VIETNAM NATIONAL UNIVERSITY OF HO CHI MINH CITY
THE INTERNATION UNIVERSITY

PROJECT

COIN TRADING WEB APP - COINSANTRA

Semester 1 - ACADEMIC YEAR 2025-2026.

Course: Web Application Development

Course Code: IT093IU

INSTRUCTOR: Dr. Nguyen Van Sinh

Ho Chi Minh City, November 2025

LIST OF PARTICIPATING STUDENTS

Number	Fullname	Student ID
1	Nguyễn Mai Hoàng Huy	*ITCSIU22304*
2	Nguyễn Viết Anh Khoa	*ITCSIU22278*
3	Đỗ Huỳnh Duy Tiến	*ITCSIU22291*

Project: Developing an Coin Trading Platform - Coinsantra
Product: https://coinsantra.vercel.app
Github: https://github.com/anhkhoa13-dev/Trade_Web_App
Google Drive (Diagrams):
https://drive.google.com/drive/folders/1pGqHNCysVFoynFj2iDd_NdWdh9XiNbOd?usp=sharing

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGE

We would like to extend our heartfelt gratitude to everyone who supported us in completing this report on the development of the Coin Trading Web Application – COINSANTRA, which was carried out as part of the Web Application Development course during Semester 1 of the Academic Year 2025–2026.

First and foremost, we sincerely thank our instructor, Dr. Nguyen Van Sinh, for his invaluable guidance, professional expertise, and constructive feedback throughout the course. His insights and continuous support greatly contributed to shaping the technical direction, system architecture, and overall quality of this project.

Secondly, we would like to express our appreciation to all team members: Nguyễn Mai Hoàng Huy, Nguyễn Viết Anh Khoa, and Đỗ Huỳnh Duy Tiến. Each member demonstrated strong commitment, responsibility, and effective collaboration, and their combined efforts were essential to the successful development and deployment of the COINSANTRA platform.

Finally, we are deeply grateful to our families and friends for their patience, encouragement, and unwavering support throughout this journey. Their understanding and motivation played an important role in helping us overcome challenges and complete this project successfully.

In conclusion, we would like to thank everyone who contributed, directly or indirectly, to bringing this project to completion.

CHAPTER 1. INTRODUCTION

1.1. OBJECTIVES

The main goal of this project is to create a cryptocurrency trading platform that easy to use and focus primarily on automatic AI bot trading. It allows users to trade coins manually – using spot trading – or subscription to an "AI bot" that trade automatically with predefined bot wallet.

Our system is designed such that to help users manage their money, their bot subscriptions, tracking profit analysis and follow successful trading strategies from bots.

1.2. SCOPE

The project covers five main areas of functionality:

User Management

- **Sign Up & Login**: Users can create an account and log in securely using their email or Google account.
- **Profile:** Users can update their personal information and change their passwords.
- **Security:** The system uses secure tokens (JWT) to keep user accounts safe.

Wallet & Payments

- **Deposits:** Users can add money to their account using VNPay.
- **Asset Tracking:** Users can check their "Wallet" to see their total balance and the coins they currently own.

Manual Trading

- **Buy & Sell:** Users can manually buy and sell cryptocurrencies.
- **History:** Users can view a detailed history of all their past transactions to track their trading activity.

AI Bot Trading (Smart Features)

- **Bot Marketplace:** Users can browse a list of AI trading bots.
- **Performance Metrics:** The system shows detailed statistics for each bot, such as Profit (PnL) and Return on Investment (ROI), helping users choose the best one.
- **Copy Trading:** Users can "subscribe" to a bot. This means the system will automatically copy the bot's trades for the user.
- **Subscription Management:** Users can pause or stop copying a bot at any time.

Administration

- **Bot Management:** Admins can create, update, and delete trading bots.
- **System Management:** Admins can manage the system's coin inventory and define coin transaction fee (deposit and withdraw coins for the platform).

1.4 RESOURCE FEASIBLITITY

1.4.1. Team Size

- Nguyễn Mai Hoàng Huy – ITCSIU22304
- Nguyễn Viết Anh Khoa – ITCSIU22278
- Đỗ Huỳnh Duy Tiến – ITCSIU22291

1.4.2. Budget Analysis

Table 1-2: Budget

Category	Description	Estimated Cost
Development Hardware	Utilizing personal devices of our team members for project development	$0
Software & Tools	Using open sources and free software (Postman, VS Code, …)	$0
Database Service (Azure)	Hosting sql database	~ $13 / month
App Service (Azure)	Hosting spring boot backend with OS linux and B1 tier	~ 13.14 / month
Storage Account (Azure)	Azure Queue Storage for handling webhook messages	<$1.00 / month (Usage-based ~$0.0004 per 10,000 operations)
Azure functions	Serverless function for Webhook processing	~ free (1 million requests and 400,000 GB-seconds free every month)

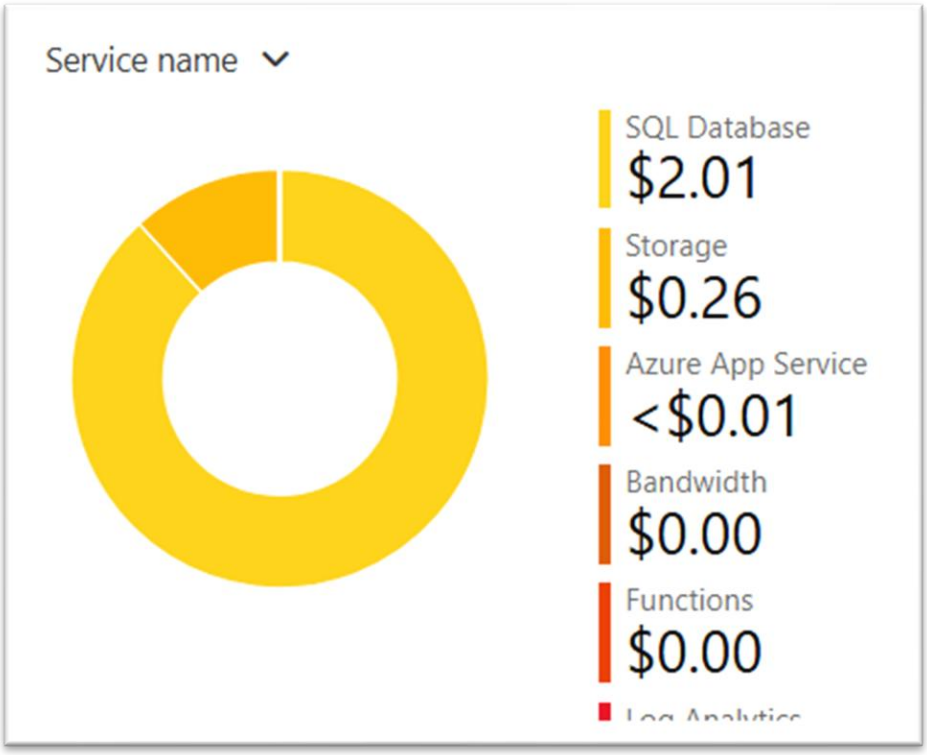

Figure 1.4-1. Cost of our project to present

As seen above, the project are hardly involve around azure services in order for the AI bot to work and communicate well with web application, which are discussed more on chapter 4 – section 4.1.3

1.4.3. Timeline

Table 1-3: Timeline

Weeks	Date	Main Deliverables
1 weeks	25/09 - 30/09	Define goals and scope, and analyze the project. Research technology (Frontend: Nextjs + React, Backend: Java Spring Boot, AI: FastAPI/LangChain, DB: SQL/MongoDB). Research some standards related to this project.
1 weeks	01/10 - 03/10	Analyze the project and research the functions to be implemented in the project. Draw use case diagrams, class diagrams.
2 weeks	05/10 - 19/10	Project setup, Authentication (NextAuth + JWT), and User Profile management.
3 weeks	20/10 - 09/11	Enable balance management, integrate real time market data aspects with CoinGecko API, integrate trading system
4 weeks	01/10 - 05/11	Implement AI bot trading logic, and AI bot management for Admin
1 weeks	15/11 - 22/11	Integrate VNPay portal for payment processing.
3 weeks	25/11 – 13/12	UI/UX refinement, responsive design, rigorous testing/bug fixing
1 weeks	14/12 21/12	Documentation & Deployment

1.5 Development Lifecycle:

1. Planning phase

- In this initial phase, the project foundation is established. The primary focus is defining the project goals, scope, and technical feasibility.
- Activities:
    - Defining core objectives and functional requirements.
    - Researching and selecting the technology stack:
        - Frontend: Next.js + React
        - Backend: Java Spring Boot
        - AI Engine: FastAPI with LangChain
        - Database: MySQL
    - Analyzing industry standards and compliance requirements for crypto-trading systems.

2. Design phase:

- This phase bridges the gap between requirements and code. The focus shifts to architectural modeling and system design.
- Activities:
    - Detailed analysis of system functions.
    - Creating architectural diagrams, including Use Case Diagrams to map user interactions and Class Diagrams to structure the backend logic.
    - Designing the database schema and API endpoints.

3. Development phase:

- This is the longest phase where the application is built iteratively. It is divided into specific functional milestones:
    - Core Infrastructure: Implementation of project setup, security protocols, authentication (NextAuth + JWT), User Profile management, and Admin management system.
    - Trading System: Integration of Wallet/Balance management and real-time market data via CoinGecko API. Implementation of the manual trading execution system.
    - AI Integration (Concurrent): Development of the AI bot trading logic using FastAPI/LangChain and the Admin management interface for bots.
    - Monetization: Integration of the VNPay payment portal to handle secure transactions.

4. Testing phase: Once the core features are implemented, the project moves into a testing state to ensure stability and usability.

- Activities:
    - UI/UX Refinement: Polishing the interface and ensuring responsive design across devices.
    - Bug Fixing: Identifying and resolving logic errors and edge cases.
    - System Testing: End-to-end testing of trading flows, AI signals, and payment

processing.

5. Deployment phase: The final stage involves preparing the application for the production environment.

- Activities**:**
    - Finalizing technical and user documentation.
    - Configuring production servers and environment variables.
    - Deploying the application to the live hosting environment.

6. Maintenance phase: To manage the source code effectively across the phases listed above, the team utilizes Git Flow, a branching model designed for project releases.

- Main (Master) Branch:
  Stores the official release history. The code here is always production-ready.
- Develop Branch:
  The integration branch for features. This contains the complete history of the project.
  CI/CD pipelines usually run tests against this branch during the Testing Phase.
- Feature Branches:
  Created from develop for every new deliverable (e.g., feature/vnpay-integration, feature/ai-bot-logic).
  Once a feature is complete, it is merged back into develop via a Pull Request (PR).
- Hotfix Branches:
  Used during the Maintenance Phase to quickly patch production issues. These are branched from main and merged back into both main and develop.

CHAPTER 2. PROBLEM ANALYSIS

2.1. PROBLEM DOMAIN

Cryptocurrency trading has become increasingly popular in recent years due to its high profit potential and continuous market operation. However, the crypto market is characterized by extreme volatility and rapid price changes, which require traders to constantly monitor the market and make timely decisions. For many individual investors, especially beginners, manual trading is challenging because it demands technical analysis skills, emotional discipline, and significant time commitment. These difficulties often lead to ineffective trading strategies and increased financial risk.

As a result, there is a growing need for trading platforms that support automation while remaining accessible and transparent to users. Many existing platforms either provide limited automation features or require complex configurations that are unsuitable for non-technical users. Moreover, integrating AI trading bots into a web-based system introduces additional challenges, such as secure communication, real-time signal processing, performance tracking, and scalability. Therefore, this project addresses the problem of building a reliable and user-friendly cryptocurrency trading platform that combines manual trading with AI-powered bot trading, allowing users to automate their strategies while maintaining control, security, and clear visibility of trading performance.

2.2. SECURITY ASPECTS

Password policy

We enforce a strict policy to prevent data breaches and secure user information by using best practices, therefore users' password:

- Must be securely hashed before inserted into database.
- Must meet the standards requirements like at least 10 characters long, include at least one uppercase, lowercase, number, and special character.

Session Timeout

Session management have been implemented to prevent account hijacking, which heavily relies on JWT token. The system handle session termination when requested token is expired or invalid, meaning that the system automatically executes logout functionality when token expired or refresh failed.

Rate Limiting

We also taking care of system overload problem by implementing the rate limiting, traffic controller.

Data Encryption

- Users' passwords are hashed before store in database.
- Bot's webhook token and secret key are hashed before store in database.

Input Validation

To ensure valid data and harmful-free from user's submission, both client-side and server-side are implemented in system:

- **Client-side validation:** Input data in form will be checked in real time before submission allowed and sent to server, which improves user experience.
- **Server-side validation:** Once the data sent from client, spring boot checked against the DTO (data transfer object) with defined constraint.

Role-Based Access Control

Role-Based Access Control have been implemented to ensure user have that roles can perform certain allowed actions. There are two main roles in the system: **ADMIN** and **TRADER.**

- Primarily, role-based validation happens when user requests endpoints in our system (except for public ones). client-side interface also automatically restrict user navigate to a protected pages – which can only be accessed by administrators.
- User roles are securely embedded directly into the authentication token (JWT) passed in the header, ensuring the server always knows who is making the request.

Authentication with JWT

To maintain high performance, stateless user verification, and security without storing session data on the server, we implemented **JSON Web Tokens (JWT)**.

- **Sign in:** successful authentication will result in a generated signed JWT that consists of 3 parts: header, payload, and signature. This token encodes essential user details - userID, role and an expiration time, which is then securely stored on the client side (HTTP-only cookie).
- **Request:** for every request to protected endpoints, token will be attached to header Authorization: Bearer<token>. The server will validate the signature against secret key and validate expiration time.

Environment variables

We strictly separate configuration from our code to prevent leaks

- These sensitive credentials like secret key for JWT, or external service keys (Google, Cloudinary, Azure …) are stored in two separate '.env' files, client and server.
- These files are never pushed to production, or GitHub (in development) and only shared securely between team members.

CHAPTER 3: REQUIREMENT CAPTURING

3.1. FUNCTIONAL REQUIREMENT

a) Authentication & Access:

The user allows to register, verify accounts, log in, refresh tokens, and log out securely with JWT and HTTP-only refresh cookies.

b) User Profile:

The user can view, update profile details, upload avatars, and change passwords.

c) Wallet & Assets:

The system provides traders with a wallet showing USD balance, net investment, and coin holdings.

d) Manual Trading:

The traders can buy and sell coins using current market prices, and updating wallet balances/holdings.

e) AI Bot Catalog (Public/User):

The system lists available AI trading bots with pagination/filtering and expose bot metrics for discovery.

f) Bot Subscriptions:

The traders can subscribe/copy a bot, allocate funds, set trade percentages, toggle active status, update config, and view subscription overviews/details with PnL/ROI.

g) Bot Signal Processing:

The system must accept authenticated bot signals (BUY/SELL) and execute trades for all active subscriptions, applying bot fees and recording trades.

h) Payments (VNPay):

The system must let traders initiate VND deposits via VNPay, return a payment URL, handle callbacks, and credit wallets on successful payment with recorded exchange rate.

i) Transaction History:

The system provides paginated histories for manual trades, bot trades (per subscription), and payments, with filtering/sorting; admins can view inventory history.

j) Admin Functions:

Admins are able to create/read/update/delete bots, view platform assets, and update coin fee settings.

3.2. NON-FUNCTIONAL REQUIREMENT

k) Performance Requirements

API Response Time: The system responds to API requests within 2 seconds under normal

load conditions and handles real-time market price fetching without blocking user operations.

Database Query Performance: The system uses pagination for all list endpoints to prevent loading excessive data, defaults to page size of 10-20 items per request, and supports configurable page sizes up to reasonable limits.

Concurrent User Support: The system supports multiple concurrent user sessions with JWT-based stateless authentication and handles concurrent trade executions with proper transaction isolation.

Asynchronous Signal Processing: The system processes bot signals asynchronously using Azure Functions with Storage Queue to prevent blocking webhook responses, and returns acknowledgment to signal sender immediately without waiting for trade execution.

l) Security Requirements

Authentication Mechanism: The system uses JWT for authentication with OAuth2 Resource Server pattern, combined with RS256 algorithm for token signing. The system must store access tokens in memory/session on client side. The system must store refresh tokens in HTTP-only, Secure, SameSite cookies.

Token Expiration: The system configures access tokens with short expiration time and refresh tokens with longer expiration time defined in application properties, and invalidates refresh tokens upon logout.

Password Security: The system hashs passwords using BCrypt algorithm.

Authorization: The system enforces role-based access control, and restricts admin endpoints to users and trading endpoints to users.

m) Reliability & Availability

Transaction Integrity: The system ensures atomic operations for trade execution (balance deduction + coin holding update) and rollbacks transactions on failure to maintain data consistency.

Data Consistency: The system uses optimistic locking where appropriate to prevent concurrent modification conflicts and validate data integrity constraints at database level.

n) Scalability Requirements

Horizontal Scaling: The system uses stateless JWT authentication to support horizontal scaling of backend servers and not rely on server-side session storage.

Database Connection Pooling: The system uses connection pooling (HikariCP) for efficient database resource management.

Lazy Loading: The system uses lazy loading for JPA entity relationships to optimize query performance.

o) Maintainability Requirements

Code Structure: The system follows domain-driven design with clear separation of concerns (Controller, Service, Repository layers), organizes backend code by feature modules (user, aibot, coin, vnpayment, history, admin) and frontend code by route groups and feature components.

Dependency Injection: The system uses constructor-based dependency injection with @RequiredArgsConstructor (Lombok) and prefers interface-based service contracts for

testability.

Configuration Management: The system externalizes configuration using application.properties and environment variables.

p) Usability Requirements

API Response Format: The system return consistent JSON response structures with data and metadata.

Error Messages: The system provides clear, actionable error messages for validation failures and uses field-level validation messages for form submissions.

Frontend User Experience: The system uses loading states for pages with data fetching, error boundaries for graceful error handling, and toast notifications for user feedback.

Theme Support: The system supports light and dark mode themes and persists theme preference across sessions.

q) Integration Requirements

API Documentation: The system documents API endpoints with clear parameter descriptions and examples.

External API Resilience: The system handles external API failures (CoinGecko, VNPay, Cloudinary) gracefully with appropriate error messages.

3.3. REQUIREMENT MODELLING

3.3.1. Use case diagram

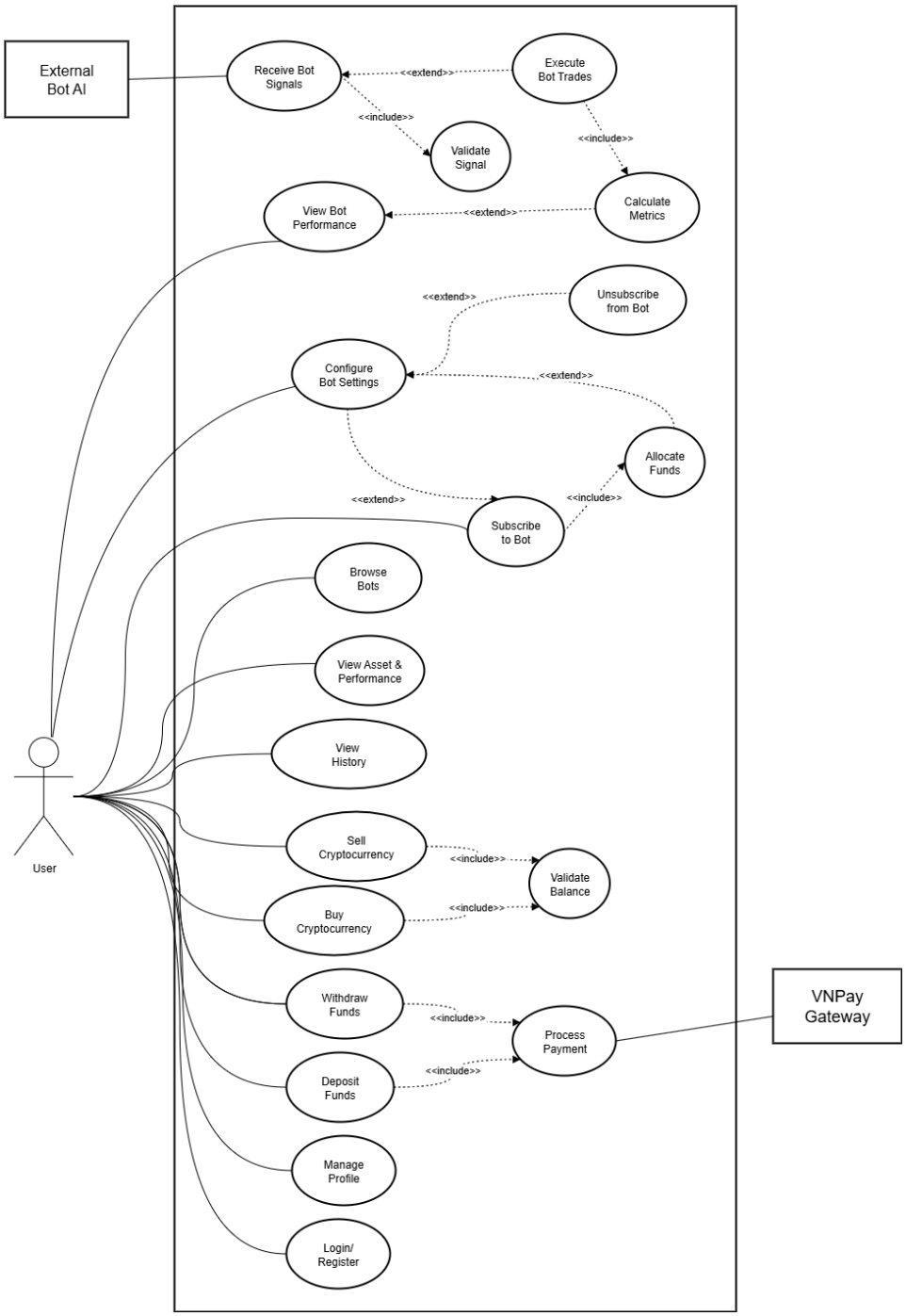

Figure 3.3-1: User Use Case diagram

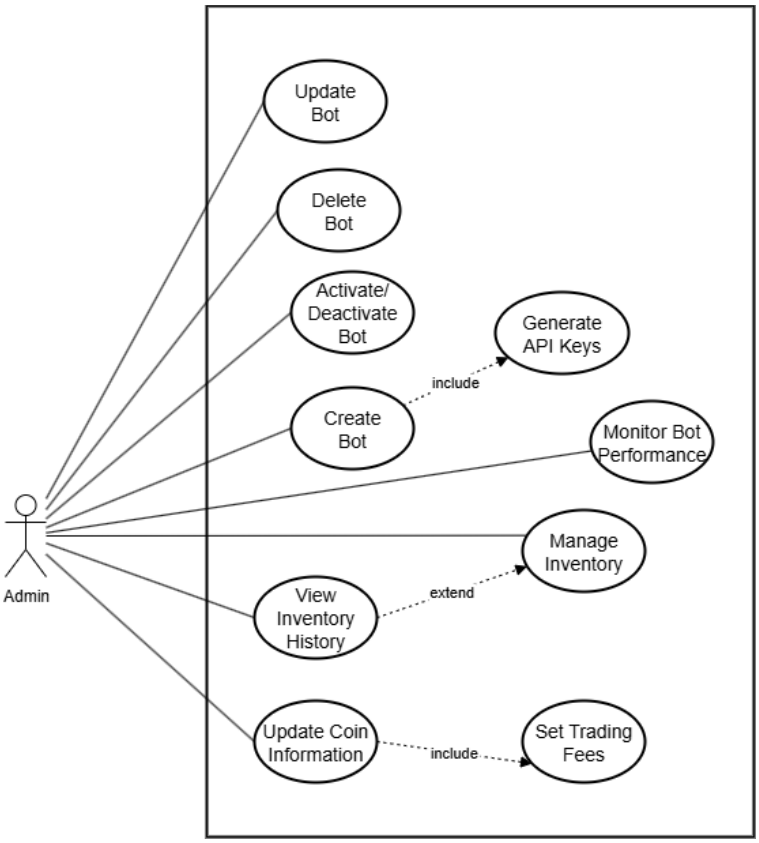

Figure 3.3-2: Admin Use Case diagram

3.3.2. Use case specification

1. Register

Table 1: Register specification

[1]	Sign Up
Actor	Guest User
Description	Guest users are able a new trading account
Pre-Conditions	User must have an email
Post-Conditions	User account created in LOCKED state; verification email sent
Main Flow	1. User navigates to registration page 2. User enters username, email, and password 3. System validates input (email format, password strength, unique email) 4. System creates user account with TRADER role 5. System creates empty wallet (balance = 0 USD) for the user 6. System generates activation token and URL token 7. System sends verification email with activation code 8. System displays success message and redirects to verification page
Exception/Alternate Flow	Email already exists and enabled → System returns "Email already taken" error Email exists but not verified → System overwrites previous registration data

2. Login with email

Table 2: User Login (Email)

[2]	Email log in
Actor	Registered User
Description	Authenticate and access the platform
Pre-Conditions	Account is activated
Post-Conditions	User authenticated; tokens issued; session active
Main Flow	1. User navigates to login page 2. User enters email and password 3. System authenticates credentials 4. System generates JWT access token (expires in 15-60 min) 5. System generates refresh token (long-lived) 6. System stores refresh token in database 7. System sets HTTP-only secure cookie with refresh token 8. System returns access token in response body 9. User redirected to homepage
Exception/Alternate Flow	Invalid credentials → System returns authentication error Account not enabled → System returns "Account not activated" error Account locked → System returns "Account locked" error

3. Login with google

Table 3: User Login (Google OAuth)

[3]	Google login
Actor	User

Description	Authenticate using Google account
Pre-Conditions	None
Post-Conditions	User authenticated; social auth profile linked
Main Flow	1. User clicks "Sign in with Google" button 2. System redirects to Google OAuth consent screen 3. User authorizes with Google 4. Google returns ID token 5. System verifies Google ID token 6. System extracts email, name, avatar from Google payload 7. System checks if user exists by email If exists: System updates profile with Google data If new: System creates user with TRADER role, enabled=true 8. System generates access and refresh tokens 9. User redirected to homepage

4. Logout

Table 4: Logout

[4]	**Google login**
Actor	Authenticated User
Description	End session and invalidate tokens
Pre-Conditions	User is logged in
Post-Conditions	Session terminated; tokens invalidated

Main Flow	1. User clicks logout button 2. System validates refresh token from cookie 3. System sets refresh token to NULL in database for the user 4. System deletes refresh token cookie (maxAge=0) 5. System returns success response 6. User redirected to login page

5. View profile

Table 5: View profile

[6]	View Profile
Actor	Authenticated User
Description	This use case allows user to view their own profile information
Pre-Conditions	User is logged in
Post-Conditions	Profile displayed
Main Flow	1. User navigates to profile page 2. System retrieves user data (name, email, phone, description, avatar, roles) 3. System displays profile information
Alternate Flow	None
Exception Flow	None

6. Update profile

Table 6: Update profile information

[7]	Update Profile
Actor	Authenticated User

Description	This use case allows trader to modify their profile details
Pre-Conditions	User is logged in
Post-Conditions	Profile updated
Main Flow	1. User edits profile fields (first name, last name, phone, description) 2. User submits form 3. System validates input 4. System updates user record 5. System displays success message
Exception/Alternate Flow	Phone number already exists → System returns uniqueness error

7. View wallet balance

Table 7: View wallet balance

[8]	View Wallet Balance
Actor	Trader
Description	Check current USD balance and coin holdings
Pre-Conditions	User is logged in as TRADER
Post-Conditions	Wallet information displayed
Main Flow	1. User navigates to wallet page 2. System retrieves wallet for user 3. System fetches all coin holdings with amounts and average buy prices 4. System displays: - Total USD balance - Net investment - List of coin holdings (symbol, amount, avg buy price)
Exception/Alternate Flow	None

8. Buy crypto

Table 8: Buy crypto

[9]	Buy Crypto
Actor	Trader
Description	Purchase cryptocurrency using USD balance
Pre-Conditions	User is logged in; user has sufficient USD balance; admin has sufficient coins
Post-Conditions	Trade executed; balances updated; transaction recorded
Main Flow	1. User selects coin and enters quantity to buy 2. User submits buy order 3. System fetches coin from database 4. System fetches current market price from CoinGecko API 5. System calculates: - Notional value = price × quantity - Fee = notional value × coin fee percentage - Total cost = notional value + fee 6. System validates user balance ≥ total cost 7. System validates admin has sufficient coin quantity 8. System deducts total cost from user wallet 9. System adds total cost to admin wallet 10. System deducts quantity from admin coin holding 11. System adds quantity to user coin holding (creates if doesn't exist) 12. System updates user's average buy price for the coin 13. System creates MANUAL BUY transaction record 14. System displays trade confirmation with execution details
Exception/Alternate Flow	Insufficient user balance → System returns error Insufficient admin coins → System returns error CoinGecko API fails → System returns error

9. Sell crypto

Table 9: Sell crypto

[10]	Sell Crypto
Actor	Trader
Description	Sell cryptocurrency for USD
Pre-Conditions	User is logged in; user has sufficient coin holdings; admin has sufficient USD
Post-Conditions	Trade executed; balances updated; transaction recorded
Main Flow	1. User selects coin and enters quantity to sell 2. User submits sell order 3. System fetches coin from database 4. System fetches current market price from CoinGecko API 5. System calculates: - Notional value = price × quantity - Fee = notional value × coin fee percentage - Net proceeds = notional value – fee 6. System validates user coin holding ≥ quantity 7. System validates admin wallet balance ≥ net proceeds 8. System deducts quantity from user coin holding (deletes if becomes 0) 9. System adds quantity to admin coin holding 10. System deducts net proceeds from admin wallet 11. System adds net proceeds to user wallet 12. System creates MANUAL SELL transaction record 13. System displays trade confirmation with execution details
Exception/Alternate Flow	Insufficient user coins → System returns error Insufficient admin USD → System returns error AI Bot Management (Admin)

10. Create trading bot

Table 10: Create trading bot

[11]	Create Trading Bot
Actor	Admin
Description	Set up new AI trading bot
Pre-Conditions	User is logged in as ADMIN
Post-Conditions	Bot created; webhook credentials generated
Main Flow	1. Admin navigates to bot management page 2. Admin enters bot details: - Name, description - Coin symbol (BTC, ETH, etc.) - Trading pair (BTC/USDT) - Risk level (LOW, MEDIUM, HIGH) - Category (AI_PREDICTIVE, TREND_FOLLOWING, DCA, etc.) - Bot fee percentage 3. Admin submits form 4. System validates input 5. System generates unique API URL for webhook 6. System generates API key and API secret 7. System creates bot with ACTIVE status 8. System displays bot details with API credentials (shown once)
Exception/Alternate Flow	None

11. Update trading bot

Table 11: Update trading bot

[12]	Update Trading Bot
Actor	Admin

Description	Modify bot configuration
Pre-Conditions	User is logged in as ADMIN; bot exists
Post-Conditions	Bot configuration updated
Main Flow	1. Admin selects bot to edit 2. Admin modifies settings (name, description, risk, category, fee) 3. Admin submits changes 4. System validates input 5. System updates bot record 6. System displays success message
Exception/Alternate Flow	None

12. Delete trading bot

Table 12: Delete trading bot

[13]	**Delete Trading Bot**
Actor	Admin
Description	Remove bot from platform
Pre-Conditions	User is logged in as ADMIN; bot exists
Post-Conditions	Bot deleted
Main Flow	1. Admin selects bot to delete 2. Admin confirms deletion 3. System deletes bot record 4. System displays success message
Exception/Alternate Flow	None

13. Browse available bot

Table 13: Browse available bot

[14]	Browse Available Bot
Actor	Trader
Description	View list of trading bots to subscribe to
Pre-Conditions	User is logged in
Post-Conditions	Bot list displayed
Main Flow	1. User navigates to AI Bots page 2. User applies filters (pagination, sort by PnL/ROI/copied, search) 3. System retrieves bots with metrics 4. System displays bot cards showing: - Bot name, description, coin symbol - Risk level, category - Performance metrics (PnL, ROI, copy count)
Exception/Alternate Flow	None

14. Subscribe bot

Table 14: Subscribe bot

[15]	Subscribe to Bot (Copy Bot)
Actor	Trader
Description	Start using a trading bot with allocated funds
Pre-Conditions	User is logged in; user has sufficient wallet balance and coin holdings; user not already subscribed to this bot
Post-Conditions	Subscription created; funds allocated; bot active
Main Flow	1. User selects bot to copy 2. User enters subscription settings: - Bot wallet balance (USDT to allocate)

	- Bot wallet coin (coin quantity to allocate) - Trade percentage per signal (1%-100%) - Optional: Max daily loss percentage 3. User submits subscription 4. System validates user is not already subscribed 5. System fetches current coin price from CoinGecko 6. System calculates net investment = bot wallet balance + (bot wallet coin × current price) 7. System validates user has sufficient assets 8. System deducts allocated USDT from user wallet 9. System deducts allocated coins from user holdings 10. System creates bot subscription record: botWalletBalance, botWalletCoin, netInvestment tradePercentage, active=true, startedAt=now 11. System displays subscription confirmation
Exception/Alternate Flow	User already subscribed → System returns error Insufficient user assets → System returns error

15. Update bot configuration

Table 15: Update bot configuration

[16]	**Update Bot Subscription Configuration**
Actor	Trader (Subscription Owner)
Description	Modify bot settings
Pre-Conditions	User owns the subscription
Post-Conditions	Subscription updated
Main Flow	1. User navigates to subscription details 2. User edits trade percentage or max daily loss 3. User submits changes

	4. System validates ownership 5. System updates subscription record 6. System displays success message
Exception/Alternate Flow	None

16. Toggle bot

Table 16: Toggle bot

[17]	**Toggle Bot**
Actor	Trader (Subscription Owner)
Description	Start or pause bot trading
Pre-Conditions	User owns the subscription
Post-Conditions	Subscription status updated; bot starts/stops trading
Main Flow	1. User clicks activate/pause toggle 2. System validates ownership a. If pausing: System sets active=false System sets stoppedAt=now b. If activating: System sets active=true System clears stoppedAt 3. System saves subscription 4. System displays status change confirmation
Exception/Alternate Flow	None

17. View subscription

Table 17: View subscription

[18]	**View Subscriptions**

Actor	Trader
Description	List all bot subscriptions with performance
Pre-Conditions	User is logged in
Post-Conditions	Subscription list displayed
Main Flow	1. User navigates to My Subscriptions page 2. User selects sort option (PnL, ROI, copied count) 3. System retrieves user's subscriptions 4. System calculates metrics for each: Current price from CoinGecko Total equity = botWalletBalance + (botWalletCoin × current price) PnL = total equity – netInvestment ROI = (PnL / netInvestment) × 100 5. System displays subscription list with metrics
Exception/Alternate Flow	None

18. View subscription details

Table 18: View subscription details

[19]	View Subscriptions Details
Actor	Trader (Subscription Owner)
Description	View detailed metrics for a specific subscription
Pre-Conditions	User owns the subscription
Post-Conditions	Detailed metrics displayed
Main Flow	1. User selects subscription 2. User selects timeframe (current, 1d, 7d) 3. System validates ownership

	4. System retrieves subscription with bot details 5. System fetches current coin price 6. System calculates: - Total equity, PnL, ROI - Win rate, trade count, average trade size 7. System retrieves bot trade history for subscription 8. System displays: - Bot info, subscription config - Performance metrics Trade history table
Exception/Alternate Flow	None

19. Receive trading signal from bot

Table 19: Receive trading signal bot

[20]	**Receive Trading Signal from External Bot**
Actor	External Python Bot
Description	Submit trading signal to platform
Pre-Conditions	Bot has valid API key/secret
Post-Conditions	Signal recorded; event queued
Main Flow	1. External bot detects trading opportunity 2. External bot sends POST request to webhook URL with: - API key/secret (authentication) - Action (BUY/SELL) - Coin symbol - Price, confidence - Timestamp 3. Azure Function receives webhook 4. System validates API credentials

	5. System creates BotSignal record with PENDING status 6. System stores raw payload for debugging System publishes SignalReceivedEvent to event queue 7. System returns 200 OK acknowledgment
Exception/Alternate Flow	Invalid API credentials → System returns 401 error

20. Process bot signal & execute auto trade

Table 20: Process bot signal & execute trade

[21]	**Process Bot Signal and Execute Trades**
Actor	System (Event Handler)
Description	Execute trades for all active subscriptions based on signal
Pre-Conditions	BotSignal exists with PENDING status
Post-Conditions	Trades executed for all active subscriptions; signal marked completed
Main Flow	1. System receives SignalReceivedEvent 2. System updates signal status to PROCESSING 3. System validates signal price > 0 4. System retrieves all active subscriptions for the bot 5. For each subscription: - System calculates trade amount = botWalletBalance × tradePercentage a. If BUY: System deducts USDT from user wallet System adds USDT to admin wallet System deducts coin from admin holdings System adds coin to user holdings b. If SELL: System deducts coin from user holdings

	System adds coin to admin holdings System deducts USDT from admin wallet System adds USDT to user wallet 6. System applies bot fee and system fee 7. System updates subscription botWalletBalance and botWalletCoin 8. System creates BOT Transaction record 9. System creates BotTrade record with signal reference 10. System updates signal status to COMPLETED 11. System sets signal processedAt timestamp
Exception/Alternate Flow	Invalid price → System logs error; no trades executed Individual trade fails → System logs error; continues with other subscriptions Insufficient user/admin balance → Trade skipped for that subscription

21. Deposit

Table 21: Deposit

[22]	Deposit
Actor	Trader
Description	Add funds to wallet via VNPay
Pre-Conditions	User is logged in
Post-Conditions	Payment transaction created; user at VNPay gateway
Main Flow	1. User navigates to deposit page 2. User enters amount in VND or USD 3. If USD selected, system converts to VND using real-time exchange rate 4. System validates minimum deposit (10,000 VND) 5. User submits deposit request 6. System generates unique vnpTxnRef 7. System creates PaymentTransaction record with PENDING status

	8. System builds VNPay payment URL with parameters: - Amount (VND × 100) - Transaction reference - Return URL - Checksum (HMAC SHA-512) 9. System returns VNPay URL 10. User redirected to VNPay gateway
Exception/Alternate Flow	None

22. Process VNPay

Table 22: Process VNPay Callback

[23]	**Process VNPay Callback**
Actor	VNPay System
Description	Complete payment and credit user wallet
Pre-Conditions	User completed payment at VNPay
Post-Conditions	Wallet credited (if success); transaction status updated
Main Flow	1. VNPay redirects user back with payment result 2. System receives callback with payment parameters 3. System validates VNPay checksum signature 4. System retrieves PaymentTransaction by vnpTxnRef 5. System checks if already processed (idempotency) 6. If payment successful (responseCode=00, transactionStatus=00): a. System fetches real-time USD/VND exchange rate b. System calculates USD amount = VND amount ÷ exchange rate c. System updates transaction: - status=SUCCESS

	- convertedAmount=USD amount - exchangeRate=current rate d. System credits user wallet balance with USD amount e. System updates wallet netInvestment with USD amount f. System displays success message 7. If payment failed: - System updates transaction status=FAILED - System displays failure message
Exception/Alternate Flow	Invalid checksum → System returns error Already processed → System returns cached result

23. View manual transaction history

Table 23: View manual transaction history

[24]	**View Manual Transaction History**
Actor	Trader
Description	Review past manual trades
Pre-Conditions	User is logged in
Post-Conditions	Transaction history displayed
Main Flow	1. User navigates to transaction history 2. User applies filters: - Trade type (BUY/SELL) - Coin name (partial match) - Date range (from/to) - Sort by (date, quantity, price, value) - Pagination 3. System retrieves filtered transactions with source=MANUAL 4. System displays transaction list with: Date, type, coin, quantity, price, value, fee

Exception/Alternate Flow	None

24. View bot transaction history

Table 24: View bot transaction history

[25]	View Bot Transaction History
Actor	Trader
Description	Review trades executed by a specific bot subscription
Pre-Conditions	User owns the subscription
Post-Conditions	Bot trade history displayed
Main Flow	1. User navigates to bot transaction history 2. User selects subscription (required) 3. User applies filters (type, coin, date range, sort) 4. System validates ownership 5. System retrieves bot trades for subscription 6. System displays trade list with: Date, type, bot name, coin, quantity, price, value, bot fee
Exception/Alternate Flow	None

25. View payment history

Table 25: View payment history

[26]	View Payment History
Actor	Trader
Description	Review deposit transactions
Pre-Conditions	User is logged in
Post-Conditions	Payment history displayed

Main Flow	1. User navigates to payment history 2. User applies filters: - Status (SUCCESS, PENDING, FAILED) - Sort by (date, amount, status) - Pagination 3. System retrieves payment transactions for user 4. System displays payment list with: Date, VND amount, USD converted amount, exchange rate, status
Exception/Alternate Flow	None

26. View bot grid with Metrics

Table 26: View bot grid with metrics

[27]	**View Bot Grid with Metrics**
Actor	All Users
Description	Browse trading bots with performance data
Pre-Conditions	None
Post-Conditions	Bot grid displayed with performance data
Main Flow	1. User navigates to bot marketplace/grid 2. User selects timeframe (current, 1d, 7d) 3. User selects sort criteria (PnL, ROI, copy count) 4. User enters search query (optional) 5. System retrieves bots with filters 6. System calculates metrics for each bot: - Total PnL across all subscriptions - Average ROI - Number of subscribers (copy count) 7. System displays bot grid with metrics
Exception/Alternate Flow	None

27. View platform assets

[27]	View Platform Assets
Actor	Admin
Description	Monitor platform wallet and holdings
Pre-Conditions	User is logged in as ADMIN
Post-Conditions	Platform assets displayed
Main Flow	1. Admin navigates to admin dashboard 2. System retrieves admin user wallet 3. System displays admin asset summary
Exception/Alternate Flow	None

28. Update coin fee

Table 27: Update coin fees

[29]	Update Coin Trading Fees
Actor	Admin
Description	Adjust fee percentages for coins
Pre-Conditions	User is logged in as ADMIN
Post-Conditions	Coin fees updated
Main Flow	1. Admin navigates to fee management 2. Admin selects coin and enters new fee percentage 3. Admin submits change 4. System updates coin fee 5. System displays success message
Exception/Alternate Flow	None

3.3.3. Class diagram

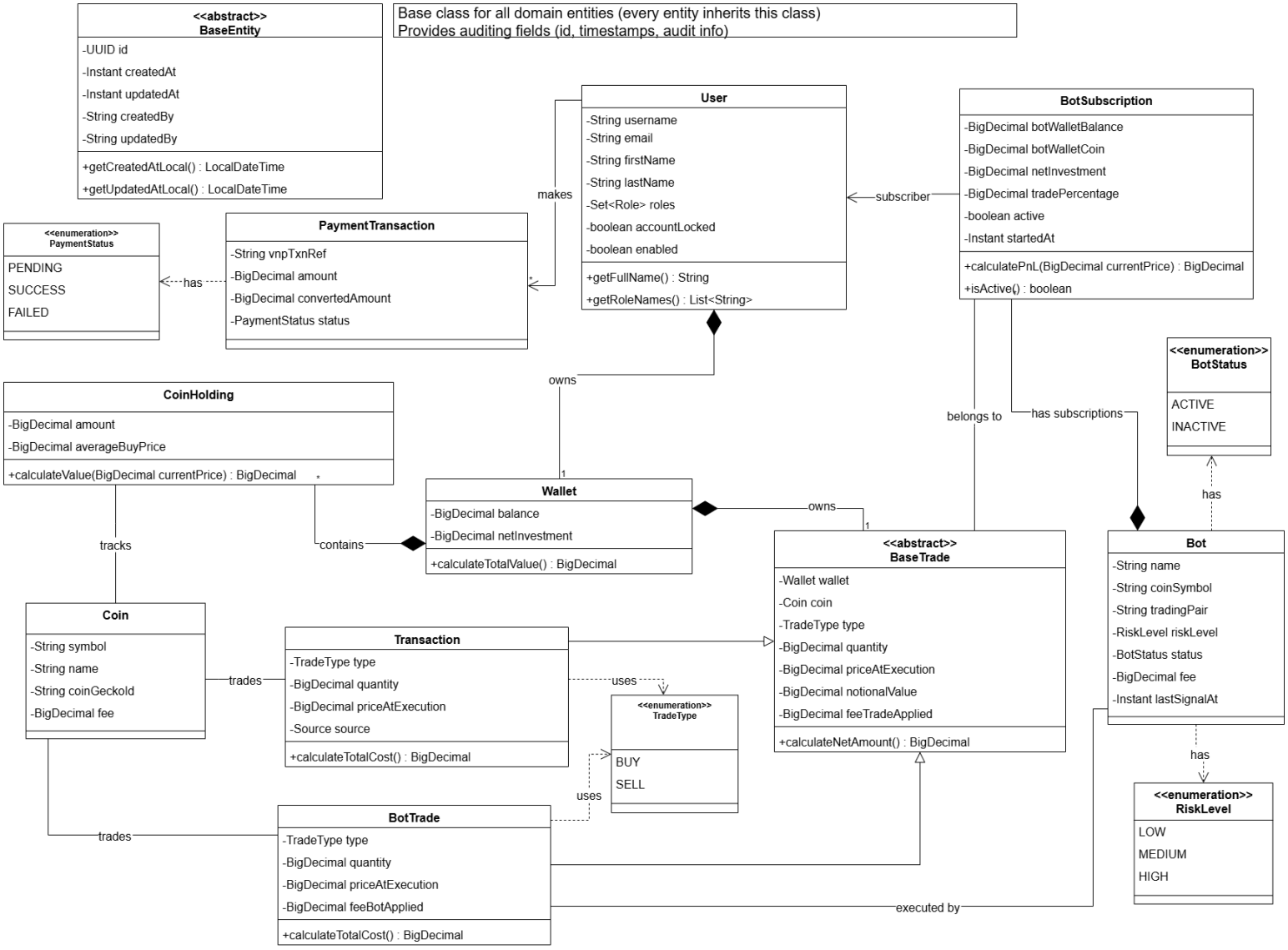

Figure 3.3-3: Class diagram

3.3.4. Sequence diagram

3.3.4.1 **General** CRUD sequence

1. Create

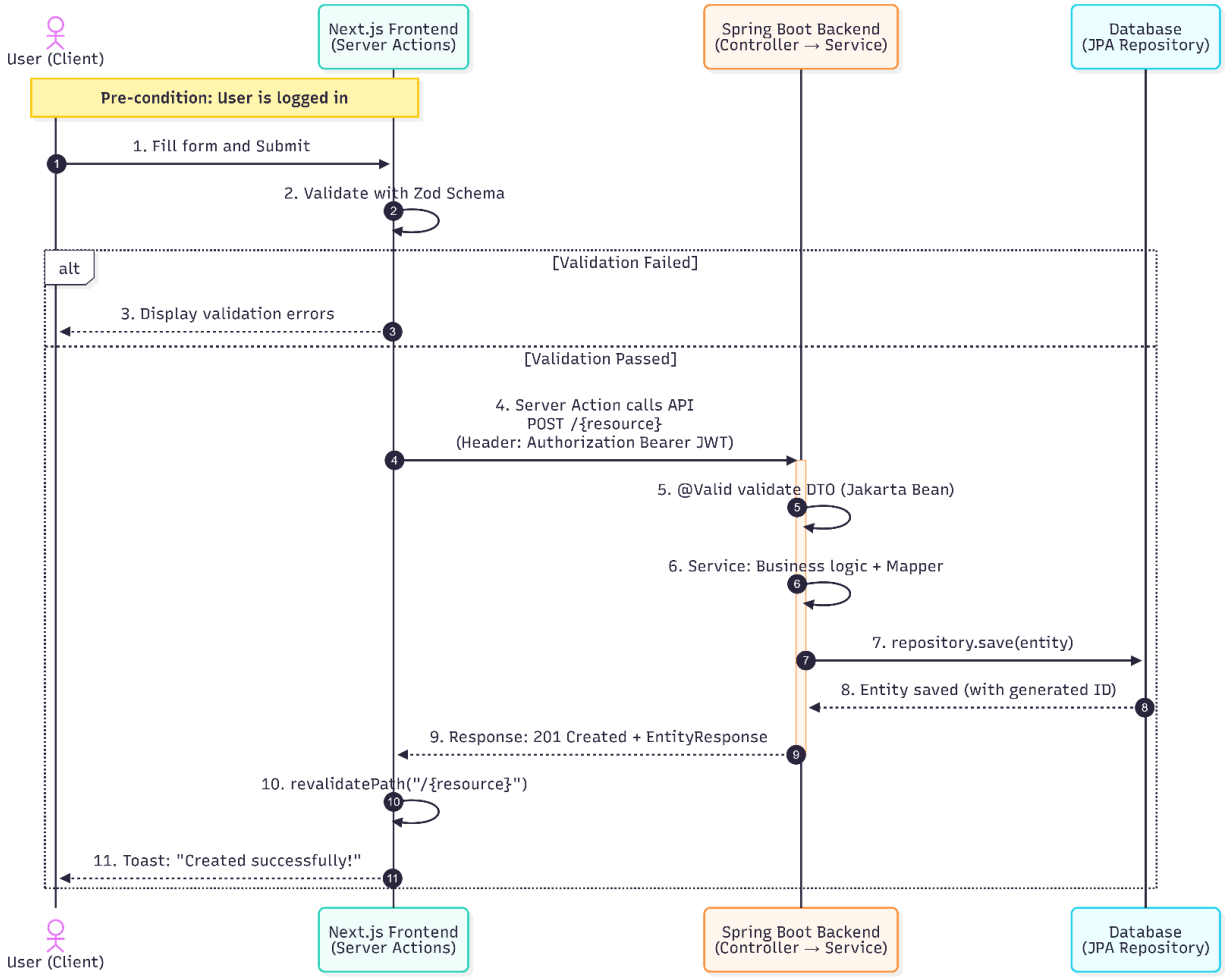

Figure 3.3-4: Generic CREATE Sequence diagram

2. Read (List)

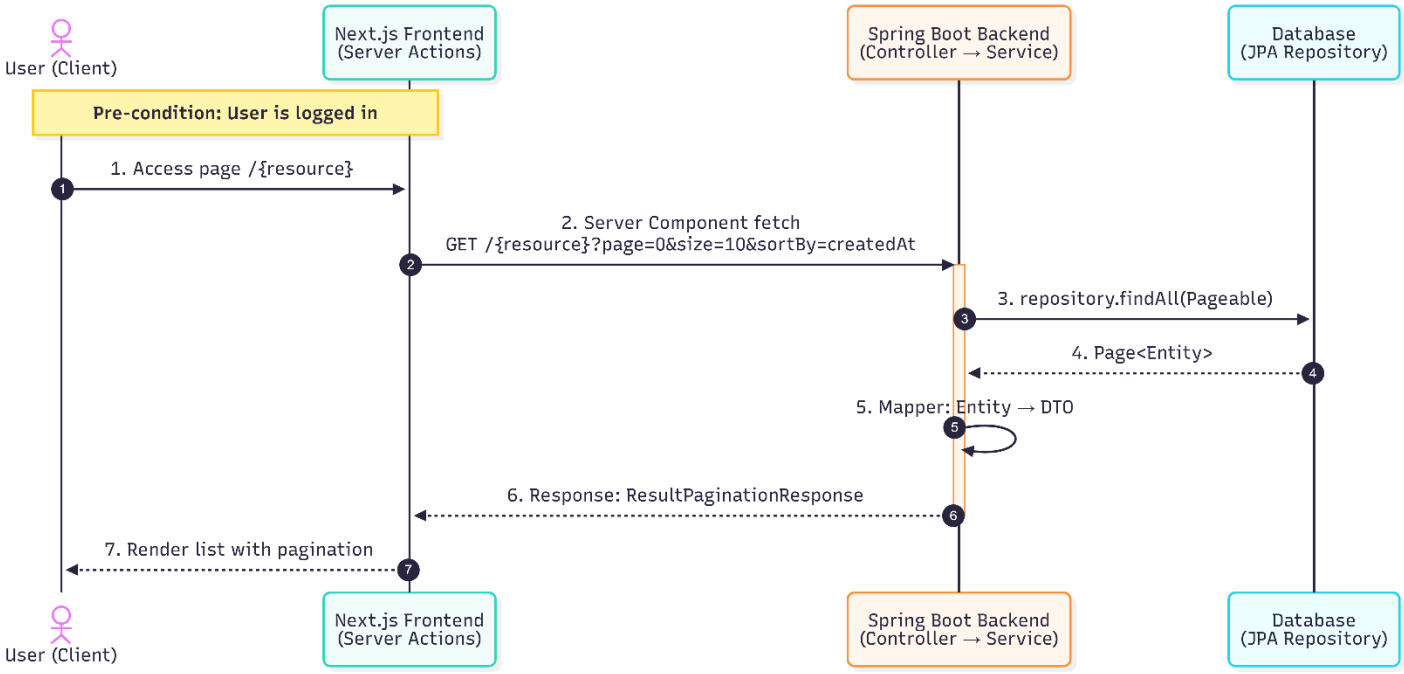

Figure 3.3-5: Generic READ Sequence diagram

3. Read (Detail)

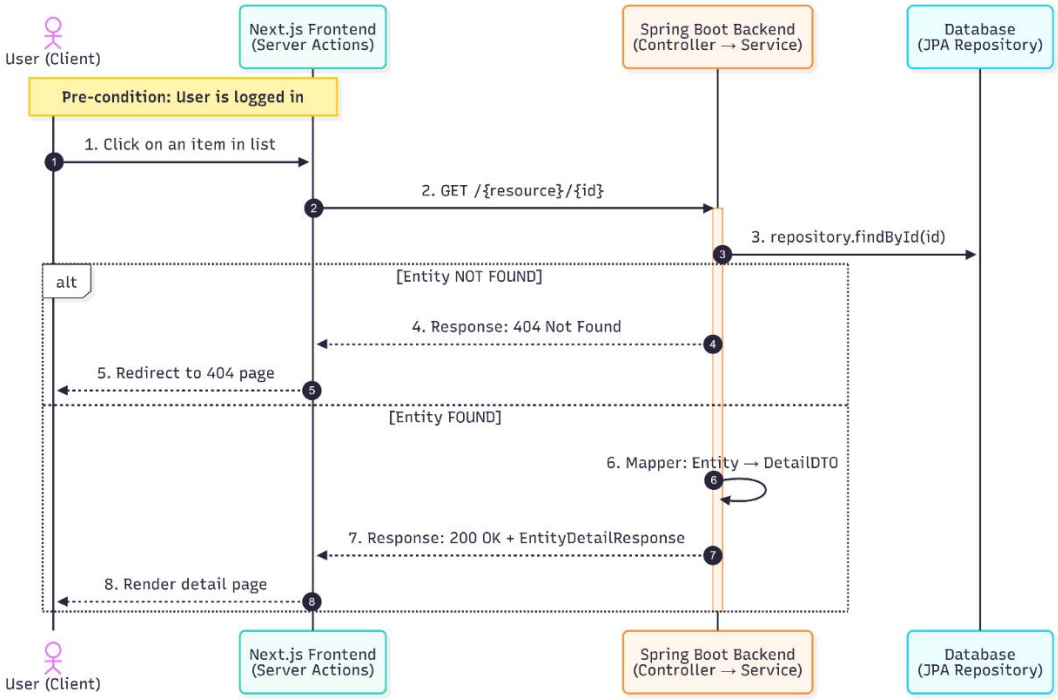

Figure 3.3-6: Generic READ details Sequence diagram

4. Update

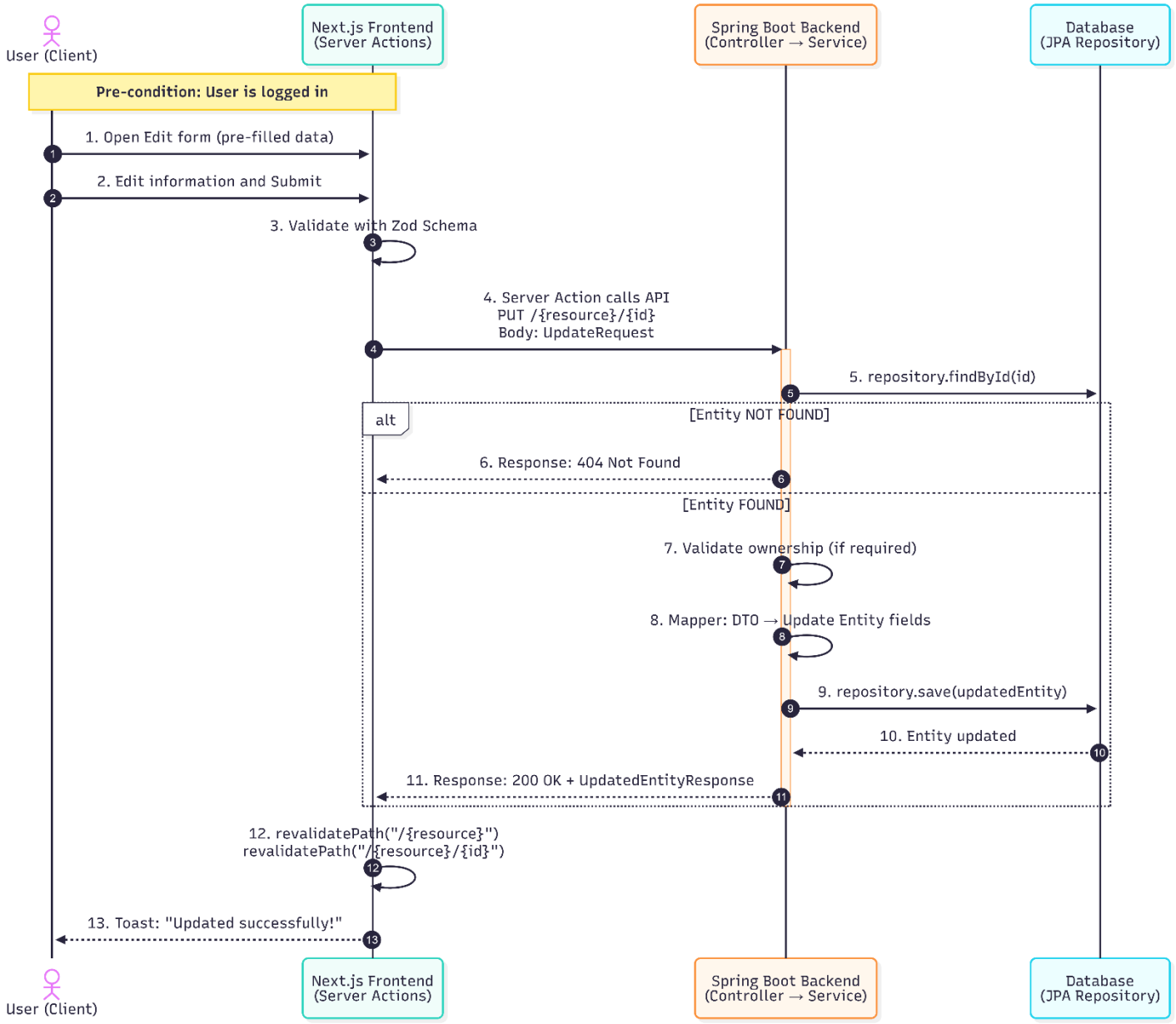

Figure 3.3-7: Generic UPDATE sequence diagram

5. Delete

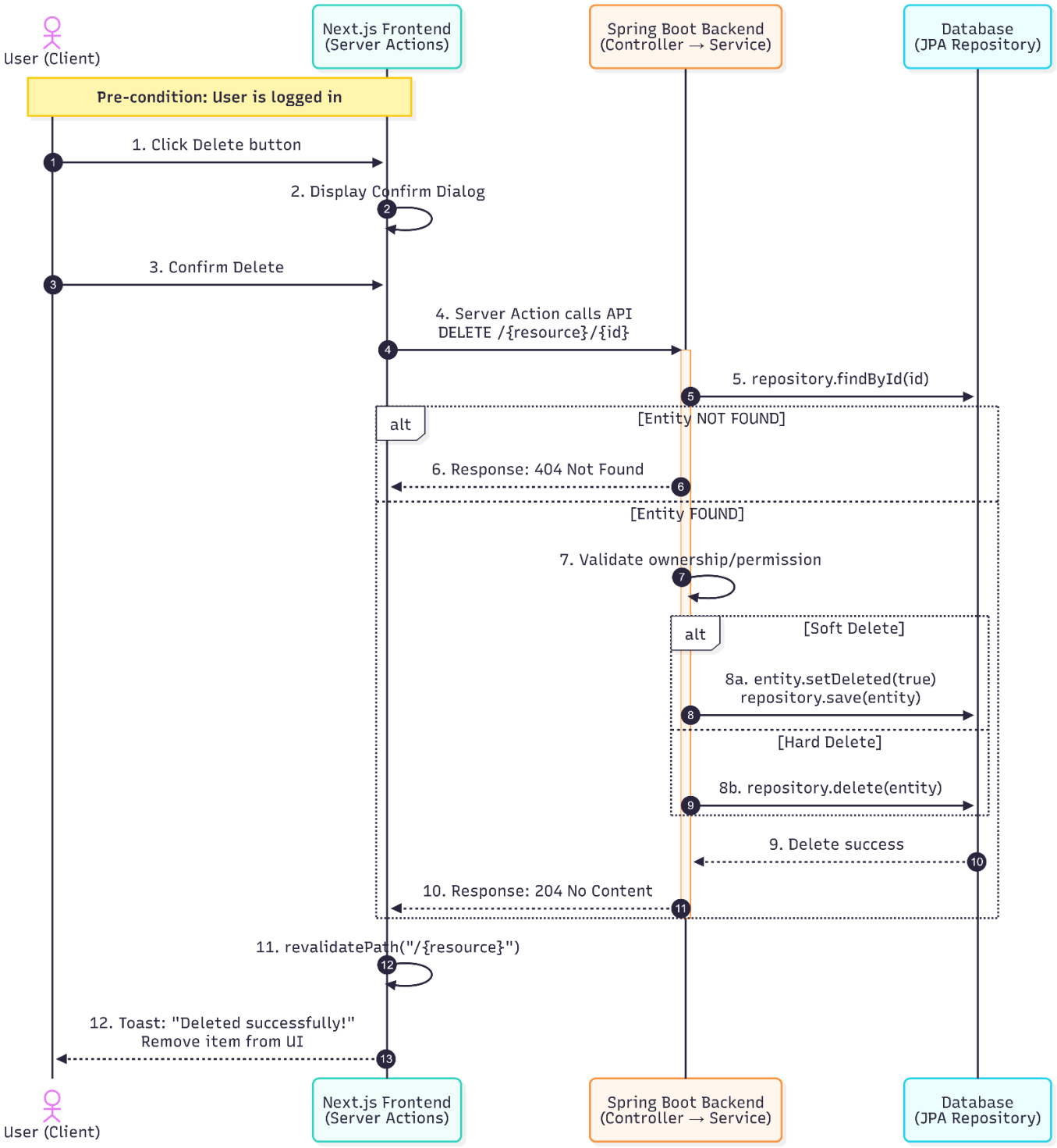

Figure 3.3-8: Generic DELETE Sequence diagram

3.3.4.2 Some specific sequences

a) Sign Up

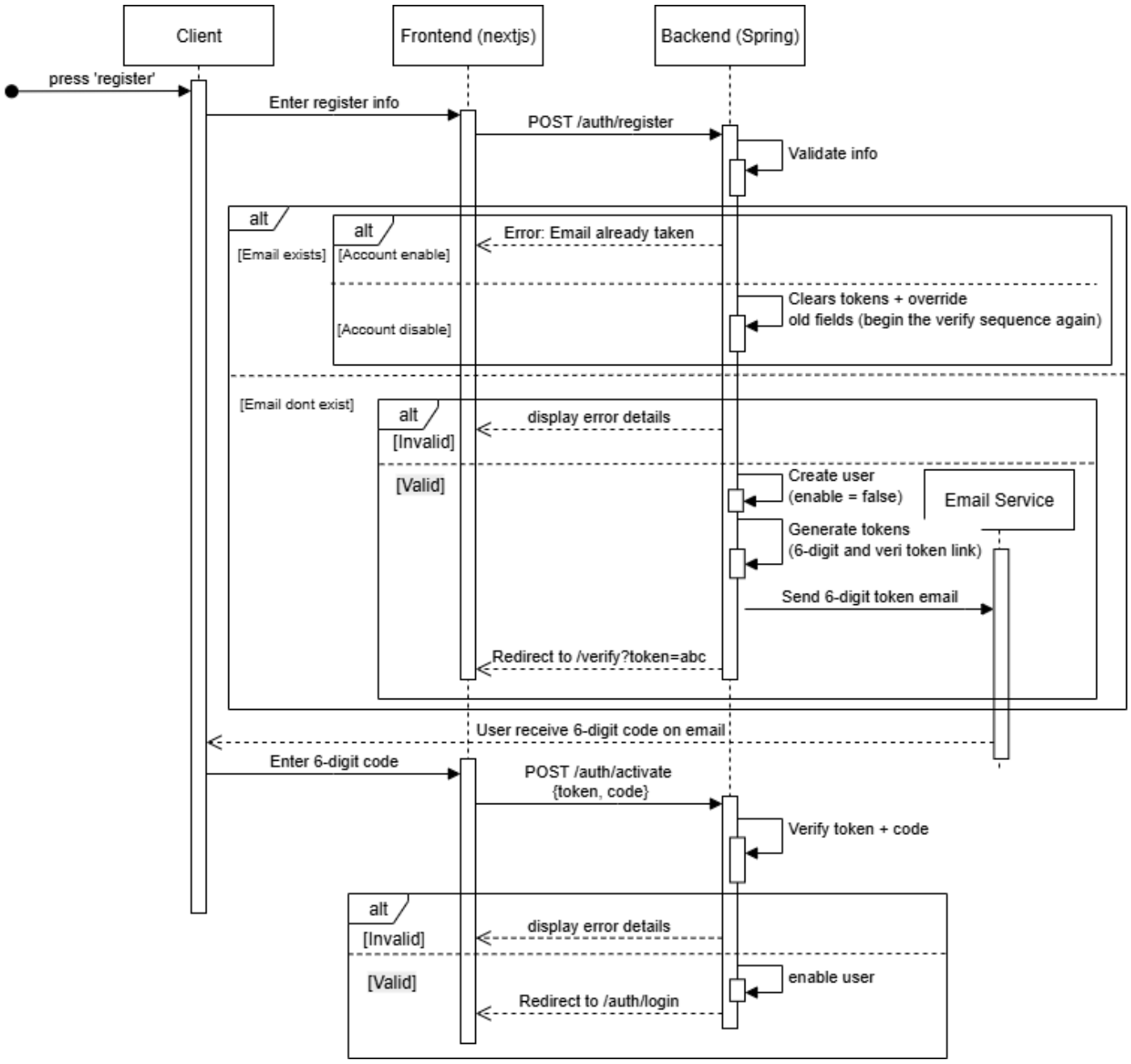

Figure 3.3-9: Sign Up Sequence diagram

b) Login with Credentials

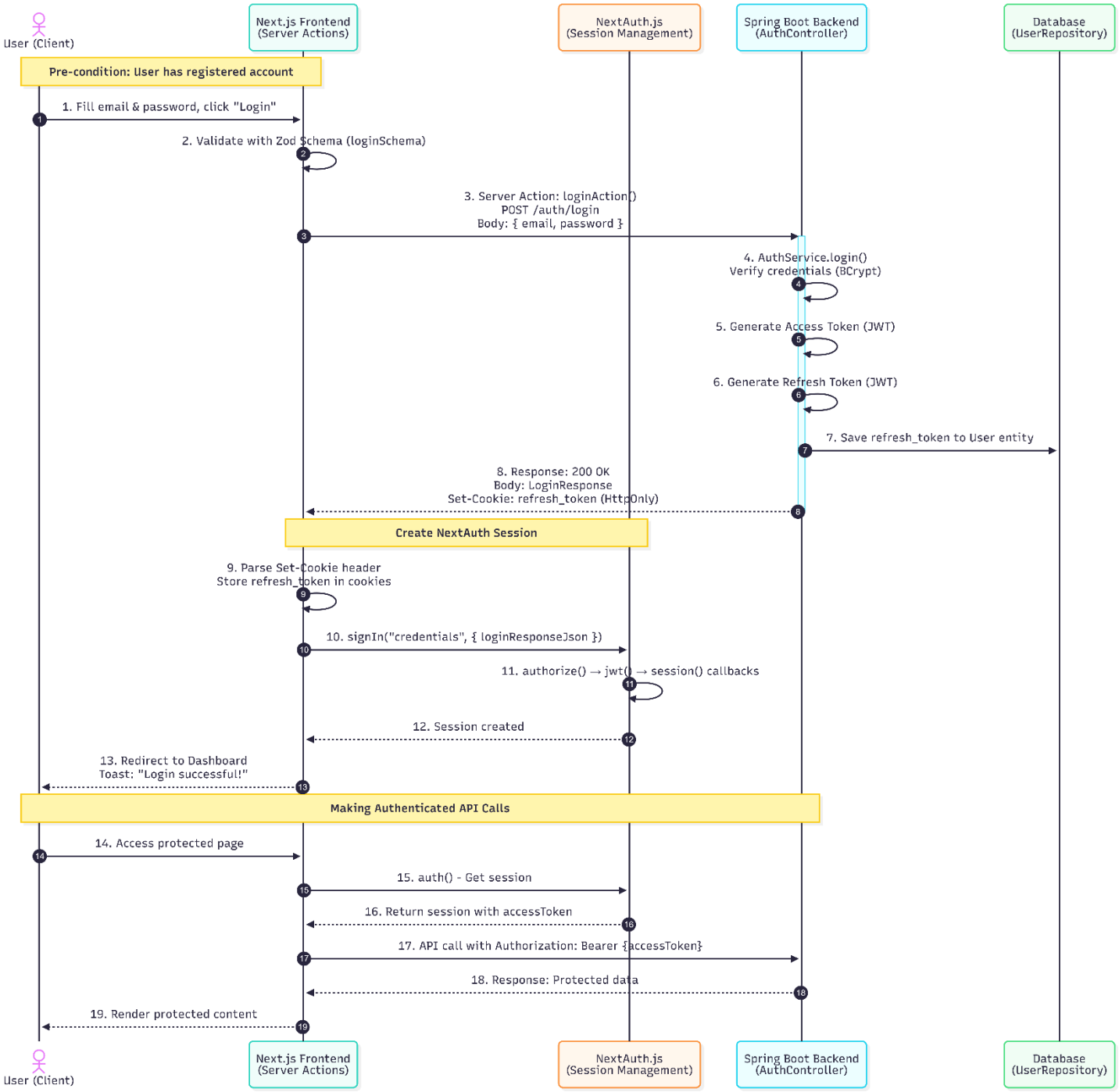

Figure 3.3-10: Login with Credentials Sequence diagram

c) **Login with google:**

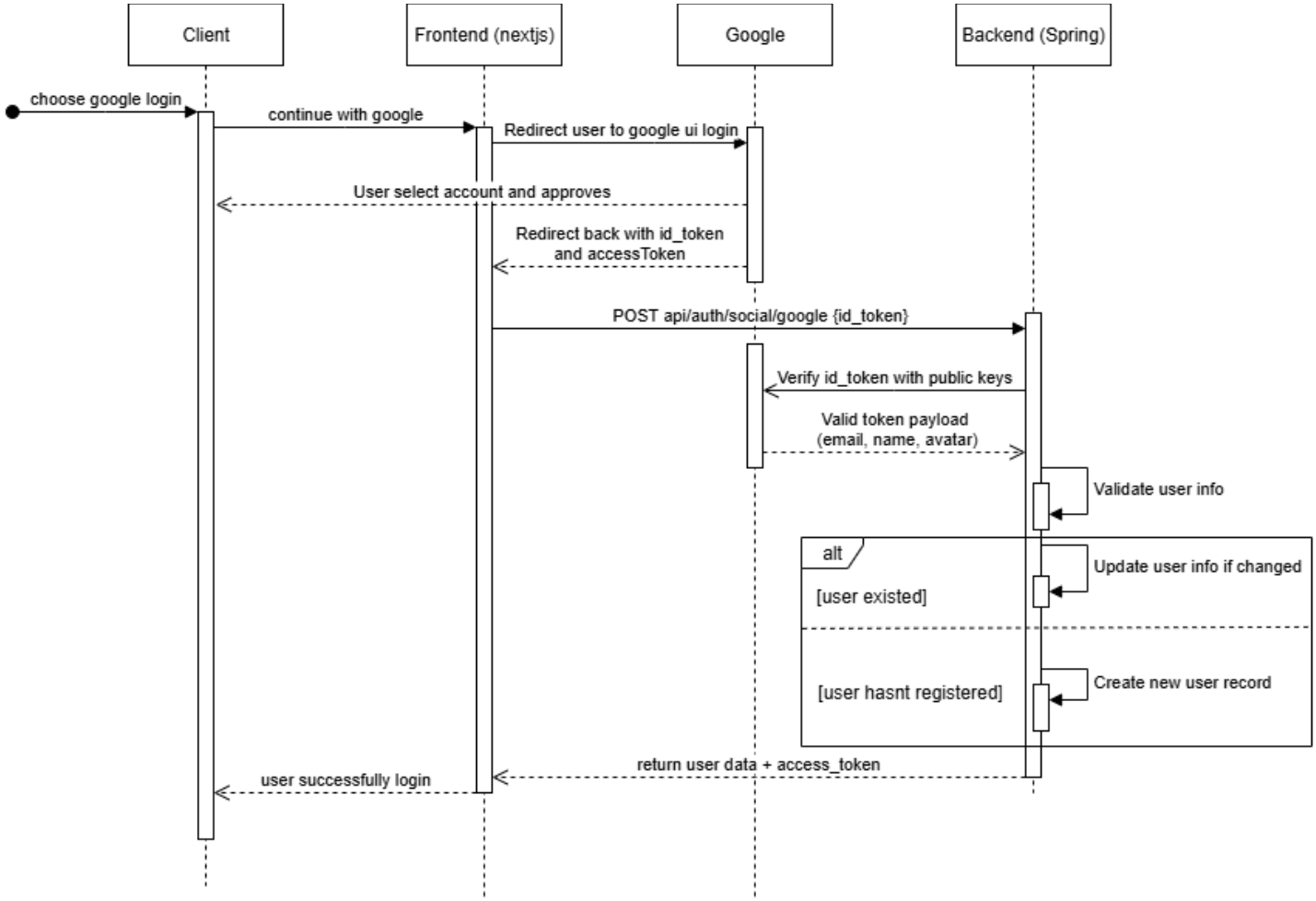

Figure 3.3-11: Login with Google Sequence diagram

d) Refresh Token

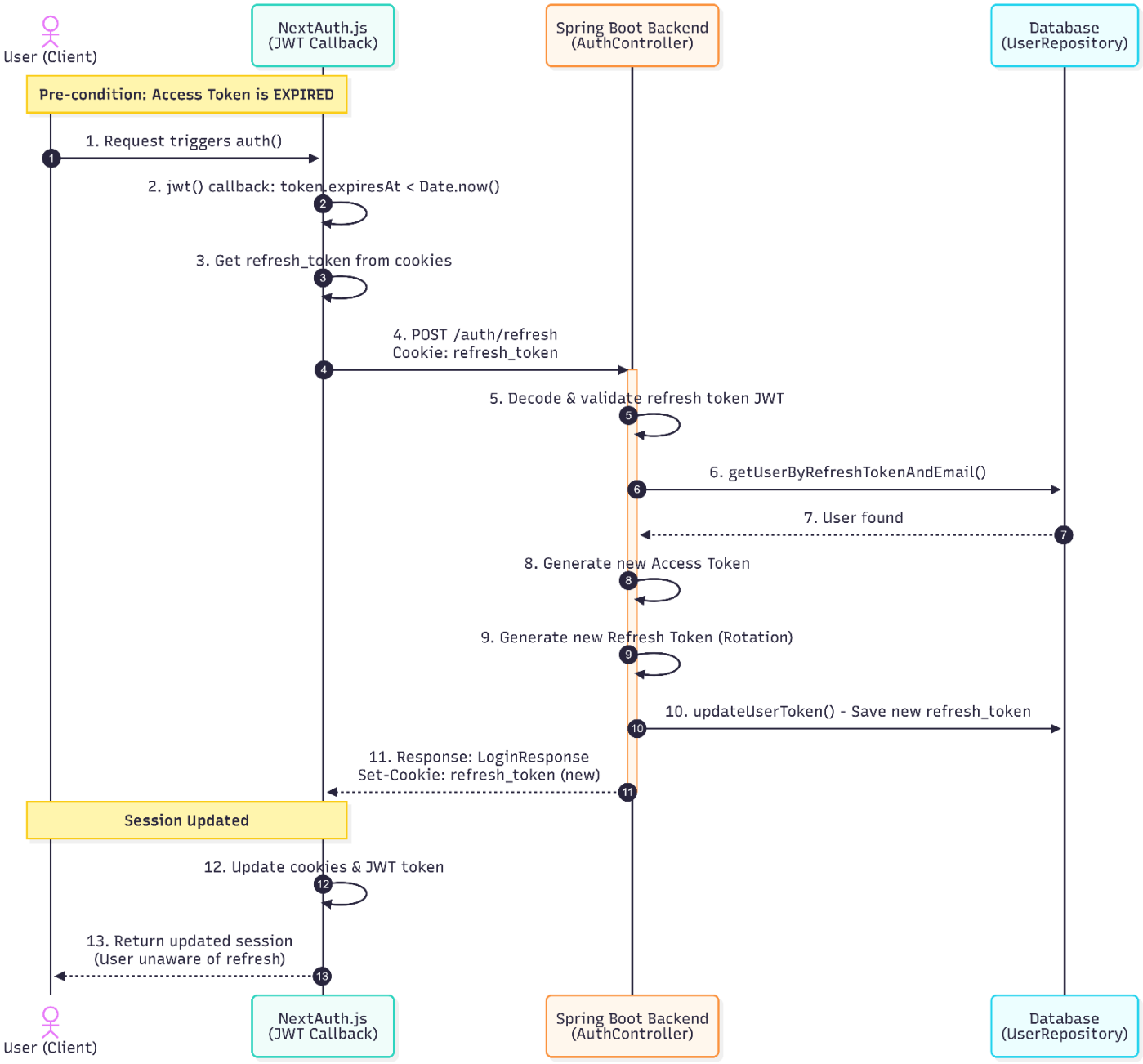

Figure 3.3-12: Refresh Token Sequence diagram

e) Buy Crypto

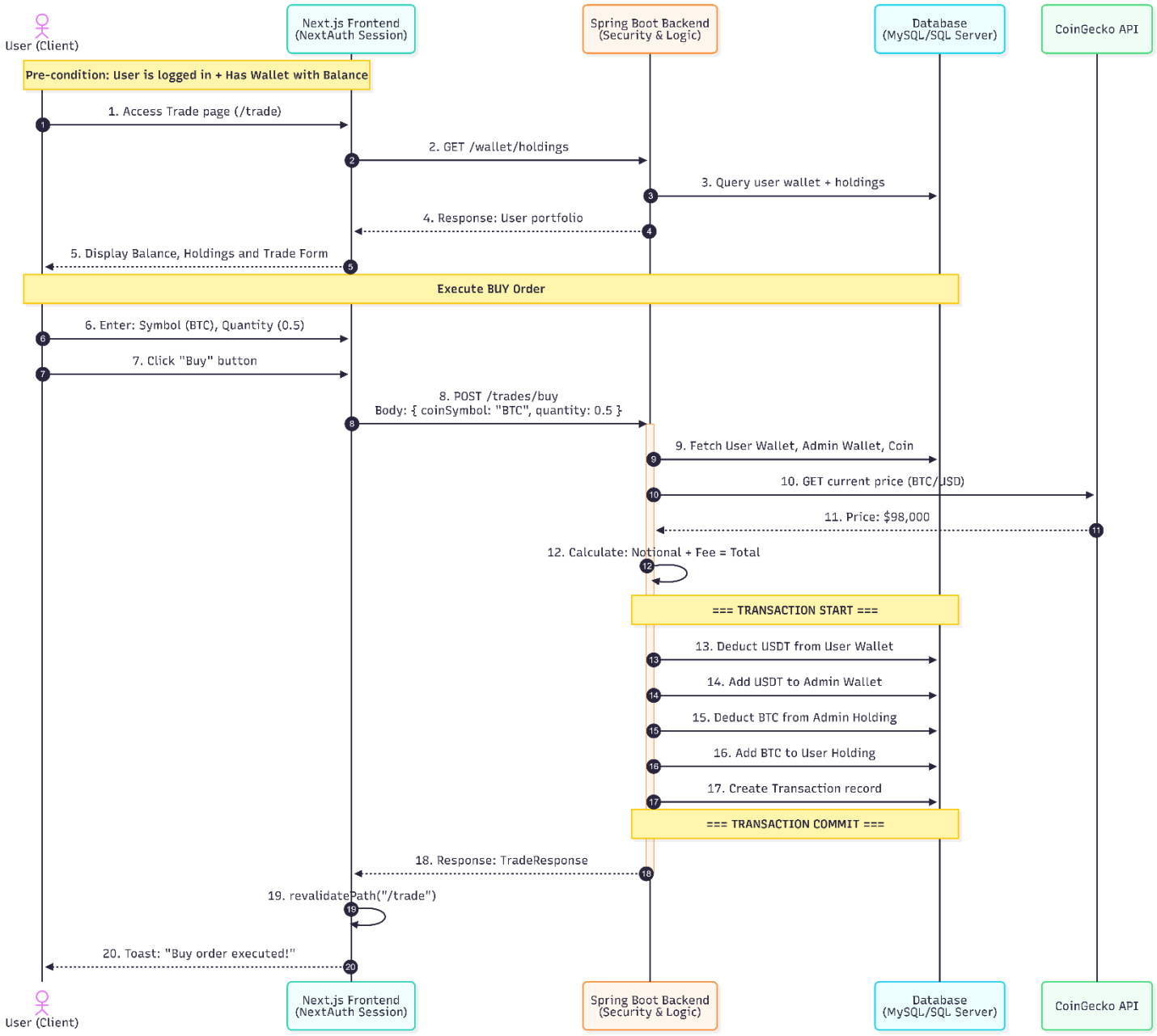

Figure 3.3-13: Buy Coin Sequence diagram

f) Sell Crypto

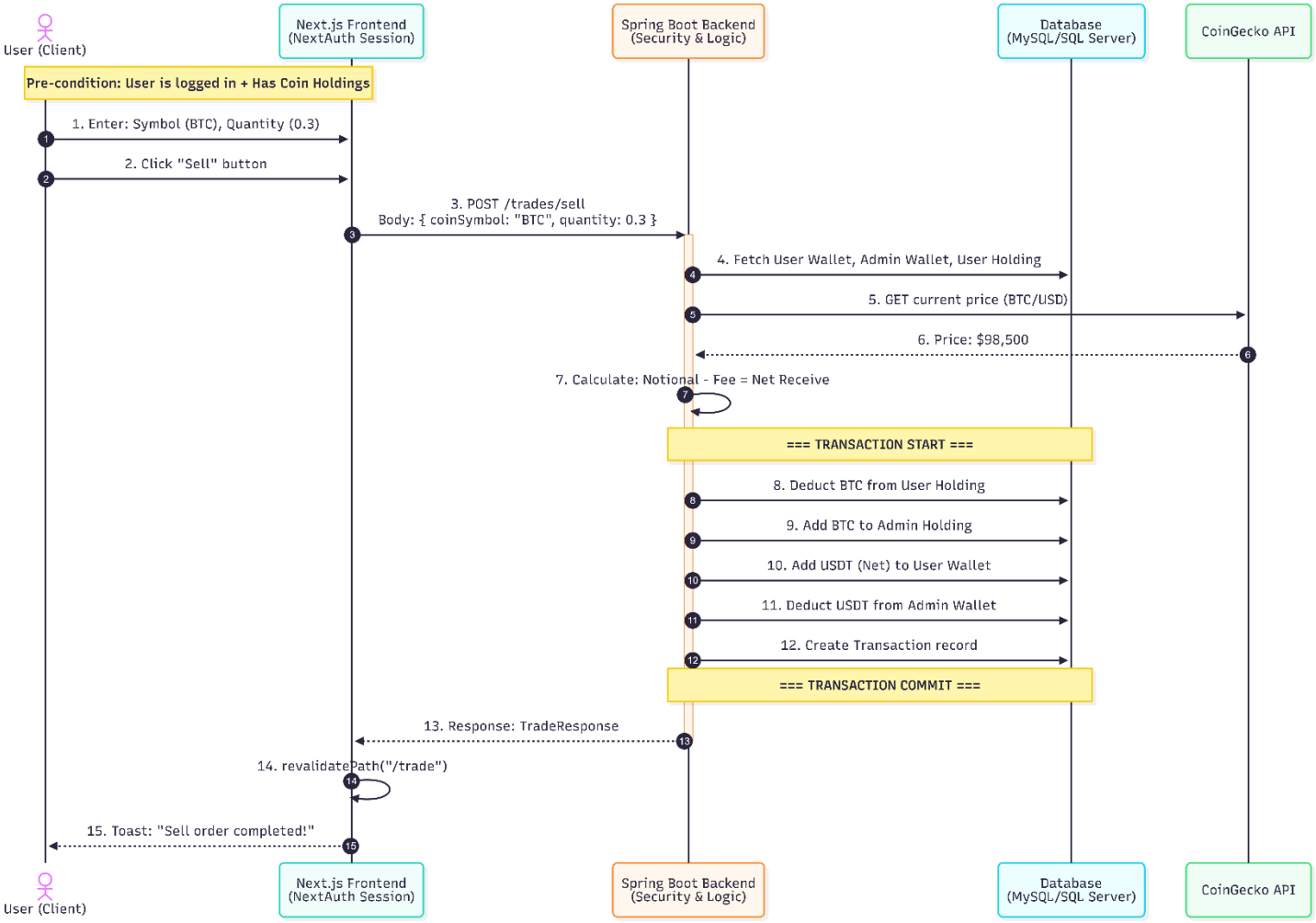

Figure 3.3-14: Sell Coin Sequence diagram

g) Copy Trading Subscription

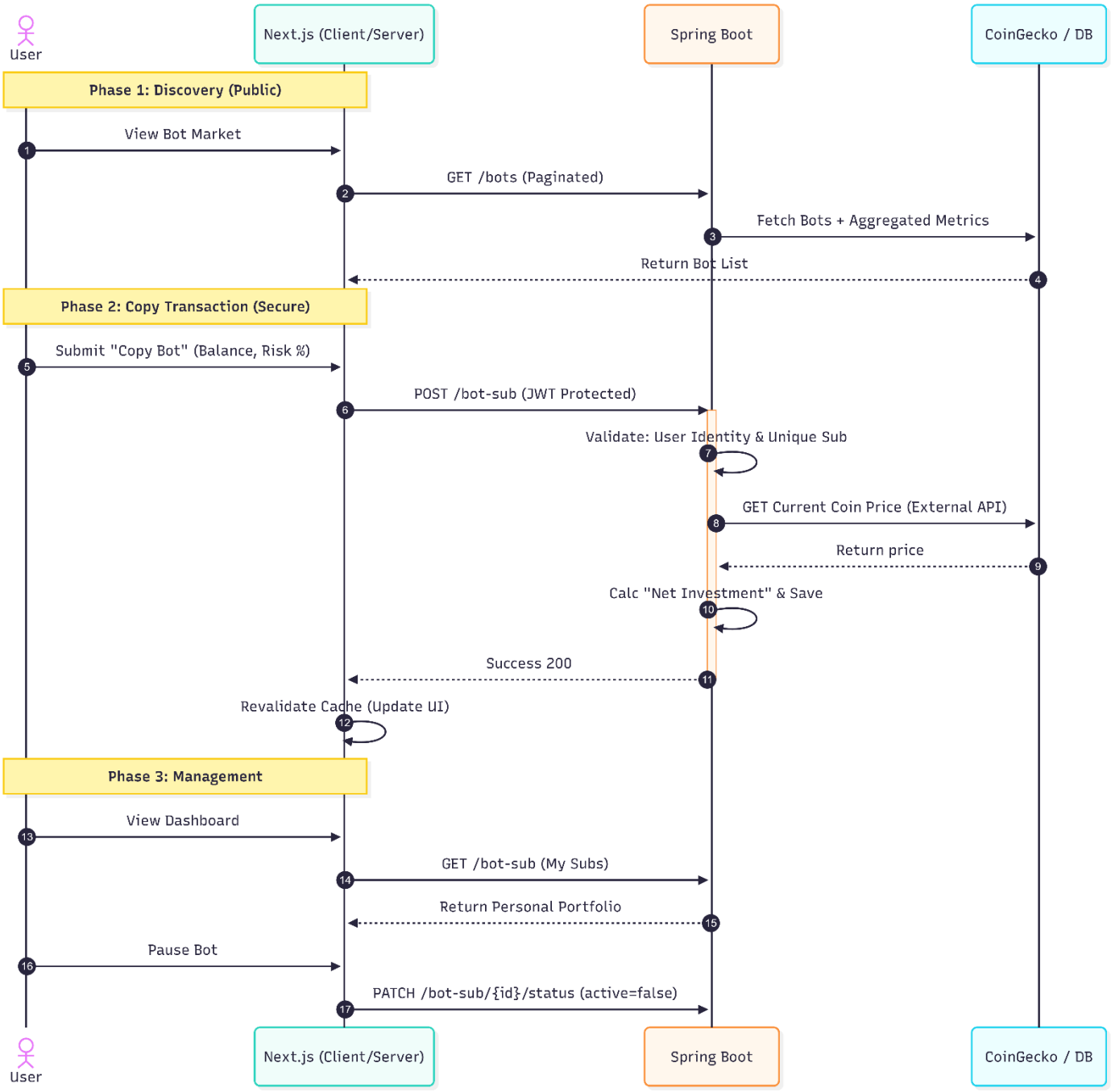

Figure 3.3-15: Copy Bot Sequence diagram

h) Bot Auto-Trading

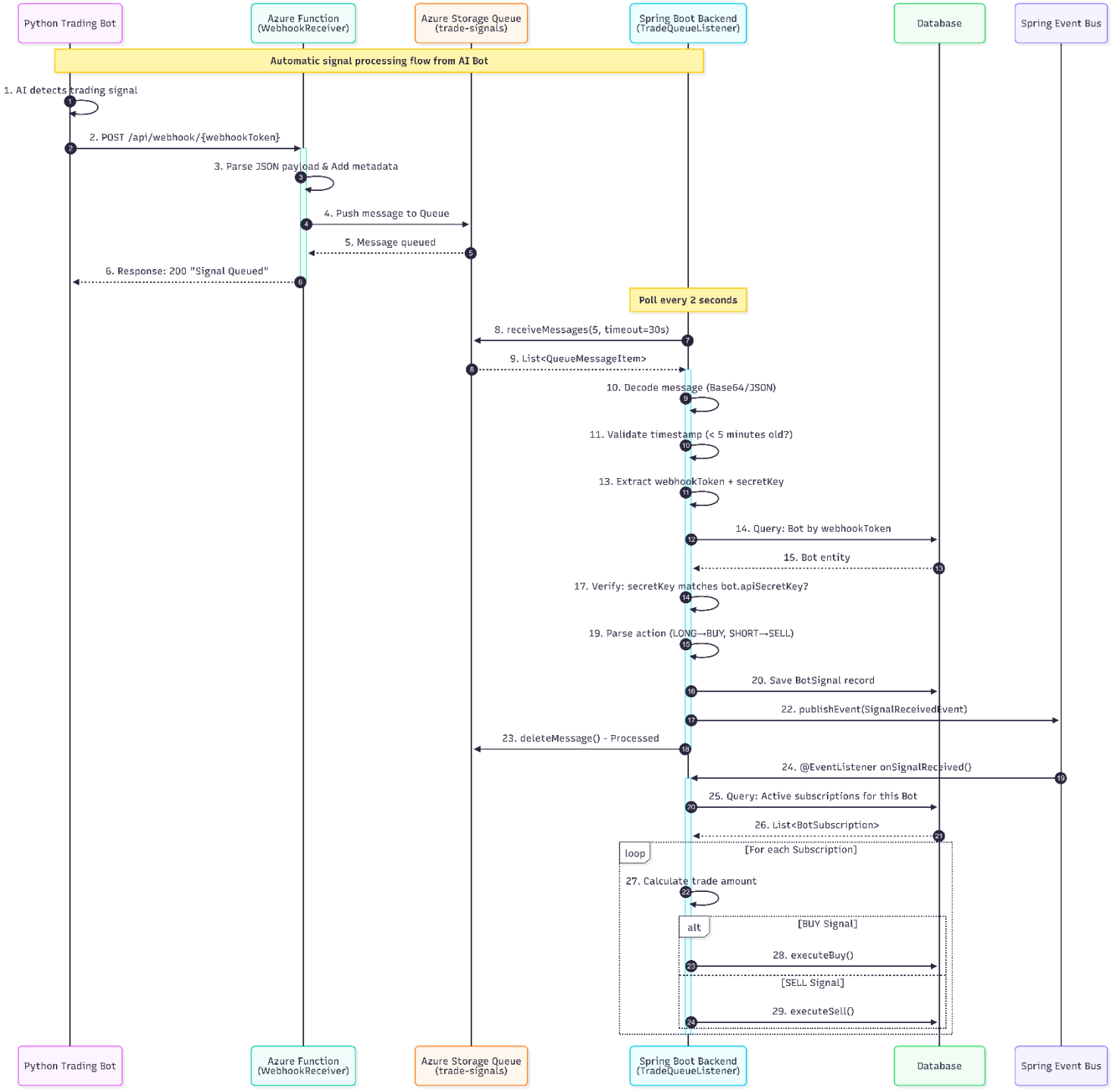

Figure 3.3-16: Bot Trading Sequence diagram

i) Bot Management (Admin)

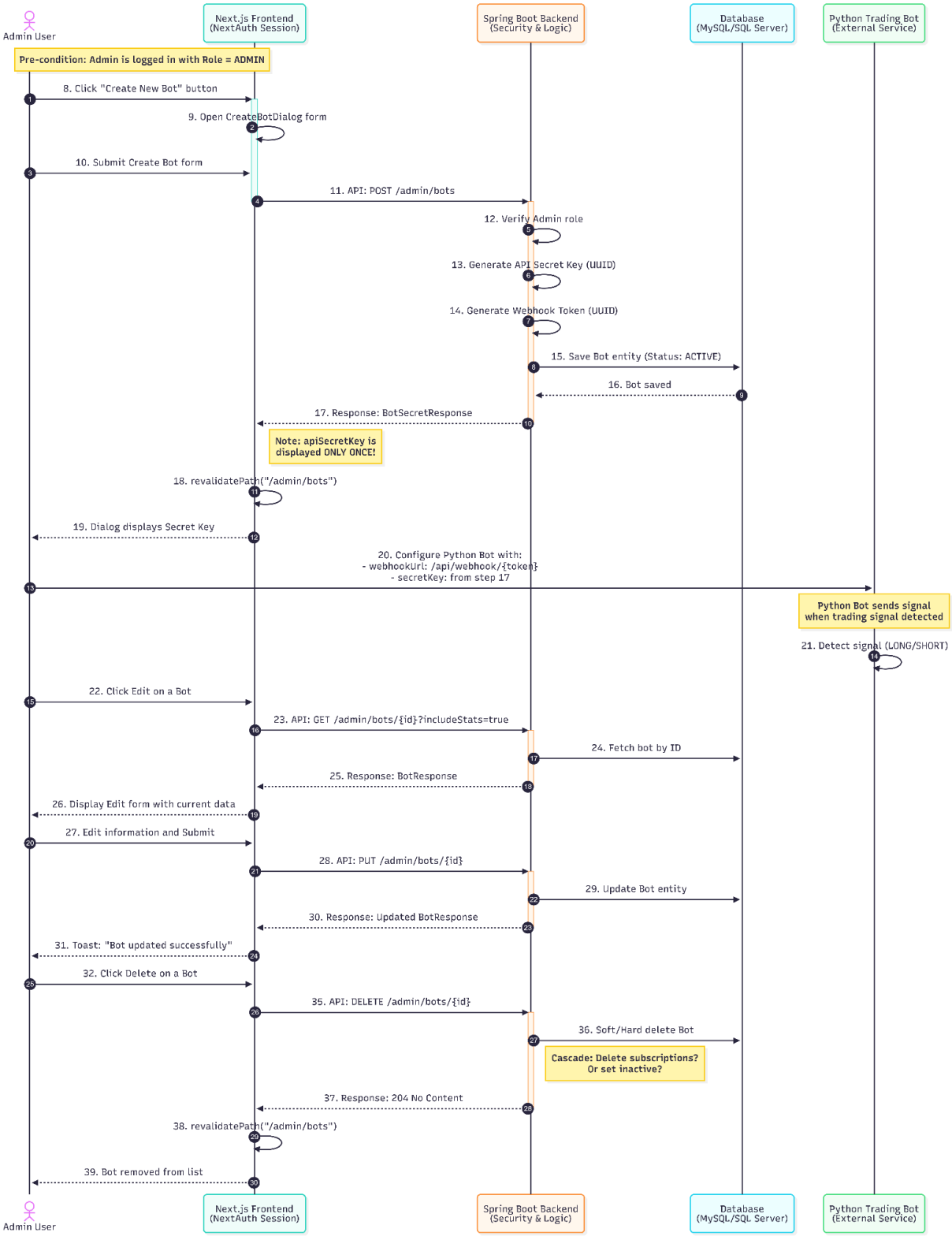

Figure 3.3-17: Admin Bot Management Sequence diagram

j) Deposit Payment (VN Pay)

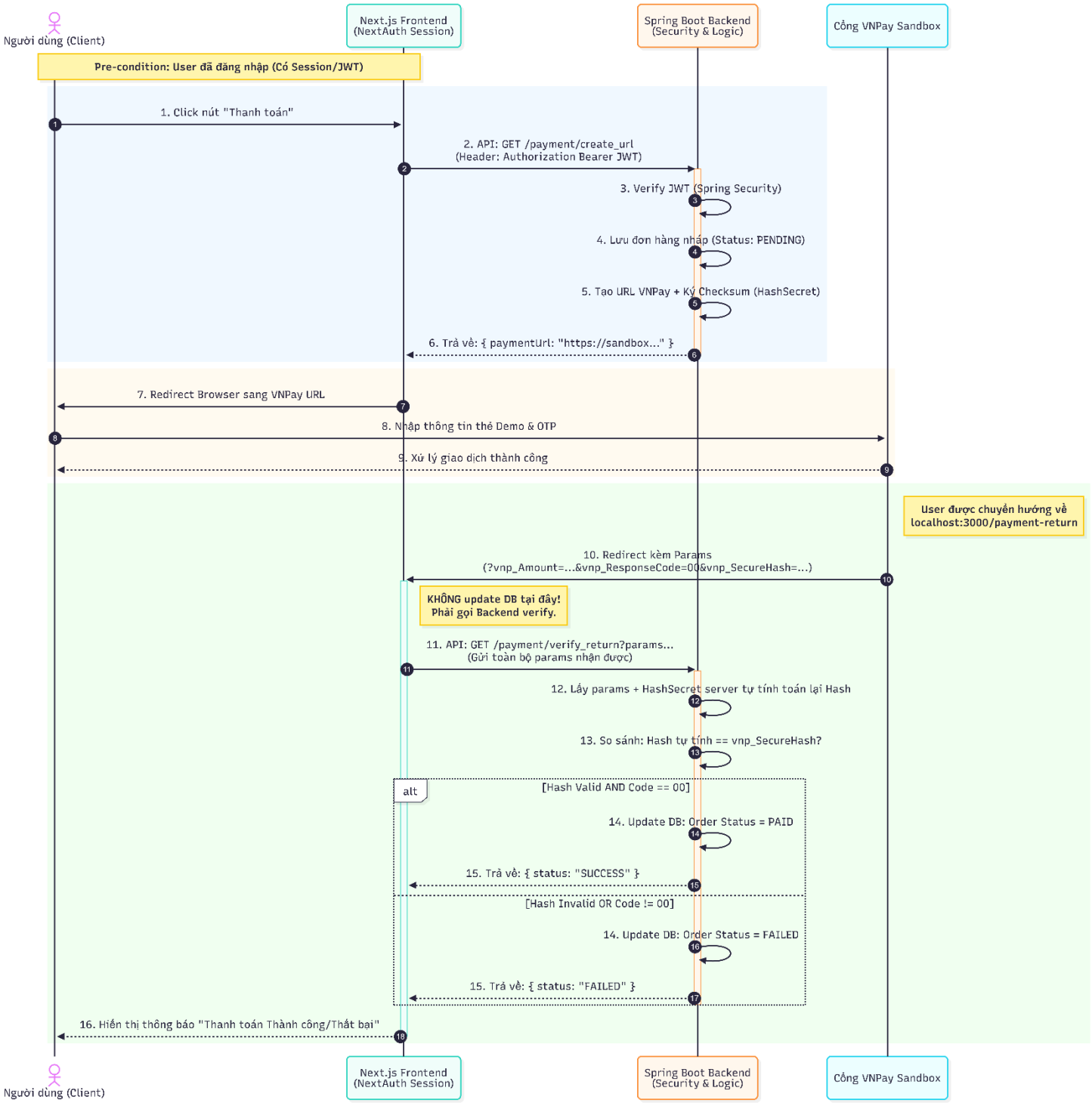

Figure 3.3-18: Deposit Payment Sequence diagram

k) Bot Subscription Detail:

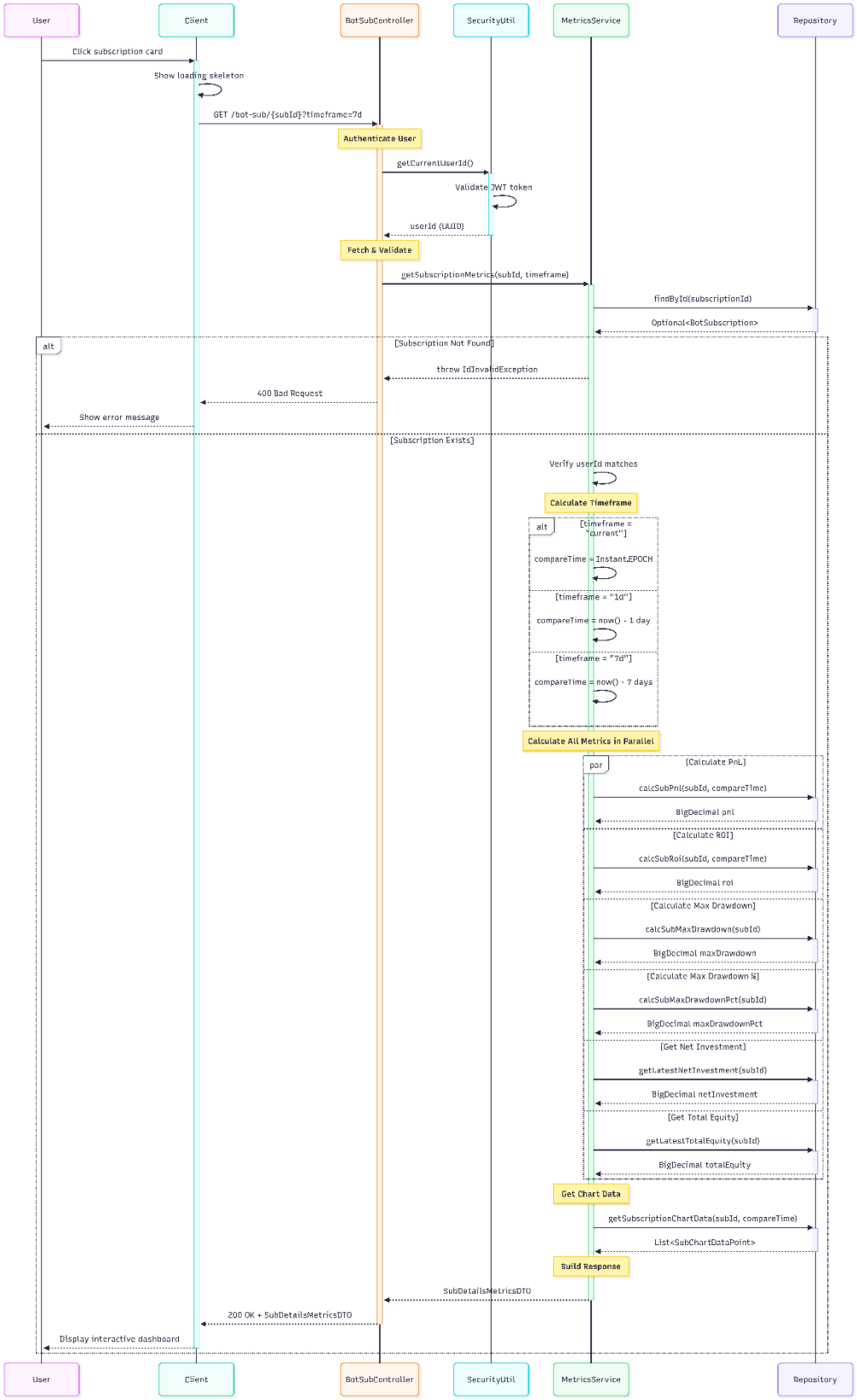

Figure 3.3-19: Bot Subscription Detail Sequence Diagram

CHAPTER 4: DESIGN & ARCHITECTURE

4.1. SYSTEM ARCHITECT

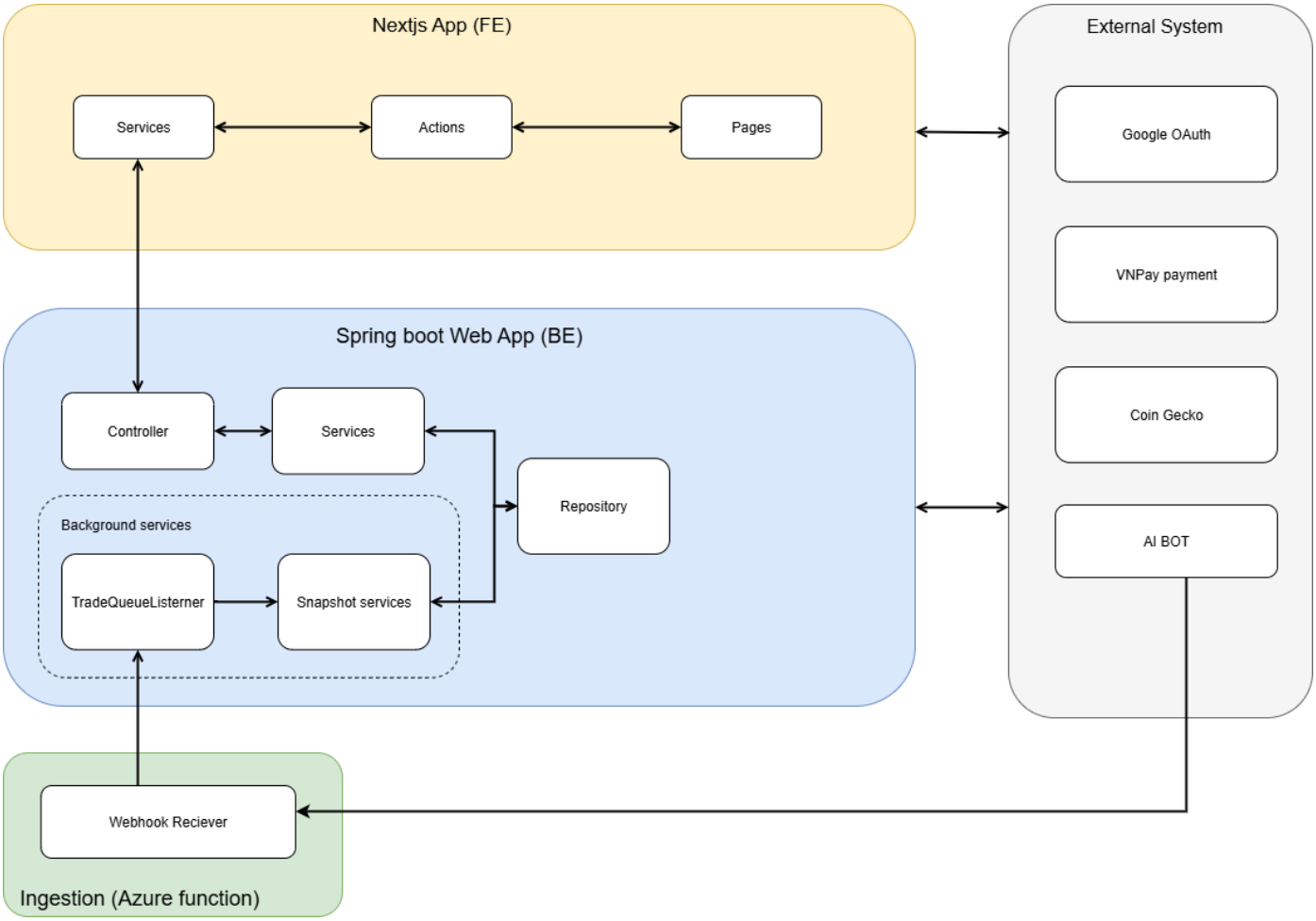

Figure 4.1-1: System Architecture

4.1.1. AI bot Integration

State 1: Bot Creation

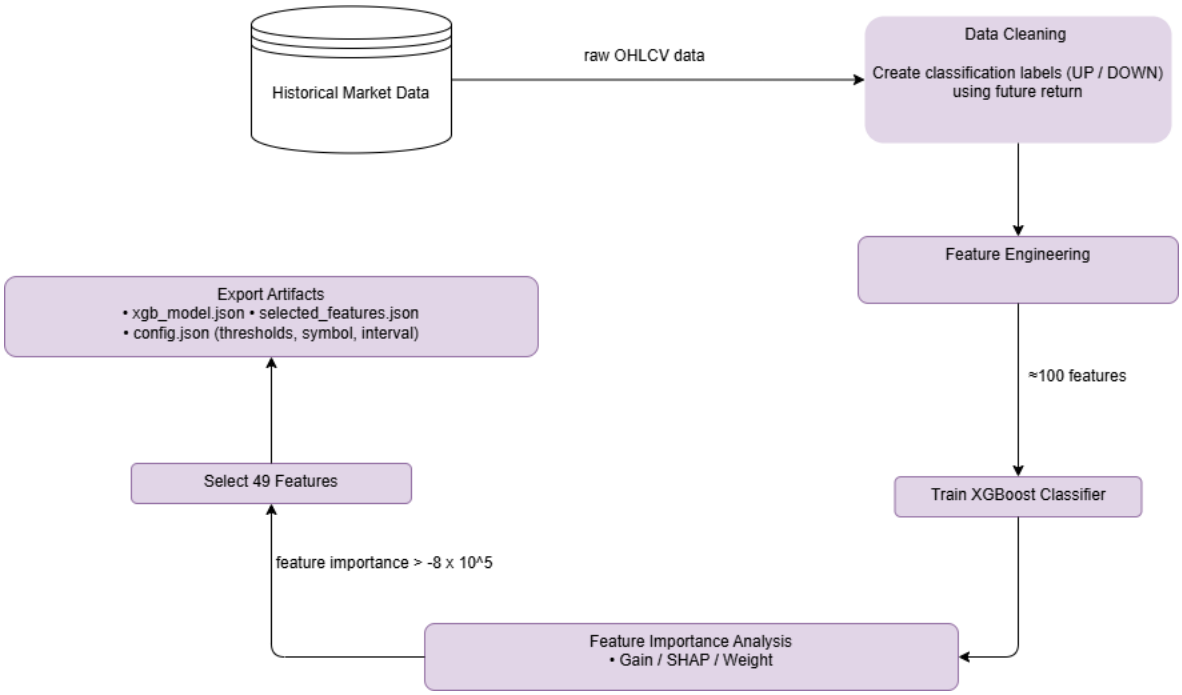

Figure 4.1-2: AI Bot flow

Step 1 – Historical Kline (OHLCV) data are collected from Binance at a 1-minute interval.

Step 2 – The data are cleaned, time-aligned, and labeled based on future price movement (up/down).

Step 3 – Approximately 100 technical features are computed from price, volume, trend, and volatility indicators.

Step 4 – An XGBoost classification model is trained using a time-series split.

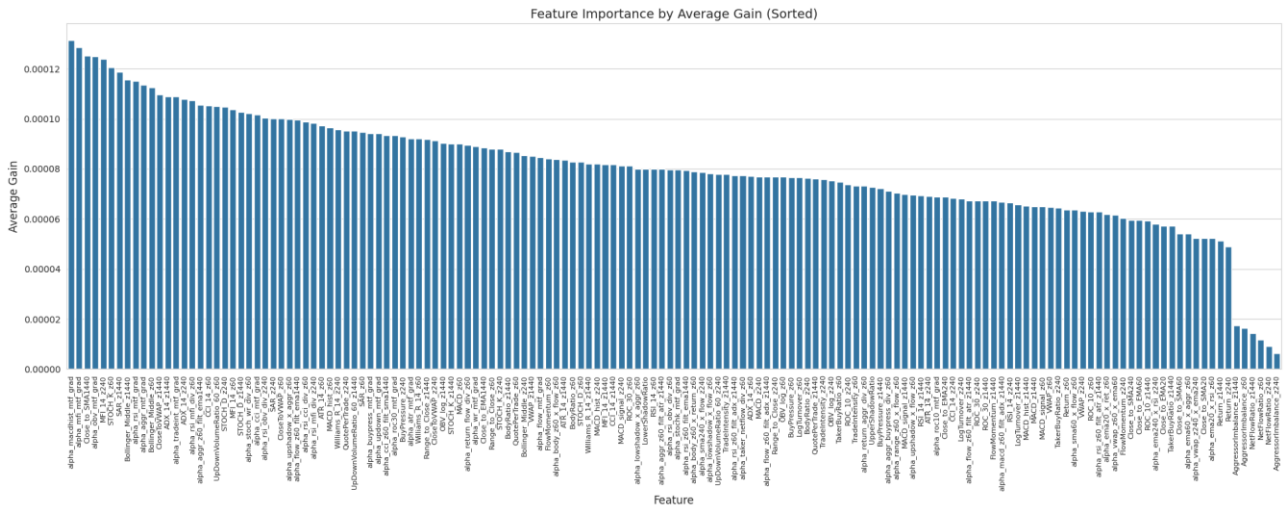

Figure 4.1-3: Feature importances by average gain

Step 5 – Feature importance is analyzed, and the top 49 features are selected, which have feature importance scores higher than 0.00008.

Step 6 – The trained model and configurations are exported (model.json, feature list, runtime config).

State 2: Bot Deployment and Usage

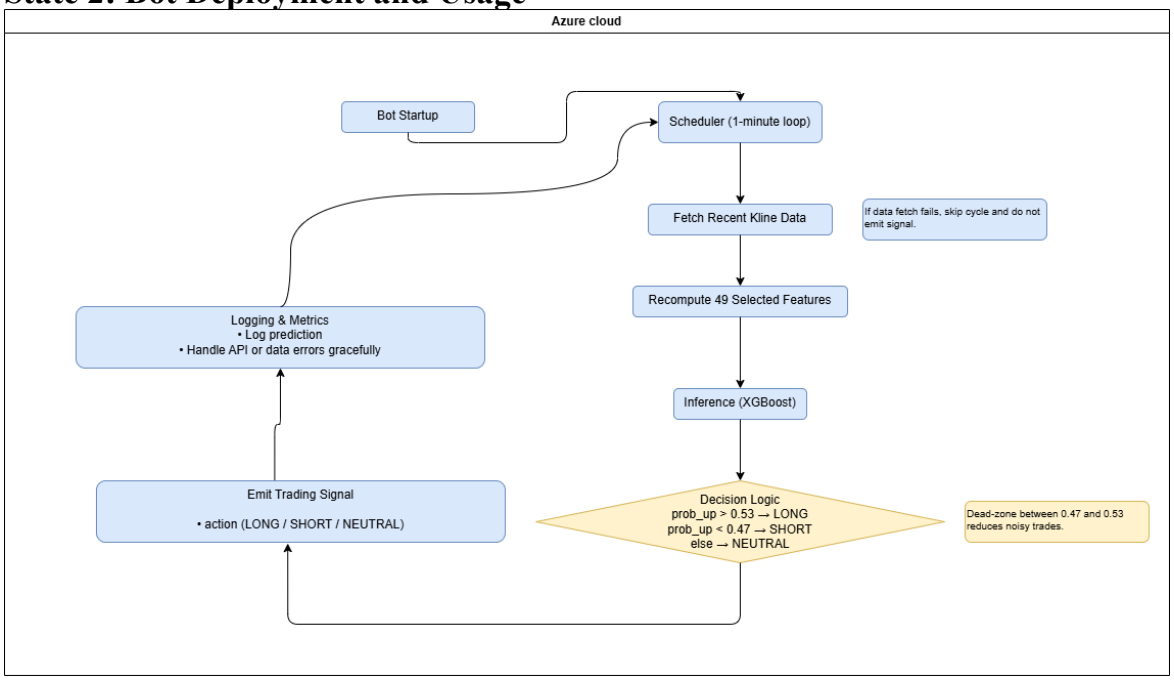

Figure 4.1-4: Ai Bot deployment

Step 7 – The bot source code and model artifacts are stored in a GitHub repository.

Step 8 – The bot is packaged into a container including dependencies and model files.

Step 9 – The container is deployed to Azure Cloud and runs as a persistent service.

Step 10 – Every minute, the bot fetches market data, computes 49 features, performs inference, and generates a trading signal (LONG, SHORT, or NEUTRAL).

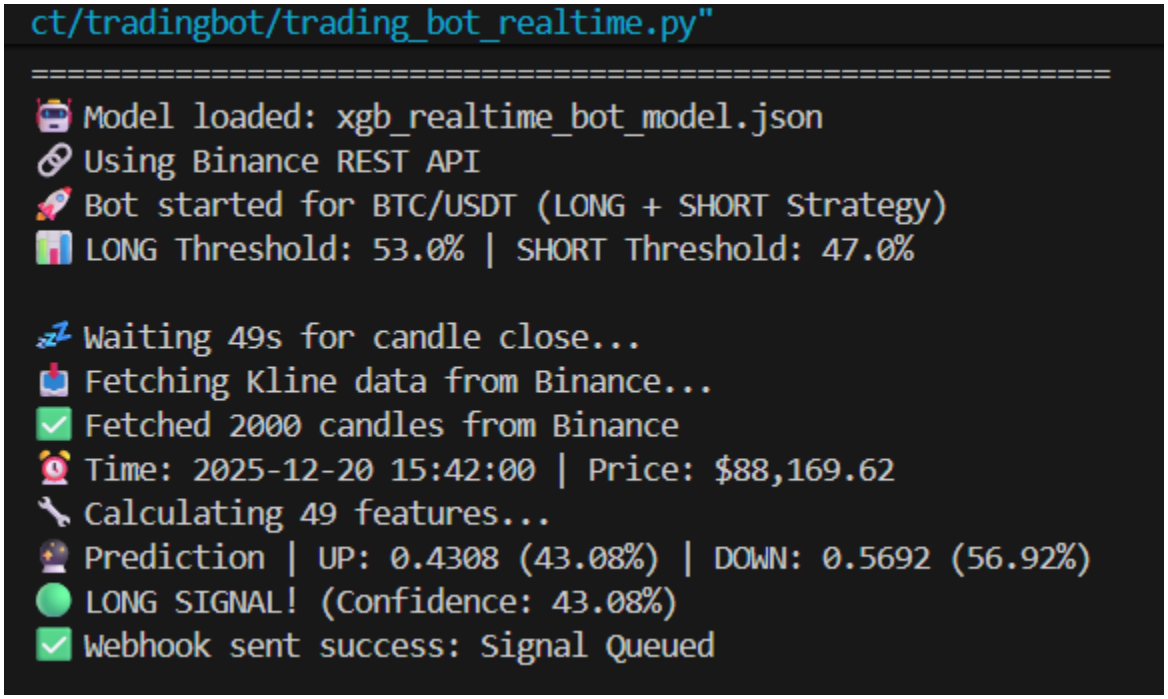

Figure 4.1-5: Ai bot sending signals

4.1.2. AI bot and web application communication

This diagram shows the flow between systems, begins with admin connecting a bot to our system to send bots' messages through webhook.

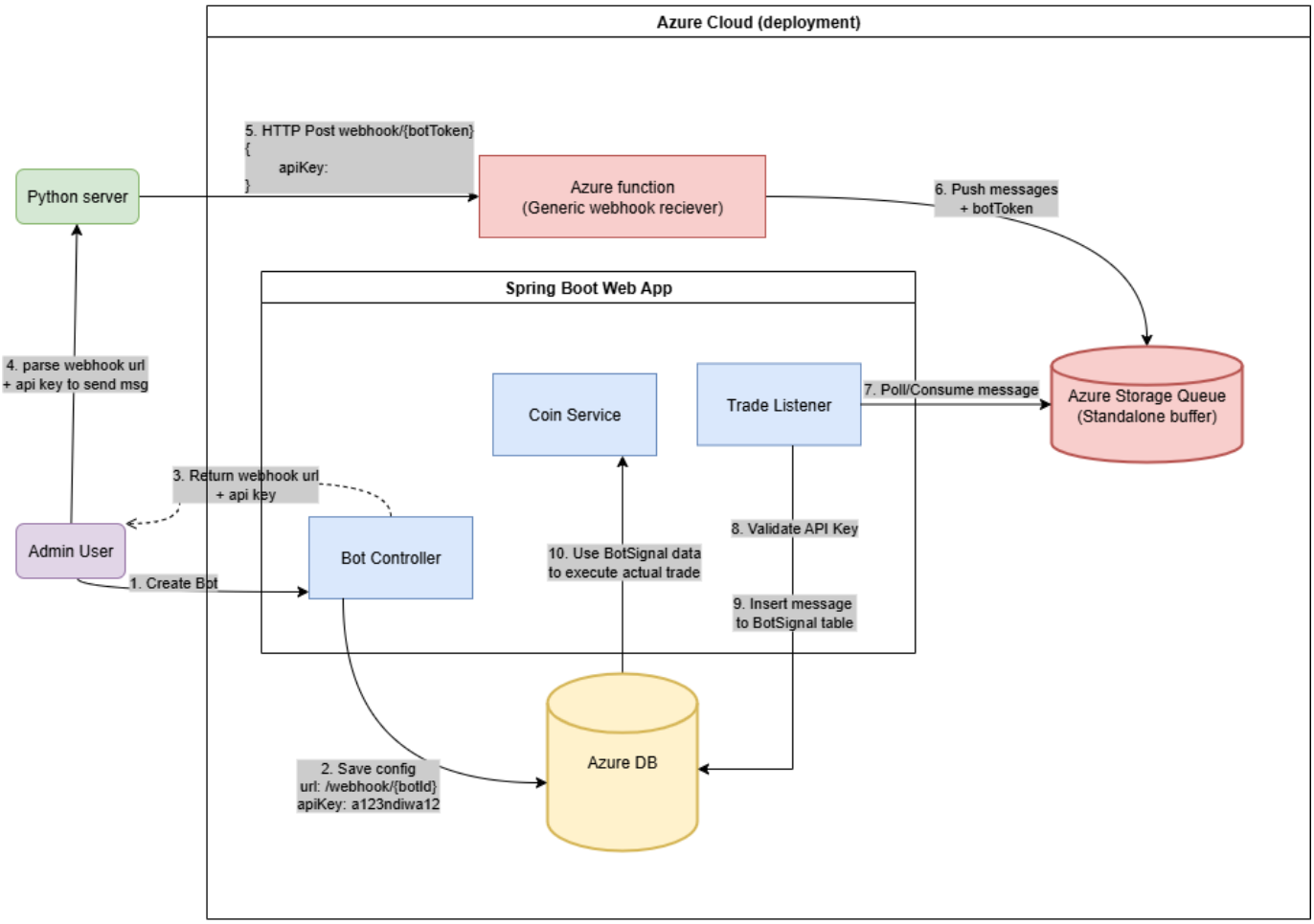

Figure 4.1-6: AI bot integration flow

a) Bot Setup & Configuration (Steps 1-4)

Before trading begins, a secure link is established between the external bot and our system.

- The Admin User creates a new bot using the bot controller.
- The system saves the configuration to the Azure DB and generates a unique Webhook URL and API Key.
- These credentials are provided to the external Python Server, allowing it to "sign" its requests so our system recognizes them as legitimate.

1. Signal Injection & Buffering (Steps 5-6)

We use a decoupled architecture to handle incoming data streams efficiently by applying a queue.

Azure function: When python server detects a trading opportunity, it sends an HTTP POST request to the provided webhook link containing the API Key to our Azure Function.

The Buffer: Instead of processing the trade immediately (which could crash the server if thousands come in at once), the Azure Function pushes the message into an Azure Storage Queue. This acts as a "waiting room," holding messages until the system is ready to process them.

2. Asynchronous Processing & Execution (Steps 7-10)

Our core application processes the queued signals one by one to ensure accuracy and security.

Polling: The Trade Listener (inside the Spring Boot App) constantly monitors the Azure Queue for new messages.

Validation: Before acting, the listener validates the API Key. If the key is incorrect, the signal is discarded to prevent unauthorized trades.

Finally, valid signals are logged into the BotSignal table for audit purposes. Finally, the Coin Service uses this data to execute the trade order in our system using the real price market.

4.2. DATABASE DESIGN

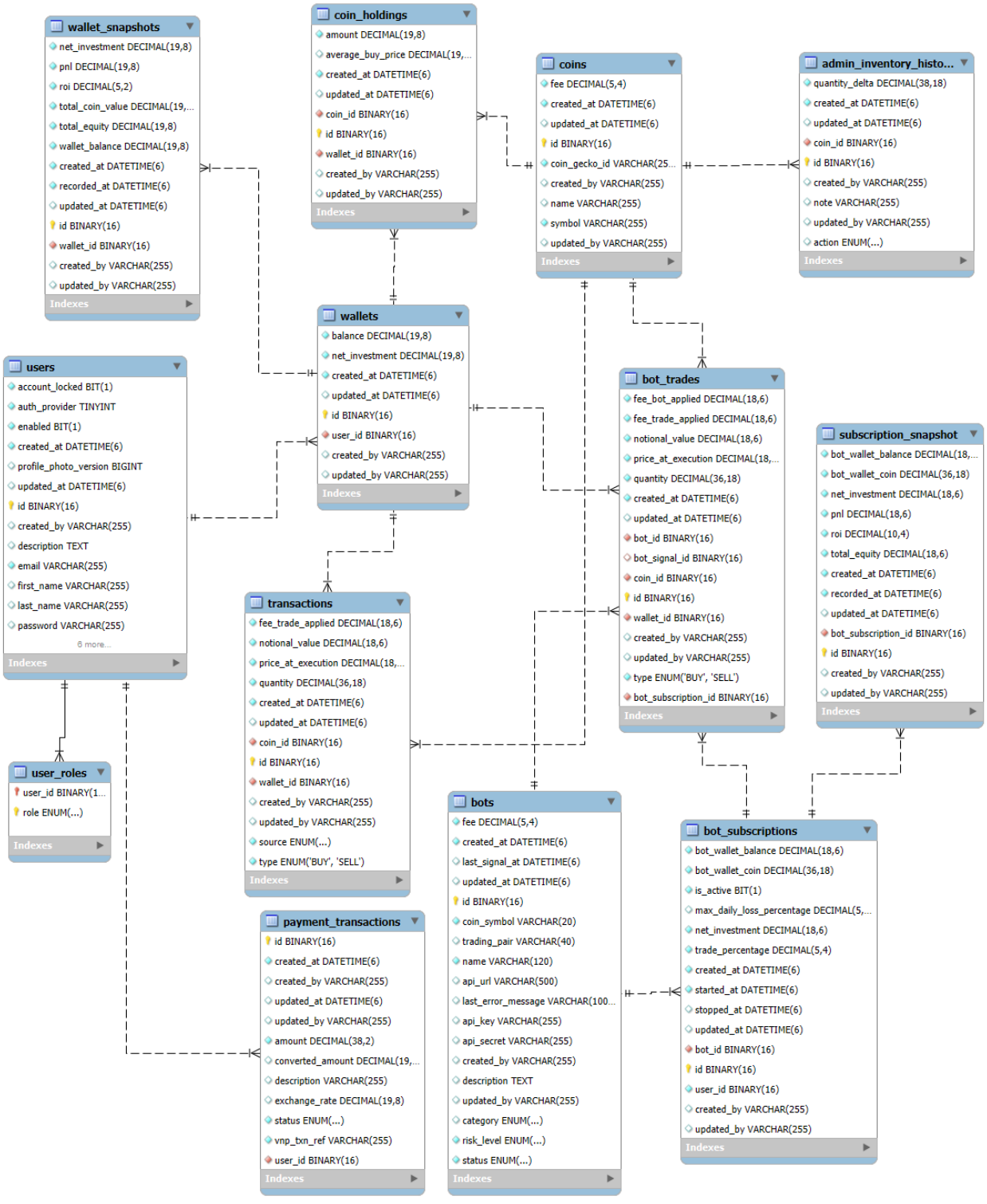

Figure 4-7: Database diagram

4.3. UI/UX DESIGN

3. Trader pages

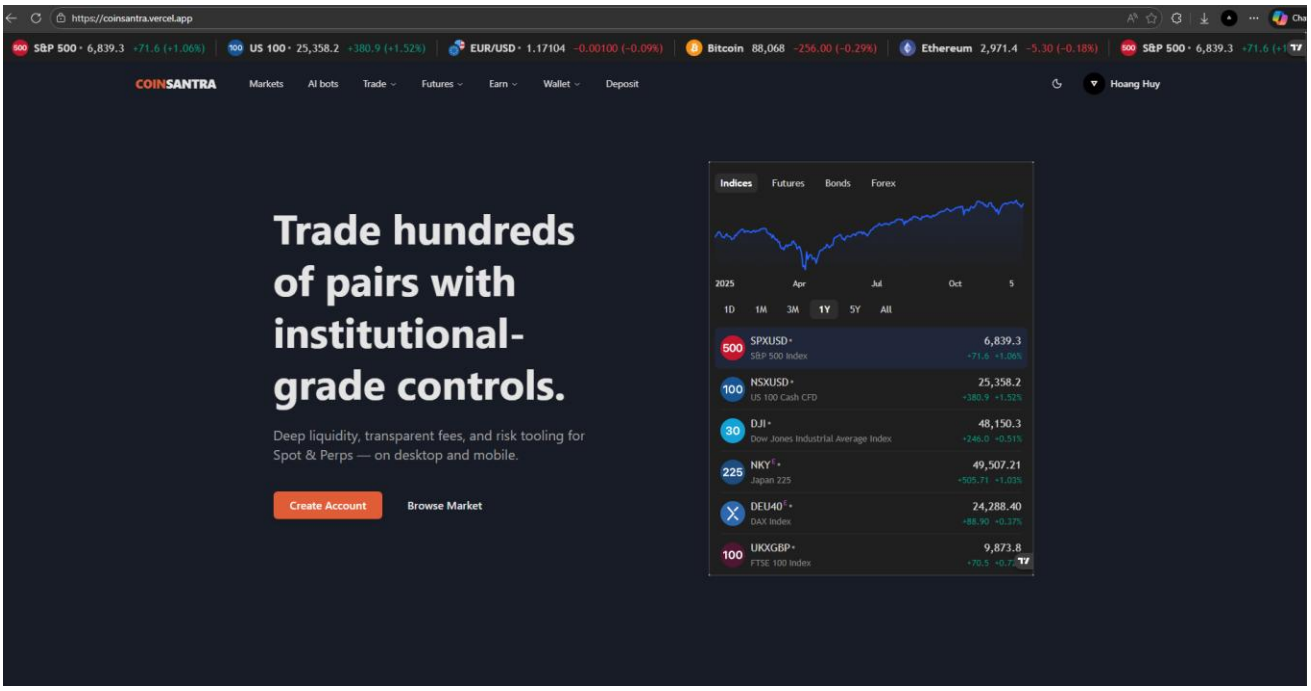

Figure 4.3-1: Home page (dark mode)

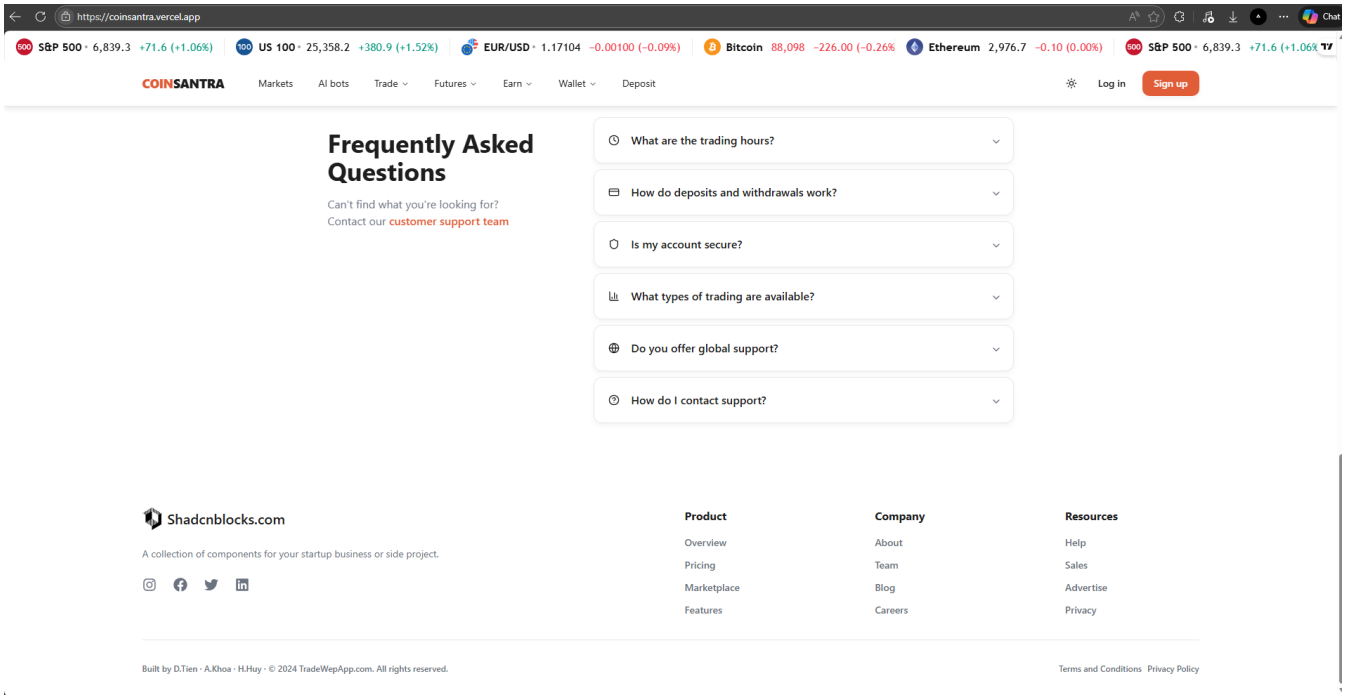

Figure 4.3-2: Home page (light mode)

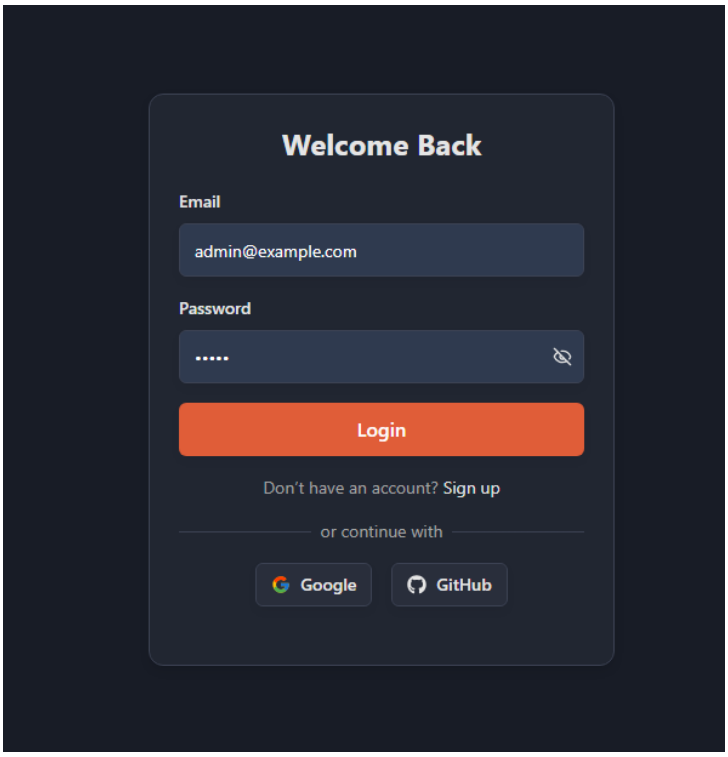

Figure 4.3-3: Login page

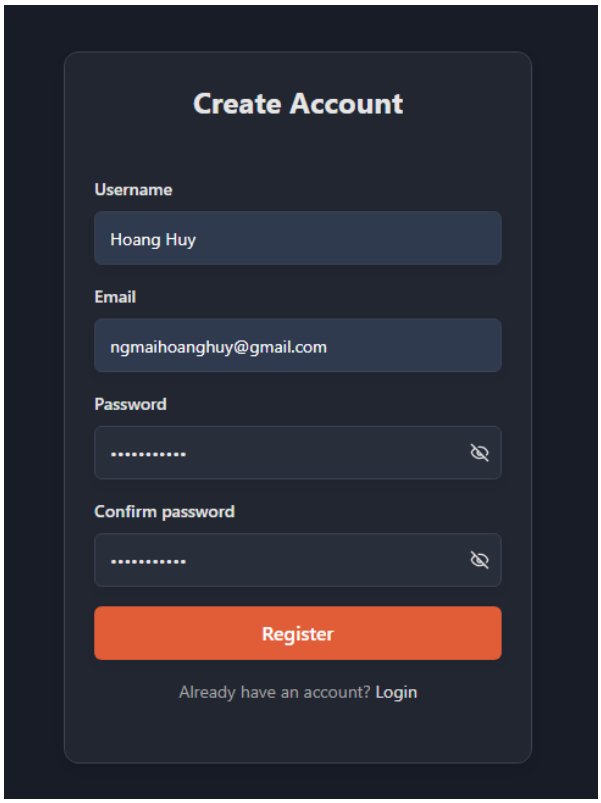

Figure 4.3-4: Register Page

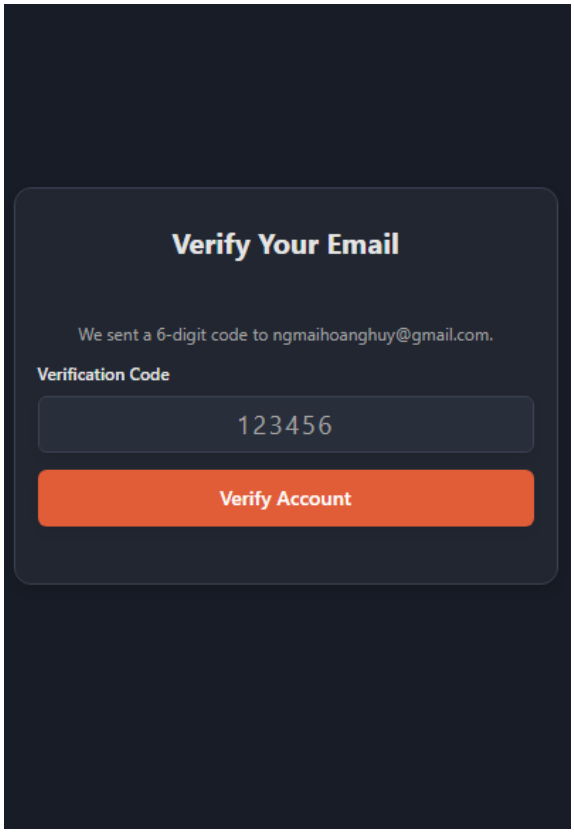

Figure 4.3-5: Verify email pages (only after registering)

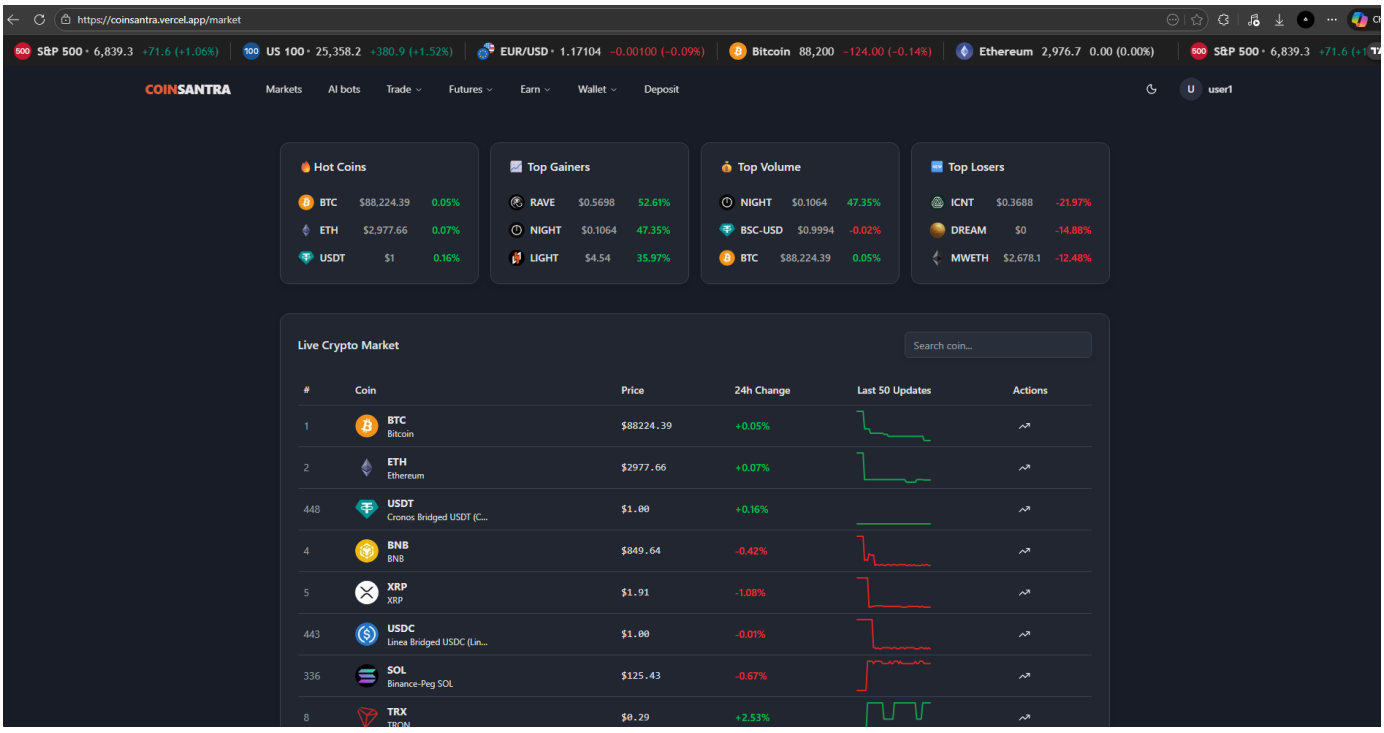

Figure 4.3-6: Market Page (stream using Binance WebSocket)

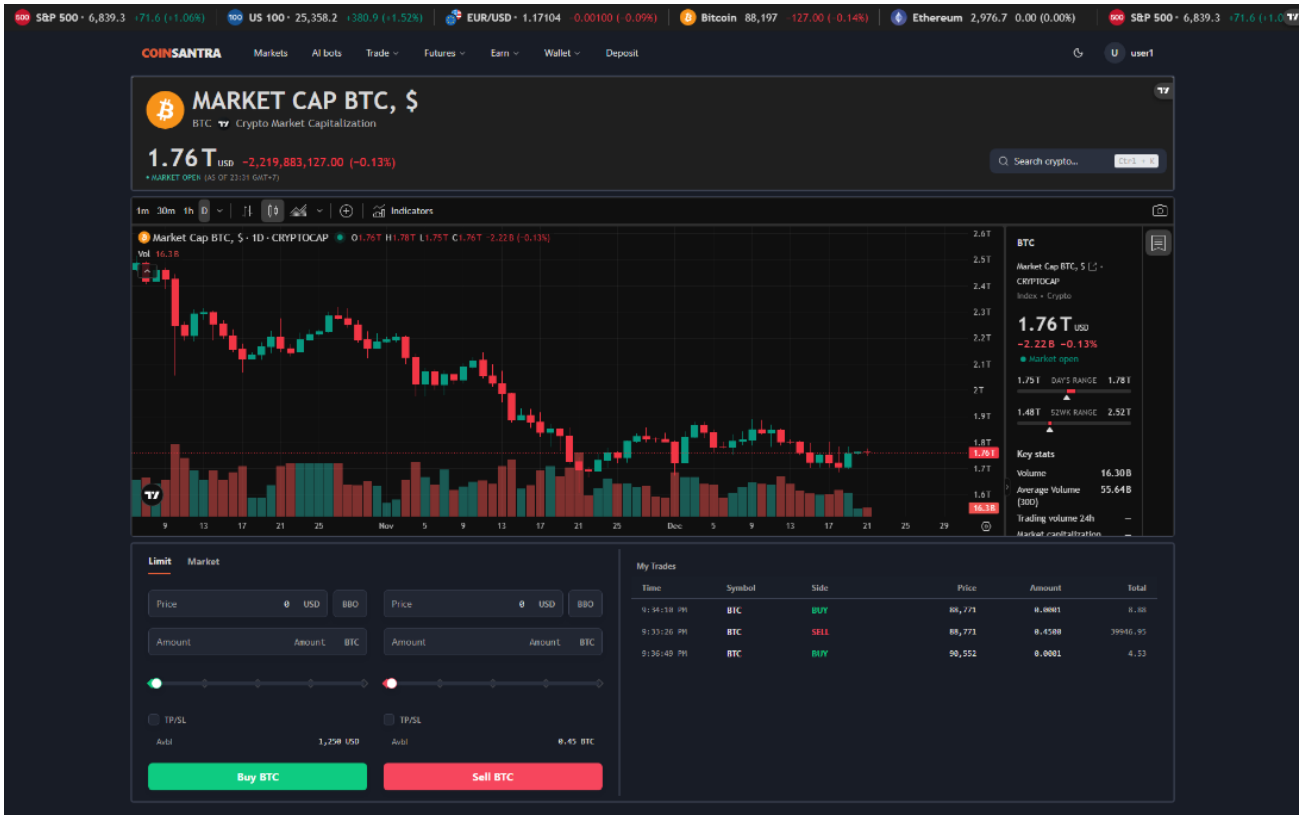

Figure 4.3-7: Trade page

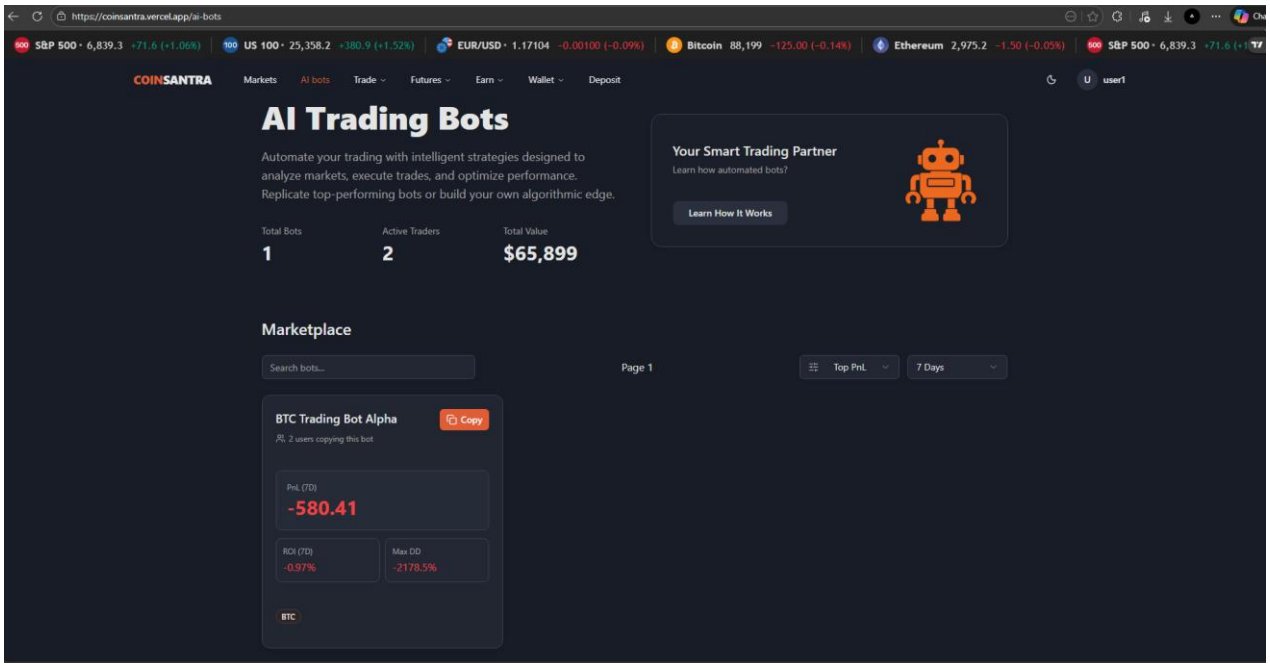

Figure 4.3-8: Ai bots pages (public url)

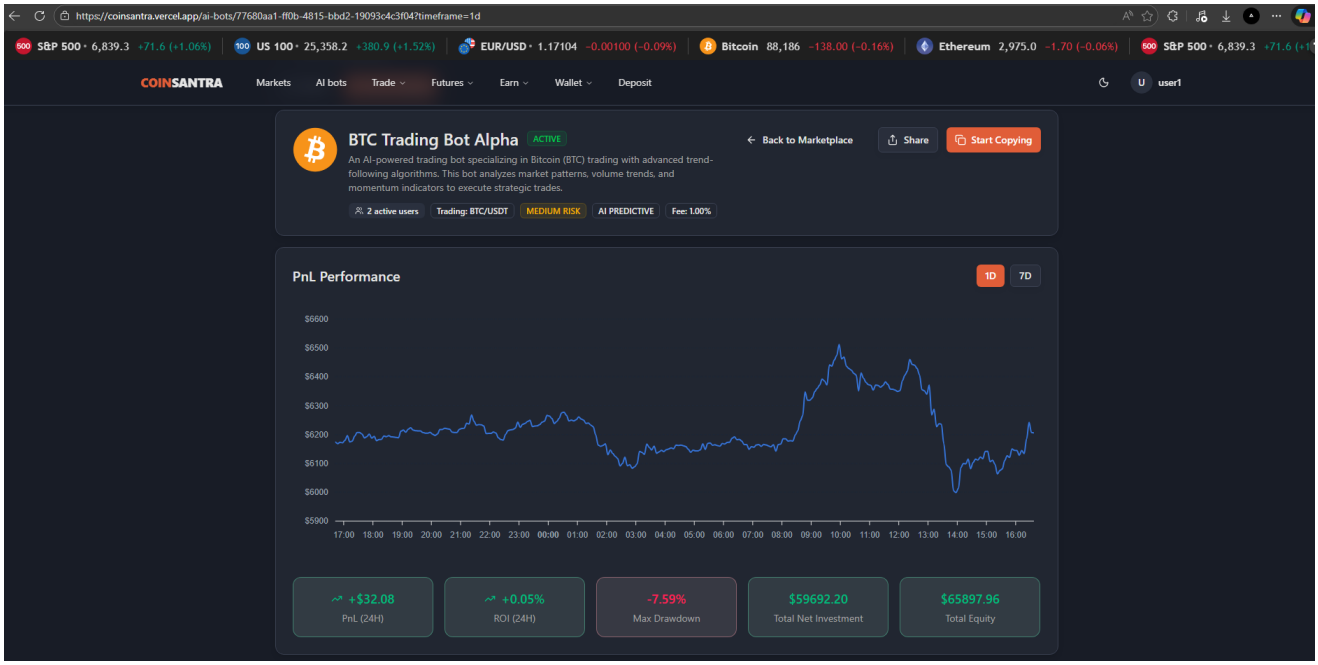

Figure 4.3-9: Bot details page

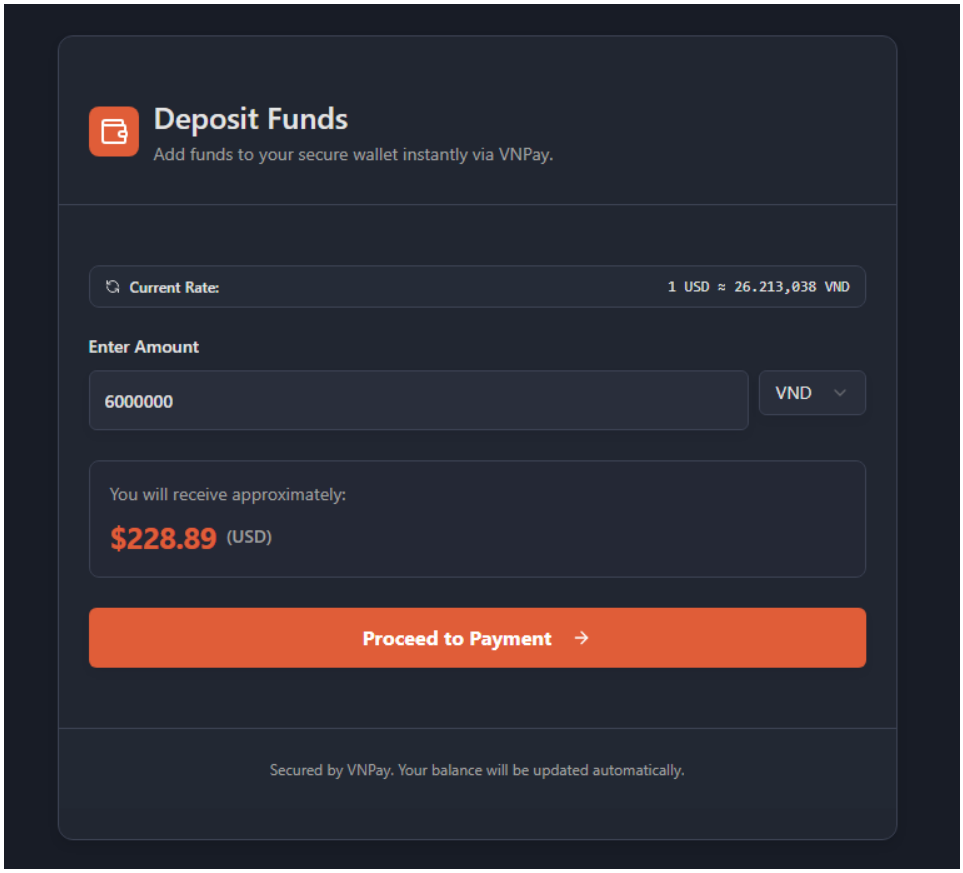

Figure 4.3-10: Deposit page (direct to vnpay)

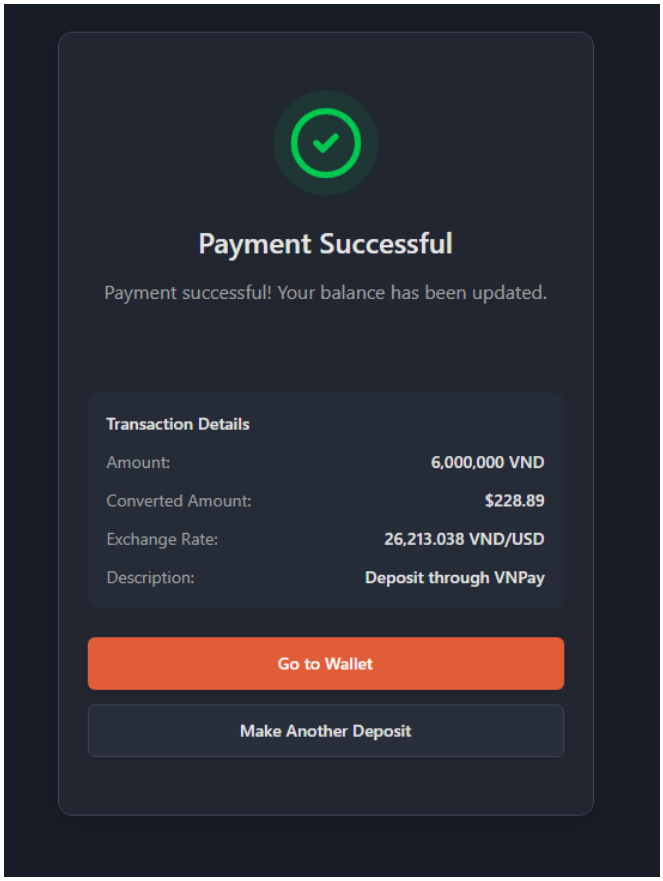

Figure 4.3-11: payment return page

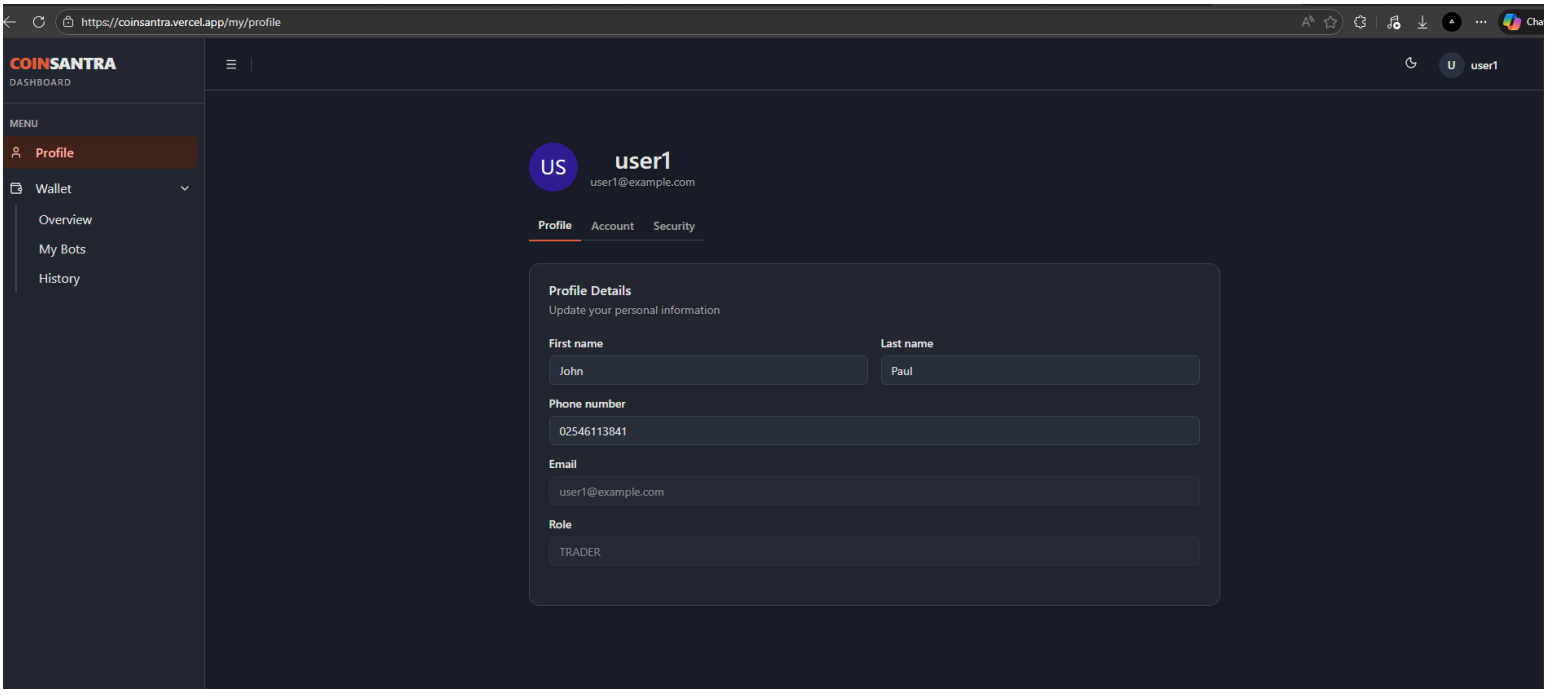

Figure 4.3-12: Profile page

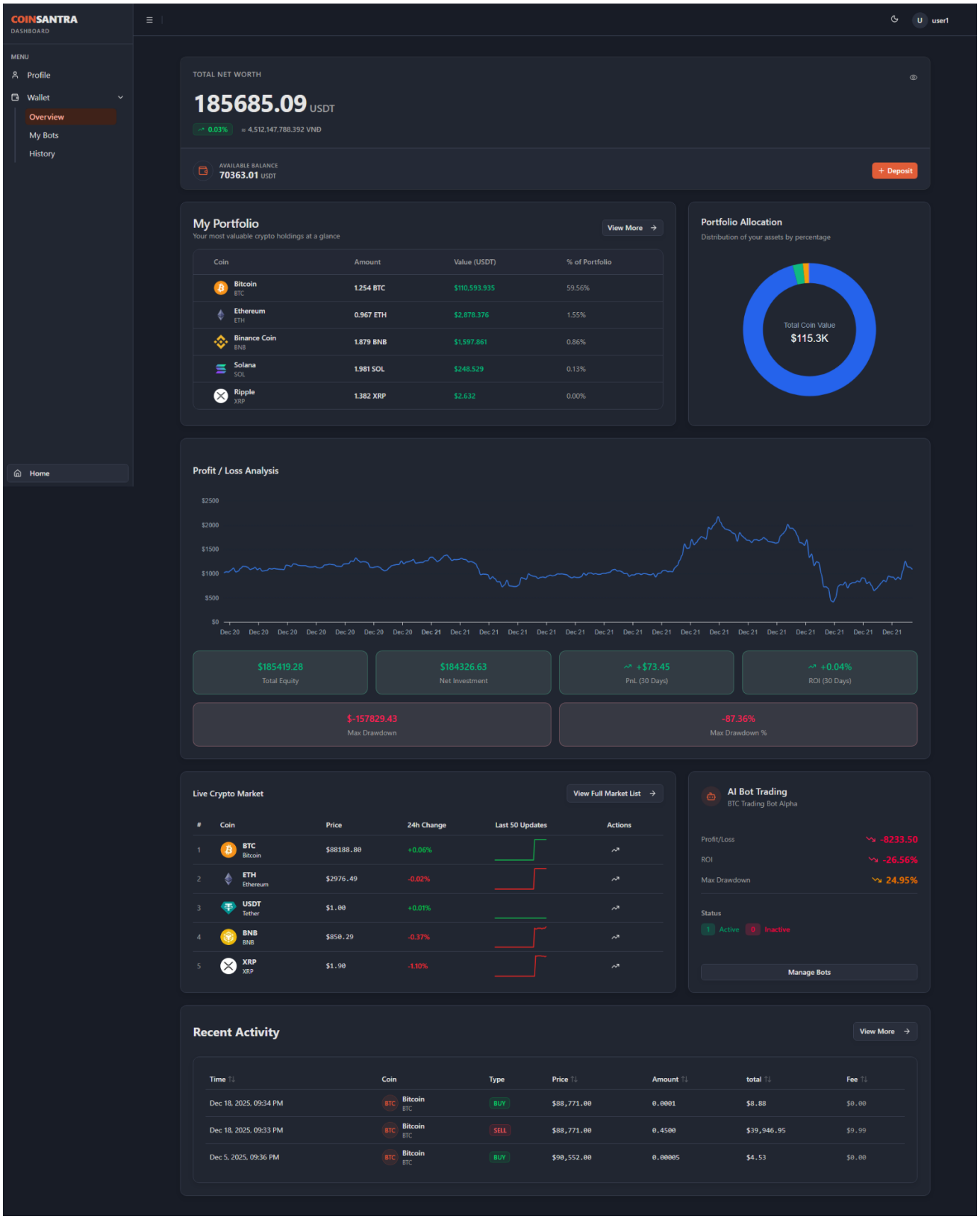

Figure 4.3-13: User's Portfolio Overview page

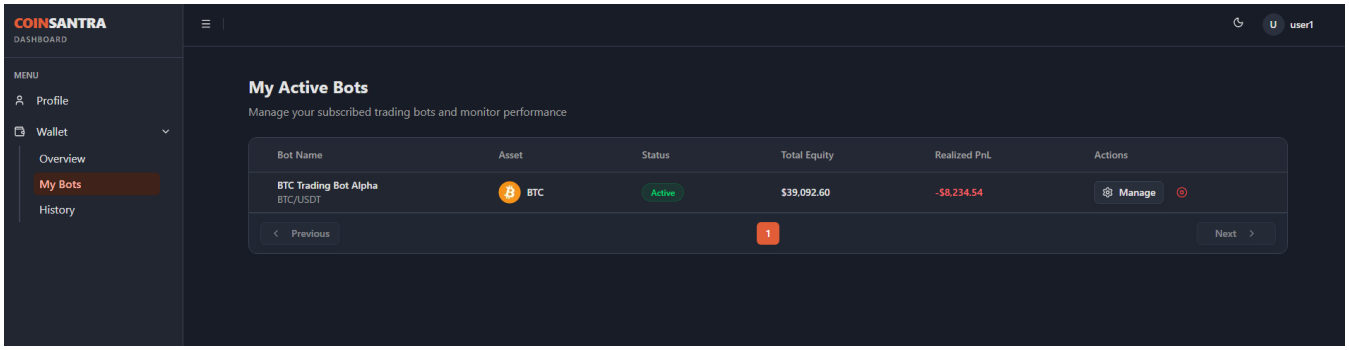

Figure 4.3-14: User's Bot Subscriptions

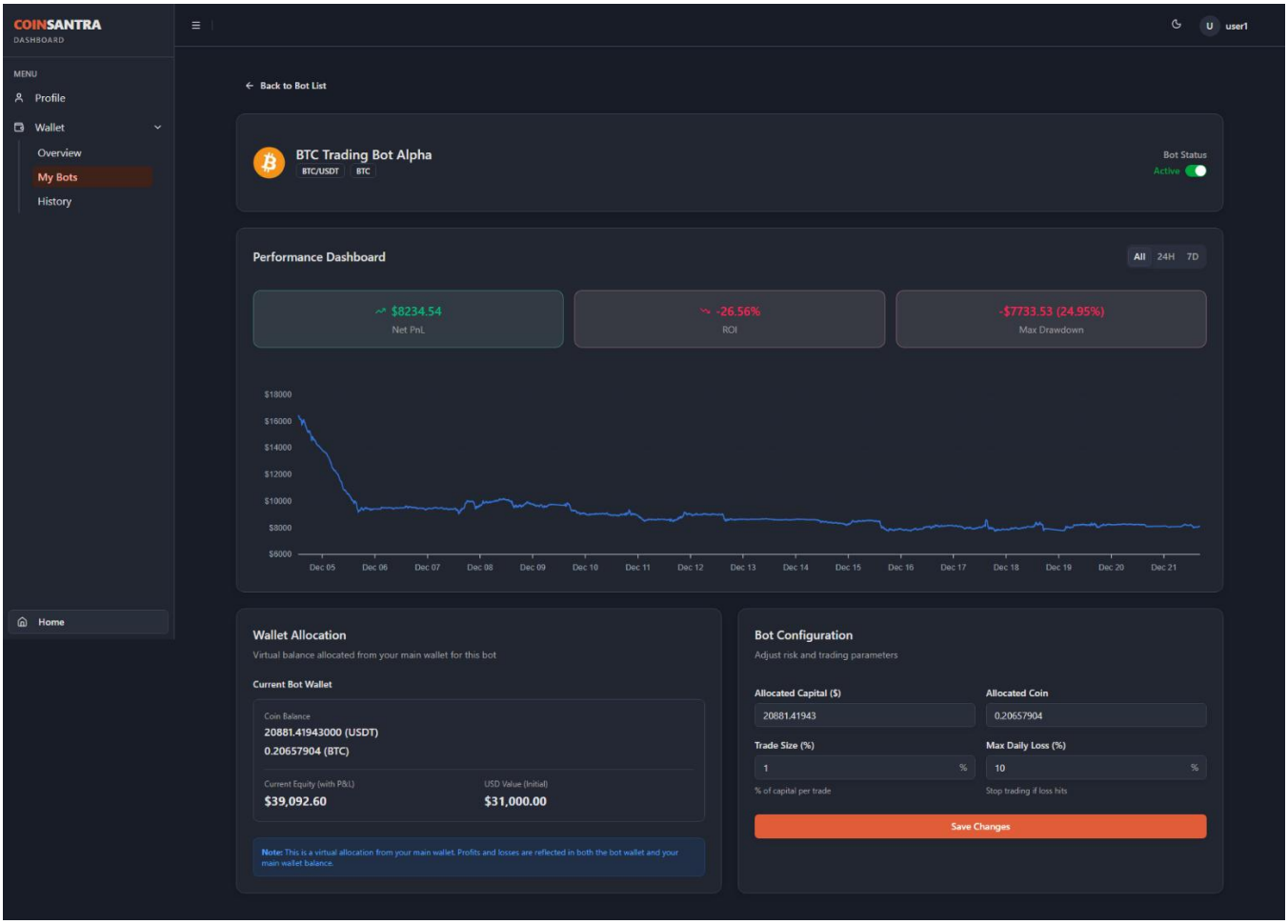

Figure 4.3-15: User's Bot Subscription Details

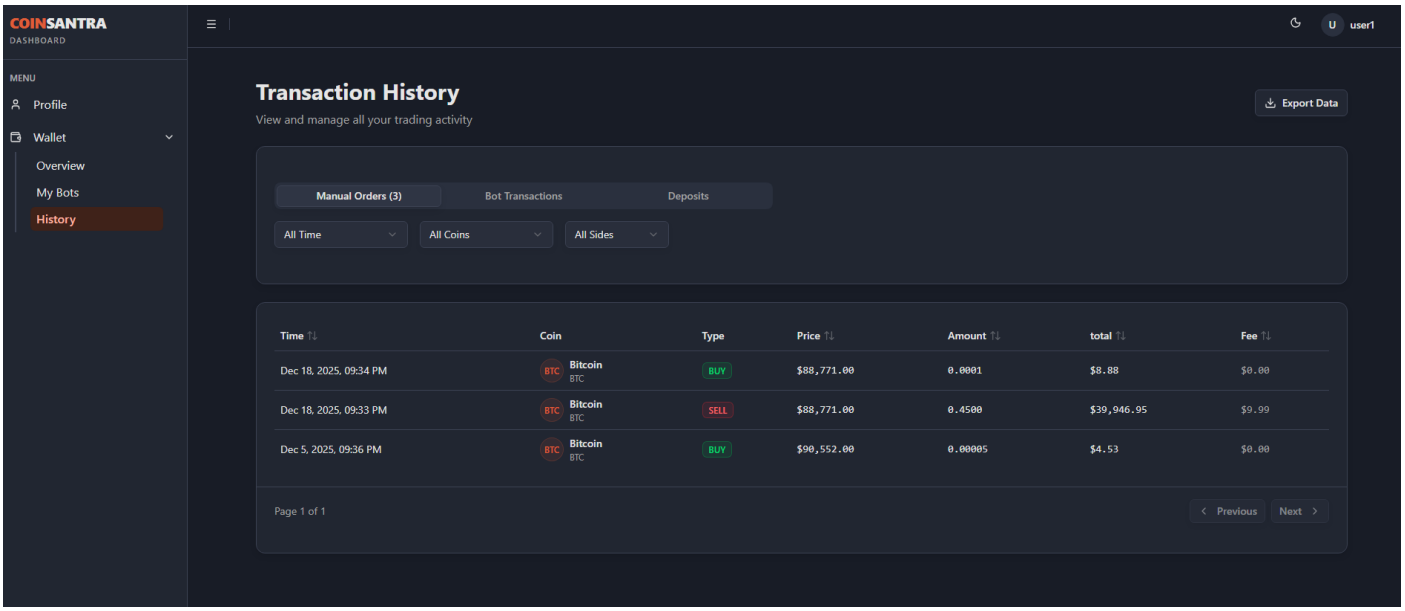

Figure 4.3-16:Manual Trading History Page

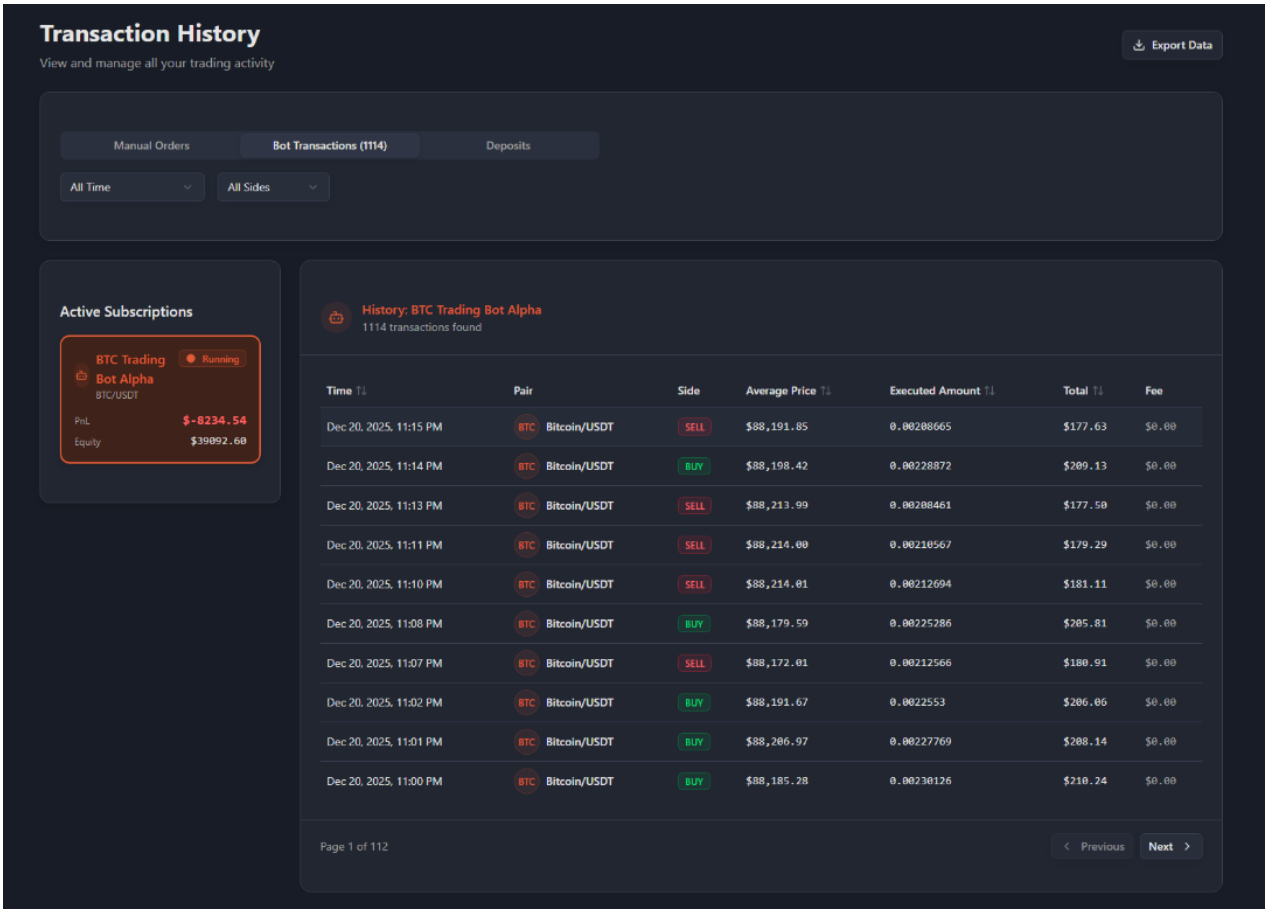

Figure 4.3-17: Bot Trading History Page

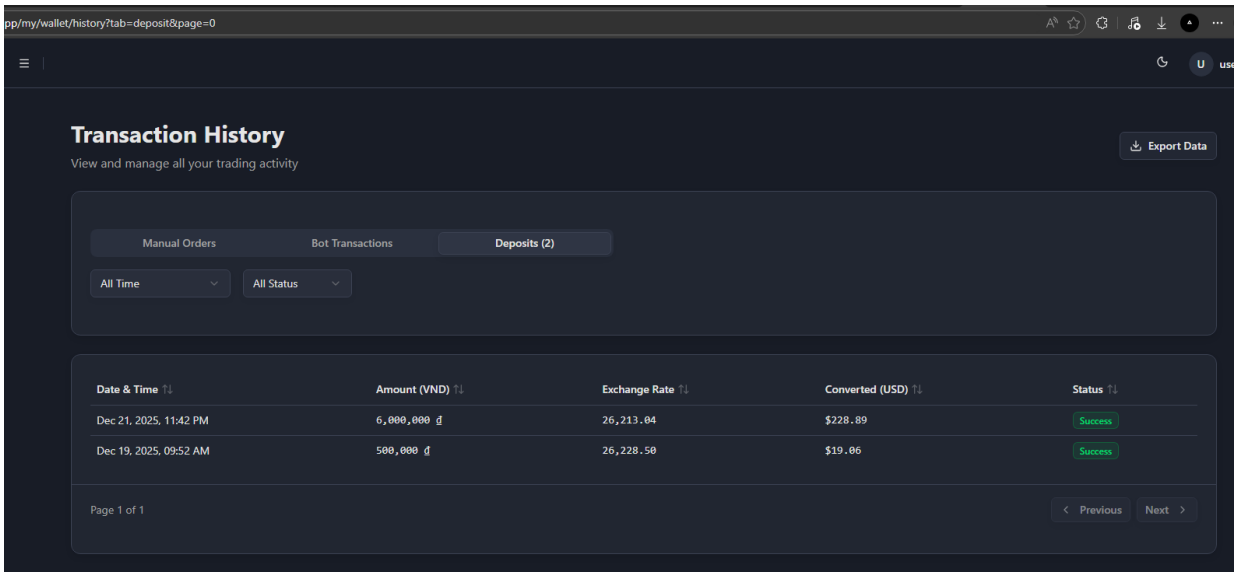

Figure 4.3-18: Deposit History Page

4. Admin pages

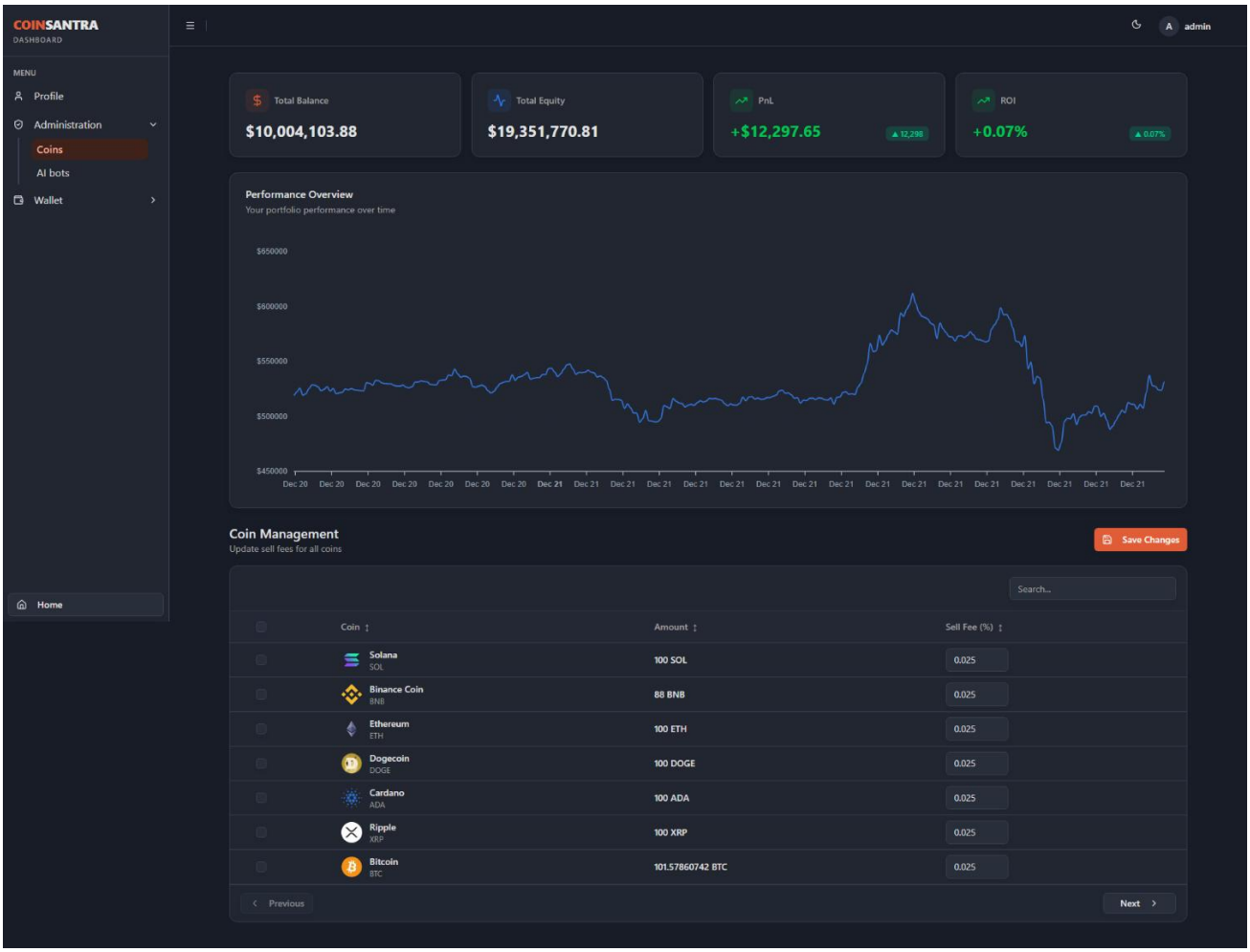

Figure 4.3-19: Admin Portfolio & Coin Management Page

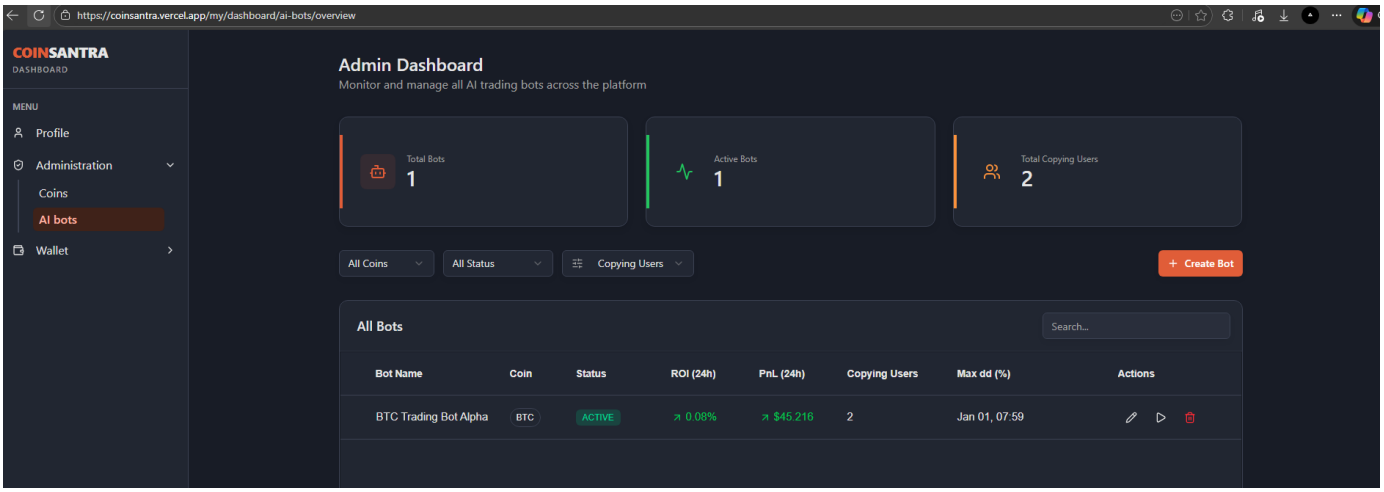

Figure 4.3-20: Admin's AI Bots management Page

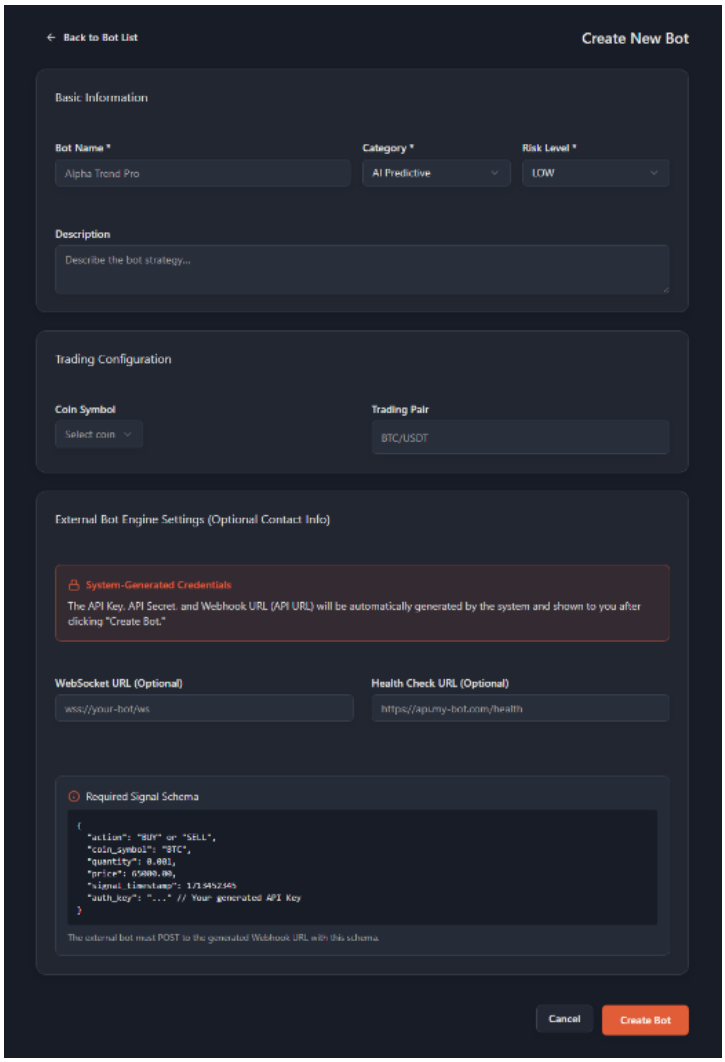

Figure 4.3-21: Create bot page

**Note: note that the external bot engine settings doesn't actually work, it just there for visual purpose.

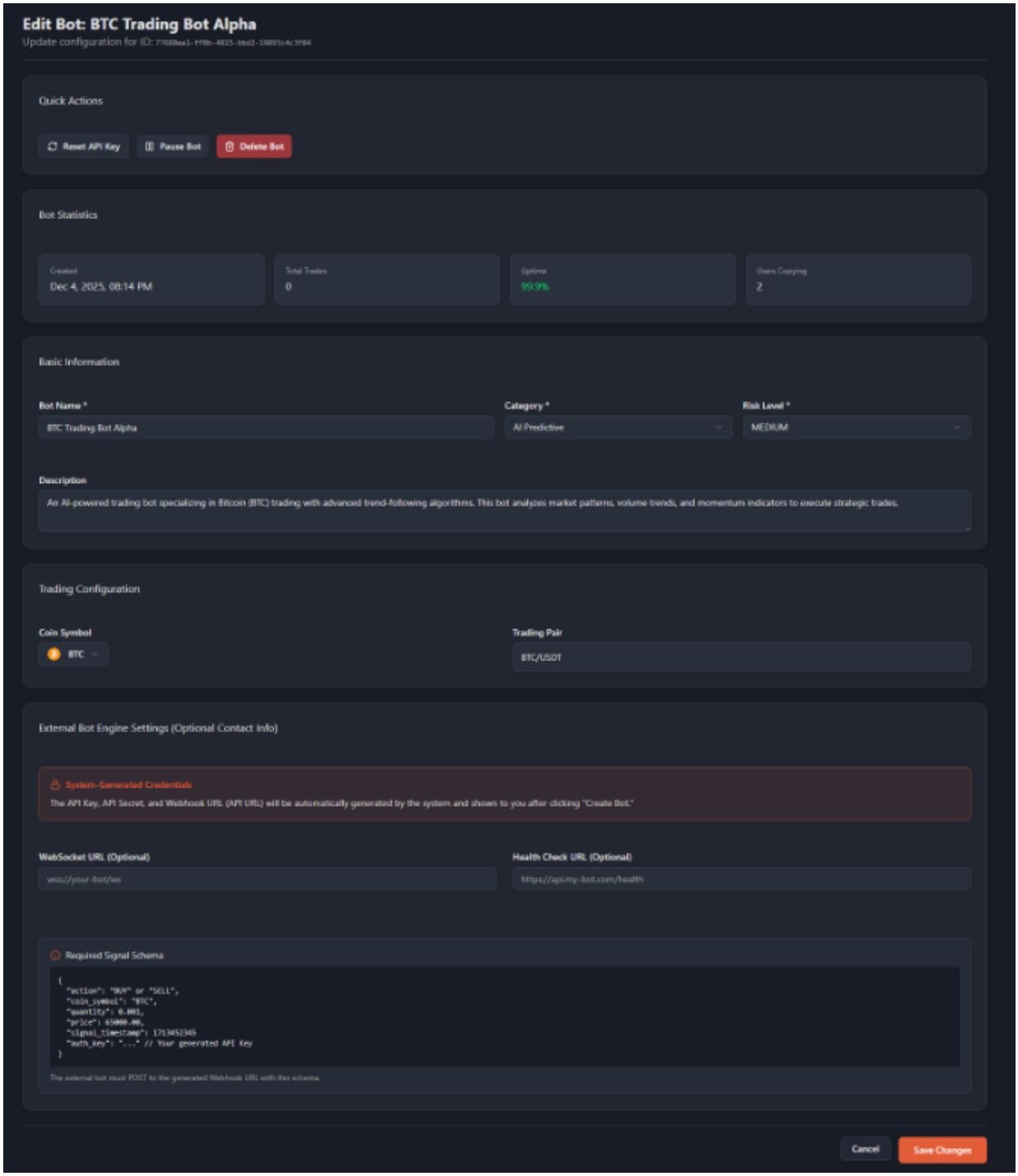

Figure 4.3-22: Edit Bot Page

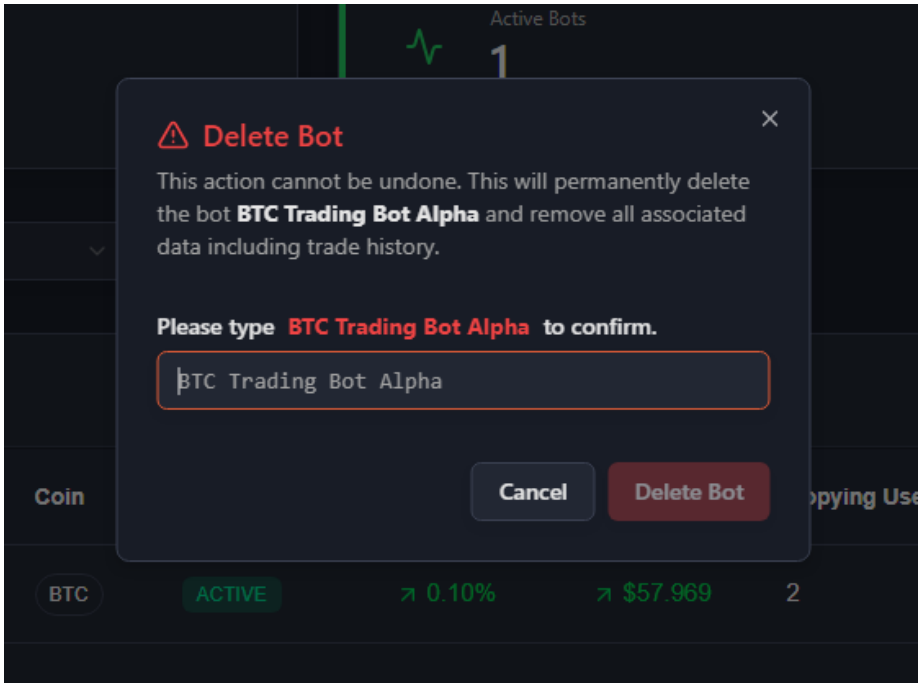

Figure 4.3-23: Delete Bot Modal

CHAPTER 5: IMPLEMENTATION

5.1. PROGRAMMING LANGUAGES

Two primary languages to ensure performance and reliability across the platform:

- **TypeScript (Frontend):** Used in Next.js and React, this language adds strict safety checks to our interface code on top of JavaScript, ensuring a stable and bug-free user experience.
- **Java 21 (Backend):** Used in Spring Boot, Java provides the enterprise-grade stability and speed required to securely process financial transactions and real-time data.

5.2. TECHNOLOGY & FRAMEWORK

5.2.1. Frontend

The tech stacks selected to build frontend project is modern React-based technologies optimized for performance and scalability. Since our project is a trading app, the frontend of our system is required to be fast, responsive, and capable of handling real time trading data.

a) Framework

Our application runs on the latest versions of **Next.js 15** and **React 19**. This combination serves as the strong foundation of our project, handling everything from how pages load to how users navigate between screens. By using Next.js, we ensure the website is fast and search-engine friendly, while React allows us to build reusable building blocks for complex features like live trading charts and interactive dashboards. The system is organized using a modern "App Router" structure, which keeps public pages, user dashboards, and secure login areas neatly separated and secure.

5. Libraries

The list below is some main libraries used in our system to handle task efficiently:

- ***UI & Design***: ***Tailwind*** **CSS** and **Shadcn** components to create a clean, professional look that works well on mobile and desktop. Framer Motion is used to add smooth animations.
- ***State Management***: We use **Zustand** to handle general app settings and **TanStack Query** to efficiently fetch and update data from the server, ensuring numbers are always fresh.
- ***Charts & Tables***: For financial analysis, we use **TradingView** widgets to display professional candlestick charts. For listing large amounts of data (like transaction history), we use **TanStack Table** to keep scrolling smooth.
- ***Auth & Session management***: Role-based access pages managed securely by **NextAuth.js**. For data entry, we use **React Hook Form** and **Zod** to catch errors and validate inputs before they are sent to the server.
- ***Live Data***: Utilizing **Binance WebSockets** to stream real-time market updates instantly to the user without needing to refresh the page.

5.2.2. Backend

Our backend built using reliable Java technology – Spring boot. We have designed it to be secure, stable, and capable of handling real-time trading signals without delays. The system is

split into two main parts: a core server for business logic and a lightweight "listener" for incoming data.

a) Spring boot

Our server is built on **Spring Boot 3.5.6** utilizing **Java 21**. The system follows a RESTful architecture and MVC model with vertical slice folder structure. To keep the system safe; we employ **Spring Security** with **JWT** and **OAuth2** for authenticated sessions, ensuring strictly controlled access to data. For data persistence, we use **Spring Data JPA** to manage interactions with both MySQL (local development) and Azure MSSQL (production). The system also leverages asynchronous processing and scheduled tasks to poll **Azure Storage Queues**, ensuring trade signals are processed in real-time without blocking the main application.

6. Libraries

- **Ingestion layer (Azure function):** a serverless tool called azure function automatically catches trading signals from external bots and send to queue.
- **Lombok**: remove boiler code – help code clean and avoid repetitive actions.
- **Notification System:** We have an integrated email service that runs in the background to send verification emails and alerts to users without slowing down the app.

5.2.3. Database

MySQL (Development): We use MySQL for our local testing environment because it is lightweight, open-source, and easy to set up on developer machines.

Azure MSSQL (Production): For the live application, we switch to Azure SQL Database. This managed service provides enterprise-level security, automatic backups, and seamless integration with our backend.

5.3. TOOLS

- **Figma**: Used for designing user interfaces and prototyping.
- **VS Code**: Our primary code editor for development.
- **Postman**: Used for testing and verifying backend API endpoints.
- **GitKraken**: A visual Git client used to manage our Gitflow branching strategy.
- **Draw.io**: Used for creating system architecture diagrams.
- **Mermaid chart**: Used for drawing sequence diagrams.
- **MySQL Workbench**: Used for managing local databases during development.

5.4. VERSION CONTROL SYSTEM

- **GitHub:** Our central repository where all project code is hosted, allowing the team to collaborate and track changes online.
- **Gitflow Workflow:** We follow the Gitflow branching strategy. This organizes our work into specific branches (like "feature," "develop," and "main") to prevent bugs from reaching the production code.

5.5. DEPLOYMENT

Our deployment strategy ensures that both parts of our application run on platforms

optimized for their specific needs.

- **Azure App Service (Backend):** We host our Spring Boot application on Azure. This service is fully managed, meaning Microsoft handles the underlying infrastructure, security patching, and scaling for our Java server.
- **Vercel (Frontend):** We deploy our Next.js application on Vercel since vercel is the creator of next.js, meaning the deployment will be automatically optimized and run faster.

Deployment url: https://coinsantra.vercel.app/

CHAPTER 6: CONCLUSION

6.1. SUMMARY

The primary goal of the **Coinsantra** project was to build a secure, real-time trading platform that bridges the gap between complex financial markets and everyday users. We set out to create a system where traders could analyze market trends using professional tools and execute trades instantly, without the lag or clutter found in older systems.

Over the course of development, we successfully built a complete web ecosystem. The application combines a high-performance frontend (Next.js) with an enterprise-grade backend (Spring Boot), ensuring that price updates are instant and user data is always protected.

6.2. CONCLUSION

In conclusion, the Coinsantra project has met its core objectives. We have delivered a stable, functional trading application that is both fast for the user and easy to maintain for developers.

The project validated our technical choices. Using Next.js 15 gave us a snappy, responsive user interface that works well on all devices. On the server side, Spring Boot provided the "heavy lifting" power needed to handle financial transactions securely, while Azure services allowed us to scale up easily. The separation of the "Admin" and "Trader" roles has also proven effective in keeping the system secure and organized.

While the current web version is fully functional, we recognize that the financial market moves fast, and there are areas we can optimize further, such as reducing the initial load time for heavy charts and expanding our payment options.

References

- **Background processing:** Distributed Scheduling with Spring Boot: the challenges & pitfalls of implementing background job
- **VN Pay:** Tích hợp VNPAY vào ứng dụng Spring Boot [Môi trường kiểm thử] - PAD
- **Spring Security.** "OAuth 2.0 Resource Server JWT." *Spring Framework Documentation*. [Online]. Available: https://docs.spring.io/spring-security/reference/servlet/oauth2/resource-server/jwt.html
- **Next.js.** "Introduction." *Next.js Documentation*. [Online]. Available: https://nextjs.org/docs
- **NextAuth.js.** "REST API." *NextAuth.js Documentation*. [Online]. Available: https://next-auth.js.org/getting-started/rest-api
- **shadcn/ui.** "The Foundation for your Design System." *shadcn/ui Documentation*. [Online]. Available: https://ui.shadcn.com
- **Discussion:** Do we still need React Query with Next.js v14? | mariossimou.dev