Portfolio & Market Insight Platform

Asignatura: Arquitectura del Software
Producto: API REST para gestión de carteras de inversión y análisis de mercado
Fecha: 12 de diciembre de 2025
Autores: Alan Ariel Salazar · Yago Ramos Sánchez
Repositorio: https://github.com/alanslzrr/portfolio_and_market_insight_platform

---

1. Introducción

Este proyecto desarrolla un producto software orientado a un problema real del sector empresarial/financiero: la **gestión de carteras de inversión** y el **análisis de mercado**. El sistema permite registrar operaciones (compra/venta), administrar portfolios, calcular métricas financieras y, de manera opcional, integrar fuentes externas para precios de mercado y generación de análisis mediante modelos de lenguaje.

La solución se implementa como una **API REST** en Python, con un diseño modular y separación de responsabilidades, incorporando principios de ingeniería del software, buenas prácticas de seguridad y mecanismos de autenticación basados en **JWT**. El backend se entrega **contenedorizado con Docker**, facilitando su ejecución y despliegue.

---

2. Objetivo general

Implementar un producto software en Python que resuelva el problema de gestión de carteras y análisis de mercado, expuesto como **API REST**, aplicando principios de arquitectura (separación de responsabilidades, modularidad), seguridad (JWT) y entrega reproducible mediante Docker.

---

3. Objetivos específicos

1. Modelar las entidades del dominio (usuarios, portfolios, activos, operaciones, análisis) mediante clases y relaciones de persistencia.
2. Exponer la solución como API REST versionada, implementando endpoints para la gestión de los recursos definidos.
3. Incorporar ciberseguridad mediante autenticación basada en tokens (JWT), manejo de sesiones y políticas de invalidación.
4. Implementar validación de datos de entrada/salida y manejo uniforme de errores.

5. Diseñar integraciones externas opcionales (Alpha Vantage para datos de mercado y OpenAI para análisis) sin comprometer el funcionamiento del núcleo.
6. Contenerizar el backend con Docker para garantizar portabilidad y reproducibilidad.

---

4. Metodología de trabajo

Se siguió una metodología **incremental e iterativa**, implementando el sistema por bloques (fundación → persistencia → validación → seguridad → repositorios → servicios → endpoints → integraciones). Esta aproximación permitió:

- Validar componentes de forma progresiva.
- Reducir complejidad al incorporar funcionalidades paso a paso.
- Aislar responsabilidades mediante capas (API, servicios, repositorios, modelos/esquemas).

Adicionalmente, se mantuvo documentación de decisiones y validaciones durante el desarrollo (ver el histórico en `avances_old_readme.md`).

---

5. Arquitectura y diseño

5.1 Enfoque arquitectónico

El backend implementa una **arquitectura modular en capas**, con bajo acoplamiento y responsabilidades separadas.

5.2 Capas principales

- **Capa de Presentación (API / FastAPI):** define endpoints REST, versionado (`/api/v1`), validación de entrada/salida y contratos.
- **Capa de Lógica de Negocio (Services):** orquesta reglas de negocio (registro, login, operaciones financieras, métricas, análisis).
- **Capa de Acceso a Datos (Repositories):** aplica el patrón Repository para aislar persistencia y consultas.
- **Modelos y Esquemas:**
  - **SQLAlchemy Models** para persistencia en PostgreSQL.
  - **Pydantic Schemas** para validación y serialización.
  - **Alembic** para migraciones.

5.3 Decisiones relevantes

- **Decimal para valores monetarios:** evita errores de precisión asociados a `float` en cálculos financieros.

- **UUID como identificadores:** reduce riesgo de enumeración de recursos y favorece escalabilidad.
- **Integraciones externas opcionales:** el núcleo del sistema funciona sin depender de Alpha Vantage/OpenAI.

---

6. Seguridad

La solución incorpora principios de ciberseguridad estudiados en la asignatura:

- **Hash de contraseñas con bcrypt**.
- **Autenticación JWT** con *access tokens* y *refresh tokens*.
- **Gestión de sesiones** (persistencia de refresh tokens) y operaciones de logout individual y global.
- **Invalidación de sesiones al cambiar contraseña**, como medida de mitigación ante compromiso de credenciales.
- **Separación de excepciones de dominio** y conversión a respuestas HTTP consistentes mediante handlers.

---

7. Tecnologías y dependencias

- **Python 3.11**
- **FastAPI** + **Uvicorn**
- **SQLAlchemy** + **Alembic**
- **PostgreSQL**
- **Pydantic** (validación y configuración)
- **httpx** (clientes HTTP)
- Integraciones opcionales:
  - **Alpha Vantage API** (datos de mercado)
  - **OpenAI API** (análisis con modelos de lenguaje)

Las dependencias del backend están declaradas en `backend/requirements.txt`.

---

8. Configuración

8.1 Variables de entorno

Crear (o editar) el archivo `config/.env` con:

```
DATABASE_URL=postgresql://username:password@localhost:5432/portfolio_db
ALPHA_VANTAGE_API_KEY=
OPENAI_API_KEY=
```

Notas:

- `ALPHA_VANTAGE_API_KEY` y `OPENAI_API_KEY` son opcionales. Sin ellas, el sistema mantiene el núcleo funcional y deshabilita únicamente las funcionalidades que dependan de dichas integraciones.

---

9. Instalación y ejecución (local)

9.1 Base de datos

Ejemplo para macOS con Homebrew:

```
brew install postgresql
brew services start postgresql
createdb portfolio_db
```

9.2 Migraciones

```
cd backend
alembic upgrade head
```

9.3 Ejecutar API

```
cd backend
uvicorn main:app --host 0.0.0.0 --port 8000
```

---

10. Ejecución con Docker (backend)

El backend se entrega contenerizado mediante el archivo `backend/Dockerfile`.

```
cd backend
docker build -t portfolio-backend .
docker run -p 8000:8000 --env-file ../config/.env portfolio-backend
```

---

11. API REST (visión general)

La API está versionada bajo `/api/v1` y organiza rutas por dominio.

11.1 Autenticación

- `POST /api/v1/auth/register`
- `POST /api/v1/auth/login`
- `POST /api/v1/auth/refresh`
- `POST /api/v1/auth/logout`
- `POST /api/v1/auth/logout-all`

11.2 Usuarios

- `GET /api/v1/users/me`
- `PUT /api/v1/users/me`
- `PUT /api/v1/users/me/password`

11.3 Portfolios

- `GET /api/v1/portfolios`
- `POST /api/v1/portfolios`
- `GET /api/v1/portfolios/{id}`
- `PUT /api/v1/portfolios/{id}`
- `DELETE /api/v1/portfolios/{id}`

11.4 Operaciones

- `GET /api/v1/operations`
- `POST /api/v1/operations`
- `GET /api/v1/operations/{id}`
- `GET /api/v1/operations/stats/{portfolio_id}`
- `PUT /api/v1/operations/{id}` (p. ej. notas; se preserva integridad del historial financiero)

11.5 Mercado (datos)

- `GET /api/v1/market/assets/search?q=<query>`
- `GET /api/v1/market/assets/{symbol}`
- `GET /api/v1/market/prices/{symbol}/current`
- `GET /api/v1/market/prices/{symbol}/historical?days=30`
- `POST /api/v1/market/assets`

---

12. Evidencias de verificación (Postman)

Las siguientes capturas corresponden a pruebas de endpoints principales realizadas en Postman.

12.1 Registro de usuario

12.2 Login (JWT)

12.3 Refresh token

12.4 Logout

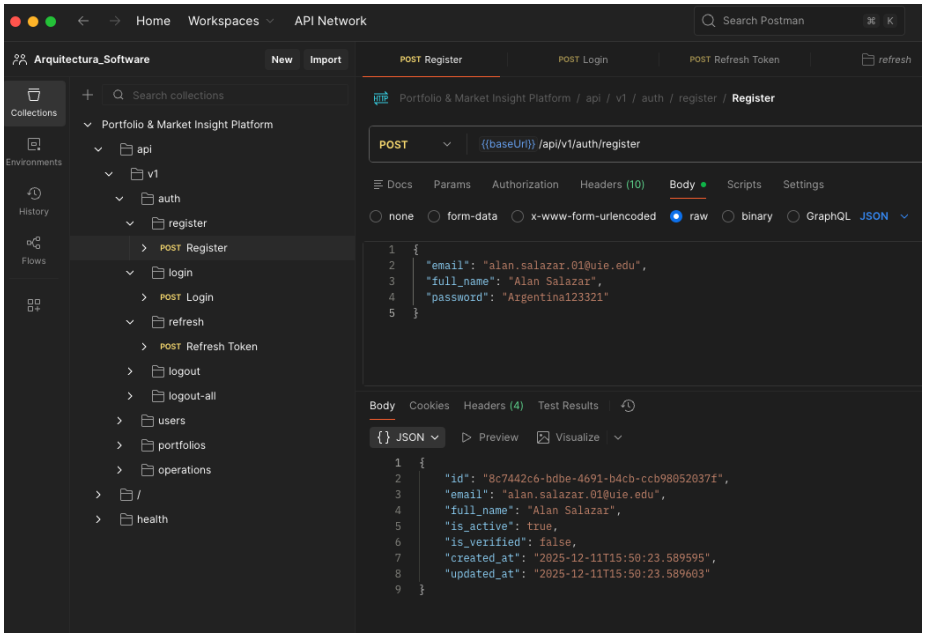

Figure 1: Registro de usuario

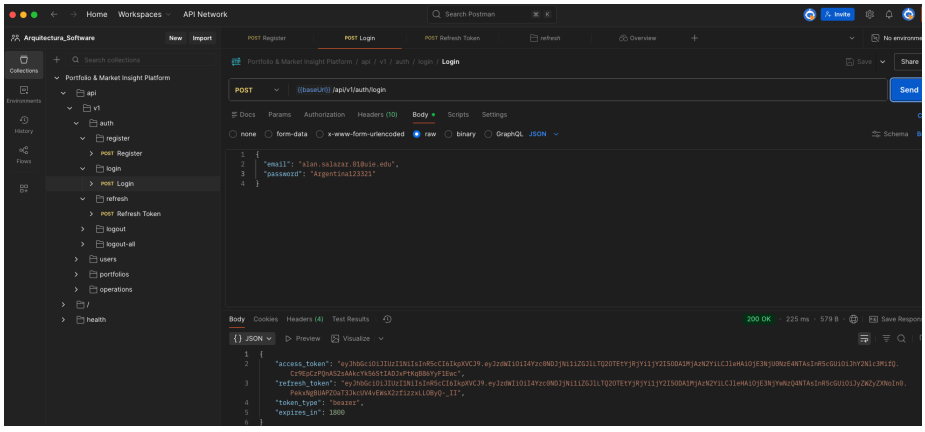

Figure 2: Login JWT

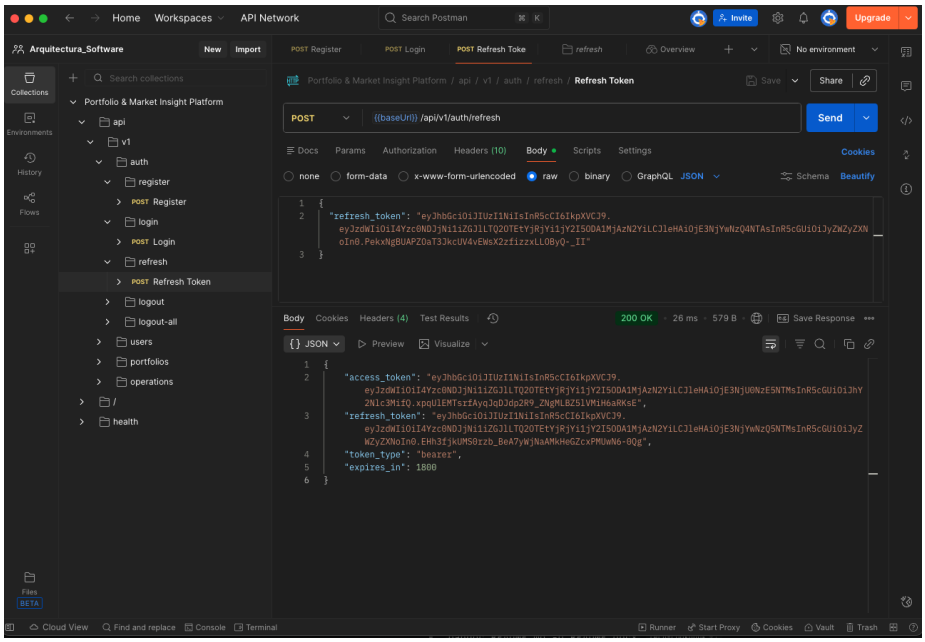

Figure 3: Refresh token

13. Estructura del repositorio (resumen)

- `backend/`: API REST (FastAPI), persistencia, seguridad, servicios, repositorios, migraciones.
- `frontend/`: interfaz Streamlit (cliente) para consumo de la API.
- `ai_module/`: módulo auxiliar de análisis/IA (componentes específicos del proyecto).
- `postman/`: colección y evidencias (imágenes) de pruebas.
- `tests/`: pruebas unitarias e integración.

---

14. Desarrollo (síntesis académica)

El desarrollo se realizó por etapas, destacando:

1. **Fundación del proyecto:** estructura de directorios, dependencias y configuración centralizada mediante variables de entorno.
2. **Persistencia:** modelos ORM con SQLAlchemy y gestión de migraciones con Alembic.
3. **Validación:** esquemas Pydantic para requests/responses, validaciones de reglas (p. ej., coherencia de solicitudes de análisis).
4. **Seguridad:** hash de contraseñas, generación/validación de JWT, manejo de sesiones con refresh tokens.

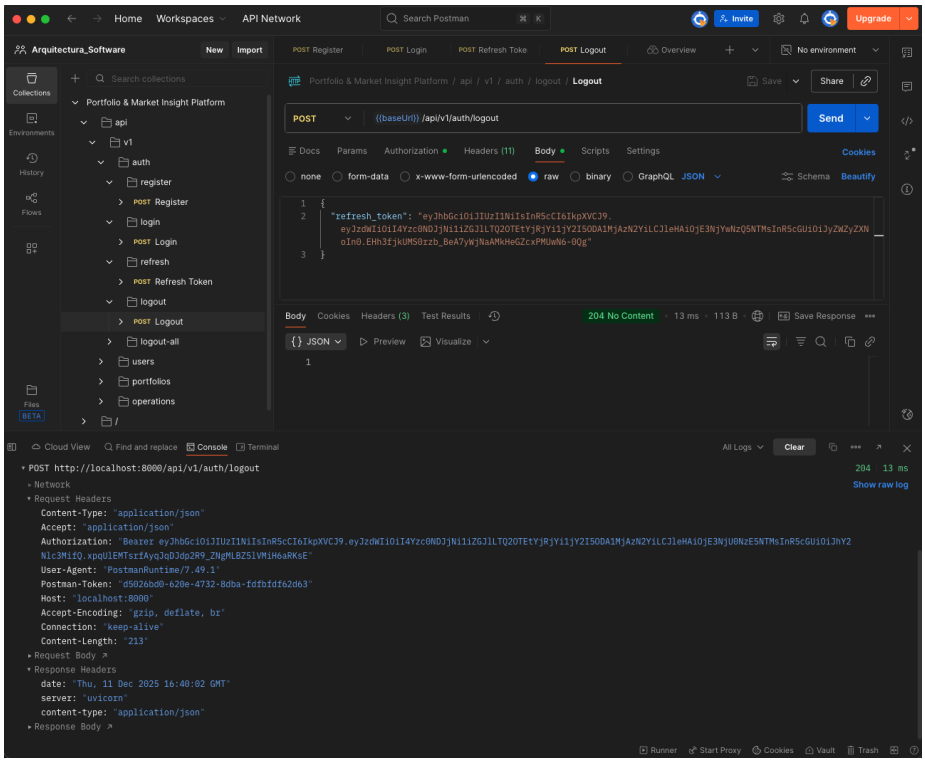

Figure 4: Logout

5. **Acceso a datos:** repositorios genéricos y específicos (User, Portfolio, Operation, Asset, Analysis).
6. **Servicios:** implementación de casos de uso (AuthService, PortfolioService, OperationService, Market/Analysis según configuración).
7. **API REST:** routers por dominio, validación de ownership (recursos por usuario) y documentación OpenAPI.
8. **Integraciones externas (opcionales):** datos de mercado (Alpha Vantage) y análisis con modelos de lenguaje (OpenAI), preservando el núcleo funcional.

Para un detalle exhaustivo del proceso y decisiones, consultar `avances_old_readme.md`.

---

16. Autores

- Alan Ariel Salazar
- Yago Ramos Sánchez