Manual de usuario – UNAL Study Assistant

Elaborado por:

- **Angie Gabriela Mesa Cardenas**
- **Juan Camilo Castiblanco Forero**
- **Oscar Alejandro Sandoval**

Presentado a

Néstor German Bolívar Pulgarín

Asignatura:

Programación Orientada a Objetos

Universiada Nacional de Colombia

2025 – 2s

Bogotá, DC

2025

Ficha Técnica

A continuación, se presentan los elementos técnicos y tecnológicos utilizados para el desarrollo de la plataforma de trading simulado **Wallet Trainer**:

A. Backend y Lógica

- **Lenguaje:** Python 3.13.
- **Framework Web:** Flask (Arquitectura MVC/MVVM).
- **Servidor de Aplicaciones:** Gunicorn (Producción en Render).
- **Motor Financiero:**
    - **CCXT:** Librería para conexión con exchanges de criptomonedas (Configurado con **Kraken** para evitar bloqueos geográficos en EE.UU.).
    - **Yfinance:** Librería para obtención de datos de mercado tradicional (Forex, Commodities y ADRs Colombianos).

B. Base de Datos y Nube

- **Plataforma:** Google Firebase.
- **Base de Datos:** Firebase Realtime Database (NoSQL). Se utiliza una estructura de árbol JSON para almacenar perfiles de usuario, configuraciones de bots y el historial transaccional (`trade_log`).
- **Hosting:** Render (PaaS).

C. Frontend (Interfaz)

- **Motor de Plantillas:** Jinja2.
- **Estilos:** Bootstrap 5 con personalización "Dark/Gold Theme" (CSS propio).
- **Visualización:** Chart.js (Gráficos de Dona y Línea) y TradingView Widgets (Velas Japonesas).
- **Visualización de Mercados: TradingView Widgets**. Se implementó la librería de *Advanced Real-Time Chart Widget* mediante inyección de JavaScript para la renderización de gráficos de velas japonesas (Candlesticks) y el *Ticker Tape* (cinta de cotizaciones superior), permitiendo análisis técnico visual independiente del backend.r

D. Infraestructura y Despliegue (DevOps)

- **Plataforma de Nube (PaaS): Render**. Se seleccionó por su capacidad de despliegue continuo (CI/CD) automatizado desde GitHub y su soporte nativo para Python a través de Gunicorn.
- **Servidor Web:** Gunicorn (Green Unicorn), un servidor WSGI HTTP para UNIX que actúa como intermediario entre la aplicación Flask y el tráfico de internet entrante en Render.

E. Proveedores de Datos de Mercado (Data Feeds)

- **Criptomonedas: Kraken Exchange**. Se utiliza la API pública de Kraken (vía librería `ccxt`) en lugar de Binance, debido a que Kraken ofrece mayor estabilidad y permisividad para peticiones provenientes de direcciones IP alojadas en centros de datos de Estados Unidos (como los de Render).

- **Mercado Bursátil: Yahoo Finance**. Proveedor de datos para acciones (Stocks), divisas (Forex) y materias primas.

Requerimientos del Software

Requerimientos Funcionales (RF)

- **RF01 - Autenticación:** El sistema debe permitir el registro e inicio de sesión seguro de usuarios mediante correo y contraseña.
- **RF02 - Visualización de Mercado:** El usuario debe poder visualizar gráficos de velas japonesas actualizados al día para cualquier activo seleccionado.
- **RF03 - Simulación de Trading:** El sistema debe permitir abrir posiciones de "Compra" y "Venta" respetando el saldo disponible y el inventario de activos (sin ventas en corto descubiertas).
- **RF04 - Análisis Automatizado:** El sistema debe proveer sugerencias de inversión basadas en indicadores técnicos (RSI y Medias Móviles) generadas por un algoritmo interno.

Requerimientos No Funcionales (RNF)

- **RNF01 - Rendimiento:** La carga de datos financieros no debe superar los 2 segundos de latencia promedio.
- **RNF02 - Disponibilidad:** El sistema debe estar disponible 24/7 mediante despliegue en la nube (Render).
- **RNF03 - Seguridad:** Las credenciales de usuario y claves de API no deben ser visibles ni almacenadas en texto plano en el repositorio de código.
- **RNF04 - Usabilidad:** La interfaz debe implementar un diseño responsivo y un tema oscuro (Dark Mode) para reducir la fatiga visual durante sesiones prolongadas de uso.

Objetivos del Proyecto

1.1. Objetivo General

Desarrollar una plataforma web de simulación financiera (Paper Trading) que integre múltiples fuentes de datos de mercado en tiempo real, permitiendo a los usuarios aplicar estrategias de inversión en activos globales y locales sin riesgo financiero.

1.2. Objetivos Específicos

1. **Implementar una arquitectura de software desacoplada** utilizando el patrón MVVM (Model-View-ViewModel) en Python para garantizar la mantenibilidad y escalabilidad del código.
2. **Integrar APIs financieras heterogéneas** (Kraken y Yahoo Finance) para unificar la visualización de precios de Criptomonedas y Acciones Colombianas (ADRs) en una sola interfaz.

3. **Desarrollar un algoritmo de conciliación bancaria** que asegure la integridad transaccional del saldo del usuario frente a operaciones de compra y venta.
4. **Desplegar la solución en una infraestructura de nube (PaaS)** utilizando prácticas de integración continua (CI/CD) para asegurar la disponibilidad del servicio.

Instalación y Despliegue

Requisitos Previos

El proyecto requiere un archivo `requirements.txt` en la raíz para gestionar las dependencias en el servidor de despliegue. Las librerías críticas son:

- `flask`: Núcleo de la aplicación.
- `firebase-admin`: Conexión segura a la BD.
- `ccxt`: Conectividad con mercados cripto.
- `yfinance`: Datos de bolsa (NYSE, NASDAQ, BVC).
- `pandas`: Procesamiento de datos para indicadores técnicos (RSI, Medias Móviles).
-

Estructura del Proyecto

El software sigue un patrón de diseño modular:

- **wallet-trainer/**
- **├── app.py # Controlador principal (Rutas Flask)**
- **├── requirements.txt # Dependencias (Flask, CCXT, Pandas...)**
- **├── Procfile # Archivo de arranque para Render**
- **├── firebase_config.py # Configuración de conexión a Firebase**
- **├── .gitignore # Archivos ignorados por Git**
-
- **├── model/ # Capa de Datos (Acceso a BD)**
- **│ ├── __init__.py**
- **│ ├── auth_service.py # Lógica de Login/Registro**
- **│ ├── bot_service.py # Lógica de Logs y Trading**
- **│ └── db_service.py # Conexión CRUD con Firebase**
-
- **├── viewmodels/ # Capa de Lógica de Negocio**
- **│ ├── __init__.py**
- **│ └── main_viewmodel.py # El "Cerebro" (Cálculos, APIs, Conciliación)**
-
- **├── static/ # Archivos Estáticos**
- **│ └── style.css # Estilos personalizados (Dark Theme)**
-
- **└── templates/ # Vistas (Interfaz HTML + Jinja2)**

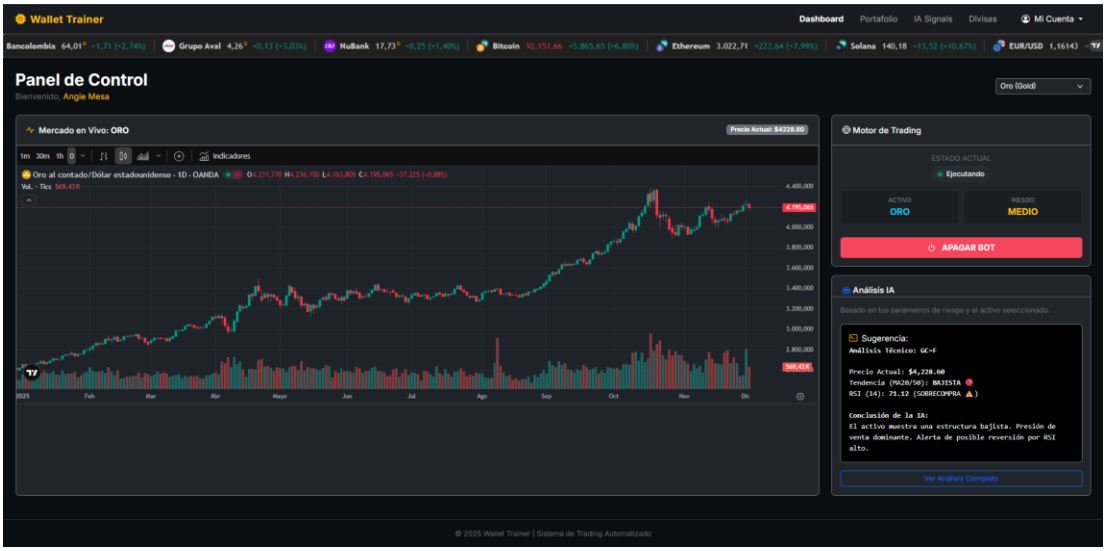
- ├── **base.html # Plantilla maestra (Navbar, Footer)**
- ├── **login.html # Inicio de sesión**
- ├── **register.html # Registro de usuario**
- ├── **forgot_password.html # Recuperación de cuenta**
- ├── **dashboard.html # Panel principal con gráficas**
- ├── **sugerencias.html # Terminal IA y Trading Manual**
- ├── **rendimientos.html # Portafolio y Auditoría**
- ├── **converter.html # Conversor Universal de Divisas**
- ├── **profile.html # Gestión de perfil de usuario**
- ├── **ajustes.html # Configuración del Bot**
- └── **api_keys.html # Gestión de llaves externas**

`app.py`: Controlador principal y rutas.
`viewmodels/`: Lógica de negocio y conciliación financiera.
`model/`: Capa de acceso a datos (DAO).
`templates/`: Vistas HTML.

Guía de Uso

1. Panel de Control (Dashboard)

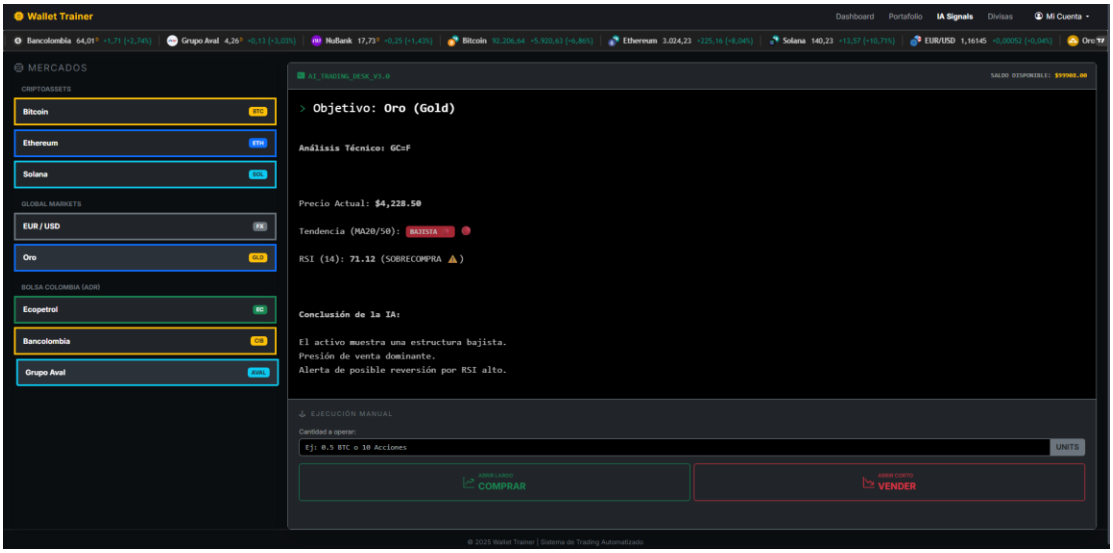
Esta es la terminal principal. El usuario tiene acceso a un selector rápido de activos que incluye Criptomonedas, Forex y **Acciones Colombianas (Ecopetrol, Bancolombia, Aval)**.



- **Gráfico en Vivo:** Widget profesional de TradingView que muestra el comportamiento del precio en tiempo real.
- **Ticker Tape:** Barra superior con cotizaciones de las principales empresas colombianas en la bolsa de Nueva York.

2. Módulo de Inteligencia Artificial (IA Signals)

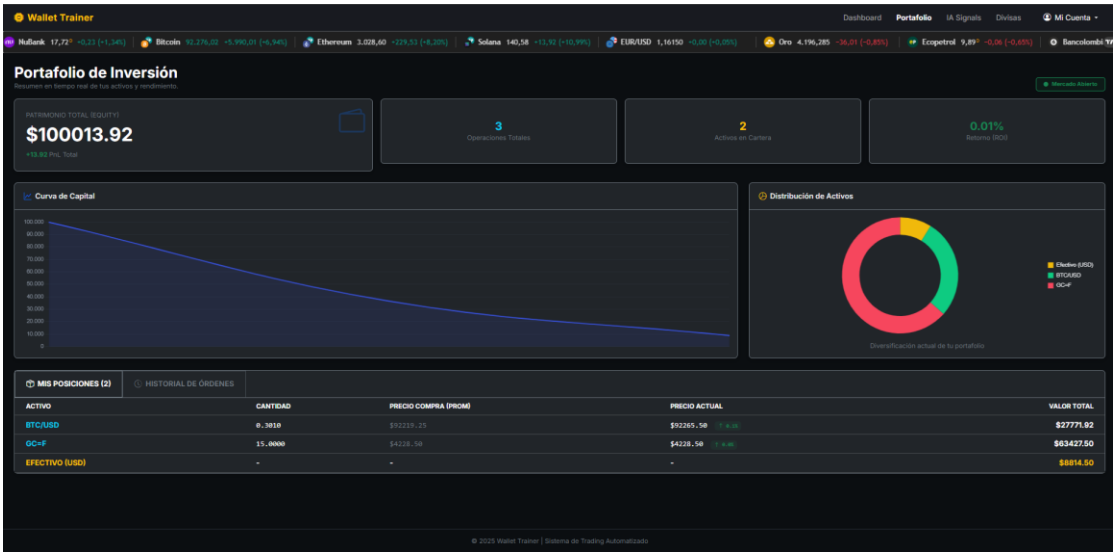
Diseñado con una estética de "Terminal Hacker", este módulo ofrece análisis técnico automatizado.



- **Análisis:** El sistema descarga el historial de precios, calcula indicadores (RSI, Cruce de Medias) y emite un veredicto (ALCISTA/BAJISTA).
- **Panel de Trading:** Permite ejecutar operaciones de compra y venta. El sistema valida el saldo disponible ($100,000 iniciales) antes de ejecutar.

3. Portafolio y Rendimiento

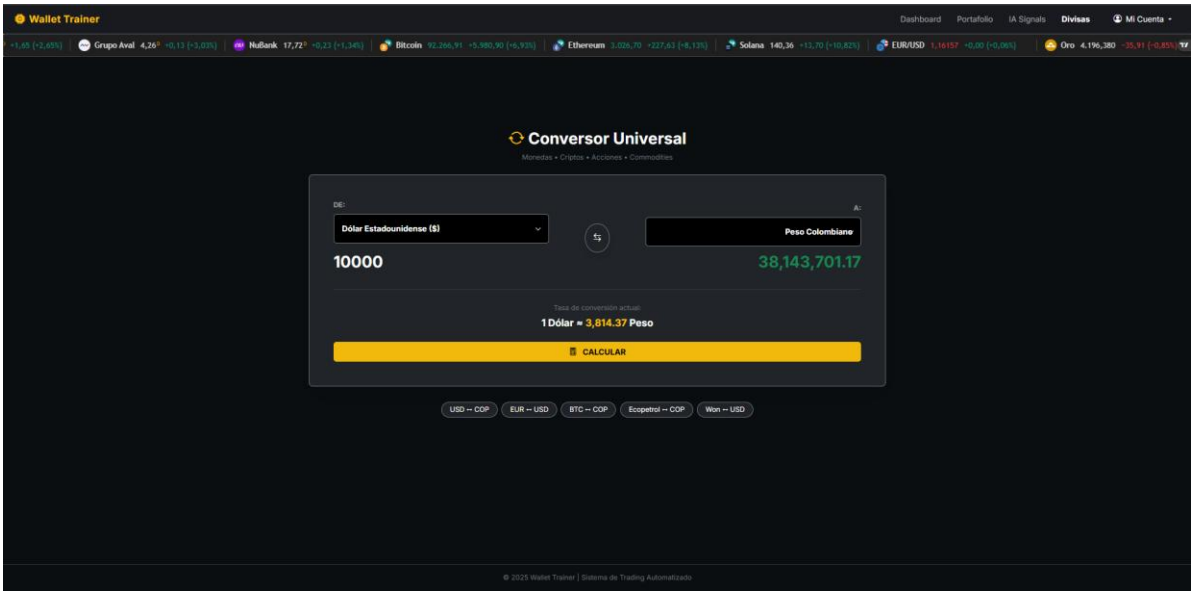
El módulo más avanzado del sistema, diseñado para la auditoría de cuentas.



- **Gráfico de Dona:** Muestra la diversificación de la cartera (Ej: 60% Bitcoin, 40% Efectivo).
- **Tabla de Posiciones:** Muestra el inventario actual, el precio promedio de compra y la rentabilidad en tiempo real (verde/rojo).

4. Conversor Universal

Herramienta auxiliar para la conversión de divisas fiat.



Utiliza el dólar como moneda puente para permitir conversiones cruzadas entre cualquier par de divisas soportado por el sistema financiero.

5. Flujo Paso a Paso: Realizar una Inversión

A continuación, se describe el procedimiento estándar para adquirir acciones de Ecopetrol (ADR):

1. **Navegación:** Diríjase al menú superior y seleccione **"IA Signals"**.
2. **Selección:** En el panel izquierdo, ubique la sección **"BOLSA COLOMBIA (ADR)"** y haga clic en **Ecopetrol**.
3. **Análisis:** Espere a que la terminal de IA cargue el análisis técnico (RSI y Tendencia).
4. **Operación:**
    - Ubique el panel inferior de "Ejecución Manual".
    - Ingrese la cantidad de acciones a comprar (Ej: 100).
    - Haga clic en el botón verde **"COMPRAR"**.
5. **Confirmación:** El sistema mostrará un mensaje de éxito y actualizará el "Saldo Disponible" en la cabecera inmediatamente.
6. **Verificación:** Diríjase a la pestaña **"Portafolio"** para ver su nueva posición en la tabla de activos.

Explicación Técnica (Algoritmos Clave)

A continuación se detallan las tres innovaciones lógicas implementadas en el backend:

1. Motor Híbrido de Precios (Data Routing)

Para solucionar el problema de fragmentación de mercados y bloqueos de IP, se implementó un "Router de Datos" en la clase `MainViewModel`. El algoritmo detecta el tipo de activo solicitado:

- Si es **Cripto (BTC, ETH):** Redirige la petición a la API de **Kraken** vía `ccxt`.
- Si es **Acción/Forex (Ecopetrol, USD/COP):** Redirige la petición a **Yahoo Finance**. Esto permite que la plataforma opere acciones colombianas en dólares (usando sus ADRs) sin conflictos de tasa de cambio.

2. Conciliación Bancaria Estricta

A diferencia de sistemas simples que guardan el saldo en una variable, Wallet Trainer implementa un sistema de **reconstrucción de saldo**. Cada vez que el usuario carga la página, el sistema ejecuta la función `_reconcile_balance`:

1. Toma el saldo inicial base ($100,000).
2. Recorre todo el historial de transacciones en Firebase.
3. Resta todas las compras y suma todas las ventas. Esto elimina bugs de inconsistencia de datos y garantiza integridad financiera.

3. Validación de Inventario (Spot Trading)

El sistema impide la venta en corto (Short Selling) no autorizada. Antes de ejecutar una orden de "VENTA", el backend invoca `_calculate_holdings`, una función que suma el inventario histórico del activo específico. Si el usuario intenta vender 1 Bitcoin pero solo posee 0.5, la transacción es rechazada a nivel de servidor, protegiendo la economía de la simulación.

4. Diseño Orientado a Objetos (Arquitectura MVVM)

El proyecto no es solo un conjunto de scripts, sino que sigue una arquitectura estructurada basada en clases:

- **Clase `MainViewModel` (El Cerebro):** Actúa como el orquestador principal. No accede directamente a la base de datos ni a la vista HTML. Su función es encapsular la lógica de negocio.
    - *Polimorfismo en Datos:* Utiliza el método `_get_symbol_and_source` para tratar indistintamente activos de Cripto (Kraken) y Bolsa (Yahoo) bajo una misma interfaz de precios.
- **Clase `BotService` (Persistencia):** Es la clase encargada de la comunicación con Firebase. Sigue el principio de responsabilidad única, encargándose solo de leer/escribir logs y configuraciones.
- **Clase `AuthService` (Seguridad):** Clase dedicada exclusivamente a la gestión de sesiones y tokens de autenticación.

El proyecto se encuentra alojado en **Render**, utilizando un flujo de trabajo moderno de DevOps:

1. **Control de Versiones:** El código fuente reside en un repositorio de **GitHub**.

2. **Integración Continua (CI):** Render está conectado al repositorio. Cada vez que se realiza un `git push` a la rama `main`, Render detecta los cambios automáticamente.
3. **Construcción (Build):** El servidor lee el archivo `requirements.txt` e instala las dependencias (`ccxt`, `flask`, `pandas`, etc.) en un entorno virtual aislado.
4. **Ejecución:** Se levanta el servicio mediante el comando `gunicorn app:app`, exponiendo la aplicación al puerto 10000 para el acceso público vía HTTPS.

5. Visualización Desacoplada (Client-Side Rendering)

Para optimizar el rendimiento del servidor, la aplicación utiliza una arquitectura de **visualización desacoplada**.

- **Lógica (Backend):** Python consulta el precio exacto (numérico) a Kraken/Yahoo en el momento de la transacción para garantizar la precisión matemática del saldo.
- **Visual (Frontend):** El navegador del cliente se conecta directamente a los servidores de datos de **TradingView** para renderizar el gráfico interactivo. Esta estrategia libera al servidor de Render de la carga de procesar y dibujar gráficos históricos, delegando esa tarea al navegador del usuario y garantizando una experiencia fluida (60 FPS) similar a una terminal profesional.

Solución de Problemas (Troubleshooting)

Problema	Causa Probable	Solución
El saldo volvió a $100,000	**Se realizó un reinicio de historial o es una cuenta nueva.**	**El sistema utiliza conciliación bancaria; si borra el historial, el saldo se reinicia matemáticamente a la base.**
Gráfico de Ecopetrol no carga	**El mercado de Nueva York (NYSE) está cerrado.**	**Los gráficos de acciones tradicionales pueden pausarse fines de semana. Las Criptos funcionan 24/7.**
Diferencia de precios	**Retraso en la API de Yahoo Finance.**	**Los precios de ADRs pueden tener un retraso de 15 minutos respecto al tiempo real dependiendo de la volatilidad.**

Conclusiones

El proyecto **Wallet Trainer** logra integrar exitosamente tecnologías heterogéneas (Python, Firebase, APIs Financieras) bajo una arquitectura robusta. Se cumplió con el objetivo de incluir activos del mercado colombiano mediante la estrategia de ADRs, permitiendo una simulación financiera globalizada sin los problemas de conversión de divisas constantes. La implementación de la **Conciliación Bancaria Estricta** demuestra un manejo avanzado de la integridad de datos, vital en sistemas financieros.