Cairo University

Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

InvestEase

Software Design Specifications

Version 1.0

ID	Name	Email	Phone
20230053	Israa Abd Elhaq	Israa Mail	01094415513
20230421	Menna Talla Gamal	Menna Mail	
20230370	Mahmoud Hosny	Mahmoud Mail	

CS251: Three Bugs
Project: InvestEase

Software Design Specification

April-2025

Contents

CS251: Three Bugs
Project: InvestEase

Software Design Specification

Team

ID	Name	Email	Mobile
20230053	Israa Abd Elhaq	israa1912005@gmail.com	01094415513
20230421	Menna Talla Gamal	mennatallagamal@outlook.com	
20230370	Mahmoud Hosny	mahmoudhosnii20@gmail.com	

Document Purpose and Audience

Purpose

Define requirements for a Sharia-compliant, multi-asset investment management app tailored to Egypt's market, addressing gaps in local financial tools.

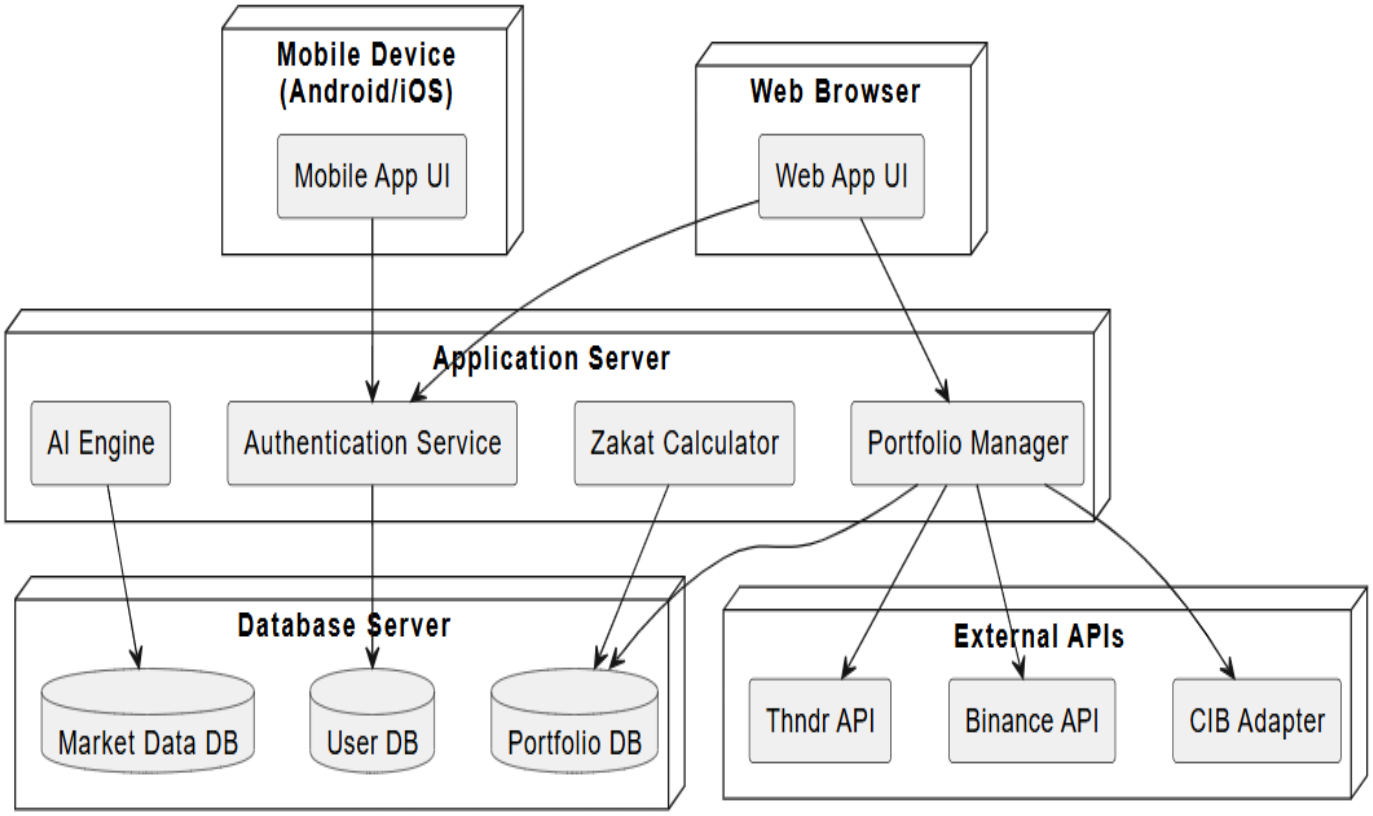
Audience

- Developers (backend, frontend)
- Stakeholders (Egyptian investors, Islamic finance experts)

Software Design Specification

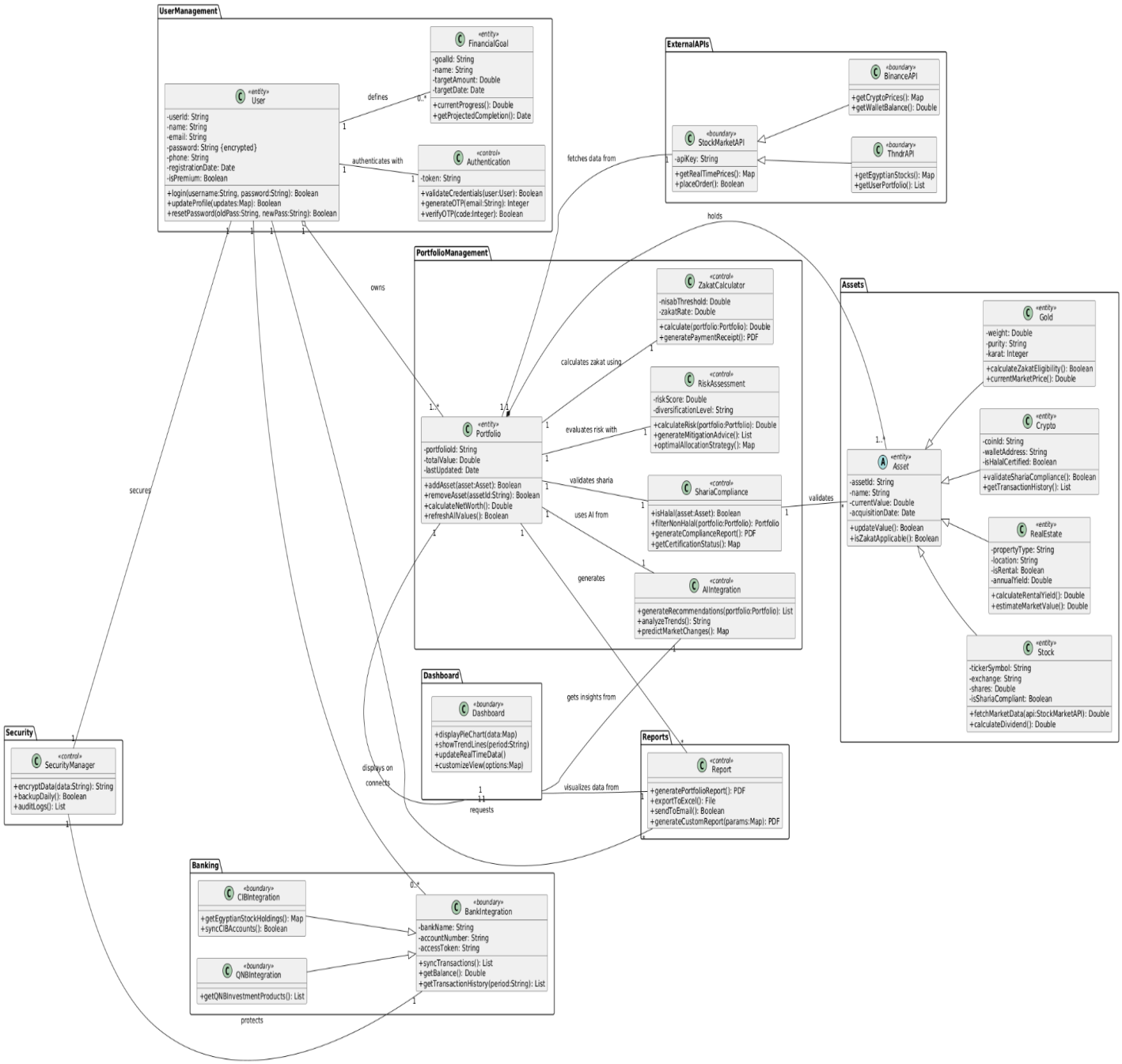
System Models

I. Architecture Diagram



Software Design Specification

II. Class Diagram(s)



Software Design Specification

III. Class Descriptions

Class ID	Class Name	Description & Responsibility
1	User	Entity class that represents the application's user, managing profile details, login credentials, and user-related operations like profile updates and password resets.
2	Authentication	Control class that handles user authentication processes including validating credentials, generating and verifying OTPs.
3	SecurityManager	Control class responsible for data encryption, daily backups, and log auditing to maintain security and integrity of user and banking data.
4	FinancialGoal	Entity class that defines a user's financial target with a name, amount, and due date, and tracks progress toward its completion.
5	Transaction	Records asset transactions such as buy, sell, deposit, and withdraw. Calculates transaction value and stores type and date.
6	Dashboard	Boundary class that provides a visual interface to the user for displaying portfolio data using charts, trends, and customizable views.
7	Portfolio	Entity class representing a user's investment portfolio, managing assets, net worth calculation, and real-time value updates.
8	RiskAssessment	Control class that evaluates the risk level of a portfolio, providing mitigation advice and suggesting optimal asset allocation.
9	ZakatCalculator	Control class that calculates the zakat amount based on the user's portfolio and generates a payment receipt accordingly.
10	AIIntegration	Control class that analyzes portfolio data, generates investment recommendations, predicts market changes, and extracts trend insights.
11	Asset	Abstract entity class representing a generic investment asset, with methods for updating value and checking zakat applicability.
12	Stock	Entity subclass of Asset representing stock investments with data like ticker symbol and exchange; can fetch market data and calculate dividends.
13	RealEstate	Entity subclass of Asset representing real estate investments, capable of estimating market value and calculating rental yields.

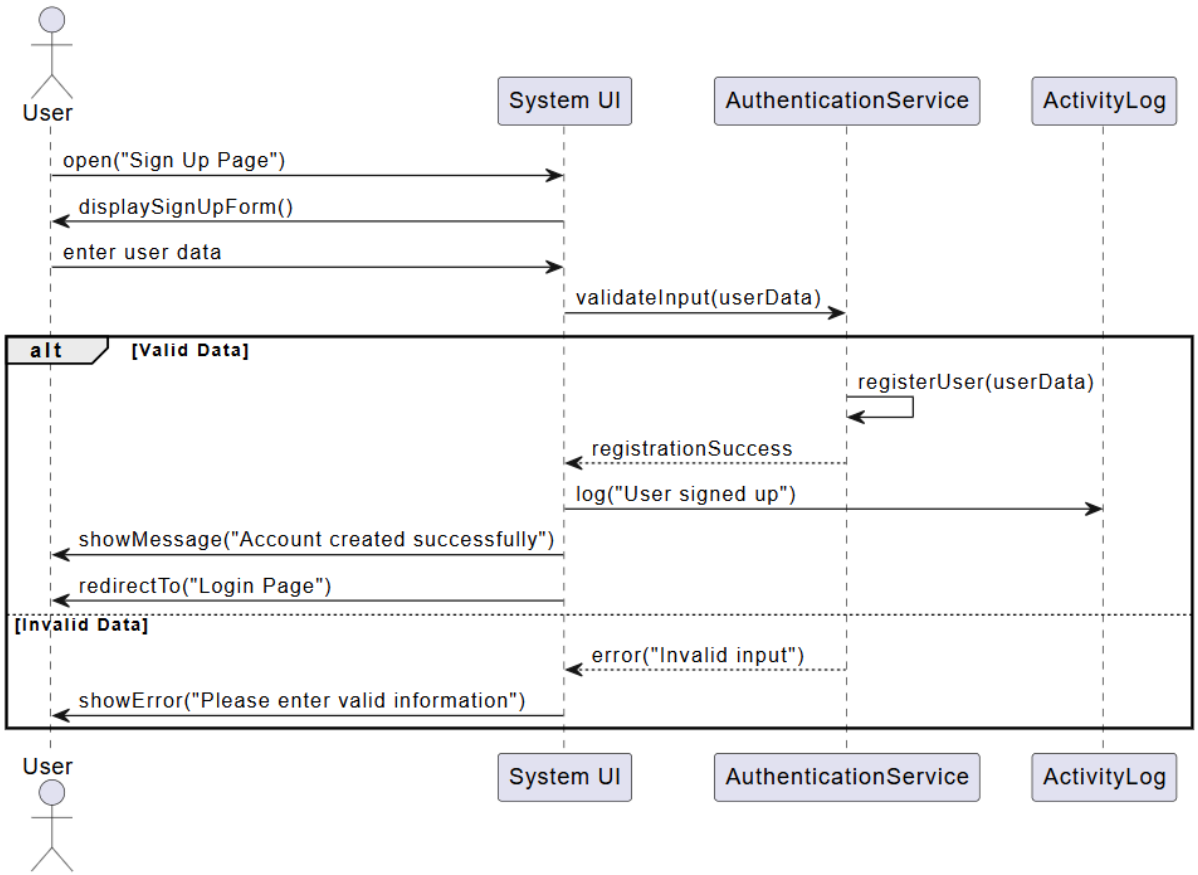
Software Design Specification

Class ID	Class Name	Description & Responsibility
14	Crypto	Entity subclass of Asset representing cryptocurrency holdings, with methods for validating Sharia compliance and viewing transaction history.
15	Gold	Entity subclass of Asset representing gold investments, including purity and karat info, and can calculate zakat eligibility.
16	BankIntegration	Boundary class that connects to user bank accounts to synchronize transactions, check balances, and retrieve account history.
17	CIBIntegration	Boundary class extending BankIntegration, providing access to Egyptian stock holdings and synchronizing CIB-specific accounts.
18	QNBIntegration	Boundary class extends BankIntegration, providing access to QNB's investment product offerings.
19	StockMarketAPI	Boundary class that interfaces with external stock market services to retrieve real-time stock prices and execute trading operations.
20	ThndrAPI	Boundary subclass of StockMarketAPI providing data specific to Egyptian stocks and user portfolios on the Thndr platform.
21	BinanceAPI	Boundary subclass of StockMarketAPI providing access to cryptocurrency prices and wallet balances on Binance.
22	Report	Control class that generates various reports (portfolio, custom), exports to Excel, and sends them via email to the user.

Software Design Specification

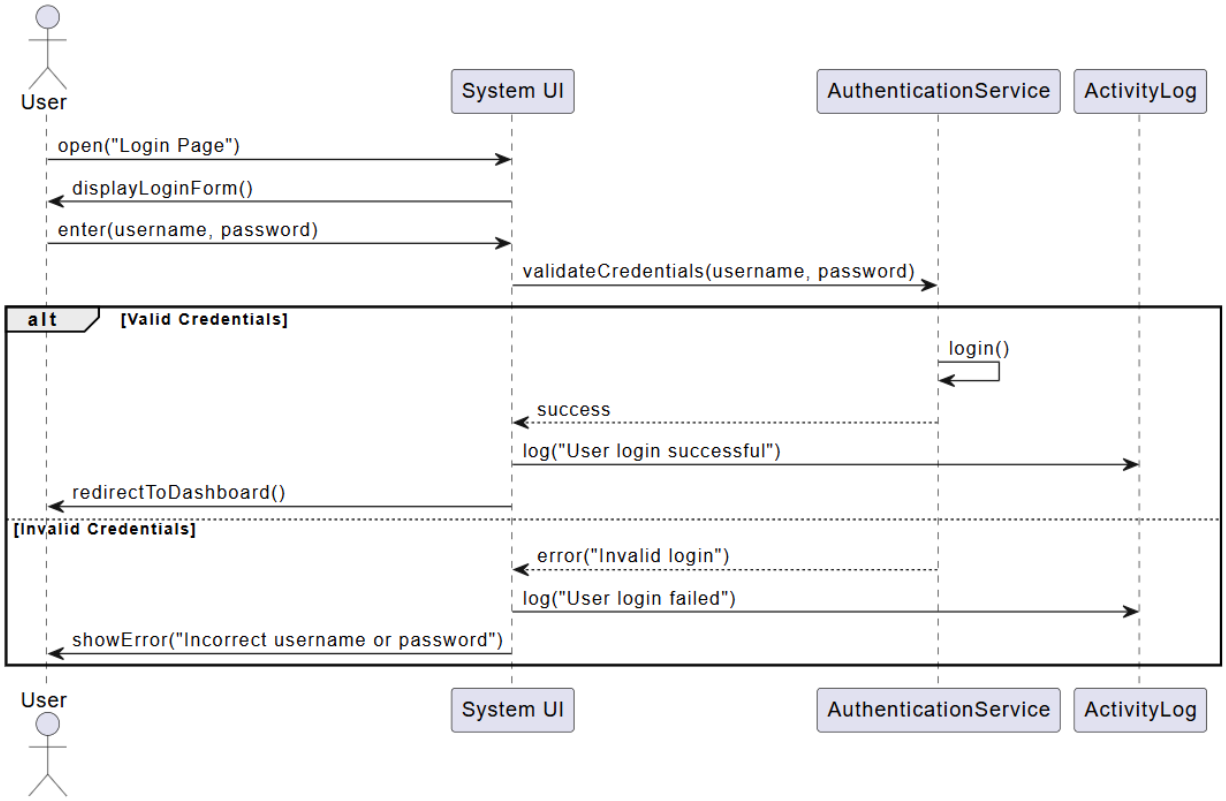
IV. Sequence diagrams

1. User Story #1:User Sign-up



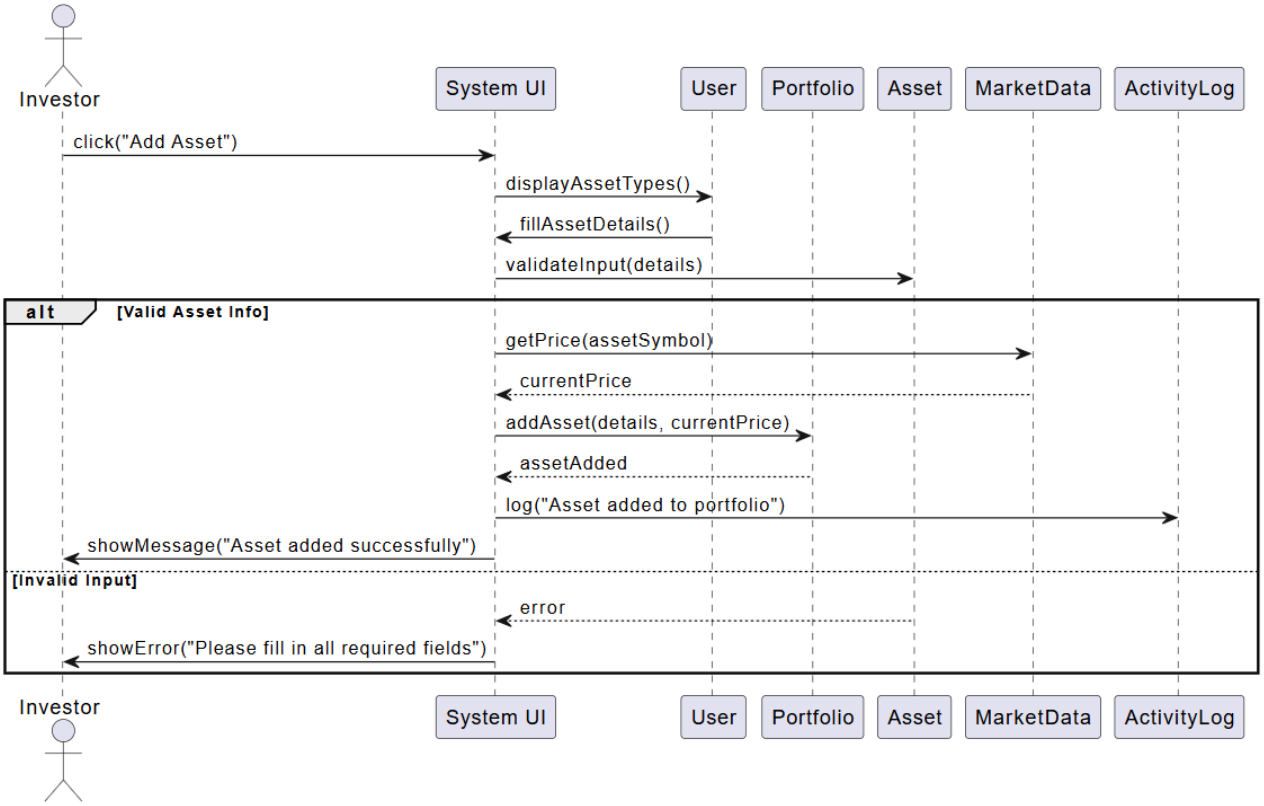
Software Design Specification

2. User Story #2:User Login



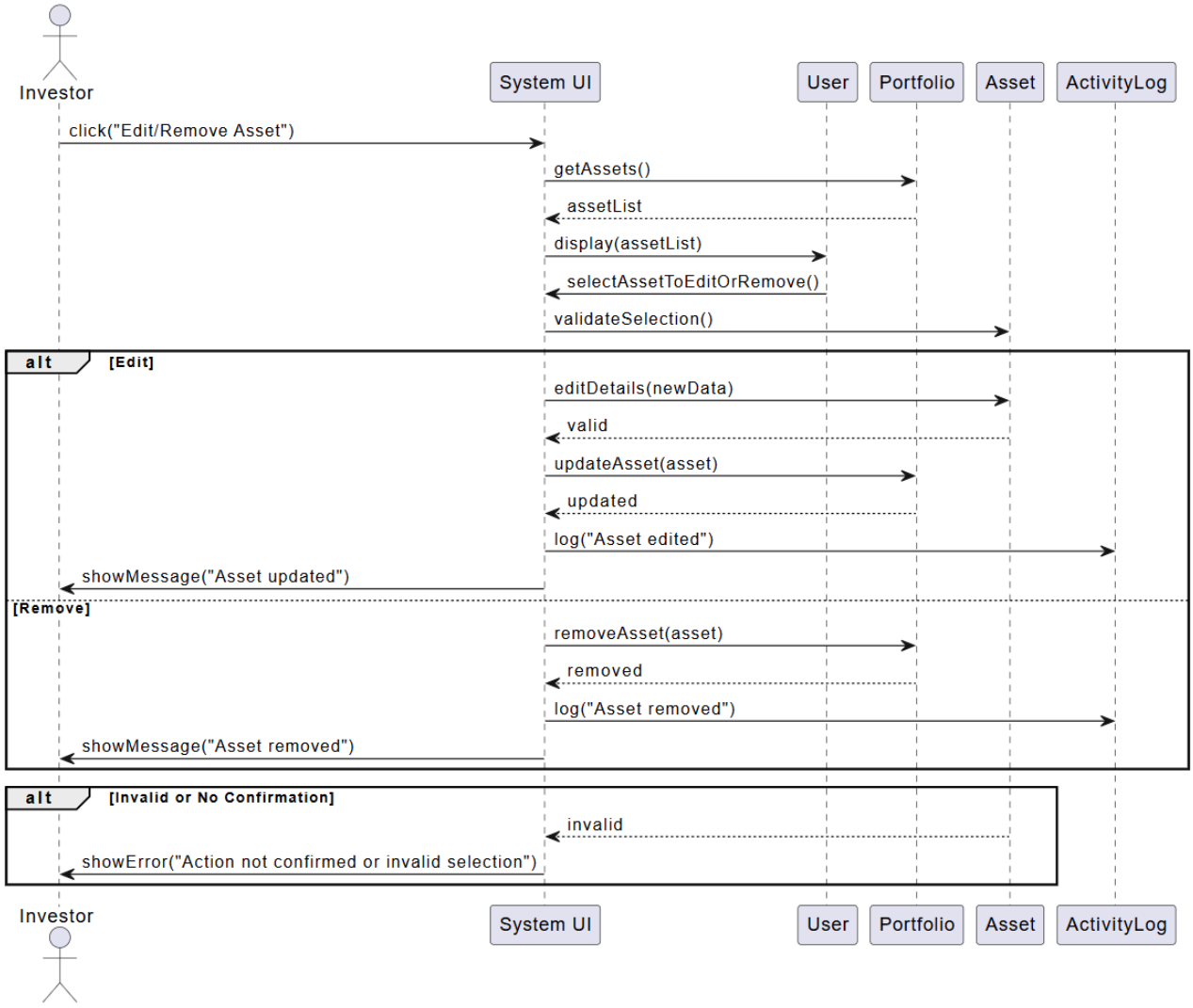
Software Design Specification

3. User Story #3: Add Assets



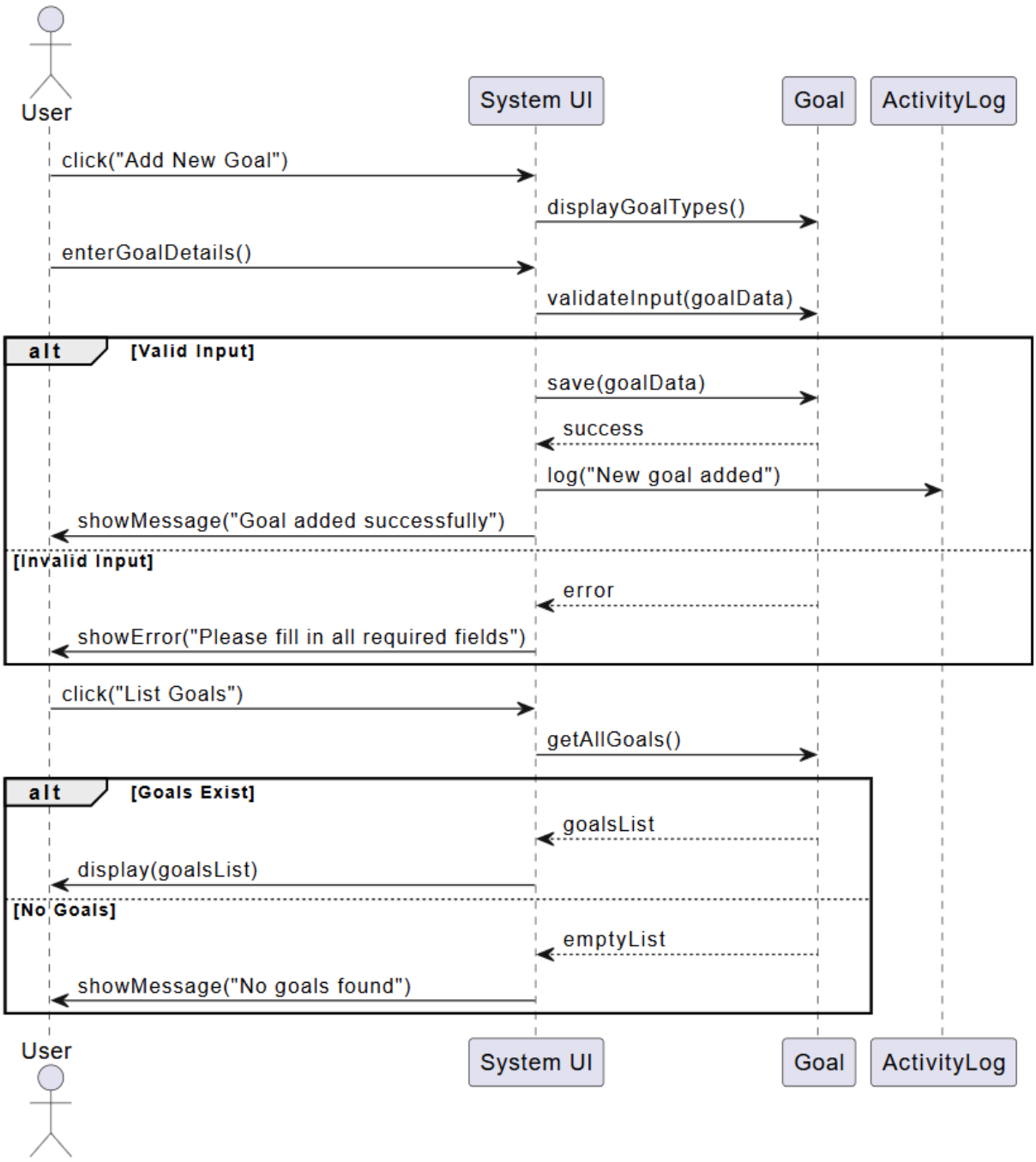
Software Design Specification

4. User Story #4: Edit/Remove Assets



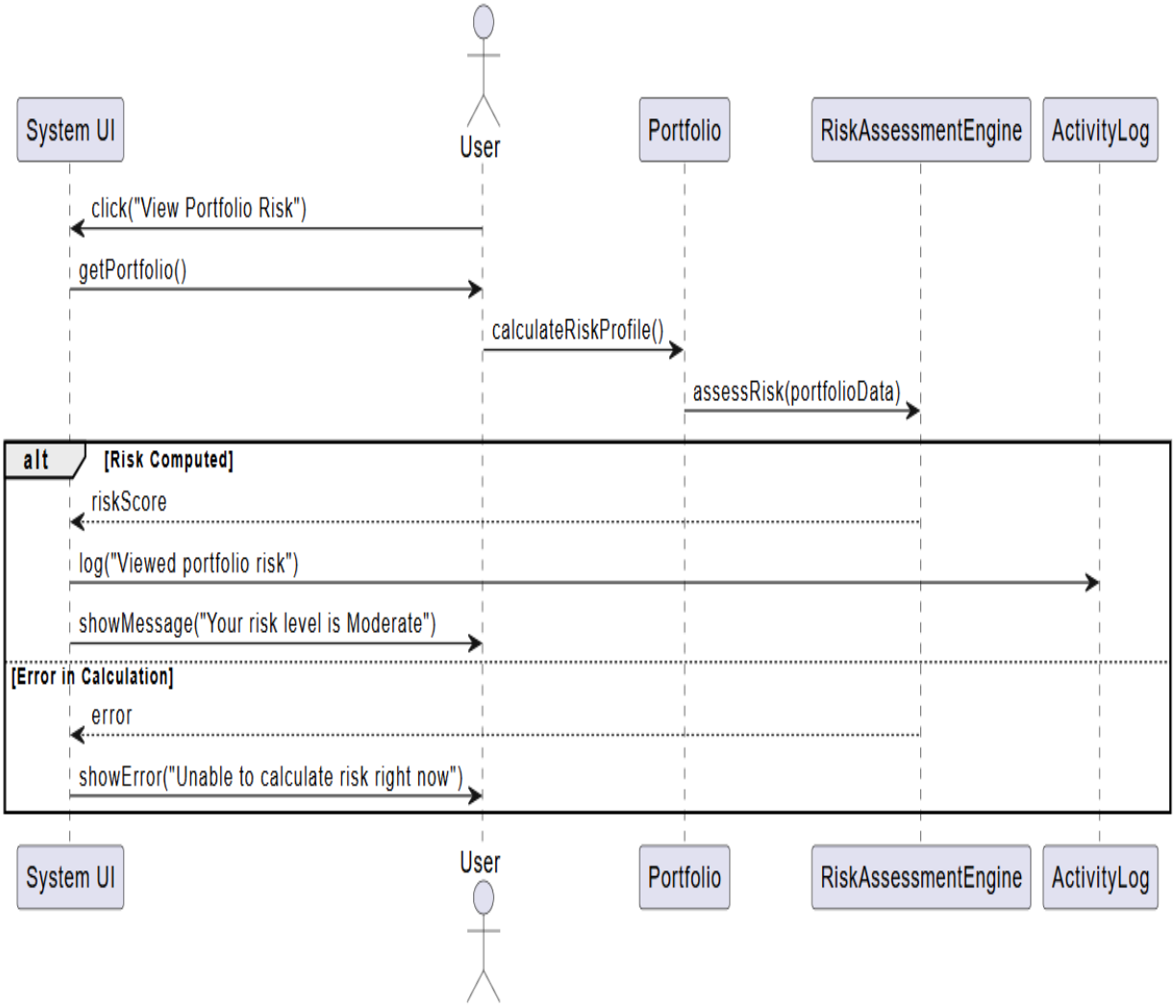
Software Design Specification

5. User Story #5: Add Financial Goals



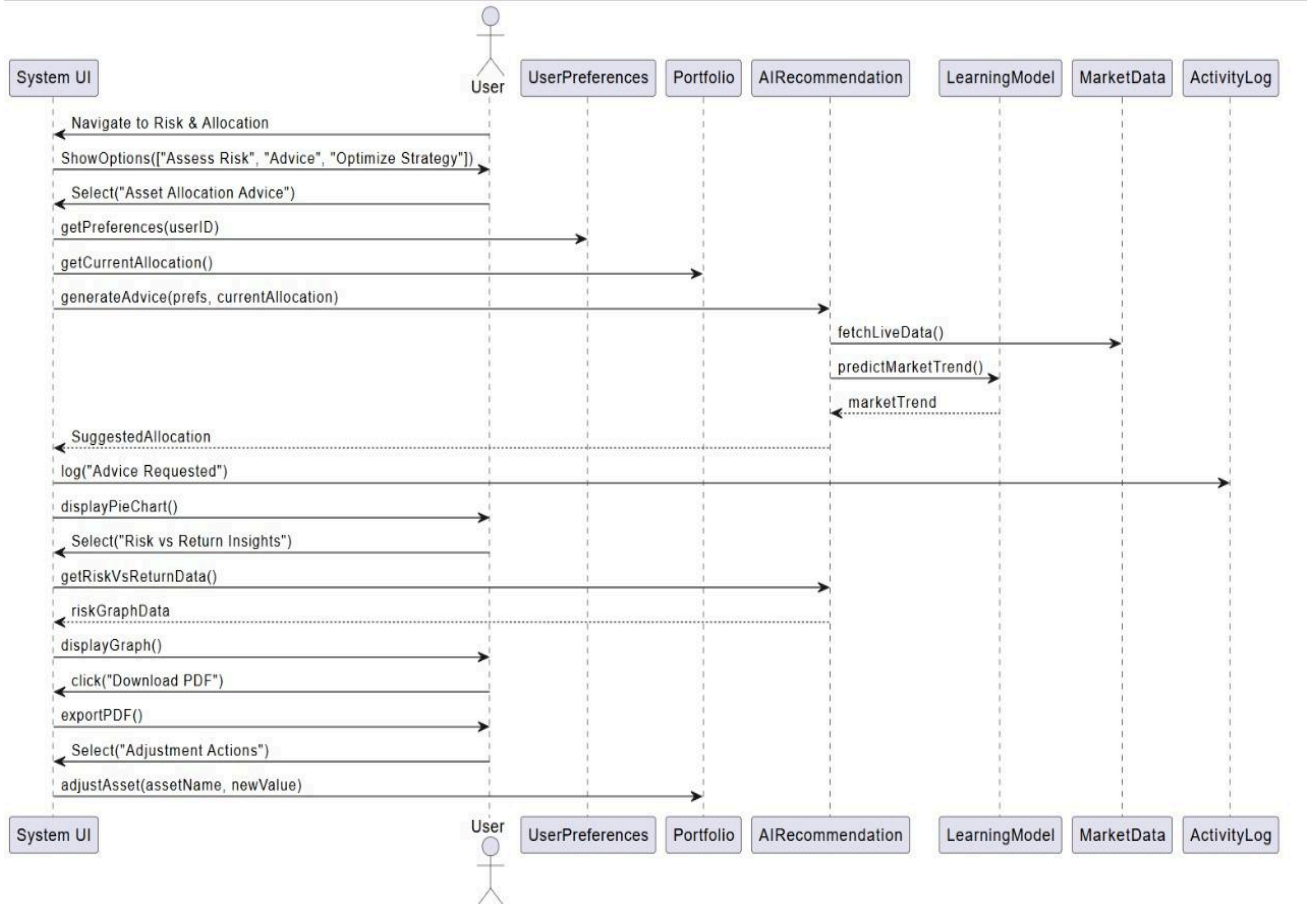
Software Design Specification

6. User Story #6:Portfolio Risk Assessment



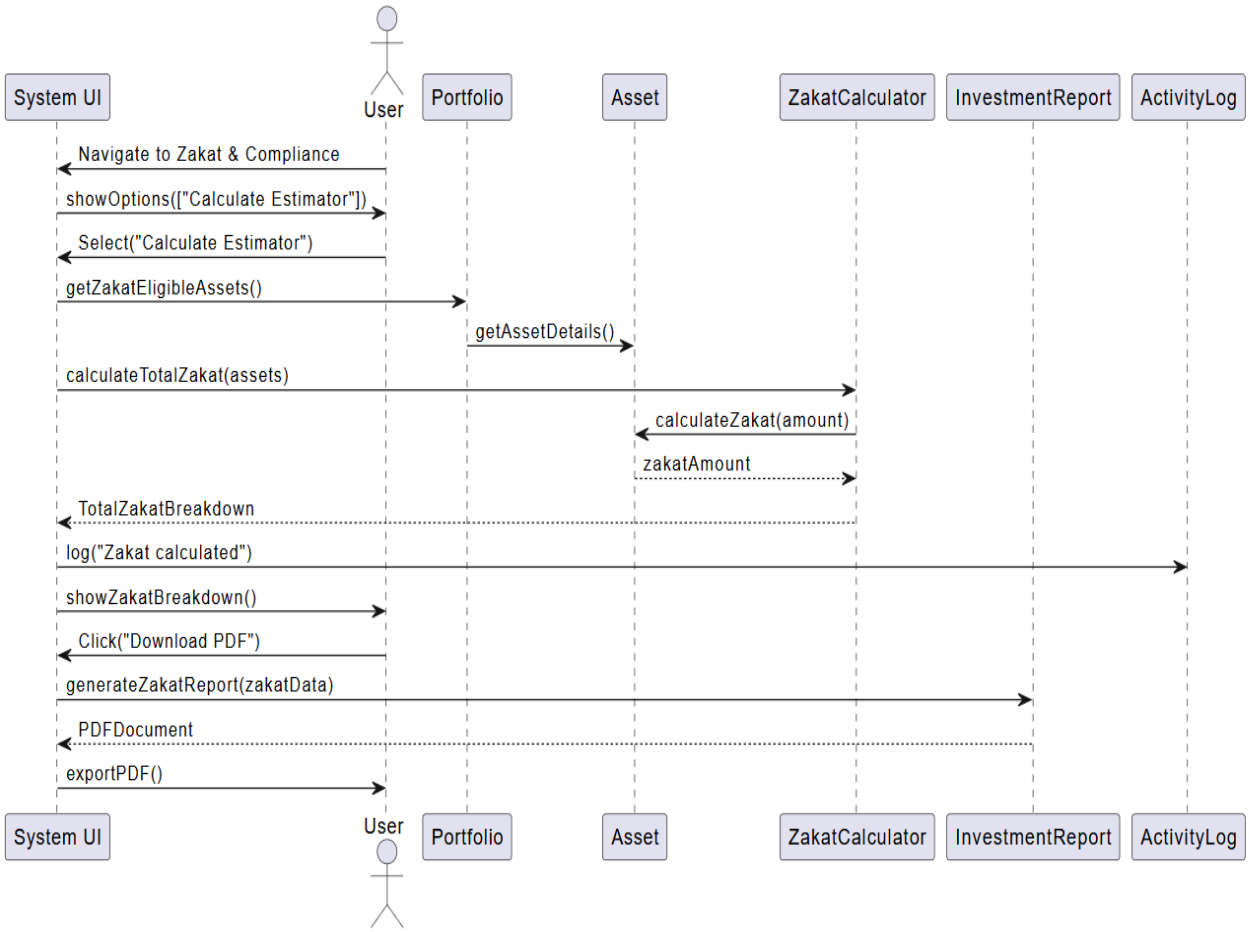
Software Design Specification

7. User Story #7: Asset Allocation Advice



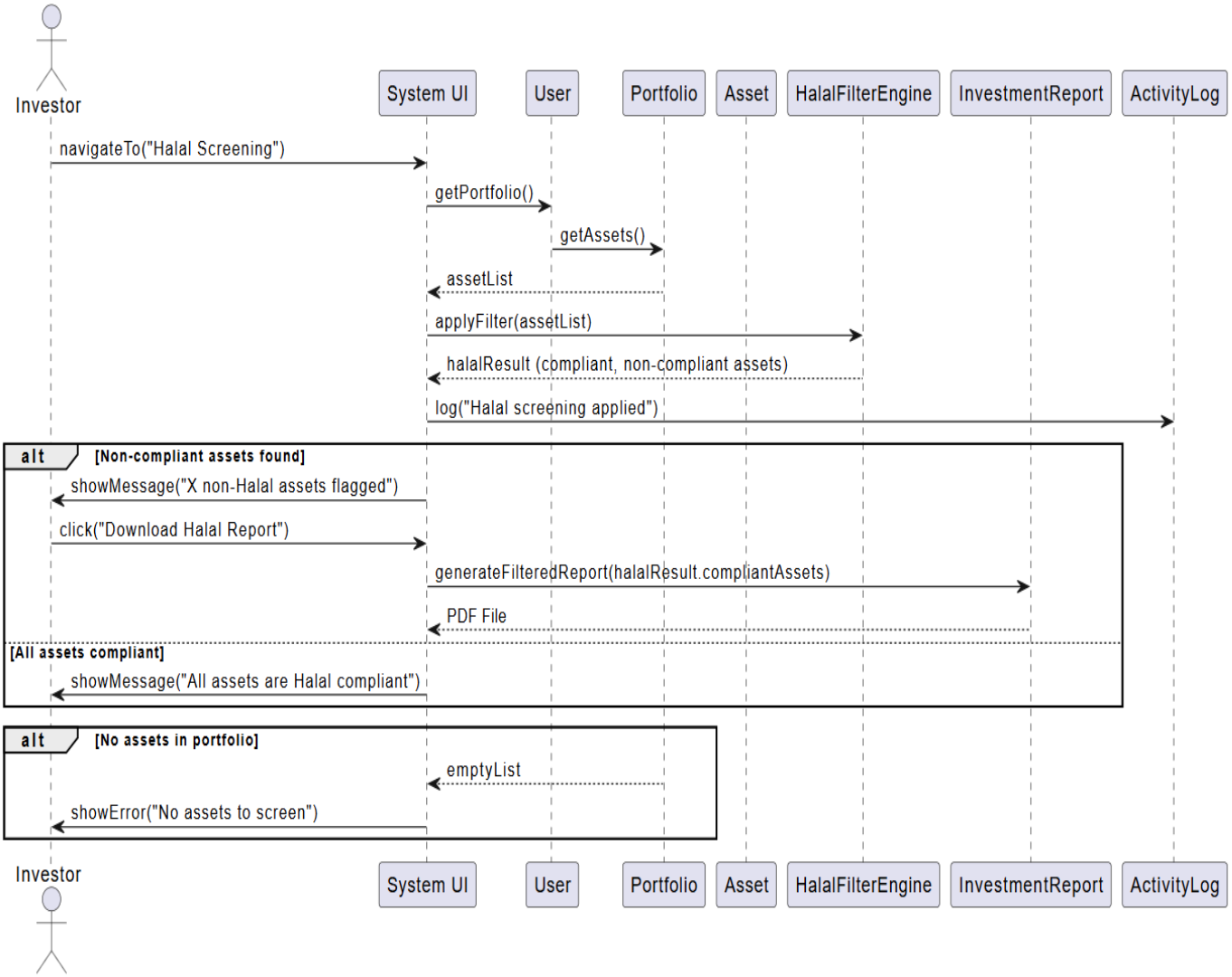
Software Design Specification

8. User Story #8: Zakat Calculation



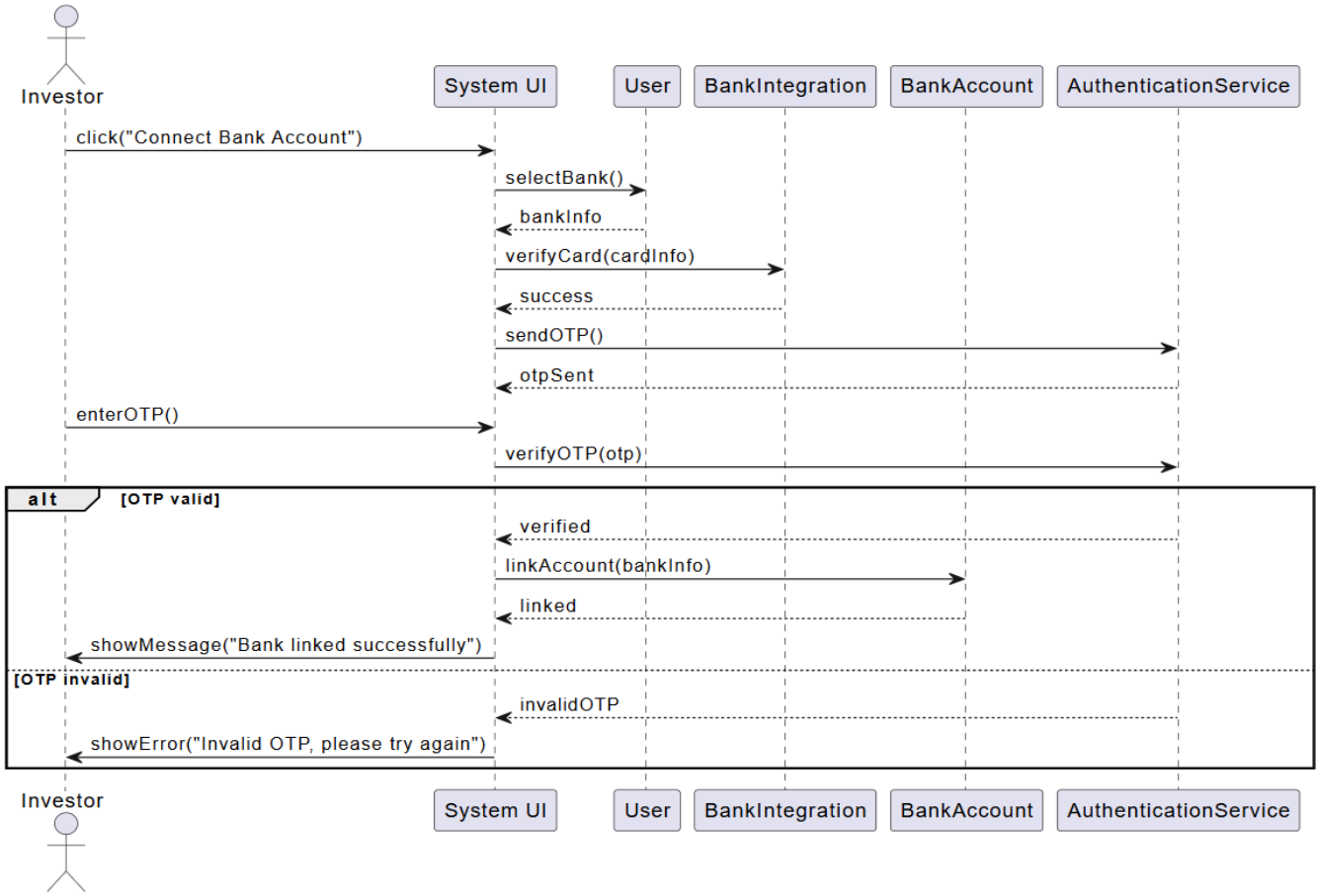
Software Design Specification

9. User Story #9: Halal Investment Screening



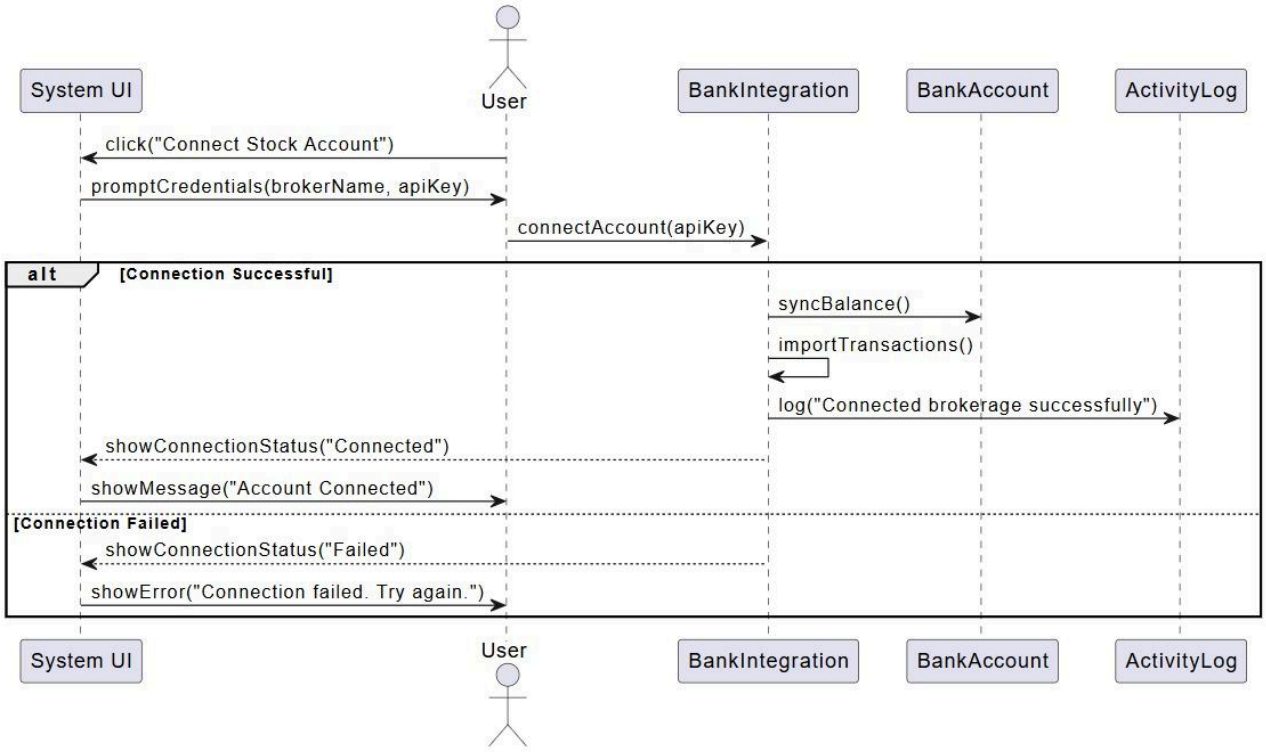
Software Design Specification

10.User Story #10: Connect Bank Account



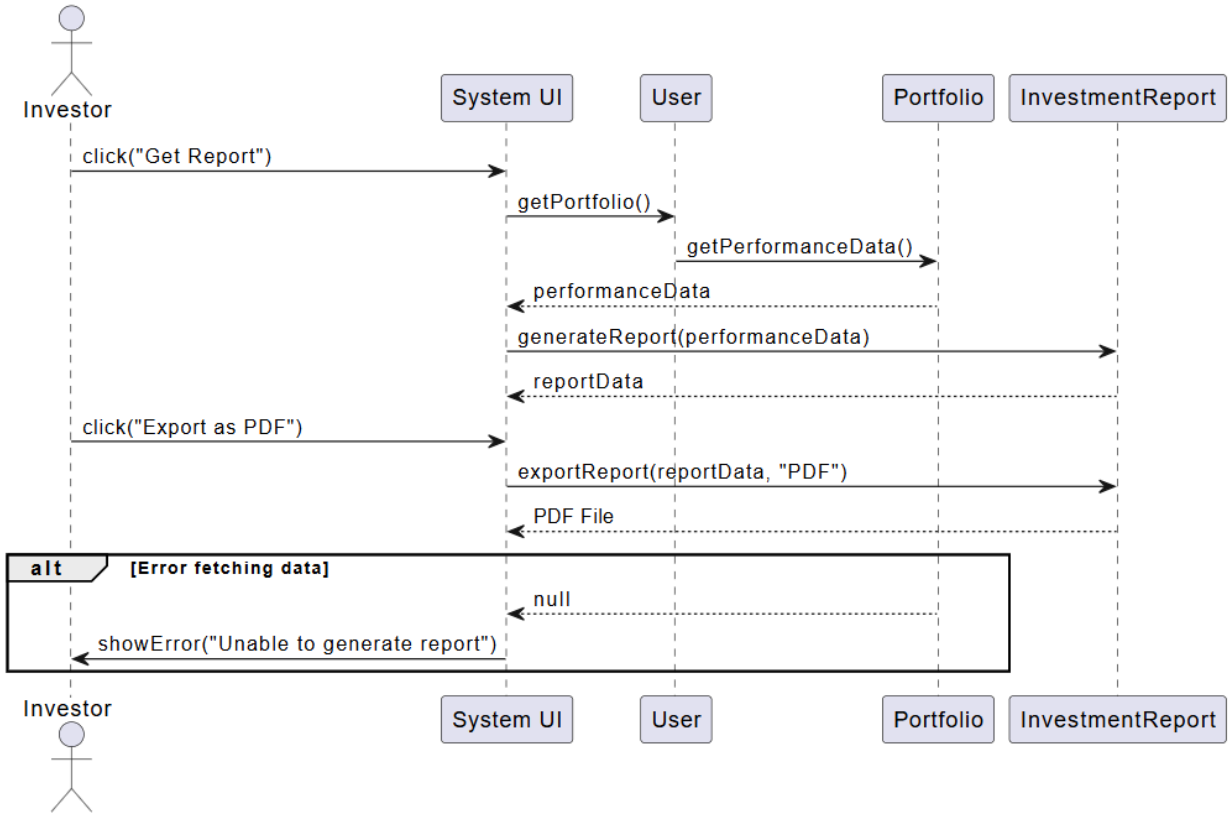
Software Design Specification

11.User Story #11: Connect Stock Account



Software Design Specification

12.User Story #12: Generate and Export Financial Reports



Software Design Specification

Class - Sequence Usage Table

Sequence Diagram	Classes Used	All Methods Used
1. **User Sign-up**	User, AuthenticationService, System UI, ActivityLog	-AuthenticationService.validateInput(), -AuthenticationService.registerUser(), -System.displaySignUpForm(), -System.showMessage(), -System.showError(), ActivityLog.log()
2. **User Login**	User, AuthenticationService, System UI, ActivityLog	-AuthenticationService.validateCredentials(), - AuthenticationService.login(), -System.displayLoginForm(), -System.redirectToDashboard(), -System.showError(), ActivityLog.log()
3. **Add Assets**	User, Portfolio, Asset, MarketData, System UI, ActivityLog	-Asset.validateInput(), -Portfolio.addAsset(), -MarketData.getPrice(), -System.displayAssetForm(), -System.showMessage(), - System.showError(), -ActivityLog.log()
4. **Edit/Remove Assets**	User, Portfolio, Asset, System UI, ActivityLog	-Portfolio.getAssets(), -Asset.editDetails(), -Portfolio.updateAsset(), -Portfolio.removeAsset(), -System.showMessage(), -System.showError(), - ActivityLog.log()
5. **Add Financial Goals**	User, Goal, System UI, ActivityLog	-Goal.validateInput(), - Goal.save(), -Goal.getAllGoals(), -System.showMessage(), -System.showError(), -ActivityLog.log()

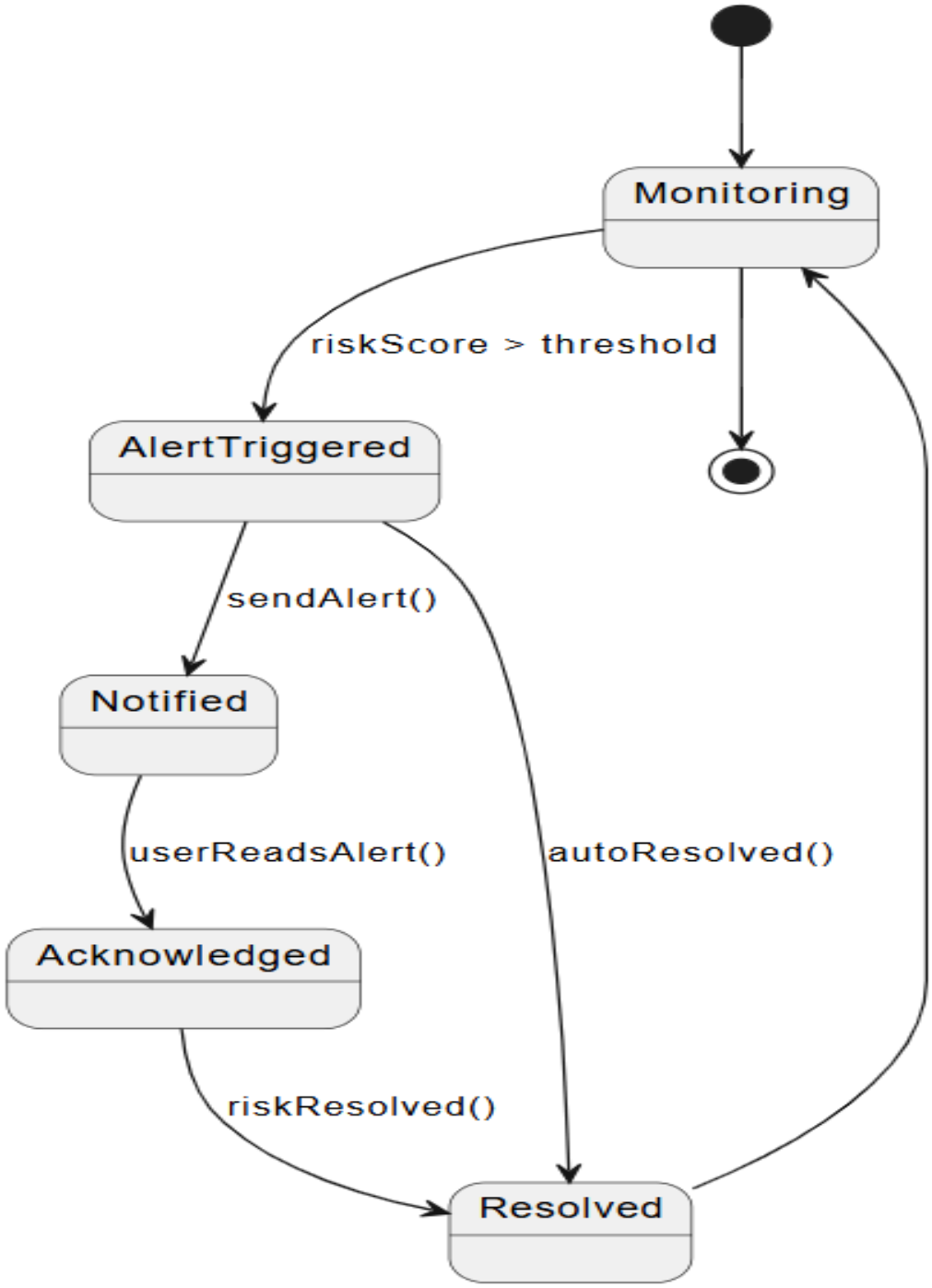
Software Design Specification

Sequence Diagram	Classes Used	All Methods Used
6. **Portfolio Risk Assessment**	User, Portfolio, RiskAssessmentEngine, System UI, ActivityLog	-User.getPortfolio(), -Portfolio.calculateRiskProfile(), -RiskAssessmentEngine.assessRisk(), -System.showMessage(), -System.showError(), -ActivityLog.log()
7. **Asset Allocation Advice**	User, UserPreferences, AIRecommendation, LearningModel, Portfolio, MarketData, System UI, ActivityLog	requestAdvice(), generateAdvice(), predictMarketTrend(), fetchLiveData(), displayPieChart(), adjustAsset(), log()
8. **Zakat Calculation**	User, Portfolio, Asset, ZakatCalculator, InvestmentReport, System UI, ActivityLog	getZakatEligibleAssets(), calculateZakat(), calculateTotalZakat(), generateZakatReport(), exportPDF(), log()
9. **Halal Investment Screening**	User, Portfolio, Asset, HalalFilterEngine, System UI, InvestmentReport, ActivityLog	-User.getPortfolio(), -Portfolio.getAssets(), -HalalFilterEngine.applyFilter(), -ActivityLog.log(), -System.showMessage(), -System.showError(), -InvestmentReport.generateFilteredReport()
10. **Connect Bank Account**	User, BankIntegration, BankAccount, AuthenticationService, System (UI)	-User.selectBank(), -BankIntegration.verifyCard(), -AuthenticationService.sendOTP(), -AuthenticationService.verifyOTP(), -BankAccount.linkAccount(), -System.showMessage(), -System.showError()
11. **Connect Stock Account**	User, BankIntegration, BankAccount, System UI, ActivityLog	User.initiateConnection(), BankIntegration.connectAccount(), BankIntegration.importTransactions(), BankAccount.syncBalance(), System.showConnectionStatus(), ActivityLog.log()
12. **Export Financial Reports**	User, Portfolio, InvestmentReport, System (UI)	-User.getPortfolio(), -Portfolio.getPerformanceData(), -InvestmentReport.generateReport(), -System.exportReport(), -System.showError()

Software Design Specification

V. State Diagram

Risk Management & Alerts :



Software Design Specification

VI. SOLID Principles

1. Single Responsibility Principle (SRP)

Each class in our system is designed to have one specific responsibility.
Example:

- **The `User` class is responsible only for managing user-related data such as profile details and login credentials.**

- **Authentication logic is handled separately by the `Authentication` class, and encryption/logging tasks are managed by `SecurityManager`.**
  This separation ensures each class has one reason to change, enhancing maintainability.

2. Open/Closed Principle (OCP)

Our system is structured to allow extension without modifying existing code.
Example:

- **The `Asset` class is an abstract entity for all investment types.**

- **New asset types like `Stock`, `Gold`, `Crypto`, and `RealEstate` extend the `Asset` class without changing its core logic.**
  This design allows us to add new asset types in the future (e.g., `Bond`) without altering existing code.

3. Liskov Substitution Principle (LSP)

We designed our inheritance hierarchy so that subclasses can replace their parent class without issues.
Example:

- **All subclasses of `Asset` (`Stock`, `Crypto`, `Gold`, `RealEstate`) implement the methods defined in the abstract `Asset` class.**

- **This allows any `Asset` object in a `Portfolio` to be treated uniformly, regardless of its specific type.**
  This ensures polymorphism works correctly and safely throughout the system.

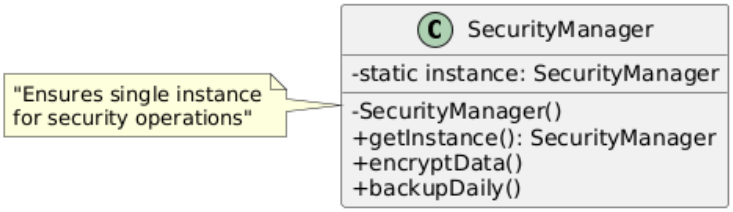
Software Design Specification

VII. Design Patterns

1. Singleton Pattern

Where used:

- `SecurityManager` class in authentication flow
- `DatabaseConnection` pool for all database operations



Description:
The Singleton pattern ensures a class has only one instance while providing a global access point to it. In our system, this is critical for centralized control of security operations and database connections.

Why used:

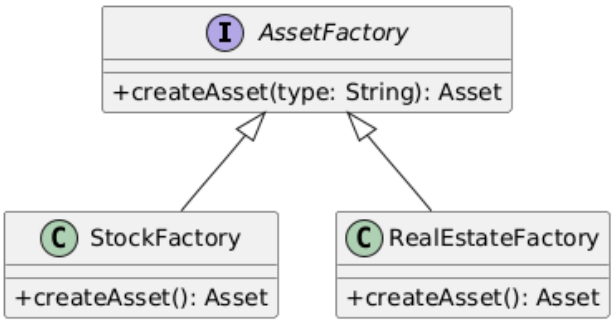
- Security operations (encryption, OTP validation) must be consistent across the entire application
- Database connections are expensive to create - pooling them improves performance
- Matches **FR01 (Secure Login)** and **Security NFRs** requiring strict access control

Benefit:

- Prevents multiple instances that could cause security conflicts
- Reduces memory overhead by reusing single instances
- Provides thread-safe access to shared resources

Software Design Specification

2. Factory Method Pattern



Where used:

- `AssetFactory` in portfolio management module
- `ReportFactory` in reporting module

Description:
Defines an interface for creating objects but lets subclasses decide which class to instantiate. Used for creating different asset types and report formats.

Why used:

- System handles multiple asset types (stocks, real estate, etc.) with different creation logic
- Need to add new asset types without modifying existing code
- Supports **FR02 (Multi-asset Tracking)** and **FR11 (Asset Management)**

Benefit:

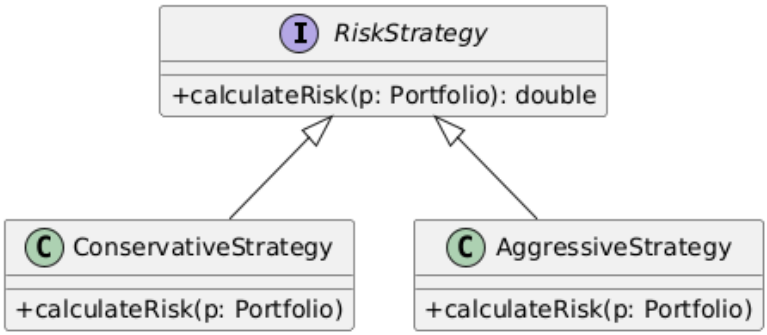
- Decouples asset creation from usage
- Simplifies adding new asset types (e.g., future "Bonds" support)
- Centralizes asset initialization logic

Software Design Specification

3. Strategy Pattern

Where used:

- Risk assessment in `PortfolioAnalysisService`
- Zakat calculation in `ZakatEngine`



Description:
Defines a family of algorithms, encapsulates each one, and makes them interchangeable. Lets the algorithm vary independently from clients.

Why used:

- Users need different risk calculation methods (conservative/aggressive)
- Zakat rules vary by asset type (gold vs crypto)
- Required by **FR05 (Risk Assessment)** and **FR14 (Zakat Calculation)**

Benefit:

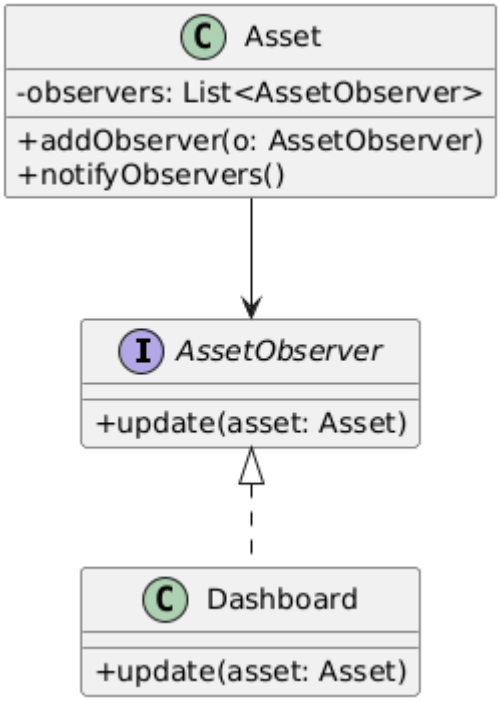
- Easy to add new calculation algorithms
- Clean separation between business logic and strategies
- Enables dynamic strategy switching at runtime

Software Design Specification

4. Observer Pattern

Where used:

- Real-time price updates in `AssetPriceService`
- Portfolio value changes in `PortfolioTracker`



Description:
Defines a one-to-many dependency between objects so when one object changes state, all its dependents are notified automatically.

Why used:

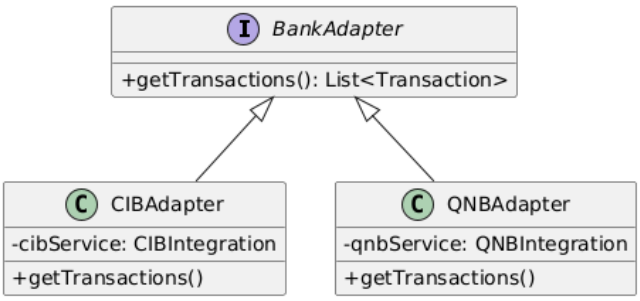
- Dashboard must reflect real-time asset value changes
- Multiple UI components need simultaneous updates
- Critical for **FR10 (Real-time Dashboard)** and **US11 (Portfolio Tracking)**

Benefit:

- Eliminates polling mechanism
- Loose coupling between assets and UI
- Supports multiple observer types (email alerts, push notifications)

Software Design Specification

5. Adapter Pattern



Where used:

- Bank integrations (`CIBAdapter`, `QNBAdapter`)
- Stock API integrations (`ThndrAPIAdapter`)

Description:
Converts the interface of a class into another interface clients expect. Let classes work together that couldn't otherwise due to incompatible interfaces.

Why used:

- Each bank has different API protocols
- Need unified interface for transaction processing
- Required by **FR08 (Bank Integration)** and **FR16 (API Connectivity)**

Benefit:

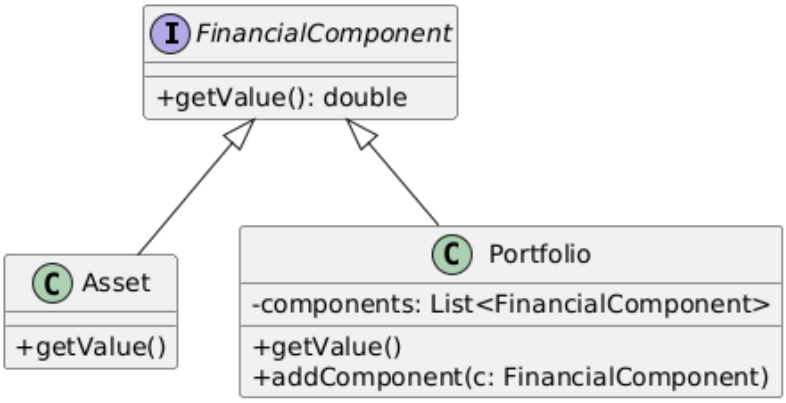
- Simplifies adding new financial institutions
- Isolates third-party API changes
- Provides consistent interface to core system

Software Design Specification

6. Composite Pattern

Where used:

- Portfolio management in `PortfolioService`
- Net worth calculation in `NetWorthCalculator`



Description:
Composes objects into tree structures to represent part-whole hierarchies. Lets clients treat individual objects and compositions uniformly.

Why used:

- Portfolios contain both assets and sub-portfolios
- Need recursive calculations for net worth
- Required by **FR03 (Net Worth Calculation)**

Benefit:

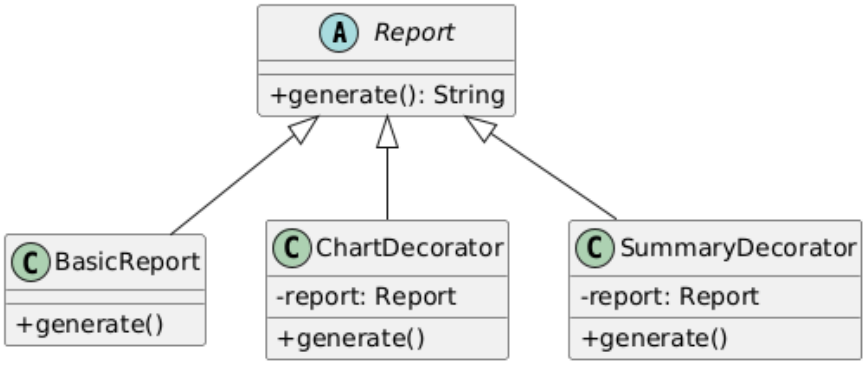
- Uniform treatment of single assets and portfolios
- Simplifies complex hierarchical operations
- Enables recursive calculations like total value

Software Design Specification

7. Decorator Pattern

Where used:

- Report generation in `ReportEngine`
- Dashboard visualizations



Description:
Attaches additional responsibilities to an object dynamically. Provides a flexible alternative to subclassing for extending functionality.

Why used:

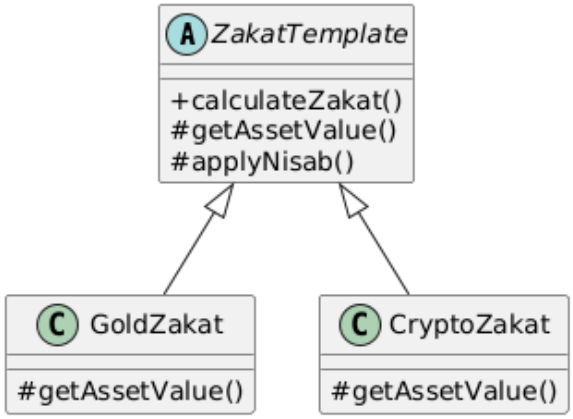
- Reports need different combinations of features
- Must add visualizations without changing core logic
- Required by **FR15 (Custom Reports)**

Benefit:

- Dynamic feature addition
- Avoids explosion of subclasses
- Preserves Single Responsibility Principle

Software Design Specification

8. Template Method Pattern



Where used:

- Zakat calculation in `ZakatService`
- Compliance reporting

Description:
Defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. Let subclasses redefine certain steps without changing the algorithm's structure.

Why used:

- Zakat calculation steps are consistent but asset-specific
- Need to enforce Sharia compliance rules
- Required by **FR14 (Zakat Calculation)**

Benefit:

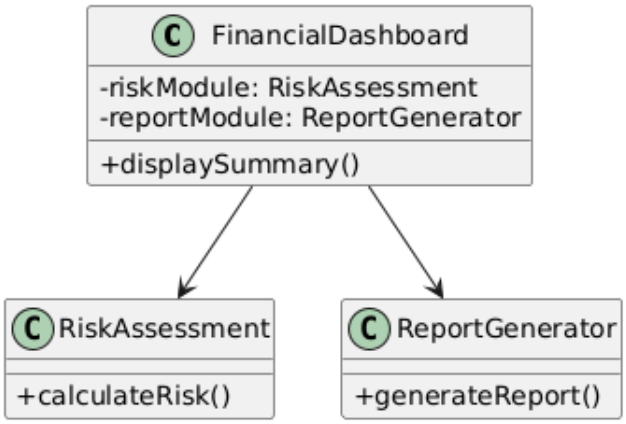
- Code reuse for common algorithm steps
- Ensures compliance with Islamic finance rules
- Easy to add new asset types

Software Design Specification

9. Facade Pattern

Where used:

- Dashboard interface (`DashboardController`)
- API gateway service



Description:
Provides a unified interface to a set of interfaces in a subsystem. Defines a higher-level interface that makes the subsystem easier to use.

Why used:

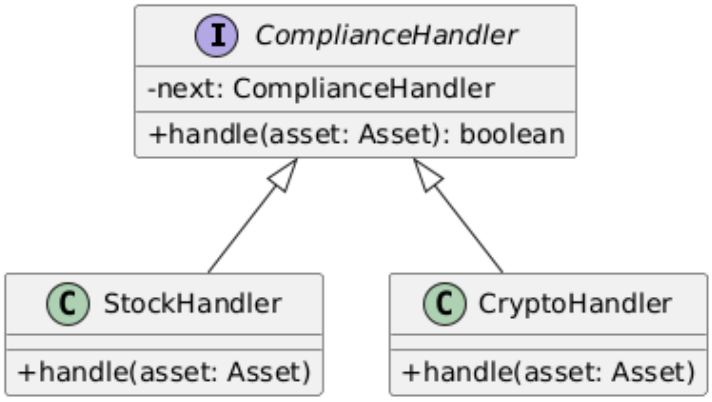
- Complex financial operations need simplified access
- Hide subsystem complexity from end-users
- Required by **FR10 (Dashboard UI)**

Benefit:

- Simplified client interface
- Reduces dependencies between subsystems
- Improves usability (**NFR Usability**)

Software Design Specification

10. Chain of Responsibility



Where used:

- Sharia compliance checking in `ComplianceEngine`
- Asset validation pipeline

Description:
Avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chains the receiving objects and passes the request along the chain.

Why used:

- Different assets require different compliance checks
- Need modular verification pipeline
- Required by **FR06 (Halal Filtering)**

Benefit:

- Easy to add new compliance rules
- Single responsibility per handler
- Dynamic chain configuration

Software Design Specification

Tools

- **Plant UML**
- **lucidChart**

Ownership Report

Item	Owners
Menna Talla Gamal	Class Diagram & sequence diagram 1, 2 & design patterns
Israa Abdelhaq	Architecture Diagram & sequence diagram 3, 4 ,7,8,9,10,11,12
Mahmoud Hosny	Sequence diagram 5, 6 & state diagram & solid