

Subject: Software Engineering on Web Application

Group report

Group name: Group 2

Member: Juncheng Zhao, Yiwei Li, Siyi Chen

# Content

<b>1. Introduction</b>	<b>3</b>
<b>2. System Architecture</b>	<b>3</b>
2.1 Distributed Design	3
2.2 Frontend Components	3
2.3 Backend Configuration	3
2.4 Database Schema	4
2.5 API Interfaces	5
<b>3. User Tasks &amp; Key Functionality</b>	<b>6</b>
3.1 Account Management	6
3.1.2 Change Password	7
3.1.3 Deposit Funds	8
3.2 Stock Trading Management	9
3.2.1 Query Single Stock Information	9
3.2.2 Execute Buy/Sell orders with AI device	9
3.2.3 Multi-stock Portfolio Trading with recommendations	10
3.3 Investment Advice generated by AI	10
3.3.1 Single stock analysis advice	10
3.3.2 User specific advice	11
3.3.3 Multiple stock portfolio advice	12
3.4 Module Parts	14
3.4.1 Single Stock Module	14
3.4.2 Multiple Stock Module	14
<b>4. Data Sources &amp; Acquisition</b>	<b>14</b>
<b>5. AI Agent Role</b>	<b>15</b>
5.1 Core Components	15
5.2 Supporting Modules	15
5.3 Key Functionalities	16
5.4 Data Management	16
5.5 System Integration	16
5.6 Security Features	16
5.7 Suggested Improvements	17
5.8 Mock data analysis	17
<b>6. Technical Details</b>	<b>19</b>
<b>7. System Testing</b>	<b>20</b>
7.1 User information module	20
7.2 Single Stock Module	22
7.3 Multiple stock module	25
<b>8. Contributions</b>	<b>26</b>

# 1. Introduction

This report details the design and implementation of a stock trading and forecasting platform that supports basic trading operations and delivers personalized investment advice via an integrated AI. The platform combines historical market data with user-specific information to transform natural language queries into structured requests and generate intelligent responses.

## Demo:

- Single Stock & Portfolio Forecasting: [Link](#)
- AI Agent Interactive Demo: [Link](#)

## 2. System Architecture

### 2.1 Distributed Design

1. **Frontend:** React/HTML/CSS/JavaScript interfaces for user workflows.
2. **Backend:** Node.js with Express.js handles RESTful APIs, business logic, and AI integration.
3. **Database:** MySQL stores all core data (user profiles, transactions, market data).

### 2.2 Frontend Components

1. Login/Registration Dashboard
2. User Information Management Dashboard
3. Stock Trading Dashboard
4. Investment Analysis Dashboard
5. AI Investment Advisory Dashboard

### 2.3 Backend Configuration

**Server Setup:** Express application with route management and middleware (CORS, body-parser).

**API Design:** RESTful endpoints with request validation and standardized JSON responses.

## 2.4 Database Schema

**User table**

Column name	Type	Description
email	VARCHAR(255)	PRIMARY KEY, Store users' email
password	VARCHAR(255) NOT NULL	Store user password
balance	DECIMAL(10, 2) DEFAULT 0.00	Store user balance in profile

**Stock transaction table**

Column name	Type	Description
timestamp	TIMESTAMP DEFAULT CURRENT_TIMESTAMP PRIMARY KEY,	To store the timestamp of every transactions. Avoid primary key conflict
email	VARCHAR(255) NOT NULL,	Store the user's email of the transaction
stock_name	VARCHAR(255)	
number	INT	Store the number of stock user bought in this transaction
current_price	DECIMAL(10, 2) NOT NULL,	
is_sold	BOOLEAN DEFAULT FALSE	It will change to 0 if the user sold all of the stock

## 2.5 API Interfaces

Function	API	Http method
User login	/api/login	post
User register	/api/register	post
Change password	/api/change-password	put
Deposit money	/api/deposit	post
Sell specific stock	/api/sell-stock	post
See stock in purchase	/api/active-stocks	get
Give advice	/ge	get
Give comparative analysis across stocks based on users' preference	/api/portfolio-recommendation	get
Give specific proportion advice on stocks	/api/multiplestock-analysis	get
AI provide personal advice	/api/ai-personal-advice	post
AI optimize portfolio weight	/api/ai-predict	post
AI stock Q&A	/api/stock-qa	post
AI chart analyze	/api/analyze-chart	post

## 3. User Tasks & Key Functionality

### 3.1 Account Management

#### 3.1.1 Registration, Login

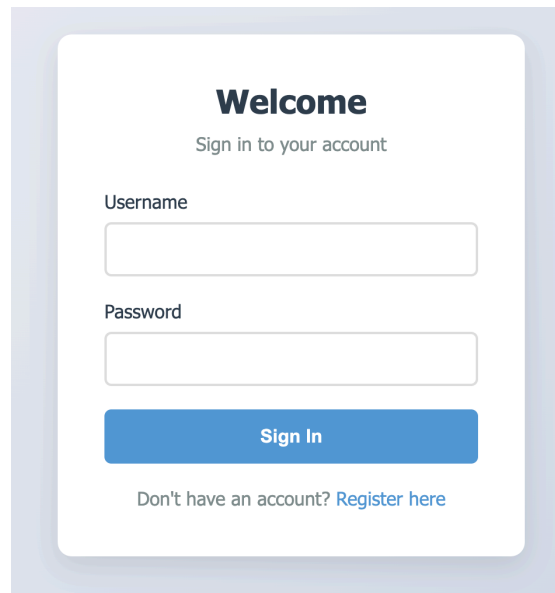
A login page UI mockup. It features a light blue gradient background. In the center is a white rounded rectangle containing the text 'Welcome' in bold, followed by 'Sign in to your account'. Below this are two input fields: 'Username' and 'Password'. A blue 'Sign In' button is positioned below the password field. At the bottom, there is a link that says 'Don't have an account? Register here'.

Fig. 3-1 Login page

The user registration feature is implemented through a combination of front-end and back-end components, ensuring both usability and security.

On the front end, the `register.html` page provides a user-friendly registration form that includes input fields for username, password, and password confirmation. It employs modern UI design principles, such as gradient backgrounds and shadow effects, to enhance visual appeal and user experience. The form includes real-time password matching validation, and if the passwords do not match or other errors occur (e.g., empty fields), appropriate error prompts are displayed to guide the user.

On the back end, the registration logic resides in `server/routes/register.js`. When a user submits the form, the server first checks whether the provided username already exists in the system. If not, the password is securely hashed using the `bcrypt` library to prevent plaintext storage, following industry best practices for data protection. The new user's information is then inserted into the `Users` table in the MySQL database, completing the registration process.

This registration module ensures a secure and intuitive onboarding experience, with proper data validation on the front end and encrypted, persistent storage on the back end.

### 3.1.2 Change Password

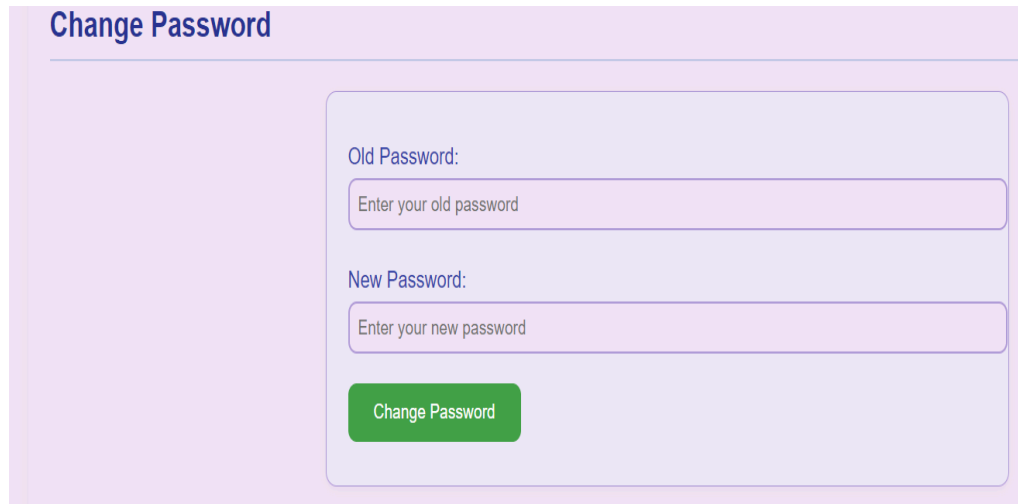
The image shows a web form titled "Change Password" in a purple header. The form itself is a light purple rounded rectangle containing two input fields. The first field is labeled "Old Password:" and has a placeholder text "Enter your old password". The second field is labeled "New Password:" and has a placeholder text "Enter your new password". Below these fields is a green button with the text "Change Password".

Fig. 3-2 Change password page

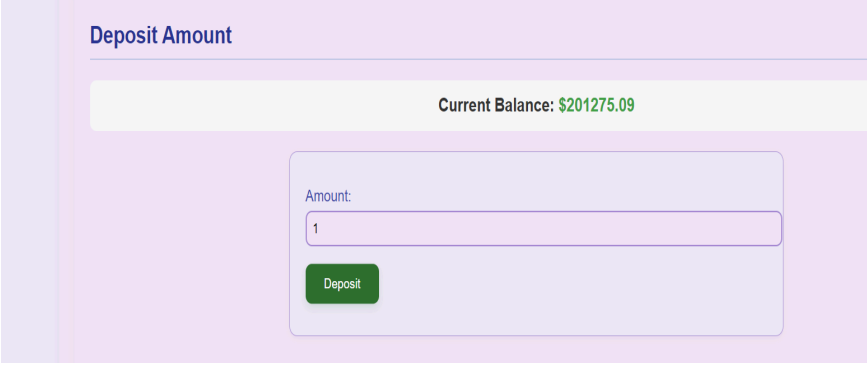
The password modification feature is carefully designed with both front-end usability and back-end security in mind.

On the front end, the `userInfo.html` page includes a password change form that requires users to enter both their current (old) password and a new password. The interface is user-friendly and provides real-time validation feedback. Upon submission, the system displays success or failure messages based on the result of the password update process, helping users understand whether their request was processed correctly.

On the back end, the logic for handling password change requests is implemented in `server/routes/userManagement.js`. When a request is received, the server first verifies the correctness of the old password by comparing it with the hashed password stored in the database using `bcrypt`. It also checks whether the new password is different from the old one, preventing users from accidentally reusing the same password. Once validated, the new password is encrypted using `bcrypt` before updating the user's record in the MySQL database.

This feature ensures that password updates are secure, reliable, and user-friendly, protecting user accounts while providing a smooth experience during credential updates.

### 3.1.3 Deposit Funds



The screenshot displays a web interface for depositing funds. At the top, a purple header bar contains the text "Deposit Amount". Below this, a light green box shows the "Current Balance: \$201275.09". The main area features a light purple box with a label "Amount:" and a text input field containing the number "1". A green "Deposit" button is positioned below the input field.

Fig. 3-3 Current balance

The deposit feature allows users to securely add funds to their account, with a well-coordinated front-end interface and robust back-end processing.

On the front end, the `userInfo.html` page includes a deposit form where users can view their current account balance and enter the desired deposit amount. The interface is designed to be intuitive, with clear input fields and responsive design. After submission, the system provides real-time feedback, displaying success or failure prompts to inform users whether the transaction was completed successfully.

On the back end, deposit requests are handled in `server/routes/userManagement.js`. When a request is received, the server first verifies whether the user exists in the system. If the verification passes, it proceeds to update the user's balance in the MySQL database by adding the specified amount. The server then returns a status response and message, which is used by the front end to display the appropriate prompt to the user.

This feature ensures a smooth and secure deposit process by combining clear front-end interaction with reliable back-end validation and database updating.



## 3.2 Stock Trading Management

### 3.2.1 Query Single Stock Information (real-time price, SMA/LMA)

In the backend module `server/routes/buyAndViewStocks.js`, the function `fetchRealTimeStockPrice` is responsible for retrieving the current stock price from external sources or APIs, ensuring that all trading operations and analyses are based on up-to-date market information.

Additionally, the module performs technical analysis by calculating Simple Moving Averages (SMA) and Long-Term Moving Averages (LMA) using historical price data. These moving averages are computed over different time periods to identify trends and provide key indicators for investment decisions. For example, a short-term SMA crossing above a long-term LMA may suggest a buy signal, while the reverse may indicate a potential sell opportunity.

Together, the integration of real-time pricing with moving average calculations provides users with accurate and timely technical analysis, forming the foundation for strategy generation and informed trading recommendations.



Fig. 3-4 Stock analysis chart

### 3.2.2 Execute Buy/Sell orders with AI device

The system's trading logic is divided into Buy, Sell, and Advice functionalities, each handled by specific server-side modules to ensure accurate and secure stock operations.

The Buy function, located in `server/routes/buyAndViewStocks.js`, begins by verifying whether the user's account balance is sufficient for the intended purchase. It then retrieves real-time stock prices, records the transaction in the database, and updates the user's remaining balance accordingly to reflect the purchase.

The Sell function, defined in `server/routes/sellStock.js`, first checks if the user holds enough shares of the specified stock. After retrieving the current market price, it calculates the proceeds from the sale, updates the user's balance, and modifies their stock holdings in the database to reflect the sale.

The Operation Suggestion module, implemented in `server/utls/singleAdviceFunction.js`, contains the `generateStrategy` function, which formulates trading strategies based on technical indicators and real-time market data. This module not only suggests buy/sell/hold actions but also incorporates risk management techniques such as stop-loss recommendations and dynamic position sizing to align with the user's investment profile.

Together, these components form a coherent backend system that manages transactions securely, updates financial records accurately, and provides intelligent, risk-aware investment suggestions.

### 3.2.3 Multi-stock Portfolio Trading with recommendations

- The optimal weighting `calculatePortfolioWeights` function calculation
- Consider the following factors: Risk tolerance capacity, Investment period, Stock correlation, Volatility, Sharpe ratio, Risk Analysis
- Calculate multiple risk indicators: Volatility, Max Drawdown, Value at Risk (VaR), Conditional Value at Risk (CVaR), Dynamic stop-loss price

For the recommendations generated by AI, It will provide the following advice consider:

Market analysis, Risk assessment, Technical analysis, Investment strategy suggestions, Data storage and management, Store using the MySQL database, User information, Transaction record, Stock data,

## 3.3 Investment Advice generated by AI

### 3.3.1 Single stock analysis advice

The *Single Stock Analysis Advice* module provides personalized investment recommendations based on user input, including stock code, investment horizon, and initial capital. It integrates historical and real-time stock data to perform technical analysis—calculating moving averages, volatility, ATR, maximum drawdown, and both VaR and CVaR for risk assessment. Based on these insights and the user's risk tolerance, it dynamically determines stop-loss prices, position sizing, and an overall risk score. Finally, it generates tailored trading strategies (buy/sell/hold), suggests the number of shares to purchase, and recommends an appropriate trading frequency.

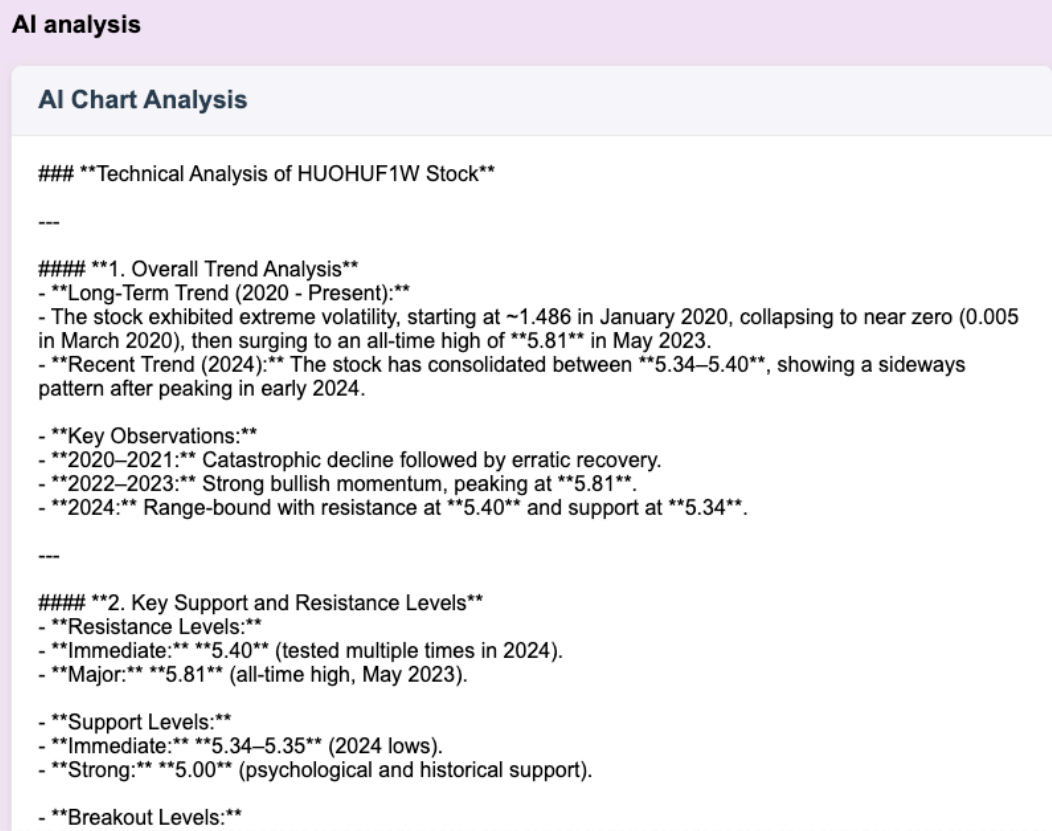


Fig. 3-5 AI analysis chart suggestions

### 3.3.2 User specific advice

The AI Personal Advice module leverages users' stock transaction records and account data to infer investment behavior and risk preferences, providing personalized trading and portfolio recommendations. It analyzes trading frequency, holding periods, and trade sizes to determine the user's investment style, and offers tailored advice including buy/sell actions, risk alerts, and fund management strategies. Future enhancements include integrating more user-specific information—such as risk tolerance, investment goals, experience level, and sector preferences—as well as incorporating macroeconomic and industry trends into the analysis. A feedback loop will allow the system to track user responses and outcomes, continuously refining its recommendations to deliver highly customized and adaptive investment guidance.

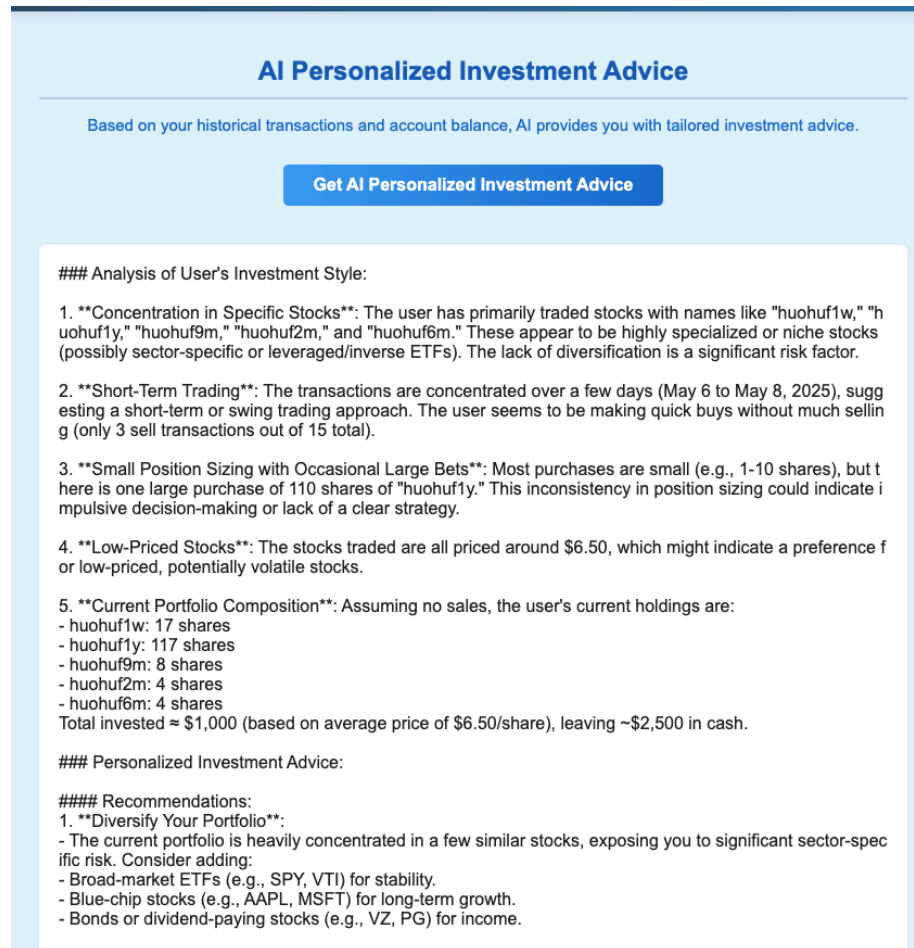


Fig. 3-6 AI analysis

### 3.3.3 Multiple stock portfolio advice

The User Specific Advice module delivers investment recommendations tailored to individual user profiles, factoring in their current portfolio holdings, investment horizon, risk tolerance, and personal financial goals. For example, users with a low risk appetite and capital preservation objectives may receive conservative allocation advice, while long-term, high-risk investors might be guided toward growth-oriented assets or sectors. This feature ensures a personalized, goal-aligned strategy for diverse investor types.

The Multiple Stock Portfolio Advice module is designed for users seeking diversification across multiple stocks. It starts by collecting selected stocks and retrieving their historical price data, as well as relevant market indices. A correlation matrix is computed to identify pairs of stocks with high, low, or negative correlations, which is essential for building a diversified portfolio. During optimization, the system

applies traditional methods such as mean-variance optimization and Sharpe ratio-based weighting to derive the optimal asset allocation. Additionally, AI-based models enhance the process by factoring in broader market trends and risk elements to dynamically predict optimal weights. The system adjusts strategies based on investment horizon—for instance, recommending high-correlation portfolios for long-term stability and low-correlation portfolios for short-term risk mitigation. Final recommendations include weight allocations, rebalancing strategies, and stop-loss guidelines. All functionalities are delivered via RESTful APIs, allowing the front end to fetch and display results seamlessly. A robust fallback mechanism ensures that traditional optimization will be used when AI predictions are unavailable, maintaining system reliability.

### Multiple Stock Portfolio Analysis

Select Stocks

☐ huohuf1w

☐ huohuf1y

☐ huohuf2m

☐ huohuf3m

☐ huohuf6m

☐ huohuf9m

☐ plbpln1m

☐ plbpln3m

☐ plbpln6m

☐ plopln1m

☐ plopln1w

☐ plopln1y

☐ plopln3m

☐ plopln6m

☐ plopln9m

Multiple Stock Analysis Suggestion

Investment Horizon:

1 year

Number of stocks to invest in:

4

Get Stock Recommendations

Analyze Portfolio Weights

Get AI-Optimized Weights

Compare Methods

Fig. 3-7 AI analysis by multiple stocks

STOCK ANALYSIS

Single Stock

Multiple Stocks

User Info & AI Agent

Logout

Select Stocks

☐ huohuf1w

☐ huohuf1y

☐ huohuf2m

☐ huohuf3m

☐ huohuf6m

☒ huohuf9m

☒ plbpln1m

☐ plbpln3m

☐ plbpln6m

☐ plopln1m

☒ plopln1w

☐ plopln1y

☐ plopln3m

☐ plopln6m

☒ plopln9m

Multiple Stock Analysis Suggestion

Investment Horizon:

1 year

Number of stocks to invest in:

4

Get Stock Recommendations

Analyze Portfolio Weights

Get AI-Optimized Weights

Compare Methods

Stock	Weight	Stock	Weight
huohuf9m	42.00%	huohuf9m	44.44%
plbpln1m	2.87%	plbpln1m	5.56%
plopln1w	45.33%	plopln1w	44.44%
plopln9m	9.80%	plopln9m	5.56%

Fig. 3-8 AI analysis by multiple stocks weight adjustment

## 3.4 Module Parts

### 3.4.1 Single Stock Module

This module delivers personalized investment advice based on a user's input of a stock ticker and available capital. It begins by retrieving the real-time stock price and accessing historical price data to calculate technical indicators such as short-term and long-term moving averages (SMA/LMA), volatility, Average True Range (ATR), maximum drawdown, Value at Risk (VaR), and Conditional VaR (CVaR). These indicators are synthesized into a structured AI prompt, which is sent to the backend's strategy generation function or AI model to produce actionable recommendations—including buy/sell signals, stop-loss levels, suggested trade frequency, and position sizing. The frontend presents both analytical charts and textual insights, helping users interpret market conditions and make informed decisions.

### 3.4.2 Multiple Stock Module

This module supports users in building and optimizing a diversified portfolio based on a list of stock tickers and a specified investment horizon. The system analyzes each stock's historical return, risk, and Sharpe ratio, and computes a correlation matrix to identify highly correlated, low-correlated, and negatively correlated stock pairs. Initial portfolio weights are assigned using mean-variance optimization adjusted by risk and Sharpe metrics. The system then refines these weights through correlation-based adjustments and normalization. For enhanced optimization, it calls the [/api/ai-predict](#) endpoint to apply AI-driven weight prediction that accounts for market trends and broader risk factors. The final output includes tailored weight allocations, rebalancing recommendations, and risk control measures, all adjusted to align with the user's investment duration and profile.

## 4. Data Sources & Acquisition

The system integrates multiple data sources to support real-time and historical analysis, user account management, and transaction tracking. For real-time market data, it uses the Stooq API via RESTful requests to fetch current stock prices, trading volume, and percentage price changes, ensuring up-to-date information for trading and analysis. For historical stock data, the system combines CSV file ingestion using csv-parser with data from third-party APIs, allowing it to build comprehensive time-series datasets. All relevant data—including parsed historical records—is stored in a MySQL database for efficient access and analysis. In parallel, user-related

data such as registration details, account balances, and transaction logs are also managed in MySQL and are updated in real time upon every user action, ensuring system consistency, accuracy, and responsiveness in both frontend display and backend logic.

## 5. AI Agent Role

### 5.1 Core Components

The AI trading agent is built on four key modules, each responsible for a distinct layer of system intelligence and execution. The `agent.py` module serves as the central controller, implementing the agent's trading logic including decision-making based on market analysis, risk assessment, and actual order execution. It also manages the agent's cash balance and stock portfolio, ensuring consistency between decisions and financial state. The `stock.py` module provides object-oriented management of individual stocks, storing their basic information, updating real-time prices, logging transactions, and producing periodic financial summaries for analysis. The `record.py` module handles persistent logging of all trading actions—timestamps, stock names, quantities, and prices—while also tracking price trends over time and allowing export to Excel for offline review. Finally, `secretary.py` acts as the AI interface layer, managing communication with AI models, validating whether a decision aligns with user goals or risk settings, and even handling advanced features like loan request management or AI-influenced

### 5.2 Supporting Modules

The supporting modules act as the backbone that enhances maintainability, usability, and modularization. `util.py` offers a set of reusable utility functions such as time formatting, configuration loading, data conversion (e.g., between JSON and DataFrame), and system-wide logging to facilitate debugging and traceability. The `templates/` directory contains the front-end layout files used in the web interface, enabling rendering of stock charts, trading dashboards, portfolio summaries, and alert messages. Meanwhile, `prompt/` holds a set of pre-written AI prompt templates that ensure consistency and domain relevance when generating requests to the AI model. These templates guide the model in generating investment advice, calculating risk, or deciding between buy/sell/hold actions based on contextual input.

## 5.3 Key Functionalities

The system provides a full suite of intelligent trading functions. For decision-making, it uses AI-generated strategies based on real-time and historical market data, taking into account price trends, moving averages, and sentiment signals. In risk control, it features dynamic stop-loss calculation, capital allocation strategies, and adjustable position sizing based on volatility. Market analysis capabilities include short- and long-term trend detection, volume and volatility analytics, and computation of key technical indicators like SMA, RSI, and ATR. Moreover, the system ensures high traceability with detailed trade logs, performance analysis tools, risk reports, and exportable investment summaries, supporting both real-time feedback and long-term auditability.

## 5.4 Data Management

Data is structured into two primary categories for clear separation of concerns: trading data and agent data. Trading data includes granular details such as order timestamps, stock symbols, executed prices, volumes, and derived technical metrics. This dataset supports accurate performance evaluation and historical replays. Agent data, on the other hand, tracks portfolio states, capital availability, active strategies, trade frequency, and long-term KPIs (e.g., win rate, max drawdown). These two layers of data are stored in a centralized MySQL database and are updated in real time to ensure synchronization between the user interface, AI decision logic, and backend storage.

## 5.5 System Integration

The AI agent is designed to operate seamlessly within a larger ecosystem through RESTful APIs. These interfaces allow external modules—such as user dashboards, transaction processors, or notification systems—to interact with the agent by querying its state, triggering trades, or receiving updates. The system also implements centralized logging and real-time monitoring for every transaction and AI decision. This allows administrators to detect anomalies, measure performance metrics, and debug errors through structured logs. Error-handling and rollback mechanisms further ensure transactional integrity and graceful degradation in case of system failure or API failure.

## 5.6 Security Features

Security in the AI agent is multi-layered. It begins with validation of every trade: the system checks account balance, current holdings, and risk limits (e.g., maximum allocation per stock). These checks



ensure users cannot over-leverage or take unbounded risk. Furthermore, the system includes exception-handling wrappers around key operations, so that failures—whether from external APIs, model errors, or database faults—do not crash the system. Instead, fallback logic or alert mechanisms are activated. Backup systems also periodically store transaction and portfolio snapshots to prevent data loss and enable recovery.

## 5.7 Suggested Improvements

To further elevate the system's performance and intelligence, several enhancements are proposed. First, integrating additional market indicators (e.g., GDP, inflation, sector rotation) would enhance macro-level decision support. Second, refining the AI engine with techniques like ensemble learning, reinforcement learning, or temporal attention models could boost adaptability and accuracy. Risk modeling could be strengthened by introducing advanced simulations such as Monte Carlo or Bayesian estimations. For large-scale deployment, performance monitoring dashboards and batch processing pipelines are recommended. Lastly, offering user-customizable AI strategies, risk thresholds, and investment goals would maximize personalization and system flexibility.

## 5.8 Mock data analysis

The figure 5-2, presents a comprehensive simulation of an AI-driven stock trading system, showcasing both transactional actions and interpretability through live event tracking and agent-generated commentary. This image displays forum-style recaps generated by two AI trading agents (Agent 0 and Agent 1) at the end of Days 1 and 2, offering valuable insights into their decision rationales and investment philosophies. Agent 1 maintains a consistent conservative strategy across both days, emphasizing disciplined risk management, minimal leverage, and selective profit-taking. On Day 1, they sold 5,000 shares of Stock B at \$40.10 to reduce exposure amid regulatory uncertainty, while continuing to hold Stock A due to its stable fundamentals and leadership-driven turnaround potential. On Day 2, they doubled down on this approach by selling 10,000 shares of Stock B to further de-risk and reaffirmed their commitment to Stock A, awaiting clearer signs of recovery. The key takeaway emphasizes trimming speculative positions and focusing on long-term value. In contrast, Agent 0 exhibits a more balanced and moderately bullish approach. On Day 1, they rated Stock B (Tech) as a strong buy, highlighting +20% growth and government support, while holding off on Stock A (Chemicals) pending leadership improvements. By Day 2, their stance shifted slightly—still optimistic on Stock A due to revenue stabilization and cash flow, but more cautious on Stock B due to profit-taking concerns. They adjusted

their portfolio by trimming B and reallocating to A, while retaining cash for new opportunities. These posts showcase how each agent develops a narrative to explain actions, blending technical and fundamental reasoning with portfolio strategy, reflecting human-like trading personas under AI governance.

### Configure Simulation Parameters

API Configuration

DeepSeek API Key:

sk-844e8505444a4c80b0b251cd914d05e3

Caution: Avoid exposing sensitive keys in client-side code in production.

Model Name (for Agent & Secretary):

deepseek-reasoner

Basic Settings

Number of Agents:

20

Total Simulation Days:

180

Daily Trading Sessions:

3

Stock Initial Settings

Stock A Initial Price:

30

Stock B Initial Price:

40

Agent Initial Property

Max Initial Property:

5000000.0

Min Initial Property:

100000.0

Loan Settings

Loan Types (comma-separated names, e.g., one-month,two-month):

one-month,two-month,three-month

Loan Durations (comma-separated days, corresponding to types):

22,44,66

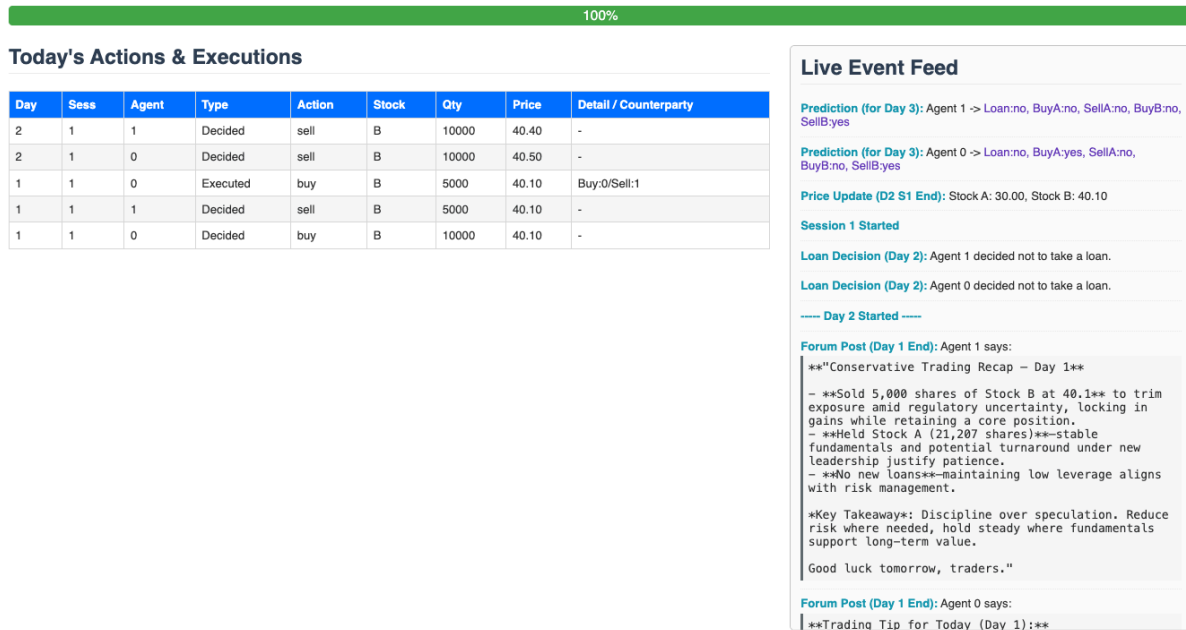
Must match the number of loan types. Enter positive integers.

Loan Rates (comma-separated, e.g., 0.027,0.03,0.033):

0.027,0.03,0.033

Must match the number of loan types. Enter nonfloats if possible.

Fig.5-1 AI agent configuration parameters page



Fig, 5-2 AI agent analysis page

## 6. Technical Details

The system adopts a modular architecture with a back-end built on the Express.js framework, utilizing RESTful APIs, router separation, and middleware for CORS, request parsing, and session management to ensure clarity and efficiency in server operations.

MySQL is used as the persistent storage layer, with connection pooling for performance optimization, and structured tables for users, transaction history, and market data.

The AI integration supports both OpenAI and DeepSeek models via asynchronous API calls, using structured prompt templates, role definitions, and context management for intelligent, explainable decision-making.

On the data-handling side, csv-parser is used for stream-based CSV processing, while Moment.js manages localized time formatting and data transformation.

The financial computation module includes advanced analytics such as volatility, beta, Sharpe ratio, maximum drawdown, VaR, and CVaR, along with portfolio optimization techniques like mean-variance analysis and correlation-based risk balancing.

The front end is built using HTML, CSS, and JavaScript with responsive design principles, and utilizes the Fetch API for real-time interaction with the backend, dynamically presenting charts and AI-driven advice.

## 7. System Testing

### 7.1 User information module

#### Register function

Sends a POST request with new user credentials. The response returns a success message and user info, confirming the registration API works correctly.

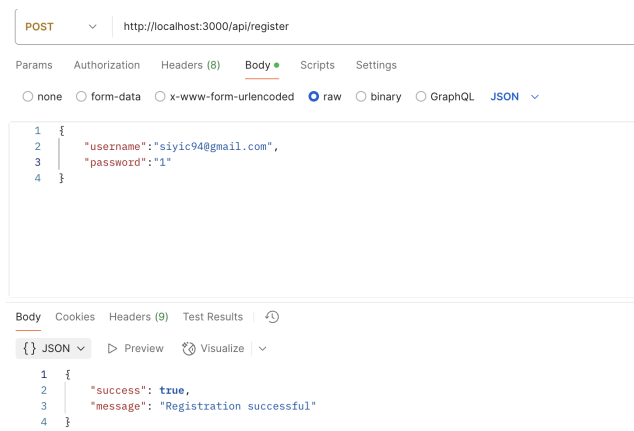


Fig. 7-1 Register function test postman test

#### Login function

Tests login using email and password. A successful response includes a token and user details, confirming correct authentication logic.

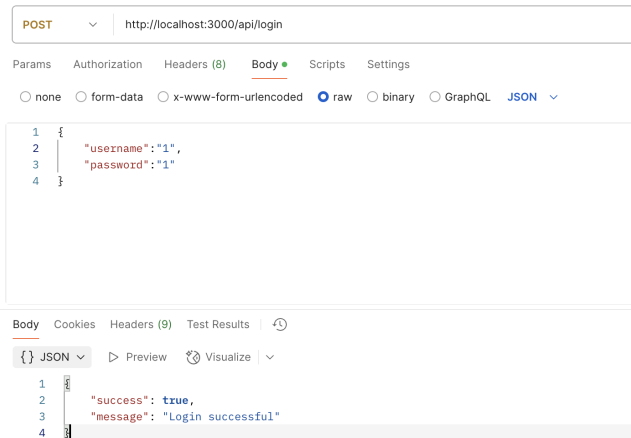


Fig. 7-2 Login function test postman test

## Reset password function

Submits old and new passwords to test the reset function. Status 200 confirms the password was successfully updated.

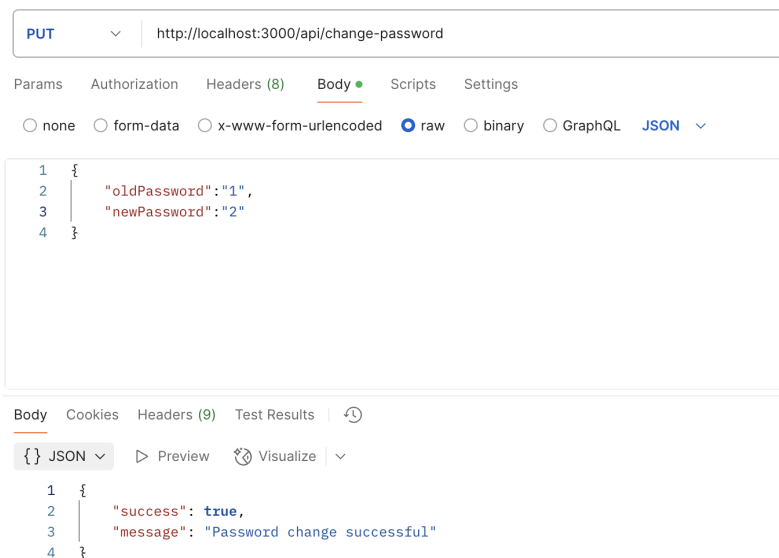


Fig. 7-3 Change password test postman test

## Deposit money test

Simulates account deposit by sending user ID and amount. Response shows updated balance, confirming deposit function works.

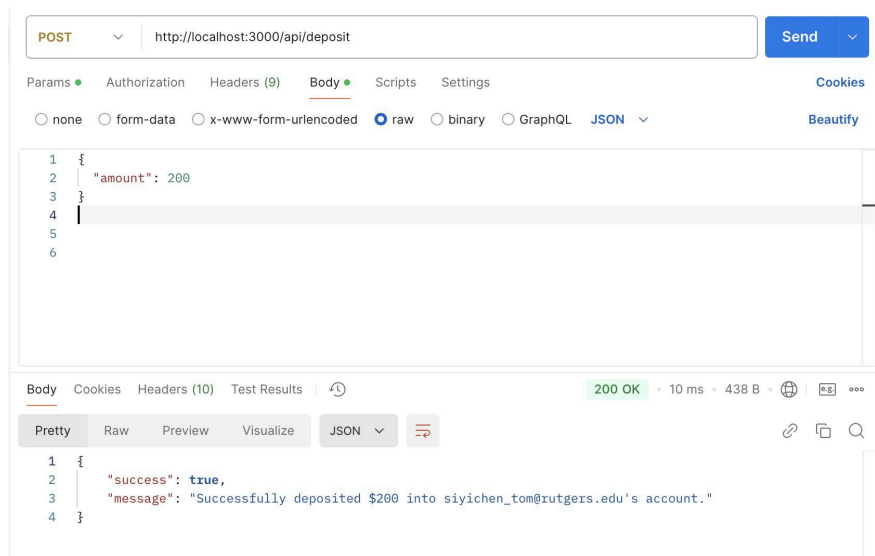


Fig. 7-4 Deposit money test postman test

## 7.2 Single Stock Module

### Purchase stocks

Sends a sell request with stock code and quantity. A valid response indicates successful purchase operation.

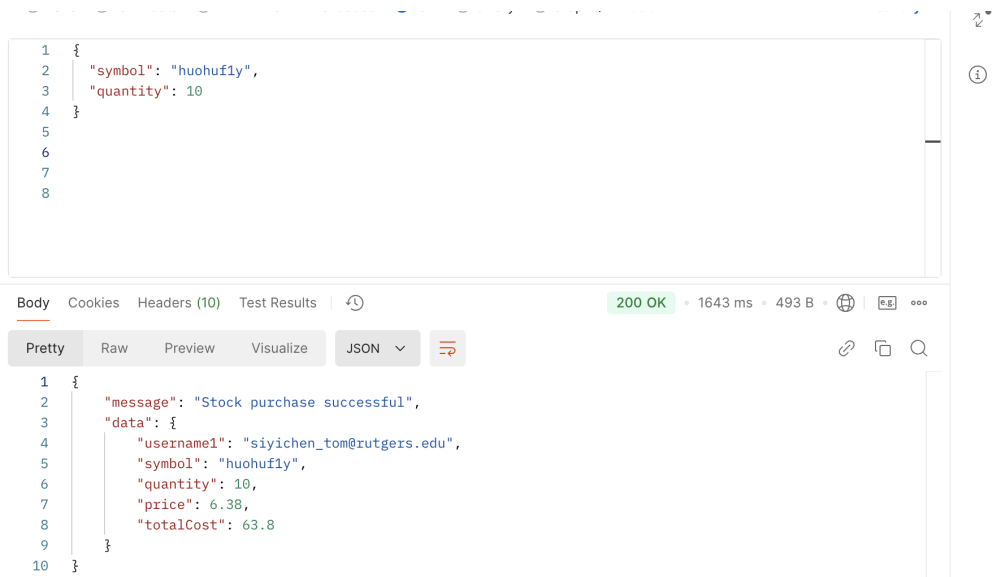


Fig. 7-5 Purchase stock test postman test

Single stock advice

Requests basic investment advice for a stock. The response includes analytical suggestions and key indicators.

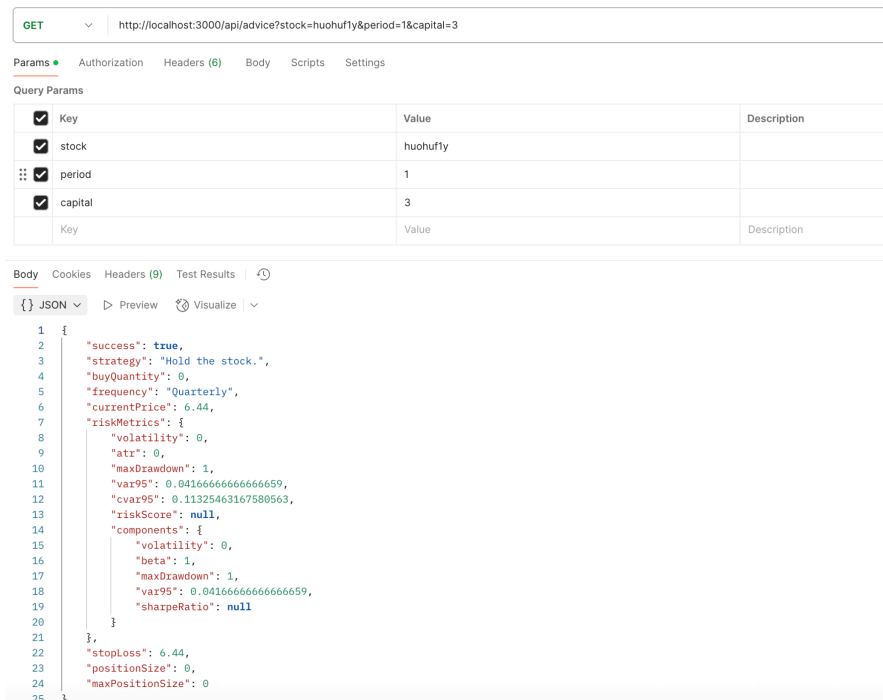


Fig. 7-6 Single stock advice postman test

Stock line chart

Retrieves historical price data for a stock to be used in chart visualization. Response includes time-series data.

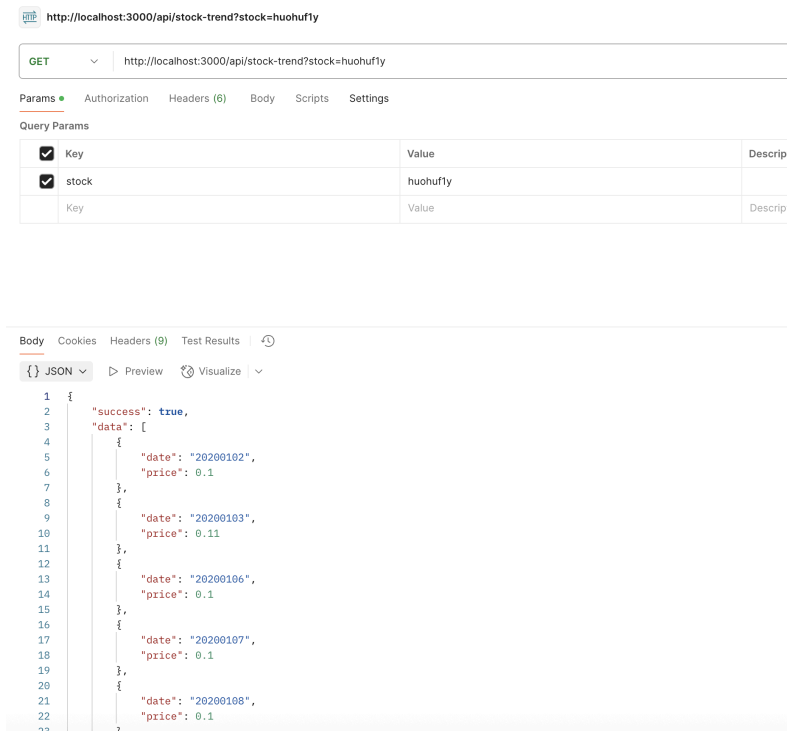


Fig. 7-7 Stock line chart postman test

## Single stock advice by AI

Calls the AI service to provide investment advice. The response includes generated natural language recommendations.

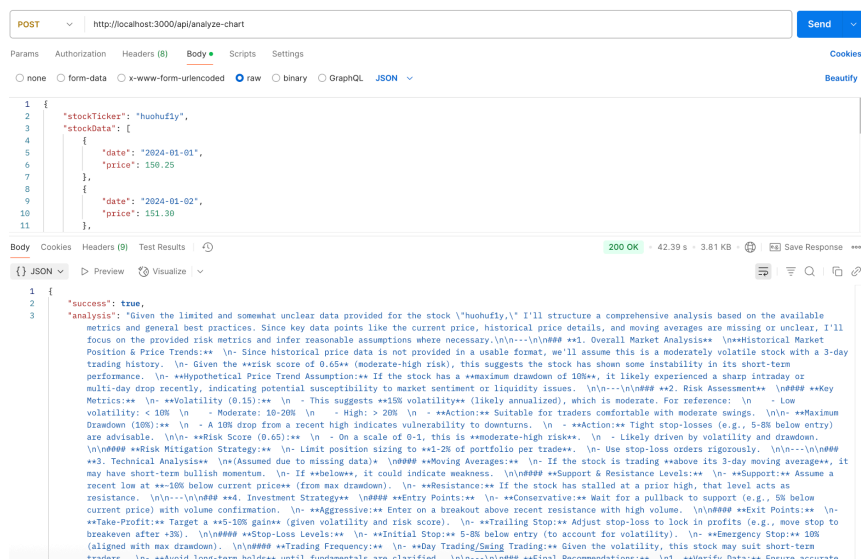




Fig. 7-8 Single stock advice by AI postman test

## 7.3 Multiple stock module

### Multiple stock recommendation function

Requests AI-based recommendations for multiple stocks. Response includes selected stock list and scores.

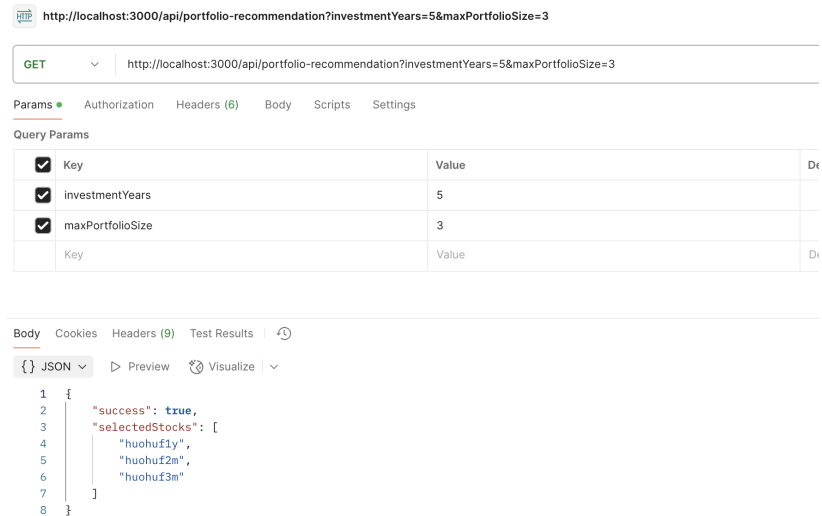


Fig. 7-9 Multiple stock recommendation function postman test

### AI weight adjustment

Submits stock risk-return data to get optimized weight distribution from AI. Response is a JSON array summing to 1.

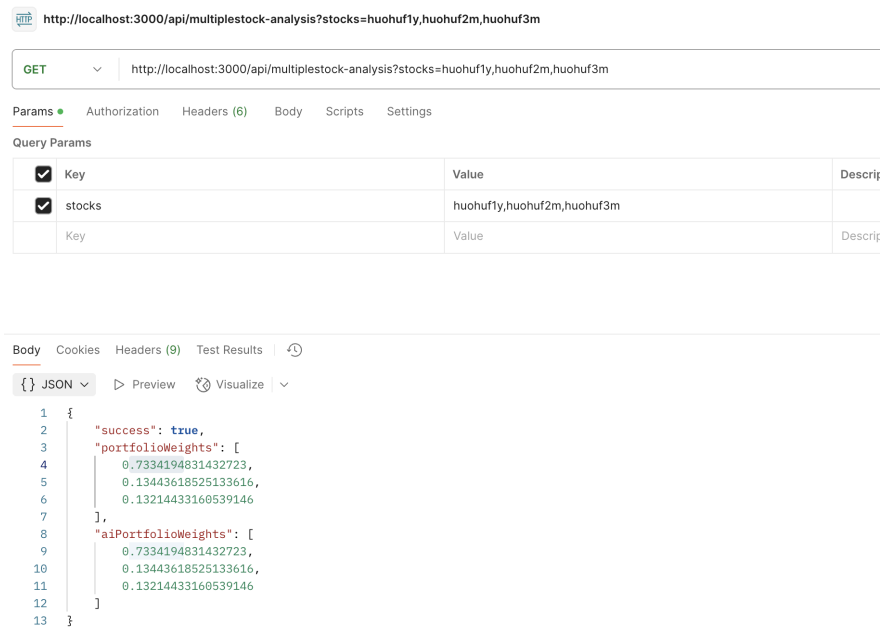


Fig. 7-10 AI weight adjustment postman test

## 8. Contributions

Group member	Responsibility
Juncheng Zhao	Frontend design; single & multiple stock modules;
Siyi Chen	Backend design (user info, DB operations, multiple stock module); frontend design;
Yiwei Li	AI agent design; single stock purchase & user info modules;