**WealthNest**

A Fintech Investment Platform

**1 Introduction**

**Definition**

WealthNest is a digital investment and portfolio management platform that enables users to securely buy and track stocks, mutual funds, and other assets. It provides wallet management, financial analytics, and real-time prices, along with gamified investing features to enhance user engagement. The platform also includes a dedicated admin panel for compliance, monitoring, and financial oversight, ensuring security and transparency at every level.

**1.1 Company Profile**

Company Name: Freshcodes Technology

Location: Adajan, Surat, Gujarat, India

Industry: Software Development, Web & Mobile Solutions

**Company Overview:**

Freshcodes Technology is a dynamic and fast-growing IT company based in Adajan, Surat, specializing in web development, mobile application development, SaaS solutions, UI/UX design, and cloud-based software services. The company focuses on delivering high-quality, scalable, and customized software solutions to businesses across diverse industries.

With a talented team of developers, designers, and project managers, Freshcodes Technology follows modern development practices such as Agile methodology, clean architecture, and cloud-first design. Their commitment to innovation and client satisfaction positions them as a trusted technology partner for startups, SMEs, and enterprises.

**Vision:**

To empower individuals and organizations through advanced digital solutions that enhance productivity and enable future-ready technology transformation.

**Mission:**

To build secure, user-centric, and high-performance software products using modern tools, strong engineering practices, and continuous innovation.

**Core Services Offered:**

• Web Application Development

• Mobile App Development (Android & iOS)

• UI/UX Design

• Cloud & API Integrations

• IT Consulting

• Maintenance & Support

• Custom Software Solutions

**Company Strengths:**

• Skilled and experienced development team

• Strong focus on quality and user experience

• Modern technology stack and development practices

• On-time project delivery with agile approach

• High client satisfaction and transparency

**1.2 Project Profile**

**Project Title**: WealthNest – A Fintech Investment & Portfolio Management Platform

**Developed By**: Ms. Patel Kruti Pravinkumar

**Organization / Internship Company**: Freshcodes Technology, Adajan, Surat

**Project Duration**: 21 July 2025 – 21 November 2025

**Technologies Used**:

• Frontend: React (Vite + TypeScript), Tailwind CSS, Recharts

• User Backend: Express.js (Node.js + TypeScript)

• Admin Backend: ASP.NET Core (.NET 8 Web API)

• Database: Supabase (PostgreSQL + Auth)

**Project Overview**

WealthNest is a secure, modern fintech platform designed to help users simulate, manage, and learn investment strategies through real-time stock and mutual fund data. The platform allows users to maintain a digital wallet, perform buy/sell actions, track portfolio performance, view analytics, and earn achievements through a gamification system.

The system includes a fully-featured Admin Portal, built entirely in ASP.NET Core, enabling administrators to manage users, monitor transactions, maintain assets, analyze platform data, and maintain transparent audit logs. All core financial and user data is stored in Supabase PostgreSQL.

**Purpose of the Project**

The purpose of the WealthNest project is to develop a professional-grade fintech application that replicates the workflow of modern investment systems:

• Managing financial assets

• Tracking portfolios

• Authenticating users securely

• Maintaining administrative control

• Generating analytics

• Logging all admin actions

• Visualizing financial data

This project demonstrates real-world software engineering practices using a hybrid technology stack (React + Express + .NET + Supabase).

### 

### 2. PROPOSED SYSTEM

### 2.1 Scope

### The proposed system, WealthNest, is a comprehensive digital investment and portfolio management platform designed to provide users with an interactive and secure environment for learning and simulating investment activities. The system enables users to explore investments in stocks and mutual funds through real-time market data, simulated transactions, and detailed portfolio analytics. WealthNest integrates multiple financial learning tools including SIP and Lumpsum calculators, a daily “Word of the Day,” and an FAQ Chatbot to enhance user understanding of investment concepts.

### The platform also incorporates a secure user account system with email verification and PIN-based transaction authentication. A virtual wallet allows users to simulate deposits and withdrawals, thereby enabling risk-free practice of real-world investment strategies. Leaderboard points and achievement badges are awarded based on portfolio value to motivate users and promote continuous engagement.

### WealthNest additionally includes a robust Admin Portal, developed using ASP.NET Core, which allows administrators to monitor system activity, manage users, view transactions, update financial assets, analyze system-wide metrics, and review audit logs for transparency. Through this two-module architecture, WealthNest aims to deliver a reliable, scalable, and educational investment environment.

### 2.2 Objectives

### The primary objective of WealthNest is to offer an intuitive, secure, and educational platform that simulates investment activities while improving user awareness of financial markets. The detailed objectives include:

### User-Side Objectives

### • To enable users to create and manage their simulated investment portfolios efficiently by tracking stocks, mutual funds, and overall portfolio performance.

### • To provide real-time market price updates for stocks and mutual funds, supporting informed and accurate investment decision-making.

### • To implement a secure wallet management system where users can simulate deposits and withdrawals without actual financial transactions.

### • To facilitate the buying and selling of assets through a PIN-based verification mechanism, ensuring transaction security.

### • To offer analytical tools such as 24-hour portfolio change, unrealised P&L, 90-day performance graphs, and recent transaction history for improved financial insight.

### • To enhance user engagement through leaderboard points, badges, and global ranking based on portfolio value.

### • To provide supplementary learning tools such as a financial chatbot, SIP/Lumpsum calculators, Word of the Day, and live news updates.

### Admin-Side Objectives

### • To provide administrators with a complete dashboard for monitoring user activity, system funds, pending transactions, and user growth.

### • To allow admins to manage users efficiently by banning or unbanning accounts and reviewing user details.

### • To enable asset management functions, including adding, editing, and deleting stocks and mutual funds.

### • To offer comprehensive access to all user transactions for monitoring compliance and system integrity.

### • To provide analytical reports and visual representations of system data using charts.

### • To maintain audit logs for every critical admin operation, ensuring transparency and accountability.

### • To allow superadmins to create and manage other admin accounts.

### 2.3 Constraints

2.3.1 Hardware Constraints

• The system requires devices with a stable internet connection to ensure real-time data retrieval for market prices and dashboard updates.

• Performance may vary on low-end or outdated devices, especially when rendering charts or processing large datasets.

• Servers hosting the platform must have sufficient storage and processing capability to maintain historical data such as transactions, market prices, and user portfolios.

2.3.2 Software Constraints

• The application requires modern, up-to-date web browsers for compatibility with React-based UI components and real-time rendering features.

• The platform relies on third-party financial APIs whose rate limits or downtime may affect the accuracy and frequency of market updates.

• Authentication and security depend on proper implementation of Supabase Auth, JWT tokens, and encryption protocols.

• The system does not support offline mode due to dependency on real-time data and cloud-based database operations.

### 2.4 Advantages

### • WealthNest simplifies investment learning by providing an easy-to-use, interactive simulation environment for users who wish to understand stock and mutual fund markets.

### • Real-time analytics, including price updates and portfolio performance charts, provide users with practical exposure to market behavior.

### • The secure wallet system, along with PIN-based trading, ensures that all simulated transactions are performed safely, mimicking real-world security standards.

### • Leaderboard points, badges, and ranking systems encourage users to remain active and improve their investment strategies while competing with peers.

### • The dedicated Admin Portal ensures effective monitoring and management of users, assets, transactions, and system-wide activities.

### • Audit logs enhance the platform’s reliability and transparency by recording all administrative operations.

### 2.5 Limitations

### • The platform does not provide offline access, as real-time pricing and cloud-based storage require continuous internet connectivity.

### • Advanced AI-driven investment insights, portfolio recommendations, or predictive analytics are not included in the current version.

### • The system focuses on simulated trading and does not support real-money transactions or integration with payment gateways.

### • International market data and multi-exchange support are not implemented, restricting the platform to selected stocks and mutual funds.

### • Deep integration with external financial systems, brokers, or third-party trading platforms is not available.

### 3. Environment Specification

### 3.1.1 Hardware requirenments

|  |  |
| --- | --- |
| Components | Specification |
| Processor (CPU) | Intel Core i3 or above (or equivalent). Required for smooth rendering of dashboards, charts, and UI components. |
| RAM | Minimum 4 GB RAM required. Recommended 8 GB for optimal performance during development and testing. |
| Hard Disk | Minimum 500 MB free space for project files, logs, and cached data. Server storage must support historical user transactions and portfolio records. |
| Monitor | 15-inch or higher with at least 1024×768 resolution for proper UI layout and admin panels. |
| Internet Connectivity | A stable internet connection is essential for live market updates, news feed, and Supabase database communication. |

### 3.1.2 Software requirenments

|  |  |
| --- | --- |
| Components | Specification |
| Operating System | Windows 10/11, Linux, or macOS (for development and execution). |
| Frontend Framework | React |
| Backend Framework | Express.js (User APIs) and ASP.NET Core 8.0 (Admin APIs). |
| Database Server | Supabase PostgreSQL (cloud-hosted). |
| Browser | Latest versions of Google Chrome, Mozilla Firefox, Microsoft Edge, or Safari. |
| Text Editor / IDE | Visual Studio Code, Visual Studio 2022 |
| Package Managers | Node.js (npm), NuGet for .NET dependencies. |
| API & Utility Tools | Postman / Thunder Client for API testing, Supabase Dashboard for DB management. |

3.2 Development Description

The development of the WealthNest platform is carried out using a modern full-stack architecture that combines powerful frontend, backend, and database technologies to deliver a secure, scalable, and interactive investment simulation system. The application is divided into two major modules: the User Module, which handles all user-facing operations, and the Admin Module, which provides administrative oversight and system management capabilities.

The frontend of the application is developed using React (Vite + TypeScript). This technology stack enables the creation of a fast, responsive, and component-driven user interface. React is used to build all major screens, including the dashboard, portfolio view, wallet system, stock/mutual fund pages, calculators, and admin interfaces. Tailwind CSS is used for styling to ensure consistency, responsiveness, and a modern appearance across all devices.

For the backend, the system uses a dual-service approach.

• The User Backend is developed using Express.js (Node.js + TypeScript), which provides RESTful APIs for user authentication, portfolio operations, wallet management, trading actions, calculators, and real-time market data integration.

• The Admin Backend is implemented entirely in ASP.NET Core (.NET 8), which offers enhanced security, strong typing, and efficient performance for administrative tasks such as managing users, assets, transactions, audit logs, and analytics dashboards.

The database layer is powered by Supabase PostgreSQL, a cloud-hosted relational database service that also provides authentication, role-based access, and secure storage. All entities such as users, wallets, transactions, portfolio holdings, assets, achievements, logs, and admin accounts are stored in a normalized relational schema. Supabase’s built-in Auth is used for secure email-based login and account verification.

The application communicates with external APIs to fetch live market prices, financial news, and other dynamic content required for real-time simulation. Recharts is used extensively to generate visual analytics such as user growth, portfolio trends, transaction status distribution, and asset comparisons. The development environment incorporates tools such as Visual Studio Code, Visual Studio 2022, Postman, and the Supabase Dashboard to streamline coding, debugging, and database management.

Overall, the development process follows modular, scalable, and industry-standard practices. Each component—frontend, backend, and database—has a clear responsibility, enabling easier maintenance, better performance, and future enhancement capabilities. This multi-service approach ensures that WealthNest functions reliably as a complete fintech simulation platform for both users and administrators.

4. System Planning

4.1 Feasibility Study

A feasibility study is carried out to determine whether the proposed system, WealthNest, can be developed and implemented successfully within the given constraints of time, technology, and resources. It evaluates the practicality of the system from technical, operational, and economic perspectives.

1. Technical Feasibility

The development of WealthNest is technically feasible because the technologies required for the system are modern, well-documented, and widely supported. The frontend is built using React with TypeScript and Vite, ensuring high performance, component reusability, and responsive UI rendering. The User Backend, developed in Express.js (Node.js), provides fast and scalable REST APIs, while the Admin Backend is implemented using ASP.NET Core (.NET 8), known for its stability, security, and efficient API development capabilities.

The centralized Supabase PostgreSQL database supports relational data storage with strong consistency, making it suitable for managing financial transactions, portfolios, holdings, audit logs, and user accounts. Supabase Auth enables secure login and verification with minimal configuration. The integration of external APIs for live stock and mutual fund prices is also supported, provided network connectivity is stable.

All required development tools—such as Visual Studio Code, Visual Studio 2022, and Postman—are freely available and compatible with the chosen technology stack. Therefore, the platform can be implemented effectively without the need for specialized hardware or proprietary software.

2. Operational Feasibility

WealthNest is operationally feasible because it addresses real-world needs for investment simulation, financial learning, and portfolio tracking. The system offers an intuitive user interface that makes it easy for users to access dashboards, analyze portfolio performance, simulate trades, and manage their wallet. The addition of real-time news, financial vocab (Word of the Day), calculators, and chatbot tools further enhances user learning and engagement.

Administrators can operate the system efficiently through the dedicated ASP.NET Core Admin Portal, which provides user management, transaction monitoring, asset control, analytics, and audit logs. These operational features ensure transparency, accountability, and smooth functioning of the system.

The system requires minimal training, as the interface is designed to be simple, visually guided, and responsive across devices. Audit trails help maintain control and ensure safe operation.

3. Economic Feasibility

The system is economically feasible because it relies on widely available, cost-effective technologies. Most tools and frameworks used—React, Node.js, ASP.NET Core, and Supabase—are either free, open-source, or offer generous free-tier usage. This significantly reduces development and maintenance costs.

Client-side hardware requirements are minimal, and hosting the system on cloud services further decreases infrastructure expenses. Since WealthNest is a simulation-based platform, there is no need for integration with payment gateways or costly real-money trading partners. Additionally, the long-term benefits of providing a modern fintech learning tool outweigh the initial investment in development.

Thus, the system is economically viable and offers excellent value in terms of educational outcomes, scalability, and maintenance efficiency.

⸻

4.2 Software Engineering Model

The Agile Software Development Model is the most suitable approach for building WealthNest because it supports flexibility, continuous development, and frequent feedback. Since WealthNest consists of several interconnected modules such as user authentication, wallet operations, portfolio management, trading simulation, financial calculators, leaderboard, and the admin portal, the Agile model allows each module to be developed and improved in small iterative cycles.

In Agile, the system is divided into iterative development phases. Each iteration includes requirement analysis, design, coding, and testing of a specific portion of the system. This helps the development team respond quickly to new ideas, UI changes, improvements, and bug fixes.

Requirement Gathering:

During this phase, all functional and non-functional requirements were identified. Requirements included user login and verification, wallet management, stock and mutual fund trading, PIN authentication, portfolio analytics, calculators, chatbot, live news integration, admin dashboards, asset management, transaction logs, and audit logging. These requirements served as the foundation for designing the database, system architecture, and user interface.

Iterative Development:

After gathering requirements, the development process moved forward in small iterations. Each iteration focused on designing and implementing a particular feature or module. For example, in one iteration, user authentication and email verification were developed, followed by wallet operations, portfolio calculations, trading functions, calculators, news integration, and finally administrative features. After each iteration, the feature was tested and refined based on outcomes.

Continuous Integration and Testing:

Every newly developed feature was immediately tested for correctness, performance, and security. Testing included verifying APIs, validating user inputs, checking database consistency, ensuring smooth UI navigation, and confirming admin operations. Early feedback from testing helped identify issues quickly and incorporate improvements in subsequent iterations.

User-Centric Improvements:

Agile also promotes frequent review and refinement. Features like portfolio design, dashboards, charts, and admin analytics were adjusted based on functionality, clarity, and usability. This approach ensured that the system remained intuitive, accurate, and aligned with the intended user experience.

The Agile model ultimately enabled WealthNest to evolve as a stable, scalable, and user-friendly investment simulation platform through continuous refinement and structured development cycles.

⸻

4.3 Risk Analysis

Risk analysis helps identify potential issues that may arise during the development and operation of WealthNest. Managing these risks ensures system reliability, security, and long-term stability.

1. Technical Risk

Technical risks may arise due to integration failures between multiple technologies (React, Express.js, .NET, Supabase). API rate limits or downtime from external services may affect real-time pricing or news features. To minimize this, error handling, retry mechanisms, and fallback data strategies were planned.

2. Security Risk

Since WealthNest deals with user authentication, wallet balances, and transaction logs, any breach can compromise sensitive data. Strong encryption, JWT validation, PIN-based transaction security, and Supabase Auth safeguards are implemented. Regular security reviews mitigate vulnerabilities.

3. Operational Risk

Users may face difficulty navigating complex modules such as trading or portfolio analysis. This risk is reduced through intuitive UI, labels, clean navigation, and help components like chatbot and tooltips.

4. Requirement Risk

Changes in requirements during development may impact timelines or create inconsistencies. Using Agile methodology reduces this risk by allowing rapid adjustments during sprint cycles.

5. Performance Risk

Rendering live prices, charts, and large transaction histories may slow down the system. Performance is optimized using caching, efficient queries, and limiting API calls.

6. Schedule Risk

Delays may occur due to unforeseen technical challenges. Proper sprint planning, buffer weeks, and progress tracking help overcome this risk.

7. Data Loss Risk

Any database issue may lead to loss of wallet data or transactions. Supabase backups, row-level security, and proper constraints mitigate this.

8. Compatibility Risk

The system must work on multiple devices and browsers. Cross-browser testing and responsive design reduce compatibility issues.

4.4 Project Schedule

4.4.1 Task Dependency

The development of WealthNest involves a series of interdependent tasks that must be completed in a logical sequence to ensure the system functions efficiently and integrates smoothly. The initial and most essential task is requirement analysis, where the functional and non-functional requirements of both the user module and the admin module are identified. Only after gathering clear requirements can the system design begin. The design phase includes database schema creation, entity relationship modeling, UI wireframing, and defining the interaction between frontend, backend, and external APIs.

Once the design is finalized, database configuration and backend setup depend on it, as the structure of tables—such as Users, Wallets, Assets, Transactions, Portfolio\_Holdings, Achievements, Admins, and Audit Logs—must be implemented according to the approved schema. Similarly, backend API development for both Express.js (User side) and ASP.NET Core (Admin side) relies heavily on this initial planning.

After the backend begins taking shape, frontend development depends on the availability of backend APIs. Features such as login verification, wallet updates, trading operations, portfolio dashboards, leaderboard calculations, and admin functionalities require working endpoints to test and validate the integration. Data flow between frontend and backend must be seamless to maintain the accuracy of market prices, investment values, and analytics dashboards.

Once all modules are developed, testing procedures depend on the completion of individual components. Unit testing ensures each module’s correctness, while integration testing verifies interactions such as buying assets, updating portfolio values, wallet balance adjustments, and admin oversight functionalities. System testing ensures that user modules, admin modules, and database operations work cohesively as a unified application.

Finally, deployment and documentation depend on the successful outcome of testing. Once testing confirms that the application is stable and reliable, the project can be deployed and prepared for submission. Thus, every phase of WealthNest’s development depends on the successful completion of previous steps, ensuring a structured and error-free implementation.

4.4.2 Timeline Chart

|  |  |  |
| --- | --- | --- |
| Week Range | Phase | Description |
| Week 1-2 | Requirement Analysis | Understanding features, studying user/admin needs, defining scope, preparing requirement documents |
| Week 3-4 | System Design | Architecture design, UI wireframes, database schema, ER diagram, module flow design |
| Week 5-7 | Frontend Development | React UI development for user and admin modules, API integration, charts, dashboards |
| Week 8-11 | Backend Development | User backend (Express.js), admin backend (.NET), API creation, Supabase integration |
| Week 12-13 | Testing & Debugging | Unit testing, integration testing, validation checks, bug fixes, improvements |
| Week 14-15 | Integration & Optimization | Full system integration, performance enhancement, UI polishing, stability checks |
| Week 16 | Documentation & Submission | Preparing project report, diagrams, screenshots, final review and submission |

**5. System Analysis**

5.1 SRS (Software Requirement Specification)

The Software Requirement Specification (SRS) defines all functional and non-functional requirements of the WealthNest system. It describes how the system should behave, the operations it must support, and the constraints under which it must operate. The SRS ensures clarity between developers, stakeholders, and evaluators, and serves as the foundation for system design, development, testing, and validation.

⸻

A. Functional Requirements (User Module)

1. User Registration and Login

• The system must allow users to register using their email, password, and required personal information.

• Email verification must be completed before login is permitted.

• The login system must authenticate users securely through Supabase Auth.

• If the logged-in email belongs to an admin, the system must redirect to the Admin Portal; otherwise, it must route to the User Dashboard.

⸻

2. User Profile Management

• Users must be able to view and update their personal information such as name, photo, and contact details.

• Users must be able to view their leaderboard points, global rank, and achievement badges.

• The profile must display joining date and account details.

⸻

3. Dashboard Overview

• After login, users must be shown a dashboard containing:

• Total portfolio value

• 24-hour change percentage

• Wallet balance

• Unrealised profit/loss

• 90-day portfolio performance graph

• Recent transactions from the last 90 days

• Dashboard must update dynamically with market price changes.

⸻

4. Wallet Management

• The system must maintain a simulated wallet for deposits and withdrawals (no real payment integration).

• Users must be able to deposit or withdraw any amount.

• The system must log every deposit and withdrawal in the Transactions table.

• A summary must show the percentage of deposits vs withdrawals.

• Wallet balance must update instantly after each operation.

⸻

5. Portfolio Management

• Users must be able to view their total portfolio value.

• The portfolio page must list:

• Stocks owned

• Mutual funds owned

• Units held

• Average buy price

• Current market price

• Current value and P/L

• Portfolio must update based on live asset prices.

• System shall export filtered SIP schedules and portfolio holdings (and optionally transactions) to CSV

⸻

6. Stock Trading (Buy/Sell)

• Users must be able to browse the list of available stocks.

• Before making the first trade, the user must set a 4-digit PIN.

• Every buy/sell operation must require PIN confirmation.

• For a buy order, the wallet balance must be decreased accordingly.

• For a sell order, the quantity must be deducted from the user’s holdings.

• All transactions must be recorded with type (buy/sell), amount, quantity, and status (completed/failed).

⸻

7. Mutual Fund Operations (SIP/Lumpsum/Redemption)

• Users must be able to invest in mutual funds using:

• SIP (Systematic Investment Plan)

• Lumpsum Amount

• Users must be allowed to redeem their mutual fund units.

• A PIN is required for all MF transactions.

• SIP, Lumpsum, and Redemption records must be maintained in the Transactions table.

⸻

8. Calculators

The system must provide three financial calculators:

• SIP Calculator: Calculates future value based on monthly investment, duration, and expected returns.

• Lumpsum Calculator: Calculates maturity value of a one-time investment.

• Goal Calculator: Calculates required monthly SIP amount to reach a future goal.

Each calculator must produce accurate financial results using standard compound interest formulas.

⸻

9. Achievement & Leaderboard System

• Users earn points = portfolio value / 100.

• Points determine achievement badges:

• 100 points – Bronze

• 500 points – Silver

• 1000 points – Gold

• 5000 points – Sapphire

• 10000 points – Diamond

• Leaderboard must rank users globally based on total points.

• User achievements must be stored in the Achievements and User\_Achievements tables.

⸻

10. Live News & Word of the Day

• The system must fetch and display real-time financial news on the home page.

• “Word of the Day” must update daily and provide financial terminology for user learning.

• News and Word of the Day must refresh automatically.

⸻

11. Chatbot

• The chatbot must answer general finance and system-related questions.

• It must guide users through features such as trading, portfolio, calculators, and wallet usage.

⸻

⸻

B. Functional Requirements (Admin Module)

1. Admin Authentication

• Admin must log in using Supabase Auth.

• Only admin emails should access the Admin Portal.

• Role-based access must be enforced:

• Superadmin → full access

• Admin/Employee → restricted access

⸻

2. Admin Dashboard

The dashboard must display real-time system metrics:

• Total users

• Banned users

• Pending transactions

• Total system funds (sum of all users’ wallet balances)

• Quick access to management modules

⸻

3. User Management

• Admin must be able to view all registered users.

• Search and filter options must be available.

• Admin can ban/unban any user.

• All ban/unban actions must be recorded in the Audit Log.

⸻

4. Transaction Management

• Admin must be able to view all user transactions.

• Filtering based on transaction type or status must be supported.

• Admin must be able to monitor buy, sell, deposit, withdrawal, SIP, and Lumpsum transactions.

⸻

5. Asset Management

• Admin can create, edit, or delete stocks and mutual funds.

• All assets must have:

• Name

• Symbol

• Type

• Price

• Updating/deleting an asset must be recorded in audit logs.

⸻

6. Reports & Analytics

The Reports page must include visual charts showing:

• User growth (line chart)

• Transaction status distribution (pie chart)

• Asset distribution by type (bar chart)

• Transaction volume trends (bar chart)

Charts must be generated using Recharts.

⸻

7. Audit Logs

• Every admin action—ban, update asset, delete asset, create admin, etc.—must be recorded.

• Logs must store timestamp, action, and details for complete transparency.

• Logs must be viewable in the Admin Portal.

⸻

8. Admin Management (Superadmin Only)

• Superadmin must be able to create new admin accounts.

• Admin list must display name, email, and role.

• Role changes must be restricted to superadmin.

⸻

C. Non-Functional Requirements

1. Usability

• The system must offer a clean, responsive, and intuitive interface for both users and admins.

• Navigation must be simple and consistent across all pages.

2. Security

• All transactions (buy, sell, SIP, redeem) must require a secure PIN.

• Passwords must be securely hashed.

• Admin operations must be backed by role-based access and audit logging.

• JWT and Supabase Auth must be used for secure access.

3. Performance

• Dashboard and analytics pages must load efficiently even with large datasets.

• Live market updates must be processed without performance degradation.

4. Scalability

• The system must support increasing users, assets, and transactions without redesign.

• Supabase PostgreSQL must scale automatically with database load.

5. Maintainability

• Code must be modular and structured to support future enhancements.

• Clean API structure should allow easy updates or replacements.

6. Availability

• The system must remain available as long as the hosting platform and internet connection are active.

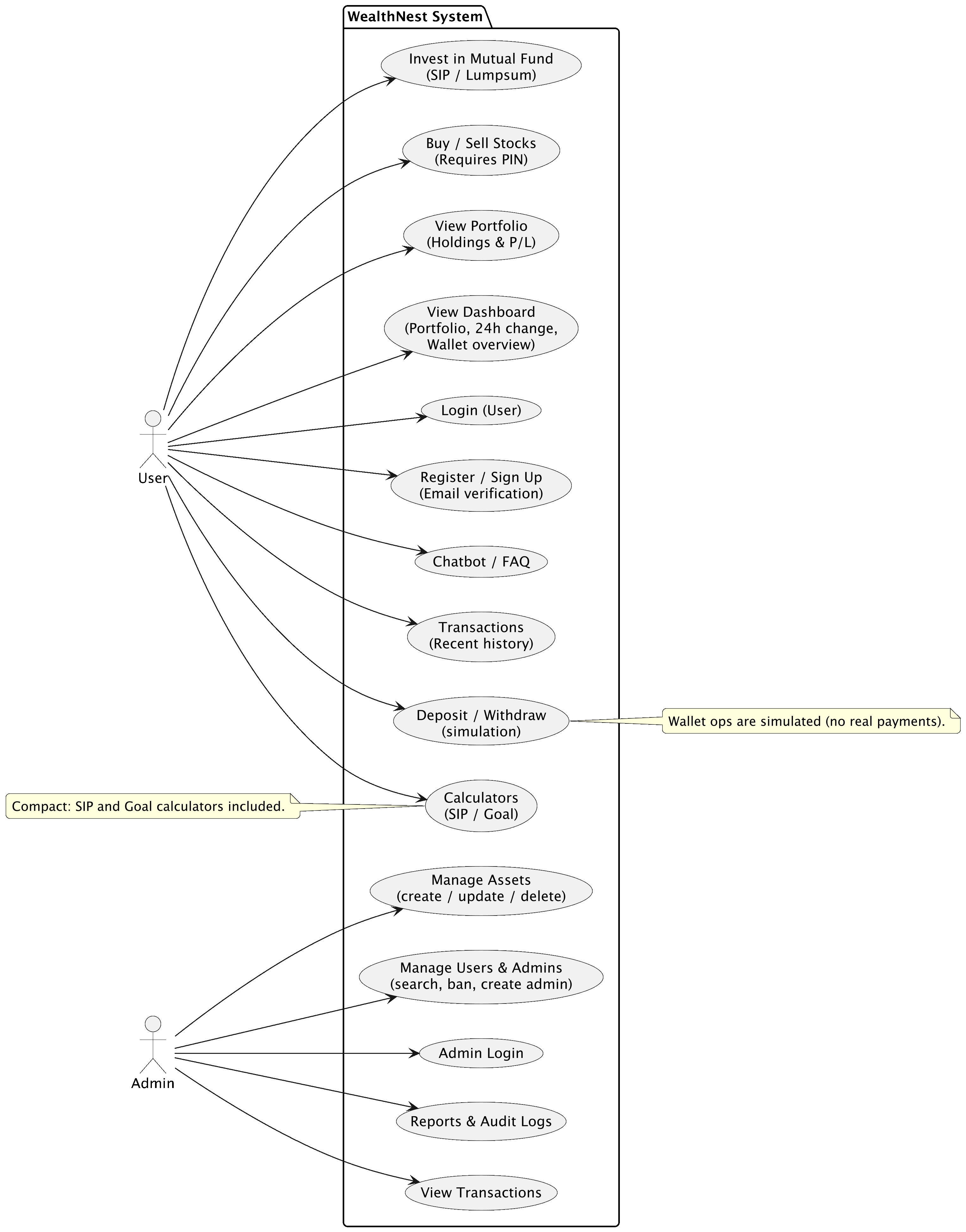
• Cloud-hosted database ensures availability without downtime.

7. Reliability

• All operations must maintain data accuracy and integrity.

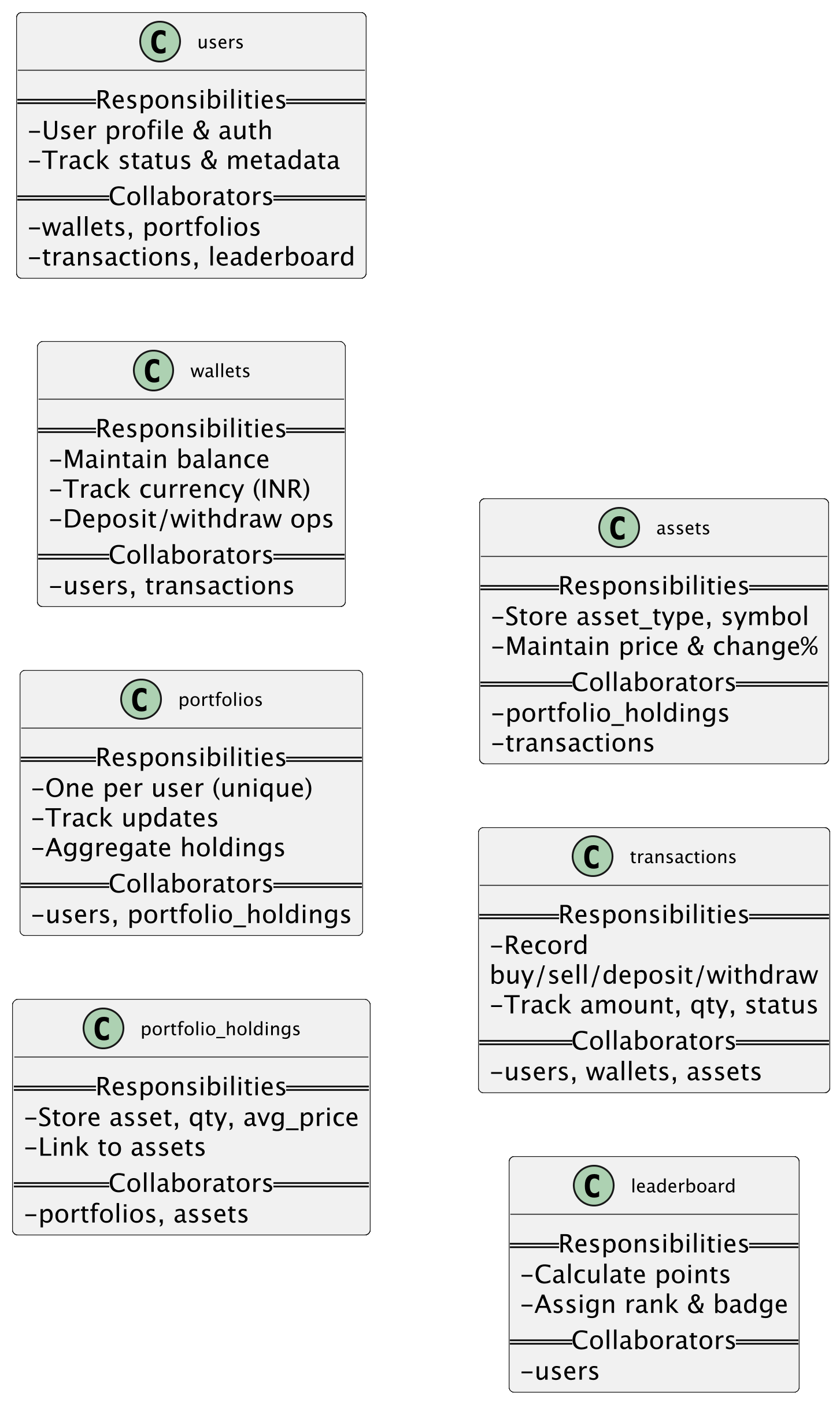
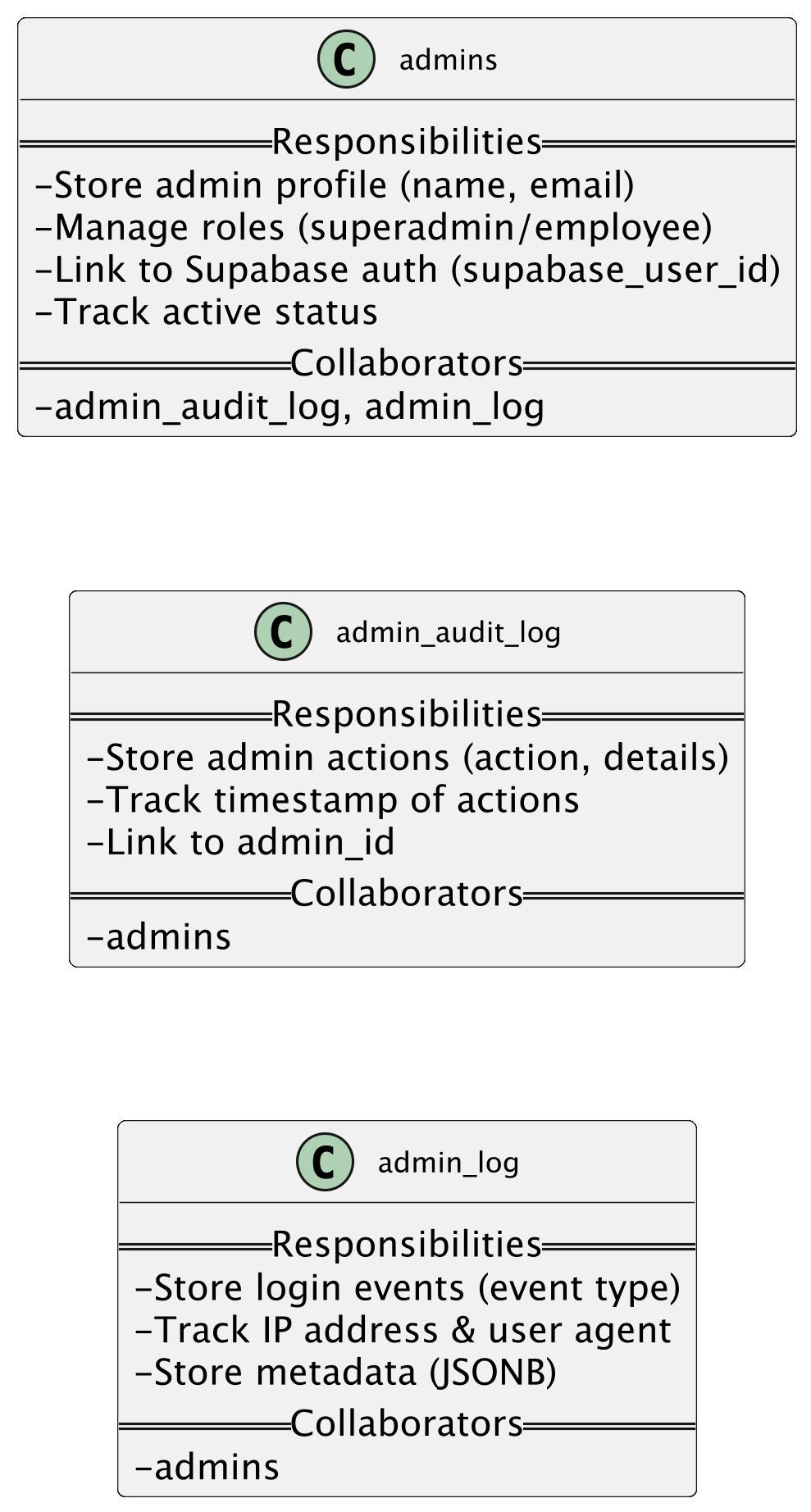
• Wallet balance and transaction records must always remain consistent.

5.2 UML Diagram

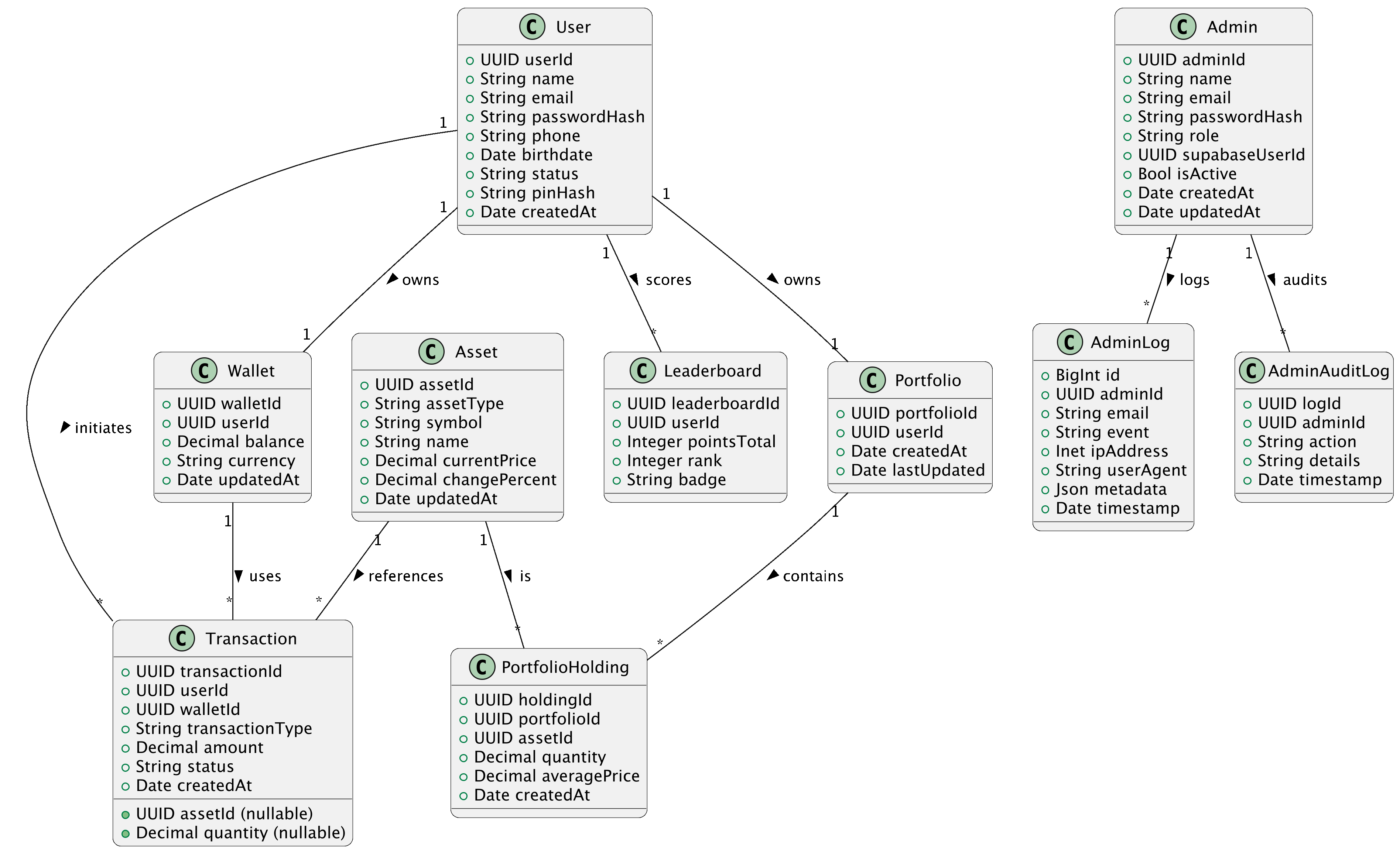


5.2.1 Use Case Diagram

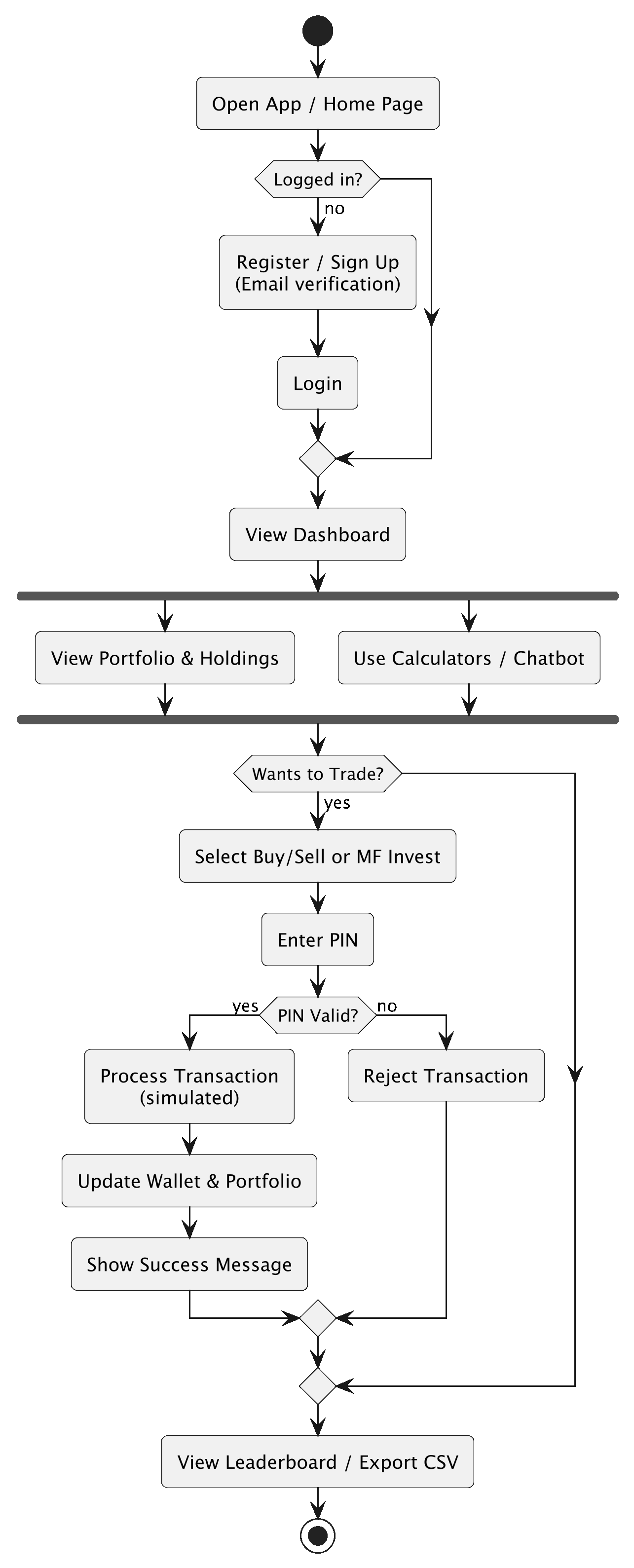
5.2.2 CRC (Class–Responsibility–Collaborator)



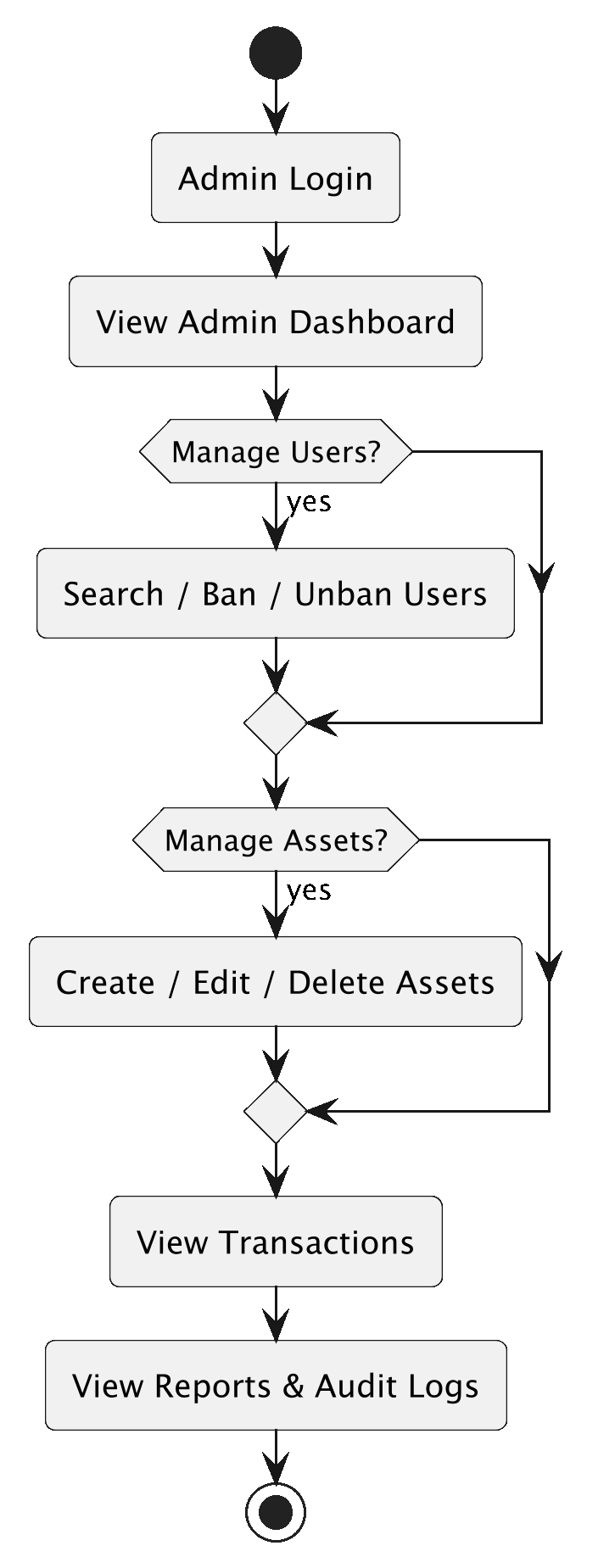
5.2.3 Class Diagram



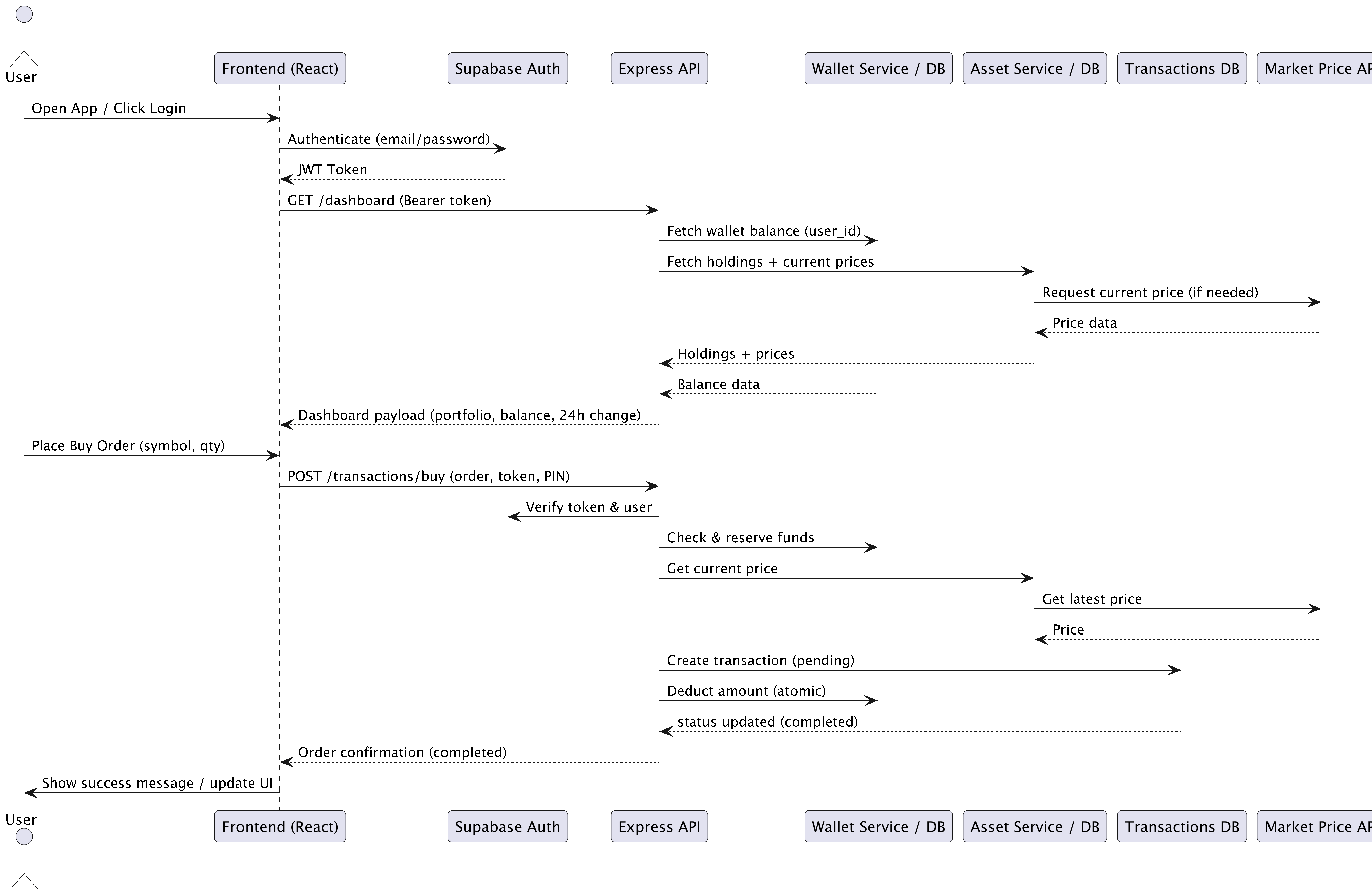
5.2.4 Activity Diagram



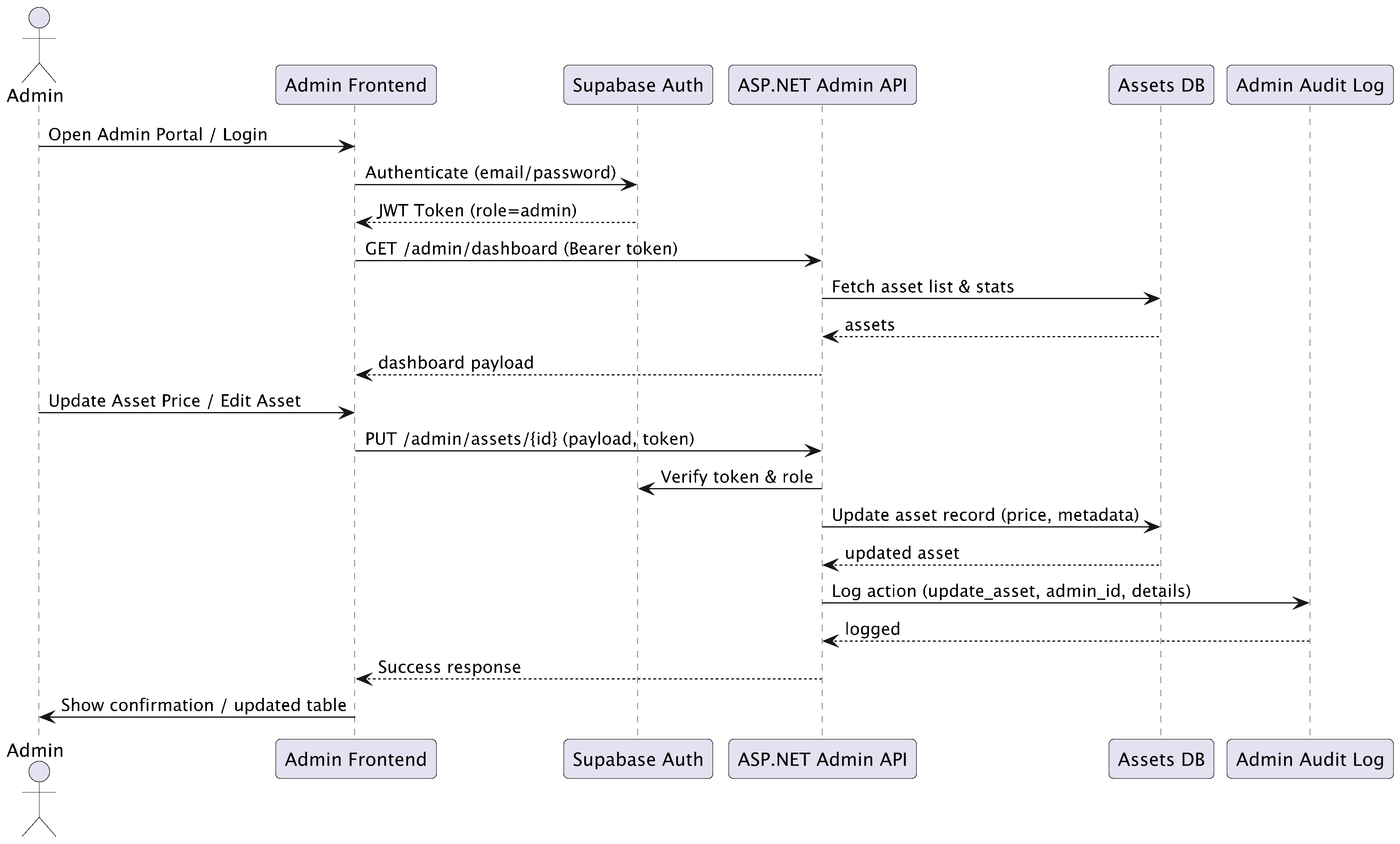
User

Admin

5.2.5 Sequence Diagram

User

Admin



**6. Software Design**

Software design defines the internal structure of the WealthNest system, including how data is stored, how users interact with the interface, and how different layers of the application communicate with each other. This section explains the database structure, user interface design, and system architecture used for developing the platform.

**6.1 Database Design**

## **USERS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| user\_id (PK) | UUID | Unique identifier for each user |
| name | VARCHAR | Full name of the user |
| email | VARCHAR | Unique email (login credential) |
| phone | VARCHAR | Contact number |
| password\_hash | VARCHAR(255) | Securely hashed password |
| birthdate | DATE | User’s date of birth |
| address | TEXT | Residential address |
| created\_at | TIMESTAMP | Account creation timestamp |
| status | VARCHAR | Active / Banned |
| pin | BIGINT | User’s transaction PIN (simulation) |

## **WALLETS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| wallet\_id (PK) | UUID | Unique wallet identifier |
| user\_id (FK) | UUID | Linked user account |
| balance | NUMERIC | Current wallet balance |
| currency | VARCHAR | Currency type (INR) |
| updated\_at | TIMESTAMP | Last updated time |

## **PORTFOLIOS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| portfolio\_id (PK) | UUID | Unique portfolio ID |
| user\_id (FK) | UUID | Owner of the portfolio |
| created\_at | TIMESTAMP | Portfolio creation time |
| last\_updated | TIMESTAMP | Last updated timestamp |

## **PORTFOLIO\_HOLDINGS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| holding\_id (PK) | UUID | Unique holding ID |
| portfolio\_id (FK) | UUID | Linked portfolio |
| asset\_id (FK) | UUID | Asset being held |
| quantity | NUMERIC | Quantity owned |
| average\_price | NUMERIC | Average buy price |
| created\_at | TIMESTAMP | Timestamp when holding was recorded |

## **ASSETS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| asset\_id (PK) | UUID | Unique asset ID |
| asset\_type | VARCHAR | stock / mutual\_fund / bond / crypto |
| symbol | VARCHAR | Trade symbol |
| name | VARCHAR | Full asset name |
| current\_price | NUMERIC | Latest price |
| updated\_at | TIMESTAMP | Last price update |
| change\_percent | NUMERIC | Price % change |

## **TRANSACTIONS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| transaction\_id (PK) | UUID | Unique transaction ID |
| user\_id (FK) | UUID | User who performed transaction |
| wallet\_id (FK) | UUID | Wallet used |
| asset\_id (FK) | UUID | Asset involved |
| transaction\_type | VARCHAR | buy / sell / deposit / withdraw |
| amount | NUMERIC | Transaction amount |
| quantity | NUMERIC | Units involved |
| status | VARCHAR | pending / completed / failed |
| created\_at | TIMESTAMP | Transaction timestamp |

## **LEADERBOARD**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| leaderboard\_id (PK) | UUID | Unique leaderboard entry |
| user\_id (FK) | UUID | User associated |
| points\_total | INTEGER | Points (portfolio\_value/100) |
| rank | INTEGER | Global rank |
| badge | TEXT | Badge level |

## **ADMINS**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| admin\_id (PK) | UUID | Unique admin ID |
| name | VARCHAR | Admin’s name |
| email | VARCHAR | Login email |
| password\_hash | VARCHAR | Hashed password |
| role | VARCHAR | superadmin / employee |
| created\_at | TIMESTAMP | Created date |
| supabase\_user\_id | UUID | Linked Supabase user |
| is\_active | BOOLEAN | Admin status |
| updated\_at | TIMESTAMP | Last update timestamp |

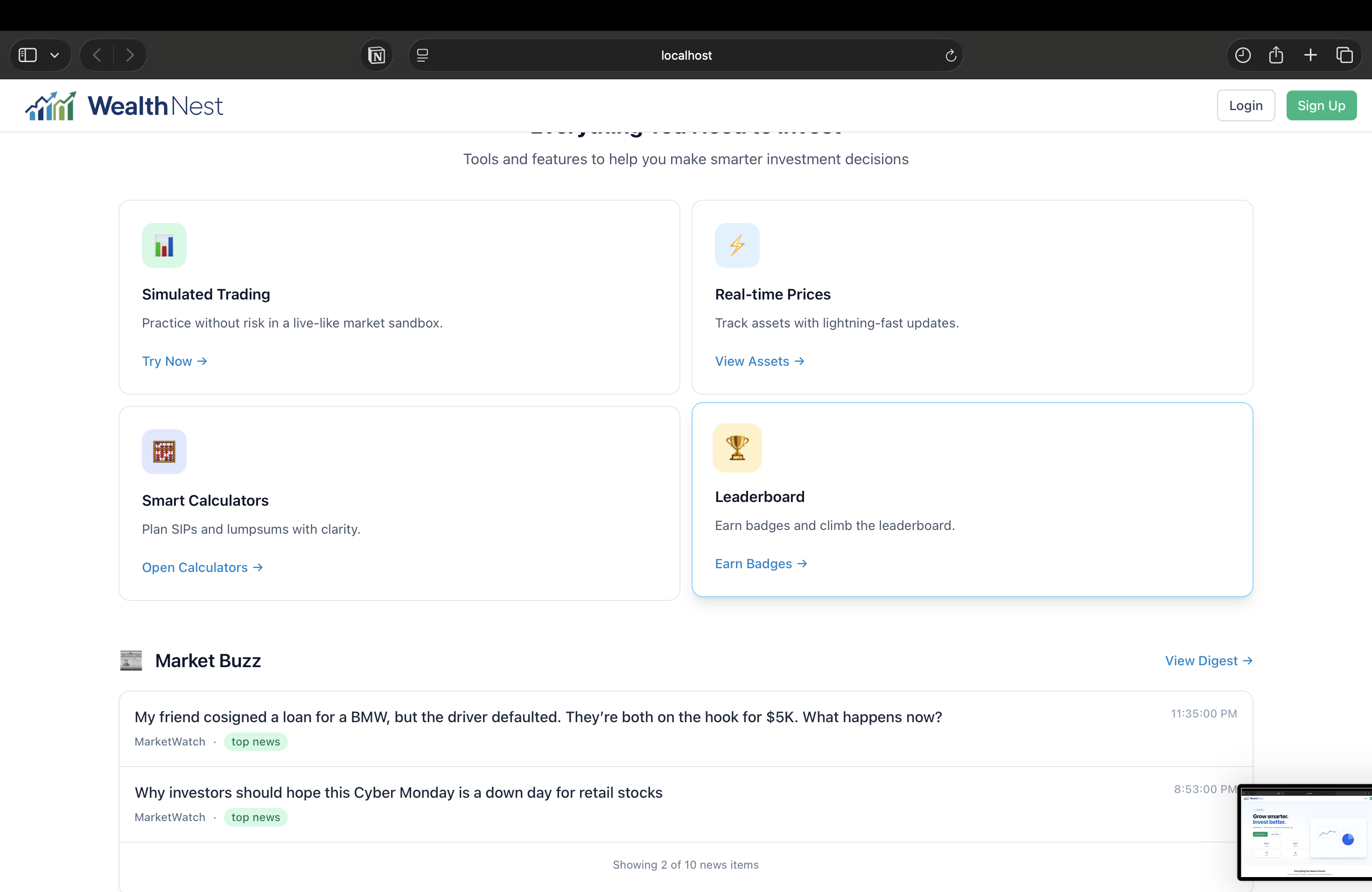
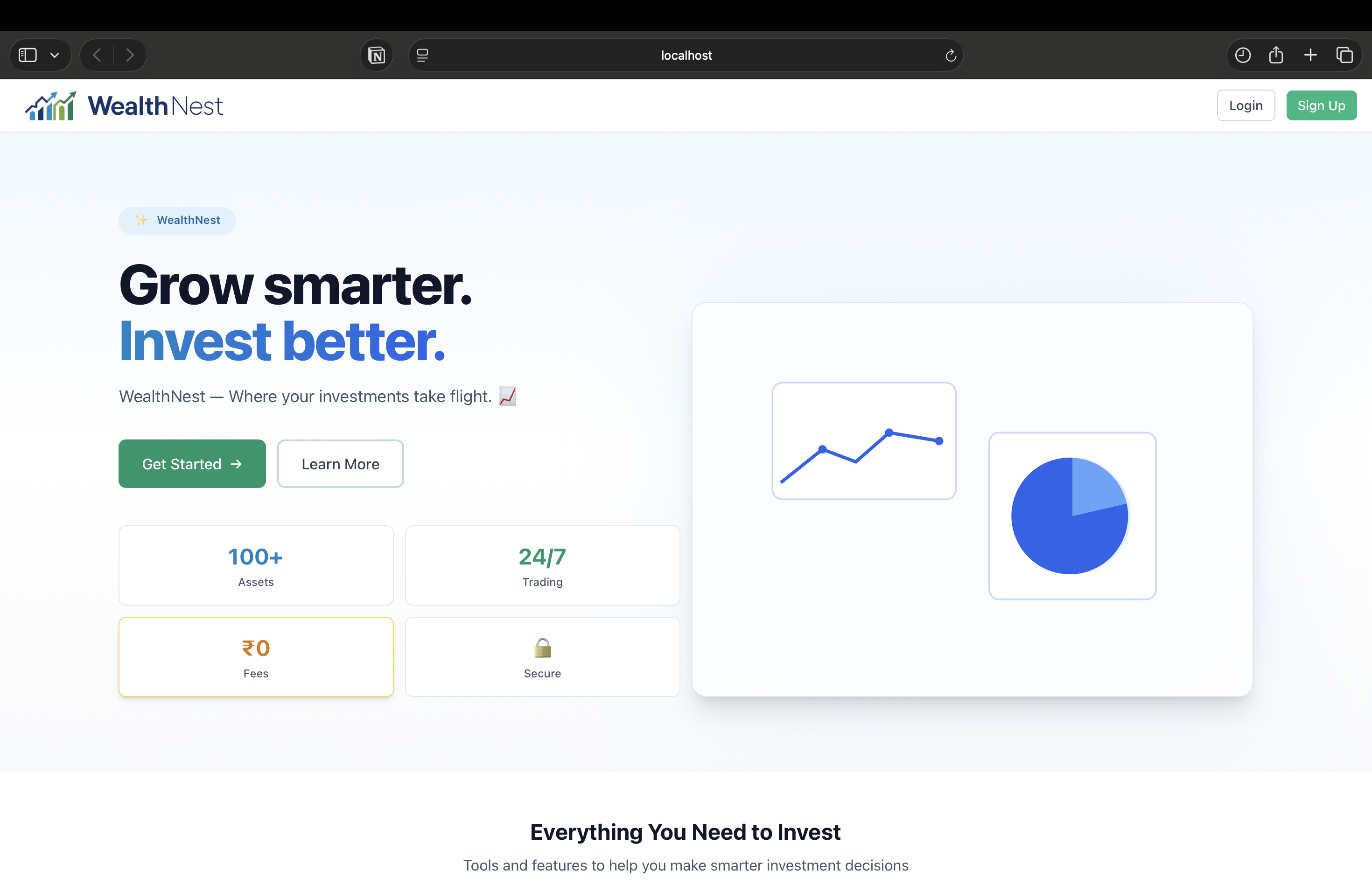
## **ADMIN\_LOG**

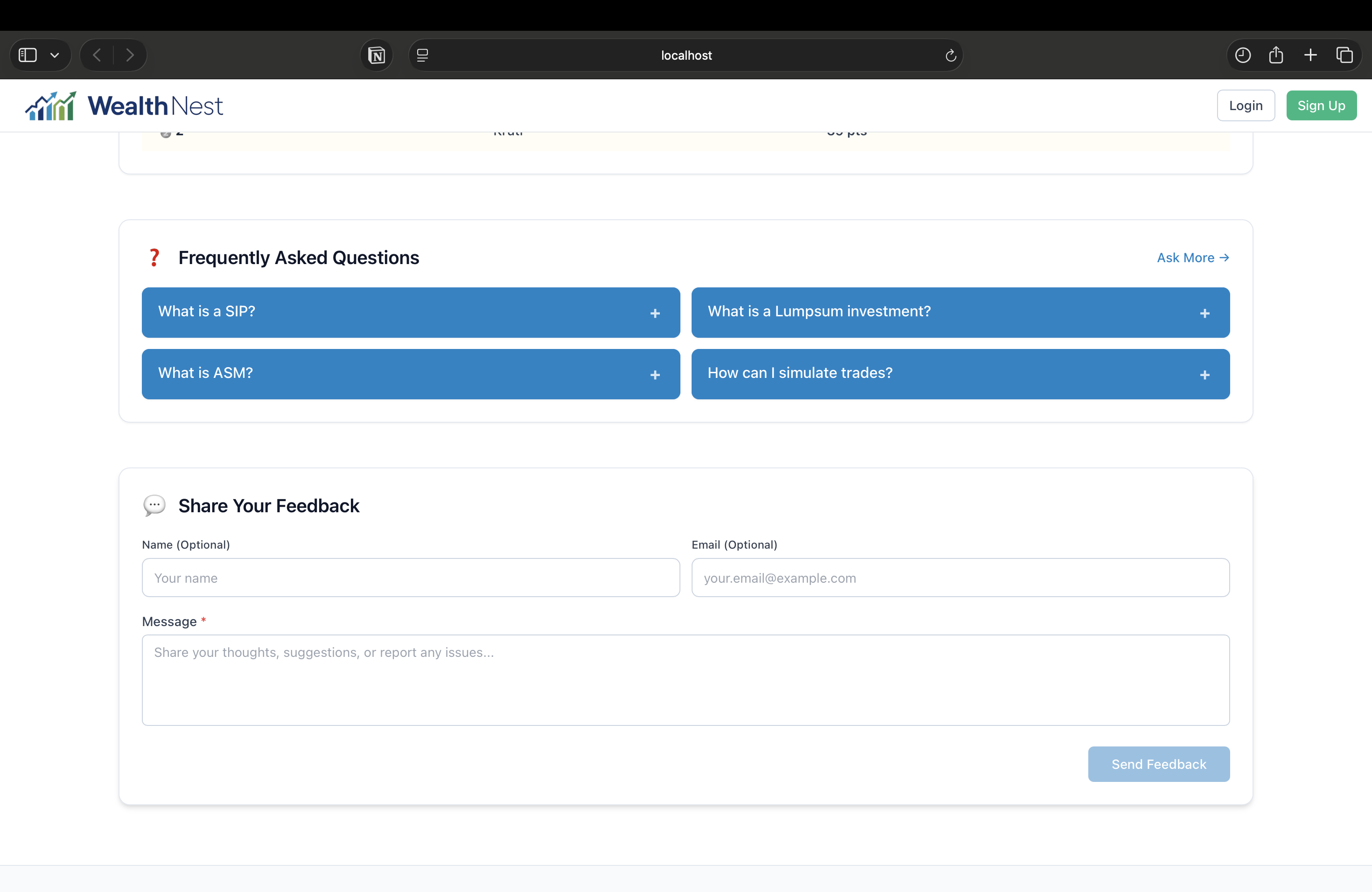
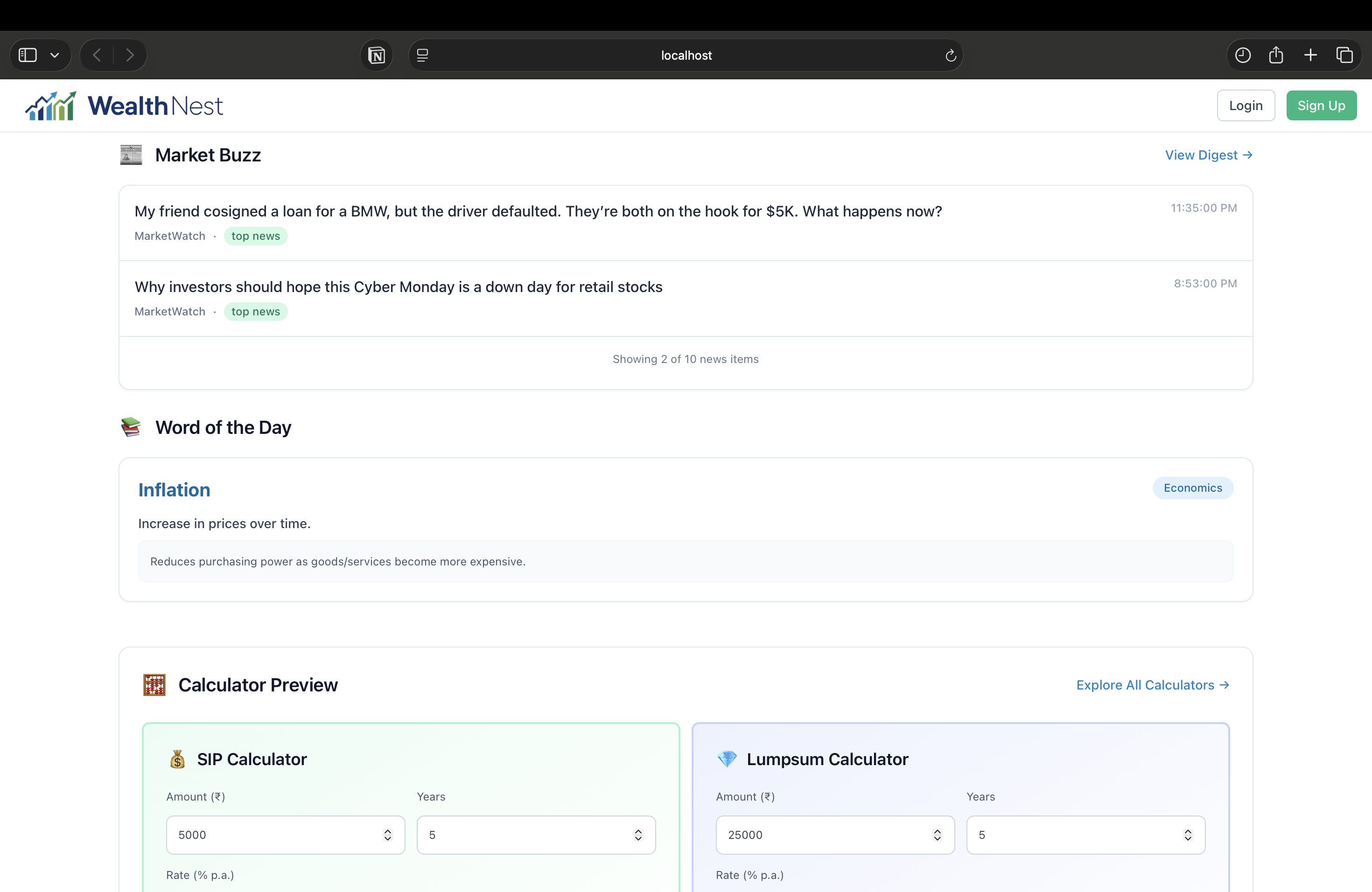
|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| id (PK) | BIGINT | Unique admin log ID |
| admin\_id (FK) | UUID | Admin reference |
| email | VARCHAR | Admin email |
| event | VARCHAR | login\_success / login\_failed / logout |
| ip\_address | INET | IP Address |
| user\_agent | TEXT | Browser/device info |
| metadata | JSONB | Metadata |
| timestamp | TIMESTAMPTZ | Event timestamp |

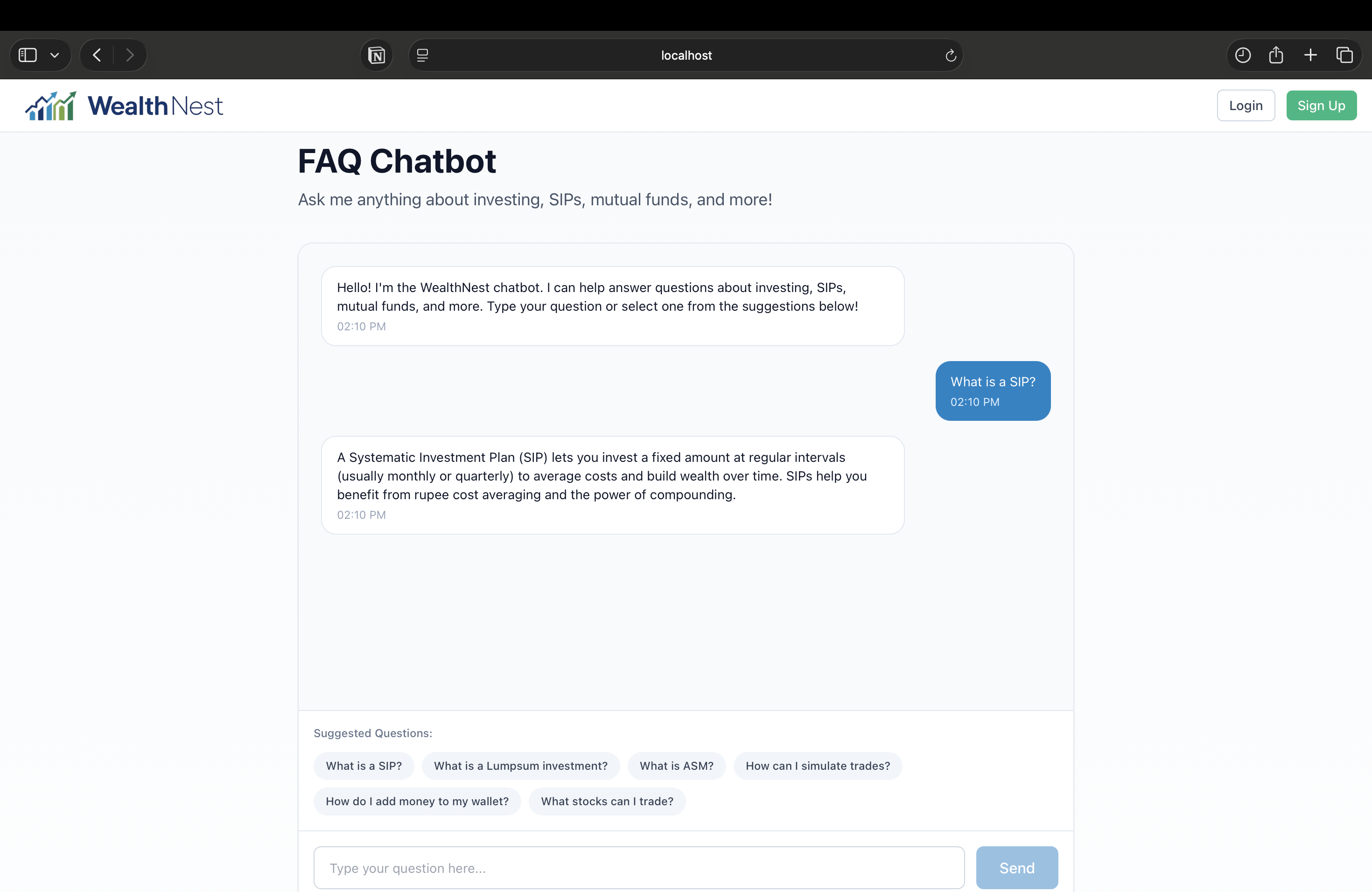
## **ADMIN\_AUDIT\_LOG**

|  |  |  |
| --- | --- | --- |
| Column Name | Data Type | Description |
| log\_id (PK) | UUID | Unique audit log entry |
| admin\_id (FK) | UUID | Admin reference |
| action | VARCHAR | Action performed |
| details | TEXT | Details of the action |
| timestamp | TIMESTAMP | Action timestamp |

**6.2 Interface Design**

The user interface of WealthNest is designed using React with TypeScript, following a component-based architecture. Tailwind CSS is used for styling to provide a modern, minimalistic, and responsive UI. The system contains two user flows: User Portal and Admin Portal.



****

**6.3 Architecture Design (3-Tier Architecture)**

WealthNest follows a 3-Tier Architecture, which divides the system into three independent layers. This increases scalability, maintainability, and modularity.

⸻

1. Presentation Layer (Frontend)

Technology: React (Vite + TypeScript)

Responsibilities:

• User and admin UI

• Forms, dashboards, charts

• Portfolio, wallet, calculators, news, chatbot

• API communication via HTTPS

• Route-based rendering for user/admin separation

This layer interacts with backend services only through REST APIs, ensuring a clean separation from business logic.

⸻

2. Business Logic Layer (Backend APIs)

Split into two independent services:

a. User Backend – Express.js

Handles:

• User authentication

• Wallet operations

• Portfolio & holdings

• Trading logic (buy/sell/SIP/lumpsum/redemption)

• Fetching assets

• Leaderboard & achievements

• Financial calculators

• News, WOTD, chatbot integration

b. Admin Backend – ASP.NET Core

Handles:

• Admin authentication

• User management

• Asset CRUD operations

• Transaction monitoring

• Reports & analytics

• Audit logging

• Admin management

This layer validates requests, enforces rules, and maintains platform logic.

⸻

3. Data Layer (Database – Supabase PostgreSQL)

Responsibilities:

• Stores all persistent data

• Maintains relational integrity

• Tracks admin actions

• Supports analytical queries

• Ensures data security via Supabase Auth and RLS

The database is cloud-hosted, ensuring availability and automated backups.

**7. Testing**

Testing is an essential phase to ensure the correctness, reliability, and stability of the WealthNest system. Various testing techniques were applied to validate both the User Module and the Admin Module, ensuring that all components work as intended, data remains consistent, and the system behaves reliably under different usage scenarios.

⸻

7.1 Unit Testing

Unit testing was performed on individual components, functions, and API endpoints to verify that each feature operates correctly in isolation. Key units tested include:

• User authentication and email verification

• Wallet updates (deposit/withdraw)

• Buy/Sell stock operations with PIN validation

• SIP, Lumpsum, and Redemption calculations

• Portfolio value computation and 24-hour change logic

• Leaderboard point calculation

• Admin asset creation, editing, and deletion

• Admin user ban/unban operations

• Audit logging of admin activities

Each function was tested with valid and invalid inputs to ensure accurate processing and error handling.

⸻

7.2 Integration Testing

Integration testing ensured that different modules of the system work together seamlessly. This involved validating the communication between the frontend, backend APIs, and the database. Important integrations tested include:

• Frontend UI with Express.js User APIs

• Admin React screens with ASP.NET Core Admin APIs

• Wallet operations updating portfolio and transaction records

• Live price APIs updating portfolio values

• Admin actions reflected in audit logs

• Leaderboard updates after trading or wallet changes

• Consistency between Portfolio, Holdings, and Transactions tables

Integration testing confirmed that workflows such as login → trade → portfolio update → leaderboard adjustment function correctly across all modules.

**8. Future Enhancement**

Even though WealthNest currently works as a simulation-based investment platform, there are several improvements that can be added in the future to make the system more powerful, realistic, and user-friendly. These enhancements can help the platform grow from a learning tool into a more advanced investment system.

⸻

1. Real-Time Market Price Integration

In the future, the platform can use more accurate real-time APIs for both stocks and mutual funds.

This will make:

• Portfolio values

• Profit/Loss

• Graphs and charts

much closer to real market behaviour instead of basic simulation.

This improves learning and gives users a more realistic experience.

⸻

2. Move from Simulation to Real Investments

Right now, all buy/sell operations are simulated.

In future versions, WealthNest may support real trading by integrating:

• Broker APIs

• Payment gateways

• Bank/UPI deposit options

This would allow users to make actual investments (with proper security and verification).

⸻

3. Stronger Security & Compliance

If real investments are added in the future, the system will require:

• KYC verification

• Secure payment handling

• Monitoring and safety checks

• Strong legal and regulatory compliance

This ensures protection for users and the platform.

⸻

4. Better Reliability for Market Data

As market data becomes more real-time, the system may include:

• Backup data providers

• Local caching when the main API fails

• Indicators to show users when data is delayed

This improves system stability and user trust.

⸻

5. More Trading & Investment Features

Future updates may add:

• Advanced order types (limit order, stop-loss)

• Real-time order status

• Scheduled SIP auto-debits (if real payments are added)

• Support for more asset types

These features make the platform more complete for learning.

⸻

6. Mobile Application

To improve accessibility, a dedicated mobile app for Android and iOS can be developed so that users can view portfolios, news, calculators, and trade easily from their phones.

⸻

7. Alerts & Notifications

The platform can include:

• Price alerts

• Market updates

• SIP reminders

• Portfolio movement notifications

These help users stay updated even when they are not using the platform.

⸻

8. Community Features

In the future, users can interact through:

• Discussion forums

• Community leaderboard

• Learning groups

This will improve engagement and financial knowledge sharing.

⸻

**9. Glossary**

|  |  |
| --- | --- |
| Term | Definition |
| Admin Panel | Backend interface used by administrators to manage the system. |
| Asset | A financial product such as a stock or mutual fund. |
| Audit Log | A record of admin actions for security and tracking. |
| Buy/Sell | Trading actions to purchase or sell assets (simulated). |
| Dashboard | Main screen showing portfolio value and recent activity. |
| Leaderboard | A ranking system based on user points and achievements. |
| Lumpsum | One-time mutual fund investment of a fixed amount. |
| Mutual Fund | A pooled investment instrument managed by fund houses. |
| PIN | A 4-digit code required to confirm trades securely. |
| Portfolio | Collection of stocks and mutual funds owned by a user. |
| SIP | A regular monthly investment plan for mutual funds. |
| Transaction | Any financial action such as buy, sell, deposit, or withdraw. |
| Unrealised P/L | Profit or loss from assets that are still held, not sold. |
| Wallet | Simulated digital balance used for trading operations. |
| Word of the Day | A daily financial term shown to help users learn |

10. Reference

1. Supabase Documentation – <https://supabase.com/docs>
2. Supabase PostgreSQL Reference – <https://supabase.com/database>
3. Express.js Documentation – <https://expressjs.com>
4. Node.js Documentation – <https://nodejs.org/en/docs>
5. .NET Official Documentation (Microsoft) – <https://learn.microsoft.com/aspnet/core>
6. Recharts Documentation – <https://recharts.org/en-US>
7. Finnhub Stock Market & News API – <https://finnhub.io/docs/api>
8. GeeksforGeeks – <https://www.geeksforgeeks.org>
9. Stack Overflow – <https://stackoverflow.com>
10. GitHub Documentation – <https://docs.github.com>

11. Youtube tuorials - <https://www.youtube.com/watch?v=2TgkVLdK4Jc>