Software Architecture and Design Specification

Project: Digital Asset and Cryptocurrency Portfolio Tracker

Version: 1.0

Authors: Harshith J, Hemashree S, Jeevitha S, Kartik Bhat Sumbly

Date: 04-10-2025

Status: Draft

Revision History

1.0 | 01/10/2025 | Hema Shree R | Initial Draft of SAD (Architecture + Design sections created)
1.1 | 02/10/2025 | Jeevitha S | Added UML Diagrams (Use Case, Sequence, Class)
1.2 | 03/10/2025 | Harshith J | Added API Design and Error Handling sections
1.3 | 04/10/2025 | Kartik Bhat Sumbly | Completed UX Design and Security Architecture sections
1.4 | 04/10/2025 | QA Review Team | Final Review and Approval for Submission

15. Approvals

Role	Name	Signature / Date
QA Lead	Jeevitha S	Jeevitha 04/10/2025
Dev Team	Harshith J (Lead) Kartik Bhat Sumbly	Harshith J 04/10/2025 Kartik Bhat Sumbly 04/10/2025
Test Eng	Hema Shree R	Hema Shree R 04/10/2025
Product Owner	Mr Nandi Keshavan	

1. Introduction

1.1 Purpose

This document specifies the architecture and design of the Digital Asset and Cryptocurrency Portfolio Tracker application. It serves as a reference for developers, testers, security auditors, and maintenance teams to ensure the system is secure, reliable, and meets all functional and non-functional requirements as defined in the SRS.

1.2 Scope

The application provides a platform for tracking digital assets and cryptocurrency portfolios. It includes:

- Real-time price monitoring of cryptocurrencies across multiple exchanges.
- Portfolio performance analysis with historical trends and ROI calculations.
- Investment recommendations and alerts.
- Data visualization for easy interpretation of portfolio performance.
- API integrations for wallets, exchanges, and fiat conversion.

The system does not handle trading or asset custody; it only tracks and analyzes existing holdings.

1.3 Audience

- Developers & Designers: Implement system architecture and modules.
- QA Engineers: Develop test cases and validate functionalities.
- Security Auditors: Ensure compliance with security standards.
- Maintenance Teams: Support updates, monitoring, and feature expansion.
- Stakeholders & Investors: Review architecture for compliance and feasibility.

.

1.4 Definitions

- **API:** Interface to fetch data from exchanges or wallets.
- **Portfolio:** Collection of digital assets owned by a user.
- **ROI:** Return on Investment for portfolio performance.
- **Fiat Currency:** Government-issued currency like USD, EUR, INR.
- **2FA:** Two-Factor Authentication for enhanced security.
- **WebSocket:** Protocol for real-time updates from APIs.

2. Document Overview

2.1 How to use this document

This document provides architectural deliverables including UML diagrams, design rationale, component descriptions, technology stack, threat models, and API specifications. It aligns each module with corresponding requirements from the SRS for traceability.

2.2 Related Documents

- SRS (Software Requirements Specification)
- Test Plan (STP)
- Requirements Traceability Matrix (RTM)

3. Architecture

3.1 Goals & Constraints

Goals:

- Real-time portfolio tracking with ≤3-second latency.
- Secure storage and transmission of API keys and credentials.
- High availability (≥99.5% uptime).

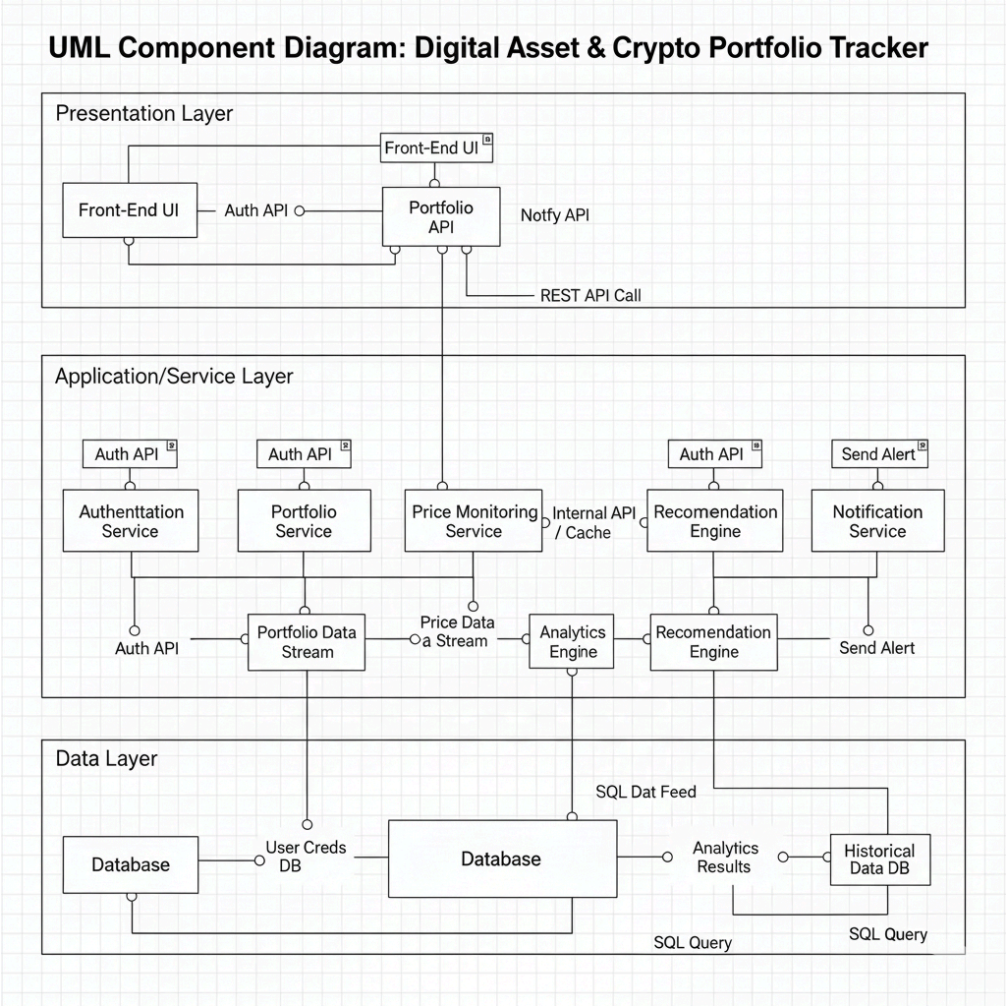
Constraints:

- API rate limits imposed by exchanges.
- Data accuracy dependent on third-party providers.
- Compliance with best practices for user data privacy.

3.2 Stakeholders & Concerns

Stakeholder	Concern
Users	Real-time prices, security, portfolio accuracy
Developers	Modularity, API integrations
Financial Analysts	Historical data analysis, visualization
Regulators	Security, data protection
System Admins	Uptime, scalability, maintainability

3.3 Component (UML) Diagram



3.4 Component Descriptions

- **Front-End UI:** Responsive dashboards, charts, and portfolio analytics interface.
- **Authentication Service:** Handles account login, 2FA, and secure password storage.
- **Portfolio Service:** Aggregates balances from wallets and exchanges, maintains cached data.

- **Price Monitoring Service:** Fetches live cryptocurrency prices via REST APIs/WebSockets.
- **Analytics Engine:** Computes ROI, profit/loss, and asset allocation for visualization.
- **Recommendation Engine:** Generates alerts and basic investment insights based on trends.
- **Notification Service:** Sends push/email alerts for portfolio and price events.
- **Database:** Stores user preferences, historical data, and cached portfolio info

3.5 Chosen Architecture Pattern and Rationale

- **Layered Architecture:** Separates concerns (presentation, business logic, data) for maintainability and modularity.
- Microservices architecture was considered but rejected due to project scale; layered design simplifies integration with multiple external APIs.

3.6 Technology Stack & Data Stores

- **Back-End:** Java, Spring Boot, Node.js (optional for WebSocket integration).
- **Front-End:** React.js (web), React Native / Flutter (mobile).
- **Database:** PostgreSQL/MySQL for persistent data; Redis for caching.
- **APIs:** REST/GraphQL for exchanges, wallets, and fiat conversion.
- **Security:** TLS 1.2+, AES-256 encryption for sensitive data.

3.7 Risks & Mitigations

Risk	Mitigation
API downtime / rate limit	Retry logic, caching, fallback data
Data breach / credential leak	AES-256 encryption, 2FA, secure logging
Performance bottlenecks	Load balancing, WebSocket streaming
Visualization errors / incorrect data	Unit and integration tests on analytics

3.8 Traceability to Requirements

Requirement ID	Module / Service
DAP-F-001 to F-004	Authentication Service
DAP-F-005 to F-007	Portfolio Service
DAP-F-008 to F-010	Price Monitoring Service
DAP-F-011 to F-013	Analytics Engine
DAP-F-014 to F-015	Recommendation & Notification

3.9 Security Architecture

Threat Modeling (STRIDE):

- **Spoofing:** Secure OAuth and API key validation.
- **Tampering:** Encrypted storage and HTTPS communication.
- **Information Disclosure:** TLS 1.2+ for all API communications.
- **Denial of Service:** Rate limiting, caching, and load balancing.
- **Elevation of Privilege:** Role-based access and 2FA enforcement.

4. Design

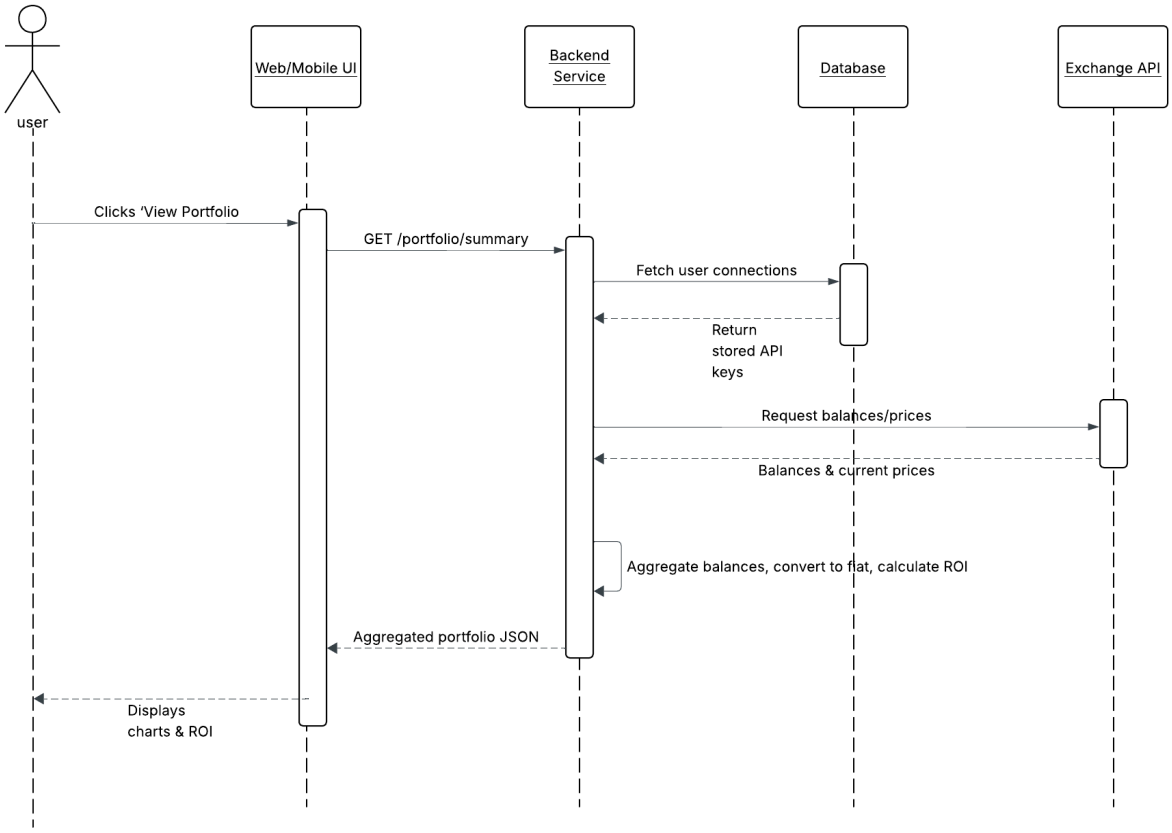
4.1 Design Overview

The Digital Asset & Cryptocurrency Portfolio Tracker follows a **layered architecture**:

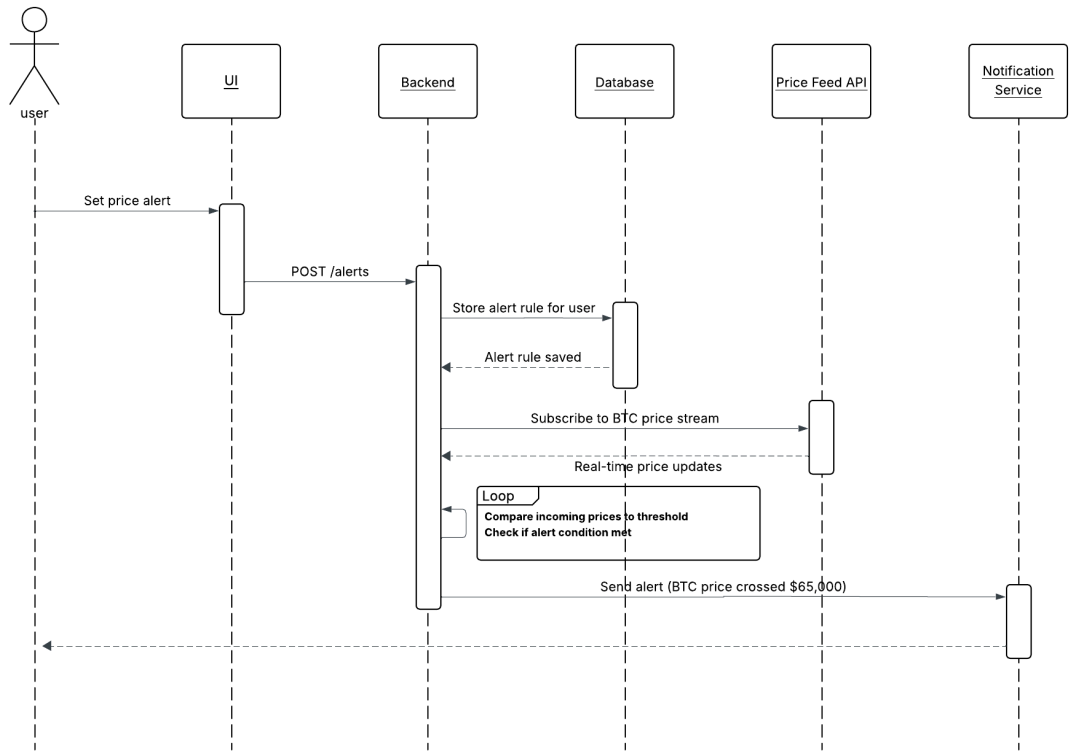
- **Presentation Layer** – Web/Mobile UI built with React (web) and Flutter (mobile).
- **Application/Service Layer** – RESTful APIs handling authentication, portfolio aggregation, analytics, and notifications.
- **Integration Layer** – Connectors for cryptocurrency exchange/wallet APIs and currency-conversion services.
- **Data Layer** – Cloud database (PostgreSQL) and caching (Redis) for user profiles, portfolio data, and historical prices.

4.2 UML Sequence Diagrams

Sequence Diagram 1: Price Alert Notification Flow



Sequence Diagram 2: Price Alert Notification Flow



4.3 API Design

POST /api/v1/portfolio/connect

- Request: `{ exchange, apiKey, secret }`
- Response: `{ status, message }`
- Errors: `400 InvalidCredentials`, `429 RateLimitExceeded`.

GET /api/v1/portfolio/summary

- Request: `{ userId }`
- Response: `{ totalValueUSD, assetBreakdown[], roi }`
- Errors: `401 Unauthorized`, `503 ServiceUnavailable`.

4.4 Error Handling, Logging & Monitoring

Error Handling: Consistent JSON error schema `{code, message}`.
Logging: Structured logs with masked sensitive data; store in centralized logging service (e.g., ELK).

Monitoring: Metrics for API latency, price-update delays, and alert-delivery success via Prometheus/Grafana.

4.5 UX Design

- Clean, responsive dashboard with light/dark themes.
- Interactive charts (ROI, asset allocation).
- Mobile-friendly navigation; WCAG 2.1 AA accessibility.
- Push/email notification settings.

4.6 Open Issues & Next Steps

- Add advanced AI-based investment recommendations.
- Support on-chain wallet signature login.
- Explore multi-language support for global user base.

5. Appendices

5.1 Glossary

- **API** – Application Programming Interface for exchanges/wallets.
- **ROI** – Return on Investment.
- **Fiat Currency** – Government-issued currency such as USD or INR.
- **2FA** – Two-Factor Authentication.

5.2 References

- IEEE 42010: Software Architecture Description.
- OWASP Top 10 Security Guidelines.
- NIST SP 800-160 for secure software development.

5.3 Tools

- **Design**: Lucidchart for UML diagrams.
- **API Docs**: Swagger/OpenAPI.
- **Monitoring**: Prometheus + Grafana.
- **CI/CD**: GitHub Actions, Docker.