HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Secure Web Development

Stock Trading Platform

Instructors:	BUI TRONG TUNG	
	DO BA LAM	
Members:	BUI DUY ANH	20225563
	CHU TRUNG ANH	20225564
	PHAM MINH TIEN	20225555

Hanoi, 5-2025

ACKNOWLEDGMENT

The successful completion of this report would not have been possible without the invaluable contributions of several individuals and groups. I extend my heartfelt appreciation to my academic advisors, Mr Bui Trong Tung and Mr Do Ba Lam, for their expert guidance and constructive feedback throughout the project and writing process. Their support was instrumental in shaping this work.

Additionally, I am grateful for the unwavering support of my family and friends. Their encouragement and understanding were essential in maintaining my motivation throughout this project.

The project may still contain mistakes or misunderstandings. If you encounter any, please don't hesitate to share them with us. We are always open to feedback and eager to listen.

ABSTRACT

This project develops a full-stack stock trading simulation platform that creates a realistic, risk-free environment for users to learn and practice stock trading. The system implements core stock market mechanisms including continuous order matching, limit and market orders, price discovery, and portfolio management.

The platform is built using a modern technology stack with Node.js and Express on the backend, React on the frontend, and PostgreSQL for data persistence. It features a secure authentication system with multi-factor verification, real-time stock price visualization with candlestick charts, and an intuitive trading interface that closely mimics real-world trading platforms.

A key technical focus is the implementation of the order book matching engine that follows price-time priority rules, maintaining separate order books for each stock and dynamically calculating reference, ceiling, and floor prices. The system also simulates complete trading sessions and generates realistic market data.

Beyond functional requirements, the project incorporates robust security measures to protect against common web vulnerabilities and employs Docker containerization for consistent deployment across environments. The architecture emphasizes modularity and separation of concerns, making the system maintainable and extensible.

This educational tool provides a safe environment for users to develop trading skills without financial risk while gaining deeper insights into stock market mechanics and price formation dynamics.

CONTENTS

Chapter 1

Introduction

1 Motivation

The Stock Trading Platform project was motivated by the need to create an accessible, risk-free environment for individuals to learn and practice stock trading. Traditional stock markets present significant barriers to entry, including financial risk and complex terminology that can intimidate newcomers. By developing a simulation platform that accurately models real market mechanics, we aim to:

- Lower the learning barrier for individuals interested in stock trading by providing a safe environment to practice without financial consequences.
- Enhance financial literacy through practical experience with trading concepts and market dynamics.
- Demonstrate complex market mechanisms like order matching, price formation, and supply-demand interactions in an educational context.
- Provide realistic trading experience that mirrors actual stock exchanges like HOSE (Ho Chi Minh Stock Exchange).

2 Objective and Scope of the Project

For more detail, view this documentation: stockFundamentalTheory.md.

The project implements a comprehensive stock trading simulation with the following scope:

Core Trading Functionality

- Continuous Order Matching System using price-time priority mechanism.
- Support for Market and Limit Orders with proper execution logic.
- Real-time Order Book maintenance for each stock.
- Dynamic Price Calculation including reference, ceiling, and floor prices.

Market Simulation

- Simplified Trading Session based on HOSE (continuous matching session without ATO/ATC).
- Opening/Closing Price Determination based on first and last matched orders.
- 7% Price Fluctuation Limit for all stocks (ceiling/floor calculations).

User Experience

- Secure Account Management with multi-factor authentication.
- Portfolio Management for tracking holdings and performance.
- Interactive Order Placement interface.
- Visual Price Charts for market analysis.

Out of Scope

- Advanced order types (Stop Loss, Take Profit) are considered for future implementation.
- Multiple stock exchange simulation (focusing only on HOSE mechanics).
- ATO/ATC auction sessions (implementing continuous matching only).
- Specialized stock categories with different fluctuation limits.

The platform serves as both an educational tool for understanding stock market fundamentals and a practical sandbox for developing trading strategies without financial risk, making it valuable for students, educators, and aspiring investors alike.

Chapter 2

Technical Stack

The Stock Trading Platform employs a modern, industry-standard technology stack designed for scalability, performance, and security. Each technology was carefully selected to fulfill specific requirements of the application.

1 Frontend Technologies

The frontend of the Stock Trading Platform is built to deliver a highly responsive, intuitive, and dynamic user experience, leveraging modern JavaScript frameworks and libraries.

1.1 Core Framework and Build Tools

- **React v18.3.0**: As the foundational JavaScript library, React is employed for building the platform's user interface, enabling a component-based architecture that promotes reusability and maintainability. Its virtual DOM enhances rendering performance, crucial for dynamic trading environments.
- **Vite v6.2.0**: This modern frontend build tool and development server significantly accelerates the development workflow with features like lightning-fast cold server start and instant Hot Module Replacement (HMR). Vite optimizes the production build process for efficient deployment.
- **React Router v6.22.0**: Essential for client-side routing and navigation, React Router ensures a seamless single-page application experience, allowing users to navigate between different views (e.g., dashboard, stock details, portfolio) without full page reloads.

1.2 Package Management

- **Yarn**: Utilized as a fast, reliable, and secure dependency management tool, Yarn ensures consistent package installations across development environments. It helps manage project dependencies efficiently, contributing to faster build times and stable development cycles compared to other package managers.

1.3 UI Components and Styling

- **Material UI v7.1.0**: This comprehensive React component library implements Google's Material Design principles, providing a rich set of pre-built, customizable UI components. Material UI ensures visual consistency, accessibility, and reduces development time for the user interface.

- **Emotion**: Utilized as a CSS-in-JS library, Emotion facilitates writing dynamic and performant CSS within React components, enabling sophisticated styling based on application state.

- **@mui/x-charts v8.3.1**: Part of the Material UI ecosystem, this library provides robust charting and data visualization components, specifically tailored for displaying financial data effectively.

- **ApexCharts v4.7.0**: An advanced charting library, ApexCharts is integrated for generating interactive financial charts, including candlestick charts, which are critical for visualizing stock price movements and technical analysis.

- **Lucide-react v0.507.0**: This library provides a collection of lightweight and customizable icons, enhancing the visual clarity and navigation of the platform.

1.4 API and State Management

- **Axios v1.8.4**: A promise-based HTTP client, Axios is used for making asynchronous API requests from the frontend to the backend services, handling data fetching and submission efficiently.

- **React Context API**: Employed for global state management, the Context API (specifically with `AuthContext` and `TradingSessionContext`) allows for efficient data sharing across the component tree, avoiding prop drilling and centralizing critical application states.

1.5 Real-time Communication

- **ItemStatusActionSSE**: Server-Sent Events (SSE) are utilized for enabling real-time, one-way communication from the server to the client. This allows for instant updates regarding critical trading information, such as stock price movements, order execution status, and portfolio changes, without requiring the client to repeatedly poll the server. This ensures a dynamic and responsive trading experience.

2 Backend Technologies

The backend infrastructure is designed for robust performance, scalability, and secure data handling, built primarily on Node.js and a relational database.

2.1 Server Framework

- **Node.js**: A cross-platform, open-source JavaScript runtime environment built on Chrome's V8 JavaScript engine. Node.js is chosen for its high performance, non-blocking I/O model, and event-driven architecture, making it ideal for handling concurrent connections required by a real-time trading platform.
- **Express.js v4.21.2**: A minimalist and flexible web application framework for Node.js, Express.js provides a robust set of features for building RESTful APIs. It simplifies HTTP request handling, routing, middleware management, and cookie/session management, accelerating backend development by reducing complex configuration.

2.2 Database

- **PostgreSQL v16.1**: A powerful, open-source object-relational database system known for its high reliability, robust feature set, and strong adherence to SQL standards. PostgreSQL is extensively used for managing the platform's structured data, including user profiles, stock information, transactions, and portfolio holdings, ensuring data consistency and supporting complex queries.
- **pg v8.14.1**: This is the non-blocking PostgreSQL client for Node.js, providing the necessary interface for the backend to interact with the PostgreSQL database.

- **PgAdmin**: A leading open-source administration and development platform for PostgreSQL, PgAdmin is used for managing, monitoring, and developing the database schema and data.

2.3 Authentication and Core Security

- **bcrypt v5.1.1**: A robust password hashing library used for securely storing user passwords by employing a strong, adaptive hashing algorithm with salting, protecting against brute-force and rainbow table attacks.
- **jsonwebtoken (JWT) v9.0.2**: Implements JSON Web Tokens (JWT) for secure, stateless token-based authentication. JWTs are used to transmit information securely between parties as a compact, URL-safe means. A JWT consists of a Header, Payload, and Signature, all encoded into a single string. Upon successful login, the server issues a JWT to the client, which is then used for subsequent authenticated requests. This approach eliminates the need for server-side session storage, enhancing scalability.
- **Passport v0.7.0**: A flexible authentication middleware for Node.js, Passport.js is used for managing various authentication strategies.
- **passport-google-oauth20 v2.0.0**: A specific strategy for Passport.js, enabling secure integration with Google OAuth 2.0 for Single Sign-On (SSO), allowing users to authenticate via their Google accounts without needing to manage separate platform-specific credentials.
- **helmet v8.1.0**: A collection of Express.js middleware functions that set various HTTP headers to help protect the application from well-known web vulnerabilities, such as XSS, clickjacking, and other code injection attacks.
- **cors v2.8.5**: Middleware for enabling Cross-Origin Resource Sharing (CORS), allowing controlled access to resources from different origins while maintaining security.
- **cookie-parser v1.4.7**: Middleware for parsing incoming request cookies, essential for handling authentication tokens stored in HTTP-only cookies.

2.4 Input Validation and Sanitization

- **joi v17.13.3**: A powerful schema description language and data validator. Joi is used for robust server-side validation of incoming request data (e.g., user inputs, API parameters) to ensure data integrity and prevent malformed requests.
- **xss v1.0.15**: This library provides Cross-Site Scripting (XSS) prevention by sanitizing user-supplied input, filtering out potentially malicious scripts before they are stored or rendered.

2.5 Email Services

- **nodemailer v7.0.3**: An easy-to-use module for sending emails from Node.js applications, used for delivering one-time passwords (OTPs) and security notifications to users.
- **otp-generator v4.0.1**: A utility library for generating unique, time-based one-time passwords, crucial for multi-factor authentication implementation.

2.6 Payment/Banking Integration

- **Sepay**: A third-party service that acts as an intermediary between our website and the banking system. When clients scan the QR code, the payment is processed by the bank. The bank then notifies Sepay, which in turn notifies our application about the transaction (if using Sepay Webhooks). Alternatively, we can manually check the transaction by calling the Sepay API.

2.7 Logging and Monitoring

- **winston v3.17.0**: A versatile logging library for Node.js applications, Winston is used for structured logging across different levels (e.g., info, warn, error), aiding in debugging, monitoring, and auditing system behavior.

3 Deployment

The deployment strategy emphasizes containerization and cloud services for scalability, reliability, and security.

3.1 Containerization

- **Docker**: Utilized for application containerization, Docker packages the application along with all its dependencies into portable containers. This ensures consistent environments across development, testing, and production.
- **Docker Compose**: A tool for defining and running multi-container Docker applications, Docker Compose is used to orchestrate the platform's various services (frontend, backend, database) efficiently within a single defined environment.

3.2 Web Server and Reverse Proxy

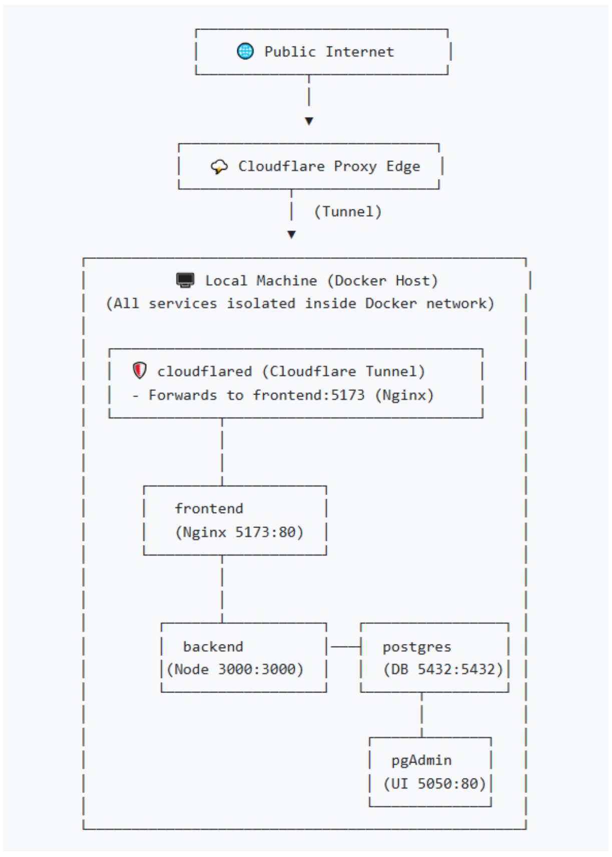
- **Nginx v1.25.3**: A high-performance HTTP and reverse proxy server. Nginx is configured to:
  - Serve static frontend assets efficiently.
  - Proxy API requests to the backend Node.js server.
  - Handle client-side routing support.
  - Provide load balancing across multiple backend instances for scalability.

3.3 Cloud Services & Infrastructure

- **Cloudflare**: A comprehensive web infrastructure and security company, Cloudflare is utilized for:
  - **Content Delivery Network (CDN)**: To cache content closer to users, reducing latency and improving loading times.
  - **DDoS Protection**: To mitigate Distributed Denial of Service attacks, ensuring platform availability.
  - **SSL/TLS Certificates**: Providing free, secure HTTPS connections for encrypted communication.
  - **Turnstile CAPTCHA service**: A privacy-preserving CAPTCHA service used to prevent automated bot interactions, particularly on login and registration forms, enhancing security without compromising user experience.
  - **Cloudflare Tunnels**: Used for securely connecting the origin server to the Cloudflare network without requiring public-facing IP addresses, providing an additional layer of security.

- **Web Application Firewall (WAF)**: Cloudflare's WAF provides an additional layer of security by filtering and monitoring HTTP traffic between the web application and the Internet. It protects against common web vulnerabilities such as SQL injection, cross-site scripting (XSS), and other OWASP Top 10 threats.

Here is our deployment diagram:


Public Internet
Cloudflare Proxy Edge
(Tunnel)
Local Machine (Docker Host)
(All services isolated inside Docker network)
cloudflared (Cloudflare Tunnel)
- Forwards to frontend:5173 (Nginx)
frontend
(Nginx 5173:80)
backend
(Node 3000:3000)
postgres
(DB 5432:5432)
pgAdmin
(UI 5050:80)

4 Code Quality and Testing Tools

Maintaining high code quality and ensuring application security are paramount, supported by dedicated static and dynamic analysis tools.

4.1 Static Code Analysis

- **Qodana**: JetBrains' code quality platform is used for automated static code analysis. Qodana performs continuous inspections to identify potential bugs, code smells, and security vulnerabilities early in the development lifecycle.

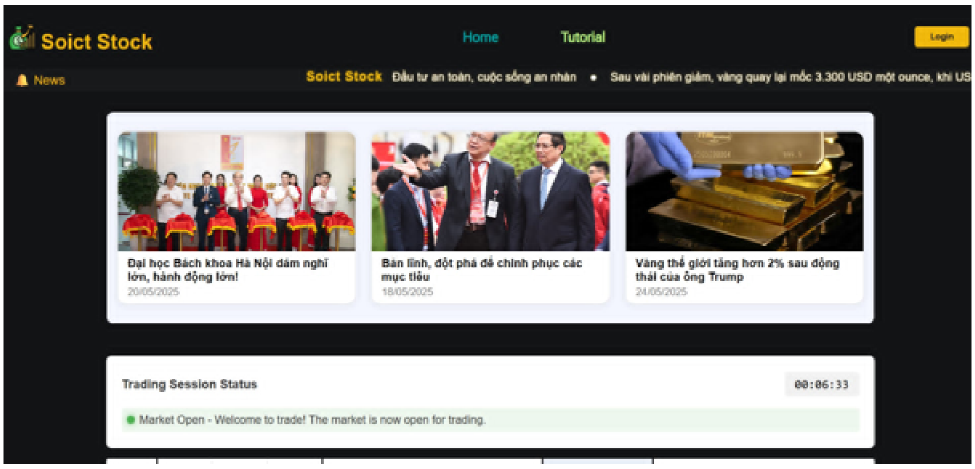
4.2 Dynamic Application Security Testing

- **OWASP ZAP (Zed Attack Proxy)**: This widely recognized open-source dynamic application security testing (DAST) tool is planned for integration. OWASP ZAP will be used to automatically and manually discover vulnerabilities in the running application, providing an essential layer of security testing, especially for penetration testing.

Chapter 3

UI/UX Design

This chapter will describe our UI/UX design and user manual. Normally, even when you are not logged in, you can still access the main page and the tutorial page. After logging in, you will have full access to our web application, which includes: Home, Tutorial, Trade, and Portfolio.



Soict Stock Đầu tư an toàn, cuộc sống an nhàn • Sau vài phiên giảm, vàng quay lại mốc 3.300 USD một ounce, khi US

Đại học Bách khoa Hà Nội dám nghĩ lớn, hành động lớn!
20/05/2025

Bản lĩnh, đột phá để chinh phục các mục tiêu
18/05/2025

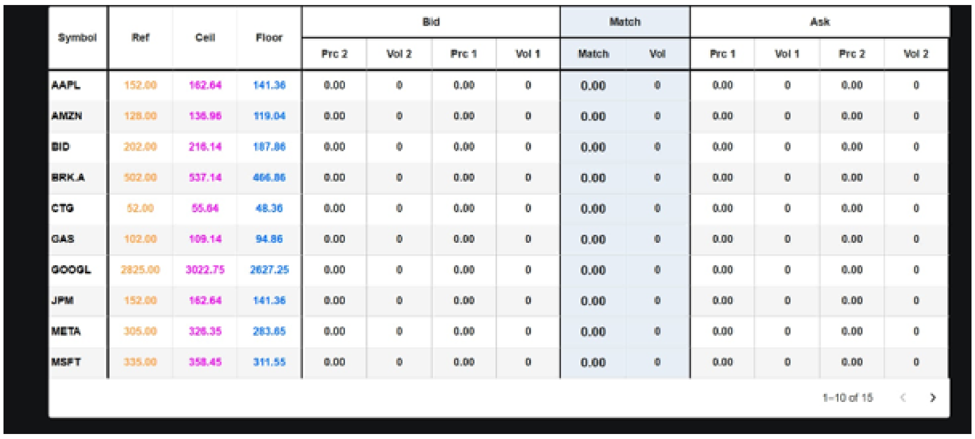
Vàng thế giới tăng hơn 2% sau động thái của ông Trump
24/05/2025

Trading Session Status 00:06:33

Market Open - Welcome to trade! The market is now open for trading.

Figure 3.1: Home page

The table in Home page presents real-time simulated stock market data for various companies, displaying key trading metrics in a structured format. Each row corresponds to a different stock symbol, such as AAPL, AMZN, GOOGL, and MSFT. For each stock, the table shows the reference price (Ref), ceiling price (Ceil), and floor price (Floor), which define the upper and lower bounds for trading that session. The table is divided into three main sections: Bid, Match, and Ask. The Bid section shows buy orders, including price levels and volumes. The Match section indicates the most recent matched price and volume. The Ask section lists current sell orders. At the bottom right, the table shows pagination with the current view displaying 1–10 of 15 entries.

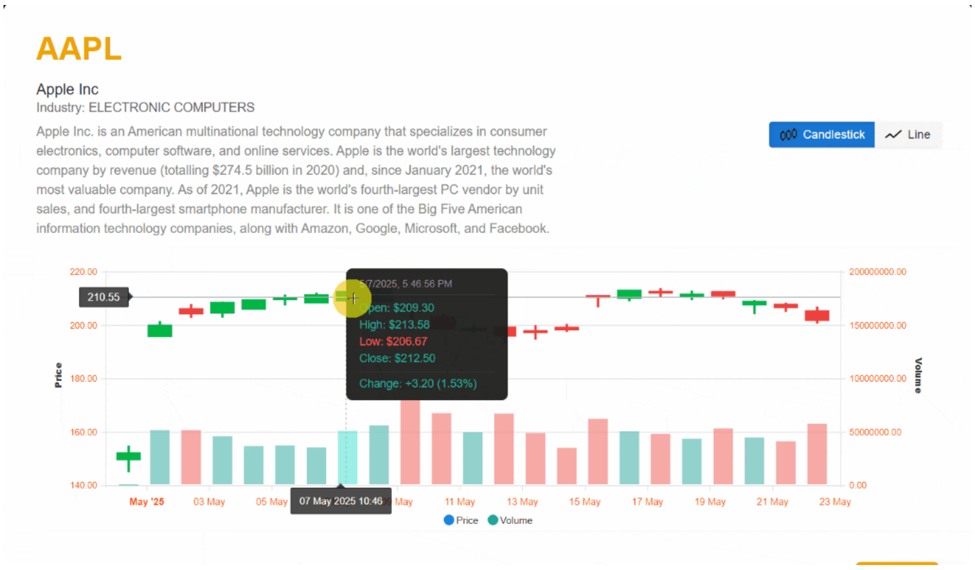


Symbol	Ref	Ceil	Floor	Bid				Match		Ask			
				Prc 2	Vol 2	Prc 1	Vol 1	Match	Vol	Prc 1	Vol 1	Prc 2	Vol 2
AAPL	152.00	162.64	141.36	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
AMZN	128.00	136.96	119.04	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
BID	202.00	216.14	187.86	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
BRK.A	502.00	537.14	466.86	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
CTG	52.00	55.64	48.36	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
GAS	102.00	109.14	94.86	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
GOOGL	2825.00	3022.75	2627.25	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
JPM	152.00	162.64	141.36	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
META	305.00	326.35	283.65	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0
MSFT	335.00	358.45	311.55	0.00	0	0.00	0	0.00	0	0.00	0	0.00	0

1–10 of 15

Figure 3.2: Trading Table in Home page

After clicking a company on the trading table, the chart will appear just below the trading table. The chart has full information about the company you just clicked on and their stock price for each day.



AAPL

Apple Inc

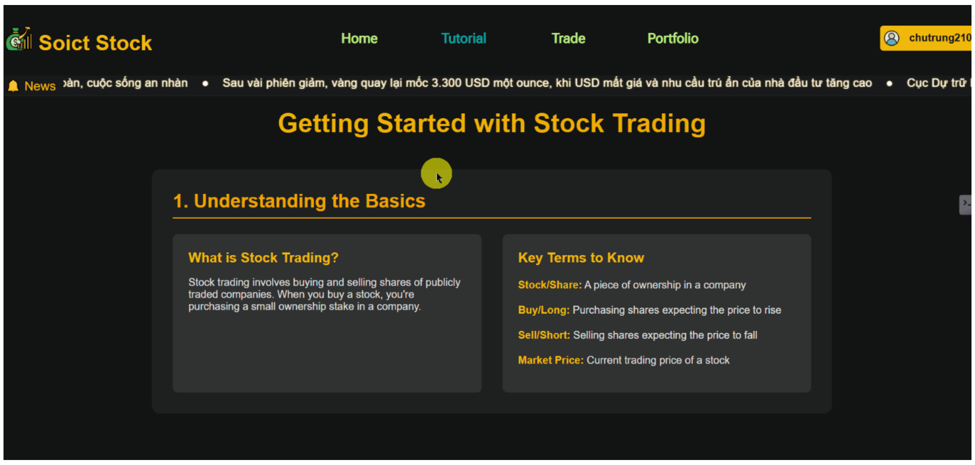
Industry: ELECTRONIC COMPUTERS

Apple Inc. is an American multinational technology company that specializes in consumer electronics, computer software, and online services. Apple is the world's largest technology company by revenue (totalling $274.5 billion in 2020) and, since January 2021, the world's most valuable company. As of 2021, Apple is the world's fourth-largest PC vendor by unit sales, and fourth-largest smartphone manufacturer. It is one of the Big Five American information technology companies, along with Amazon, Google, Microsoft, and Facebook.

5/7/2025, 5:46:56 PM
Open: $209.30
High: $213.58
Low: $206.67
Close: $212.50
Change: +3.20 (1.53%)

Figure 3.3: A Interactive Chart in Home page

The following is a tutorial page to help you get started easily with trading and using our trading platform.



News ờàn, cuộc sống an nhàn • Sau vài phiên giảm, vàng quay lại mốc 3.300 USD một ounce, khi USD mất giá và nhu cầu trú ẩn của nhà đầu tư tăng cao • Cục Dự trữ

Getting Started with Stock Trading

1. Understanding the Basics

What is Stock Trading?

Stock trading involves buying and selling shares of publicly traded companies. When you buy a stock, you're purchasing a small ownership stake in a company.

Key Terms to Know

Stock/Share: A piece of ownership in a company

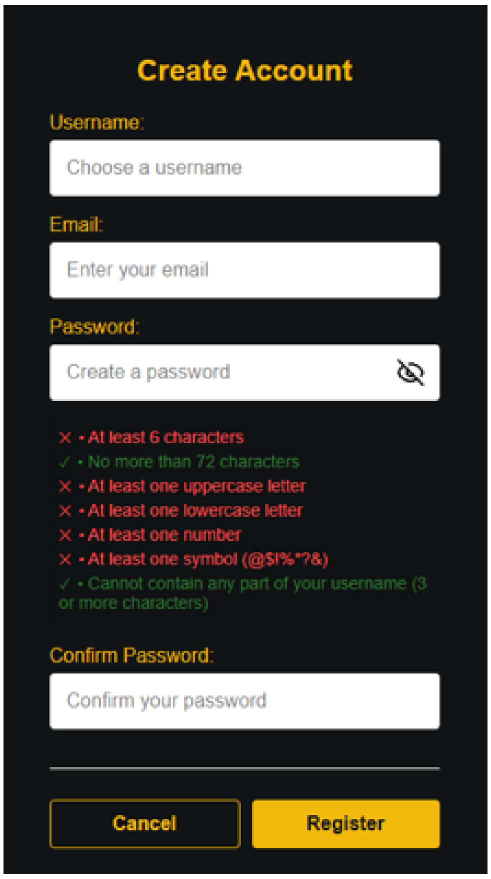
Buy/Long: Purchasing shares expecting the price to rise

Sell/Short: Selling shares expecting the price to fall

Market Price: Current trading price of a stock

Figure 3.4: Tutorial page

To have full access to our web app, you need to register. To keep your account secure, your password must comply with our password policy.



Create Account

Username:

Choose a username

Email:

Enter your email

Password:

Create a password

× • At least 6 characters
✓ • No more than 72 characters
× • At least one uppercase letter
× • At least one lowercase letter
× • At least one number
× • At least one symbol (@$!%*?&)
✓ • Cannot contain any part of your username (3 or more characters)

Confirm Password:

Confirm your password

Figure 3.5: Register

After registering, you will be redirected to login form. You can also log in with your Google account.

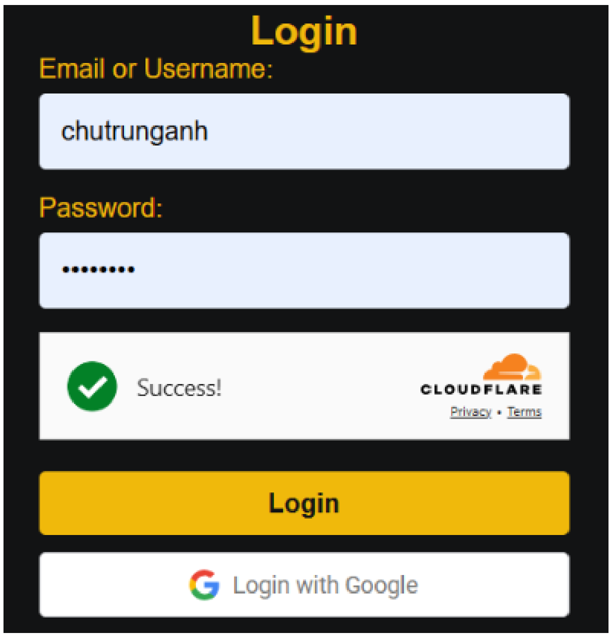


Figure 3.6: Enter Caption

If you forget your password or need to reset it, we will give you an OTP by email to reset the password.

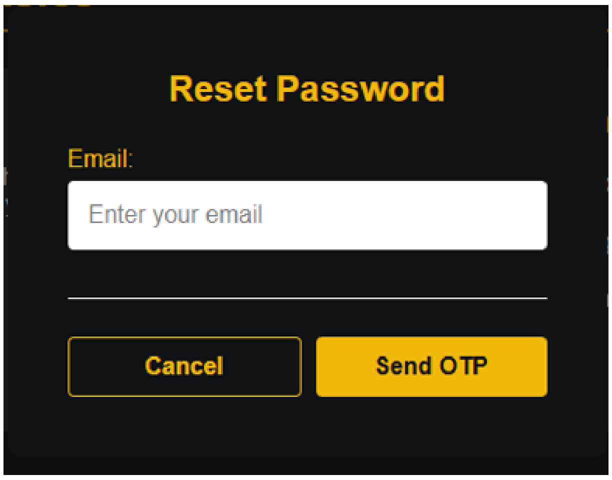
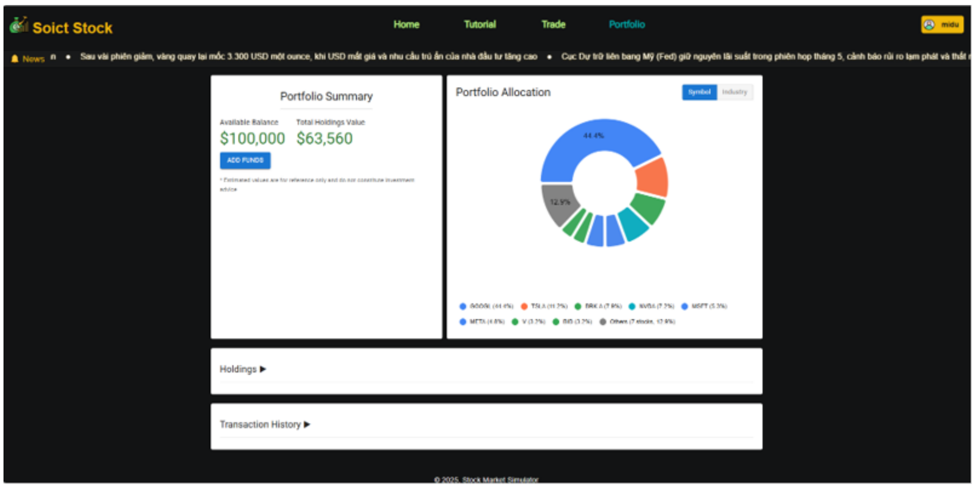


Figure 3.7: Caption

After registering an account, by default, you will have $100,000 and various stocks to start trading. This will show in Portfolio page.



Soict Stock

Home Tutorial Trade Portfolio

midu

News • Sau vài phiên giảm, vàng quay lại mốc 3.300 USD một ounce, khi USD mất giá và nhu cầu trú ẩn của nhà đầu tư tăng cao • Cục Dự trữ liên bang Mỹ (Fed) giữ nguyên lãi suất trong phiên họp tháng 5, cảnh báo rủi ro lạm phát và thất

Portfolio Summary

Available Balance: $100,000

Total Holdings Value: $63,560

ADD FUNDS

* Estimated values are for reference only and do not constitute investment advice

Portfolio Allocation

Symbol Industry

44.4%
12.9%
● GOOGL (44.4%) ● TSLA (11.2%) ● BRK.A (7.8%) ● NVDA (7.2%) ● MSFT (5.3%)
● META (4.8%) ● V (3.2%) ● BID (3.2%) ● Others (7 stocks, 12.9%)

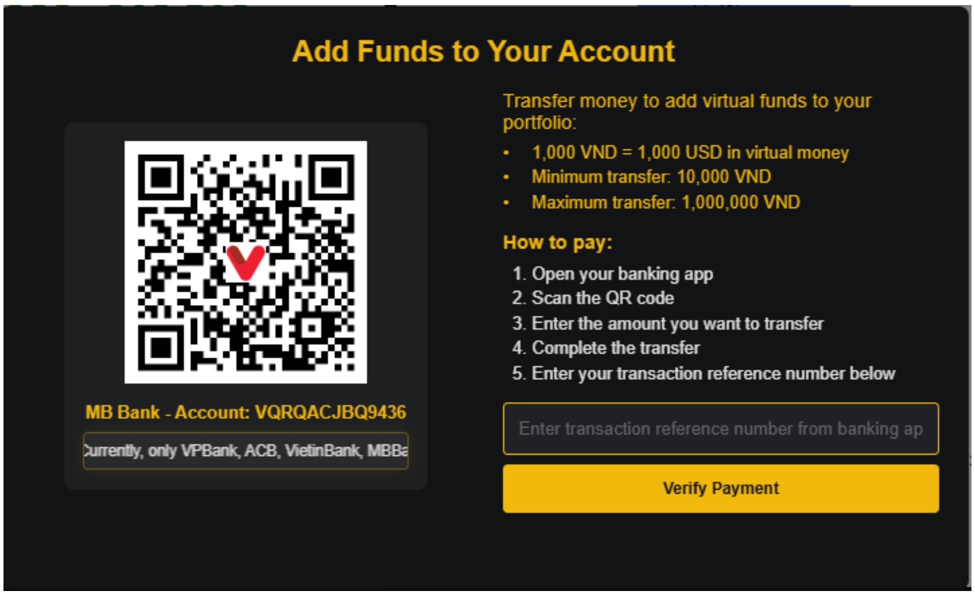
Holdings ▶

Transaction History ▶

© 2025. Stock Market Simulator

Figure 3.8: Portfolio

You can add more fund to your account by clicking the 'add fund' button and the interface will pop up. The conversion rate is set at 1,000 VND for 1,000 USD in virtual money. Users can transfer an amount ranging from a minimum of 10,000 VND to a maximum of 1,000,000 VND. To complete the payment, users need to open their banking app, scan the provided QR code, enter the desired transfer amount, and complete the transaction. After the transfer, they must input the transaction reference number into the designated field and click the "Verify Payment" button to confirm.



Add Funds to Your Account

MB Bank - Account: VQRQACJBQ9436

Currently, only VPBank, ACB, VietinBank, MBBa

Transfer money to add virtual funds to your portfolio:

- 1,000 VND = 1,000 USD in virtual money
- Minimum transfer: 10,000 VND
- Maximum transfer: 1,000,000 VND

How to pay:

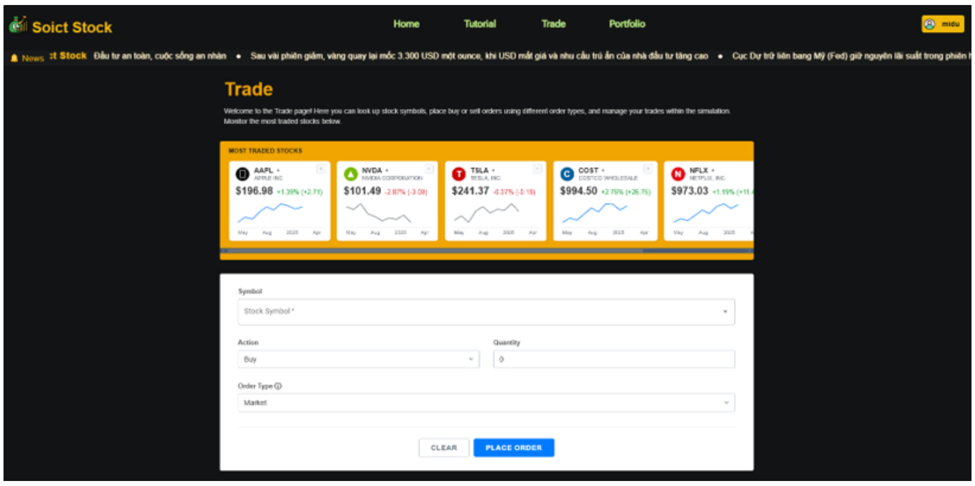
1. Open your banking app
2. Scan the QR code
3. Enter the amount you want to transfer
4. Complete the transfer
5. Enter your transaction reference number below

Enter transaction reference number from banking ap

Verify Payment

Figure 3.9: Caption

In the trade page, at the top, users can view a list of the most traded stocks, such as AAPL, NVDA, TSLA, COST, and NFLX. Below the stock highlights is a trading form where users can input a stock symbol, choose an action (Buy or Sell), specify the quantity, and select the order type (Limit and Market). Once the necessary details are filled in, users can place their simulated trade by clicking the "Place Order" button. There is also a "Clear" button to reset the form. This interface is designed to help users practice trading strategies in a virtual environment.



Soict Stock Đầu tư an toàn, cuộc sống an nhàn • Sau vài phiên giảm, vàng quay lại mốc 3.300 USD một ounce, khi USD mất giá và nhu cầu trú ẩn của nhà đầu tư tăng cao • Cục Dự trữ liên bang Mỹ (Fed) giữ nguyên lãi suất trong phiên

Trade

Welcome to the Trade page! Here you can look up stock symbols, place buy or sell orders using different order types, and manage your trades within the simulation. Monitor the most traded stocks below.

MOST TRADED STOCKS

AAPL APPLE INC $196.98 +1.39% (+2.71)

NVDA NVIDIA CORPORATION $101.49 -2.87% (-3.00)

TSLA TESLA, INC $241.37 -0.07% (-0.15)

COST COSTCO WHOLESALE $994.50 +2.76% (+26.75)

NFLX NETFLIX, INC $973.03 +1.19% (+11.

Symbol: Stock Symbol *

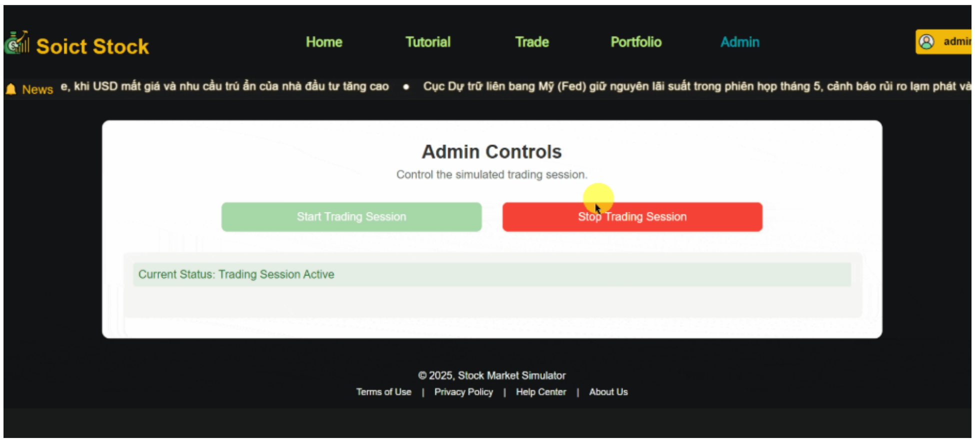
Action: Buy

Quantity: 0

Order Type: Market

Figure 3.10: Caption

In addition, we develop the admin features to control the trading time. We will provide a admin account to access this. As an admin, you have a authorize to open/close a trading session.



e, khi USD mất giá và nhu cầu trú ẩn của nhà đầu tư tăng cao • Cục Dự trữ liên bang Mỹ (Fed) giữ nguyên lãi suất trong phiên họp tháng 5, cảnh báo rủi ro lạm phát và

Admin Controls

Control the simulated trading session.

Current Status: Trading Session Active

© 2025, Stock Market Simulator

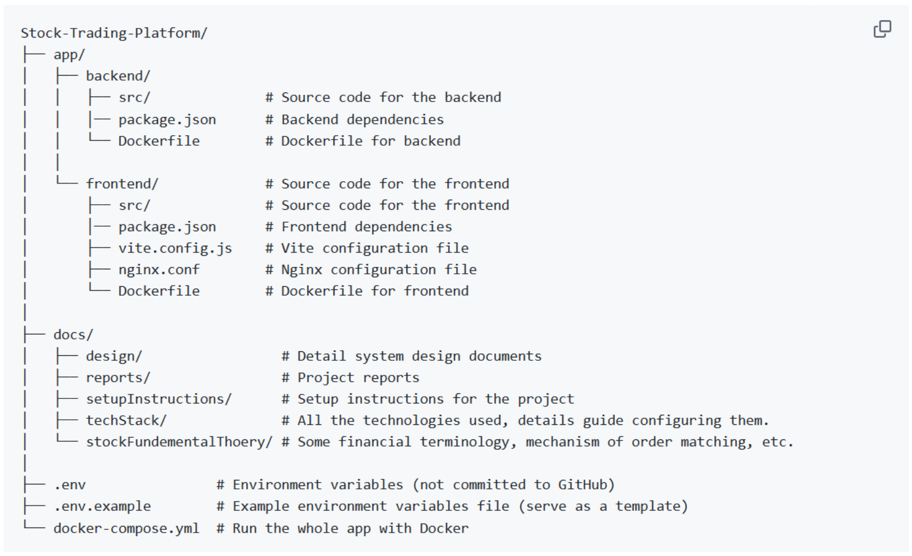
Figure 3.11: Caption

Chapter 4

System Design and Architecture

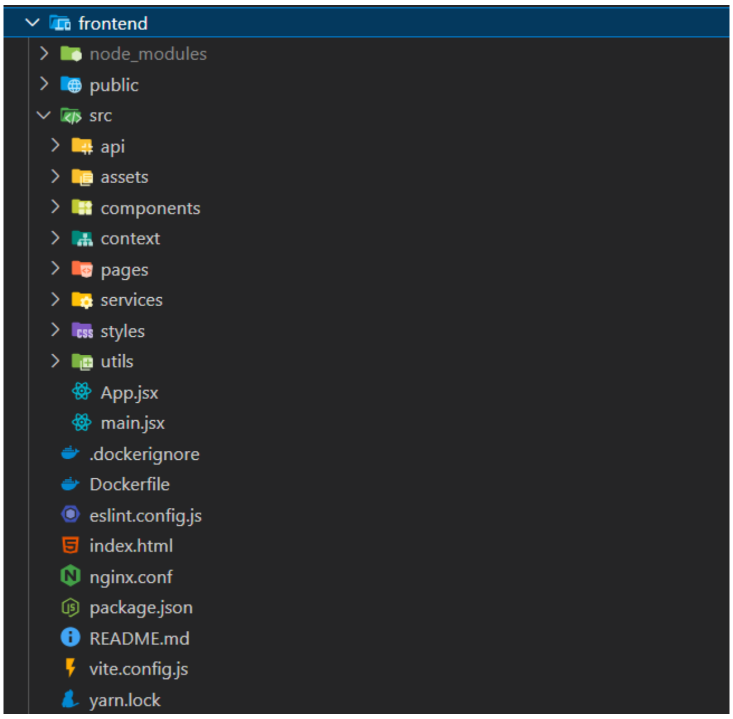
1 Overview

The Stock Trading Platform follows a modern full-stack architecture with clear separation of concerns between frontend, backend, and supporting components. A thorough examination of the project structure reveals a well-organized approach to development and deployment:



```
Stock-Trading-Platform/
├── app/
│   ├── backend/
│   │   ├── src/              # Source code for the backend
│   │   ├── package.json      # Backend dependencies
│   │   └── Dockerfile        # Dockerfile for backend
│   │
│   └── frontend/             # Source code for the frontend
│       ├── src/              # Source code for the frontend
│       ├── package.json      # Frontend dependencies
│       ├── vite.config.js    # Vite configuration file
│       ├── nginx.conf        # Nginx configuration file
│       └── Dockerfile        # Dockerfile for frontend
│
├── docs/
│   ├── design/                 # Detail system design documents
│   ├── reports/                # Project reports
│   ├── setupInstructions/      # Setup instructions for the project
│   ├── techStack/              # All the technologies used, details guide configuring them.
│   └── stockFundementalThoery/ # Some financial terminology, mechanism of order matching, etc.
│
├── .env                # Environment variables (not committed to GitHub)
├── .env.example        # Example environment variables file (serve as a template)
└── docker-compose.yml  # Run the whole app with Docker
```

2 Frontend



```
frontend
  node_modules
  public
  src
    api
    assets
    components
    context
    pages
    services
    styles
    utils
    App.jsx
    main.jsx
  .dockerignore
  Dockerfile
  eslint.config.js
  index.html
  nginx.conf
  package.json
  README.md
  vite.config.js
  yarn.lock
```

This application is developed as a Single Page Application (SPA) using React and React Router DOM for client-side routing.

2.1 Entry point

The application's entry point is composed of three main files. The index.html file serves as the primary HTML template loaded by the browser. The main.jsx file acts as the JavaScript entry point, responsible for rendering the React application into the DOM and setting up global providers such as context or state management tools. Finally, App.jsx functions as the root React component, managing the overall layout, application routing using react-router-dom, and integrating any global context providers needed across the app. Together, these files establish the foundation for the client-side architecture.

2.2 Build and Tooling

The application is built with **Vite**, enabling fast development and optimized production builds. **ESLint** is configured to enforce code quality and consistency throughout the codebase. Additionally, the project includes a **Dockerfile** and **Nginx configuration**, supporting containerized deployment and ensuring the app can be run in various of settings.

Key Folders:

- **components/**: Contains reusable UI components (e.g., Header, Footer, Modal, RoleProtectedRoute).
- **pages/**: Contains top-level page components for each route/view (e.g., Home, Trade, Portfolio, Admin, Tutorial).
- **context/**: Holds React Context files for global state management (e.g., AuthContext, TradingSessionContext).
- **api/**: Contains modules for API calls to the backend.
- **assets/**: Stores static files such as images and icons.
- **styles/**: Contains global and component-specific CSS files.
- **services/**: (If present) For business logic or API service abstractions.
- **utils/**: Utility/helper functions used across the app.

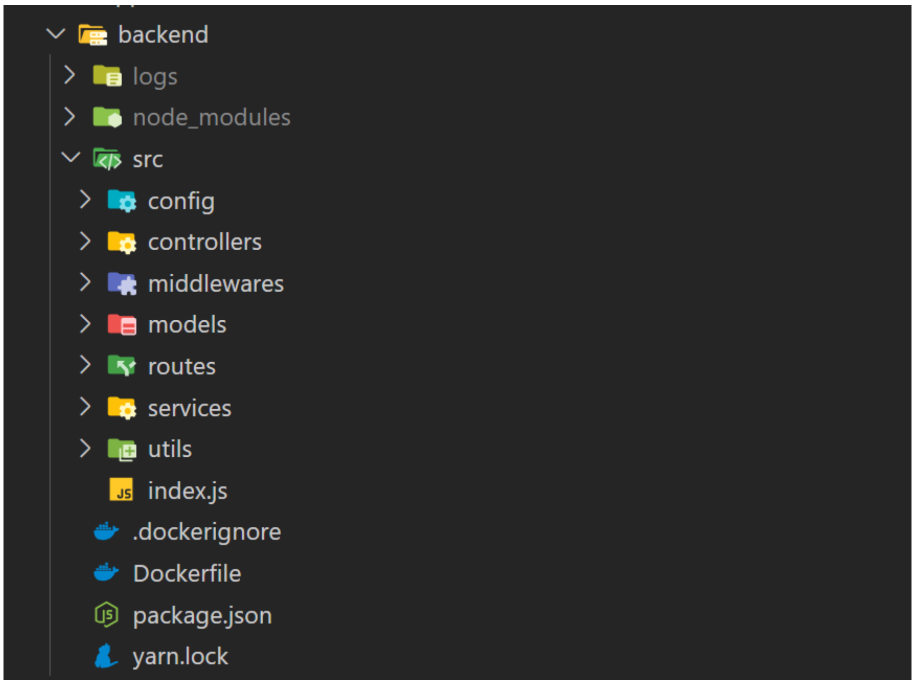
2.3 Routing

The application uses *react-router-dom* for client-side routing. Routes are defined in the **App.jsx** file, where each path is mapped to its corresponding page component. For protected pages such as Trade, Portfolio, and Admin, the app utilizes a **RoleProtectedRoute** component to enforce authentication and authorization. This ensures that login users can access these routes and only admin user can access the Admin page.

2.4 Component Design

The application has a modular, component-based architecture. Each component is intended to be reusable and follow the single-responsibility concept, making the software more maintainable and scalable. Page components act as containers, responsible for data retrieval, state management, and logic processing. In contrast, presentational components are entirely concerned with rendering the user interface, guaranteeing a clear distinction between logic and presentation.

3 Backend



```
backend
  logs
  node_modules
  src
    config
    controllers
    middlewares
    models
    routes
    services
    utils
    index.js
  .dockerignore
  Dockerfile
  package.json
  yarn.lock
```

Here is the overview of backend workflow:

`index.js` → `routes` → `(middlewares)` → `controllers` → `services` → `models` → `services` → `controllers` → `(middlewares)` → **Response to client**

For more specific:

`index.js` listens for incoming requests from front-end and directs them to the appropriate route → The route handler is called, which is defined in the `routes` directory. The route handler specifies the endpoint and the HTTP method (GET, POST, PUT, DELETE) → The route then calls the corresponding controller function → The request might go through some middleware functions (e.g., validation, logging) in the `middlewares` folder before reaching the `controller` → The controller function takes in the request, then passes parameters to service functions to perform the actual business logic here (see `controllers` folder) → The service functions, see `services` folder, may interact with the database model (see `models` folder) → The controller receives the data from the service functions and processes it as needed → Finally, the response (usually in JSON format) is sent back to the client.

Here is a brief overview of the most important folders in the backend:

3.0.1 config

This folder contains configuration files for the entire backend.

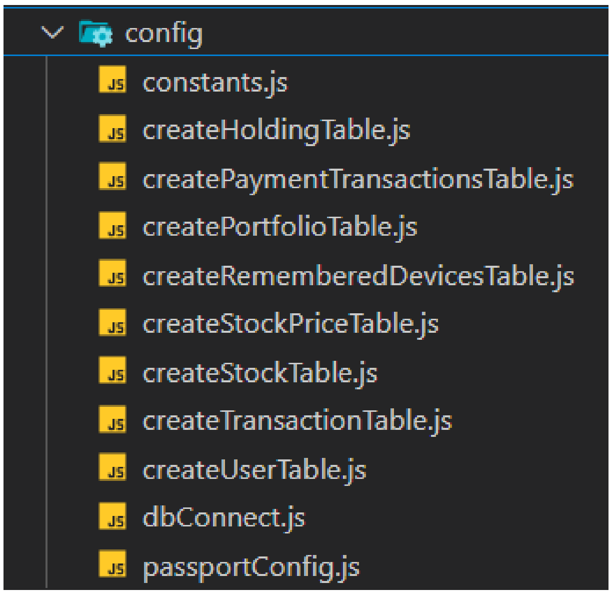

config
constants.js
createHoldingTable.js
createPaymentTransactionsTable.js
createPortfolioTable.js
createRememberedDevicesTable.js
createStockPriceTable.js
createStockTable.js
createTransactionTable.js
createUserTable.js
dbConnect.js
passportConfig.js

Figure 4.1: Enter Caption

It includes:

- Global constants
- Postgres Database connection setup
- Table creation and initial data seeding
- Passport configuration for Google OAuth2 authentication

3.0.2 controllers

Controller functions receive requests, extract necessary data, call corresponding service functions, and return the appropriate response.

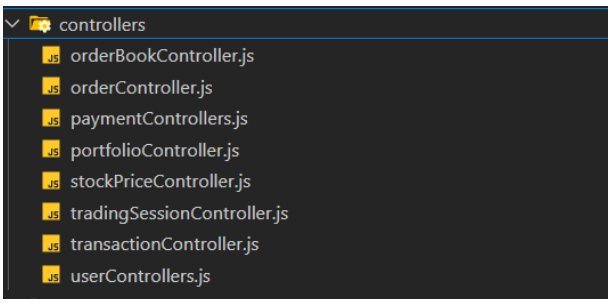
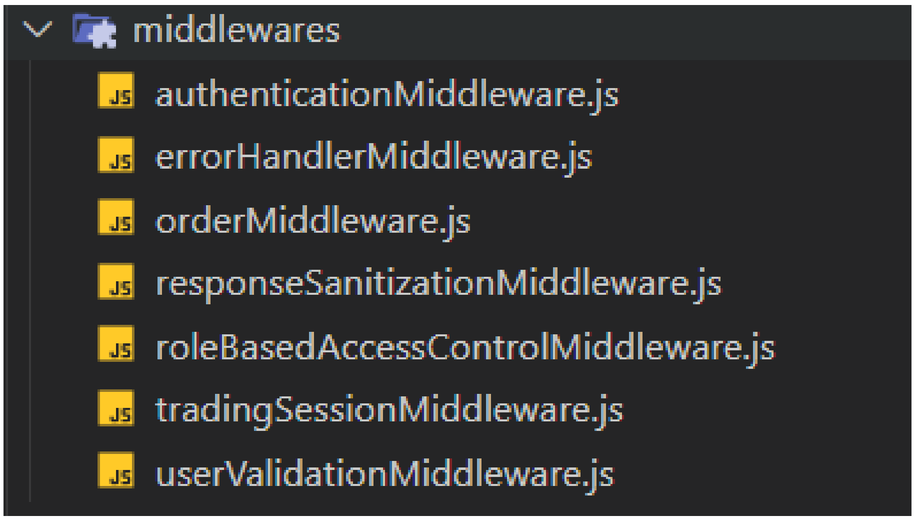

controllers
orderBookController.js
orderController.js
paymentControllers.js
portfolioController.js
stockPriceController.js
tradingSessionController.js
transactionController.js
userControllers.js

Figure 4.2: Enter Caption

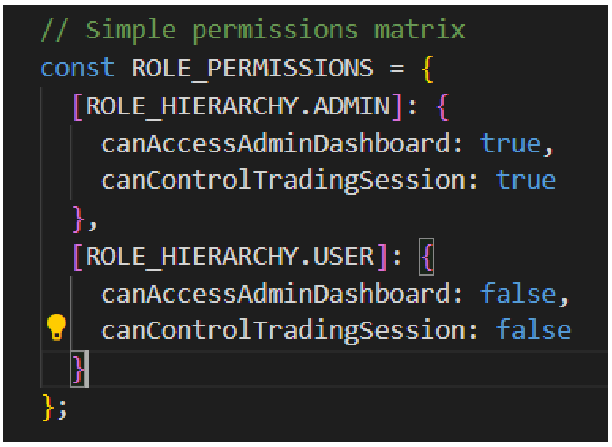


```
middlewares
    authenticationMiddleware.js
    errorHandlerMiddleware.js
    orderMiddleware.js
    responseSanitizationMiddleware.js
    roleBasedAccessControlMiddleware.js
    tradingSessionMiddleware.js
    userValidationMiddleware.js
```

3.0.3 middleware

Middleware functions check the requests before they reach the controller:

- **authenticationMiddleware:** Verifies access and refresh JWT tokens before allowing access.
- **errorHandlingMiddleware:** Captures and formats errors from controllers/services into a consistent response.
- **orderValidationMiddleware:** Ensures order requests include all required fields and follow constraints:
  - With all Sell orders: must not exceed the user's current stock holdings.
  - With all Limit orders: price must be within the range of ceiling and floor price.
  - Buy limit orders: user's balance must be sufficient to pay for the amount stock value he want to buy.
- **responseSanitizationMiddleware:** Formats /escapes harmful characters in response/output to prevent XSS.
- **roleBasedAccessControlMiddleware:** Checks if the user has permission for a given route. We use the Permissions Matrix as follow to defined pesmission for each roles:

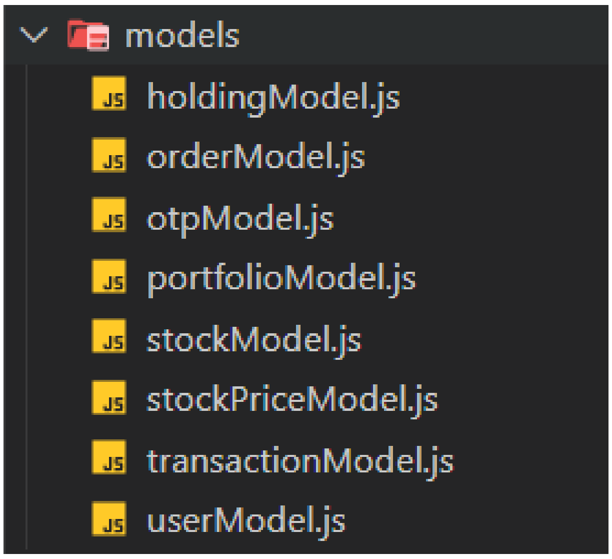


```
// Simple permissions matrix
const ROLE_PERMISSIONS = {
  [ROLE_HIERARCHY.ADMIN]: {
    canAccessAdminDashboard: true,
    canControlTradingSession: true
  },
  [ROLE_HIERARCHY.USER]: {
    canAccessAdminDashboard: false,
    canControlTradingSession: false
  }
};
```

- **tradingSessionMiddleware:** Ensures users can place orders only when the session is open.
- **userValidationMiddleware:** Validates login/registration data (email, password, etc, or any other characters that can lead to XSS/SQL injection).

3.0.4 models

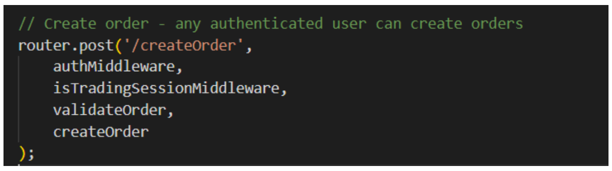
Define the data structures and interact with the database. Each model corresponds to a table in the database.



```
models
    holdingModel.js
    orderModel.js
    otpModel.js
    portfolioModel.js
    stockModel.js
    stockPriceModel.js
    transactionModel.js
    userModel.js
```

3.0.5 routes

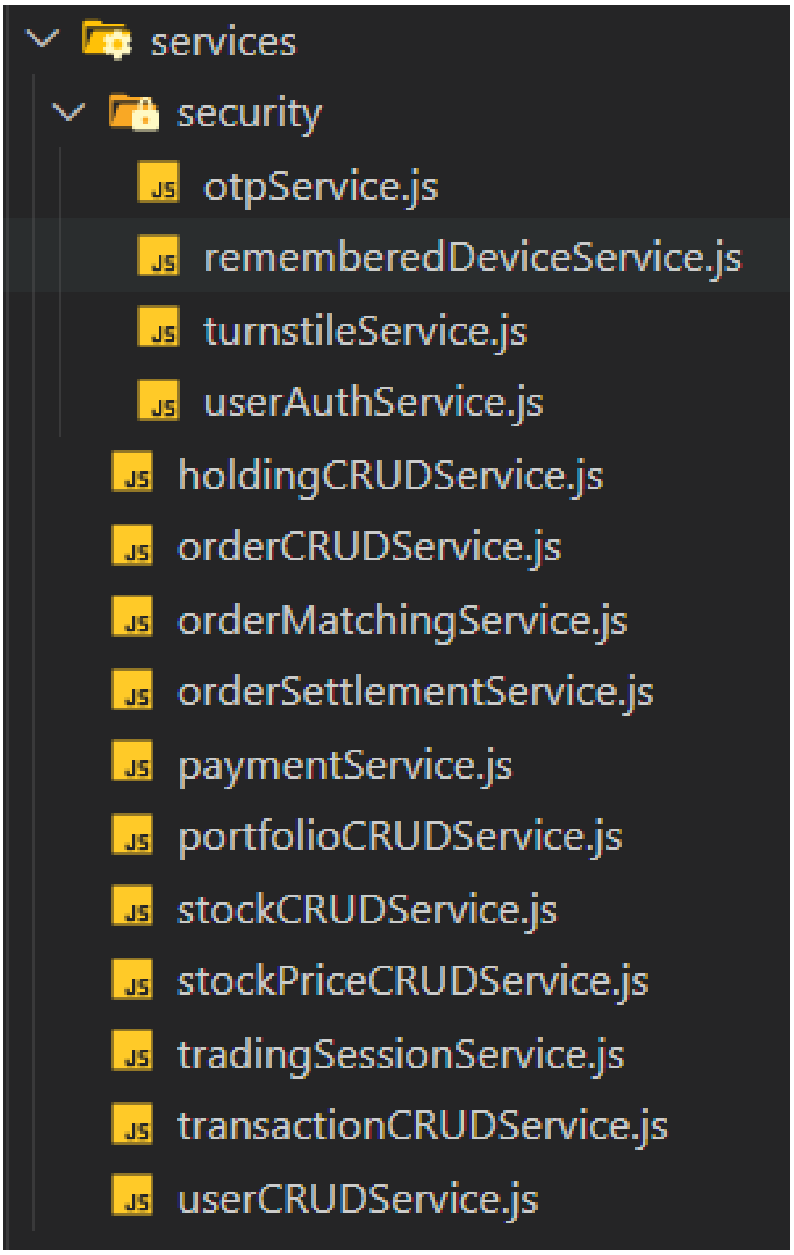
Define the API endpoints and map them to the appropriate controller functions. Before access to the controller, we put middlewares before that to protect the controller functions, somethings like this:



```
// Create order - any authenticated user can create orders
router.post('/createOrder',
    authMiddleware,
    isTradingSessionMiddleware,
    validateOrder,
    createOrder
);
```

Before can access to the createOrder servcie, a request will be verified if the user have been authenticated, the trading session still open and the order is valid in price/volume.

3.0.6 services



```
services
  security
    otpService.js
    rememberedDeviceService.js
    turnstileService.js
    userAuthService.js
  holdingCRUDService.js
  orderCRUDService.js
  orderMatchingService.js
  orderSettlementService.js
  paymentService.js
  portfolioCRUDService.js
  stockCRUDService.js
  stockPriceCRUDService.js
  tradingSessionService.js
  transactionCRUDService.js
  userCRUDService.js
```

Contain the actual business logic of the application. Each service handles a specific domain or functionality. In this report, I will ignore the common CRUD (Create, Read, Update, Delete) services for models like userCRUDService.js, stockCRUDService.js, and transactionHistoryCRUDService.js, as they are straightforward and follow the same pattern, instead I will focus one some important core services that implement the main functionality of the application:

- **orderCRUDService.js**:
  Handles order creation and cancellation. Its functions return an order object that will be used in `orderMatchingService.js`.

- **orderMatchingService.js**:
  More details about this service are documented at: techStack_1.md

  This is the heart of the system. When it receives an order, it places it into one of two queues for correspoding order:

  - Buy limit orders
  - Sell limit orders

  Orders are prioritized by price (and by time also, but the default behavior First-In-First-Out of queue, there is no need to process this):

  - Higher prices have priority in buy orders.
  - Lower prices have priority in sell orders.

  Market orders are not queued since they are matched immediately upon arrival. When a new order enters the order book, it attempts to match with the opposite type of order in the queue:

  - If no match is found, the order is added to the order book.
  - If a match is found, the order is executed.

  Matching rules:

  - A sell order matches with a buy order that has a price equal to or higher than the ask price.
  - A buy order matches with a sell order that has a price equal to or lower than the bid price.

  Partial matches may occur. For example:

The order book has a sell limit order at price 160 with volume 5.
A new buy order arrives at price 170 with volume 9.
The system matches all 5 volume at 160, leaving a remaining buy order of volume 4 at 170 in the order book.

In such cases, the order is split into:

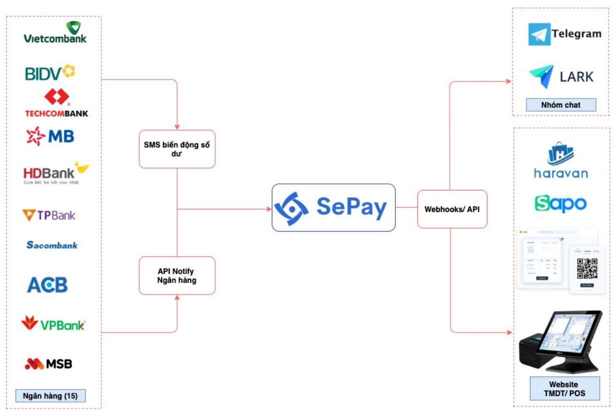
– A matched part (returned to the user)

– An unmatched part (added back to the order book)

Once an order is matched (fully or partially), it calls `orderSettlementService.js` to process settlement, update user balances, stock holdings, and create transaction history for both buyer and seller.

- **orderSettlementService.js**:
  This service updates user balances and stock holdings after an order is matched. It ensures that financial transactions are recorded correctly and that both buyer and seller accounts are updated.

- **paymentService.js**:



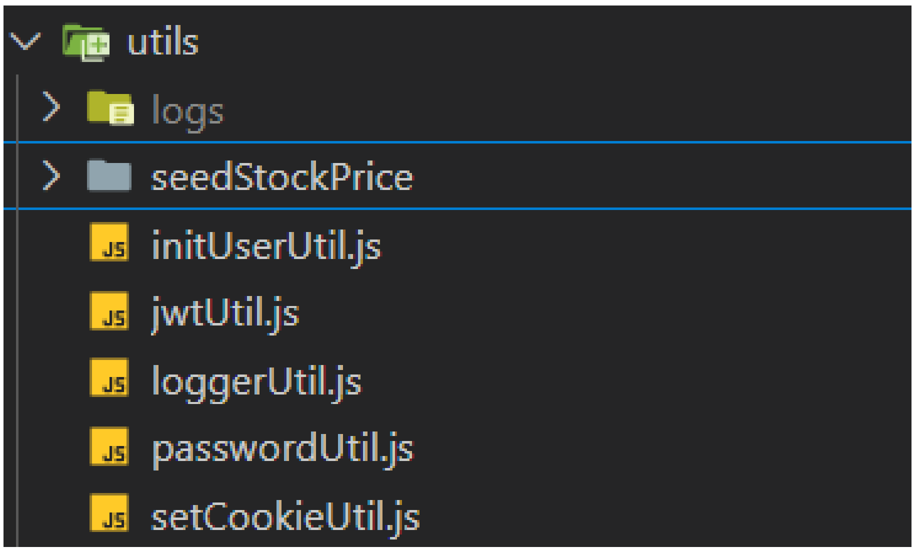
Handles banking transactions through **Sepay**. The process:

1. The user scans a QR code and transfers money to Sepay.
2. Sepay transfers the money to the bank.
3. The bank sends a notification back to Sepay.
4. This service calls Sepay's API to check the notification.
5. The user inputs the transaction reference number.
6. Sepay returns the transaction details.

7. The service extracts the transferred amount, updates the user's virtual balance in the database, and creates a transaction history for the user.

- **tradingSessionService.js**:
  Manages trading sessions, including opening, closing, and checking session status. It coordinates with the frontend trading session context to ensure:
  - Users can place orders only when the session is open.
  - Only **admins** can open or close trading sessions.
  - Session status is publicly accessible to all users.
- **security/**:
  Contains security-related services. These will be discussed in the **Security Features** section below.

3.0.7 utils



```
utils
  logs
  seedStockPrice
  initUserUtil.js
  jwtUtil.js
  loggerUtil.js
  passwordUtil.js
  setCookieUtil.js
```

- **seedStockPrice**: Contains a script to fetch real-world stock prices and insert them into the `stockPrice` table in the database. Guild to run this script can be found inside that folder.
- **initUserUtil.js**: Initializes sample users for testing, including both an admin account and a regular user account.
- **jwtUtil.js**: Provides functions to generate and verify both access and refresh tokens.
- **setCookieUtil.js**: Wraps tokens into cookies with `HttpOnly`, `Secure`, and `SameSite` properties enabled.

- **passwordUtil.js**: Defines the application's password policy.
- **loggerUtil.js**: Configures log types and log levels, and exports custom logging functions for use across the backend.

4 Database

We design the database following these requirements:

- **User Management**: Allows user registration, login, profile management. Each new user will have default 100 000$ in the portfolio balance and 10 stock each of all stock symbol.
- **Stock Trading Simulation**: Users can buy and sell stocks and monitor their portfolio change.
- **Market Data**: Provides stock prices, company information.
- **Transaction History**: Records users' buy and sell activities.
- **Banking fund**: Allow user to fund their virtual balance with real money.

We provide the database design file in form of `.io`, which can be opened with Draw.io for easy looking, see it here: DatabaseDesignDraft.drawio

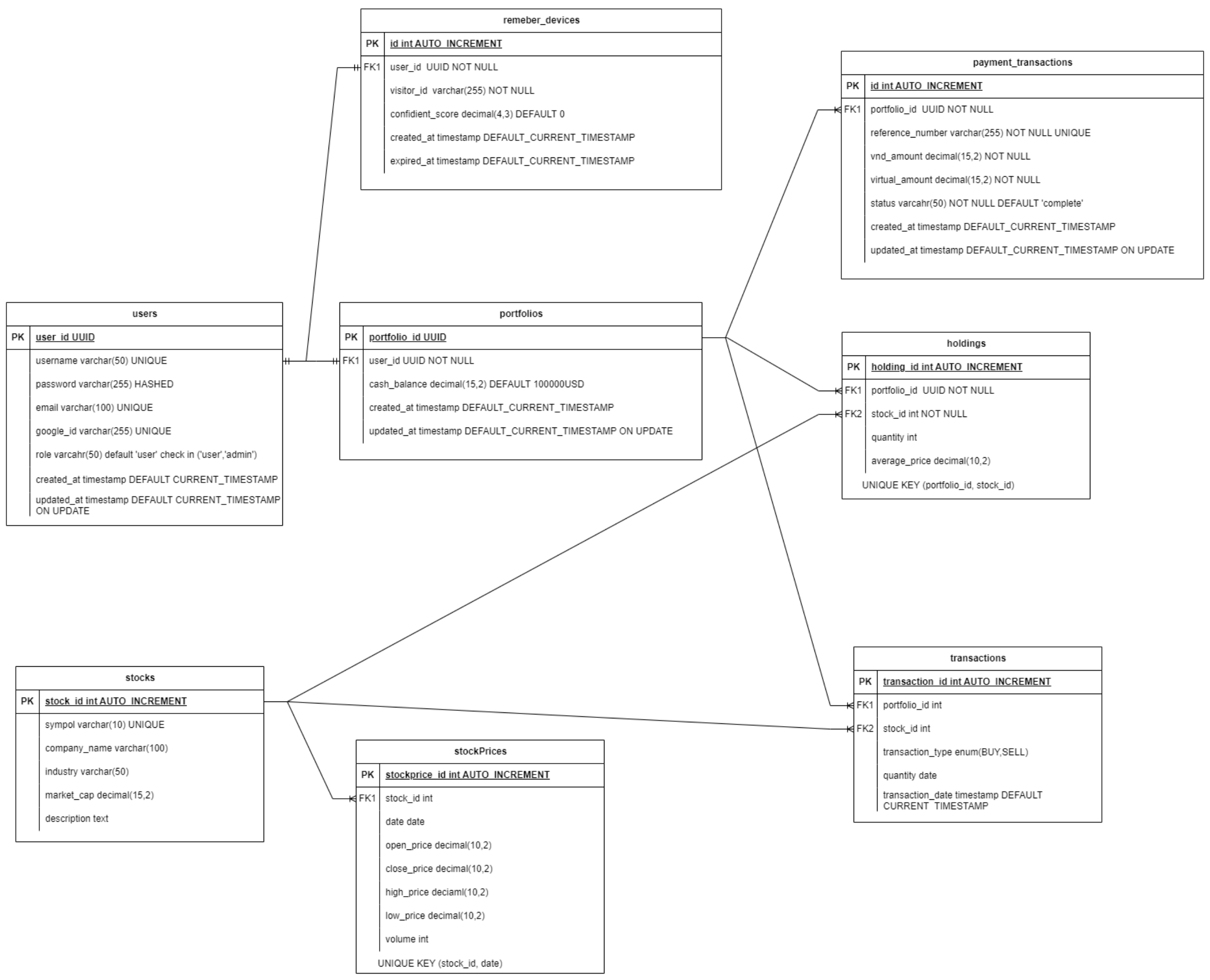

remeber_devices
PK id int AUTO_INCREMENT
FK1 user_id UUID NOT NULL
visitor_id varchar(255) NOT NULL
confidient_score decimal(4,3) DEFAULT 0
created_at timestamp DEFAULT_CURRENT_TIMESTAMP
expired_at timestamp DEFAULT_CURRENT_TIMESTAMP
payment_transactions
PK id int AUTO_INCREMENT
FK1 portfolio_id UUID NOT NULL
reference_number varchar(255) NOT NULL UNIQUE
vnd_amount decimal(15,2) NOT NULL
virtual_amount decimal(15,2) NOT NULL
status varcahr(50) NOT NULL DEFAULT 'complete'
created_at timestamp DEFAULT_CURRENT_TIMESTAMP
updated_at timestamp DEFAULT_CURRENT_TIMESTAMP ON UPDATE
users
PK user_id UUID
username varchar(50) UNIQUE
password varchar(255) HASHED
email varchar(100) UNIQUE
google_id varchar(255) UNIQUE
role varcahr(50) default 'user' check in ('user','admin')
created_at timestamp DEFAULT CURRENT_TIMESTAMP
updated_at timestamp DEFAULT CURRENT_TIMESTAMP ON UPDATE
portfolios
PK portfolio_id UUID
FK1 user_id UUID NOT NULL
cash_balance decimal(15,2) DEFAULT 100000USD
created_at timestamp DEFAULT_CURRENT_TIMESTAMP
updated_at timestamp DEFAULT_CURRENT_TIMESTAMP ON UPDATE
holdings
PK holding_id int AUTO_INCREMENT
FK1 portfolio_id UUID NOT NULL
FK2 stock_id int NOT NULL
quantity int
average_price decimal(10,2)
UNIQUE KEY (portfolio_id, stock_id)
stocks
PK stock_id int AUTO_INCREMENT
sympol varchar(10) UNIQUE
company_name varchar(100)
industry varchar(50)
market_cap decimal(15,2)
description text
stockPrices
PK stockprice_id int AUTO_INCREMENT
FK1 stock_id int
date date
open_price decimal(10,2)
close_price decimal(10,2)
high_price deciaml(10,2)
low_price decimal(10,2)
volume int
UNIQUE KEY (stock_id, date)
transactions
PK transaction_id int AUTO_INCREMENT
FK1 portfolio_id int
FK2 stock_id int
transaction_type enum(BUY,SELL)
quantity date
transaction_date timestamp DEFAULT CURRENT_TIMESTAMP

Table 1: Users

Stores information about users.

- `user_id`: UUID, PRIMARY KEY
- `username`: VARCHAR(50), UNIQUE
- `password`: VARCHAR(255)
- `email`: VARCHAR(100), UNIQUE
- `google_id`: VARCHAR(255), UNIQUE
- `role`: varchar(50) default ’user’ check in (’user’, ’admin’)
- `created_at`: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP
- `updated_at`: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP

Table 2: Stocks

Stores available stock information.

- `stock_id`: INT, AUTO_INCREMENT, PRIMARY KEY
- `symbol`: VARCHAR(10), UNIQUE
- `company_name`: VARCHAR(100)
- `industry`: VARCHAR(50)
- `market_cap`: DECIMAL(15,2)
- `description`: TEXT

Note: Market capitalization is calculated as current closing price $\times$ *total shares outstanding. The price comes from the `stockPrices` table.*

Table 3: StockPrices

Stores historical stock prices for candlestick charts.

- `price_id`: INT, AUTO_INCREMENT, PRIMARY KEY
- `stock_id`: INT, FOREIGN KEY to `Stocks(stock_id)`
- `date`: DATE
- `open_price`, `high_price`, `low_price`, `close_price`: DECIMAL(10,2)
- `volume`: INT
- UNIQUE KEY (`stock_id`, `date`)

Table 4: Portfolios

Stores each user's investment portfolio.

- `portfolio_id`: UUID, PRIMARY KEY
- `user_id`: INT, FOREIGN KEY to `Users(user_id)`, UNIQUE
- `cash_balance`: DECIMAL(15,2), DEFAULT 100 000
- `created_at`, `updated_at`: TIMESTAMP

Table 5: Holdings

Stores stocks currently held by the user.

- `holding_id`: INT, AUTO_INCREMENT, PRIMARY KEY
- `portfolio_id`: INT, FOREIGN KEY to `Portfolios(portfolio_id)`
- `stock_id`: INT, FOREIGN KEY to `Stocks(stock_id)`
- `quantity`: INT
- `average_cost`: DECIMAL(10,2)
- UNIQUE KEY (`portfolio_id`, `stock_id`)

Table 6: Transactions

Stores the user’s buy/sell history.

- `transaction_id`: INT, AUTO_INCREMENT, PRIMARY KEY
- `portfolio_id`: INT, FOREIGN KEY to `Portfolios(portfolio_id)`
- `stock_id`: INT, FOREIGN KEY to `Stocks(stock_id)`
- `transaction_type`: ENUM(’BUY’, ’SELL’)
- `quantity`: INT
- `price`: DECIMAL(10,2)
- `transaction_date`: TIMESTAMP, DEFAULT CURRENT_TIMESTAMP

Table 7: remembered_devices

Stores trusted/remembered devices for users, usually skip 2FA next time login.

- `id`: SERIAL, PRIMARY KEY
- `user_id`: UUID, NOT NULL, FOREIGN KEY to `Users(id)`, ON DELETE CASCADE
- `visitor_id`: VARCHAR(255), NOT NULL — unique identifier for the device/browser

- `confidence_score`: DECIMAL(4,3), DEFAULT 0 — likelihood the device truly belongs to the user
- `created_at`: TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP
- `expires_at`: TIMESTAMP WITH TIME ZONE, NOT NULL — when the device should expire
- **UNIQUE**(`user_id`, `visitor_id`) — a user cannot have the same device remembered twice

Table 8: payment_transactions

Stores records of payments from real VND banking to virtual money in the portfolio system.

- `id`: SERIAL, PRIMARY KEY
- `portfolio_id`: UUID, NOT NULL, FOREIGN KEY to `Portfolios(portfolio_id)`
- `reference_number`: VARCHAR(255), NOT NULL, UNIQUE — payment reference ID (e.g., from banking or payment gateway)
- `vnd_amount`: DECIMAL(15,2), NOT NULL — amount of Vietnamese Dong transferred
- `virtual_amount`: DECIMAL(15,2), NOT NULL — amount of virtual money received in platform
- `status`: VARCHAR(50), DEFAULT 'completed' — could also be 'pending', 'failed', etc.
- `created_at`: TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP
- `updated_at`: TIMESTAMP WITH TIME ZONE, DEFAULT CURRENT_TIMESTAMP

Table Relationships

- **Users ↔ Portfolios, remember_devices**: One-to-one
- **Portfolios → Holdings, Transactions, payment_transactions**: One-to-many
- **Stocks → StockPrices, Holdings, Transactions**: One-to-many

Chapter 5

Security Features

A more accessible summary of this section can be found in the Security Checklist on the project's GitHub repository: Security Checklist.

1 Password Authentication

1.1 [Finished] Enforce password policy

- Verified at both frontend and backend this policy:
  - Password length: 6–72 characters.
  - At least 1 uppercase letter, 1 number, and 1 special character (`@$!?%*&`).
  - Must not contain 3 or more consecutive characters from the username.
- Refer to `passwordUtils.js` for implementations.
- *This password policy is a simplified version from VNDIRECT Securities JSC.*

1.2 [Finished] Secure password storage

- Use `bcrypt` for salting + slow hashing when storing passwords.
- Refer to `userAuthService.js` for implementations.

1.3 [Finished] Prevention of password guessing

- Use CAPTCHA provided by `Cloudflare Turnstile` to block spam/automated logins.

- Validate on both frontend and backend.
- Refer to `turnstileService.js` for implementations.

1.4 [Finished] Password recovery

- Send OTPs to the user's email with expiration. For demo, use `Ethereal` email.
- OTPs are 8 characters (mixed lower, upper case and numbers), expiry in 1 minute (testing only), generated with `otp-generator`.
- Refer to `otpService.js`, `userAuthService.js`, `AuthContext.jsx` for implementations.

2 Session Authentication & Management

2.1 [Finished] Secure mechanisms for using access tokens

- Use `JWT` with:
  - Short-lived Access Tokens (1-minute expiry for testing).
  - Longer-lived Refresh Tokens (7-day expiry).
  - Tokens include a timestamp in payload before signing to ensure uniqueness.
- Return tokens via cookies with:
  - `Secure` (enabled in production only),
  - `HttpOnly`,
  - `SameSite='Strict'`.
- On logout or browser close: clear cookies on client, revoke Refresh Token on server.
- Refer to `jwtUtil.js`, `setCookieUtil.js`, `authenticationMiddleware.js`, `AuthContext`

2.2 [Finished] CSRF defense

- Only allow requests from the frontend origin in `index.js`:

```
app.use(cors({
  origin: process.env.FE_URL}));
```

- We do not provide public API.
- Cookies use `SameSite='Strict'`.

2.3 [Finished] Session hijacking defense

- Using tokens with cookies as mentioned above.

3 Authorization

3.1 [Finished] Implement suitable access control: MAC, DAC, RBAC

- Implemented **Role-Based Access Control (RBAC)** with permission matrix:

```
const ROLE_PERMISSIONS = {
  [ROLE_HIERARCHY.ADMIN]: {
    canAccessAdminDashboard: true,
    canControlTradingSession: true
  },
  [ROLE_HIERARCHY.USER]: {
    canAccessAdminDashboard: false,
    canControlTradingSession: false
  }
};
```

- Roles:
  - Public guests: Home, Tutorial.
  - Logged-in users: + Trade, Portfolio.
  - Admins: + Admin page.
- Refer to `roleBasedAccessControlMiddleware.js`.

4 Input Validation & Output Sanitization

4.1 [Finished] Input validation and sanitization

- Validate using:

  - `userValidationMiddleware.js` (login, register, reset),
  - `orderMiddleware.js`, `tradingSessionMiddleware.js`.

- Sanitize response with:
  - `responseSanitizationMiddleware.js`, `errorHandlerMiddleware.js`.
- Use `joi` and `xss` libraries.
- Configure CSP (Content Security Policy).

4.2 [Finished] Protection against injection attacks

- Use parameterized SQL queries like `$1`. See `userCRUDService.js`.
- Configure CSP with `helmet`. See `index.js`.

4.3 [Partial] Prevention of path traversal

- Prevent IDOR by:
  - Using `UUID` instead of auto-increment ID in sensitive tables (`userModel.js`, `portfolioModel.js`).
  - Extract IDs from JWT instead of request. See `portfolioController.js`, `orderController.js`.
  - Return 404 for unauthorized access (`NotFoundPage.jsx`).

4.4 [Finished] File upload restriction

- No file upload in current app version.

5 Sensitive Information Leakage

5.1 [Partial] Minimization of sensitive information leakage about servers

- Configure `nginx.conf` for security headers.

5.2 [Finished] Minimization in response

- Disable `X-Powered-By` via `helmet`.

5.3 [Finished] Mitigate Clickjacking

- Set `X-Frame-Options: DENY` via `helmet`.

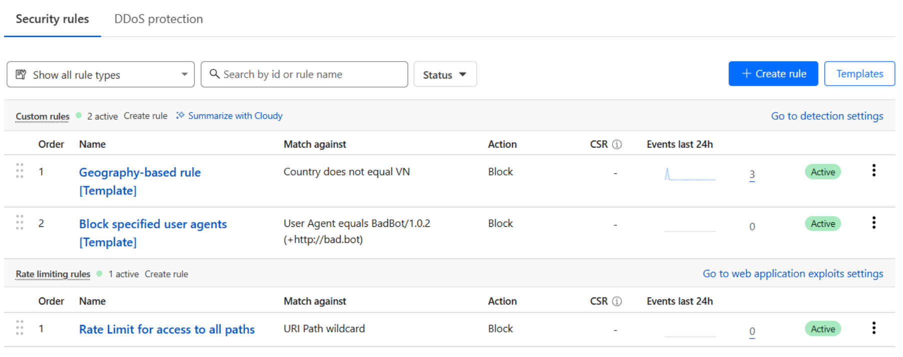
6 Compliance with Standards

6.1 [Finished] HTTPS implementation

- Use `Cloudflare Tunnels` for public access and SSL.

6.2 [Partial] Mitigation of DoS attacks

- Requests go through Cloudflare proxy.
- Configure WAF: rate-limiting, geo-IP, user-agent blocks.
- Only prebuilt rules enabled, not yet customized/tested.



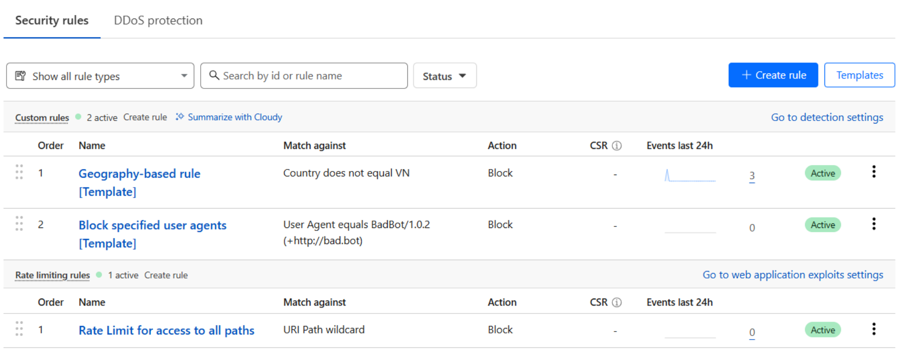
Custom rules ● 2 active Create rule Summarize with Cloudy — Go to detection settings

Order	Name	Match against	Action	CSR	Events last 24h		
1	Geography-based rule [Template]	Country does not equal VN	Block	-	3	Active	⋮
2	Block specified user agents [Template]	User Agent equals BadBot/1.0.2 (+http://bad.bot)	Block	-	0	Active	⋮

Rate limiting rules ● 1 active Create rule — Go to web application exploits settings

Order	Name	Match against	Action	CSR	Events last 24h		
1	Rate Limit for access to all paths	URI Path wildcard	Block	-	0	Active	⋮

Block IP addresses that do not come from Vietnam, block bad user agents (to prevent reconnaissance tools), rate limiting to prevent DDoS attacks, and more.



Custom rules ● 2 active Create rule Summarize with Cloudy Go to detection settings

Order	Name	Match against	Action	CSR	Events last 24h		
1	Geography-based rule [Template]	Country does not equal VN	Block	-	3	Active	⋮
2	Block specified user agents [Template]	User Agent equals BadBot/1.0.2 (+http://bad.bot)	Block	-	0	Active	⋮

Rate limiting rules ● 1 active Create rule Go to web application exploits settings

Order	Name	Match against	Action	CSR	Events last 24h		
1	Rate Limit for access to all paths	URI Path wildcard	Block	-	0	Active	⋮

6.3 [Finished] Secure storage and management of sensitive values

- Store secrets in `.env` file. `.env` not committed to GitHub. Load from environment, not hardcoded.
- Provide `.env.example` as template.

7 Security Testing

7.1 [Finished] Code review with automated tools

- Scan with `Qodana`.
- All `Critical`, `High` fixed. Some `Moderate` remain.

7.2 [Finished] Penetration testing with tools

- Use ZAP Proxy.
- No `High` alerts. Some `Medium`, `Low`, `Informational` left.

8 Bonus

8.1 [Finished] Multi-factor authentication

- After login, send OTP to email.
- Implemented device recognition with `fingerprintJS`.
- Refer to `otpService.js`, `LoginForm.jsx`.

8.2 [Haven't Implemented] Advanced session hijacking prevention

- Track user IPs, detect unfamiliar devices/browsers.

8.3 [Partial] Advanced HTTP flood prevention

- Configure Cloudflare WAF for the domain.

8.4 [Finished] Single Sign-On (SSO)

- Implemented Google OAuth 2.0 using `passport`.
- See `passportConfig.js`.

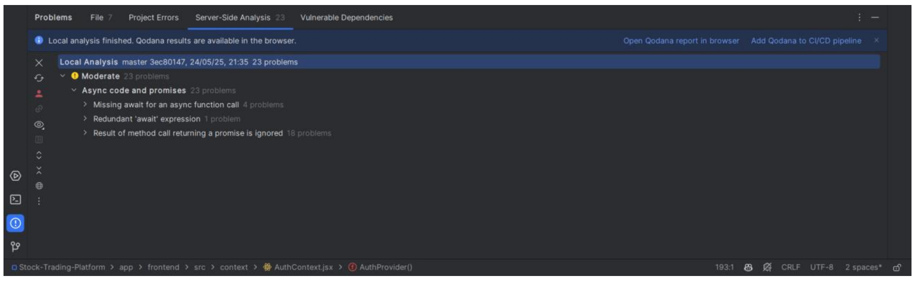
Chapter 6

Security Evaluation

This chapter outlines the comprehensive security testing employed to validate the functionality, security, and performance of the Stock Trading Platform. It details the test scenarios executed, presents the results obtained from functional testing, and provides an in-depth evaluation of the application's security posture based on static and dynamic analysis tools.

Static Application Security Testing with Qodana

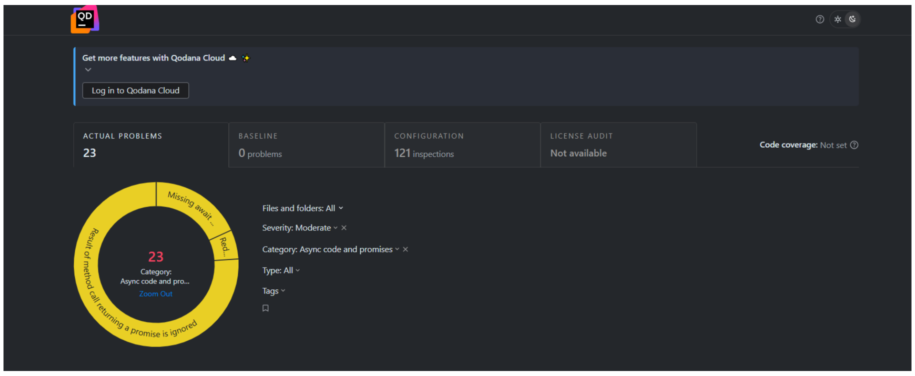
The source code of our application was analyzed using Qodana from JetBrains to identify potential security vulnerabilities, code smells, bugs, and maintainability issues. The summary of the Qodana analysis is as above:



Local analysis finished. Qodana results are available in the browser.

```
Local Analysis master 3ec80147, 24/05/25, 21:35 23 problems
  Moderate 23 problems
    Async code and promises 23 problems
      Missing await for an async function call 4 problems
      Redundant 'await' expression 1 problem
      Result of method call returning a promise is ignored 18 problems
```

Stock-Trading-Platform › app › frontend › src › context › AuthContext.jsx › AuthProvider()   193:1 CRLF UTF-8 2 spaces*


Get more features with Qodana Cloud
Log in to Qodana Cloud
Missing await...
Red...
Result of method call returning a promise is ignored
Zoom Out

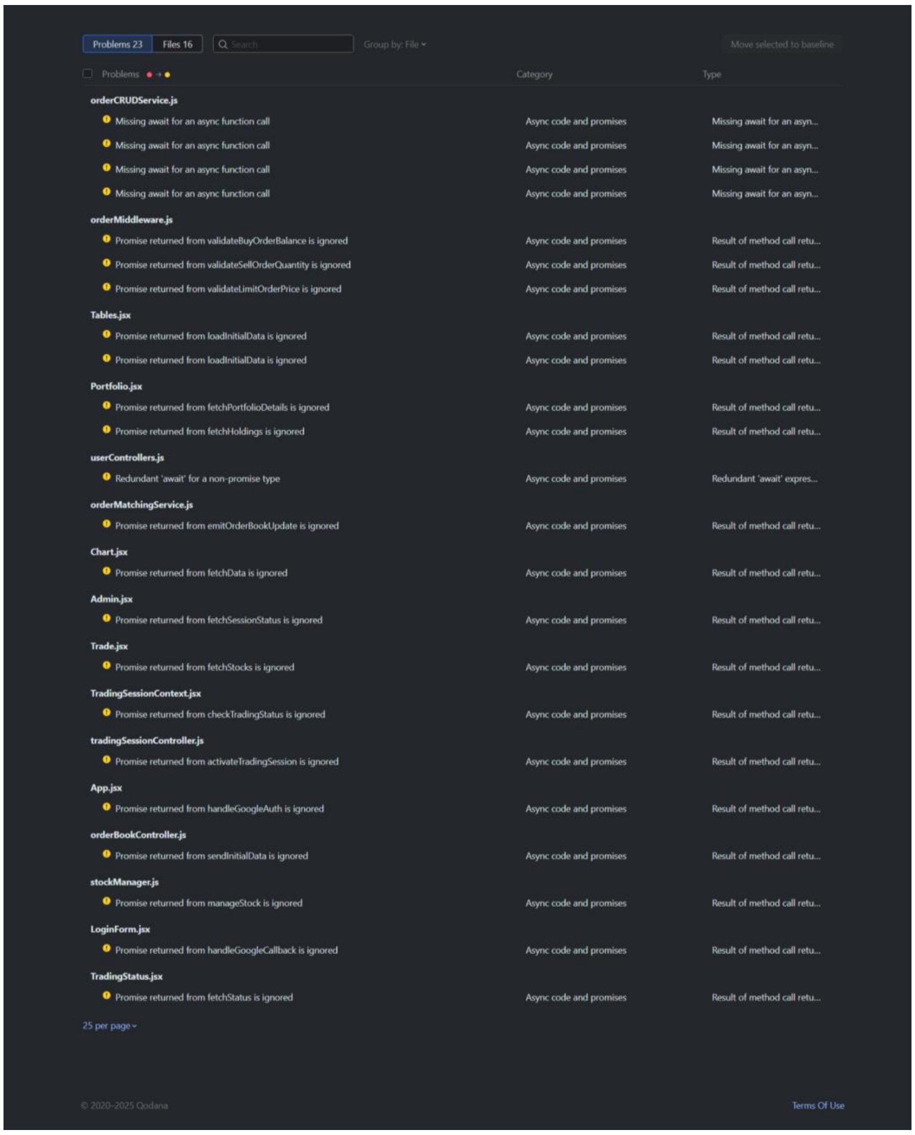
ACTUAL PROBLEMS	BASELINE	CONFIGURATION	LICENSE AUDIT
23	0 problems	121 inspections	Not available

Code coverage: Not set

23
Category:
Async code and pro...

Files and folders: All
Severity: Moderate
Category: Async code and promises
Type: All
Tags

All “critical”, “high” warning have been fixed, still moderate warnings left. Due to time constraints, our group has not finished fixing these warning yet. Detail on warnings:



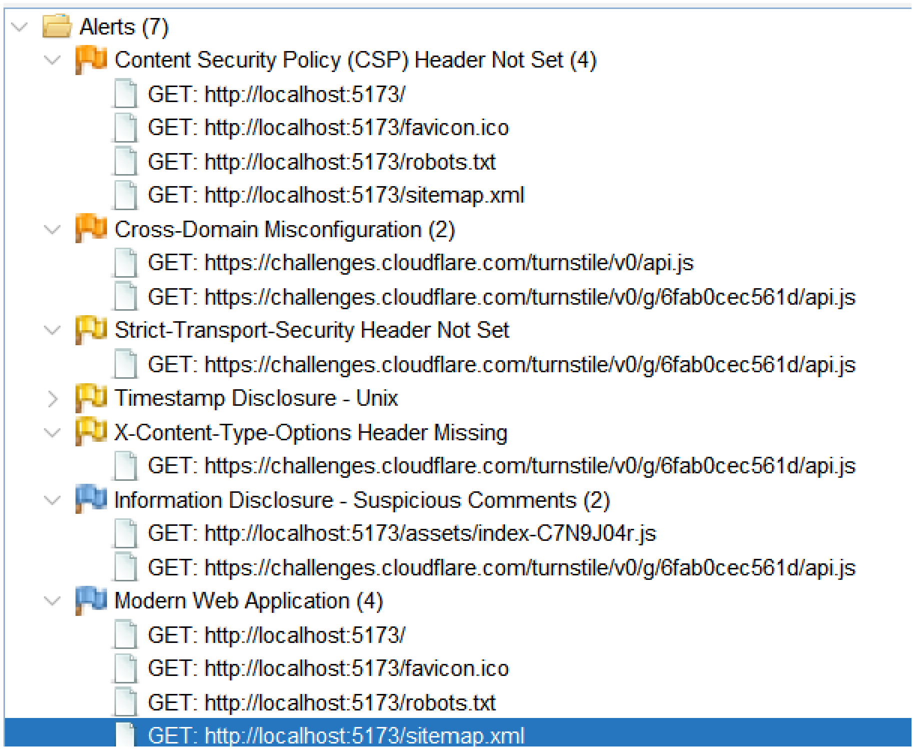
Problems ● → ●	Category	Type
orderCRUDService.js		
Missing await for an async function call	Async code and promises	Missing await for an asyn...
Missing await for an async function call	Async code and promises	Missing await for an asyn...
Missing await for an async function call	Async code and promises	Missing await for an asyn...
Missing await for an async function call	Async code and promises	Missing await for an asyn...
orderMiddleware.js		
Promise returned from validateBuyOrderBalance is ignored	Async code and promises	Result of method call retu...
Promise returned from validateSellOrderQuantity is ignored	Async code and promises	Result of method call retu...
Promise returned from validateLimitOrderPrice is ignored	Async code and promises	Result of method call retu...
Tables.jsx		
Promise returned from loadInitialData is ignored	Async code and promises	Result of method call retu...
Promise returned from loadInitialData is ignored	Async code and promises	Result of method call retu...
Portfolio.jsx		
Promise returned from fetchPortfolioDetails is ignored	Async code and promises	Result of method call retu...
Promise returned from fetchHoldings is ignored	Async code and promises	Result of method call retu...
userControllers.js		
Redundant 'await' for a non-promise type	Async code and promises	Redundant 'await' expres...
orderMatchingService.js		
Promise returned from emitOrderBookUpdate is ignored	Async code and promises	Result of method call retu...
Chart.jsx		
Promise returned from fetchData is ignored	Async code and promises	Result of method call retu...
Admin.jsx		
Promise returned from fetchSessionStatus is ignored	Async code and promises	Result of method call retu...
Trade.jsx		
Promise returned from fetchStocks is ignored	Async code and promises	Result of method call retu...
TradingSessionContext.jsx		
Promise returned from checkTradingStatus is ignored	Async code and promises	Result of method call retu...
tradingSessionController.js		
Promise returned from activateTradingSession is ignored	Async code and promises	Result of method call retu...
App.jsx		
Promise returned from handleGoogleAuth is ignored	Async code and promises	Result of method call retu...
orderBookController.js		
Promise returned from sendInitialData is ignored	Async code and promises	Result of method call retu...
stockManager.js		
Promise returned from manageStock is ignored	Async code and promises	Result of method call retu...
LoginForm.jsx		
Promise returned from handleGoogleCallback is ignored	Async code and promises	Result of method call retu...
TradingStatus.jsx		
Promise returned from fetchStatus is ignored	Async code and promises	Result of method call retu...

25 per page

© 2020-2025 Qodana

Terms Of Use

Dynamic Application Security Testing with OWASP ZAP



```
Alerts (7)
  Content Security Policy (CSP) Header Not Set (4)
    GET: http://localhost:5173/
    GET: http://localhost:5173/favicon.ico
    GET: http://localhost:5173/robots.txt
    GET: http://localhost:5173/sitemap.xml
  Cross-Domain Misconfiguration (2)
    GET: https://challenges.cloudflare.com/turnstile/v0/api.js
    GET: https://challenges.cloudflare.com/turnstile/v0/g/6fab0cec561d/api.js
  Strict-Transport-Security Header Not Set
    GET: https://challenges.cloudflare.com/turnstile/v0/g/6fab0cec561d/api.js
  Timestamp Disclosure - Unix
  X-Content-Type-Options Header Missing
    GET: https://challenges.cloudflare.com/turnstile/v0/g/6fab0cec561d/api.js
  Information Disclosure - Suspicious Comments (2)
    GET: http://localhost:5173/assets/index-C7N9J04r.js
    GET: https://challenges.cloudflare.com/turnstile/v0/g/6fab0cec561d/api.js
  Modern Web Application (4)
    GET: http://localhost:5173/
    GET: http://localhost:5173/favicon.ico
    GET: http://localhost:5173/robots.txt
    GET: http://localhost:5173/sitemap.xml
```

No alerts at ‘High‘ level, only alerts at ‘Medium‘, ‘Low‘ and ‘Informational‘ level left.