Cairo University

Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

Tharwa

Software Design Specifications

Version 1.5

Name	ID	Email
Fatema El-Zhraa Ahmed Mohamed El-Fiky	20230280	fatmaelfeky922@gmail.com
Nagham Wael Mohamed	20231189	naghamw63@gmail.com
Aly El-Deen Yasser Ali	20231109	ali.el.badry.747@gmail.com

April of 2025

CS251: Optimum
Project: Tharwa

Software Design Specification

Contents

CS251: Optimum
Project: Tharwa

Software Design Specification

Team

ID	Name	Email	Mobile
20230280	Fatema El-Zhraa Ahmed Mohamed El-Fiky	fatmaelfeky922@gmail.com	01221990828
20231189	Nagham Wael Mohamed	naghamw63@gmail.com	01007600773
20231109	Aly El-Deen Yasser Ali	ali.el.badry.747@gmail.com	01286964627

Document Purpose and Audience

Purpose.

This SDS defines the design and structure of the Personal Investment Management Software. The software aims to help users track, analyze, and optimize their investment portfolios, ensuring informed financial decisions. It serves as a reference for consistent development, testing, and future improvements.

Audience.

1. **Development Team**
    - Software Engineers/Developers: Backend and frontend developers who will implement the system architecture and codebase based on the design specifications.
    - Technical Leads: Senior developers who will oversee implementation and ensure alignment with design decisions.
    - QA Engineers: Testers who will use the design documentation to create test cases and verify system behavior.
2. **System Architects**
    - Professionals responsible for reviewing and approving the high-level system design and ensuring it meets all technical requirements.
3. **Project Stakeholders**
    - *Product Owners*: Non-technical stakeholders who need to understand how design decisions fulfill business requirements.

- *Islamic Finance Experts*: Domain specialists who will verify Sharia-compliance aspects of the design.
- *Banking Integration Partners*: Technical representatives from partner institutions (e.g., CIB) who need to understand integration points.

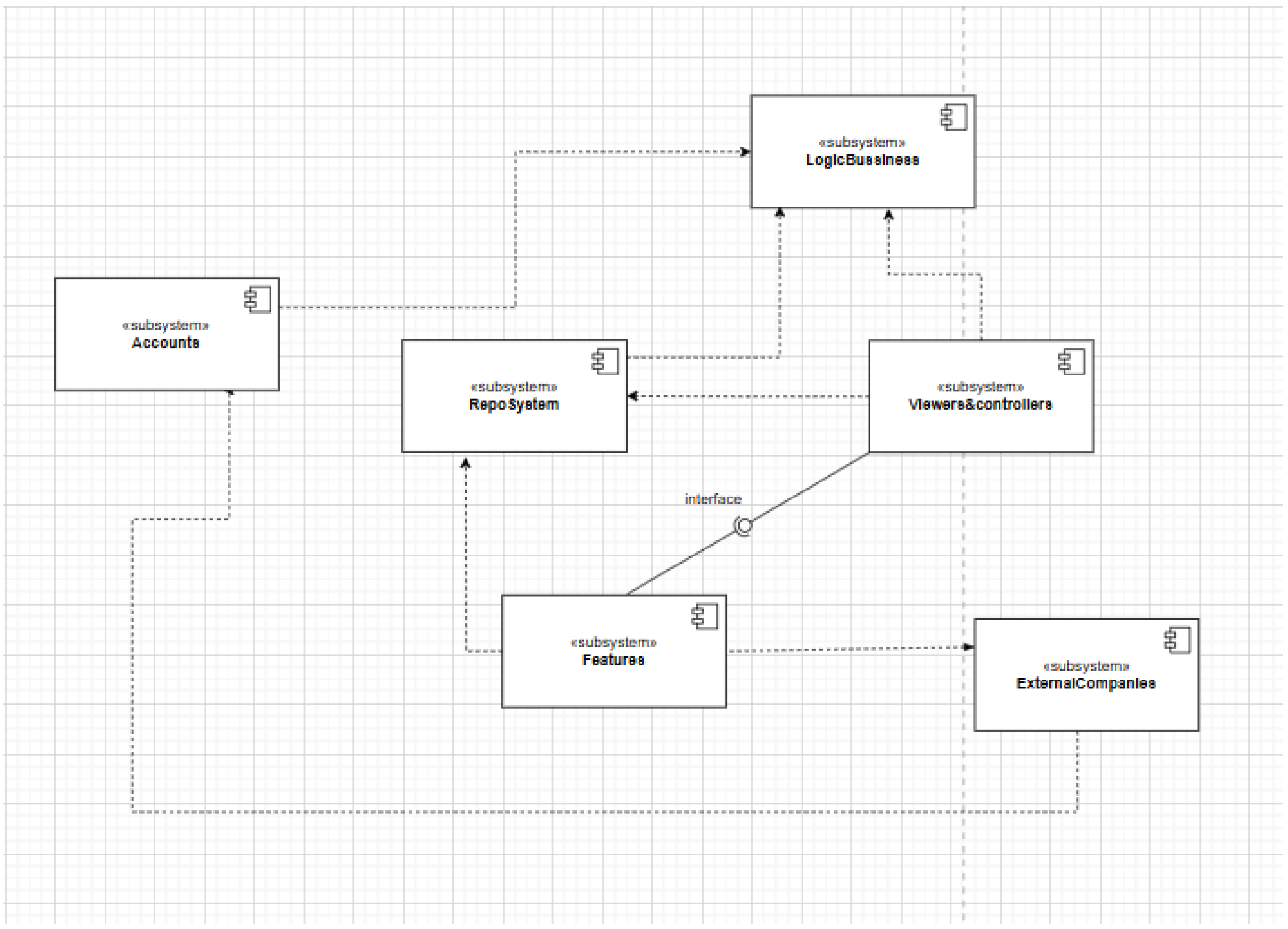
4. **Maintenance Team**
    - Future developers who will maintain, update, or extend the system and need comprehensive design documentation.

CS251: Optimum
Project: Tharwa

Software Design Specification

System Models

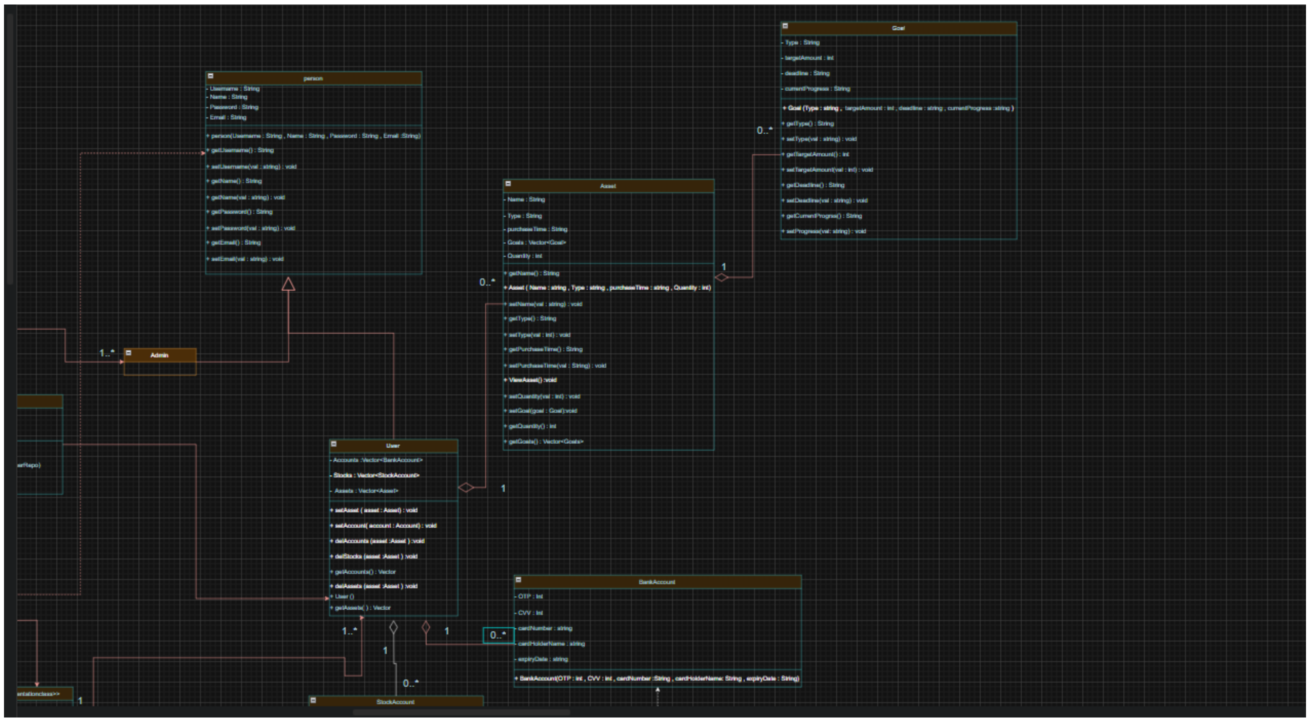
I. Architecture Diagram


«subsystem» LogicBussiness
«subsystem» Accounts
«subsystem» RepoSystem
«subsystem» Viewers&controllers
interface
«subsystem» Features
«subsystem» ExternalCompanies

CS251: Optimum
Project: Tharwa

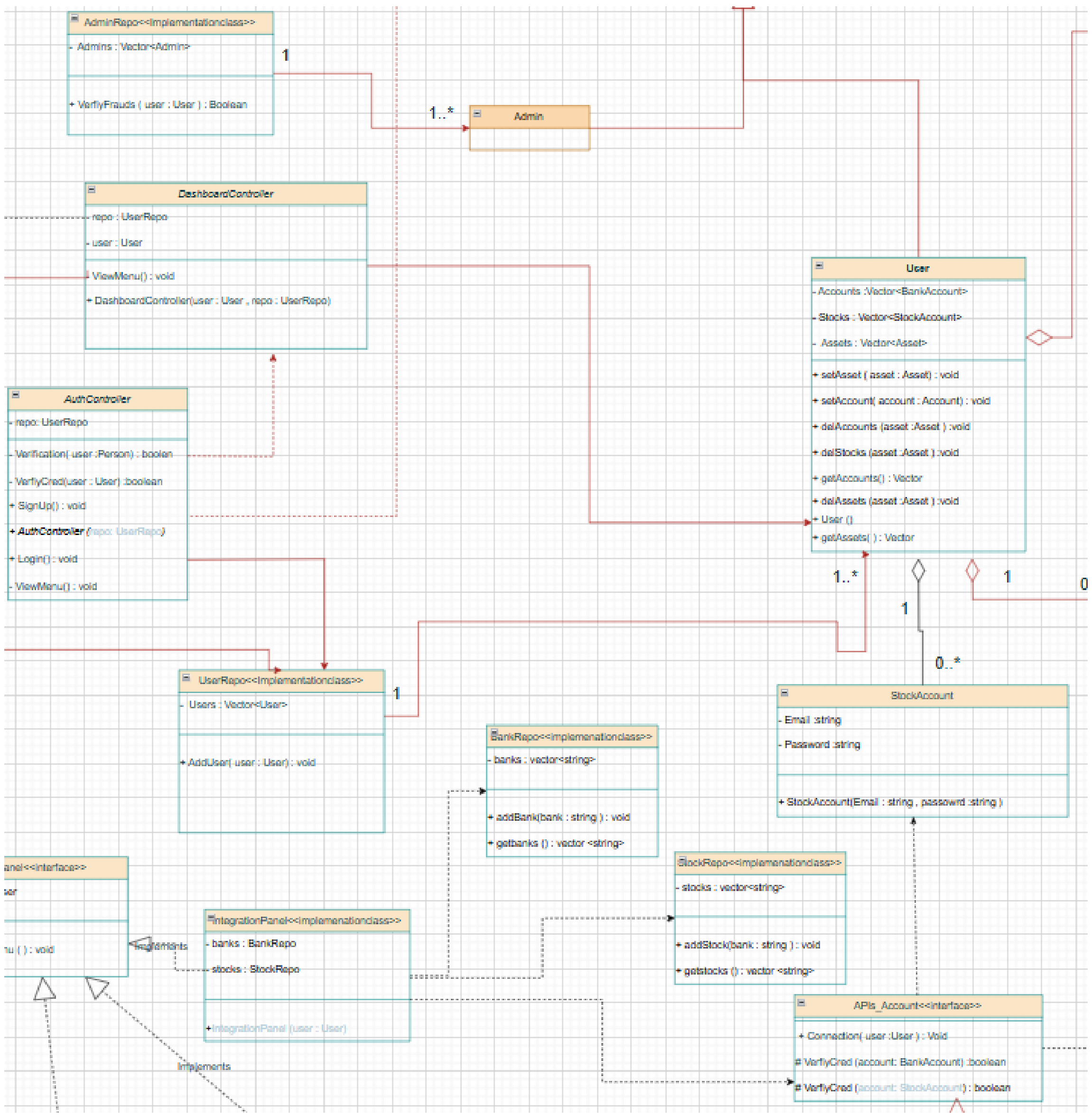
Software Design Specification

II. Class Diagram(s)


person
- Username : String
- Name : String
- Password : String
- Email : String
+ person(Username : String , Name : String , Password : String , Email :String)
+ getUsername() : String
+ setUsername(val : string) : void
+ getName() : String
+ getName(val : string) : void
+ getPassword() : String
+ setPassword(val : string) : void
+ getEmail() : String
+ setEmail(val : string) : void
Admin
1..*
Asset
- Name : String
- Type : String
- purchaseTime : String
- Goals : Vector<Goal>
- Quantity : int
+ getName() : String
+ Asset (Name : string , Type : string , purchaseTime : string , Quantity : int)
+ setName(val : string) : void
+ getType() : String
+ setType(val : int) : void
+ getPurchaseTime() : String
+ setPurchaseTime(val : String) : void
+ ViewAsset() :void
+ setQuantity(val : int) : void
+ setGoal(goal : Goal):void
+ getQuantity() : int
+ getGoals() : Vector<Goals>
0..*
1
Goal
- Type : String
- targetAmount : int
- deadline : String
- currentProgress : String
+ Goal (Type : string , targetAmount : int , deadline : string , currentProgress :string)
+ getType() : String
+ setType(val : string) : void
+ getTargetAmount() : int
+ setTargetAmount(val : int) : void
+ getDeadline() : String
+ setDeadline(val : string) : void
+ getCurrentProgrse() : String
+ setProgress(val: string) : void
0..*
User
- Accounts :Vector<BankAccount>
- Stocks : Vector<StockAccount>
- Assets : Vector<Asset>
+ setAsset (asset : Asset) : void
+ setAccount(account : Account) : void
+ delAccounts (asset :Asset) :void
+ delStocks (asset :Asset) :void
+ getAccounts() : Vector
+ delAssets (asset :Asset) :void
+ User ()
+ getAssets( ) : Vector
1
1..*
1
1
0..*
0..*
BankAccount
- OTP : int
- CVV : int
- cardNumber : string
- cardHolderName : string
- expiryDate : string
+ BankAccount(OTP : int , CVV : int , cardNumber :String , cardHolderName: String , expiryDate : String)
StockAccount
1

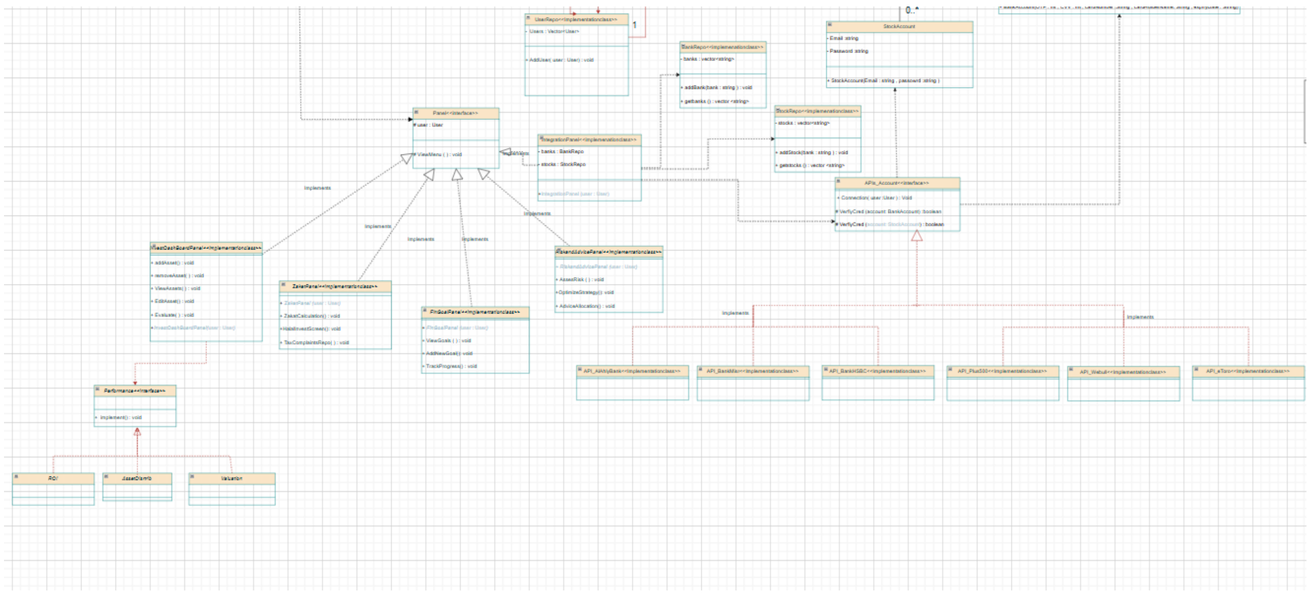
CS251: Optimum
Project: Tharwa

Software Design Specification


AdminRepo<<Implementationclass>>
- Admins : Vector<Admin>
+ VerifyFrauds (user : User) : Boolean
1
1..*
Admin
DashboardController
- repo : UserRepo
- user : User
+ ViewMenu() : void
+ DashboardController(user : User , repo : UserRepo)
User
- Accounts :Vector<BankAccount>
- Stocks : Vector<StockAccount>
- Assets : Vector<Asset>
+ setAsset (asset : Asset) : void
+ setAccount(account : Account) : void
+ delAccounts (asset :Asset) :void
+ delStocks (asset :Asset) :void
+ getAccounts() : Vector
+ delAssets (asset :Asset) :void
+ User ()
+ getAssets( ) : Vector
AuthController
- repo: UserRepo
- Verification(user :Person) : boolean
- VerifyCred(user : User) :boolean
+ SignUp() : void
+ AuthController (repo: UserRepo)
+ Login() : void
- ViewMenu() : void
1..*
1
1
0..*
0
UserRepo<<Implementationclass>>
- Users : Vector<User>
+ AddUser(user : User) : void
1
BankRepo<<implemenationclass>>
- banks : vector<string>
+ addBank(bank : string) : void
+ getbanks () : vector <string>
StockAccount
- Email :string
- Password :string
+ StockAccount(Email : string , passowrd :string)
StockRepo<<implemenationclass>>
- stocks : vector<string>
+ addStock(bank : string) : void
+ getstocks () : vector <string>
anel<<interface>>
ser
nu ( ) : void
implements
IntegrationPanel<<implemenationclass>>
- banks : BankRepo
- stocks : StockRepo
+ IntegrationPanel (user : User)
Implements
APIs_Account<<interface>>
+ Connection(user :User) : Void
# VerifyCred (account: BankAccount) :boolean
# VerifyCred (account: StockAccount) : boolean

Description Link for the design:

Software Design Specification



https://drive.google.com/file/d/1ur9hwcOiEpEGw7NfDLpcxsdzo6gH-VJE/view?usp=sharing

III. Class Descriptions

ID	Class Name	Description & Responsibility
1.	**Asset**	**Description**: Represents a financial asset (stocks, real estate, crypto, gold) in a user's portfolio. **Responsibilities**: • Store asset details (Name, Type, Quantity, purchaseTime). • Link to financial goals (Goals: Vector<Goal>).
2.	**Person**	**Description**: Base class storing common attributes for all human actors. **Responsibilities**: • Manage core attributes (Username, Name, Password, Email).
3.	**User** (Extends Person)	**Description**: Represents an investor with linked assets and bank accounts. **Responsibilities**: • Maintain collections of Bank Accounts, Stock Accounts and Assets • Provide methods to fetch assets/accounts.
4.	**BankAccount**	**Description**: Stores linked bank/card details for transactions. **Responsibilities**: • Secure sensitive data (OTP, CVV, cardNumber, cardHolder,expire date).
5.	**Goal**	**Description**: Tracks financial objectives (e.g., retirement savings).

ID	Class Name	Description & Responsibility
		Responsibilities: • Store goal metrics (targetAmount, deadline, currentProgress). • Update progress via setters (setProgress()).
6.	**AuthController**	**Description**: Handles authentication and user sessions. **Responsibilities**: • Verify credentials (Verification(user: Person): boolean). • Manage signup/login flows (SignUp(), Login()). • Interact with UserRepo for persistence.
7.	**ZakatPanel** (Interface)	**Description**: Defines contracts for Sharia-compliant zakat operations. **Responsibilities**: • Declare methods for zakat calculation (ZakatCalculation()). • Enforce halal screening (HalalInvestScreen()).
8.	**RiskAndAdvicePanel** (Interface)	**Description**: Specifies risk assessment and investment advice features. **Responsibilities**: • Define risk analysis methods (AssetRisk()). • Outline strategy optimization (OptimizeStrategy()).
9.	**FinGoalPanel** (Interface)	**Description**: Template for financial goal tools

ID	Class Name	Description & Responsibility
		Responsibilities: • Declare goal tracking methods (ViewGoals(), TrackProgress()). • Support adding new goals (AddNewGoal()).
10.	**Admin(Extends Person)**	**Description** Represents an administrator in the system. **Responsibilities:** Potentially manage users, verify fraud activities, and oversee the system's operations.
11.	**AdminRepo<<Implementation>>**	**Description** Represent the interface that the admin will interact with Responsibilities: • Check the Authorization of Admin License • Verify the Frauds of user (VerfiyFrauts (user : User))
12.	**DashboardController**	**Description** Controls the interaction between the user and the system's dashboard, coordinating with user data through a repository. **Responsibilities** • Manage the current User session. • Handle user operations by interacting with the UserRepo. • Display the dashboard menu to the user using ViewMenu() •

ID	Class Name	Description & Responsibility
13.	**UserRepo<Implementation>**	**Description** Repository class that stores all system users and their associated bank APIs. **Responsibilities:** • Maintain a Vector of User objects. • Provide access to add new user data through the method addUser(User user).
14.	**InvestDashboardPanel (Interface)**	**Description** Interface defining user investment-related operations on the dashboard. **Responsibilities:** o **Add new** assets to a user's profile. o Remove assets from a user's profile. o View a user's current assets. o Edit user assets. o Evaluate **the user's investment performance.**
15.	**Performance (Interface)**	**Description:** Interface for measuring different aspects of user performance. **Responsibilities:** • Define a common implement() method that will be customized for various performance evaluations like ROI (Return on Investment), Asset Distribution, and Valuation.

ID	Class Name	Description & Responsibility
16.	Api_Accounts	**Description:** It is an interface that defines the standard operations required for connecting to and verifying user credentials across different financial account systems. It serves as a template to ensure that all financial service implementations follow a consistent structure for integration and authentication. **Responsibility:** • Specify a method to establish a connection between the user and the financial account (Connection(User user)). • Specify a method to verify the user's credentials (VerifyCred(User user)). • Enforce a unified protocol that must be followed by all implementing classes, such as API_AlAhlyBank, API_BankMisr, API_BankHSBC, API_Plus500, API_Webull, and API_eToro. • Allow flexibility for each financial API to provide its own specific connection and verification logic while maintaining a common interface.
17.	RiskandAdvicePanel (<<ImplementationClass>>)	**Description:** Provides analytical functions for assessing risk and generating personalized financial advice. **Responsibilities:** - Assess user financial risk based on integrated data. - Optimize investment strategies to improve financial outcomes. - Advise on appropriate asset allocation using both stock and bank dat
18.	IntegrationPanel (<<ImplementationClass>>)	**Description:** Central controller that integrates a user’s bank and stock data into a unified view.

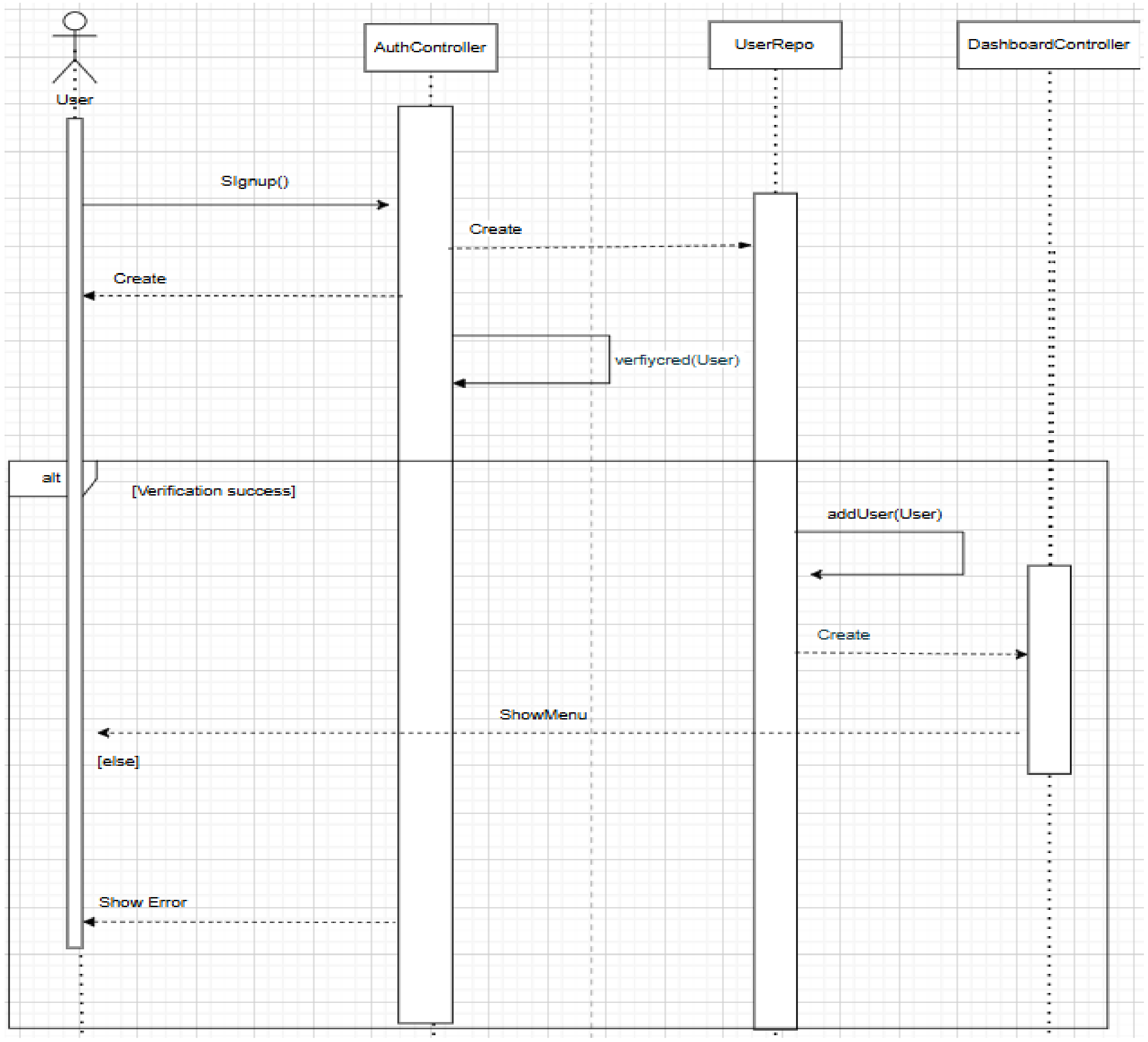
ID	Class Name	Description & Responsibility
		Responsibilities: - Aggregate and manage instances of BankRepo and StockRepo. - Bind financial data to a specific user for consistent access. - Serve as the data source for risk and strategic financial analysis.
19.	StockRepo	**Description:** Manages stock-related data for users, including the addition and retrieval of stock holdings. **Responsibilities:** - Store a list of stocks associated with a user. - Provide methods to add new stocks to the repository. - Enable access to a user's entire stock portfolio for downstream use.
20.	BankRepo	**Description:** Handles the storage and retrieval of a user's associated banking institutions. Responsibilities: - Store a list of banks linked to a specific user. - Provide methods to add new bank entries. - Enable retrieval of all associated banks for integration or verification purposes.
21.	StockAccount	**Description:** **Encapsulates login credentials for a user's stock trading or investment account.** **Responsibilities:** - Store and protect user credentials (email and password). - Provide a constructor for secure initialization of account objects.

ID	Class Name	Description & Responsibility
		- Support the verification process for stock-related APIs

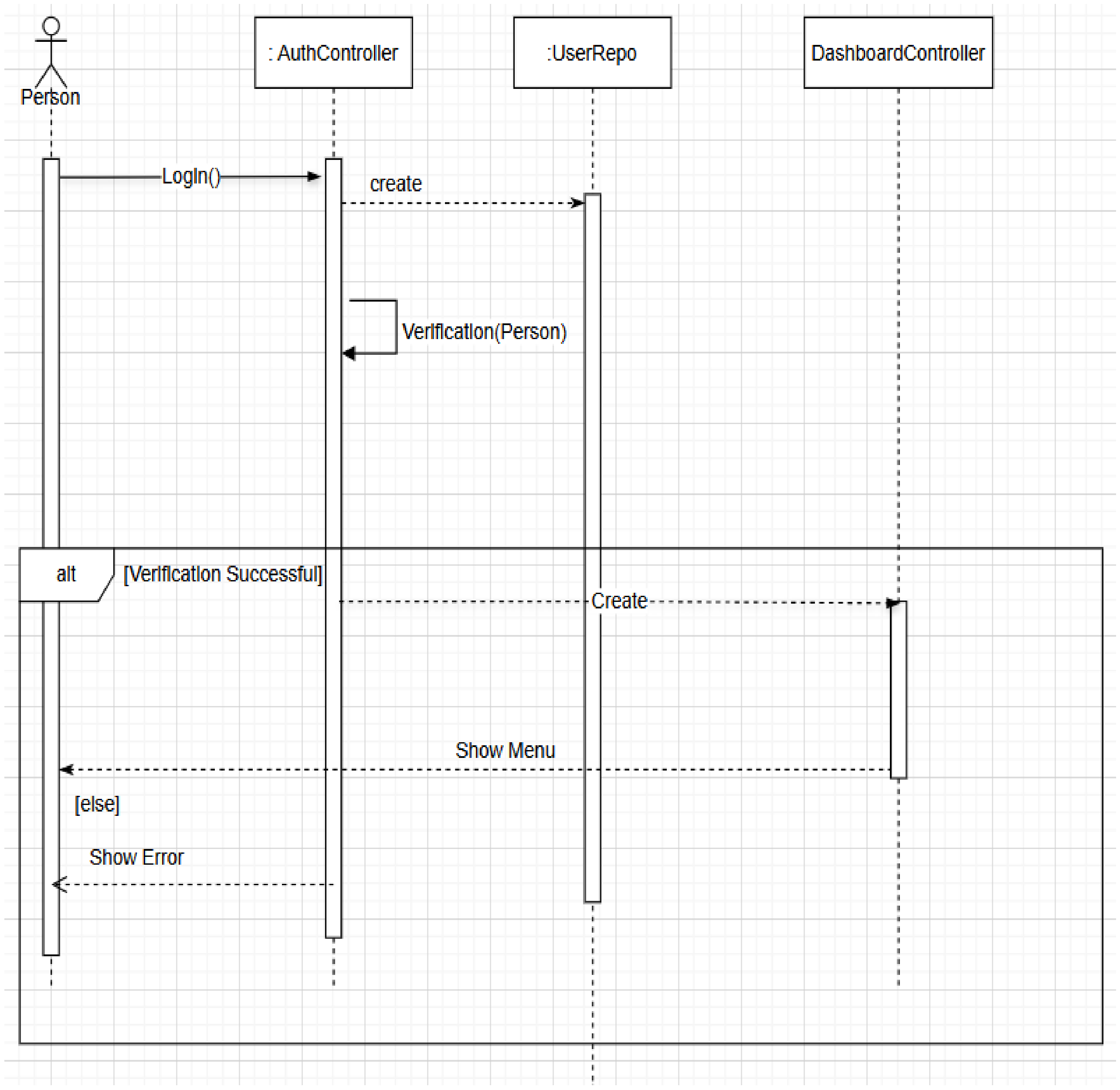
Software Design Specification

IV. Sequence diagrams

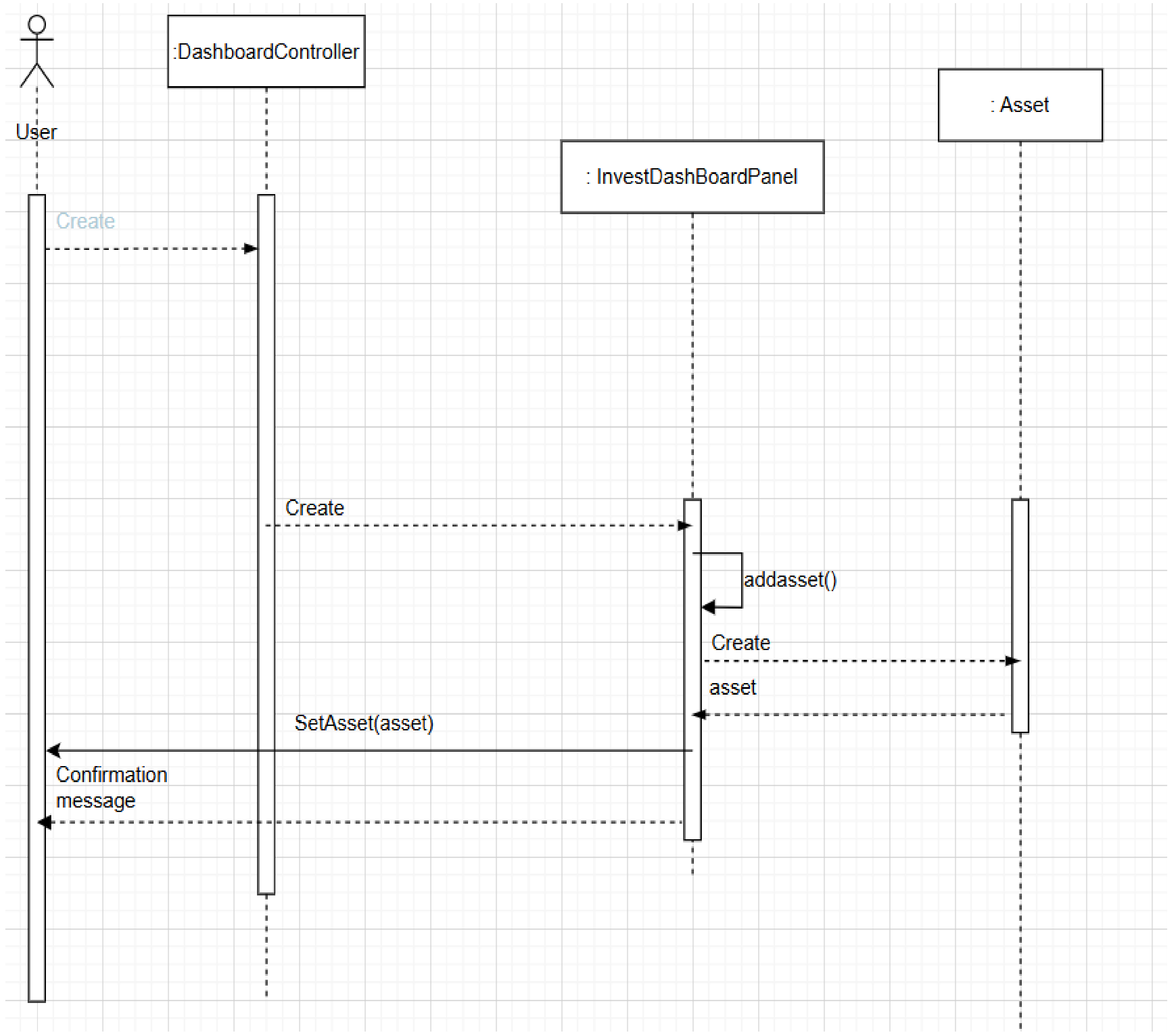
Sign up Use case



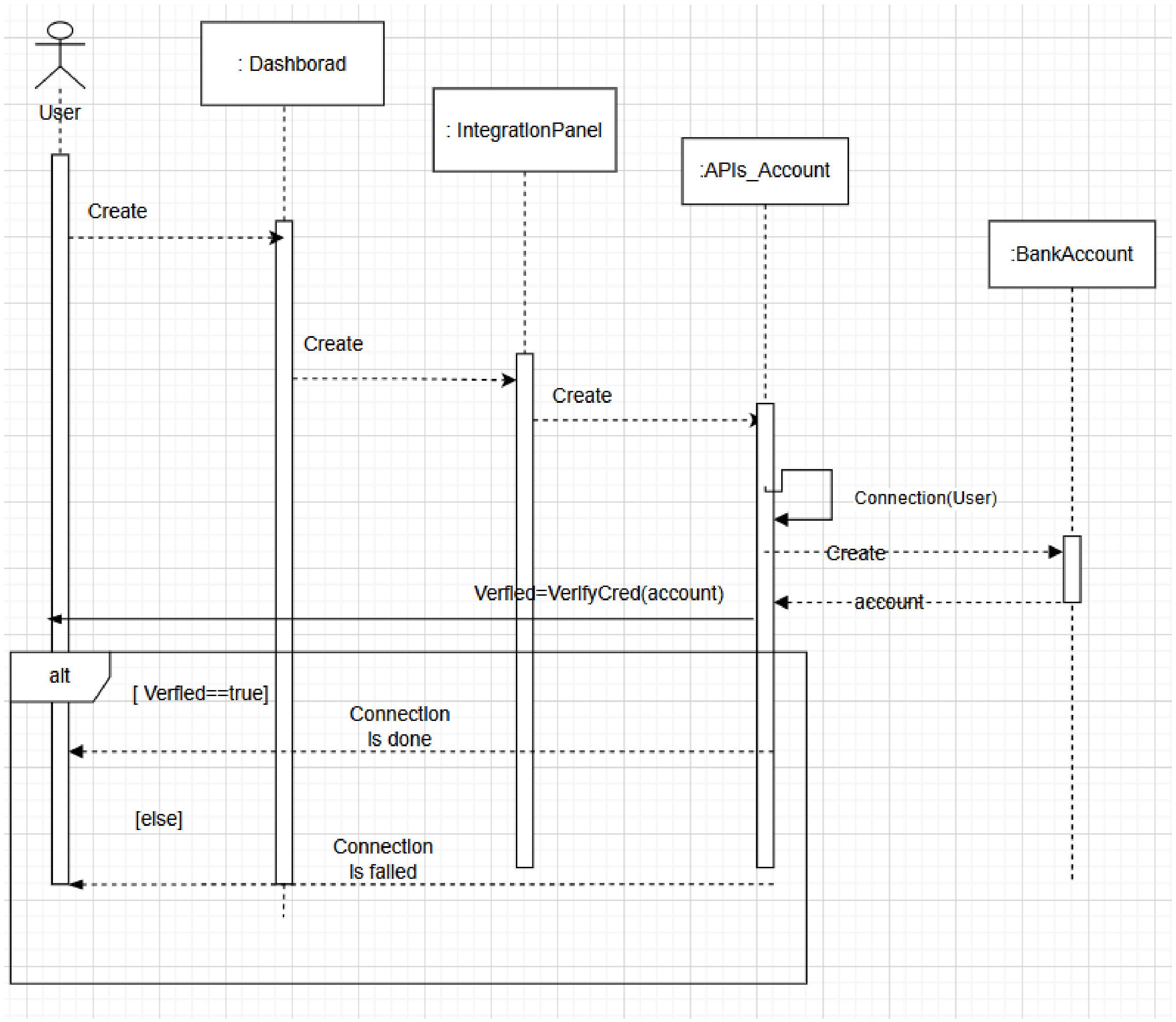
Login Use Case



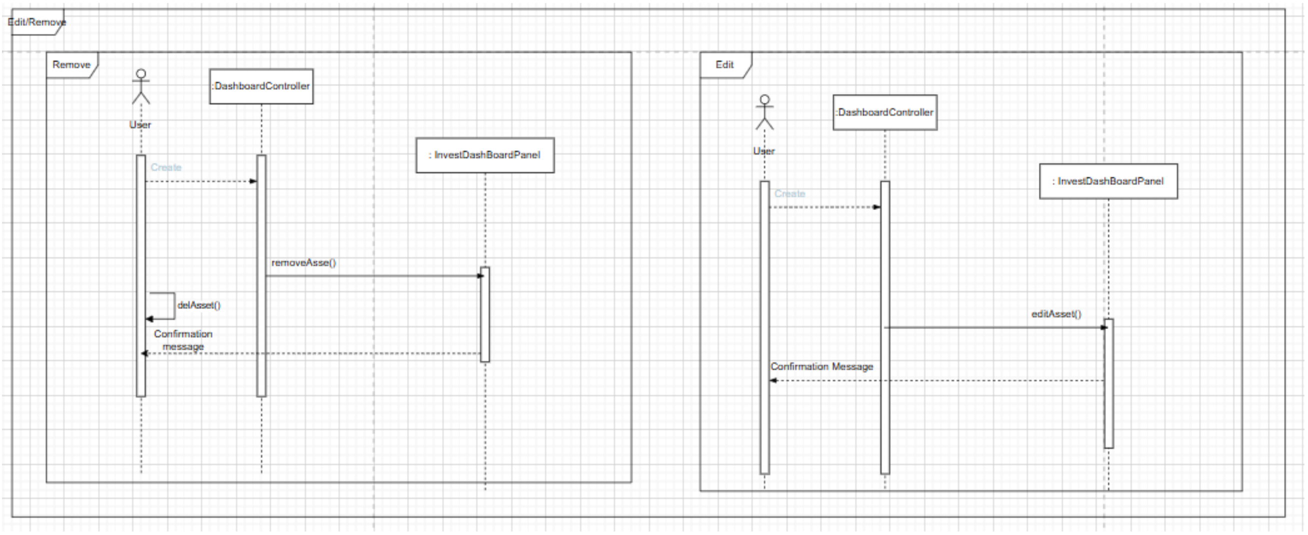
Add Assets Use Case.



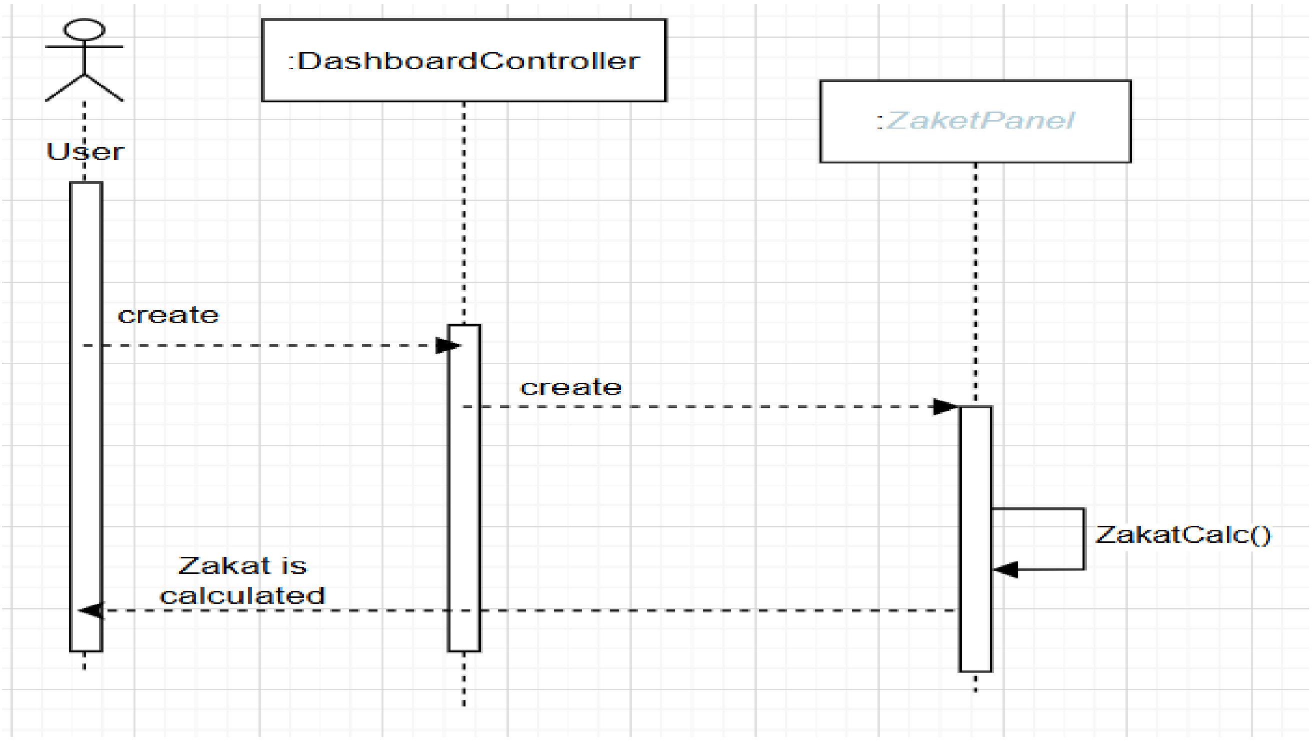
Connect Bank account Use case



Edit and remove assets use case



Zakat Calculation Use Case



Software Design Specification

Class - Sequence Usage Table

Sequence Diagram	Classes Used	All Methods Used

1. Sign Up	Authcontroller
	User
	UserRepo
	DashboardController
		Signup()
		VeerifyCred(user)
		addUser(user)

2. Login	AuthController
	UserRepo
	Person
	DashboardController
		Login()
		Verification(User : Person)

3. AddAssets	DashBoardController
	<<Utility>>InvestDashBoardPanel
	User
	Asset
		addAsset(user)
		setAsset(asset)

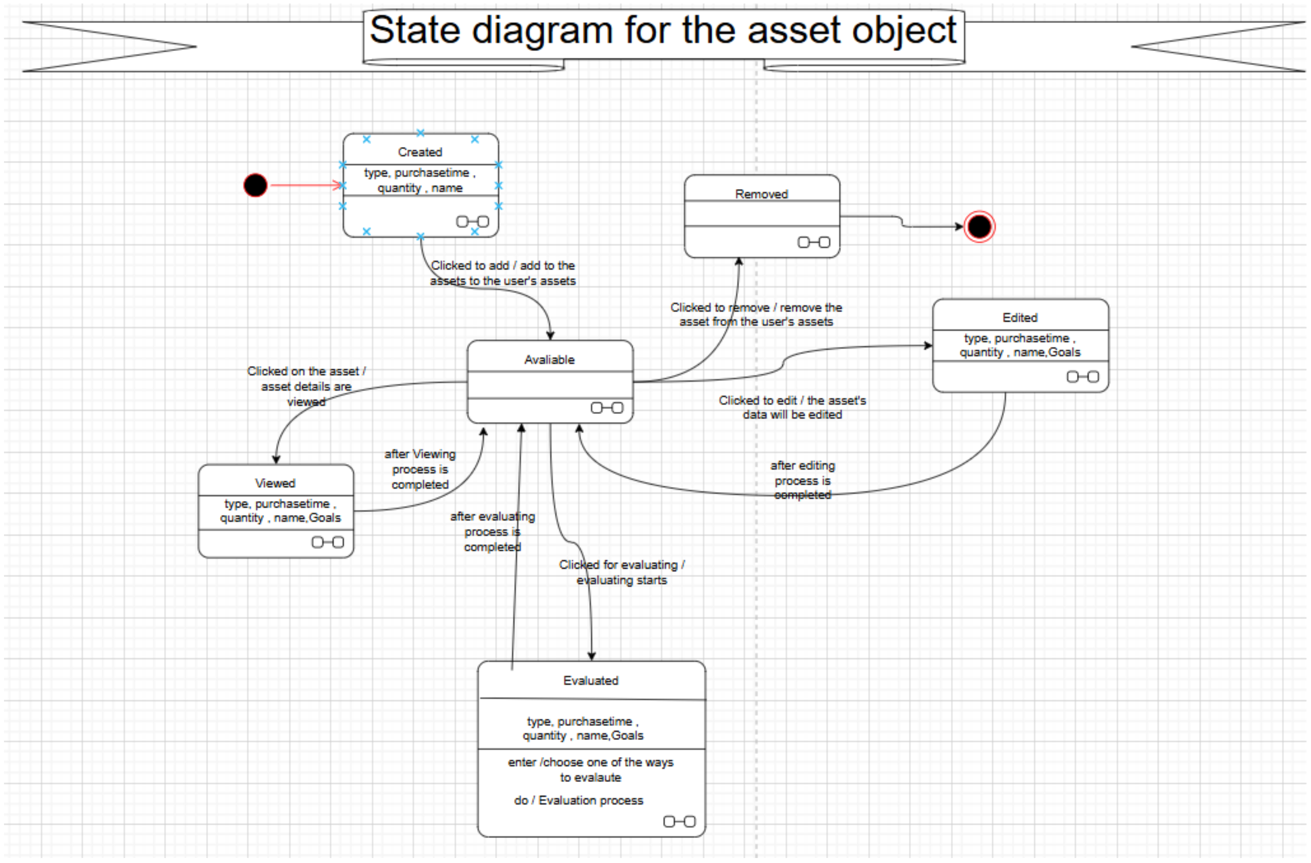
4. Connect Bank account	Dashboard
	integrationPanel
	API_account
	User
	bankAccount
		Connection(user)
		verifyCred(account)

5. Edit and remove assets	DashboardController
	InvestmentDashBoardPanel
		addAssets()
		RemoveAssets()
		delAssets()

6. Zakat Calculation	DashboardController ZakatPanel	ZakatCalc()

Software Design Specification

V. State Diagram


State diagram for the asset object
Created
type, purchasetime , quantity , name
Removed
Clicked to add / add to the assets to the user's assets
Clicked to remove / remove the asset from the user's assets
Edited
type, purchasetime , quantity , name,Goals
Avaliable
Clicked on the asset / asset details are viewed
Clicked to edit / the asset's data will be edited
after Viewing process is completed
after editing process is completed
Viewed
type, purchasetime , quantity , name,Goals
after evaluating process is completed
Clicked for evaluating / evaluating starts
Evaluated
type, purchasetime , quantity , name,Goals
enter /choose one of the ways to evalaute
do / Evaluation process

VI. SOLID Principles

Single Responsibility principle:

In our class diagram, we adhered to the **Single Responsibility Principle (SRP)** by ensuring that **each class and interface has one distinct responsibility**. For example:

- **AuthController** focuses solely on handling user authentication, specifically login and sign-up functionalities, without mixing in unrelated logic.
- **Panel** serves as a base interface, with each child class providing its own implementation based on the specific panel type. This separation ensures that changes in one panel type do not affect others.
- **User**, **Goal**, and **Assets** classes are responsible only for storing and managing data relevant to their domain. Each class encapsulates its own related data and logic, maintaining a clear and focused responsibility.

This design approach improves **modularity**, **maintainability**, and **ease of future extension** in the system.

Liskov Principle:

In our class diagram, we followed the Liskov Substitution Principle (LSP) by designing our interfaces and base classes in a way that allows their derived classes to be used interchangeably without affecting the correctness of the program. Example:

- The Panel interface defines a common method ViewMenu() that all its child classes (e.g., ZaketPanel, FinGoalPanel, RiskAndAdvicePanel) implement. These child classes can be used wherever a Panel type is expected, without altering the behavior of the system.
- Similarly, the APIs_Account interface is implemented by multiple classes such as API_AlAhlyBank, API_BankMisr, and API_HSBC. Each of these can substitute the APIs_Account reference while maintaining consistent functionality for connection and credential verification.

This structure enhances **scalability** and reduces the risk of breaking existing code when new features are introduced.

Open Closed Principle

By ensuring that subclasses extend the behavior of their base classes without breaking existing functionality, we maintain code robustness, polymorphism, and flexibility.

In this part of our class diagram, we continued to follow the Open/Closed Principle by allowing the system to be extended through composition and associations without modifying existing classes:

- The User class aggregates a list of Assets and BankAccounts. If we need to support new types of assets or bank accounts in the future, we can extend the system by adding new classes or types, without changing the existing User structure.
- The use of composition between Asset and Goal also supports OCP. Each Asset can be associated with multiple Goals, and new goal-related logic can be added without modifying the Asset class.
- The person superclass is inherited by both User and Admin, allowing for future expansion of user roles through inheritance rather than editing the person class directly.

This structure ensures that core components remain stable and untouched, even when new functionalities are introduced.

Interface Segregation Principle :

We applied the Interface Segregation Principle by breaking our Panel interface into small, focused roles (like FilesPanel for assets, FinGoalPanel for goals, etc.) instead of one bloated interface. This way, each panel only implements what it needs—for example, RiskandAdvicePanel handles risk calculations but doesn't force unrelated methods like AddNewGoal() from FinGoalPanel. Clients (like controllers) only depend on the specific panels they use, avoiding "fat interfaces" that would require empty method overrides.

Why This Follows ISP:

1. No Forced Dependencies:

o A DashboardController using FilesPanel won't need to know about ZakaiCalculation() from ZakeiPanel.

2. Relevant Contracts:

- o Each panel interface contains only methods relevant to its purpose (e.g., Evaluate() for FilesPanel, OptimizeStrategy() for RiskandAdvicePanel).

Software Design Specification

VII. Design Pattern

Strategy pattern:

We used the Strategy pattern with the Performance interface and APIs_Account. The Performance interface sets a common rule (implement()), While APIs_Account interface sets the connection and verfiyCred as a common methods between the different APIS , keeping everything clean and modular.

Repository pattern:

We implemented the Repository design pattern to centralize data access logic for User and Admin entities, separating business logic from database operations. The UserRepo and AdminRepo classes act as intermediaries between the application and the data layer, encapsulating queries (e.g., AddUser, VerifyFrauds) and managing in-memory collections (Vector<User>, Vector<Admin>). This pattern simplifies maintenance by providing a single point of change for data operations, improves testability (via mock repositories), and ensures consistent access rules across the application. For example, AdminRepo handles fraud verification independently, while UserRepo manages user-specific panel assignments, demonstrating clear separation of concerns. Moreover , we made the BankRepo and StockRepo that stores the companies that we deal with and the admin could handle that part , but that should be added as future adds. the admin could add the company names that the system deals with and ofcourse that will help the developers and designers could add if they need in the future to add APIS of the new added companies.

Model-Viewer-Controller pattern :

We used MVC to cleanly separate our investment app's data, interface, and logic. The Model (like Person, Asset, Goal, and repos) handles data and business rules, the View (Panel interface and implementations) displays information to users, and the Controller (DashboardController, AuthController) manages input—like when a user logs in (AuthController) or views their portfolio (DashboardController). This keeps code organized: models focus on data, views on presentation, and controllers on connecting them without letting any layer do too much. Key Benefits: • Easy Updates: Change UI (Panel views) without touching data models. • Reusable Models: Person/Asset work the same whether shown in ZakeiPanel or FinGoalPanel. • Clear Flow: Controllers decide which view to show based on user actions (e.g., login → dashboard). Why MVC Fits Your Design:

Software Design Specification

1. Models (Person, Asset, repos):

   o Store data (e.g., Person.getEmail()). o Handle business logic (e.g., UserRepo.AddUser()).

2. Views (Panel implementations):

   o Display data (e.g., RiskandAdvicePanel shows risk stats). o Get user input (e.g., EditAsset() in FilesPanel).

3. Controllers:

   o AuthController: Manages login/signup, checks credentials. o DashboardController: Loads the right Panel based on user role/actions.

CS251: Optimum
Project: Tharwa

Software Design Specification

Tools

- Draw.io

Ownership Report

Item	Owners
Architecture Diagram, Part of Class Diagram, Add, part of edit and remove assets Sequence Diagrams, connect Bank account sequence diagram, Zakah sequence diagram and state diagram.	Fatema El-Zhraa Ahmed Mohamed El-Fiky
Part of Class Diagram, Sign up Sequence Diagram solid principle, File Organization, Purpose of SDS and Class Responsibility.	Aly El-Deen Yasser Ali
Part of Class Diagram , login Sequence Diagram, design patterns report part of edit and remove assets Sequence Diagrams and Audience of SDS	Nagham Wael