Cairo University

Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

Personal Budgeting App

Software Design Specifications

Version 1.0 2025

CS251: Code Father
Project: Personal Budgeting App

Software Design Specification

Contents

CS251: Code Father
Project: Personal Budgeting App

Software Design Specification

Team

ID	Name	Email	Mobile
20231129	ماركو رأفت زكريا سوارس	marco782005@gmail.com	01274813802
20230438	مينا ماهر توفيق صالح جرجس	mina005smart@gmail.com	01206201907
20231126	كيرلس اكرم ماضي عبد المسيح	Kerolus.Akram@gmail.com	01210990045

Document Purpose and Audience

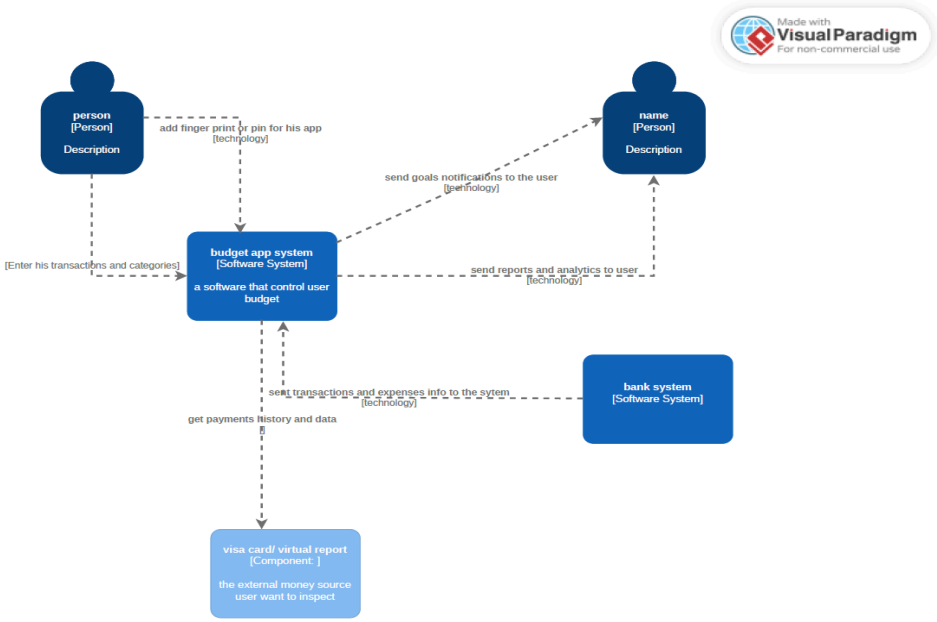
This project is about creating and implementing a budget app for users to manage their money and it contains the way of implementation for this app using different diagrams its made specifically for Technicals and software engineers

Software Design Specification

System Models

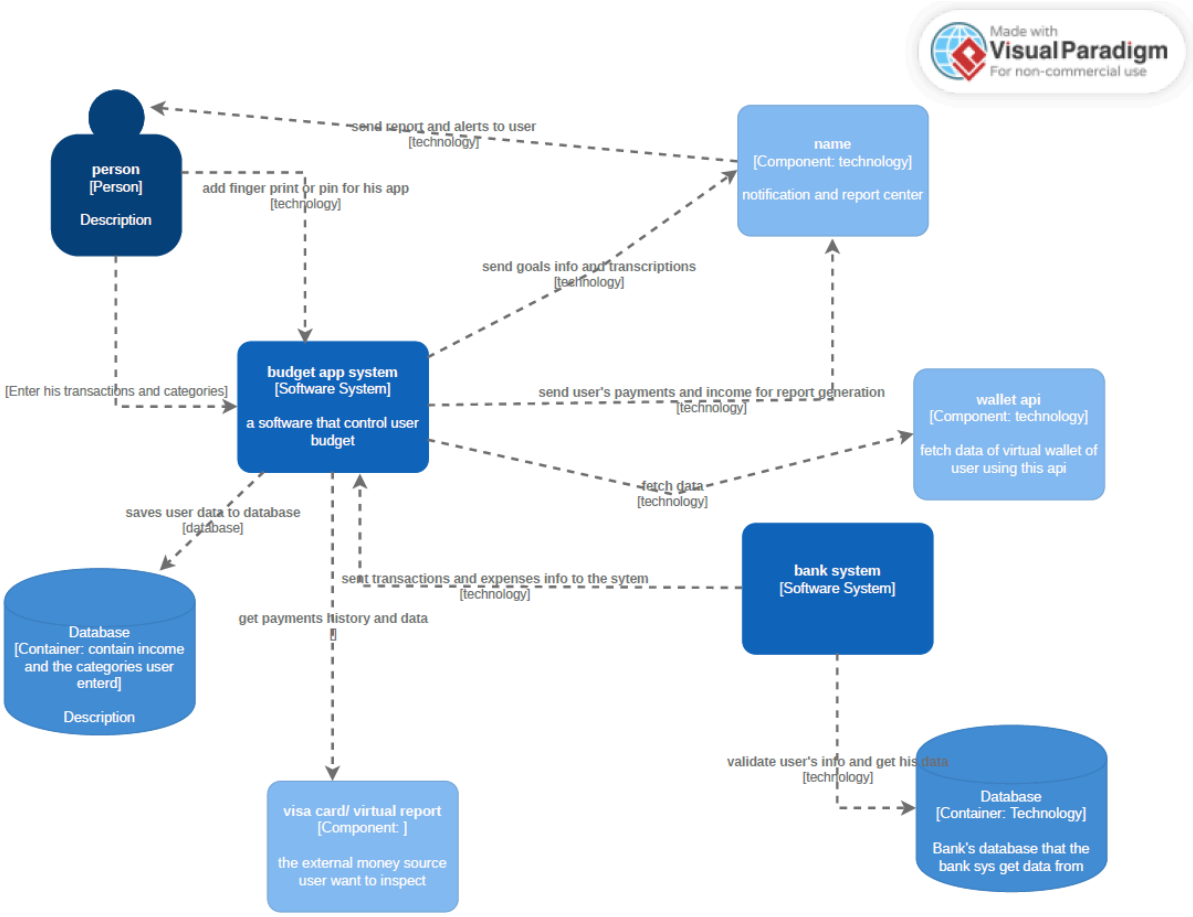
I. Architecture Diagram

1. **system context diagram** :diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with



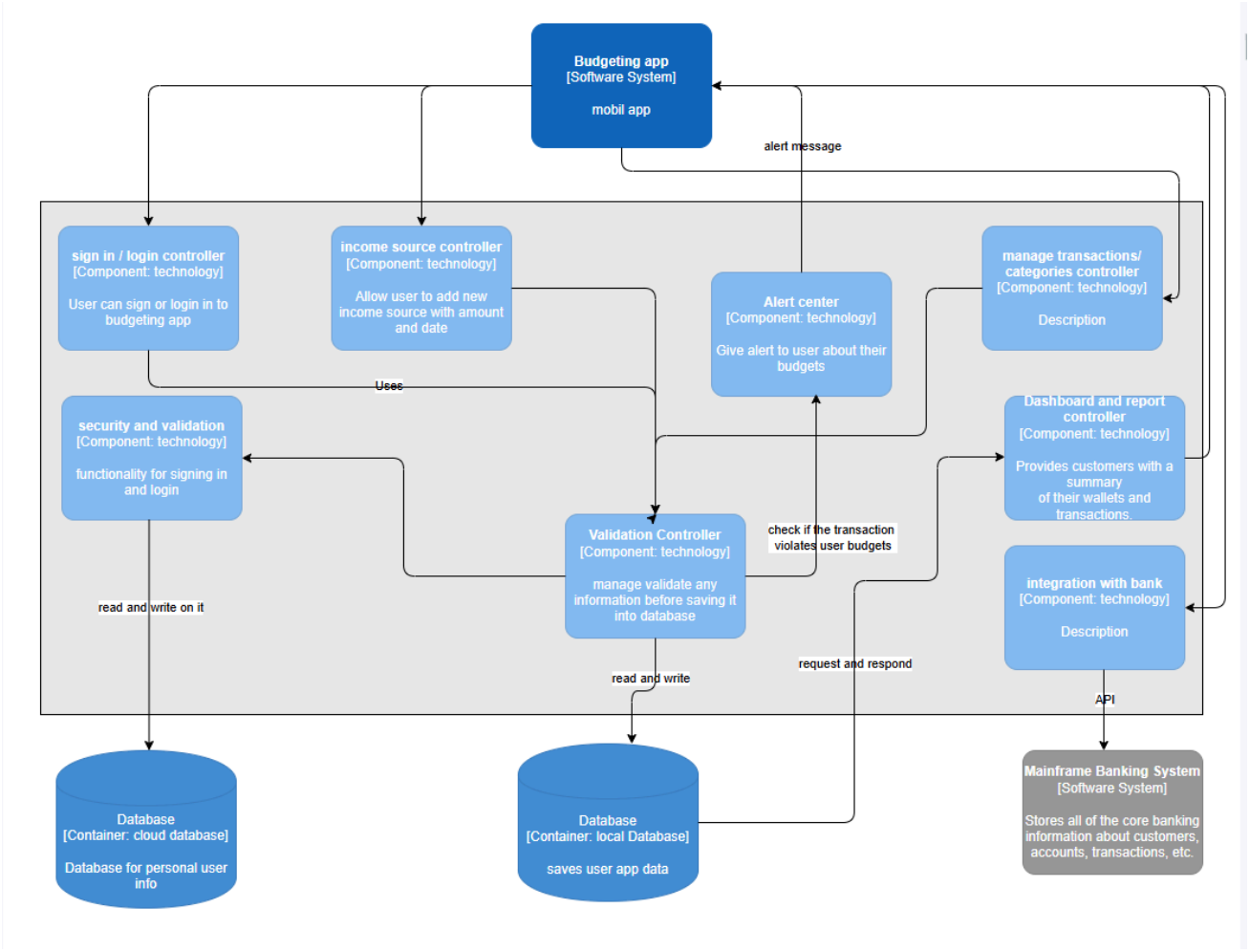
Software Design Specification

2. **container diagram:** **The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it**



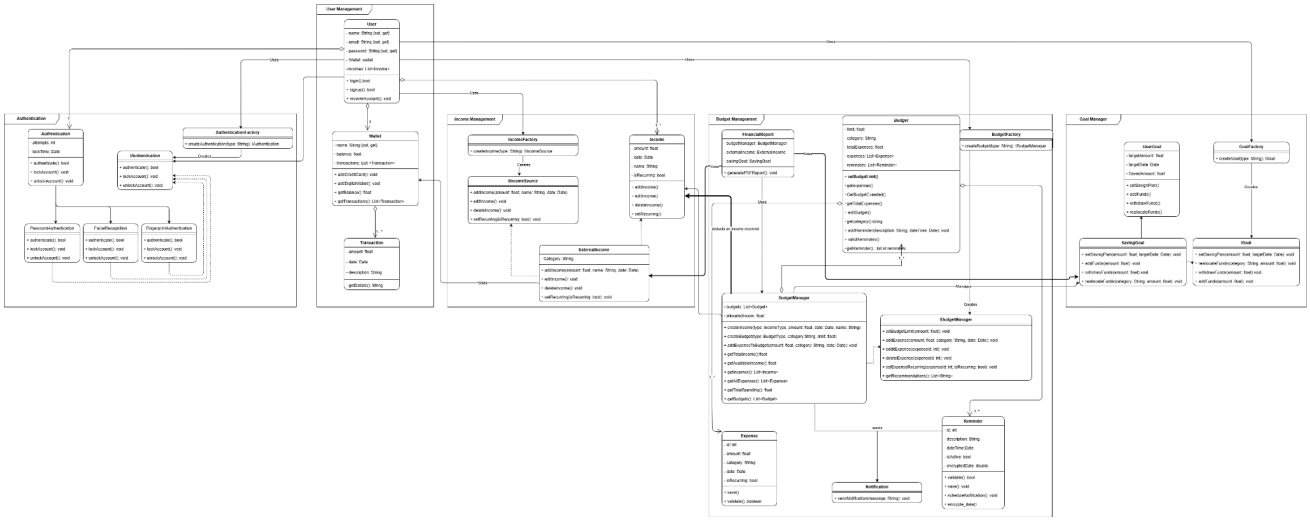
Software Design Specification

3. **component diagram:** the Component diagram shows how a container is made up of a number of “components”, what each of those components are, their responsibilities and the technology/implementation details



Software Design Specification

II. Class Diagram(s)



[Click here for high quality]

Software Design Specification

III. Class Descriptions

ID	Name	Description
1	User	Represents a user of the system, managing account details and initiating interactions with income, wallet, budgets, goals, and reminders.
2	Wallet	Manages the user's financial balance and transaction history, supporting the addition of payment methods like credit cards and digital wallets.
3	FacialRecognition	Implements facial recognition-based authentication for user login and account security.
4	FingerprintAuthentication	Implements fingerprint-based authentication for user login and account security.
5	PasswordAuthentication	Implements password-based authentication for user login and account security.
6	AuthenticationFactory	Creates instances of authentication methods (e.g., facial, fingerprint, password) based on the specified type.
7	ExternalIncome	Manages external income sources (e.g., salary), supporting recurring income and integration with the wallet.
8	IncomeFactory	Creates instances of income sources (e.g., ExternalIncome) based on the specified type.
9	BudgetManager	Manages budgets, expenses, reminders, and notifications, including setting limits, adding

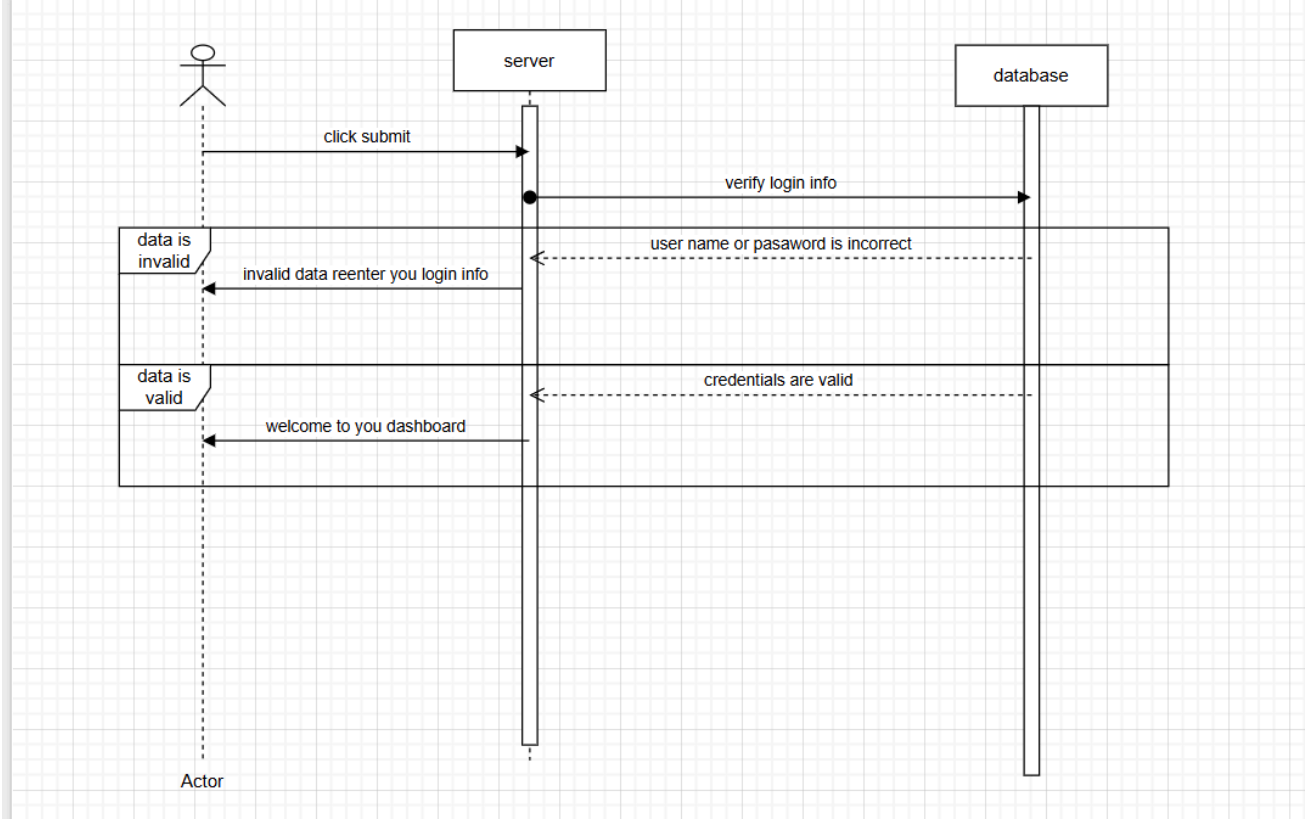
Software Design Specification

		expenses, and scheduling recurring transactions or reminders.
10	Expense	Represents individual expense transactions, including amount, category, date, and recurring status.
11	Reminder	Represents reminders for bill payments or financial tasks, handling validation, saving, and scheduling notifications.
12	FinancialReport	Generates PDF reports based on budget, income, and savings goal data for financial analysis.
13	BudgetFactory	Creates instances of budget managers (e.g., BudgetManager) based on the specified type.
14	SavingGoal	Manages savings goals, allowing users to set plans, add/withdraw funds, and reallocate funds.
15	GoalFactory	Creates instances of goal types (e.g., SavingGoal) based on the specified type.

Software Design Specification

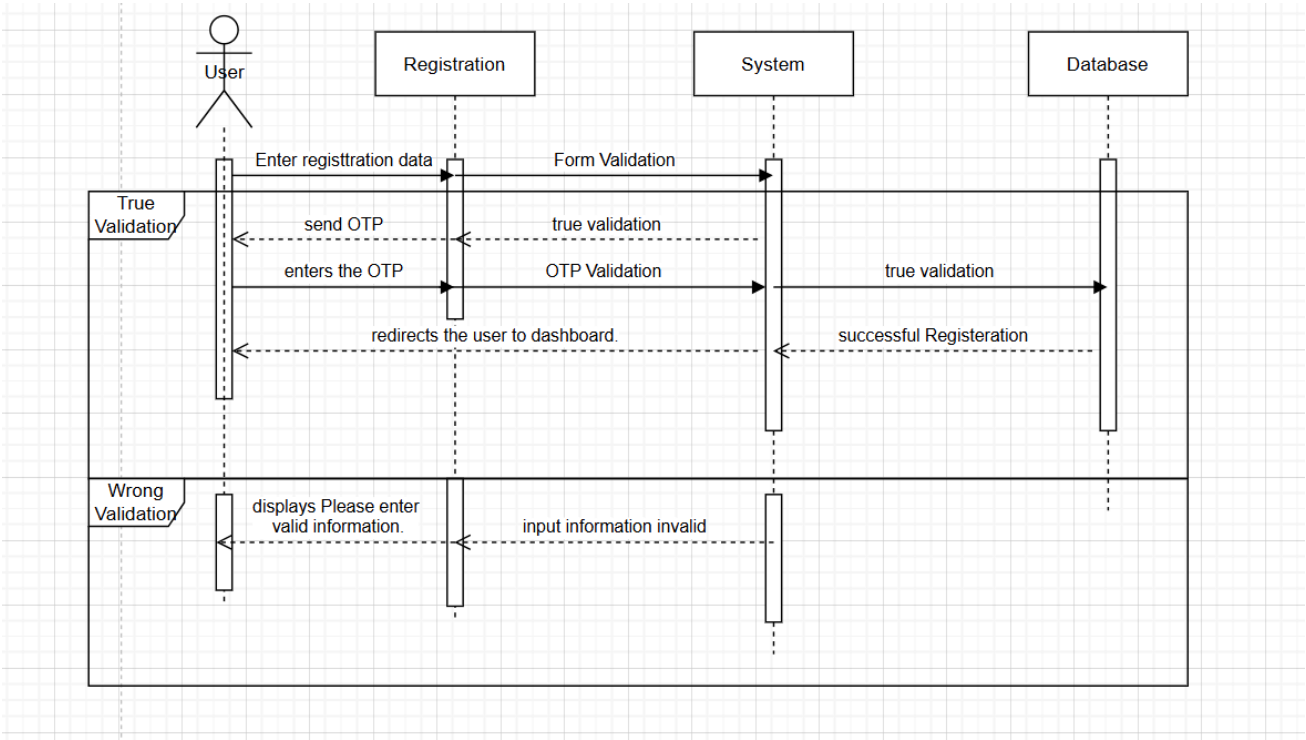
IV. Sequence diagrams

user story 1: Login process



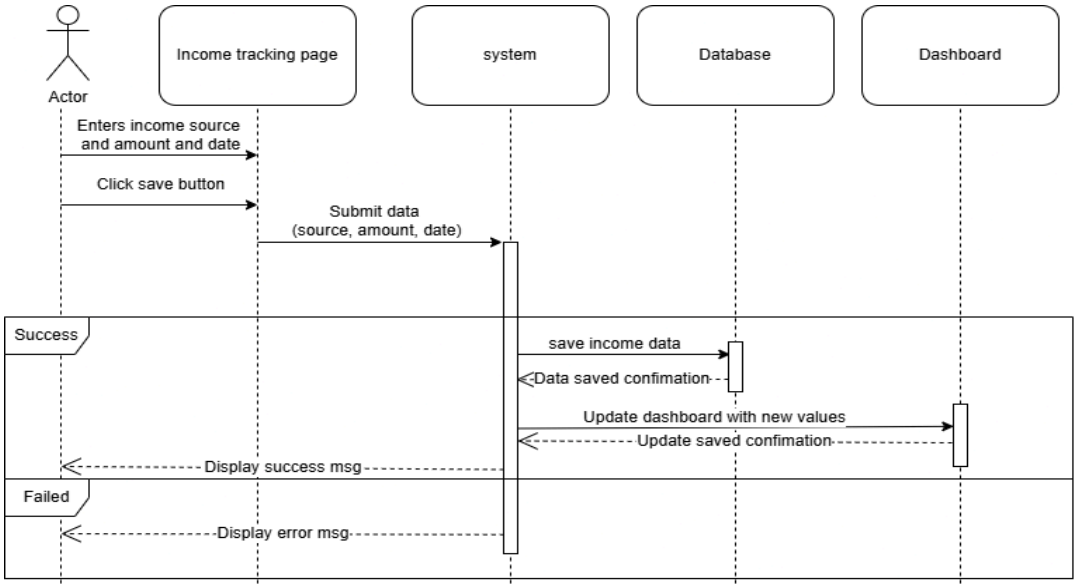
Software Design Specification

user story 2: signup process

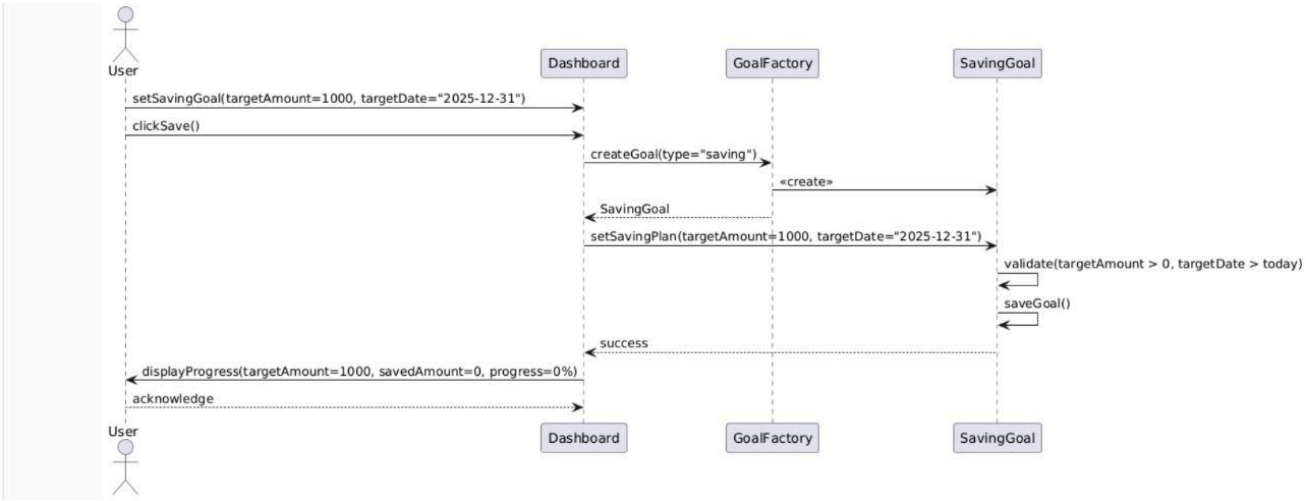


Software Design Specification

user story 3: Add income source and display it into dashboard

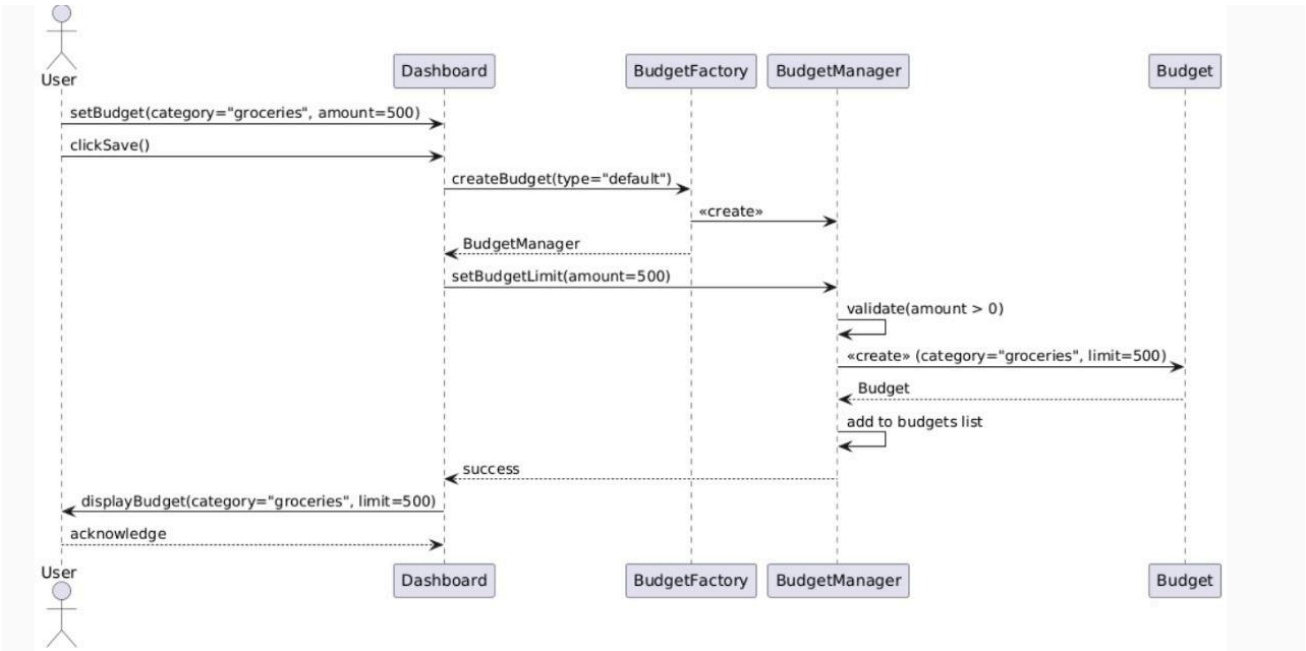


user story 6 : saving goals

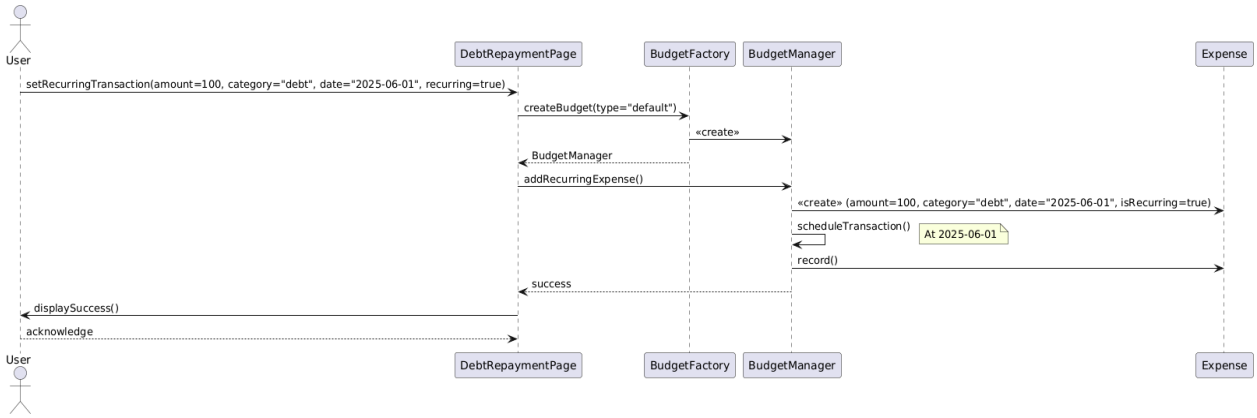


Software Design Specification

user story 5: set budget



user story 9:Transactions



Software Design Specification

Class - Sequence Usage Table

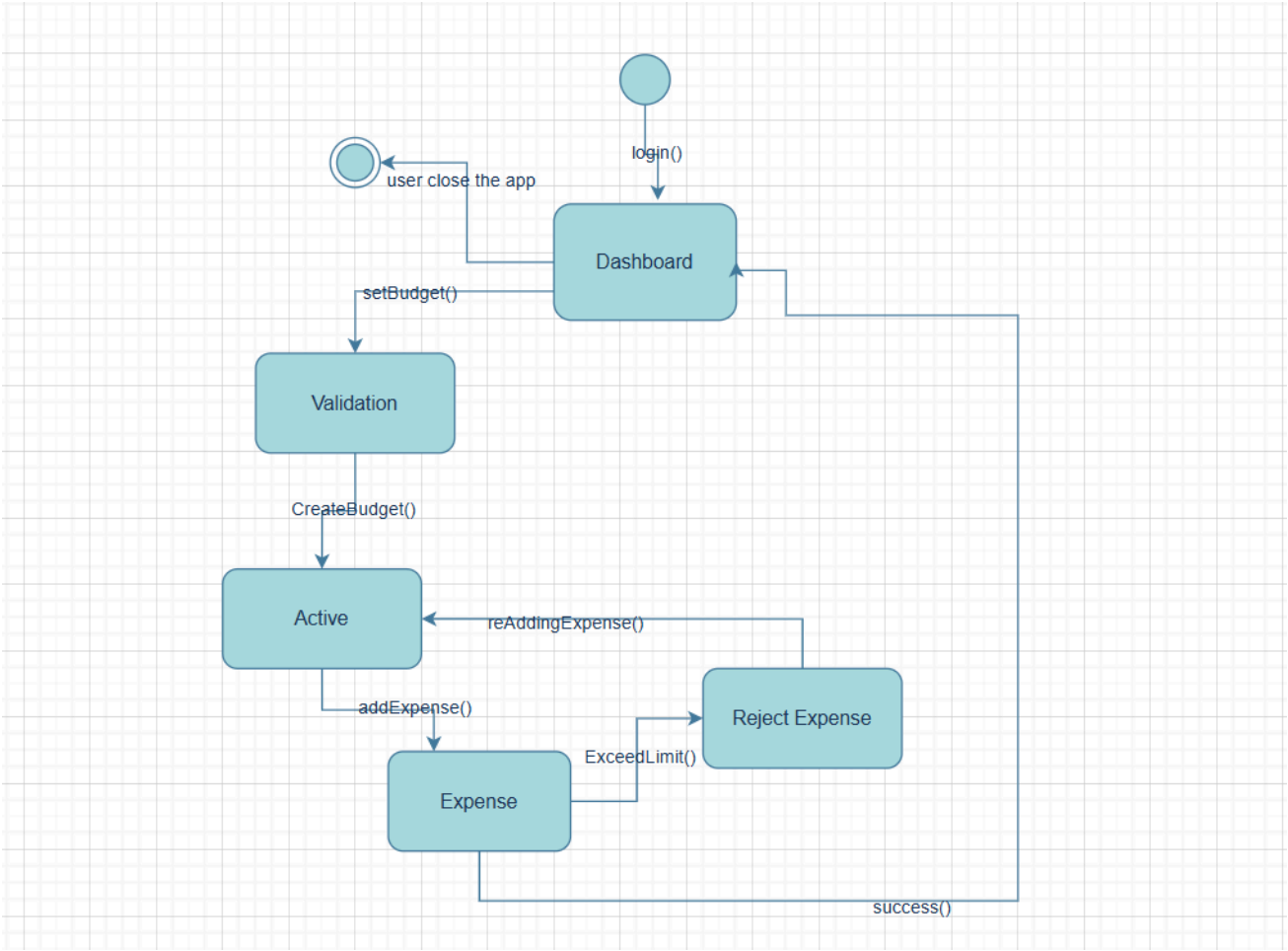
Sequence Diagram	Classes Used	All Methods Used
SignUP	UserClass	SignUpPage, PasswordAuthentication
	SignUpPage	UserClass, AuthenticationFactory, User
	AuthenticationFactory	PasswordAuthentication
	PasswordAuthentication	Notification
	Notification	PasswordAuthentication
	User (Actor)	SignUpPage
Login	UserClass	LoginPage, PasswordAuthentication
	LoginPage	UserClass, AuthenticationFactory, User
	AuthenticationFactory	PasswordAuthentication
	PasswordAuthentication	UserClass
	User (Actor)	LoginPage
Add income source	UserClass	IncomeFactory, Income, Wallet
	IncomeFactory	ExternalIncome
	ExternalIncome	Wallet
	User (Actor)	UserClass
budgeting and analysis	User	None (actor only)
	Dashboard	None
	BudgetFactory	createBudget(type: String)

Software Design Specification

Sequence Diagram	Classes Used	All Methods Used
	BudgetManager	setBudgetLimit(amount: float)
	Budget	None (creation only)
saving and goal	Person	None (actor only)
	UserGoal	None
	GoalFactory	createGoal(type: String)
	SavingGoal	setSavingPlan(amount: float, targetDate: Date)
Transactions	User	None (actor only)
	DebtRepaymentPage	None
	BudgetFactory	createBudget(type: String)
	BudgetManager	None
	Expense	None

Software Design Specification

V. State Diagram



VI. SOLID Principles

1. Single Responsibility Principle (SRP)

Each class has a clear, single responsibility:

- Manages account and interactions.

Software Design Specification

- `Wallet`: Manages financial balance and transactions.
- `BudgetManager`: Handles budgeting and expenses.
- `Reminder`: Deals with reminders only.
- `Authentication`, `Income`, `Budget`, `UserGoal`: Abstract base classes with core functionality.

2. Open/Closed Principle (OCP)

Classes are open for extension but closed for modification:

- Factories like `AuthenticationFactory`, `IncomeFactory`, `BudgetFactory`, and `GoalFactory` allow adding new types without modifying existing logic.
- Abstract classes and interfaces (e.g., `IIncomeSource`, `IGoal`) allow new implementations like `ExternalIncome`, `SavingGoal`.

3. Liskov Substitution Principle (LSP)

Subtypes can replace their base types:

- `ExternalIncome` can be used wherever `Income` or `IIncomeSource` is expected.
- `SavingGoal` substitutes `UserGoal` or `IGoal`.
- All concrete authentication classes (`FacialRecognition`, `PasswordAuthentication`, etc.) can substitute `IAuthentication`.

4. Interface Segregation Principle (ISP)

Interfaces like `IAuthentication`, `IIncomeSource`, `IBudgetManager`, and `IGoal` define small, specific contracts, keeping clients from depending on unused methods.

5. Dependency Inversion Principle (DIP)

High-level modules depend on abstractions:

- `User` depends on interfaces like `IAuthentication`, `IIncomeSource`, not concrete classes.
- Factory classes depend on interfaces and return abstractions.

VII. Design Patterns

1. Factory Method Pattern

- `AuthenticationFactory`, `IncomeFactory`, `BudgetFactory`, `GoalFactory`: Create concrete instances of their respective interface types based on input.

2. Strategy Pattern

- `User` uses different strategies (`FacialRecognition`, `FingerprintAuthentication`, `PasswordAuthentication`) for authentication through the `IAuthentication` interface.

3. Template Method Pattern

- Abstract classes like `Authentication`, `Income`, `Budget`, and `UserGoal` define template behavior; concrete classes override specific methods.

4. Composite Pattern

- `BudgetManager` contains multiple `Budget`, `Expense`, and `Reminder` objects, treating them uniformly as components of a larger budget plan.

Software Design Specification

5. Dependency Injection (manual)

    - Through factory methods and interface usage, dependencies like `IIncomeSource`, `IGoal`, etc., are injected instead of hardcoded.

Tools

- **Plant uml**
- **Visual paradigm**
- **Draw.io**
- **Google docs**

Ownership Report

Name	Item
Mina Maher	Architecture diagram: System context, Container. Class Diagram Sequence diagrams 1,5,6,9. Design Patterns.
Marco Raafat	Architecture diagram: Component diagram. Sequence diagrams 2. State Diagram. Design Patterns.
Kerolus Akram	Architecture diagram: Component diagram. Class Diagram Sequence diagrams 3. SOLID Principles.