Stock Analyst AI: A Full-Stack Application for Real-Time Stock Analysis and Portfolio Management

Nishad Bagade (MT2024102)

November 8, 2025

GitHub Repository: https://github.com/dyinghorizon/Elimentary_Task

Project Presentation: https://drive.google.com/drive/folders/1Tmq1WvOqb9WMdyyf0J7crds86ReImtyn?usp=drive_link

Abstract

This report presents Stock Analyst AI, a full-stack web application for stock market analysis and portfolio management. The system integrates real-time financial data from Yahoo Finance, leverages Google's Gemini API for AI-powered stock analysis, and implements role-based access control for financial analysts and individual investors. The architecture comprises a FastAPI backend with SQLite database and a React-based frontend with interactive visualization. Key features include real-time market data integration, AI-driven trading recommendations with portfolio allocation strategies, 30-day technical trend visualization, transaction-based portfolio management with profit/loss tracking, and role-differentiated access controls.

Contents

1 Introduction **3**
- 1.1 Project Overview . . . 3
- 1.2 Objectives . . . 3

2 System Architecture **3**
- 2.1 Technology Stack . . . 3
  - 2.1.1 Backend Infrastructure . . . 3
  - 2.1.2 Frontend Architecture . . . 3
- 2.2 Database Schema . . . 4
- 2.3 Architecture Pattern . . . 4

3 Core Functionality **4**
- 3.1 Real-Time Market Data Integration . . . 4
- 3.2 AI-Powered Stock Analysis . . . 4
  - 3.2.1 Prompt Engineering . . . 5
  - 3.2.2 Output Validation . . . 5
- 3.3 Portfolio Management System . . . 5
  - 3.3.1 Position Consolidation . . . 5
- 3.4 Technical Visualization . . . 6
- 3.5 Access Control and Security . . . 6

4 Implementation Challenges **6**
- 4.1 Financial Data Source Selection . . . 6
- 4.2 AI Output Consistency . . . 6
- 4.3 Portfolio Aggregation . . . 6
- 4.4 Frontend State Management . . . 7

5 Performance Evaluation **7**
- 5.1 System Performance . . . 7
- 5.2 Implemented Features . . . 7
- 5.3 Security Implementation . . . 7

6 Potential Future Enhancements **8**
- 6.1 Technical Improvements . . . 8
- 6.2 Feature Additions . . . 8
- 6.3 Scalability Considerations . . . 8

7 Conclusion **9**

1 Introduction

1.1 Project Overview

Stock Analyst AI is a web-based platform that provides stock market analysis through integration of artificial intelligence and real-time market data. The application serves two user types—financial analysts and individual investors—with role-appropriate interfaces and access privileges.

1.2 Objectives

The primary objectives of this project are:

- Implement a secure full-stack architecture for financial data processing
- Integrate real-time stock market data through financial APIs
- Develop AI-powered analysis capabilities for trading insights
- Create visualization tools for technical price trend analysis
- Build portfolio management functionality with transaction tracking
- Implement role-based access control for security

2 System Architecture

2.1 Technology Stack

2.1.1 Backend Infrastructure

The backend is built on FastAPI [1], a Python web framework with automatic API documentation and async support. The system uses SQLite [9] for data persistence. Authentication is handled through JSON Web Tokens (JWT) [7] with bcrypt [10] password hashing.

Financial data is sourced through the yfinance library [3], which provides access to Yahoo Finance data. AI analysis is powered by Google's Gemini 2.0 Flash model [5].

2.1.2 Frontend Architecture

The frontend is developed using React [2] with Vite build tool. HTTP communication uses Axios for request handling. Data visualization uses Recharts [8], a charting library for React. The interface uses custom CSS for styling.

2.2 Database Schema

The SQLite database consists of three tables:

Users Table: Stores authentication credentials (bcrypt-hashed passwords) and role assignments (analyst or investor).

Portfolios Table: Transaction-based ledger where each entry represents a buy or sell operation with quantity, price, and timestamp.

Reports Table: Archives analysis outputs, linking each AI-generated recommendation to the user and timestamp.

2.3 Architecture Pattern

The application follows a three-tier architecture:

1. **Presentation Layer:** React single-page application
2. **Application Layer:** FastAPI RESTful services
3. **Data Layer:** SQLite database

3 Core Functionality

3.1 Real-Time Market Data Integration

The system interfaces with Yahoo Finance through the yfinance library [3]. Initial implementation explored Finnhub API [4], but free tier limitations led to migration to Yahoo Finance, which offers comprehensive data without API key restrictions.

The data retrieval system fetches:

- Current market price and previous close for change calculations
- Intraday high, low, and volume metrics
- 30-day historical price series for trend analysis
- Company metadata including name and ticker symbol

3.2 AI-Powered Stock Analysis

The analysis engine uses Google's Gemini 2.0 Flash model [5] for generating trading recommendations. A key implementation challenge was ensuring consistency between recommendations and portfolio allocation percentages.

3.2.1 Prompt Engineering

The system uses structured prompt engineering. The AI model receives:

- Real-time market data (current price, daily change, volume)
- Historical context (30-day trend with momentum calculation)
- Explicit recommendation rules mapping trend strength to allocation percentages
- Formatting constraints for structured output

The recommendation framework:

- **STRONG BUY:** 18-25% portfolio allocation for strong uptrends (¿5% 30-day gain)
- **BUY:** 10-17% allocation for moderate uptrends (2-5% gain)
- **HOLD:** 5-12% allocation for sideways consolidation
- **SELL:** 0-5% allocation for downtrends

3.2.2 Output Validation

Post-processing logic validates AI outputs to prevent contradictory recommendations. If recommendation category mismatches allocation percentage (e.g., SELL with 15% allocation), the system applies correction rules to enforce consistency.

3.3 Portfolio Management System

The portfolio uses a transaction-based ledger approach. Each buy or sell operation creates a database entry with positive or negative quantities. This design provides:

- Complete transaction history
- Support for position reconstruction
- Audit trail capabilities

3.3.1 Position Consolidation

The system aggregates multiple transactions per stock symbol:

- Total position size (sum of transaction quantities)
- Weighted average cost basis across purchases
- Real-time valuation using current market prices
- Unrealized profit/loss in absolute and percentage terms

Portfolio summary includes total portfolio value, aggregate profit/loss, and position count.

3.4 Technical Visualization

The platform generates 30-day price trend charts using Recharts [8]. The visualization processes historical data into time-series format with date labels and price scaling. Charts provide context for AI recommendations.

3.5 Access Control and Security

Authentication: JWT-based [7] token system with 24-hour expiration. Tokens encode user identity and role for stateless authentication.

Password Security: Bcrypt [10] hashing with work factor set to 12.

Authorization: Role-based access control (RBAC) enforced at API endpoint level.

Role Definitions:

- **Investors:** Access to personal portfolio and own analysis reports
- **Analysts:** Access to portfolios and reports of assigned investor accounts

Security Measures:

- SQL injection prevention through parameterized queries
- CORS configuration for cross-origin request control
- Input validation at API boundaries

4 Implementation Challenges

4.1 Financial Data Source Selection

Challenge: Finnhub API [4] free tier had restrictive rate limits.

Solution: Migrated to Yahoo Finance via yfinance library [3], providing comprehensive market data without restrictions.

4.2 AI Output Consistency

Challenge: Early iterations produced inconsistent outputs (e.g., SELL recommendations with 20% allocations).

Solution: Implemented structured prompt engineering with explicit rules and post-generation validation logic to correct mismatches.

4.3 Portfolio Aggregation

Challenge: Transaction-based storage created multiple database entries per stock, causing duplicate displays.

Solution: Developed SQL aggregation using GROUP BY with SUM for quantity totals and AVG for cost basis. Consolidated view shows single entry per stock.

4.4 Frontend State Management

Challenge: Complex interdependencies led to stale data displays when navigating between views.

Solution: Implemented centralized state management with useEffect hooks. Automatic data reloading triggers on view changes.

5 Performance Evaluation

5.1 System Performance

Table 1: Application Performance

Performance Metric	Measured Value
API Response Latency	< 2 seconds
AI Analysis Generation	3-5 seconds
Portfolio Data Load	< 1 second
Historical Data Retrieval	1-2 seconds
Chart Rendering	< 500ms

5.2 Implemented Features

The application implements:

- ✓ User authentication with role differentiation
- ✓ Real-time stock market data integration
- ✓ AI-powered analysis with recommendations
- ✓ 30-day price trend visualization
- ✓ Portfolio management with buy, sell, and remove operations
- ✓ Role-based access controls
- ✓ Historical analysis report storage
- ✓ Consolidated portfolio views with P/L tracking

5.3 Security Implementation

The system includes:

- Bcrypt password hashing (work factor: 12)
- JWT tokens (24-hour expiration)

- Role-based API endpoint protection
- Parameterized SQL queries
- CORS configuration
- Input validation

6 Potential Future Enhancements

6.1 Technical Improvements

- Component modularization for better code organization
- Testing suite implementation
- Enhanced input validation
- Loading state indicators
- Mobile-responsive design improvements

6.2 Feature Additions

- Additional technical indicators (RSI, MACD, Bollinger Bands)
- Watchlist functionality
- Price alert system
- Portfolio performance analytics
- News feed integration
- Report export functionality (CSV, PDF)

6.3 Scalability Considerations

For production deployment, the following could be considered:

- Migration from SQLite to PostgreSQL
- Caching layer for market data
- WebSocket for real-time updates
- Load balancing for multiple users
- Containerization for deployment

7 Conclusion

This project implements a full-stack financial analysis platform integrating web technologies, market data APIs, and generative AI. The system addresses key challenges: data reliability, AI consistency, security, and user experience.

Technical achievements include Yahoo Finance integration for market data, structured prompt engineering for consistent AI recommendations, transaction-based portfolio architecture, and role-based security model.

The implementation provides a functional foundation for stock analysis and portfolio management, with identified pathways for future enhancement and production deployment.

References

[1] FastAPI Framework, *FastAPI - Modern Web Framework for Python*, https://fastapi.tiangolo.com/, Accessed: 2025.

[2] React Team, *React - A JavaScript Library for Building User Interfaces*, https://react.dev/, Accessed: 2025.

[3] Ran Aroussi, *yfinance - Yahoo Finance Market Data Downloader*, https://pypi.org/project/yfinance/, Accessed: 2025.

[4] Finnhub, *Finnhub Stock API - Real-Time Stock Market Data*, https://finnhub.io/, Accessed: 2025.

[5] Google, *Gemini API - Google's Generative AI*, https://ai.google.dev/, Accessed: 2025.

[6] Anthropic, *Claude AI - AI Assistant by Anthropic*, https://www.anthropic.com/claude, Accessed: 2025.

[7] Auth0, *JSON Web Tokens (JWT) - Introduction*, https://jwt.io/introduction, Accessed: 2025.

[8] Recharts, *Recharts - A Composable Charting Library*, https://recharts.org/, Accessed: 2025.

[9] SQLite, *SQLite - SQL Database Engine*, https://www.sqlite.org/, Accessed: 2025.

[10] Niels Provos and David Mazières, *A Future-Adaptable Password Scheme*, USENIX Annual Technical Conference, 1999.