ProjectOverview

📌 Project: Stock Portfolio Analytics API

🧠 Language

- **Python 3.11+**

🏗️ Backend Stack

- **FastAPI** (web framework)
- **PostgreSQL** (database)
- **SQLAlchemy** (ORM)
- **JWT** (authentication)
- **Redis** (optional caching)
- **Pytest** (testing)
- **Docker** (optional but strong)

🖥️ System Setup (Inside WSL Only)

1️⃣ Update System

sudo apt update && sudo apt upgrade

2️⃣ Install Python Tools

sudo apt install python3-pip python3-venv

Check:

```
python3 --version
pip3 --version
```

3️⃣ Install PostgreSQL

```
sudo apt install postgresql postgresql-contrib
```

Start PostgreSQL:

```
sudo service postgresql start
```

Access PostgreSQL:

```
sudo -u postgres psql
```

Create database:

```
CREATE DATABASE portfolio;
CREATE USER portfolio_user WITH PASSWORD 'yourpassword';
ALTER ROLE portfolio_user SET client_encoding TO 'utf8';
ALTER ROLE portfolio_user SET default_transaction_isolation TO 'read committed';
ALTER ROLE portfolio_user SET timezone TO 'UTC';
GRANT ALL PRIVILEGES ON DATABASE portfolio TO portfolio_user;
```

Exit:

```
\q
```

4️⃣ Install Redis (Optional but Recommended)

```
sudo apt install redis-server
sudo service redis-server start
```

Check:

```
redis-cli ping
```

Should return:

```
PONG
```

📁 Project Setup

Create project folder:

```
mkdir portfolio-api
cd portfolio-api
```

Create virtual environment:

```
python3 -m venv venv
source venv/bin/activate
```

📦 Install Python Dependencies

```
pip install fastapi
pip install uvicorn
pip install sqlalchemy
pip install psycopg2-binary
pip install python-jose[cryptography]
pip install passlib[bcrypt]
pip install python-multipart
pip install pydantic
pip install redis
pip install httpx
pip install pytest
pip install python-dotenv
```

Freeze requirements:

```
pip freeze > requirements.txt
```

🔐 Environment Variables

Create `.env` file:

```
DATABASE_URL=postgresql://portfolio_user:yourpassword@localhost/portfolio
SECRET_KEY=your_super_secret_key
STOCK_API_KEY=your_external_api_key
```

🌐 External API

Choose one stock API:

- Finnhub
- Alpha Vantage
- Twelve Data

You will:

- Register
- Get API key
- Store in `.env`

🗂️ Recommended Folder Structure

```
portfolio-api/
│
├── app/
│   ├── main.py
│   ├── database.py
│   ├── models.py
│   ├── schemas.py
│   ├── auth.py
│   ├── routers/
│   │   ├── users.py
│   │   └── portfolio.py
│   └── services/
│       └── stock_service.py
│
├── tests/
├── .env
├── requirements.txt
```

You can add Docker later if you want.

🚀 Run the API

uvicorn app.main:app --reload

Open in browser:

http://127.0.0.1:8000/docs

Swagger UI auto-generates documentation.

🎯 Minimum Required Features

- User registration
- User login (JWT)
- Add stock position
- Remove stock position
- View portfolio
- Calculate total portfolio value
- Calculate gain/loss percentage
- Fetch live stock price from external API

🧠 Strong Additions (If Time Allows)

- Redis caching for stock prices

- Background tasks
- Pagination
- 5–10 Pytest tests
- Docker containerization
- Deployment (Render, Railway, etc.)

README

Stock Portfolio Analytics API

This project is a production-style backend API built using FastAPI and PostgreSQL.

The API allows users to securely register and authenticate, manage stock positions, retrieve real-time financial data from external APIs, and compute portfolio performance metrics including total value, allocation breakdown, and percentage gain/loss.

The system is structured using a layered architecture with database persistence, authentication middleware, and external API integration to simulate real-world backend engineering practices.

4-Week Plan

🗓️ WEEK 1 — Foundation & Authentication

🎯 Goal:

Have a running FastAPI app with:

- Database connected
- User registration
- JWT login working

If you finish Week 1 correctly, the rest becomes straightforward.

Day 1–2: Project Skeleton

- Create project structure
- Setup virtual environment
- Install dependencies
- Connect FastAPI to PostgreSQL
- Create `database.py`

Deliverable:

- App runs
- DB connection works
- `/docs` loads

Day 3–4: Database Models

Create tables:

- `users`
- `portfolio_positions`

Fields for `users`:

- id
- email
- hashed_password
- created_at

Fields for `portfolio_positions`:

- id
- user_id (foreign key)
- stock_symbol
- quantity
- purchase_price
- created_at

Deliverable:

- Tables auto-create
- Relationships working

Day 5–6: Authentication

Implement:

- Password hashing (bcrypt)
- Register endpoint
- Login endpoint
- JWT token creation
- Dependency for protected routes

Deliverable:

- You can:
    - Register
    - Login
    - Access protected route with token

If this works cleanly → you're on track.

🗓️ WEEK 2 — Portfolio Core Logic

🎯 Goal:

Users can manage stock positions and see calculated portfolio value.

Day 1–2: CRUD for Portfolio

Endpoints:

- Add stock
- Remove stock
- View user portfolio

Deliverable:

- Positions stored in DB
- Only authenticated users can access their own data

Day 3–4: External Stock API Integration

- Connect to stock API
- Fetch live price
- Handle API errors
- Add simple retry logic

Deliverable:

- Endpoint returns current price for symbol

Day 5–6: Portfolio Calculations

Compute:

- Total portfolio value

- Total invested capital
- Gain/loss %
- Per-stock performance

This is where thinking happens.

Deliverable:

- `/portfolio/summary` endpoint returns:
    - total_value
    - total_gain_loss_percent
    - breakdown per stock

Now your app feels real.

🗓️ **WEEK 3 — Professional Polish**

🎯 **Goal:**

Make it look internship-level.

Day 1–2: Refactor Architecture

Separate:

- Routers
- Services
- Models

- Schemas

Move stock logic into:

services/stock_service.py

Deliverable:

- Clean layered structure
- No messy giant files

Day 3–4: Add Caching (Redis)

Cache stock prices:

- Cache per symbol
- Expire after 60 seconds

Deliverable:

- External API not called every time

This shows engineering maturity.

Day 5–6: Add Testing

Write:

- 3–5 authentication tests
- 3–5 portfolio tests

Even basic tests = huge resume boost.

Deliverable:

- `pytest` runs successfully

🗓️ WEEK 4 — Production Mindset

🎯 Goal:

Make it look deployable & serious.

Day 1–2: Error Handling & Validation

Add:

- Proper HTTP status codes
- Input validation
- Clean error responses

Make API feel polished.

Day 3–4: Docker (Optional but Strong)

Add:

- Dockerfile
- docker-compose for:
    - FastAPI
    - PostgreSQL

    - Redis

Deliverable:

- `docker-compose up` runs everything

Day 5–6: Documentation & Deployment

- Write strong README
- Add architecture diagram (simple)
- Deploy to:
    - Render
    - Railway
    - Fly.io

Even if small — public URL matters.