VNIVERSITAT DE VALÈNCIA

Grado en Ingeniería Multimedia

VNIVERSITAT DE VALÈNCIA

Trabajo Fin de Grado

TFG Web App: Plataforma de Simulación de Inversión en Bolsa basada en Análisis Fundamental

Autor: Javier Sanchis Lahiguera

Tutor: Santiago Carlos Ezpeleta Plaza

Enero 2025

VNIVERSITAT DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria ETSE-UV

Trabajo Fin de Grado

TFG Web App: Plataforma de Simulación de Inversión en Bolsa basada en Análisis Fundamental

Autor: Javier Sanchis Lahiguera

Tutor: Santiago Carlos Ezpeleta Plaza

Declaración de autoría:

Yo, Javier Sanchis Lahiguera, declaro la autoría del Trabajo Fin de Grado titulado "TFG Web App: Plataforma de Simulación de Inversión en Bolsa basada en Análisis Fundamental" y que el citado trabajo no infringe las leyes en vigor sobre propiedad intelectual. El material no original que figura en este trabajo ha sido atribuido a sus legítimos autores.

Valencia, 11 de diciembre de 2025

Fdo: Javier Sanchis Lahiguera

Resumen:

La falta de educación financiera es una barrera significativa para muchos inversores particulares, quienes a menudo carecen de herramientas prácticas para experimentar con estrategias de inversión a largo plazo sin arriesgar capital real. Este TFG presenta el diseño y desarrollo de una aplicación web dedicada al análisis fundamental de empresas y la simulación de carteras de inversión.

El sistema, desarrollado en Python con el *framework* Flask, integra datos bursátiles reales a través de la API de Yahoo Finance y ofrece un novedoso "Modo Carrera". Este modo gamifica la experiencia de inversión, permitiendo al usuario gestionar su patrimonio a través de diferentes escenarios históricos y eventos macroeconómicos simulados.

Desde una perspectiva técnica, el proyecto destaca por la implementación de un ciclo de vida de desarrollo de software moderno, integrando pruebas automatizadas (*testing*) y despliegue continuo (CI/CD) mediante GitHub Actions y Render. Los resultados demuestran la viabilidad de crear herramientas financieras educativas robustas y escalables con costes de infraestructura mínimos.

Abstract:

The lack of financial literacy acts as a significant barrier for many retail investors, who often lack practical tools to experiment with long-term investment strategies without risking real capital. This TFG presents the design and development of a web application dedicated to fundamental company analysis and investment portfolio simulation.

The system, developed in Python using the Flask framework, integrates real stock market data via the Yahoo Finance API and features a novel Çareer Mode."This mode gamifies the investment experience, allowing users to manage their wealth across various historical scenarios and simulated macroeconomic events.

From a technical perspective, the project stands out for its implementation of a modern software development lifecycle, integrating automated testing and continuous deployment (CI/CD) using GitHub Actions and Render. The results demonstrate the feasibility of creating robust and scalable educational financial tools with minimal infrastructure costs.

Resum:

La manca déducació financera és una barrera significativa per a molts inversors particulars, els quals sovint manquen déines pràctiques per a experimentar amb estratègies dínversió a llarg termini sense arriscar capital real. Aquest TFG presenta el disseny i desenvolupament dúna aplicació web dedicada a lánàlisi fonamental démpreses i la simulació de carteres dínversió.

El sistema, desenvolupat en Python amb el *framework* Flask, integra dades borsàries reals a través de lÁPI de Yahoo Finance i ofereix un nou "Mode Carrera". Aquest mode ludifica léxperiència dínversió, permetent a lúsuari gestionar el seu patrimoni a través de diferents escenaris històrics i esdeveniments macroeconòmics simulats.

Des dúna perspectiva tècnica, el projecte destaca per la implementació dún cicle de vida de desenvolupament de programari modern, integrant proves automatitzades (*testing*) i desplegament continu (CI/CD) mitjançant GitHub Actions i Render. Els resultats demostren la viabilitat de crear eines financeres educatives robustes i escalables amb costos dínfraestructura mínims.

Agradecimientos:

En primer lugar quiero agradecer a todos aquellos que me han apoyado durante todos estos años.
En segundo lugar...

Índice general

Capítulo 1

Introducción

1.1. Introducción

La inversión en mercados financieros ha ganado popularidad en los últimos años, facilitada por el acceso a plataformas digitales. Sin embargo, la falta de educación financiera lleva a muchos inversores novatos a cometer errores costosos. Según estudios clásicos de Barber y Odean, basados en más de 60.000 hogares estadounidenses, los inversores particulares que operan con mayor frecuencia obtienen rentabilidades anuales entre 5 y 10 puntos porcentuales inferiores a las de un índice de mercado amplio, principalmente debido a sesgos conductuales como el exceso de confianza y a una operativa excesiva [1].

Este TFG presenta el diseño e implementación de una aplicación web orientada al análisis fundamental de empresas y la simulación de estrategias de inversión a largo plazo. La herramienta, denominada "TFG Web App", permite a los usuarios practicar estrategias como el *Dollar Cost Averaging* (DCA) en un entorno libre de riesgo, utilizando datos históricos reales del mercado obtenidos a través de Yahoo Finance [2].

A diferencia de los simuladores de *trading* tradicionales, que suelen fomentar la especulación a corto plazo, esta propuesta se centra en la inversión sosegada y el análisis de métricas fundamentales (PER, deuda, crecimiento), ofreciendo un "Modo Carrera"gamificado para evaluar el desempeño del usuario ante distintos escenarios macroeconómicos.

1.2. Estructura de la memoria

El resto de este documento se estructura de la siguiente manera:

- En el **Capítulo 2** se detallan la motivación del proyecto y los objetivos principales y específicos que se pretenden alcanzar.
- El **Capítulo 3** revisa el estado del arte, analizando las herramientas existentes y justificando las tecnologías seleccionadas (Python, Flask, CI/CD).

- El **Capítulo 4** aborda la planificación del proyecto, incluyendo la especificación de requisitos y la estimación de costes.
- El **Capítulo 5** describe el desarrollo del sistema, desde el análisis y diseño de la arquitectura hasta los detalles de implementación.
- El **Capítulo 6** presenta las pruebas realizadas y los resultados obtenidos.
- Finalmente, el **Capítulo 7** expone las conclusiones y las líneas de trabajo futuro.

Capítulo 2

Motivación y Objetivos

2.1. Motivación

La inversión en mercados financieros requiere no solo conocimientos teóricos, sino también experiencia práctica para gestionar la incertidumbre y la psicología del inversor. Actualmente, muchas herramientas de simulación bursátil se centran excesivamente en el *trading* intradía o especulativo, dejando de lado estrategias de inversión a largo plazo como el *Value Investing* [3] o el *Dollar Cost Averaging* (DCA).

La motivación principal de este proyecto surge de la necesidad de proporcionar una herramienta tecnológica que cubra este vacío. Se busca desarrollar una plataforma que no solo permita consultar datos, sino que "gamifique"la experiencia de inversión a largo plazo mediante un "Modo Carrera". Este enfoque de gamificación [4] expondrá al usuario a eventos macroeconómicos simulados (crisis de deuda, inflación, burbujas sectoriales), obligándole a tomar decisiones estratégicas en un entorno controlado pero basado en datos reales.

Desde el punto de vista de la Ingeniería Multimedia, este proyecto está motivado por el interés en aplicar tanto principios de diseño de experiencias interactivas como buenas prácticas de desarrollo web moderno [5]. No basta con que la aplicación “funcione”; se busca ofrecer una interfaz clara, usable y visualmente coherente para el usuario [6], al tiempo que se implementa un ciclo de vida de desarrollo profesional, integrando pruebas automatizadas y despliegue continuo (CI/CD) que garanticen la robustez, el rendimiento y la mantenibilidad del sistema.

2.2. Objetivos

A continuación se detalla el objetivo principal del proyecto y se desglosa en subobjetivos específicos y verificables.

2.2.1. Objetivo Principal

El objetivo principal de este TFG es el diseño, desarrollo e implementación de una aplicación web robusta para el análisis fundamental de empresas y la simulación de carteras de inversión, integrando datos de mercado reales y mecanismos de validación automática de código.

2.2.2. Objetivos Específicos

Para alcanzar la meta principal, se han definido los siguientes subobjetivos:

- **Integración y automatización de datos financieros:** Desarrollar módulos capaces de extraer, normalizar y procesar datos históricos de precios y dividendos provenientes de APIs externas (Yahoo Finance [2]), gestionando la ausencia de datos o inconsistencias temporales.
- **Desarrollo de un motor de simulación ("Modo Carrera"):** Implementar la lógica de negocio necesaria para simular escenarios de inversión a largo plazo, incluyendo la generación aleatoria de eventos macroeconómicos (shocks de mercado, noticias sectoriales) y el cálculo de métricas de rendimiento (CAGR, *drawdown*, volatilidad).
- **Implementación de Calidad de Software (QA):** Configurar un entorno de Integración Continua (CI) utilizando GitHub Actions [7] que ejecute automáticamente baterías de pruebas unitarias (con `pytest` [8]) y análisis estático de código (linter) en cada modificación del repositorio.
- **Diseño de Interfaz de Usuario (UI):** Construir una interfaz web intuitiva basada en el *framework* Flask [9] que permita al usuario interactuar con las herramientas de análisis y visualizar la evolución de su patrimonio simulado mediante gráficos dinámicos.

Capítulo 3

Estado del Arte

3.1. Análisis de soluciones existentes

En el mercado actual existen numerosas herramientas de seguimiento de carteras y simulación bursátil. Sin embargo, muchas de ellas se centran en el *trading* especulativo a corto plazo o bien carecen de un enfoque claramente pedagógico orientado a la inversión a largo plazo y a la adquisición de hábitos de inversión saludables.

3.1.1. Investopedia Simulator

Investopedia Simulator [10] es una de las herramientas más conocidas para la simulación bursátil. Permite operar con dinero virtual en tiempo casi real, utilizando órdenes de compra y venta similares a las de un bróker tradicional. No obstante, su enfoque es fundamentalmente transaccional: el usuario practica la ejecución de operaciones, pero no existe un modo estructurado que simule el paso de los años, la aparición de eventos macroeconómicos ni un hilo conductor que le obligue a reflexionar sobre decisiones de largo plazo.

3.1.2. Simuladores bancarios (p. ej., La Bolsa Virtual)

Distintas entidades financieras y plataformas nacionales ofrecen simuladores que replican la interfaz de su propio bróker, como es el caso de La Bolsa Virtual [11] u otros simuladores ligados a cuentas demo. Estos productos suelen ser robustos en cuanto a datos de mercado, pero su componente pedagógico es limitado: en general no profundizan en métricas fundamentales (PER, deuda, crecimiento, etc.) ni guían al usuario en la construcción de una estrategia de inversión sostenida en el tiempo, sino que se centran en la compra–venta en función del precio.

3.1.3. Diferenciación de la propuesta

La aplicación desarrollada en este TFG (*TFG Web App*) se diferencia de estas soluciones en varios aspectos. En primer lugar, integra un *Modo Carrera* que simula el paso del tiempo y la aparición de eventos macroeconómicos predefinidos (crisis, periodos de bonanza, cambios de tipos de interés), de manera que el usuario debe ir adaptando su asignación de capital turno a turno. En segundo lugar, el foco está en la inversión a largo plazo y en el análisis fundamental de empresas, utilizando métricas básicas y estrategias como el *Dollar Cost Averaging*, en lugar de fomentar la operativa especulativa de muy corto plazo. Finalmente, la interfaz y las mecánicas están diseñadas con un enfoque gamificado y didáctico, buscando que el usuario aprenda mientras experimenta con escenarios realistas pero sin riesgo.

3.2. Análisis tecnológico

Para el desarrollo de la aplicación se han evaluado distintas alternativas tecnológicas, seleccionando aquellas que mejor se adaptan a los requisitos del proyecto: simplicidad de despliegue, facilidad de mantenimiento y adecuación a un entorno de prototipado típico de un Trabajo de Fin de Grado.

3.2.1. Backend: Python y Flask

Se ha seleccionado **Python** [12] como lenguaje principal por su amplio uso en el ámbito financiero y de ciencia de datos, así como por la disponibilidad de librerías maduras para el tratamiento de series temporales. En particular, el uso de `pandas` [13] y `yfinance` [2] simplifica la descarga, limpieza y agregación de precios y dividendos históricos.

Como *framework* web se ha utilizado **Flask** [9], por tratarse de un micro-*framework* ligero y flexible que encaja bien con el tamaño y alcance del proyecto. Flask permite definir de forma sencilla las rutas de la aplicación, exponer endpoints para la obtención de datos en formato JSON y servir las plantillas HTML que conforman la interfaz de usuario.

3.2.2. Integración Continua (CI): GitHub Actions

Para asegurar un mínimo de calidad del software se ha configurado un flujo de **Integración Continua** mediante **GitHub Actions** [7]. En cada *push* al repositorio se ejecutan automáticamente las pruebas unitarias definidas con `pytest` [8] y herramientas de análisis estático como `ruff` [14]. Este proceso permite detectar errores de forma temprana y reduce la probabilidad de introducir regresiones en la lógica de cálculo de carteras y métricas de rendimiento.

3.2.3. Despliegue y alojamiento (CD): Render

La puesta en producción de la aplicación se realiza en la plataforma en la nube **Render** [15], que ofrece un modelo de tipo PaaS (*Platform as a Service*). Render se ha elegido principalmente por su integración directa con GitHub, porque ofrece una versión gratuita y por la sencillez de configuración: una vez definido el servicio, la plataforma detecta automáticamente los cambios en la rama correspondiente, construye la imagen de la aplicación y la despliega, implementando así un flujo básico de **Despliegue Continuo (CD)**.

Frente a alternativas más complejas como AWS o Azure, Render resulta más apropiado para un proyecto académico, ya que minimiza las tareas de administración de infraestructura y proporciona de serie aspectos como el certificado SSL y el acceso mediante HTTPS.

3.2.4. Fuente de datos: Yahoo Finance

Como fuente principal de datos de mercado se utiliza la API (no oficial) de **Yahoo Finance**, accedida a través de la librería `yfinance` [2]. Esta opción se ha elegido por ser gratuita, ofrecer datos ajustados por dividendos y disponer de una cobertura razonablemente amplia de *tickers* internacionales. Aunque existen alternativas profesionales más completas (Bloomberg, Refinitiv, etc.), su coste y complejidad las hacen poco viables en el contexto de un TFG. La elección de Yahoo Finance supone un compromiso adecuado entre disponibilidad de datos y simplicidad de integración para el alcance del proyecto.

Capítulo 4

Planificación y especificación

4.1. Identificación de Requisitos

En este apartado se recogen los requisitos principales que debe cumplir la aplicación desarrollada. Se distinguen, por un lado, los requisitos funcionales asociados al comportamiento observable por el usuario y, por otro, los requisitos no funcionales relacionados con la calidad, el despliegue y la experiencia de uso.

4.1.1. Requisitos Funcionales

El sistema deberá permitir, como mínimo, las siguientes funcionalidades:

- **RF-01 Gestión de sesiones:** El sistema gestionará el estado del usuario mediante sesiones anónimas (basadas en cookies y un *session_id*) para el *Modo Carrera*, permitiendo seguir el progreso de cada partida sin necesidad de un registro persistente con usuario y contraseña.
- **RF-02 Visualización de rentabilidad:** La aplicación generará gráficos dinámicos que comparen la rentabilidad porcentual de los activos individuales y de la cartera total frente a un *benchmark* de referencia (por ejemplo, el S&P 500), de forma que el usuario pueda interpretar fácilmente la evolución de su estrategia.
- **RF-03 Simulación (*Modo Carrera*):** El sistema permitirá al usuario gestionar una cartera virtual a lo largo de periodos históricos, tomando decisiones de compra, venta o rebalanceo en cada turno. Cada decisión afectará a la evolución futura del patrimonio simulado.
- **RF-04 Eventos aleatorios:** Se generarán eventos macroeconómicos (noticias, crisis, inflación, cambios de tipos, etc.) que impacten en la valoración de la cartera, con el objetivo de evaluar cómo reacciona el inversor ante cambios de ciclo y situaciones de estrés de mercado.

4.1.2. Requisitos No Funcionales

Además de las funcionalidades anteriores, la aplicación debe cumplir una serie de requisitos no funcionales:

- **RNF-01 Disponibilidad y despliegue:** La aplicación estará desplegada en la nube mediante la plataforma **Render** [15], utilizando el nivel gratuito. Debido a las limitaciones de este plan, el servicio puede entrar en suspensión tras un periodo de inactividad, requiriendo un breve tiempo de arranque (*cold start*) en la siguiente petición.
- **RNF-02 Calidad del código:** El desarrollo seguirá el estándar de estilo **PEP 8** [16], verificado mediante la herramienta `ruff` [14], y contará con una estrategia de pruebas automatizadas implementada con `pytest` [8], para asegurar la fiabilidad de la lógica financiera y reducir la aparición de regresiones.
- **RNF-03 Usabilidad y diseño *responsive*:** La interfaz web debe ser adaptable (*responsive*), garantizando una visualización correcta tanto en monitores de escritorio como en dispositivos móviles, y manteniendo una organización clara de los elementos de análisis y simulación.

4.2. Metodología de Trabajo

A lo largo del desarrollo se ha seguido una metodología de trabajo ágil e incremental, organizando el proyecto en pequeños entregables que se iban validando de forma continua. El ciclo de desarrollo se ha apoyado en las siguientes prácticas:

- **Control de versiones:** Se ha utilizado **Git** [17] como sistema de control de versiones, trabajando con ramas específicas (*feature branches*) para cada nueva funcionalidad o corrección. Esto ha facilitado el aislamiento de cambios y la integración progresiva en la rama principal del proyecto.
- **Diario de desarrollo:** Se ha mantenido un diario de desarrollo donde se registran los avances, los problemas encontrados y las decisiones técnicas adoptadas. Este diario ha servido como herramienta de autoevaluación y también como apoyo para la planificación de los siguientes pasos.

4.3. Planificación Temporal

El desarrollo del proyecto se ha estructurado en cuatro hitos principales:

1. **Investigación y configuración inicial:** Búsqueda y evaluación de posibles fuentes de datos financieros, selección de la API (Yahoo Finance [2]) y puesta en marcha del entorno de desarrollo, incluyendo la configuración básica de CI/CD [7].

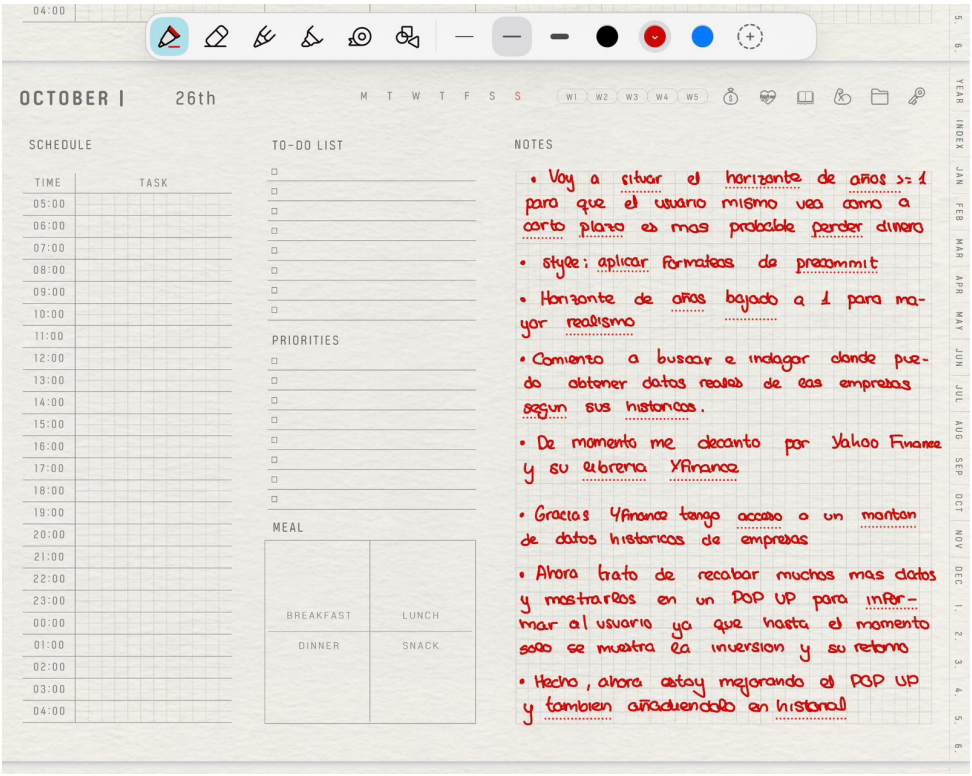

Figura 4.1: Ejemplo del diario de desarrollo utilizado durante el proyecto.

2. **Desarrollo del *backend*:** Implementación de la lógica financiera (descarga y procesado de datos, cálculo de métricas) y del motor de simulación del *Modo Carrera* en Python [12], así como la definición de los endpoints de la API interna.

3. **Desarrollo del *frontend*:** Diseño e implementación de las plantillas Jinja2 [18] y de los componentes visuales de la aplicación, incluyendo los gráficos interactivos con Chart.js [19] y las vistas de empresas, análisis y modo carrera.

4. **Despliegue y documentación:** Configuración del servicio en Render, pruebas finales en el entorno desplegado y elaboración de la memoria del TFG y del manual de usuario asociado a la *TFG Web App*.

4.4. Estimación de Costes

Aunque se trata de un proyecto académico, es posible realizar una estimación aproximada de costes considerando el tiempo de desarrollo y los recursos materiales utilizados. La Tabla 4.1 recoge un presupuesto orientativo.

Concepto	**Unidades**	**Coste unitario**	**Total**
Ingeniero Junior	300 horas	20 €/h	6.000 €
Amortización portátil	4 meses	20 €/mes	80 €
Servidor Render	4 meses	0 € (Free Tier)	0 €
TOTAL			**6.080 €**

Cuadro 4.1: Presupuesto estimativo del proyecto.

Capítulo 5

Desarrollo del Proyecto

5.1. Análisis del Sistema

El análisis se centra en comprender la interacción entre el usuario y el sistema para cumplir con los requisitos establecidos. Al tratarse de una aplicación de acceso libre sin persistencia de usuarios en base de datos, el modelo de interacción se simplifica, priorizando la inmediatez de uso.

5.1.1. Actores del Sistema

Se identifica un único actor principal:

- **Usuario Inversor:** Persona interesada en practicar estrategias de inversión o consultar datos fundamentales. No requiere privilegios administrativos ni registro previo.

5.1.2. Casos de Uso

Los principales casos de uso identificados se describen a continuación:

- **CU-01 Consultar Empresa:** El usuario introduce un *ticker* (ej. AAPL) y el sistema devuelve su cotización actual, datos fundamentales y gráficos históricos.
- **CU-02 Iniciar Simulación (Modo Carrera):** El usuario configura una nueva partida definiendo capital inicial, dificultad y horizonte temporal.
- **CU-03 Avanzar Turno:** Dentro de la simulación, el usuario decide qué activos comprar o vender y solicita avanzar al siguiente periodo. El sistema calcula los rendimientos y genera eventos aleatorios.
- **CU-04 Comparar con Benchmark:** El sistema superpone la rentabilidad de la cartera del usuario con la del índice S&P 500.

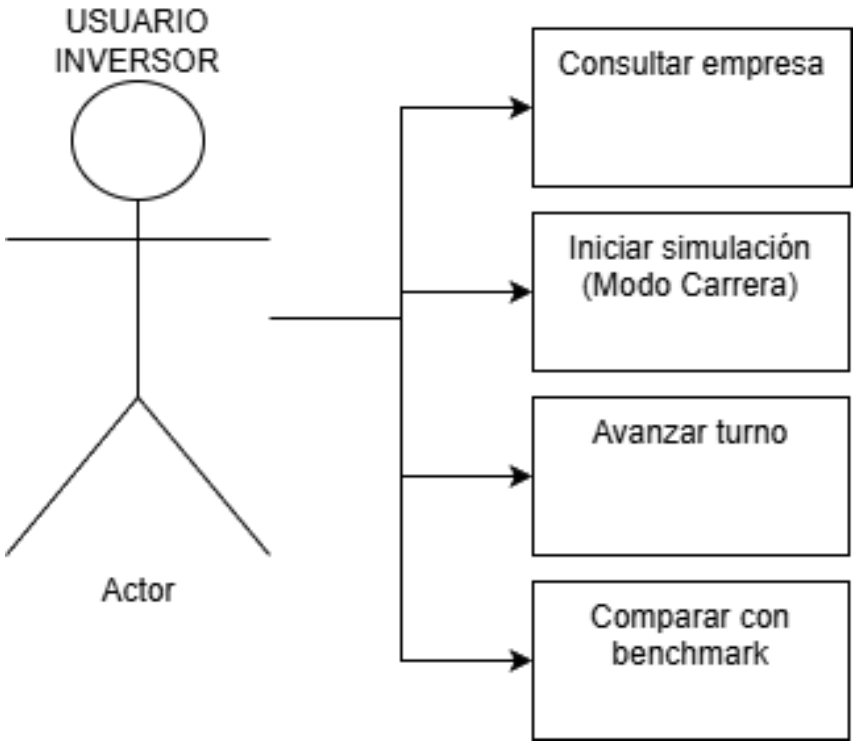

Figura 5.1: Diagrama de Casos de Uso del sistema.

5.2. Diseño del Sistema

Para satisfacer los requisitos funcionales y no funcionales, se ha optado por una arquitectura web basada en el patrón **Modelo-Vista-Controlador (MVC)** [20], adaptada al *microframework* Flask.

5.2.1. Arquitectura de Software

La aplicación se estructura en capas lógicas claramente diferenciadas para facilitar el mantenimiento y la escalabilidad:

- **Capa de Presentación (Vista):** Responsable de la interfaz de usuario. Implementada mediante plantillas HTML5 renderizadas con **Jinja2** [18], estilos CSS3 personalizados (diseño *responsive*) y gráficos interactivos generados con la librería **Chart.js** [19].
- **Capa de Control (Controlador):** Gestiona la lógica de la aplicación y el enrutamiento HTTP. Se ha implementado en Python utilizando **Flask Blueprints** [9] para modularizar el código:
  - `routes.py`: Maneja las rutas generales y la consulta de datos públicos.
  - `career.py`: Encapsula toda la lógica compleja del simulador ("Modo Carrera"), incluyendo la máquina de estados de la simulación y el generador de eventos.
- **Capa de Datos y Servicios (Modelo):**

- **Proveedor de Datos:** Integración con la API de **Yahoo Finance** (vía librería `yfinance`) para la obtención de series temporales en tiempo real.
- **Almacenamiento:** Debido a la naturaleza anónima de la aplicación, la persistencia es efímera o basada en archivos JSON estáticos (`empresas.json`) para catálogos, delegando el estado de la sesión al navegador del cliente o memoria temporal.

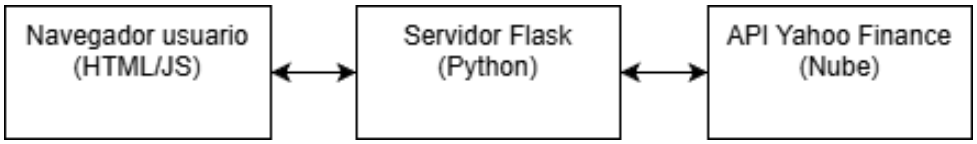

Figura 5.2: Arquitectura de alto nivel de la aplicación.

5.2.2. Diseño de la Interfaz (UI/UX)

El diseño visual prioriza la claridad de los datos financieros. Se ha implementado un sistema de *grids* CSS y *Media Queries* para asegurar que las tablas de datos y los gráficos se adapten fluidamente a pantallas móviles, cumpliendo el requisito RNF-03. Los gráficos de Chart.js se han configurado para ser reactivos al redimensionamiento de la ventana.

5.3. Implementación

La implementación del sistema se ha llevado a cabo siguiendo las especificaciones de diseño, priorizando la modularidad y el uso de librerías estándar de la industria. A continuación, se detallan los componentes más relevantes del desarrollo.

5.3.1. Motor de Simulación Financiera

El núcleo de la aplicación reside en el módulo `career.py`, encargado de gestionar la lógica del "Modo Carrera". Este módulo implementa una máquina de estados que procesa los turnos de simulación.

Para garantizar la consistencia de los datos históricos, se ha implementado una función robusta de extracción de datos que gestiona la conexión con la API de Yahoo Finance. El siguiente fragmento muestra cómo se procesan y normalizan los precios ajustados para evitar errores en el cálculo de rentabilidades a largo plazo:

Listado 5.1: Extracto de la función de normalización de precios en career.py

```
def _build_normalized_series_map(tickers, start, end):
    series_map = {}
    # Separamos activos normales de la liquidez (CASH)
    non_cash = [t for t in tickers if not _is_cash(t)]
```

```
    for ticker in non_cash:
        # Descarga y normalizacion base 100
        data = _fetch_adj_close(ticker, start, end)
        if data:
            series_map[ticker] = _normalize_base100(data)

    return series_map
```

El sistema normaliza todas las series temporales a una "Base 100.ªl inicio del periodo de inversión, lo que facilita la comparación visual entre activos de precios muy dispares (por ejemplo, comparar una acción de 150$ con una de 2.000$).

5.3.2. Gestión de Eventos Aleatorios

Uno de los requisitos funcionales clave (RF-04) es la generación de eventos que alteren el mercado. Se ha implementado un sistema de "baraja de cartas"ponderada según la dificultad elegida por el usuario.

El algoritmo decide en cada turno si ocurre un evento (positivo o negativo) basándose en probabilidades predefinidas. La estructura de datos para un evento típico se define en un diccionario extensible:

Listado 5.2: Definición de un evento macroeconómico negativo

```
{
    "id": "macro_inflation_shock",
    "name": "Shock inflacionario inesperado",
    "scope": "portfolio", # Afecta a toda la cartera
    "impact_pct": -0.085, # Impacto del -8.5%
    "remaining_turns": 3, # Duracion del efecto
    "difficulty_label": "hard"
}
```

5.3.3. Controlador y Rutas (API)

El archivo `routes.py` actúa como el controlador principal, exponiendo los puntos de entrada (*endpoints*) de la aplicación. Se ha utilizado el patrón *Blueprint* de Flask para organizar las rutas.

Destaca la implementación del *endpoint* de análisis, que realiza validaciones de entrada estrictas antes de procesar cualquier simulación, asegurando que parámetros como el "horizonte temporal.º el .ªporte mensual"(DCA) sean coherentes:

Listado 5.3: Validación de parámetros en el controlador

```
@bp.post("/analisis")
def crear_analisis():
    datos_brutos = request.get_json(silent=True) or {}
    # Normalizacion y saneamiento de entrada
```

```
    datos = _normalizar_payload(datos_brutos)
    # Validacion de reglas de negocio
    errores = _validar_payload(datos)

    if errores:
        return jsonify({"valido": False, "errores": errores}), 400

    return _registrar_analisis(datos)
```

5.3.4. Interfaz de Usuario (Frontend)

La interfaz se ha construido utilizando HTML5 semántico y CSS3 moderno. Para la visualización de datos, se ha integrado la librería **Chart.js** [19], que renderiza los gráficos financieros en el lado del cliente (navegador) a partir de los datos JSON servidos por el backend.

Para cumplir con el requisito de diseño responsivo (RNF-03), se han utilizado *Media Queries* y unidades relativas (como `clamp()`), permitiendo que la navegación y las tablas de datos se adapten fluidamente a dispositivos móviles sin perder legibilidad.

Capítulo 6

Pruebas y Resultados

6.1. Diseño de Pruebas

Para garantizar la fiabilidad del sistema, se ha seguido una estrategia de pruebas mixta que combina la verificación automática del código (Caja Blanca) con la validación funcional de la interfaz de usuario (Caja Negra).

6.1.1. Pruebas Automáticas (QA)

El aseguramiento de la calidad se ha integrado en el ciclo de desarrollo mediante herramientas de integración continua.

Análisis Estático y Estilo

Se ha utilizado la herramienta `ruff` [14] para auditar el cumplimiento del estándar PEP8 [16] y detectar errores de sintaxis o patrones peligrosos antes de la ejecución. Esto garantiza un código base limpio y mantenible.

Pruebas Unitarias

Se ha implementado una batería de pruebas unitarias utilizando el *framework* **pytest** [8]. Estas pruebas verifican la lógica financiera crítica de forma aislada, asegurando que funciones como el cálculo del interés compuesto o la normalización de precios devuelvan resultados matemáticamente exactos.

Integración Continua (CI)

Cada vez que se realiza un envío de código (*push*) al repositorio, un flujo de trabajo de **GitHub Actions** [7] levanta un entorno virtual, instala las dependencias y ejecuta la

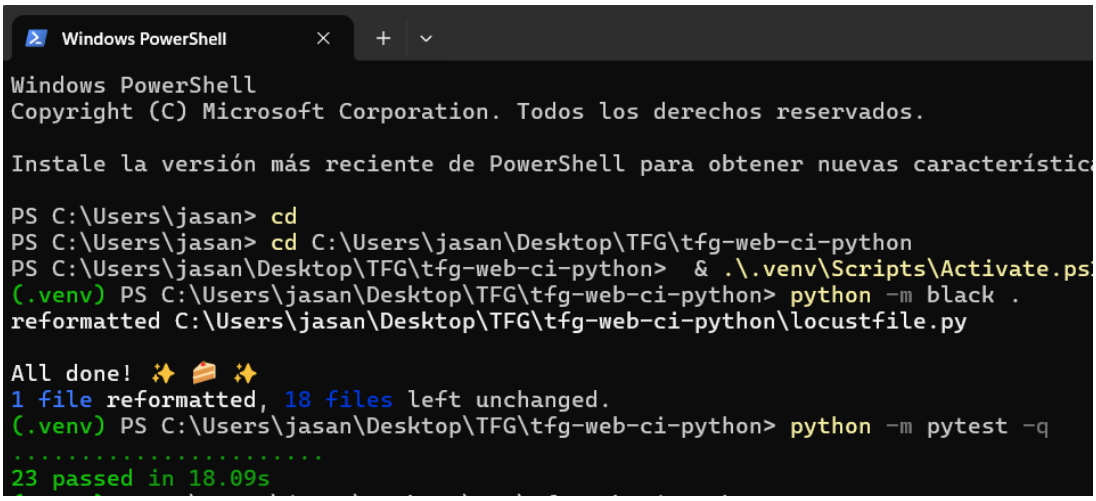

Figura 6.1: Resultado de la ejecución exitosa de la batería de pruebas automatizadas.

totalidad de las pruebas. Si alguna prueba falla, el despliegue se bloquea automáticamente, impidiendo que llegue código defectuoso a producción.

6.2. Pruebas Funcionales y Resultados

Se han realizado pruebas manuales para validar los requisitos funcionales definidos en el Capítulo 4, verificando la correcta integración entre el *frontend*, el *backend* y la API externa.

6.2.1. Validación de la Interfaz de Usuario

La interfaz ha demostrado ser intuitiva y robusta. A continuación se muestran los resultados finales de las pantallas principales:

6.2.2. Adaptabilidad (Responsive Design)

Se ha verificado el correcto funcionamiento en dispositivos móviles, cumpliendo el requisito RNF-03. Los menús de navegación colapsan correctamente y los gráficos de `Chart.js` [19] se redimensionan sin perder legibilidad.

6.2.3. Pruebas de Rendimiento y Estabilidad en Producción

Durante la fase de despliegue y validación en el entorno de producción (Render, nivel gratuito), se detectó una incidencia crítica relacionada con la gestión de recursos. Al iniciar sesiones de simulación con horizontes temporales extensos (superiores a 10 años) o generar informes complejos, la aplicación sufría terminaciones abruptas del servicio.

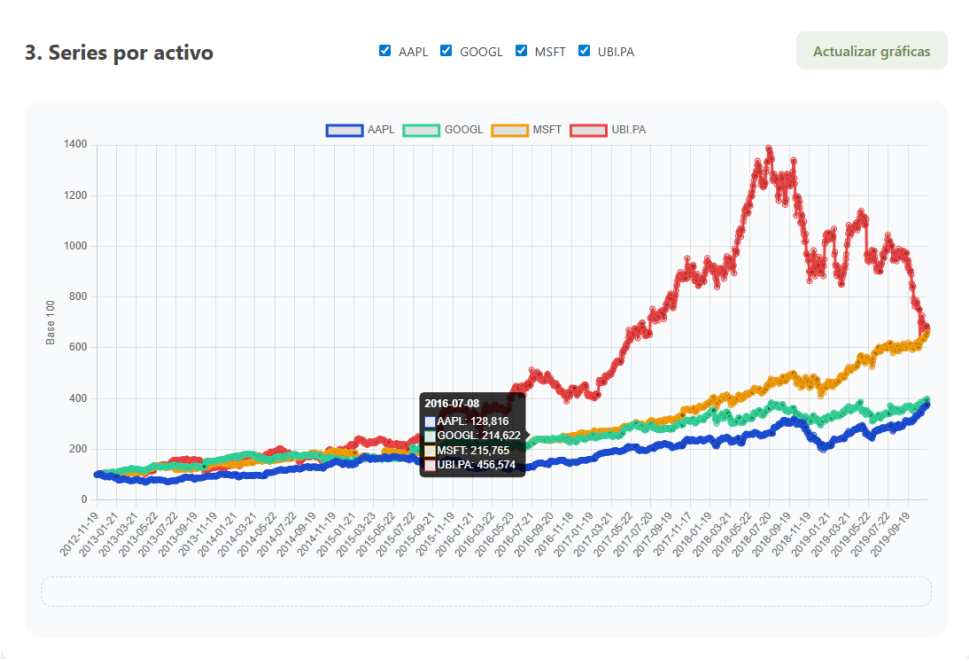

Figura 6.2: Visualización comparativa de rentabilidad de los activos en cartera.

El análisis de los registros del servidor (*logs*) reveló errores de tipo `SIGKILL` y `OOM (Out of Memory)`, indicando que el proceso de la aplicación excedía el límite estricto de 512 MB de memoria RAM impuesto por el contenedor. Adicionalmente, se observaron errores de *timeout* (tiempo de espera agotado) en el servidor Gunicorn al realizar cálculos combinatorios intensivos para el informe final.

Para solucionar estos problemas y garantizar la estabilidad del sistema, se implementaron las siguientes medidas correctivas en el módulo `career.py`:

- **Optimización de memoria caché:** Se sustituyó el almacenamiento global de series históricas (que provocaba fugas de memoria) por el uso del decorador `@lru_cache` de la librería estándar, limitando el almacenamiento en memoria a las últimas 32 consultas y forzando la liberación de recursos no utilizados.
- **Reducción de precisión de datos:** Se implementó la conversión de los *DataFrames* de `pandas` al tipo de dato `float32`, reduciendo el peso de las series temporales en memoria en aproximadamente un 50 % sin comprometer la precisión necesaria para la simulación.
- **Acotación de la complejidad computacional:** Se ajustó el algoritmo de generación de carteras teóricas, reduciendo el límite de combinaciones a evaluar (de 30 a 15 candidatos) y forzando la recolección de basura (`gc.collect()`) tras procesos intensivos de descarga de datos.
- **Ajuste de parámetros de dificultad:** Se recalibraron los rangos de años permitidos en la configuración de dificultad para alinearlos con la capacidad de procesamiento del plan gratuito.

Tras aplicar estas optimizaciones, se verificó mediante nuevas pruebas de estrés que el

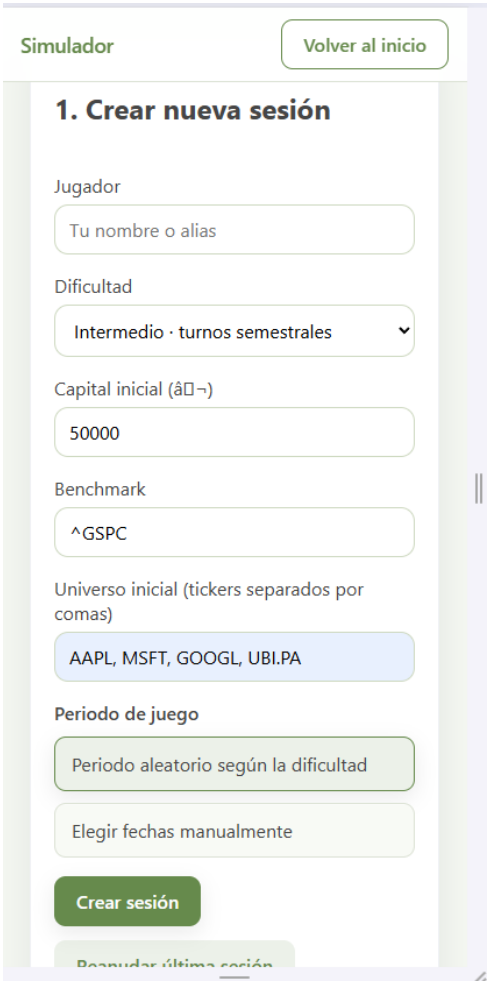

Figura 6.3: Adaptación de la interfaz a dispositivos móviles.

consumo de memoria se mantiene estable por debajo del umbral crítico, permitiendo la ejecución fluida del ciclo completo de simulación.

6.3. Revisión Presupuestaria

Tras la finalización del desarrollo, se procede a comparar los costes estimados en la fase de planificación con los costes reales incurridos.

Debido a la naturaleza académica del proyecto y al uso de un modelo de desarrollo ágil personal, no se han producido desviaciones significativas en los costes directos. El uso del plan gratuito de Render [15] ha permitido mantener el coste de infraestructura en 0 €, tal y como se planificó.

Se concluye que el proyecto ha sido económicamente viable y se ha ejecutado dentro de los márgenes presupuestarios establecidos.

Concepto	**Estimado**	**Real**	**Desviación**
Recursos Humanos	6.000 €	6.000 €	0 %
Hardware	80 €	80 €	0 %
Infraestructura Cloud	0 €	0 €	0 %
TOTAL	**6.080 €**	**6.080 €**	**0 %**

Cuadro 6.1: Comparativa de costes estimados vs. reales.

Capítulo 7

Conclusiones y Trabajo Futuro

7.1. Conclusiones

El desarrollo de este Trabajo de Fin de Grado ha permitido alcanzar satisfactoriamente el objetivo principal: el diseño e implementación de una plataforma web funcional para el análisis y simulación de inversiones financieras.

Tras completar el ciclo de desarrollo, se extraen las siguientes conclusiones principales:

- **Cumplimiento de Objetivos:** Se ha logrado automatizar la extracción de datos financieros reales y construir un "Modo Carrera"jugable que cumple con la función pedagógica planteada. La aplicación permite simular estrategias como el DCA en periodos históricos reales, ofreciendo un valor diferencial respecto a otros simuladores estáticos.

- **Robustez Técnica:** La integración de prácticas de Ingeniería de Software modernas, específicamente el pipeline de CI/CD con GitHub Actions [7] y el despliegue automático en Render [15], ha sido determinante. Esto ha permitido detectar errores de regresión de forma temprana y asegurar que la versión en producción siempre sea estable.

- **Viabilidad Económica:** La revisión presupuestaria confirma que el desarrollo de herramientas complejas es viable con costes de infraestructura nulos o muy reducidos gracias a las capas gratuitas de los proveedores Cloud actuales, lo que democratiza el acceso a la tecnología financiera.

- **Experiencia de Usuario:** Las pruebas funcionales han validado que la interfaz es capaz de presentar gran cantidad de datos (gráficos, tablas) de forma legible en dispositivos móviles, superando uno de los retos de diseño más habituales en aplicaciones financieras.

7.2. Trabajo Futuro

A pesar de que el sistema es funcional y cumple los requisitos iniciales, el desarrollo ha abierto nuevas vías de mejora y expansión. Si se dispusiera de más tiempo y recursos, se proponen las siguientes líneas de trabajo futuro:

7.2.1. Mejoras Técnicas

- **Persistencia de Usuarios:** Implementar una base de datos relacional como PostgreSQL [21] para permitir el registro de usuarios, guardar historiales de partidas y crear un ranking global de jugadores.
- **Ampliación de Mercados:** Integrar APIs adicionales para incluir criptomonedas, materias primas o mercados europeos, ya que actualmente la herramienta se centra en el mercado estadounidense (S&P 500).

7.2.2. Mejoras Funcionales

- **Inteligencia Artificial:** Desarrollar un módulo de IA que actúe como .ªsesor virtual", analizando la cartera del usuario y sugiriendo movimientos basados en patrones históricos o métricas de riesgo.
- **Modo Multijugador:** Implementar competiciones en tiempo real donde varios usuarios gestionen carteras simultáneamente bajo las mismas condiciones de mercado.

7.3. Conclusión Personal

La realización de este proyecto ha supuesto un reto integrador que ha permitido consolidar competencias adquiridas durante el grado, desde la programación *backend* con Python [12] hasta la gestión de proyectos ágiles. Más allá del resultado técnico, el proyecto ha servido para profundizar en la intersección entre la tecnología y las finanzas, un sector con gran proyección profesional.

Apéndice A

Manual Técnico y Despliegue

A.1. Estructura del Proyecto

El código fuente de la aplicación se organiza siguiendo el patrón de diseño de Flask [9]. A continuación se muestra la estructura de directorios principal y la función de cada módulo:

```
/tfg-web-ci-python
|-- .github/            # Flujos de CI/CD (GitHub Actions)
|-- app/
|   |-- data/           # Datos estáticos (JSON)
|   |-- static/         # CSS, JS e imágenes
|   |-- templates/      # Plantillas HTML (Jinja2)
|   |-- __init__.py     # Factoría de la aplicación
|   |-- career.py       # Lógica del "Modo Carrera"
|   `-- routes.py       # Controlador (Rutas)
|-- tests/              # Tests unitarios (pytest)
|-- requirements.txt    # Dependencias
`-- run.py              # Entry point
```

A.2. Instalación Local

Para ejecutar el entorno de desarrollo se requieren Python 3.11+ [12] y Git [17].

1. Clonar el repositorio:

   ```
   git clone https://github.com/sanlaja/tfg-web-ci-python.git
   ```

2. Crear entorno virtual e instalar dependencias:

```
python -m venv .venv
source .venv/bin/activate  # Windows: .venv\Scripts\activate
pip install -r requirements.txt
```

3. Ejecutar la aplicación:

```
python run.py
```

Si la ejecución es correcta, la aplicación estará disponible localmente en:

`http://localhost:5000`

Para acceder a la versión de producción desplegada en la nube (Render [15]), se puede visitar:

`https://tfg-web-ci-python.onrender.com`

Bibliografía

[1] Brad M. Barber and Terrance Odean. Trading is hazardous to your wealth: The common stock investment performance of individual investors. *The Journal of Finance*, 55(2):773–806, 2000.

[2] Ran Aroussi. yfinance: Yahoo! finance market data downloader, 2024.

[3] Benjamin Graham. *The Intelligent Investor: The Definitive Book on Value Investing.* HarperCollins, 2003. Referencia clave en Value Investing.

[4] Sebastian Deterding et al. From game design elements to gamefulness: defining gamification. In *Proceedings of the 15th international academic MindTrek conference.* ACM, 2011. Definición académica de gamificación.

[5] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall, 2008.

[6] Steve Krug. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability.* New Riders, 2014. Manual de referencia en usabilidad web.

[7] GitHub. *GitHub Actions Documentation*, 2024. Continuous Integration and Delivery (CI/CD) Platform.

[8] Holger Krekel et al. *pytest: helps you write better programs*, 2024. Testing framework.

[9] Pallets Projects. *Flask Documentation (3.0.x)*, 2024. Web Framework for Python.

[10] Dotdash Meredith. Investopedia stock simulator, 2024. Plataforma de educación financiera.

[11] Fusión Media Ltd. La bolsa virtual, 2024. Simulador de bolsa en español.

[12] Python Software Foundation. Python language reference, version 3.12, 2024. Accedido: 2024-12-01.

[13] The pandas development team. pandas: powerful python data analysis toolkit, 2024. Librería de análisis de datos.

[14] Charlie Marsh. Ruff: An extremely fast python linter, 2024. Herramienta de análisis estático.

[15] Render. Render cloud hosting, 2024. Cloud Application Hosting Platform.

[16] Guido Van Rossum, Barry Warsaw, and Nick Coghlan. Pep 8 – style guide for python code, 2001. Estándar de estilo de código Python.

[17] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2014. Sistema de control de versiones.

[18] Pallets Projects. *Jinja2 Documentation (3.1.x)*, 2024. Motor de plantillas para Python.

[19] Chart.js Contributors. *Chart.js Documentation (4.4.x)*, 2024. Librería de gráficos para JavaScript.

[20] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002. Referencia sobre el patrón MVC.

[21] The PostgreSQL Global Development Group. *PostgreSQL 16 Documentation*, 2024. Sistema de gestión de bases de datos relacional.