards to the requirements. The application is tested, and an implementation plan is proposed.

In conclusion, we managed to create an open-source software solution that allows users to perform *backtesting* on the stock market and that provides value to the community. This application, being open source, has the advantage of being extensible by the community, so a series of future works applicable to the project and the application are also proposed.

Keywords: *backtesting;* Python; open-source code; stock market; trading

Tabla de Contenidos

Índice de figuras

Índice de tablas

Índice de algoritmos

Glosario

Definiciones

- **API**: *Application Programming Interface*, o «Interfaz de programación de aplicaciones», hace referencia a una pieza de código que permite a diferentes aplicaciones comunicarse entre sí para compartir información y funcionalidades.
- ***Backtesting*:** Proceso de aplicar una estrategia a datos históricos para determinar su rendimiento.
- **Caché:** Almacenamiento temporal de datos que permite acelerar el acceso a información ya consultada previamente.
- **Código abierto:** Referencia a software cuyo código fuente está disponible para ser estudiado, modificado y distribuido por cualquier persona.
- **Criptomonedas:** Monedas digitales o virtuales que utilizan criptografía para seguridad y operan independientemente de un banco central.
- ***Entrypoint*:** Punto de entrada de una aplicación o sistema, donde comienza la ejecución del código o proceso.
- ***Exchange*:** Plataforma digital que facilita la compra, venta e intercambio de activos financieros, como criptomonedas y valores.
- ***Extreme Programming* (XP):** Metodología de gestión de proyectos informáticos planteada por Kent Beck.
- ***FinTech*:** Del término en inglés *financial technology*, o «tecnología financiera» en castellano, que hace referencia a todo el repertorio tecnológico referente a las finanzas.
- ***Framework*:** Conjunto organizado de librerías y convenciones que proporcionan una estructura base para el desarrollo de aplicaciones, facilitando tareas comunes y promoviendo buenas prácticas de codificación.
- **IDE:** *Integrated Development Environment*, o «Entorno de Desarrollo Integrado», hace referencia a un software que proporciona distintas utilidades para la programación o desarrollo de software.
- **IEEE:** *Institute of Electrical and Electronics Engineers*, o «Instituto de Ingenieros Eléctricos y Electrónicos» en castellano.
- **Multiplataforma:** Capacidad de un software de funcionar en más de un sistema operativo o plataforma de hardware.
- **ODS:** Objetivos de Desarrollo Sostenible.
- ***Script*:** Archivo de texto con una serie de comandos o instrucciones que son ejecutadas por un programa específico.

- ***Storyboard*:** Herramienta de planificación visual que utiliza una secuencia de dibujos o imágenes para representar las escenas de un proyecto.
- ***Suite* de herramientas:** Conjunto de programas informáticos relacionados entre sí, diseñados para facilitar la realización de una tarea específica.
- **TDD:** *Test-Driven Development*, o en español, «Desarrollo guiado por pruebas», metodología de desarrollo de software que enfatiza la creación de pruebas automáticas antes del desarrollo del código.
- ***Trade*:** Acción de comprar o vender activos en mercados financieros.
- ***Trader*:** Persona que realiza *trading*.
- ***Trading*:** Acción de compraventa de instrumentos financieros con el objetivo de obtener un beneficio.
- ***Trading* algorítmico:** Tipo de *trading* que se basa en la aplicación de reglas, procesos y/o algoritmos para la compraventa.
- ***Trading* automático:** Tipo de *trading* algorítmico que se realiza automáticamente, sin la necesidad de la intervención del usuario.
- ***Web scraping*:** proceso de recopilar información de forma automática de la *web*.

CAPÍTULO 1
Introducción

El *trading* es la acción de compraventa de instrumentos financieros con el objetivo de obtener un beneficio [1]. Éste se diferencia de las inversiones en que se adquieren instrumentos financieros sin esperar ningún rédito más allá del que se pueda obtener con su venta [2].

Es por esto por lo que el *trading* puede resultar muy complicado, ya que los precios de los instrumentos son muy sensibles a un gran número de variables. Entre otros, los factores políticos, económicos y sociales afectan a los precios de los instrumentos en las bolsas de valores, lo que hace que resulte muy complicado predecir su valor [3].

Sin embargo, desde la creación de la bolsa, se ha intentado explicar los precios de los instrumentos de muchas formas, e incluso hoy en día se intentan predecir con la inteligencia artificial [3]. Una de estas fue la Teoría de Dow, creada por Charles Henry Dow a finales del siglo XIX. Ésta evolucionó y dio lugar al análisis técnico [4].

El análisis técnico es una metodología que consiste en analizar y predecir la dirección de los precios mediante el estudio de los datos pasados del mercado, principalmente el precio y el volumen, aunque se puede estudiar distintos indicadores [5]. Para muchos *traders*, el análisis técnico es una herramienta clave en la elección de estrategias de inversión y es una de las formas principales de analizar los mercados.

Una de las distintas formas en las que se puede aplicar el análisis técnico, es la creación de una estrategia de compraventa basada en distintos indicadores. Un ejemplo de estrategia para las acciones de la empresa X sería: comprar si el precio baja de $10\ ^{€}/_{\text{acción}}$ y vender si el precio alcanza los $15\ ^{€}/_{\text{acción}}$.

Este tipo de *trading* utilizando reglas, proceso o incluso algoritmos, es conocido como *trading* algorítmico [6]. Este tipo de *trading* se realiza comúnmente con el *trading* automático, ya que se requiere de *trading* algorítmico para el *trading* automático. La cantidad y tipología de estrategias utilizadas en el *trading* algorítmico es muy grande y variada, aunque todas ellas tienen riesgos asociadas.

Aunque la estrategia propuesta anteriormente sea muy simple, siempre cabe la posibilidad de que funcione para algún instrumento. Una de las maneras para comprobar si las estrategias creadas son efectivas es el *backtesting*.

El *backtesting* es el proceso de aplicar una estrategia a datos históricos para determinar su rendimiento. Esta estrategia permite a los *traders* probar sus estrategias sin riesgo de capital real [7]. Esta práctica consiste en aplicar reglas y parámetros predeterminados a los datos de precios anteriores para simular el rendimiento que hubiese tenido. Es un paso crucial para el desarrollo y la validación de estrategias de inversión, ya que proporciona un medio para evaluar el rendimiento hipotético sin riesgo financiero [8].

Hoy en día existen muchas herramientas que permiten realizar *backtesting*, pero muchas de ellas tienen interés económico detrás o requieren de conocimiento de programación por parte del usuario. Es por ello por lo que en este trabajo fin de grado se quiere desarrollar una solución de código abierto que permita a todos los usuarios realizar *backtesting* sobre mercados de valores.

1.1 Motivación

El mundo del *trading* es muy amplio y apasionante, y la comunidad que se ha generado en torno al mismo suele ser muy acogedora. Existe una gran cantidad de foros[1] que ayudan a los principiantes a iniciarse en este mundo y empezar a utilizar herramientas de *trading*.

Sin embargo, a medida que se avanza en dificultad y profundidad de contenidos, los foros dejan de servir como guía estructurada, y son las academias, cursos o herramientas de pago las que toman la delantera y proporcionan una utilidad real para los usuarios.

Por otra parte, la mayoría de las herramientas de *backtesting* requieren de conocimientos de programación, y por tanto existe un gran grupo de *traders* que no tienen acceso a este tipo de herramientas.

Como ya se ha mencionado, el *backtesting* es una práctica muy beneficiosa a la hora de realizar *trading*, ya que permite a los usuarios probar si las estrategias planteadas son útiles en ciertos mercados, y, por tanto, validar de antemano los métodos de compraventa.

Es por ello por lo que, el hecho de crear una herramienta de *backtesting* al alcance de todos los usuarios y hecha por y para la comunidad, puede ser muy positivo para el mundo del *trading*. Con ella, muchos *traders* sin conocimiento de programación, pueden acercarse al mundo del

[1] Un ejemplo es el subforo de «r/Trading» en Reddit: <`https://www.reddit.com/r/Trading/`>.

backtesting y explorar esta rama, anteriormente de difícil aproximación por sus requisitos de acceso, ya sean monetarios o de conocimientos.

La motivación principal de este proyecto es la de democratizar una parte de las herramientas *FinTech*, fomentando la innovación y la inclusividad en los mercados financieros.

1.2 Objetivos

El objetivo general del trabajo es crear una solución software de código abierto que permita a los usuarios realizar *backtesting* sobre el mercado de valores y que aporte valor a la comunidad.

Para alcanzar este objetivo general se proponen los siguientes objetivos específicos:

- Estudiar las diferentes soluciones actuales y los posibles problemas en ellas
- Analizar aspectos legales y éticos de un proyecto enfocado al *backtesting*
- Aplicar una metodología que ayude a producir software de calidad
- Utilizar y generar solo software de código abierto
- Generar una solución sencilla de cara al usuario

1.3 Impacto esperado

Al eliminar la barrera de los conocimientos de programación, un abanico más amplio de personas, incluidos operadores no técnicos, analistas y aficionados, pueden dedicarse al *backtesting*. Esta democratización permite a más personas probar estrategias de *trading*, lo que puede conducir a una mayor innovación y diversidad en los enfoques del *trading*.

Por otra parte, el software de código abierto suele ir acompañado de una comunidad y documentación. Los usuarios pueden aprender no sólo a utilizar la aplicación, sino también las mejores prácticas en *backtesting* y estrategias de *trading*.

De hecho, con más usuarios capaces de hacer *backtesting* fácilmente, podría haber un aumento en el desarrollo y perfeccionamiento de las estrategias de *trading*. Esto podría dar lugar a estrategias más sólidas y diversas disponibles en el mercado.

Por otro lado, como proyecto de código abierto, se busca que la comunidad de programadores *traders* cree una subcomunidad en torno a la aplicación. Ésta puede contribuir a mejorarla y compartir estrategias, aumentando así el valor global de la herramienta. Además, ello permite que la aplicación sea ampliada de sus fronteras iniciales, permitiendo incluso la creación de herramientas de IA en la aplicación o el trading automático.

En cuanto al impacto sobre la sociedad en general, se puede utilizar los Objetivos de Desarrollo Sostenible, ODS[2] en adelante, como marcadores del impacto del proyecto. De hecho, el proyecto tiene cierta relación con algunos ODS, entre otros el «ODS 17: Alianzas para lograr los Objetivos» y el «ODS 9: Industria, innovación e infraestructura», ya que la aplicación es de código abierto, lo que fomenta la colaboración global y la mejora constante de la herramienta, es decir, la innovación. Además, también mantiene relación con el «ODS 4: Educación de calidad», ya que permite compartir ideas y conocimientos con colaboradores y la aplicación puede servir como una herramienta de aprendizaje sobre estrategias a probar.

En general, esta aplicación pretende ser de gran utilidad para un amplio sector de la comunidad de *trading*, democratizando el análisis financiero y promoviendo la innovación y el desarrollo dentro de la misma.

1.4 Metodología

Al ser este un proyecto realizado por una sola persona, las metodologías ágiles no parecen las más adecuadas, debido a que una de las bases de estas metodologías es la de permitir una comunicación fluida entre personas del mismo equipo y entre equipos del mismo proyecto [9]. Sin embargo, estas mismas metodologías pueden adaptarse para proyectos de una sola persona, eliminando todos los procesos enfocados a la comunicación y manteniendo todas las demás ventajas que ofrecen este tipo de metodologías.

De entre todas las metodologías en este proyecto se utiliza la metodología de *Extreme Programming*, XP en adelante. Según su creador, Kent Beck [10], XP es una metodología ágil de desarrollo de software que hace hincapié en la satisfacción del cliente, el trabajo en equipo y la mejora continua. XP da prioridad a la entrega de software eficiente y de alta calidad mediante el fomento de la comunicación, la colaboración y la adaptabilidad.

Para adaptarla a la situación particular del proyecto, se ha decidido sustituir las partes de comunicación y colaboración por la organización y revisión propia. Sin embargo, se han podido utilizar algunos de los puntos más prácticos de XP, como el desarrollo iterativo, el desarrollo manejado por pruebas o las pruebas continuas.

El uso de XP en proyectos en solitario puede ser beneficioso, ya que fomenta un enfoque disciplinado para el desarrollo de software, garantizando que el producto sea adaptable a los

[2] Información disponible en:
<https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible>

requisitos cambiantes. El enfoque en la simplicidad y la mejora continua de XP puede ayudar a gestionar la complejidad y mantener la calidad de su trabajo.

1.5 Estructura de la memoria

El contenido de este trabajo se organiza en capítulos, siendo los capítulos del 1 al 7 los que comprenden el contenido principal de este.

En el Capítulo 1 se introduce el trabajo, presentando la motivación y objetivos de este, así como explicando la metodología usada a lo largo del mismo.

En el Capítulo 2 se explora el contexto actual del *backtesting*, estudiando su estado del arte para después exponer una crítica al mismo y realizar una propuesta que consiga mejorarlo.

En el Capítulo 3 se analiza el problema que supone aplicar la propuesta de mejora del estado del arte, presentando los requisitos de esta y analizando otros aspectos como la legalidad y la ética que la conciernen. Además, se plantea una solución firme a este problema, junto con un plan de trabajo y un presupuesto.

En el Capítulo 4, se realiza el diseño de la solución propuesta anteriormente, presentando sus actividades, su arquitectura y las tecnologías que se pretende usar.

En el Capítulo 5 se detalla el proceso ý desarrollo de la solución, desde el entorno donde se realiza el desarrollo hasta la implantación y despliegue de este. Además, se muestran los resultados del desarrollo y se explica la metodología de pruebas utilizada.

En el Capítulo 6 se explicarán las conclusiones extraídas del trabajo respecto a los objetivos marcados (véase 1.2).

En el Capítulo 7 se listan una serie de mejoras aplicables a la solución que quedan fuera del alcance del TFG, pero que pueden aportar valor a la solución.

1.6 Convenciones

Se listan las convenciones seguidas a lo largo del trabajo para facilitar su lectura:

- El código fuente se representará utilizando la letra `Consolas` y solo se empleará para representar código.
- Las palabras extranjeras y extranjerismos derivados se marcarán utilizando la *cursiva*. Todas estas palabras, junto a otros tecnicismos, acrónimos y siglas aparecerán definidas en el glosario.

- También se pondrán en *cursiva* las expresiones en latín.
- Los nombres propios provenientes del extranjero no se pondrán en cursiva, aunque sean palabras extranjeras y empezarán siempre por mayúscula. Esto incluye también a nombres de revistas, programas informáticos, librerías, etc.
- Se entrecomillará según sea necesario durante el documento utilizando las comillas españolas («»). El uso de las otras comillas (“” y ‘’) quedará reservado para el entrecomillado anidado.
- Se anotarán a pie de página las páginas con información de las herramientas mencionadas a lo largo del documento. En ningún caso se tratará de citas a bibliografía.
- Los hipervínculos o URL se representarán con el siguiente formato: <`dirección`>
- Las citas y la bibliografía se presentan utilizando el formato del IEEE[3].

[3] Estándar disponible en <`https://ieeeauthorcenter.ieee.org/wp-content/uploads/IEEE-Reference-Guide.pdf`>

CAPÍTULO 2
Contexto y situación actual del *backtesting*

El *backtesting* es una herramienta que permite analizar los resultados de una estrategia utilizando datos históricos. El proceso de realización de *backtesting* sigue los siguientes pasos generales [11, 12]:

1. Diseño de la estrategia
2. Obtener los datos históricos del mercado
3. Limpieza y procesado de datos
4. Fase de *backtesting*
5. Ajuste y validación de la estrategia

De hecho, todas las herramientas disponibles en el contexto tecnológico actual siguen esta estructura general. Cabe destacar, además, que una estrategia tiene siempre dos acciones principales: comprar y vender.

Partiendo de este esquema básico surgen todas las herramientas presentes en el estado del arte actual, y existe una gran variedad tanto de aplicaciones como de estrategias para utilizar.

2.1 Estado del arte del *backtesting*

Los usuarios tienen a su alcance una gran variedad de opciones a la hora de realizar *backtesting*. Sin embargo, se podrían englobar en tres grandes grupos: la programación completa del sistema, el uso de *frameworks* de programación ya prestablecidos, y, por último, el uso de aplicaciones completas creadas por terceros.

2.1.1 Creación del sistema íntegramente

La primera opción es la realización del *backtesting* de forma manual, utilizando un lenguaje de programación y diseñando todos los aspectos del proceso. Para realizarlo se puede utilizar cualquier lenguaje de programación, pero requiere de un gran conocimiento en el proceso de

backtesting, así como de programación para poder plasmarlo en la aplicación. Un lenguaje de programación adecuado para este propósito sería C++[4].

C++ es un lenguaje de programación compilado que prioriza la eficiencia, el rendimiento y la portabilidad, siendo una buena elección para manejar grandes conjuntos de datos, como se necesita en el *backtesting* [12]. Utilizando esta opción la programación tanto del entorno de trabajo como de las estrategias quedaría a cargo del usuario. Una posible estructura del código para la creación de una estrategia se puede encontrar en el Anexo B.1.

Por otra parte, aun no siendo un lenguaje de programación *per se*, también deberíamos mencionar las herramientas de hojas de cálculo, como Microsoft Excel[5], o Excel, de aquí en adelante. Esto es debido a que junto con el uso de Power Query[6] para la recogida de datos de distintas fuentes [13], se pueden crear sistemas completos de *backtesting* desde cero. En el Anexo B.2 podemos encontrar una captura de pantalla donde vemos un panel creado con Excel capaz de realizar *backtesting* sobre un activo del mercado de valores.

2.1.2 *Frameworks* de lenguajes de programación

Siguiendo con la programación, una opción mucho más cómoda que programar tu propio sistema de *backtesting* desde cero, es utilizar soluciones ya creadas que permitan realizar el análisis histórico de manera mucho más cómoda y eficiente.

Estas librearías están creadas y mantenidas normalmente por la comunidad de programadores que tengan interés en el *trading*, dejando así librerías con interfaces muy centradas en facilitar su uso a la hora de realizar *backtesting*. Afortunadamente, existen una gran variedad de opciones de este tipo, gracias a las oportunidades que brinda el código abierto.

Normalmente se tendrá que hacer uso de dos librerías, una para la fase de obtención de datos, y otra para la realización de la fase de *backtesting*.

Existen infinidad de lenguajes de programación, pero este trabajo se centrará en el uso de Python[7], ya que es un lenguaje multiparadigma y extremadamente versátil, que presenta una gran facilidad a la hora de escribir y escalar código [14]. Además, Python tiene una comunidad muy grande, con un ecosistema muy amplio librerías de código abierto de fácil integración entre sí, gracias al repositorio de librerías del lenguaje, llamado PyPI[8].

[4] Información disponible en: <https://isocpp.org/>
[5] Información disponible en: <https://www.microsoft.com/es-es/microsoft-365/excel>
[6] Información disponible en: <https://learn.microsoft.com/en-us/power-query/power-query-what-is-power-query>
[7] Información disponible en: <https://www.python.org/>
[8] Información disponible en: <https://pypi.org/>

Recogida de datos

Existen dos formas principales para la recogida de datos financieros: el *web scraping* y el uso de una API. La primera consigue extraer datos de páginas *web* de manera automática y la segunda utiliza un servicio propietario de algún banco de datos para obtener los datos de manera estructurada.

Dentro del ecosistema de Python existen una gran variedad de librerías que permiten realizar la recogida de datos utilizando las dos técnicas. Sin embargo, el *web scraping*, aunque es útil en muchos aspectos, tiene mucha incertidumbre respecto a la forma de la *web* de la cuál obtiene la información y es muy susceptible a los cambios de esta, que pueden ser periódicos y no notificados [15]. Por tanto, esta opción no se explora más en este trabajo. La opción de obtener datos de una API es mucho más estable, por lo que se explorarán librerías que nos permitan realizar este proceso de obtención de datos.

En primer lugar, YFinance[9], es una librería que obtiene datos de la API que pone a disposición del público la página *web* de Yahoo Finance[10], una de las páginas de información financiera más utilizadas dentro de la comunidad del *trading*. Esta página tiene siempre datos actualizados y los presenta de manera muy ordenada. Su principal problema es que la librería de Python no está mantenida ni reconocida por Yahoo Finance, sino que está hecha por la comunidad, aunque realmente no impide su uso.

Por otra parte, tenemos Alpha Vantage[11], que es una librería que facilita la conexión con la API homónima y que requiere de una llave gratuita tras identificarse. Además, ofrece una gran cantidad de empresas.

Siguiendo la misma línea que la anterior opción, tenemos la librería de Barchart[12], que también facilita la conexión con la API homónima, pero solo contiene datos históricos de los 2 años anteriores.

Por último, tenemos la librería Pandas Datareader[13], que permite leer y acceder a datos de diferentes fuentes financiera, así como implementar un sistema de caché de peticiones. Se trata una solución de más bajo nivel que las anteriores que permite controlar las peticiones de manera más personalizada.

[9] Información disponible en: <https://github.com/ranaroussi/yfinance>
[10] Información disponible en: <https://es.finance.yahoo.com/>
[11] Información disponible en: <https://pypi.org/project/alpha-vantage/>
[12] Información disponible en: <https://github.com/femtotrader/barchart-ondemand-client-python>
[13] Información disponible en: <https://pandas-datareader.readthedocs.io/en/latest/>

Realización del backtesting

Para la realización de *backtesting* existen distintas librerías con varios motores implementados, capaces de realizar estrategias y proporcionar métricas de evaluación para éstas.

En primer lugar, tenemos la librería Backtesting.py[14], la cuál es ampliamente utilizada. Esta librería es de fácil uso y permite integrarse con facilidad con otras herramientas del mundo financiero. Además, proporciona grandes utilidades para la creación de software y test. Por otra parte, no tiene indicadores técnicos para las estrategias, pero lo compensa con la gran facilidad de integración a otras librerías de estos indicadores. También tiene la ventaja que permite visualizar interactivamente las estrategias. Para acabar, tiene una excelente documentación.

Por otra parte, existe la librería Backtrader[15], cuyo lema principal es proporcionar gran capacidad de uso. De hecho, tiene integrada la recolección de datos en la misma librería y mantiene una filosofía de creación de estrategias basada en la programación orientada a objetos. Una de las principales ventajas de Backtrader es que cuenta con la mayor comunidad de usuarios y colaboradores, los cuáles ayudan a mantener y mejorar los proyectos. No obstante, la última actividad en su repositorio[16] fue en abril de 2023, lo cual indica que puede haber cierta inactividad en el mantenimiento.

Por último, tenemos la librería de Basana[17], que permite realizar *trading* algorítmico desde Python, y ofrece una funcionalidad de *backtesting*. De hecho, el foco de la librería es el de permitir a los usuarios realizar *trading* en tiempo real, aunque mayoritariamente para mercados de criptomonedas. Además, la librería se integra bien con otras librerías del mundo del *trading*. Sin embargo, al permitir la realización de *trading* en vivo, la implementación de evaluación de estrategias mediante *backtesting*, resulta más complicada que las otras opciones vistas.

2.1.3 Herramientas de *backtesting*

La última opción es aquella más rápida y fácil para los usuarios, que es utilizar un programa completo de realización de *backtesting*. Con las herramientas de *backtesting* el usuario tiene mucha más comodidad de uso, ya que solamente se centra idear y probar la estrategia, y no tiene que integrar librerías o diseñar el sistema desde cero.

Esta solución es la utilizada por la mayoría de las personas con interés de realizar *backtesting*, ya que es la que tiene la barrera de entrada mínima, y permite obtener resultados más bien instantáneos.

---

[14] Información disponible en: <https://kernc.github.io/backtesting.py/doc/backtesting>

[15] Información disponible en: <https://www.backtrader.com/docu/>

[16] Actividad del repositorio: <https://github.com/mementum/backtrader/activity>

[17] Información disponible en: <https://basana.readthedocs.io/en/latest/>

Existen distintos tipos de soluciones dependiendo de la plataforma en la que se encuentren, o de la forma de la implementación de estrategias, aunque al final el resultado tiende a ser el mismo.

En primer lugar, tenemos la herramienta de TrendSpider[18], que es una *suite* de herramientas que permite realizar *backtesting*, generación de ideas en base al mercado y análisis técnico, *trading* algorítmico, entre otros. Es una herramienta muy completa, con inteligencia artificial integrada y herramientas muy intuitiva. Sin embargo, no es de código abierto, y el coste de uso es de $65\ ^{\$}/_{\text{mes}}$.

En segundo lugar, tenemos TradingView[19], que es una página que contiene una gran cantidad de herramientas, así como foros y comunidades enteras dedicadas al *trading*. Es una de las opciones más usadas para muchísimas utilidades necesarias en el *trading* y su versión gratuita ofrece una gran cantidad de herramientas, entre ellas el *backtesting*. El problema es que existe una barrera de entrada para utilizar su herramienta, y es que se requiere conocimientos de programación para realizar las estrategias, y esto limita mucho su uso. No obstante, gracias a los foros y a la comunidad *trader* que tiene esta página *web*, su uso puede ser fácil. También tiene una versión de pago con funcionalidades agregadas. No obstante, pese a tener una versión gratuita, ninguna es de código abierto y no puede ser mejorada por la comunidad.

Por otra parte, tenemos una solución centrada en el *trading* algorítmico, QuantConnect[20]. Esta herramienta cuenta con distintos planes de pago y un plan gratuito, y permite al usuario realizar *trading* en vivo, y evaluar sus estrategias con *backtesting,* aunque con un límite de usos. Sin embargo, se tienen que programar las estrategias que se quieran utilizar, por lo que hay cierta barrera de entrada en su uso.

Por último, tenemos la herramienta FinViz[21], que es otra *suite* de herramientas centrados en la visualización de activos o gestión de porfolios, entre otras. FinViz tiene una herramienta de *backtesting* disponible para los usuarios en su plan de pago «FINVIZ*Elite», que cuesta $25\ ^{\$}/_{\text{mes}}$. Además de la herramienta de *backtesting*, con el plan de pago también obtienes otras ventajas en las demás herramientas.

[18] Información disponible en: <`https://trendspider.com/`>
[19] Información disponible en: <`https://es.tradingview.com/`>
[20] Información disponible en: <`https://www.quantconnect.com/`>
[21] Información disponible en: <`https://finviz.com/`>

2.2 Crítica al estado del arte

El mundo del *backtesting* de estrategias de *trading* en mercados de valores es vasto y muy explorado. Existen distintos tipos de soluciones y una gran comunidad con interés en compartir y ayudarse mutuamente. Sin embargo, hay debilidades dentro del estado del arte que pueden mejorarse. Éstas se pueden observar resumidas en la Tabla 1.

Empezando con las soluciones desde cero, aunque pueden ser útiles casos de uso muy concretos, tienen muy poca escalabilidad y adaptabilidad. Además, son más propensos a tener problemas de seguridad y bugs cuando se salen de su caso de uso concreto. Otro problema muy importante es la dificultad y el coste que conlleva crear el sistema desde cero.

Por otra parte, los *frameworks* disponibles en la comunidad están muy bien mantenidos e implementados, y viene con una gran cantidad de funcionalidades. Sin embargo, todas tienen una debilidad clave, la gran barrera de entrada que supone el hecho de tener que saber programar para poder utilizar estas soluciones. A nivel individual, cada solución puede mejorarse, ya sea implementando nuevos indicadores o arreglando incidencias gracias que son código abierto.

Por último, las herramientas son la opción más accesible y cómoda para toda la comunidad de *trading*. Sin embargo, la mayoría de las soluciones son de pago o requieren de conocimientos de programación, y no hay ninguna herramienta enteramente de código abierto.

	Debilidad clave
Creación de un sistema desde cero	Coste grande y poca escalabilidad
Uso de frameworks	Barrera de entrada con la programación
Uso de herramientas	No existen soluciones de código abierto

Tabla 1: Resumen de los puntos débiles del estado del arte.
Elaboración propia.

2.3 Propuesta

En base al análisis del estado del arte, se propone realizar un proyecto de código abierto que consista en la creación de una herramienta de *backtesting*. Para ello, se utilizará un *framework* que sirva como base del motor de *backtesting*, por lo que se aprovechará al máximo de las bondades del código abierto.

Respecto a la selección de librerías, Basana se descartará para el trabajo, ya que no está centrada en *backtesting* y ofrece funcionalidades que no se utilizan en la aplicación. Luego, quedan Backtesting.py y Backtrader.

Backtrader parece una solución sólida, con una gran comunidad y con muchas funcionalidades. Además, evita el tener que utilizar otra librería para la obtención de datos, o para utilizar otras métricas de análisis técnico, como ya se ha mencionado. Sin embargo, su falta de actividad en el repositorio parece preocupante teniendo en cuenta la posibilidad de que tenga algún tipo de fallo importante.

Por ello, se utilizará Backtesting.py en la solución propuesta. Además de esta librería, para la recogida de datos también hay distintas opciones. En primer lugar, Barchart se descarta debido a su falta de datos históricos y Alpha Vantage también por el inconveniente de tener que crear una cuenta para su uso, lo cual implicaría tener que autenticar a usuarios en la herramienta. Por otra parte, Pandas Datareader puede ser una buena idea si se utiliza una fuente de datos fiable, aunque la herramienta no necesita tanta gestión sobre las peticiones ni ninguna optimización, ya que solamente necesita una petición por estrategia. Con lo que nos quedamos con la herramienta de YFinance, ya que es una opción muy utilizada y con buenas características.

CAPÍTULO 3
Análisis del problema

Se propone realizar una herramienta que permita realizar *backtesting* sobre mercados de valores. Para poder crearla, primero se debe analizar cuáles son los requisitos y el contexto del problema. En la Figura 1 se puede observar de manera satírica la situación que se pretende resolver con esta herramienta. El problema que resolver es el de proporcionar a los *traders* una manera de probar sus estrategias de compraventa antes de realizar sus operaciones reales.

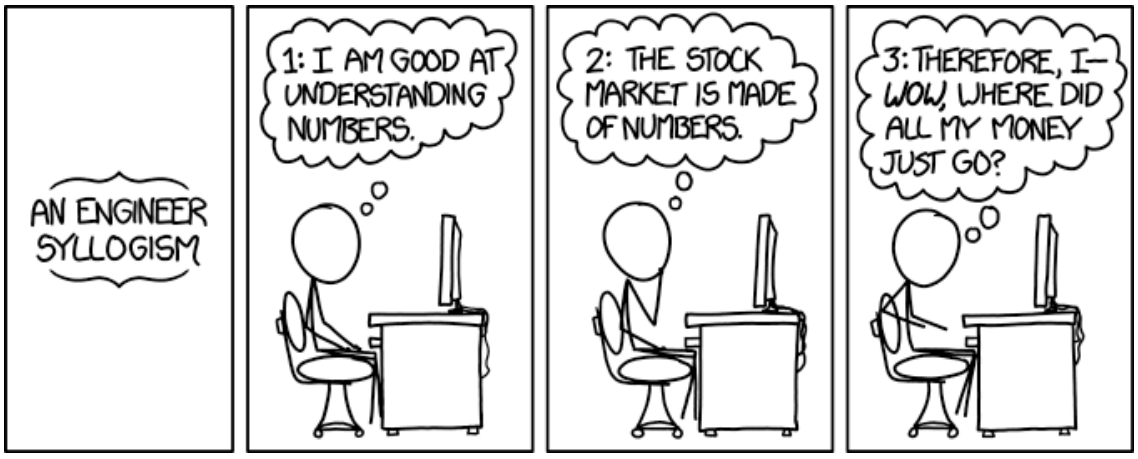


Figura 1: Storyboard *donde se narra de manera satírica una situación típica. Extraído de [16].*

Para poder conseguir una solución óptima se deben tener en cuenta las posibles funcionalidades que puede requerir un usuario, así como otros aspectos que pueden afectar al desarrollo y al despliegue de la aplicación.

3.1 Requisitos

La solución propuesta deberá tener en cuenta el uso que los usuarios harán de ella. Para ello es importante definir unos requisitos que permitan a estos usuarios utilizar la aplicación para los usos que esta pretenda soportar sin ningún tipo de problema.

Para ello, se pueden extraer ciertos requisitos de la historia presentada en la Figura 2, donde un *trader* quiere invertir en bolsa y no sabe cómo. Como solución, se presenta el principal caso de uso de la aplicación, que es el probar estrategias de inversión en el mercado de valores.

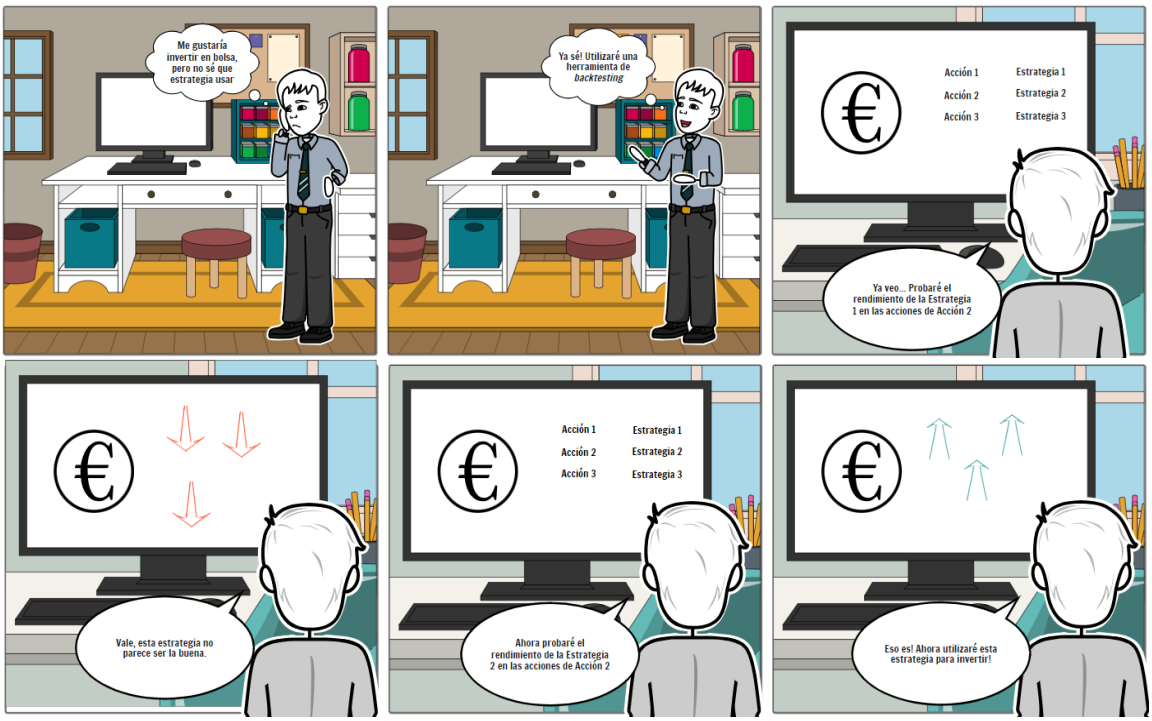


Figura 2: Storyboard *sobre la historia de usuario principal de la solución*
Elaboración propia

En base a esto se analizan los requisitos que va a tener que cumplir la solución.

3.1.1 Requisitos funcionales

La solución debe cumplir los siguientes requisitos funcionales, que son aquellos que describen lo que un sistema debe hacer, es decir, las funciones y características específicas del sistema desde la perspectiva del usuario [17].

- Permite obtener los valores históricos del mercado de valores
- Permite obtener distintas frecuencias de los valores históricos
- Permite configurar distintas estrategias de compraventa
- Permite simular distintas estrategias de compraventa
- Calcula distintas métricas sobre las simulaciones
- Permite visualizar el resultado de la estrategia

3.1.2 Requisitos no funcionales

La solución debe cumplir los siguientes requisitos no funcionales, que son aquellos que se refieren a cómo el sistema realiza sus funciones, abarcando aspectos como la seguridad, la usabilidad, el rendimiento, la compatibilidad, y la fiabilidad [17].

- Obtener resultados de las estrategias en menos de 5 segundos
- Capacidad de conectarse a Internet
- Minimizar las peticiones a Internet guardando los datos históricos temporalmente para repetir estrategias
- La aplicación estará disponible para el usuario en cualquier momento
- La interfaz de usuario será accesible y clara, sin desarrollar animaciones o similares.
- Los datos extraídos de las estrategias no se compartirán a ninguna conexión externa
- Compatibilidad con Linux, Windows y MacOS

3.2 Otros aspectos que analizar

Además de los requisitos ya presentados, resulta interesante analizar algunos puntos adicionales, que pueden resultar un impedimento durante el desarrollo si no se tienen en cuenta.

3.2.1 Análisis de seguridad

Si la solución termina requiriendo de un servidor, es importante tener en cuenta que se implementen conexiones y sesiones de cliente seguras, para evitar posibles ataques a este servidor. Mantener el sistema operativo limpio de servicios inactivos y actualizado y seleccionar cuidadosamente las tecnologías utilizadas para la implementación de la autenticación y cifrado de datos de usuarios son ejemplos de medidas a tomar [18].

Si, en cambio, termina siendo una aplicación de escritorio, no existen riesgos de seguridad a contemplar, ya que las conexiones se realizarán utilizando la librería YFinance y no existirá ningún tipo de servidor propio.

3.2.2 Análisis energético

En cualquier caso, las computaciones pueden resultar muy costosas, pero realmente no tendrá un impacto energético real más allá que cualquier aplicación usual, o cualquier servidor web. Aun así, se debe pretender optimizar los cálculos al máximo.

3.2.3 Análisis del marco legal y ético

Existen distintos niveles que analizar a nivel legal y ético en la solución, ya que se plantean distintos retos referentes a estos temas en la solución.

Análisis de la recogida de datos

La recogida de datos se realiza utilizando la librería YFinance, que se ha creado utilizando la API de Yahoo Finance.

Esta librería tiene un descargo de responsabilidad legal importante en su documentación, que hace referencia a los términos de uso[22] de la API de Yahoo Finance, que prohíben la creación de obras derivadas de la especificación de la API y requieren de atribución a Yahoo Finance por los datos.

Análisis legal del código abierto

El código abierto es susceptible a ser plagiado con facilidad, así como facilidad para que el autor no reciba ningún tipo de reconocimiento por su código. Por ello existen distintos mecanismos para preservar los derechos del autor en el código abierto.

En primer lugar, se debe tener en cuenta la Ley de Propiedad Intelectual [19], que regula los derechos de autor y derechos conexos, así como las excepciones a estos derechos. En el caso del código abierto, este se basa en licencias voluntarias que regulan estos derechos de autor.

Existen distintos tipos de licencias, como la Apache License[23] o la MIT License[24], y se tiene que utilizar la acorde a las necesidades de los proyectos, y sobre todo teniendo en cuenta en las implicaciones que tiene la elección de la licencia en el bien común que se pretende alcanzar realizando el proyecto como código abierto [20].

Análisis ético del código abierto

El código abierto se centra en promover valores como la transparencia, la colaboración y la equidad en el desarrollo y uso de la tecnología. Éste democratiza el acceso a ciertas herramientas y desafía las estructuras de poder tradicionales, promoviendo una mayor inclusión y diversidad. Además, el código abierto impulsa la innovación y la eficiencia al permitir que los desarrolladores construyan sobre el trabajo de otros, evitando así la duplicación innecesaria de esfuerzos. Esta colaboración abierta puede acelerar el desarrollo de soluciones a problemas complejos y contribuir al bien común. Sin embargo, también plantea desafíos éticos relacionados con la seguridad, la privacidad y la sostenibilidad de los proyectos. La dependencia de comunidades voluntarias para el mantenimiento y la seguridad del software puede llevar a vulnerabilidades si no se gestionan adecuadamente. Por tanto, es crucial equilibrar los principios de apertura y

[22] Términos disponibles en: <`https://legal.yahoo.com/us/en/yahoo/terms/product-atos/apiforydn/index.html`>
[23] Licencia disponible en: <`https://www.apache.org/licenses/LICENSE-2.0.txt`>
[24] Licencia disponible en: <`https://www.mit.edu/~amini/LICENSE.md`>

colaboración con la responsabilidad de asegurar que el software sea seguro, fiable y utilizado de manera ética.

3.3 Identificación y análisis de posibles soluciones

A la hora de presentar las soluciones se deben tener en cuenta tanto los requisitos como los otros aspectos analizados del problema. Esto nos lleva a dos soluciones posibles: una aplicación de escritorio y una aplicación web.

3.3.1 Aplicación de escritorio

Esta solución consistiría en realizar una aplicación de escritorio con la capacidad de ser multiplataforma. La aplicación tendría en un mismo equipo todos los componentes necesarios y, excepto por los datos, que son extraídos de Internet, toda la funcionalidad estaría contenida en la misma solución.

Esta solución tiene la ventaja de que los usuarios tendrán un programa instalado en su ordenador, sin tener que crear ningún tipo de cuenta y sin preocupaciones sobre sus datos o sus estrategias.

Por otra parte, puede resultar una desventaja el hecho de tener que utilizar espacio de almacenamiento del ordenador e instalar en el equipo propio un software de terceros para cierto tipo de usuarios.

3.3.2 Aplicación web

Esta solución consistiría en desplegar un servidor con la lógica del programa y una aplicación web que será ejecutada en el navegador de los usuarios, que tendrá la interfaz de usuario de la aplicación. En este caso todas las peticiones para los datos se realizan desde la interfaz de usuario, para no sobrecargar las peticiones a la API desde el servidor, por lo que este solo se encarga del *backtesting* en sí.

Esta solución tiene la ventaja de que no requiere de ningún tipo de instalación, y los usuarios pueden acceder de su navegador.

La desventaja principal viene de que este tipo de aplicaciones requiere de un despliegue de un servidor, lo cual es costoso y requiere de funcionalidades adicionales. Incluso, por seguridad, podría requerir la autenticación de usuarios, lo que implica un paso extra para el uso de la aplicación por parte de los usuarios.

3.4 Solución propuesta

Ambas propuestas resultan interesantes para la creación de una herramienta de *backtesting*, sin embargo, la aplicación resulta más atractiva debido a que resulta más fácil de desarrollar que la aplicación web y tiene menos desventajas.

La solución consistirá, pues, en una aplicación de escritorio multiplataforma que se conectará a Internet para recoger los datos históricos y tendrá la capacidad de realizar *backtesting* sobre mercados de valores. Además, la solución será de código abierto. El nombre pensado para el proyecto será: «Strategist».

3.4.1 Plan de trabajo

Para trabajar en la solución se propone un plan de trabajo iterativo divido en fases. La idea principal es que el proyecto siga en desarrollo por la comunidad, por lo que es importante plantear un plan de trabajo que deje una solución extensible a futuro. El plan es el siguiente:

Fase I. Recogida de información y diseño

a. Se recoge información sobre las herramientas actuales
b. Se escoge un marco de trabajo para el proyecto
c. Se diseña la arquitectura del proyecto

Fase II. Planificación

a. Se analizan las historias de usuarios y las actividades
b. Se marcan objetivos semanales para obtener los requisitos analizados

Fase III. Iteraciones de desarrollo

Cada semana se realiza la siguiente iteración

a. Creación de las pruebas para las nuevas funcionalidades
b. Implementar las funcionalidades para que pasen las pruebas
c. Utilizar el control de versiones para añadir la nueva funcionalidad a la aplicación
d. Revisar los objetivos y actualizar los objetivos de la semana siguiente

Fase IV. Despliegue

a. Desplegar versiones con los objetivos cumplidos y sin fallos
b. Documentar la instalación del paquete

Fase V. Aceptación y repetición

a. Comprobar la aceptación por los usuarios y aprender de sus aportaciones
b. Repetir el plan

Como se puede observar es un plan pensado para un desarrollo continuo de la aplicación, y de índole iterativa. Durante la realización de esta memoria, solamente se alcanzará hasta la Fase IV de la primera iteración, aunque el plan está pensado para que se pueda repetir.

3.4.2 Presupuesto

El desarrollo y despliegue, debido a la naturaleza de la solución propuesta no tiene costes fijos, y solamente tiene un coste variable dependiendo de las horas invertidas. Esto es debido a que la distribución de la aplicación se realizará mediante una herramienta gratuita mencionada más adelante, y al ser de escritorio, no tiene ningún servidor ni servicio externo que requiere de más gastos.

Aunque se podría realizar un estimado de horas invertidas en el proyecto y obtener un coste, esto puede ser conflictivo con los principios del código abierto. Esto es debido a que, en realidad, el proyecto es fruto de una aportación voluntaria a la comunidad, y no se espera ningún rédito o beneficio de este, por lo que, a nivel personal, no tiene sentido hablar de costes de mano de obra.

CAPÍTULO 4
Diseño de la solución: Strategist

La solución Strategist consiste en una aplicación de escritorio multiplataforma que permite obtener datos históricos de mercados de valores y aplicar estrategias de *backtesting* sobre estos.

4.1 Actividades

Para poder definir una arquitectura que permita cumplir con los requisitos de la solución analizada, primero se deben de conocer las actividades que se van a realizar. Para ello, se presentan una serie de historias de usuario, que son una definición de alto nivel de un requisito que contiene la información justa para que el desarrollador pueda estimar el esfuerzo requerido para implementarse [21]. Además, según [22], estas se deben redactar usando la fórmula: «Como (rol), quiero (algo) para (beneficio)». Las historias de usuario de Strategist son:

- Como usuario, quiero poder tener una lista de activos para tener una gran variedad de opciones sobre las que realizar *backtesting*.
- Como usuario, quiero poder tener una lista de estrategias para elegir aquella que se ajuste más al activo o al mercado que deseo estudiar.
- Como usuario, quiero poder configurar la estrategia elegida para ajustar la estrategia a la situación del mercado estudiado y sacar el mayor rendimiento posible.
- Como usuario, quiero poder observar los resultados de la estrategia gráfica y numéricamente para comprender si la estrategia ha tenido éxito o fracaso.

A partir de estas historias de usuario se generan las actividades de la aplicación, disponibles en la Figura 3. A grandes rasgos las actividades consisten en:

- Elegir activo: El usuario elige un activo a su elección y escribe su símbolo
- Comprobar activo: Se comprueba si el activo elegido por el usuario es válido
- Acceder a la lista de estrategias: El usuario entra a la aplicación y observa la lista de estrategias disponibles.
- Realizar *backtesting*: El usuario entra a la aplicación, elige un activo y una estrategia, configura la estrategia y obtiene los resultados del *backtesting* sobre el activo.
- Configurar una estrategia: El usuario entra a la aplicación y después de elegir una estrategia para hacer backtesting, debe configurar sus parámetros.

- Visualizar resultado: El usuario entra a la aplicación y después de realizar *backtesting*, puede decidir visualizar sus resultados en una gráfica.

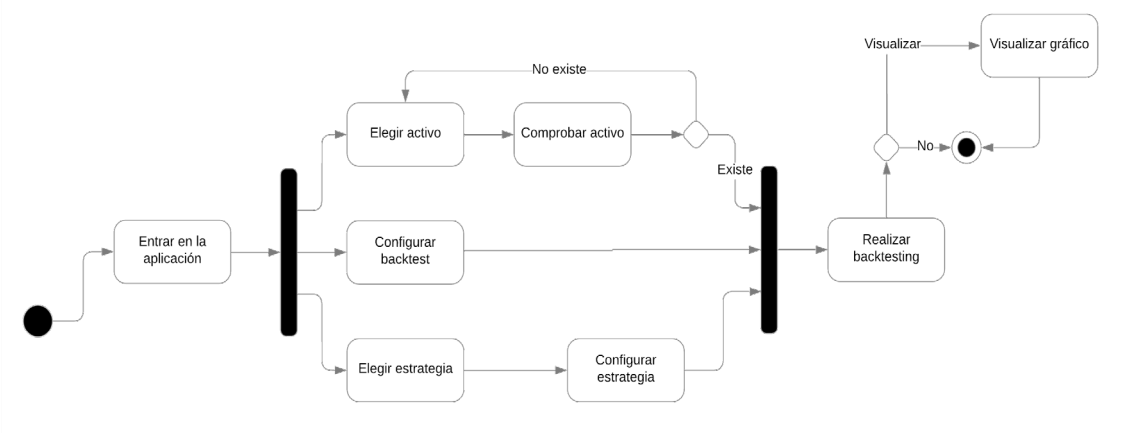


Figura 3: Diagrama de actividades de la aplicación Strategist. Elaboración propia.

4.2 Arquitectura del sistema

Seleccionar una buena arquitectura de software resulta crucial para el desarrollo de una aplicación. Esto es debido a que, gracias a elegir una buena arquitectura de software, la implementación de las soluciones resulta más escalable y extensible [23].

La aplicación, al tratarse de una aplicación de escritorio, vemos que se trata de un sistema personal, ya que está pensado en ejecutarse en un solo ordenador y no tiene elementos distribuidos en su diseño [24]. Sin embargo, es importante mencionar que sí que existe un elemento externo al sistema personal, que es la fuente de datos, aunque no forma parte del desarrollo ni alcance del proyecto, ya que se utiliza mediante una API pública y no está mantenida ni creada por el proyecto. Por eso, aun teniendo este nivel externo, la aplicación es de un nivel de segmentación física, solamente.

Basándonos en la naturaleza de la aplicación, sus requisitos y sus actividades se puede concluir que la arquitectura más ventajosa para este proyecto sería una arquitectura de dos capas, donde se separa la capa de presentación de la de lógica de negocios y datos. Esta división se trata de una división lógica, que se respetará durante el desarrollo, pero no constituirá una división física y estará todo en un mismo ejecutable. En la Figura 4 se puede observar la división por capas de la aplicación.

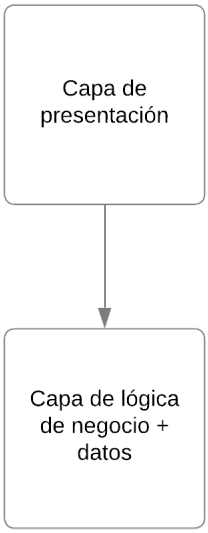


Figura 4: Representación simplificada de la arquitectura de Strategist. Elaboración propia.

La capa de lógica de negocio y datos estará basada en las opciones que nos dan las librerías Backtesting.py y YFinance para realizar *backtesting* y obtener datos históricos. Por otra parte, la capa de presentación, que tendrá comunicación con la capa de lógica y datos, será una interfaz de usuario con tres vistas principales: configuración del *backtest*, resultados y resultados gráficos.

Esta arquitectura es adecuada para el proyecto ya que permite comunicación entre las capas y aun así deja modularidad con tal de, si fuese necesario en un futuro, poder crear distintas capas de presentación para otro tipo de casos de uso.

4.3 Arquitectura detallada

Dentro de la arquitectura bicapa planteada se deben considerar la estructura de cada una de las capas, así como su contenido. Esta arquitectura de capas puede entenderse como dos paquetes principales: el paquete «Motor» y el paquete «IGU». El paquete «Motor» será el encargado de gestionar toda la lógica de negocio y el tratamiento de datos, mientras que el paquete «IGU» será el encargado de la interfaz de usuario, así como su comunicación con el «Motor». Cabe remarcar, que, el motor será agnóstico de la interfaz de usuario. Por otra parte, el paquete «Motor» tendrá incluido otro paquete «Estrategias», que contendrá la implementación de las distintas estrategias utilizadas para el *backtesting*. En la Figura 5 se puede observar el diagrama de paquetes que resume esta información.

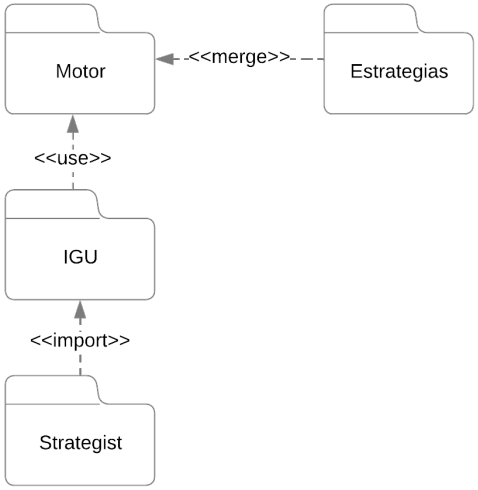


Figura 5: Diagrama de paquetes UML de la aplicación Strategist.
Elaboración propia.

4.3.1 Capa de lógica de negocio y datos

Esta capa se encarga de tanto realizar el *backtesting* como obtener los datos históricos de los activos del mercado de valores. Por ello, dentro de esta capa encontraremos distintas funcionalidades, todas centradas en permitir la realización de *backtesting*. Para resumirlas, se ha creado el diagrama de la Figura 6.

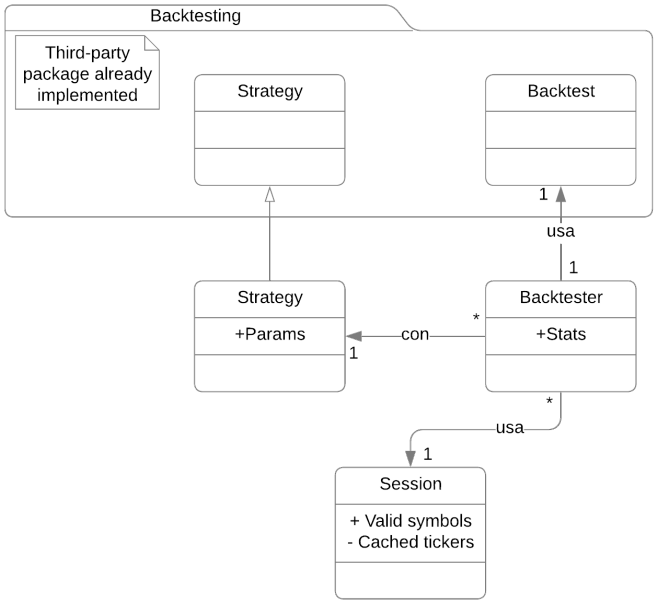


Figura 6: Diagrama de clases de la capa de lógica de negocio y datos de Strategist.
Elaboración propia.

Empezando por la clase «Backtester», esta será la clase principal de la capa lógica, ya que con ella se podrá realizar todas las funciones necesarias para hacer *backtesting*. Esta clase contará con un parámetro público de las estadísticas. Por último, vemos que esta clase usa la clase «Session» con una relación uno a muchos, es decir, que la clase «Backtester» solo usará una instancia de la clase «Session», mientras que esta última podrá usarse en las instancias que sea de la clase «Backester». Por otra parte, la clase «Bakctester» también usa la clase «Backtesting.Backtest» con una relación uno a uno, es decir que una instancia de «Bakctester» estará relacionada siempre con una instancia de la clase «Backtesting.Backtest», y viceversa. Por último, vemos que la clase «Backtester» se relaciona con la clase «Strategy» con una relación uno a muchos, siendo que una instancia de la clase «Bakctester» siempre tendrá asociada una instancia de «Strategy», mientras que una instancia de esta última podrá asociarse a las instancias de «Backtester» que sea.

La clase «Session», por otra parte, se encarga de la lógica de obtención de datos, permitiendo obtener los históricos de algún activo, así como comprobar su validez. Tiene un atributo público con el registro de símbolos válidos hasta el momento, así como un atributo privado con el que se pretende implementar un sistema de caché de históricos.

El paquete «Backtesting» representa que las clases en su interior están extraídas de la librería Backtesting.py, utilizada en el proyecto. Las clases de este paquete no tienen sus atributos representados, ya que no deben implementarse en el proyecto. La clase «Backtesting.Backtest» es la clase que permite realizar *backtesting* de la librería de terceros Backtesting.py. Por otra parte, la clase «Backtesting.Strategy» es la clase abstracta base para la creación de estrategias de *backtesting*, proporcionada por la librería ya mencionada.

Por último, la clase «Strategy» es una especialización de la clase «Backtesting.Strategy», con la que se le añade la capacidad de configuración a la estrategia que representa con el atributo «Params». Todas las estrategias de la aplicación se implementarán usando esta clase.

Cabe remarcar un aspecto importante, y es el hecho de que la translación a código de este diseño no se implementa enteramente con programación orientada a objetos. Es decir, algunas entidades representadas como clases o interfaces no tienen por qué tener una implementación como clase o interfaz en el código, si la tecnología utilizada lo permite y siempre que resulte «buena práctica» no hacerlo.

4.3.2 Capa de presentación

Esta capa es la encargada de manejar la interfaz de usuario, es decir, permitir al usuario realizar todos las actividades de manera directa. Conforme a la arquitectura, esta capa no tendrá ningún tipo de lógica de negocio o similares, y solamente tiene las vistas definidas anteriormente.

Las vistas están muy relacionadas con las actividades de la sección **¡Error! No se encuentra el origen de la referencia.**, y presentan solamente la funcionalidad necesaria, para que quede una interfaz de usuario clara y usable. El diagrama de flujo entre vistas es el presente en la Figura 7.

Cabe destacar, que para el diseño de la interfaz de usuario se han utilizado maquetas, que se irán mostrando más adelante. Las maquetas son herramientas útiles en el desarrollo de interfaces de usuario, ya que permiten centrarse en el diseño de una interfaz que permita cumplir los requisitos y actividades de una aplicación sin tener que preocuparse por la funcionalidad [25]. Las maquetas creadas son de baja fidelidad, ya que no se implementa ningún tipo de funcionalidad o navegación de usuario en las maquetas, e incluso hay alguna falta de contenido real en éstas [26]. La parte positiva es la facilidad de creación de estas, lo cual es muy adecuado para Strategist.

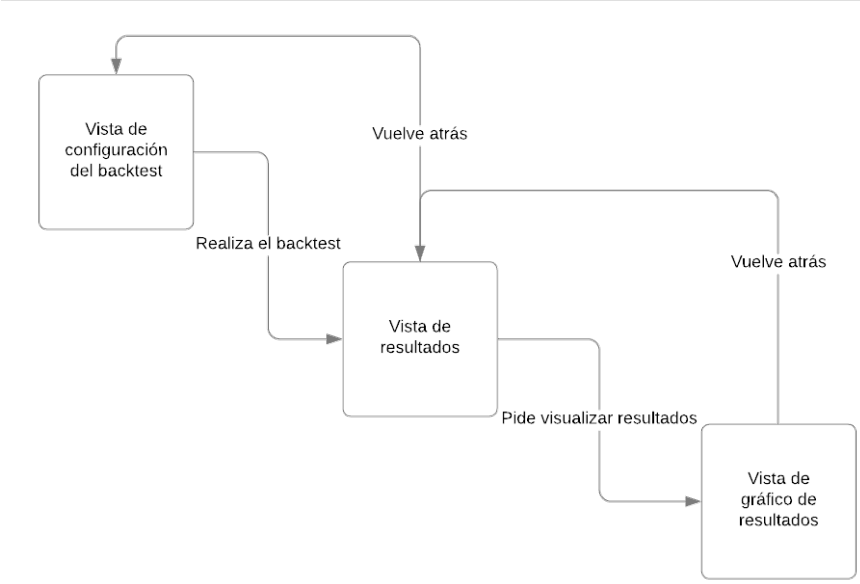


Figura 7: Diagrama de la relación entre vistas de la aplicación Strategist. Elaboración propia.

Vista de configuración del backtest

Esta es la vista principal de la aplicación. El usuario al entrar a la aplicación Strategist se encuentra con esta vista, que permite configurar todos los aspectos del *backtest*. Esta vista implementa las actividades de: «Elegir activo», «Comprobar activo», «Elegir estrategia», «Configurar una estrategia» y «Configurar *backtest*».

En la Figura 8 podemos observar la maqueta de la vista de configuración del *backtest*. En esta se puede observar las distintas acciones que debe realizar el usuario para configurar el

backtest, lo cual incluye las actividades mencionadas anteriormente. Además, desde el botón *backtest* se podrá acceder a la vista de resultados, realizando así la fase de *backtesting*.

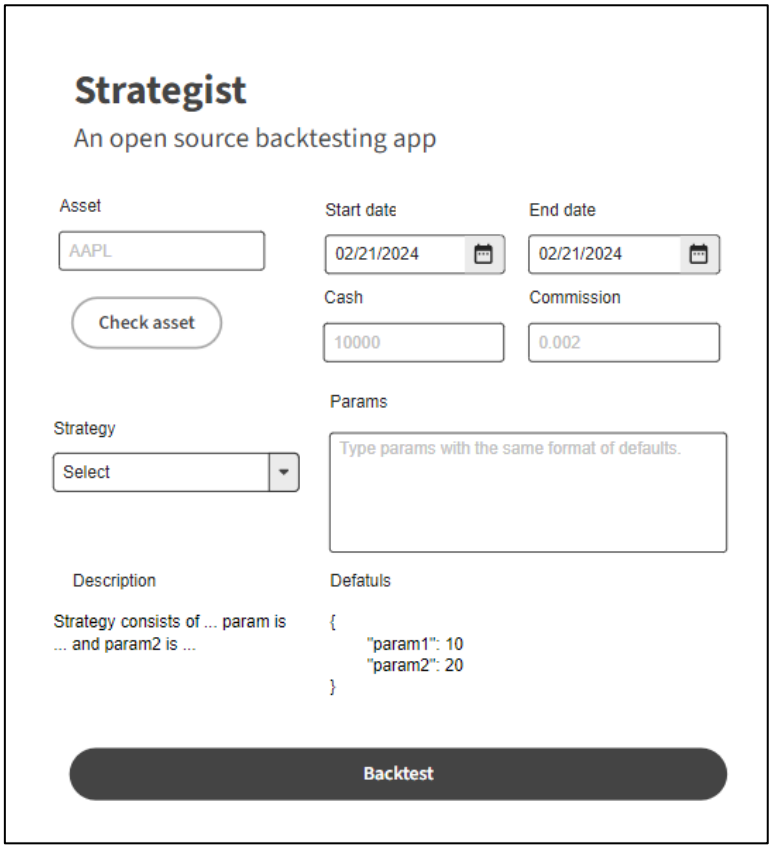


Figura 8: Maqueta de la vista de configuración de backtest.
Elaboración propia.

Vista de resultados

Esta es la vista que presenta los resultados del *backtesting*. Resulta muy interesante de cara al usuario, ya que es una de las formas para descubrir si el rendimiento de la estrategia es el buscado o no. Esta vista implementa el caso de uso de «Realizar *backtest*», ya que se pueden observar los resultados de éste.

En la Figura 9 se puede observar la maqueta de la vista. Esta maqueta es muy sencilla y se obtendrán todas las estadísticas disponibles sobre el *backtest* realizado. Desde el botón de atrás se puede volver a la vista de configuración de *backtest*, mientras que con el botón de visualizar gráfico se puede ir a la vista de gráfico de resultados.

Vista de gráfico de resultados

Esta vista viene proporcionada por la librería de Backtesting.py, por lo que no está maquetada. En esta vista se pueden observar los resultados de manera gráfica a través del tiempo ficticio durante el *backtesting*. Esta vista puede resultar muy útil para comprender los tipos de *trades* que realiza la estrategia estudiada, ya que los gráficos incluyen esta información. Esta vista implementa el caso de uso de «Visualizar resultado».

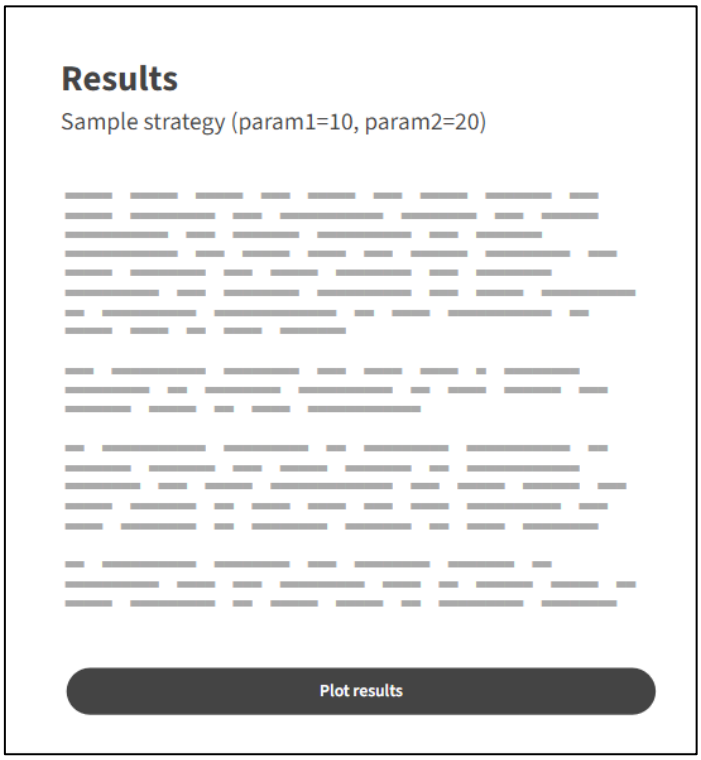


Figura 9: Maqueta de la vista de resultados.
Elaboración propia

4.4 Tecnologías usadas

La aplicación Strategist sigue la filosofía del código abierto, por lo que todas las tecnologías usadas para su funcionamiento también deben serlo. Durante la exploración del estado del arte (véase 2.1) se exploraron distintas opciones tanto para la realización del *backtesting* mediante la programación como para la obtención de datos históricos. En la sección de propuesta sobre el estado del arte (véase 2.3) se decide que las librerías más adecuadas para estas funcionalidades son Backtesting.py y YFinance, respectivamente. Ambas librerías son de código abierto y tienen una amplia gama de funcionalidades útiles para la creación de Strategist.

Por otra parte, tenemos la parte de interfaz de usuario de la aplicación. Existen una gran cantidad de librerías que permiten realizar interfaces de usuario en Python. Gracias a que la aplicación está escrita en Python, la mayoría de estas librerías tienen la capacidad de ser multiplataforma. La interfaz de usuario de la aplicación Strategist está basada en la librería Eel[25]. Eel es una librería de código abierto que permite crear aplicaciones de escritorio en Python utilizando la combinación de HTML, CSS y Javascript, usualmente utilizado en el desarrollo de aplicaciones web. Esta librería ofrece la ventaja de, partiendo de la base de conocimientos del desarrollo web, poder crear aplicaciones de escritorio fácilmente. Además, la interfaz de usuario puede reutilizarse si hubiese un cambio de arquitectura, virando hacia un sistema de cliente-servidor o similares. Por otra parte, Eel facilita la comunicación entre la interfaz de usuario y la lógica de negocios mediante su integración de Python en Javascript y viceversa, lo cual la hace el candidato perfecto para la aplicación de Strategist.

Por último, cabe mencionar que para que Strategist sea de código abierto, el código debe ser accesible y modificable por cualquier persona que quiera trabajar en el proyecto. GitHub[26] es la solución ideal para esta situación, ya que, con un repositorio público, cualquier persona puede descargar y utilizar el código. Para la correcta utilización de GitHub, se debe utilizar Git[27], que es una herramienta de control de versiones ampliamente utilizada.

[25] Información disponible en: <`https://github.com/python-eel/Eel`>
[26] Información disponible en: <`https://github.com/`>
[27] Información disponible en: <`https://git-scm.com/`>

CAPÍTULO 5
Desarrollo de la solución Strategist

Strategist se desarrolla siguiendo los principios del código abierto, las buenas prácticas a la hora de desarrollar software y teniendo en mente los requisitos de la solución. El desarrollo de la solución se ha realizado manteniendo un control de versiones limpio con Git, mediante el guardando versiones conforme el cumplimiento de los hitos marcados. Se ha utilizado un entorno de desarrollo adecuado para la programación en Python. Por otra parte, siguiendo las bases de la metodología XP, se ha utilizado una metodología de TDD, que se explicará más adelante. Esto ha resultado en una aplicación funcional, con resultados aceptables respecto a los requisitos.

5.1 Entorno de desarrollo

Para el desarrollo de la solución se ha utilizado un entorno de desarrollo integrado, o IDE. Este tipo de herramientas permiten desarrollar software de manera eficiente, ya que ofrecen una gran cantidad de útiles, como, por ejemplo, enfatizado de sintaxis o autocompletado de código. Para el proyecto se ha utilizado el editor Visual Studio Code[28], o VSCode, en conjunto con las extensiones necesarias para un correcto desarrollo de software en Python utilizando este editor.

Además, para mantener el código ordenado se han utilizado tanto un gestor de dependencias, como distintas herramientas de calidad del código. Estas últimas, integradas con el IDE han hecho la experiencia de desarrollo más eficiente.

5.1.1 Dependencias

La gestión de dependencias de un proyecto puede resultar una tarea sumamente compleja. Sin un gestor de dependencias esto puede resultar imposible, ya que, según [27], se estima que el 76% del código de una aplicación viene dado por sus dependencias de código abierto.

En Python existen distintas opciones para gestionar las dependencias, siendo la creación de un entorno virtual mediante el módulo «venv[29]» de la librería estándar del lenguaje. Sin embargo,

[28] Información disponible en: <`https://code.visualstudio.com/`>
[29] Información disponible en: <`https://docs.python.org/3/library/venv.html`>

la aproximación tomada por este proyecto ha sido utilizar el gestor de dependencias llamado Poetry[30].

Poetry es un gestor de dependencias y empaquetador para aplicaciones escritas en Python que permite gestionar las dependencias de un proyecto de manera fácil, ordenada y determinista, ya que permite fijar las versiones de las dependencias que se utilizaran. Poetry permite también conocer que dependencias tienen a su vez otras dependencias, lo cual puede resultar útil para decidir que dependencias son imprescindibles o no. Por último, cabe destacar que Poetry permite diferenciar de manera cómoda entre dependencias imprescindibles para el funcionamiento de la aplicación, dependencias imprescindibles para desarrollar la aplicación y dependencias no necesarias pero que aportan funcionalidad extra a la aplicación.

Además, Poetry permite empaquetar y publicar librerías y aplicaciones en Python, aunque esta funcionalidad está más relacionada con el despliegue de la aplicación.

5.1.2 Buenas prácticas

Las llamadas «buenas prácticas» son un conjunto de recomendaciones que, aplicadas en un contexto razonable, dan como resultado un código legible, reutilizable y escalable. Existen una gran cantidad de buenas prácticas para muchas situaciones, y algunas difieren incluso entre lenguajes de programación. Sin embargo, es importante tenerlas en cuenta para obtener un software de calidad.

En el caso de Strategist, Python es un lenguaje interpretado de tipado dinámico, por lo que es muy susceptible a errores en tiempo de ejecución, ya que no hay un análisis estático estricto.

Para intentar disminuir los errores provenientes de esta situación, y tener un código fuente legible y seguro, se utilizan dos analizadores estáticos de código: Pylint[31] y MyPy[32].

Pylint es una herramienta de análisis estático de código centrada en reducir los *code smells*, que son partes del código que pueden llevar a errores o a malas prácticas. Además, Pylint permite de cierta manera estandarizar ciertas decisiones sobre el aspecto del código.

Por otra parte, MyPy es una herramienta de análisis estático de código centrada en la seguridad de las anotaciones de tipos en Python. Esto es debido a que, aunque sea un lenguaje de tipado dinámico, Python permite anotar los tipos de las variables. El problema viene en que estos tipos son ignorados por el intérprete, por lo que, sin una herramienta que los verifique, pueden

[30] Información disponible en: <`https://python-poetry.org/`>
[31] Información disponible en: <`https://pylint.readthedocs.io/en/stable/`>
[32] Información disponible en: <`https://mypy-lang.org/`>

llevar a errores. Utilizar tipos en Python puede resultar muy ventajoso, ya que evita muchos errores en tiempo de ejecución y facilita el desarrollo de aplicaciones.

Por último, para trabajar en proyectos donde potencialmente haya muchos colaboradores, es importante mantener ciertos estándares de formato y de programación. Esto hace que el código fuente resulte homogéneo y fácil de entender. Aunque Pylint ayude a cumplir esta condición, resulta deficiente en algunos aspectos, por lo que también se utiliza Autopep8[33]. Autopep8 es una herramienta de formato que aplica las reglas descritas por la documentación de Python (véase el PEP 8[34]) al código fuente.

5.2 Desarrollo de código abierto

Una de las partes fundamentales del proyecto de Strategist es que va a ser de código abierto. En el apartado de tecnologías usadas (véase 4.4) se explica que se utilizan tanto Git como GitHub. Estas dos herramientas son clave para que Strategist sea de código abierto. Sin embargo, se tienen que definir unas bases de cómo utilizar estas herramientas a nivel de contribuidor, ya que, aunque durante el desarrollo inicial no haya problema, una vez publicado el código, pueden participar muchos colaboradores en el proyecto. Por ello se crean dos ficheros: «CONTRIBUTORS.md» y «CODE_OF_CONDUCT.md», ambos disponibles en el Anexo C.1.

El fichero «CONTRIBUTORS.md», explica el procedimiento que debe seguir un desarrollador que quiera contribuir de alguna manera al proyecto de Strategist. Cabe remarcar, que, como se explica en el fichero, el estilo de los mensajes de guardado de versiones es siguiendo el estándar de Conventional Commits[35], ya que con él se puede mantener el repositorio de Git ordenado.

Por otra parte, el fichero «CODE_OF_CONDUCT.md», hace referencia al comportamiento que debe tener cualquier persona que decida participar, ya sea comentando sobre la herramienta, como contribuyendo al proyecto.

Además, cabe recordar la necesidad de utilizar licencias, mencionada durante el análisis del problema (véase 0), por lo que también se incluye un tercer fichero «LICENSE», que contiene una licencia para el proyecto Strategist. Esta licencia es la «GNU General Public License[36]» en su tercera versión, que garantiza a los usuarios la libertad de ejecutar, estudiar, compartir (copiar) y modificar el software. Sin embargo, la parte positiva es que establece que cualquier trabajo derivado debe estar disponible bajo la misma licencia, asegurando que las libertades otorgadas

[33] Información disponible en: <`https://pypi.org/project/autopep8/`>
[34] PEP disponible en: <`https://peps.python.org/pep-0008/`>
[35] Estándar disponible en: <`https://www.conventionalcommits.org/`>
[36] Licencia disponible en: <`https://www.gnu.org/licenses/gpl-3.0.html`>

por la versión original se mantengan en las versiones modificadas, y continuando así la cadena de código abierto.

5.3 Aspectos importantes

Durante el desarrollo, se ha respetado el diseño previo de la aplicación, teniendo en mente los requisitos de la aplicación. La aplicación, tiene dos paquetes de Python principales, el paquete de la aplicación y las pruebas. Las pruebas se explican en la sección 5.5.

El paquete de la aplicación, llamado «strategist» contiene el *entrypoint* de la aplicación, que es un fichero llamado «__main__.py» que simplemente inicia la aplicación. Para iniciarla, hace uso de la función «start» del módulo del fichero «app.py», que inicia la interfaz de usuario. Además de esto, el paquete contiene tanto el paquete Motor como el paquete IGU (véase la Figura 5), que en el código son llamados «engine» y «gui». Estos dos paquetes contienen la funcionalidad de la aplicación.

Para poder seguir las explicaciones del desarrollo, se adjunta el árbol de ficheros del proyecto en el Anexo C.2.

5.3.1 Implementación del paquete «engine» (Motor)

El paquete «engine» contiene toda la lógica de negocio y datos de la aplicación. Éste expone desde su base la clase «Backtester», cuya definición se encuentra en el archivo «backtester.py». Esta clase se crea a partir de todos los datos necesarios para la configuración de un *backtest*. Estos son: la estrategia a utilizar, el símbolo del activo, la fecha de inicio y final de los datos históricos, la cantidad de dinero y la comisión por operación. Cabe remarcar, que con la creación de una instancia de la clase ocurre dos eventos significativos: se inicializa un objeto «Backtest» de la librería Backtesting.py con los datos iniciales y se obtiene la instancia de la clase «Session» con la que se descarga el histórico del activo. Por último, la clase tiene tres métodos: la propiedad «stats», que devuelve los resultados del *backtest*; el método «run», que ejecuta el *backtest* y devuelve sus resultados; y el método «plot» que guarda en una vista la gráfica de resultados del último *backtest*.

Por otra parte, tenemos la clase «Session», cuya definición se encuentra en el archivo «data.py». Esta clase es relativamente especial, ya que se trata de una clase que sigue el patrón «singleton», que consiste en que solo puede haber una instancia de la clase en la aplicación [28]. Esto es debido a que se requiere implementar una caché de datos históricos, y para que esta sea compartida entre *backtests*, solo debe haber una instancia de «Session». Esta clase tiene dos métodos públicos: «is_valid_symbol», que devuelve si el símbolo es un activo reconocido; y «get_data», que devuelve los datos históricos de un símbolo en un período de fechas.

Cabe destacar que está clase implementa un sistema de cachés que recuerda los últimos 5 datos históricos distintos consultados. Para ello se definen dos métodos privados que son ejecutados a la hora de obtener los datos históricos. Los métodos se pueden encontrar en el Algoritmo 4. Además, se puede observar que la estructura de datos utilizada para guardar la caché es la cola de prioridad, que mantendrá una lista ordenada de los datos de más antiguos a menos.

```
def _data_in_cache(self, symbol, date_period):
    # Remove the cached tickers that are older than 1 day.
    now = time.time()
    for entry in self._cached_tickers:
        if now - entry.last_used > 60 * 60 * 24:
            self._cached_tickers.remove(entry)
    return any(
       entry.name == symbol and entry.date_period == date_period
       for entry in self._cached_tickers)

def _cache_dat(self, symbol, date_period, data):
    if len(self._cached_tickers) >= 5:
        heapq.heapreplace(
            self._cached_tickers,
            _TickerEntry(symbol, data, date_period, time.time())
        )
    else:
        heapq.heappush(
            self._cached_tickers,
            _TickerEntry(symbol, data, date_period, time.time())
        )
```

Algoritmo 1: Definición de los métodos del sistema de caché de la clase «Session»
Elaboración propia

Volviendo a la clase «Backtester», ésta requiere como argumento en su creación la especificación de una estrategia. Las estrategias y su lógica se encuentran definidas en el paquete «strategies».

Implementación del paquete «strategies» (Estrategias)

En la base del paquete «strategies» se expone en primer lugar, una lista de estrategias permitidas en la constante «ALLOWED_STRATEGIES», que contiene el nombre, la descripción y los parámetros por defecto de éstas. La lista sirve como información que después utilizar la interfaz de usuario. Las estrategias disponibles actualmente son:

- Cruce de medias móviles simple: Consiste en controlar los cruces de dos indicadores de medias móviles, uno de largos periodos y el otro de cortos. Cuando el indicador de corto supera al indicador de largo se debe entrar en una operación, y salir en el caso contrario.

- Cruce de medias móviles exponenciales: Análogo al cruce de medias móviles simples, pero calculando los indicadores aplicando un multiplicador a la diferencia entre el precio actual y el resultado de la media móvil anterior, y añadiéndolo al resultado anterior. Esto hace que sea más sensible a nueva información.
- Umbral de índice relativo de fuerza: Consiste en controlar los cruces del índice relativo de fuerza con unos valores umbral prefijados. Si cruza por debajo al umbral inferior se debe de entrar en una operación, y al contrario si supera el umbral superior. El índice relativo de fuerza mide la velocidad y el cambio de los movimientos de precio, y oscila entre 0 y 100.

Por otra parte, también se expone la clase abstracta «Strategy» la hereda de la clase «backtesting.Strategy» y tiene tres métodos abstractos propios: la propiedad «params», que devuelve los parámetros de la estrategia; el método estático «get_default_params», que devuelve los parámetros por defecto de la estrategia; y el método «configure», que configura la clase y se devuelve a sí misma configurada.

Esta clase se define debido a la estrategia utilizada para la creación y configuración de estrategias. La clase «Backtester» requiere como dependencia la clase de la estrategia a testear, y no una instancia de esta. Gracias a que Python permite las operaciones de orden superior, se realiza una adaptación del patrón «factory», que originalmente consiste en crear una clase que devuelva instancias preconfiguradas de otra no accesible normalmente [29]. La adaptación realizada consiste en que ahora las clases de estrategia se crearán usando una función que devuelve la clase configurada con los parámetros pasados a esta misma función. Se puede observar cómo funciona la configuración de estrategias y como se deben pasar a la clase «Backtester» en el Algoritmo 2.

```
class SMACrossStrategy(Strategy):
    # Strategy parameters
    short_lag: int
    long_lag: int

    ...

    @classmethod
    def configure(cls, **kwargs):
        default_params = cls.get_default_params()
        cls.short_lag = kwargs.get("short_lag") or \
            default_params["short_lag"]
        cls.long_lag = kwargs.get("long_lag") or \
            default_params["long_lag"]
        return cls
```

```
    ...

def sma_cross_factory(short_lag=None, long_lag=None):
    return SMACrossStrategy.configure(
        short_lag=short_lag,
        long_lag=long_lag
    )

...

sma_cross = sma_cross_factory()
Backtester(sma_cross, ...)
```

Algoritmo 2: Simplificación del uso del patrón «factory» para la creación y configuración de estrategias
Elaboración propia.

5.3.2 Implementación del paquete «gui» (IGU)

El paquete «gui» por otra parte, provee la funcionalidad de interfaz de usuario a la aplicación. Aunque la interfaz de usuario se divide en vistas, debido a la naturaleza del código de la librería Eel y la relativa sencillez del proyecto, todo el código de Python se encuentra en un único fichero «__init__.py», que solamente expone para importar el método «run», que permite iniciar la interfaz de usuario; y la constante «WEB_ASSETS_PATH», que contiene la ruta a la carpeta con los contenidos de las vistas de la interfaz de usuario.

En la carpeta «static» se pueden encontrar todos los archivos relacionados con las tecnologías web usadas para la interfaz de usuario. Hay un fichero HTML por cada vista, un fichero CSS que aplica estilos generales y retoca algunos estilos propios de cada vista, y un fichero de Javascript, que controla la lógica de la interfaz de usuario.

El fichero de Javascript es capaz de controlar la lógica de usuario gracias a que Eel permite exponer funciones escritas en Python a Javascript, que puede tratar los resultados de esta. El funcionamiento es sencillo, y se puede observar en el Algoritmo 3.

```
# __init__.py
@eel.expose
def validate_symbol(symbol):
    return session.is_valid_symbol(symbol)

// main.js
function mark_validation(is_valid) {
    if (is_valid) {
        ...
    } else {
        ...
    }
```

```
}

// get_symbol already defined. It retrieves symbol from the UI.
eel.validate_symbol(get_symbol())(mark_validation)
```

Algoritmo 3: Simplificación del proceso de comunicación entre la lógica de interfaz de usuario. Elaboración propia.

Para la implementación de las vistas, se ha seguido las maquetas definidas durante el diseño de la capa de presentación (véase 4.3.2).

5.4 Resultados

Para ilustrar los resultados del desarrollo de la aplicación, se recrea una situación habitual, donde un usuario decide realizar *backtesting* sobre el activo de AAPL.

Primero, se comprueba que el activo existe, para luego configurar los elementos del *backtest*, siendo la fecha de inicio el 10/01/2023 y la fecha de final el 10/2/2024, 10.000 euros la cantidad de dinero y 0.002 la comisión por operación. Después, se selecciona la estrategia «SMA Cross» y luego se configuran los parámetros deseados, en este caso se pueden ver, junto con el resultado de aplicar todo lo demás, en la Figura 10.

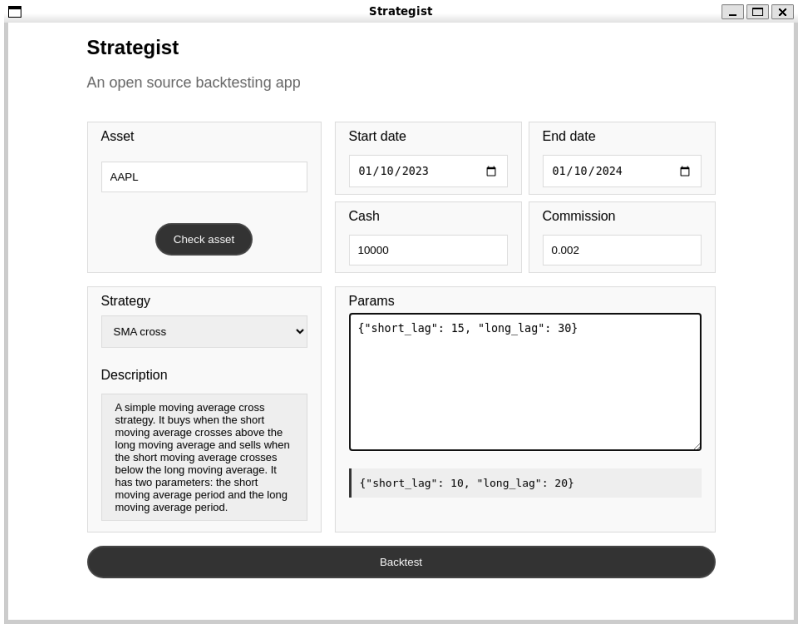
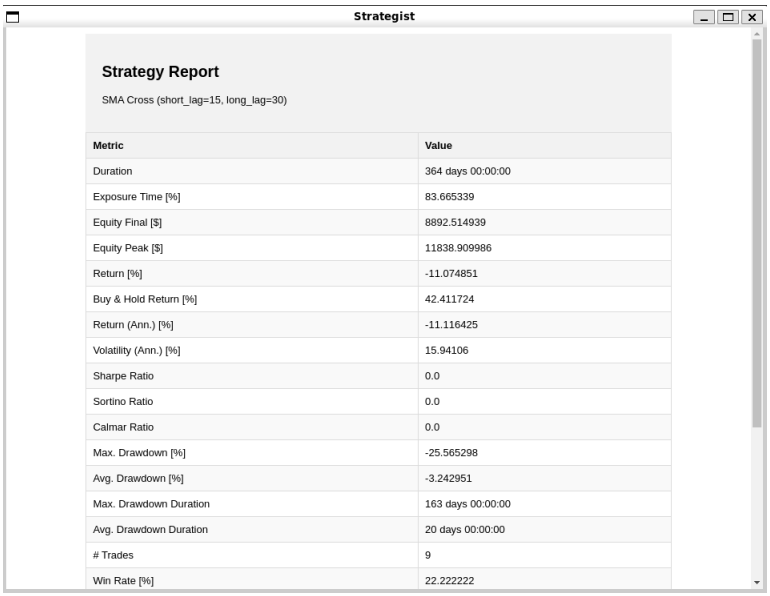


Figura 10: Resultado de la vista de configuración de backtest*. Elaboración propia.*

Al ejecutar el *backtest*, obtenemos los resultados de la Figura 11. Podemos observar que el resultado es malo, ya que el retorno termina siendo negativo.



Metric	Value
Duration	364 days 00:00:00
Exposure Time [%]	83.665339
Equity Final [$]	8892.514939
Equity Peak [$]	11838.909986
Return [%]	-11.074851
Buy & Hold Return [%]	42.411724
Return (Ann.) [%]	-11.116425
Volatility (Ann.) [%]	15.94106
Sharpe Ratio	0.0
Sortino Ratio	0.0
Calmar Ratio	0.0
Max. Drawdown [%]	-25.565298
Avg. Drawdown [%]	-3.242951
Max. Drawdown Duration	163 days 00:00:00
Avg. Drawdown Duration	20 days 00:00:00
# Trades	9
Win Rate [%]	22.222222

Figura 11: Resultado de la vista resultados.
Elaboración propia

Para observar cual ha sido la evolución temporal, se decide observar el gráfico de los resultados. La Figura 12 muestra que hubo un máximo de retorno de hasta el 118.2%, pero con el tiempo cayó.

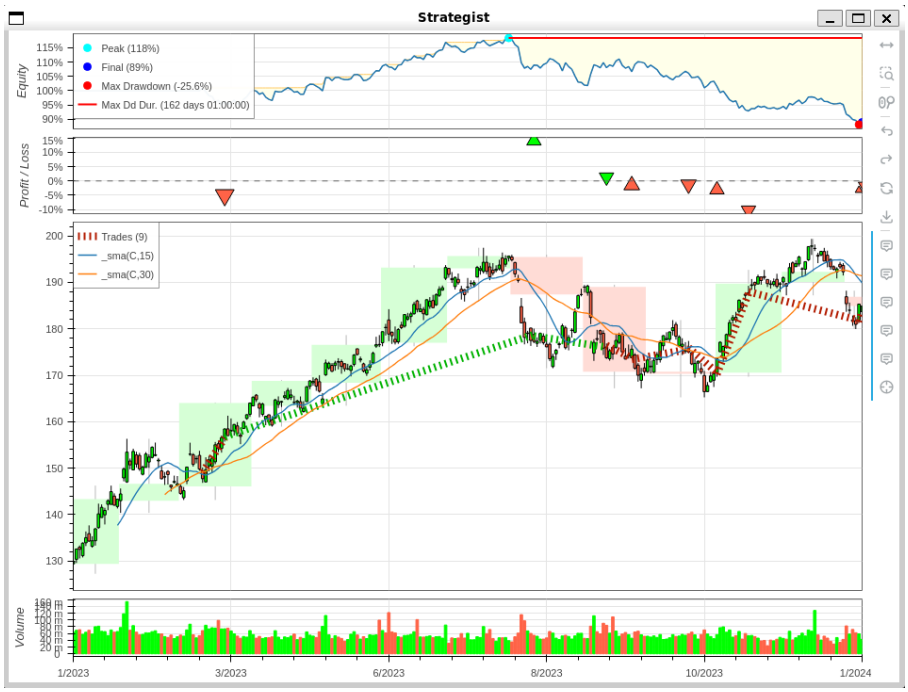


Figura 12: Resultado de la vista de gráfico de resultados.
Elaboración propia.

5.5 Pruebas y TDD

La metodología utilizada en este trabajo, XP, proporciona una forma de desarrollo muy peculiar: el desarrollo basado en pruebas, TDD, por sus siglas en inglés, en adelante.

Según [29], esta metodología de desarrollo se puede dividir en dos fases. La primera es escribir las pruebas de la nueva funcionalidad a añadir. Es importante que todas las pruebas de la nueva funcionalidad fallen al inicio. En la segunda fase, se trata de desarrollar la nueva funcionalidad, haciendo que las pruebas planteadas al inicio se superen.

Esta metodología centra el desarrollo en las pruebas unitarias, dejando al terminar el desarrollo un código fuente probado y relativamente seguro. Además, los códigos desarrollados en TDD son mucho más fáciles de depurar, ya que las pruebas son específicas y puede remarcar donde se encuentra el problema.

El único inconveniente que puede tener el TDD es que consume significativamente más tiempo comparado con el modo habitual de trabajo, que es escribir las pruebas después del código [30].

Para el proyecto, se ha utilizado Pytest[37] como *framework* de pruebas, ya que ofrece la posibilidad de reducir la cantidad de código repetitivo que necesita la librería estándar de Python.

Las pruebas se encuentran en un directorio separado a la aplicación. Es importante mencionar que no se ha considerado realizar pruebas sobre la interfaz de usuario, ya que es difícil aislar las funcionalidades para las pruebas unitarias. Por ello, solamente se han realizado pruebas sobre las funcionalidades del paquete «engine».

La metodología TDD tiene la ventaja de que obliga a definir con claridad cual va a ser el uso de tus clases, por lo que las pruebas realizadas están centradas en el uso real de las funciones.

5.6 Implantación

Para acabar, una vez terminada la versión de la aplicación, esta debe poder ejecutarse en los entornos requeridos. Por suerte, Python es multiplataforma, y puede ejecutarse en cualquier sistema operativo, con la precondición de que el sistema tenga el intérprete de Python instalado.

Para sortear esta precondición, existen distintos sistemas de empaquetado en el ecosistema de Python. Debido a la librería de interfaz de usuario utilizada, la librería más adecuada para esto es PyInstaller[38].

Esta librería permite empaquetar la aplicación en un ejecutable único que puede ser ejecutado sin necesidad de tener Python instalado. Sin embargo, tiene un problema. Aunque la librería funciona en Windows, MacOS y Linux, los ejecutables no son multiplataforma, por lo que habría que generar un ejecutable por sistema operativo soportado.

Por otra parte, Poetry también tiene funcionalidades de gestión de empaquetados y publicación de librerías. Para Strategist, resulta muy conveniente poder integrar las dos herramientas, ya que abre camino a futuras mejoras, como las tuberías de desarrollo e integración continuos.

[37] Información disponible en: <`https://docs.pytest.org/en/stable/`>
[38] Información disponible en: <`https://pyinstaller.org/en/stable/`>

CAPÍTULO 6
Conclusiones

El mundo del *trading* es muy amplio y apasionante. La diferencia principal del *trading* con la inversión en instrumentos financieros es que el rédito principal de las operaciones de *trading* siempre viene de la diferencia de precios entre la compra y la venta [2].

La misma naturaleza de las operaciones de *trading* las hace muy complicadas de estudiar y realizar de manera sistemática, ya que el precio de un instrumento financiero está determinado por una gran cantidad de factores y suele ser muy difícil de predecir [3]. A raíz de esto, surgió el análisis técnico, una rama del estudio financiero que pretende predecir el precio de instrumentos financieros.

Basándose en esta teoría, surge una nueva forma de *trading* basa en reglas o estrategias, llamada *trading* estratégico [6]. Las estrategias utilizadas en este tipo de *trading* pueden probarse con datos históricos utilizando herramientas de *backtesting*, y con ello obtener información sobre la efectividad de las estrategias e, incluso, obtener información sobre el mercado sobre el que se ha operado. Además, el *backtesting* permite obtener todas estas ventajas sin un riesgo de capital real asociado.

Sin embargo, la gran mayoría de estas herramientas no son de acceso gratuito y código abierto, por lo que, la motivación de este trabajo es la de democratizar este tipo de herramientas *FinTech*, fomentando la innovación y la inclusividad en los mercados financieros.

Para ello, se han estudiado las soluciones actuales, divididas en tres grandes grupos: la creación de sistemas íntegramente, los *frameworks* y las herramientas. A partir de estas se ha realizado una crítica y búsqueda de problemas, que se han utilizado como base para realizar la propuesta de valor de realizar un proyecto de código abierto.

Asimismo, se han analizado los aspectos éticos y legales que puede tener un proyecto enfocado al *backtesting*, específicamente uno de código abierto. Esto ha sido útil para plantear una aplicación segura y razonable para la comunidad. Con ello se ha propuesto la solución Strategist, junto con sus requisitos y un plan de trabajo.

Además, para poder proveer una solución sencilla y usable para el usuario, se ha realizado un diseño enfocado al usuario, planteando sus actividades, así como realizando una interfaz de usuario que permita un uso cómodo y fácil de la aplicación.

Por otra parte, durante el planteamiento como el desarrollo de la aplicación se ha utilizado la metodología XP, que resulta en casi todos los proyectos en soluciones software de calidad. También cabe remarcar el hecho de que todo el proyecto se ha realizado desde una óptica del código abierto, utilizando en todo momento código abierto.

Por consiguiente, todos los objetivos específicos han sido cubiertos y conseguidos durante la realización del trabajo.

En conclusión, el objetivo general del TFG se cumple, ya que Strategist es una solución software que permite realizar *backtesting* y consigue aportar valor a la comunidad, gracias a lo comentado anteriormente.

Por tanto, la aportación de este TFG consiste en una solución de código abierto que permite democratizar la realización del *backtesting* de estrategias de trading sobre el mercado de valores, aportando valor a la comunidad de *trading*.

La gran ventaja del código abierto es la gran capacidad de ampliar funcionalidades y mejorar que tienen los proyectos. Por eso, este trabajo no es más que un punto de partida para Strategist, que seguirá creciendo y nutriéndose de lo que la comunidad quiera aportar.

6.1 Relación del trabajo desarrollado con los estudios cursados

El grado en ingeniería informática ha aportado una base y unos conocimientos fundamentales muy necesarios para la creación de este trabajo de fin de grado. Aunque todas las asignaturas cursadas han aportado su grano de arena, ha habido tres en las que se ha basado la gran parte del trabajo.

Primero de todo, cabe destacar todas las asignaturas de programación de la carrera, ya que gracias a su contenido se ha podido desarrolla e implementar un software de calidad y eficiente. En cuanto a las asignaturas específicas, la asignatura de «Gestión de proyectos» ha brindado conocimientos sobre las distintas metodologías existentes para gestionar proyectos y ha permitido elaborar un plan de trabajo para el proyecto. Por otra parte, la asignatura de «Ingeniería del software» ha permitido diseñar el software del proyecto de manera escalable y eficiente, manteniendo en todo momento un foco en los requisitos del proyecto. Por último, la asignatura de «Interfaces persona computador» ha servido como base para el diseño de una interfaz de usuario usable, intuitiva y adaptada las historias de usuario de la aplicación.

CAPÍTULO 7
Trabajos futuros

Aunque se ha cumplido el objetivo general del TFG, este tiene limitaciones, sobre todo en funcionalidades. Por ello se listan distintas funcionalidades que podrían añadirse al proyecto en el futuro.

Ampliar catálogo de estrategias

Incluir más estrategias en la aplicación. El número actual de estrategias es limitado, y puede extenderse fácilmente.

Conexión con *exchanges*

Permitir que la aplicación se conecte a *exchanges* y utilizar las estrategias para realizar *trading* algorítmico. Esto cambia un poco el paradigma de la aplicación, aunque puede ser integrable.

Creador de estrategias

Incluir un creador de estrategias que permita, mediante un sistema de bloques predefinidos o similar, crear nuevas estrategias para realizar *backtesting.*

Ampliar el repertorio de pruebas

Además de las pruebas unitarias de TDD, sería muy interesante realizar también pruebas de integración, pruebas *end-to-end* o pruebas de despliegue.

Integración de tuberías de CI/CD

Para un proyecto como Strategist podría resultar muy interesante incluir tuberías de CI/CD, ya que permiten mejorar el desarrollo de nuevas funcionalidades y la integración en el proyecto de estas.

Histórico local de *backtests* realizados

Sería interesante mantener un histórico localmente de las estrategias realizadas en los últimos días, para así comparar con mayor facilidad los resultados de cada una.

Referencias

[1] «What is trading?» en *IG*. <https://www.ig.com/en/trading-need-to-knows/what-is-trading> [Consulta: 8 de julio de 2023].

[2] FOLGER, J. (2023). «Investing vs. Trading: What's the difference?» en *Investopedia*, 30 de julio. <https://www.investopedia.com/ask/answers/12/difference-investing-trading.asp> [Consulta: 12 de agosto de 2023].

[3] SINGH, A. (2023). «Stock prices prediction using machine learning and deep learning», en *Analytics Vidhya*, 1 de mayo. <https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python> [Consulta: 12 de agosto de 2023].

[4] DE LA LOMA, A. (2022). «Breve historia del análisis técnico: Charles Dow» en *FinanCer Training*, 10 de mayo. <https://financertraining.com/breve-historia-del-analisis-tecnico-charles-dow> [Consulta: 12 de agosto de 2023].

[5] KIRKPATRICK, C. D. y DAHLQUIST, J. A. (2006). *Technical analysis: The Complete Resource for Financial Market Technicians*. Financial Times Prentice Hall.

[6] PERRIN-MONLOUIS, P. (2021). «Invierte como un cristiano», Edubourse, 21 de octubre. <https://edubourse.com/es/guide-bourse-investir-comme-chretien> [Consulta: 11 de septiembre de 2023].

[7] CFI TEAM (2022). «Backtesting» en *Corporate Finance Institute*, 28 de diciembre. <https://corporatefinanceinstitute.com/resources/data-science/backtesting> [Consulta: 20 de agosto de 2023].

[8] THAKAR, C. y SINGH, V. (2023). «How to backtest, analysis, strategy, and more» en *QuantInsti*, 5 de septiembre. <https://blog.quantinsti.com/backtesting> [Consulta: 11 de septiembre de 2023].

[9] COCKBURN, A. (2002). *Agile software development*. Addison-Wesley Longman Publishing.

[10] BECK, K. y ANDRES, C. (2004). *Extreme programming explained: Embrace Change*. Pearson Education.

[11] NI, J. y ZHANG, C. (2005). «An efficient implementation of the backtesting of trading strategies» en *Parallel and Distributed Processing and Applications*, Y. Pan, D. Chen, M. Guo, J. Cao, J. Dongarra. Heidelberg, Berlin: Springer. vol. 3758, p. 126-131. <https://dx.doi.org/10.1007/11576235_17> [Consulta: 13 de octubre de 2023].

[12] OLIVEIRA, C. (2023). «Backtesting trading strategies in C++» en *Options and Derivatives Programming in C++23*. Berkeley, CA: Apress. p. 241-247. <https://dx.doi.org/10.1007/978-1-4842-9827-5_14> [Consulta: 15 de octubre de 2023].

[13] RAI, R. (2022). «Extracting and analyzing live stock data in Excel» en *Medium*, 8 de enero. <https://krrai77.medium.com/extracting-and-analyzing-live-stock-data-in-excel-33695e9e1324> [Consulta: 6 de noviembre de 2023].

[14] WORSLEY, S. (2022), «What is Python? - The Most Versatile Programming Language» en *DataCamp*, 7 de marzo. <https://www.datacamp.com/blog/all-about-python-the-most-versatile-programming-language> [Consulta: 25 de noviembre de 2023].

[15] DONGO, I. *et al.* (2021), «A qualitative and quantitative comparison between web scraping and API methods for Twitter credibility analysis» en *International Journal of Web Information Systems*, vol. 17, no. 6, p. 580-606. <https://dx.doi.org/10.1108/ijwis-03-2021-0037> [Consulta: 12 de diciembre de 2023].

[16] *Backtesting.py API documentation*. <https://kernc.github.io/backtesting.py/doc/backtesting/> [Consulta: 12 de diciembre de 2023].

[17] PRESSMAN, R. S. (2010). *Ingenieria de software: Un enfoque práctico*. México, D. F.: McGrawHill Educación.

[18] SCARFONE, K., JANSEN, W., y TRACY, M. (2008). «Guide to General Server Security» en *National Institute of Standards and Technology Special Publications*, vol. 800, no. 123. <https://dx.doi.org/10.6028/NIST.SP.800-123> [Consulta: 12 de diciembre de 2023].

[19] España. Real Decreto Legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia. *BOE*, 22 de abril de 1996, núm. 97, p. 14369-14396.

[20] SEN, R., SUBRAMANIAM, C. y NELSON, M. L. (2008). «Determinants of the Choice of Open Source Software License» en *Journal of Management Information Systems*, vol. 25, no. 3, p. 207-240. <https://dx.doi.org/10.2753/mis0742-1222250306> [Consulta 12 de enero].

[21] AMBYSOFT INC. (2023). «User stories: An agile introduction» en *The Agile Modeling*, 26 de noviembre. <https://agilemodeling.com/artifacts/userstory.htm> [Consulta: 12 de enero de 2024].

[22] COHN, M. (2004). *User stories applied: For Agile Software Development*. Boston, MA: Addison-Wesley Professional.

[23] 10XERS (2023), «The importance of software architecture for the success of your project», en *Medium*, 23 de septiembre. <https://10xers.medium.com/the-importance-of-software-architecture-for-the-success-of-your-project-3abdaed5f335> [Consulta: 17 de enero de 2024].

[24] APPSIERRA (2023), «Software Architecture: N Tier, 3 Tier, 1 Tier, 2 Tier Architecture» en *Appsierra*, 2 de noviembre. <https://www.appsierra.com/blog/tiers-in-software-architecture> [Consulta: 17 de enero de 2024].

[25] UXPIN (2023), «What is a Mockup - The final layer of UI design», en *Studio by UXPin*, 3 de abril. <https://www.uxpin.com/studio/blog/what-is-a-mockup-the-final-layer-of-ui-design> [Consulta: 23 de enero de 2023].

[26] UXPIN (2023), «High-Fidelity prototyping vs Low-Fidelity - Which to choose when?» en *Studio by UXPin*, 10 de marzo. <https://www.uxpin.com/studio/blog/high-fidelity-prototyping-low-fidelity-difference> [Consulta: 24 de enero de 2024].

[27] SYNOPSYS (2023), «Open source Security and Analysis report», enero.<https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html> [Consulta: 2 de febrero de 2023]

[28] FREEMAN E. *et al.* (2004). *Head First Design Patterns*. O'Reilly Media, Inc.

[29] BECK, K. (2002). *Test-Driven Development by Example*. Addison-Wesley Professional.

[30] CANFORA, G. (2006). «Evaluating advantages of test driven development: a controlled experiment with professionals» en *ISESE '06: Proceedings Of The 2006 ACM/IEEE International Symposium On Empirical Software Engineering*. p. 364-371. <https://dx.doi.org/10.1145/1159733.1159788> [Consulta: 10 de febrero de 2023].

ANEXO A
Relación con los Objetivos de Desarrollo Sostenible de la Agenda 20-30

El anexo detalla el grado de relación de este TFG con los ODS de la Agenda 20-30.

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. **Fin de la pobreza**				X
ODS 2. **Hambre cero**				X
ODS 3. **Salud y bienestar**			X	
ODS 4. **Educación de calidad**		X		
ODS 5. **Igualdad de género**			X	
ODS 6. **Agua limpia y saneamiento**				X
ODS 7. **Energía asequible y no contaminante**				X
ODS 8. **Trabajo decente y crecimiento económico**			X	
ODS 9. **Industria, innovación e infraestructuras**		X		
ODS 10. **Reducción de las desigualdades**			X	
ODS 11. **Ciudades y comunidades sostenibles**				X
ODS 12. **Producción y consumo responsables**				X
ODS 13. **Acción por el clima**				X
ODS 14. **Vida submarina**				X
ODS 15. **Vida de ecosistemas terrestres**				X
ODS 16. **Paz, justicia e instituciones sólidas**			X	
ODS 17. **Alianzas para lograr objetivos**		X		

Tabla 2: Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

A priori, este TFG no tiene una relación directa con los Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas. Sin embargo, esto no disminuye la importancia que tiene estudiar los avances tecnológicos con estos objetivos de desarrollo globales. Aunque la alineación con los ODS puede no ser directa o explícita, es crucial comprender y articular las diversas formas en que esta herramienta puede contribuir a estos objetivos o influir en ellos. Por ello se ha realizado una tabla resumen donde se puede ver la contribución de este TFG a los ODS (véase en la Tabla 2).

Los ODS que tienen más relación con el proyecto son el «ODS 17: Alianzas para lograr los Objetivos» y el «ODS 9: Industria, innovación e infraestructura». Esto es debido a que a que la aplicación es de código abierto, y por tanto insta a la colaboración entre personas para lograr mejorar la herramienta. Esto permite conectar a personas de todo el mundo en un esfuerzo común para mejorar la aplicación, fomentando así la innovación. De hecho, gracias al código abierto personas diversas pueden unir sus distintas maneras de pensamiento para idear nuevas maneras de mejorar la aplicación.

Sobre el «ODS 4: Educación de calidad», el código abierto también permite intercambiar ideas y conocimiento con todos los colaboradores, por lo que se fomenta la enseñanza y se democratizan los conocimientos. Además, la aplicación puede usarse también como herramienta de aprendizaje sobre las posibles estrategias a probar.

La relación de la aplicación con los objetivos restantes establecidos en la Tabla 2 es más indirecta. Por lo que respecta al «ODS 3: Salud y bienestar», la estabilidad financiera y la reducción del estrés que puede proporcionar el éxito del comercio pueden contribuir indirectamente a la salud mental. En cuanto al «ODS 5: Igualdad de género» y el «ODS 10: Reducción de desigualdades», la democratización de la aplicación con el código abierto podría ayudar a igualar las condiciones en el sector financiero, aunque su impacto depende de su alcance y uso en las diferentes comunidades. Sobre el «ODS 16: Paz, justicia e instituciones sólidos», se contribuye a éste mediante la intención de desarrollar una comunidad financiera más inclusiva y educada, lo cual, aunque de manera muy indirecta, apoya el espíritu de sociedades pacíficas e inclusivas. Por último, se contribuye al «ODS 8: Trabajo digno y crecimiento económico» debido a que se facilita el acceso a los mercados financieros y permite a un mayor número de personar participar en actividades comerciales, fomentando un entorno financiero inclusivo que apoye estrategias de inversión diversificadas y probadas, con lo que se promueve una inversión más segura y el crecimiento económico general.

En resumen, la aplicación de *backtesting* de código abierto se alinea de forma variable con varios ODS, siendo su mayor impacto el fomento del crecimiento económico y la innovación, seguido de las contribuciones a la educación, la cooperación y, en menor medida, la igualdad de género, la reducción de las desigualdades, la salud y la paz. El alcance de su impacto depende en gran medida de su adopción, integración y utilización en diversos contextos.

ANEXO B
Creación de algoritmos de *backtesting*

Existen distintas maneras para crear algoritmos de *backtesting*, pero todas deben seguir los pasos mencionados en el Capítulo 2. En este anexo se tratan algunas de las principales maneras de crear un sistema de *backtesting*.

B.1 Implementación en de un sistema completo en C++

En el Algoritmo 4 podemos encontrar el esquema para la implementación de un sistema simplificado de *backtesting*. Completando este código con datos reales podríamos obtener un sistema básico funcional.

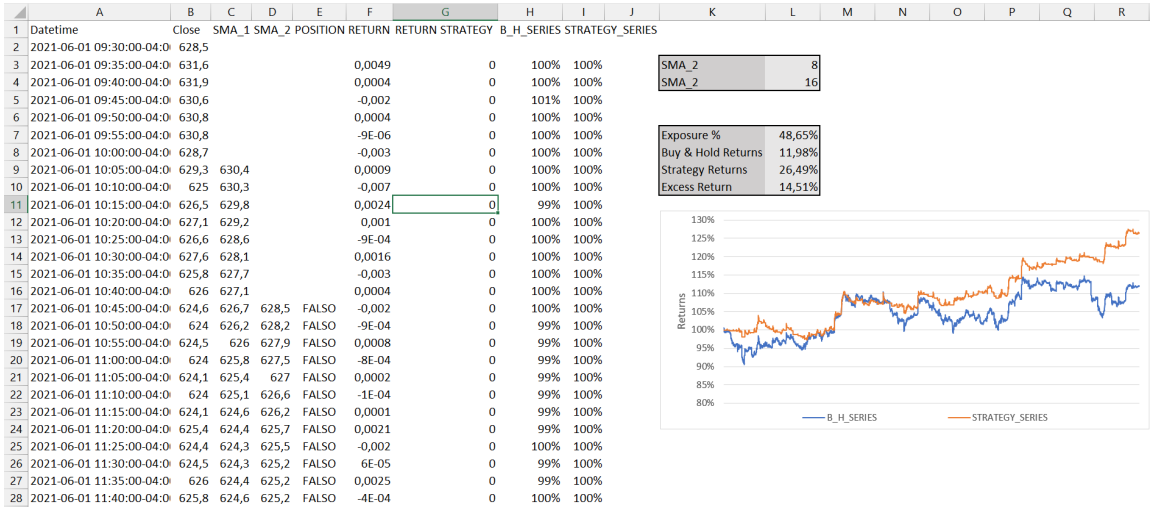
```
class Data {
    // Define la clase para los datos
};

std::vector<Data> getData() {
    // Implementa la obtención de datos históricos.
};

class YourStrategyClass {
private:
    // Guardar los datos históricos para usarlos como
    // referencia.
    std::vector<Data> historicalData;
public:
    YourStrategyClass() {
        // Inicializar los parámatros y la estratégia.
    }

    void processData(const Data& data) {
        // Implementar la lógica para procesar los datos.
    }

    bool shouldEnterTrade() {
        // Implementar la lógica para determinar si comprar o
```

```
        // no.
    }

    bool shouldExitTrade() {
        // Implementar la lógica para determinar si vender o no.
    }
};

int main() {
    // Step 1: Obtener los datos históricos.
    std::vector<Data> historicalData = getData();
    // Step 2: Inicializa tu estrategia.
    YourStrategyClass strategy;
    // Step 3: Procesa los datos históricos y ejecuta las
    // transacciones.
    for (size_t i = 0; i < historicalData.size(); ++i) {
        const Data& currentData = historicalData[i];
        strategy.processData(currentData);
        if (strategy.shouldEnterTrade()) {
            std::cout << "Buy at timestamp: "
                << currentData.timestamp
                << std::endl;
            // Realiza las acciones pertinentes para la compra
        } else if (strategy.shouldExitTrade()) {
            std::cout << "Sell at timestamp: "
                << currentData.timestamp
                << std::endl;
            // Realiza las acciones pertinentes para la venta
        }
    }
    // Step 4: Muestra las métricas de la estrategia.

    return 0;
}
```

Algoritmo 4: Esquema de implementación de un algoritmo de backtesting *básico en C++. Elaboración propia a partir de [12].*

B.2 Implementación de un sistema completo en hoja de cálculo

En la captura de pantalla de la Figura 13 podemos observar el planteamiento de un sistema básico de *backtesting* sobre el valor de las acciones de un activo. Este tipo de implementación es limitada debido a la herramienta sobre la que se realiza, sin embargo, podemos ver que es viable y funcional para usos puntuales.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1	Datetime	Close	SMA_1	SMA_2	POSITION	RETURN	RETURN STRATEGY	B_H_SERIES	STRATEGY_SERIES									
2	2021-06-01 09:30:00-04:0	628,5																
3	2021-06-01 09:35:00-04:0	631,6				0,0049	0	100%	100%		SMA_2	8						
4	2021-06-01 09:40:00-04:0	631,9				0,0004	0	100%	100%		SMA_2	16						
5	2021-06-01 09:45:00-04:0	630,6				-0,002	0	101%	100%									
6	2021-06-01 09:50:00-04:0	630,8				0,0004	0	100%	100%									
7	2021-06-01 09:55:00-04:0	630,8				-9E-06	0	100%	100%		Exposure %	48,65%						
8	2021-06-01 10:00:00-04:0	628,7				-0,003	0	100%	100%		Buy & Hold Returns	11,98%						
9	2021-06-01 10:05:00-04:0	629,3	630,4			0,0009	0	100%	100%		Strategy Returns	26,49%						
10	2021-06-01 10:10:00-04:0	625	630,3			-0,007	0	100%	100%		Excess Return	14,51%						
11	2021-06-01 10:15:00-04:0	626,5	629,8			0,0024	0	99%	100%									
12	2021-06-01 10:20:00-04:0	627,1	629,2			0,001	0	100%	100%									
13	2021-06-01 10:25:00-04:0	626,6	628,6			-9E-04	0	100%	100%									
14	2021-06-01 10:30:00-04:0	627,6	628,1			0,0016	0	100%	100%									
15	2021-06-01 10:35:00-04:0	625,8	627,7			-0,003	0	100%	100%									
16	2021-06-01 10:40:00-04:0	626	627,1			0,0004	0	100%	100%									
17	2021-06-01 10:45:00-04:0	624,6	626,7	628,5	FALSO	-0,002	0	100%	100%									
18	2021-06-01 10:50:00-04:0	624	626,2	628,2	FALSO	-9E-04	0	99%	100%									
19	2021-06-01 10:55:00-04:0	624,5	626	627,9	FALSO	0,0008	0	99%	100%									
20	2021-06-01 11:00:00-04:0	624	625,8	627,5	FALSO	-8E-04	0	99%	100%									
21	2021-06-01 11:05:00-04:0	624,1	625,4	627	FALSO	0,0002	0	99%	100%									
22	2021-06-01 11:10:00-04:0	624	625,1	626,6	FALSO	-1E-04	0	99%	100%									
23	2021-06-01 11:15:00-04:0	624,1	624,6	626,2	FALSO	0,0001	0	99%	100%									
24	2021-06-01 11:20:00-04:0	625,4	624,4	625,7	FALSO	0,0021	0	99%	100%									
25	2021-06-01 11:25:00-04:0	624,4	624,3	625,5	FALSO	-0,002	0	100%	100%									
26	2021-06-01 11:30:00-04:0	624,5	624,3	625,2	FALSO	6E-05	0	99%	100%									
27	2021-06-01 11:35:00-04:0	626	624,4	625,2	FALSO	0,0025	0	99%	100%									
28	2021-06-01 11:40:00-04:0	625,8	624,6	625,2	FALSO	-4E-04	0	100%	100%									

Figura 13: Captura de pantalla de un sistema de backtesting *en Excel. Extraído de [14]*

ANEXO C
Detalles de implementación durante el desarrollo

Este anexo contiene los detalles de implementación que complementan la información expuesta durante el Capítulo 5.

C.1 Contenido de los ficheros de documentación del código abierto

Se exponen los contenidos de los ficheros «CONTRIBUTORS.md» en la Figura 14 y «CODE_OF_CONDUCT.md» en la Figura 15. Estos ficheros deben ser seguidos por cualquier colaborador interesado en participar en el proyecto.

Contributing to the Project

Thank you for your interest in contributing to the project! We appreciate your help in making this project better.

Table of Contents

- Conventional Commits
- Python Development Best Practices
- Submitting a Pull Request
- Code of Conduct

Conventional Commits

We follow the Conventional Commits specification for our commit messages. Please ensure that your commit messages adhere to this format. This helps us maintain a clean and organized commit history.

Python Development Best Practices

When contributing code, please keep the following Python development best practices in mind:

- Write clean and readable code.

- Follow the PEP 8 style guide for Python code.
- Use meaningful variable and function names.
- Write docstrings for functions and classes.
- Include unit tests for new code and ensure all tests pass. It is important to follow the TDD strategy.

Submitting a Pull Request

To contribute to the project, please follow these steps:

1. Fork the repository and create a new branch for your contribution.
2. Make your changes and ensure that the code runs without any errors.
3. Ensure that all tests pass.
4. Commit your changes using the Conventional Commits format.
5. Push your branch to your forked repository.
6. Open a pull request against the main repository's `main` branch.

Code of Conduct

Please note that by contributing to this project, you are expected to follow the project's Code of Conduct. Be respectful and considerate towards others and help create a positive and inclusive community.

We appreciate your contributions and look forward to your pull requests!

Figura 14: Contenido del archivo «CONTRIBUTORS.md».
Elaboración propia

Code of Conduct

Introduction

We, as contributors and maintainers, pledge to make participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

Expected Behavior

We expect all contributors and community members to:

- Be respectful and considerate towards others.
- Use inclusive language and avoid derogatory or offensive comments.
- Be open to constructive feedback and ideas.
- Exercise empathy and understanding towards others' perspectives.
- Be patient and supportive towards new contributors.
- Focus on the project's goals and avoid personal attacks or harassment.

Unacceptable Behavior

The following behaviors are considered unacceptable and will not be tolerated:

- Harassment, discrimination, or any form of offensive behavior.
- Intimidation, threats, or any form of personal attacks.
- Publishing others' private information without their consent.
- Any behavior that creates an unsafe or hostile environment.
- Reporting and Enforcement

If you witness or experience any unacceptable behavior, please report it to the project maintainers at support@strategist.com. All reports will be reviewed and investigated promptly. The project maintainers are committed to maintaining the confidentiality of the reporter's identity.

Anyone found to be engaging in unacceptable behavior may be temporarily or permanently banned from the project's community spaces, as determined by the project maintainers.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html

Figura 15: Contenido del archivo «CODE_OF_CONDUCT.md».
Elaboración propia

C.2 Árbol de ficheros del paquete «strategist»

Este anexo contiene en la Figura 16 el árbol de ficheros del paquete «strategist».

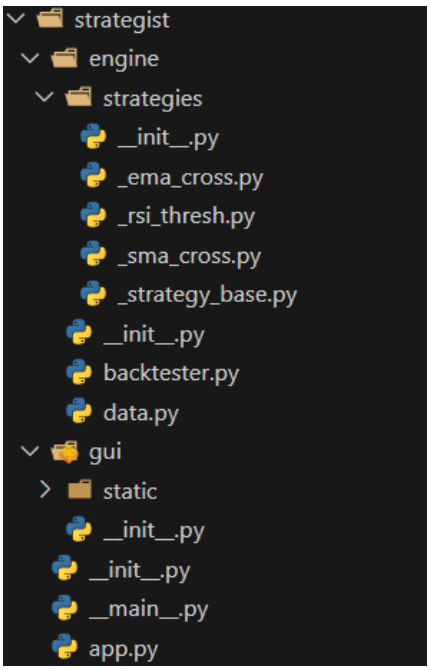


Figura 16: Árbol de ficheros del paquete «strategist»
Elaboración propia