Cairo University
Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

Budget Tracker

Software Design Specifications

Salsabil Bahaa Eldin

Ola Ghoneim

Mariem Refaey

Version 2.0

12th May 2025

CS251
Project: <Budget Tracker>

Software Design Specification

Contents

CS251
Project: <Budget Tracker>

Software Design Specification

Team

ID	Name	Email	Mobile
20230586	Salsabil Bahaa Eldin	Salsabilbahaa2005@gmail.com	01124350668
20231232	Ola Ghoneim Hammad	olaghoneim38@gmail.com	01080972599
20231163	Mariem Refaey	refaeymariem@gmail.com	01003145059

Document Purpose and Audience

This document explains how the software system will be built. It describes the overall architecture, detailed design, components, interfaces, and data structures. It acts as a guide for developers to implement the system correctly. It also shows how the system meets the requirements defined in the SRS (Software Requirements Specification).

Target Audience:

- Software Developers: to implement the system according to the design.
- Software Architects: to review and approve the design structure.
- Project Managers: to understand how the system will be built and track progress.
- Testers: to prepare detailed test cases based on the design.
- Technical Leads: to ensure the best practices are followed and the design is feasible.
- Future Maintainers: to maintain, update, or enhance the system later.

System Models

I. Architecture Diagram

The chosen architecture is Microservices Architecture. It is suitable for our Project because of

• **Modularity:** The system's features (like income/expense tracking, budget setup, reporting, and bank syncing) can be built, deployed, and scaled separately, enhancing development efficiency.

• **Scalability:** Microservices enable specific components to scale according to their needs (e.g., the reporting module can handle increased demand during high-traffic periods).

Software Design Specification

• **Technological Flexibility**: Each component can leverage the most suitable technology for its purpose (e.g., a data-intensive reporting service can use a robust backend, while the UI remains lightweight).

• **Ease of Updates**: With independent services, updates or fixes to one feature (like budget tracking) won't disrupt the rest of the system, simplifying maintenance.

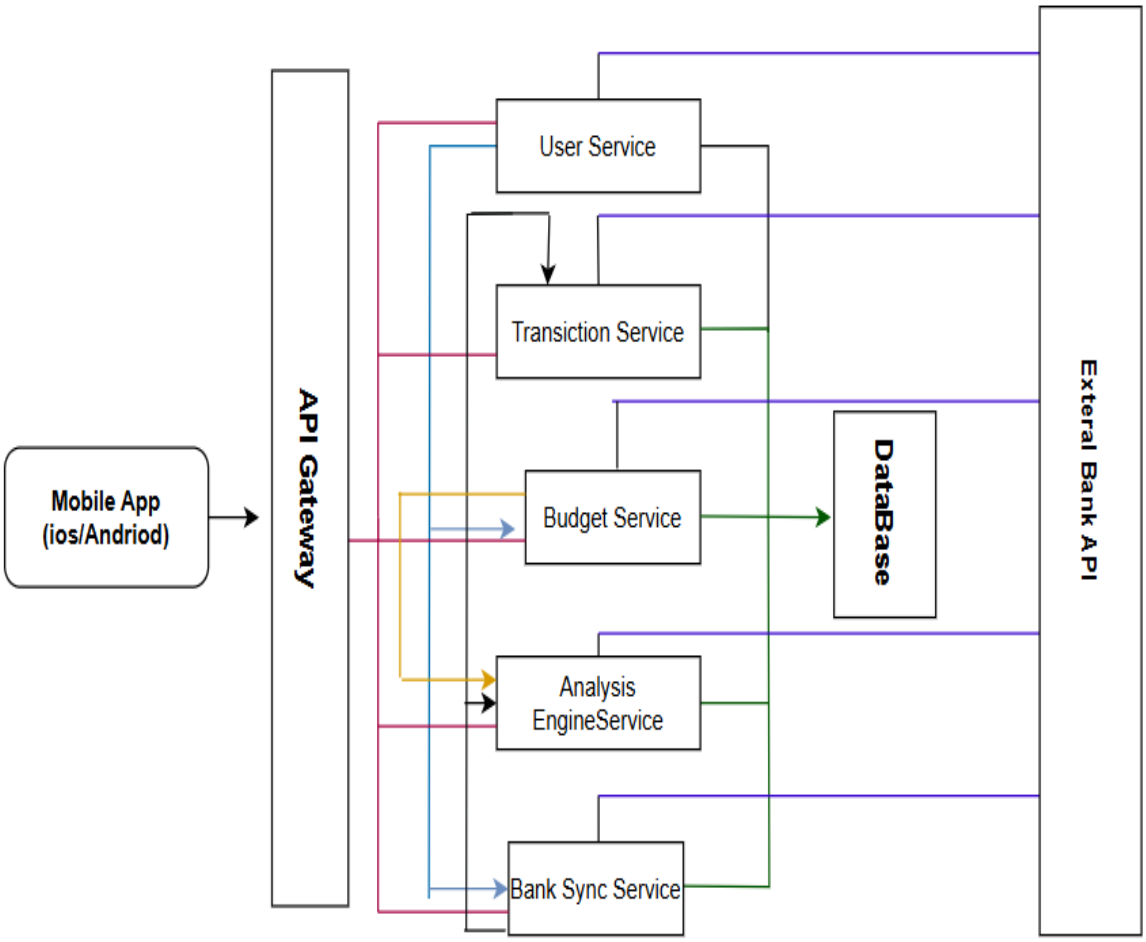
• **Diverse User Needs**: The software serves a wide range of users (from those wanting detailed control to those preferring simplicity), and microservices allow for customized experiences to meet these varying preferences.

System Components

The system can be divided into the following components (or microservices):

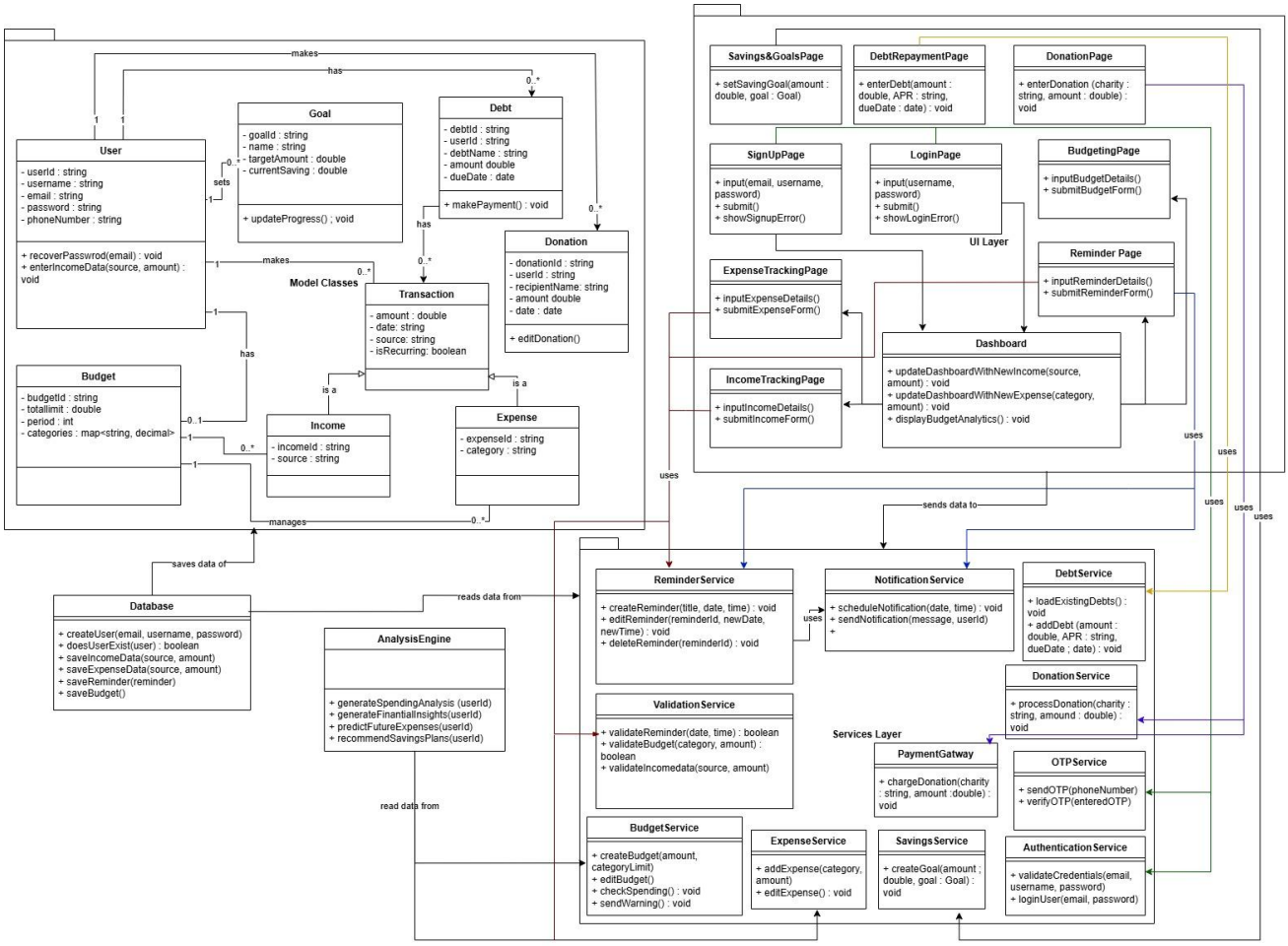
1. **User Service**: Handles user authentication, profiles, and preferences.
2. **Transaction Service**: Manages income and expense tracking, including categorization (e.g., utilities, groceries).
3. **Budget Service**: Manages budget creation, spending limits, and goal tracking (e.g., saving for a home or car).
4. **Reporting Service**: Generates visual reports, charts, and insights on spending trends and goal progress.
5. **Bank Integration Service**: Syncs with bank accounts, e-wallets, or credit cards for automated transaction imports.
6. **Frontend (UI)**: A web-based interface for users to interact with the system.
7. **API Gateway**: Acts as a single entry point for the frontend to communicate with the microservices.
8. **Database(s)**: Each microservice has its own database for independence (e.g., PostgreSQL for transactions, MongoDB for reporting).

Software Design Specification



Software Design Specification

II. Class Diagram(s)



III. Class Descriptions

Class ID	Class Name	Description & Responsibility
1.	Database	Manages the storage and retrieval of users, budget, incomes, and expenses data
2.	AnlysisEngine	Analyzes the spending, generates financial insights, and predicts future expenses using past data.
3.	ReminderService	Allows users to create, edit, and delete financial reminders for payments, savings, and other events.
4.	NotificationService	Schedules and sends notifications to users about reminders, budget warnings, or goal progress.

Software Design Specification

Class ID	Class Name	Description & Responsibility
5.	BudgetService	Allows users to create, update, and manage their budgets and spending limits.
6.	OTPService	Sends and verifies One-Time Passwords (OTP) for phone number verification during signup or recovery.
7.	AuthenticationService	Handles user authentication, including validating credentials, logging in users, and managing sessions.
8.	ExpenseService	Manages adding, editing, deleting, and categorizing expenses.
9.	ValidationService	Ensures that reminders, budgets, incomes, expenses, and user inputs are valid and meet defined rules.
10.	User	Represents a user entity containing information like userId, username, email, phone number, and credentials.
11.	Budget	Represents the user's budget details, including overall budget and specific category allocations.
12.	Goal	Represents financial goals set by the user, such as saving for Umrah, vacations, or major purchases.
13.	Transaction	Represents any financial transaction (income or expense) made by the user, with date, amount, and category.
14.	Income	Contains information about the user's income sources, amounts, dates, and whether they are recurring.
15.	Expense	Contain details of user expenses, including category, amount, date, and recurrence status.
16.	Debt	Represents the user's debts, including debt name, total amount, remaining balance, and repayment history.
17.	Donation	Represents user donation entries, including recipient name, amount, and date of donation.
18.	SignUpPage	UI screen for users to register by providing email, username, password, and phone number.
19.	LoginPage	UI screen for users to enter credentials and access their personalized dashboard.
20.	BudgetingPage	UI screen where users can set up, edit, and view their budgets and budget recommendations.
21.	ExpenseTrackingPage	UI screen for users to add, view, and manage their expenses by category.
22.	IncomeTrackingPage	UI screen for users to add, view, and manage their income sources and amounts.
23.	RemindetPage	UI screen where users create, view, and manage their financial reminders.

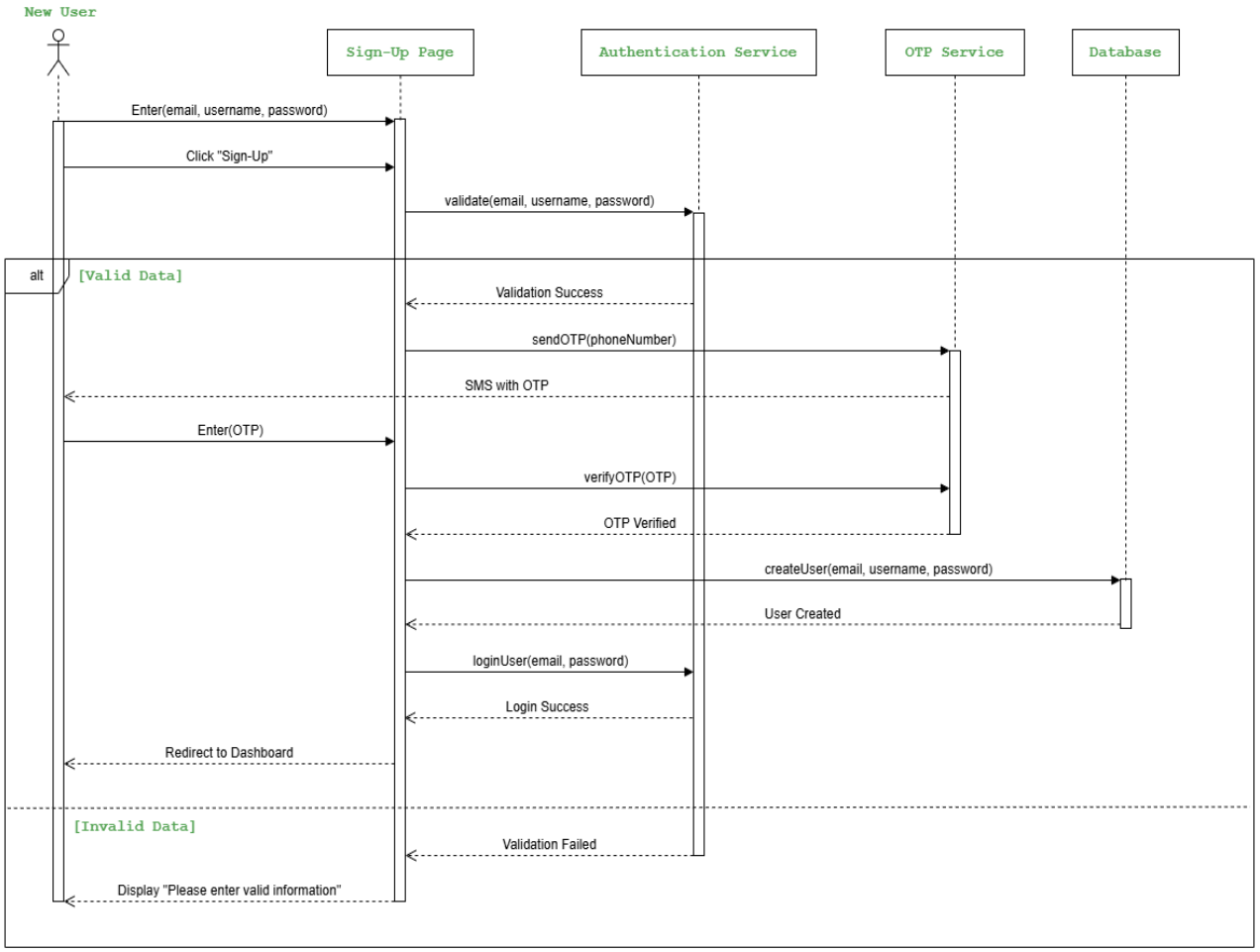
Software Design Specification

Class ID	Class Name	Description & Responsibility
24.	Dashboard	Central UI screens show an overview of the user's financial information, like budget, goals, debts, and reports.

IV. Sequence diagrams

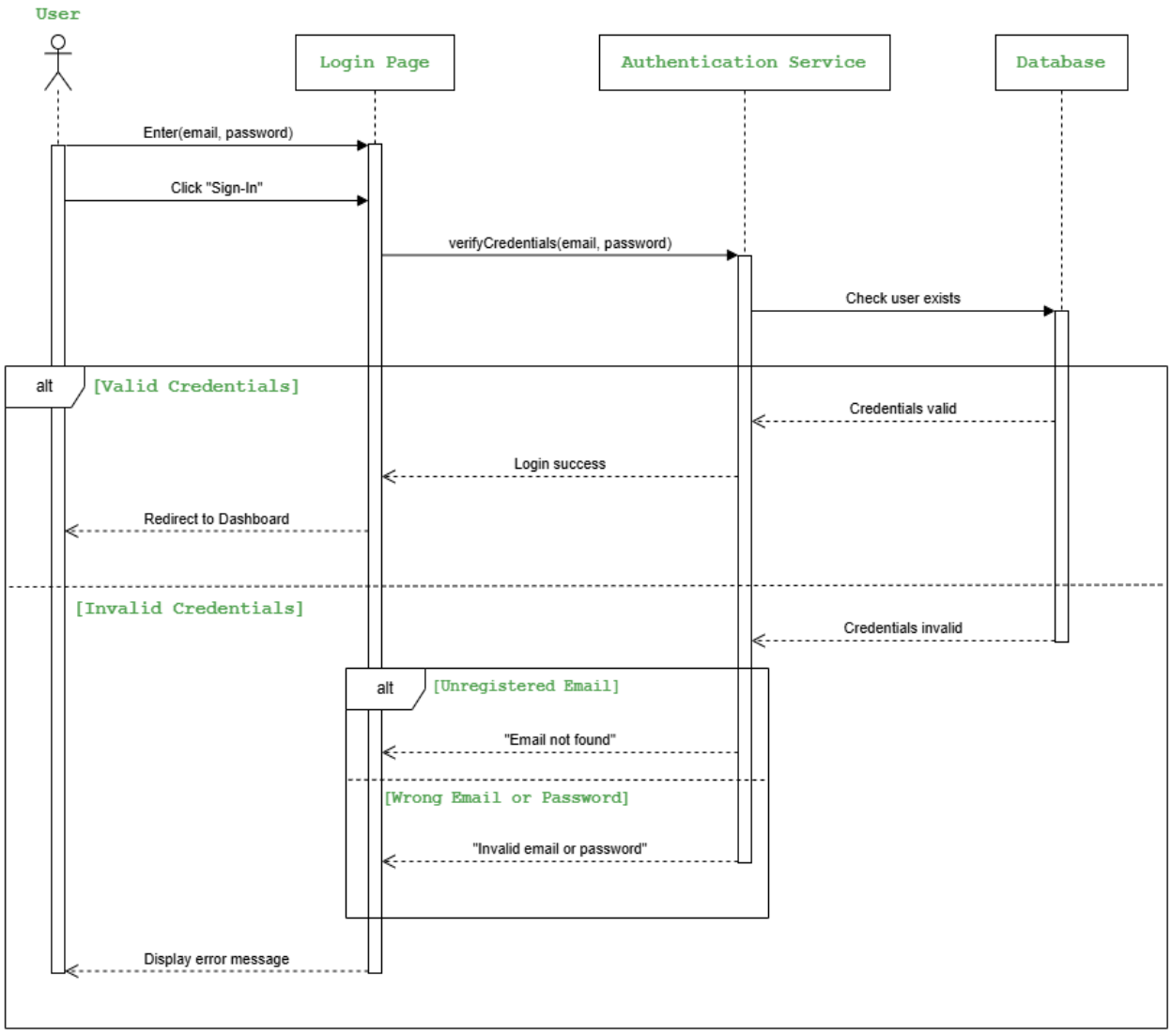
Sequence diagrams in better quality: https://drive.google.com/drive/folders/1il4-EgqYiLz4hqv9t3zVask5WMfIUKYG?usp=drive_link

1. Sign-Up



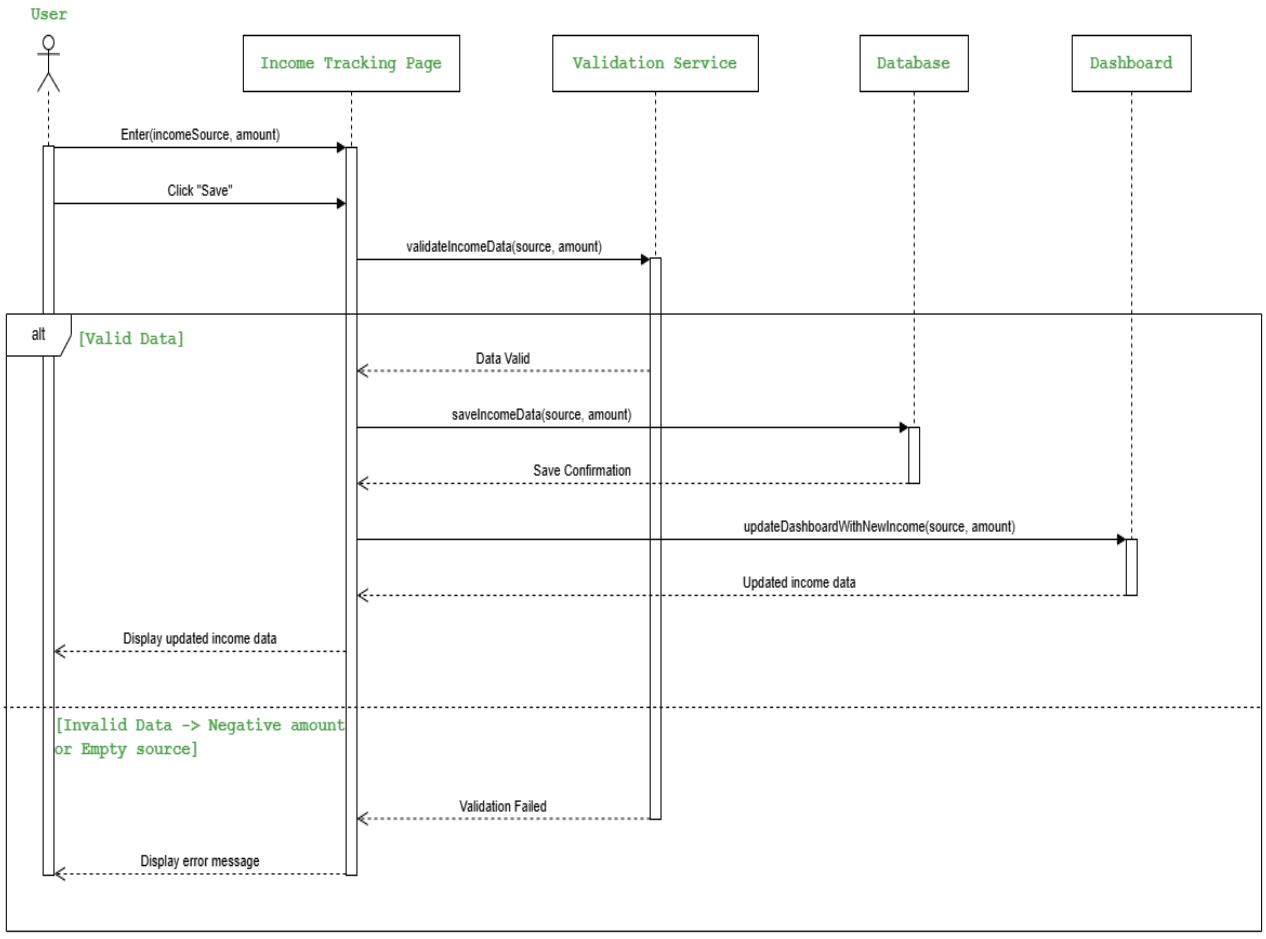
Software Design Specification

2. Login



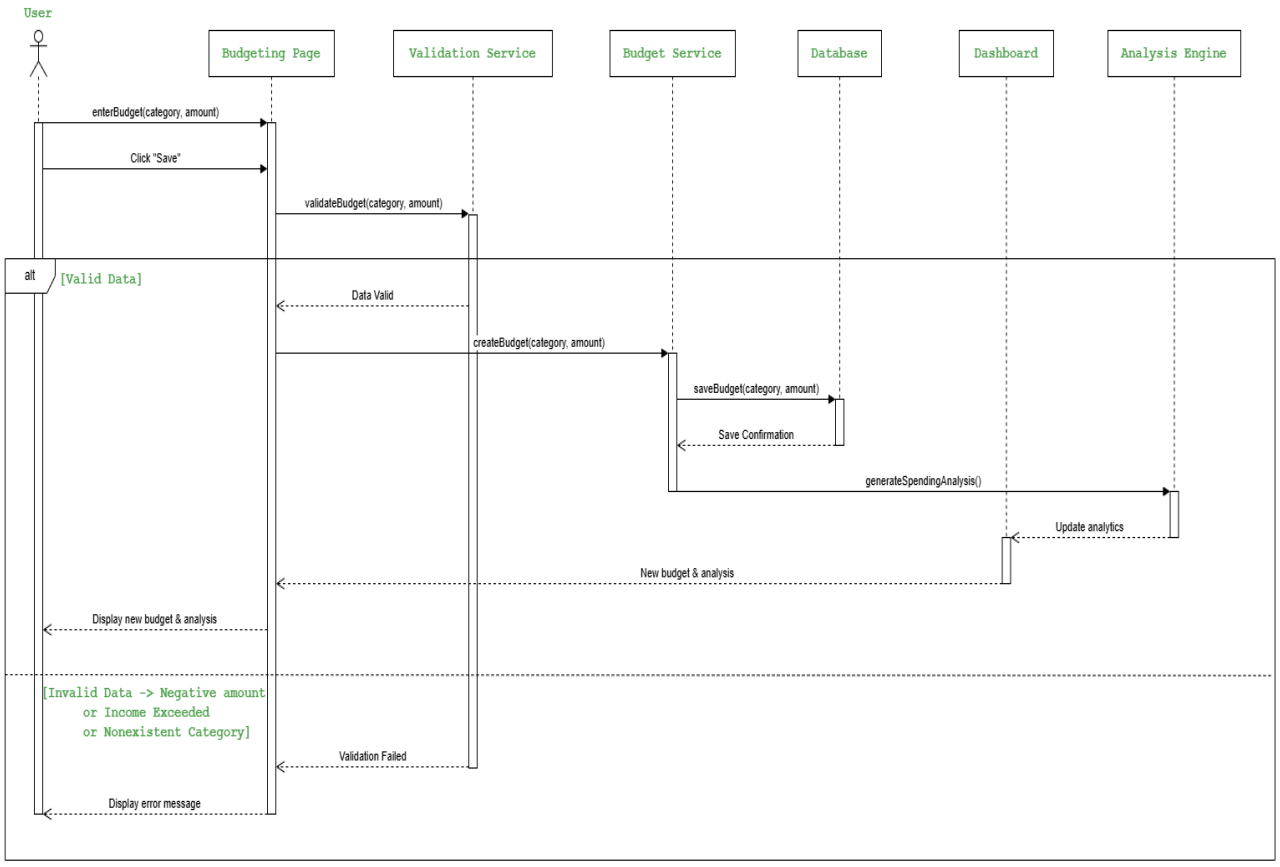
Software Design Specification

3. Tracking Income



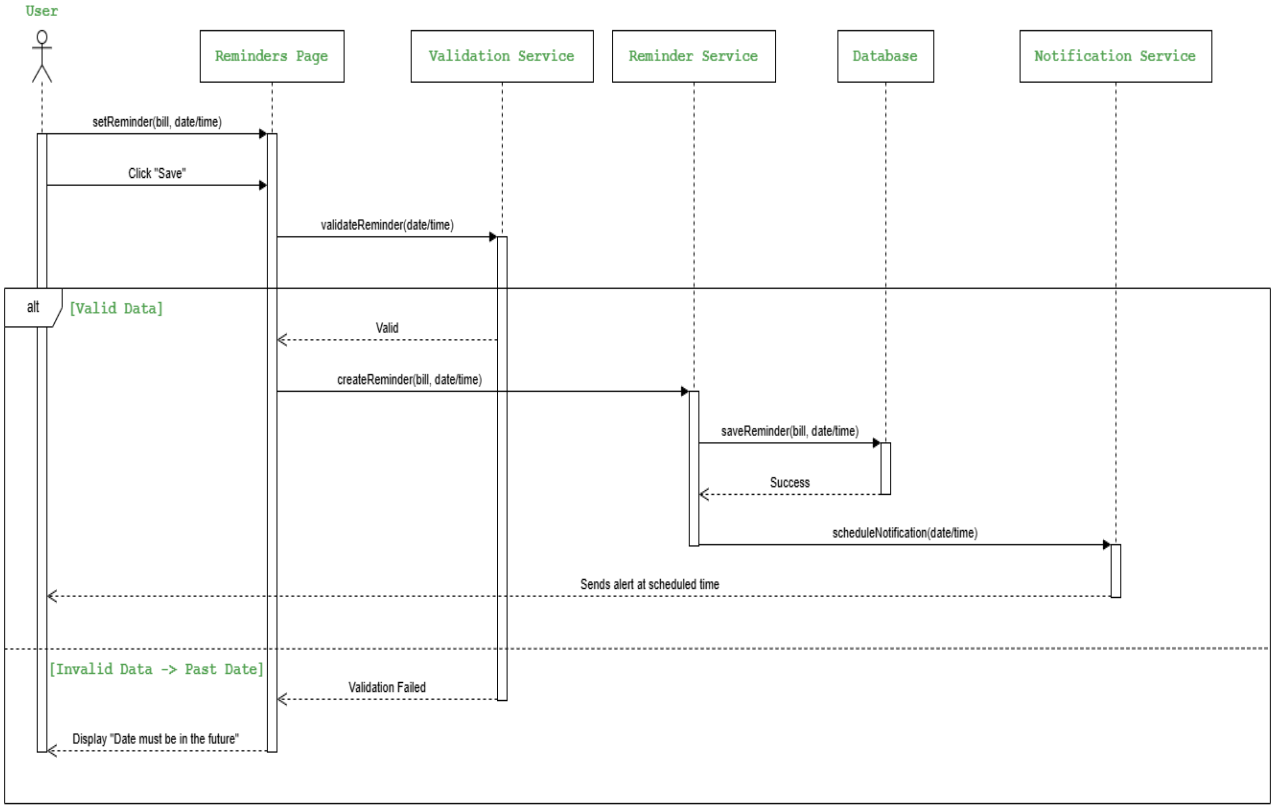
Software Design Specification

4. Budgeting & Analysis



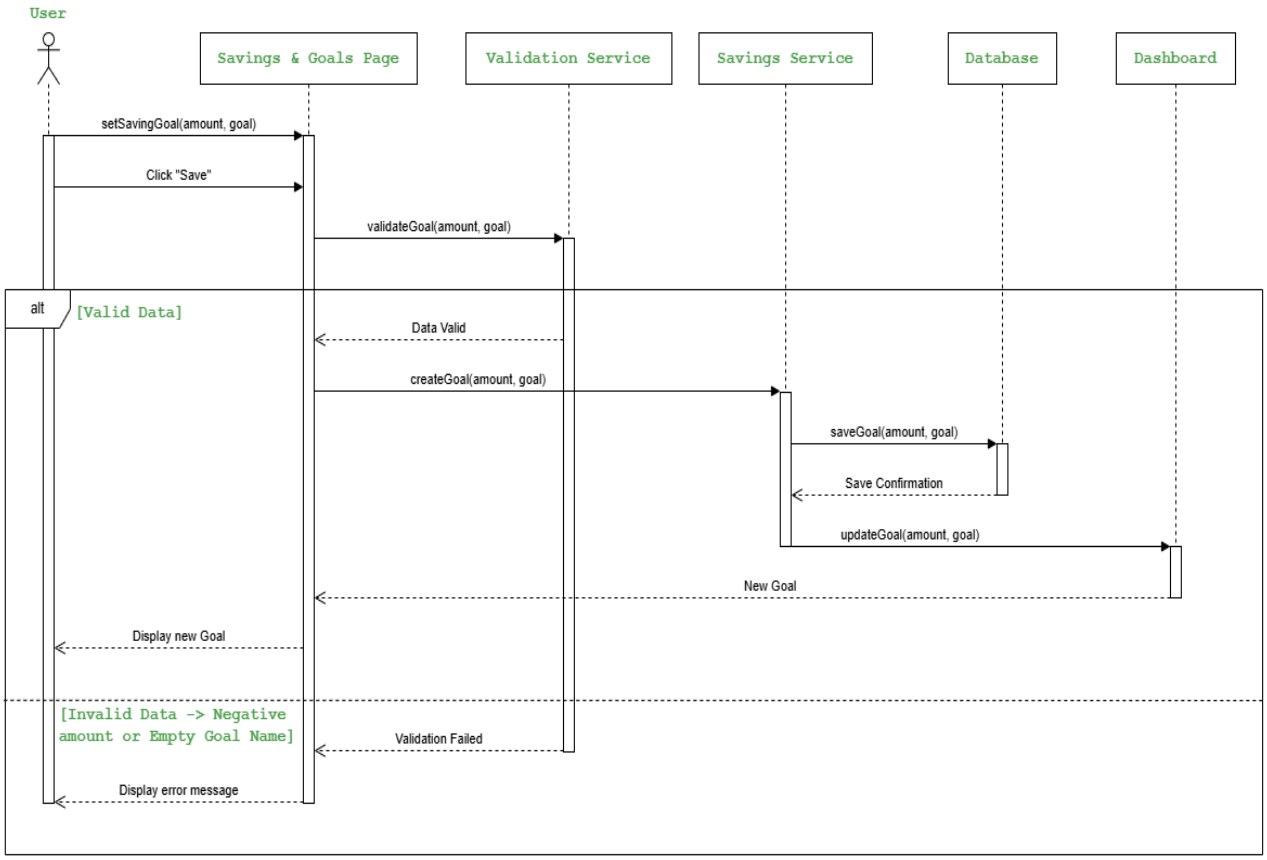
Software Design Specification

5. Reminders



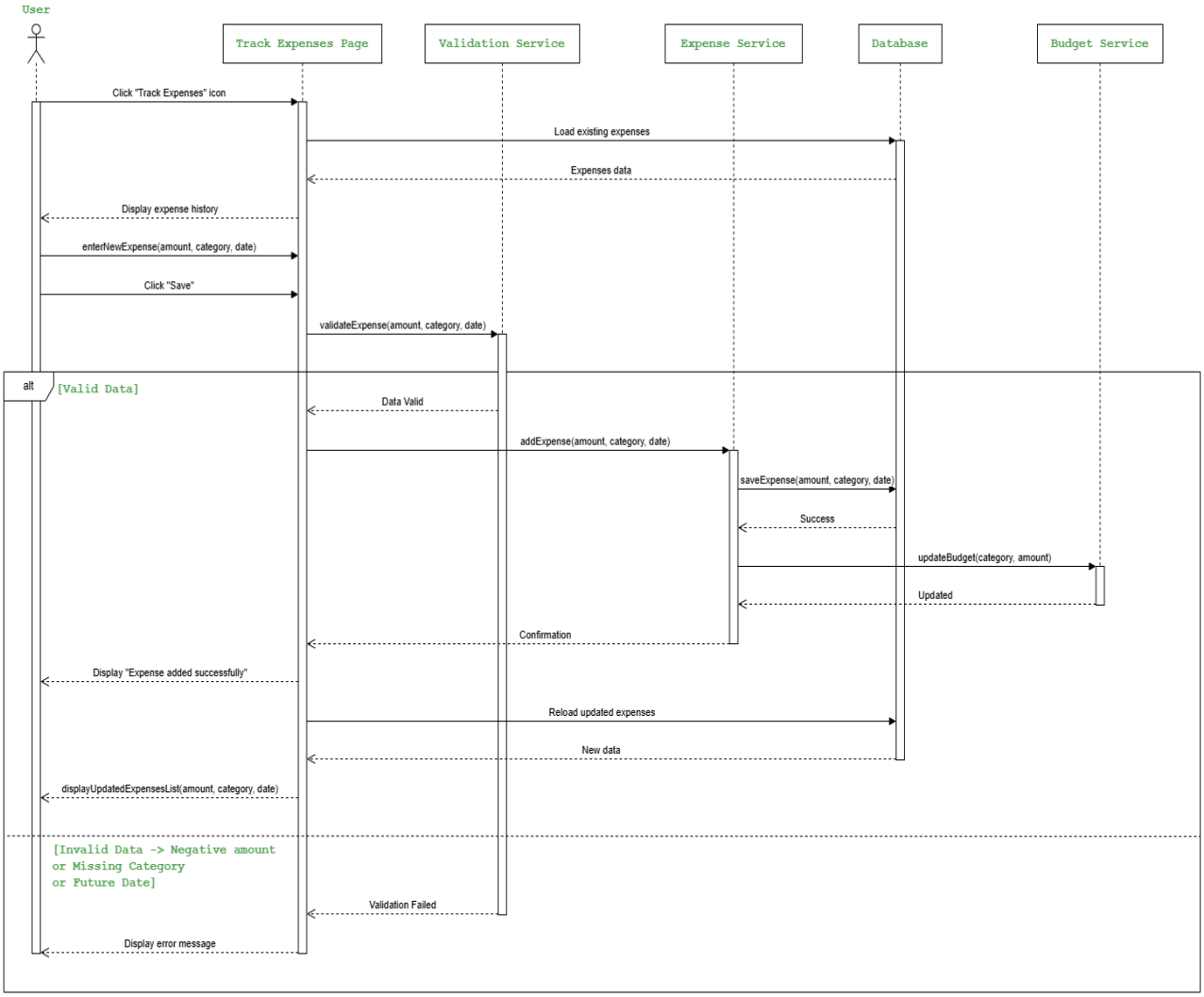
Software Design Specification

6. Savings & Goals



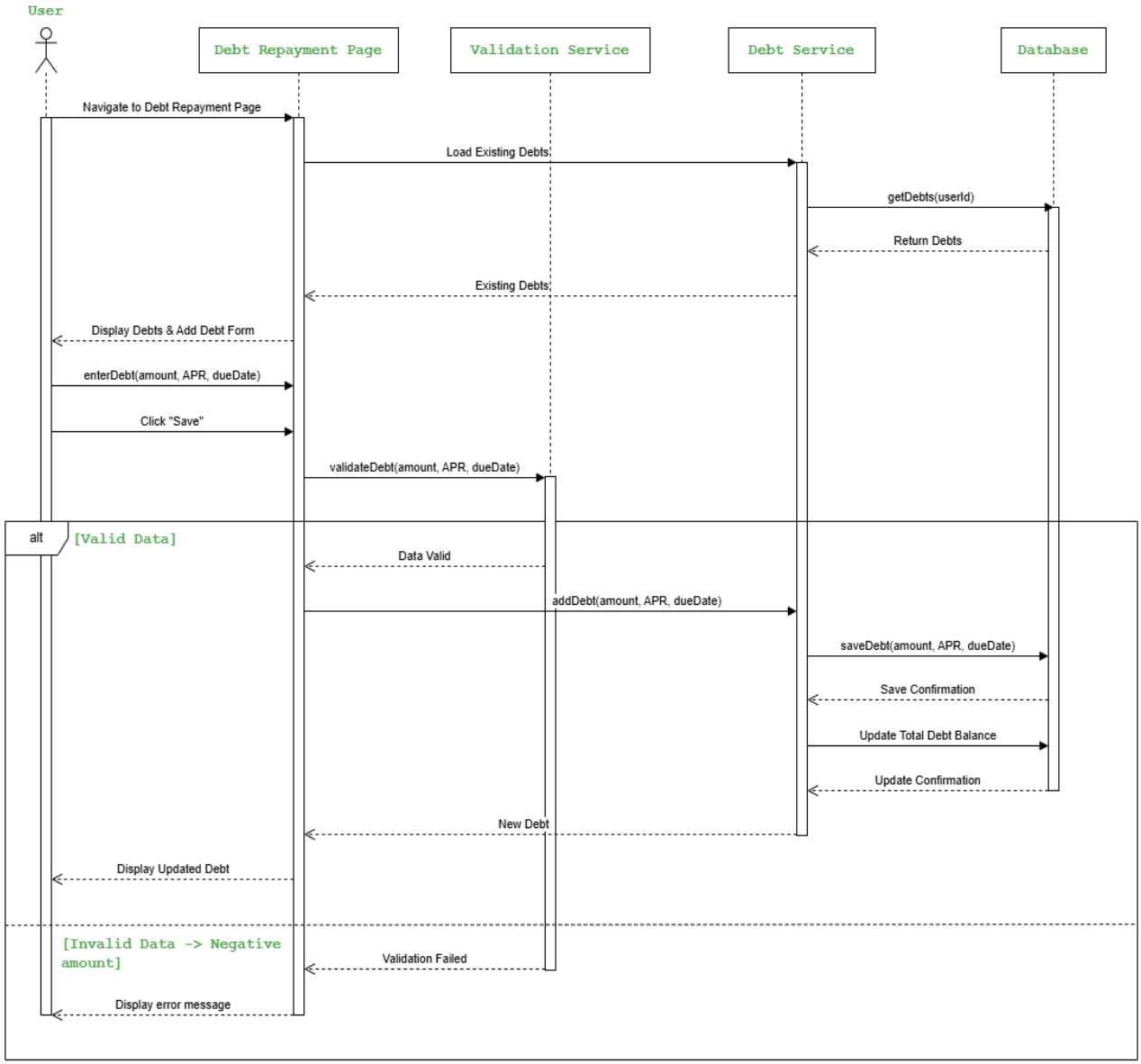
Software Design Specification

7. Expense Tracking



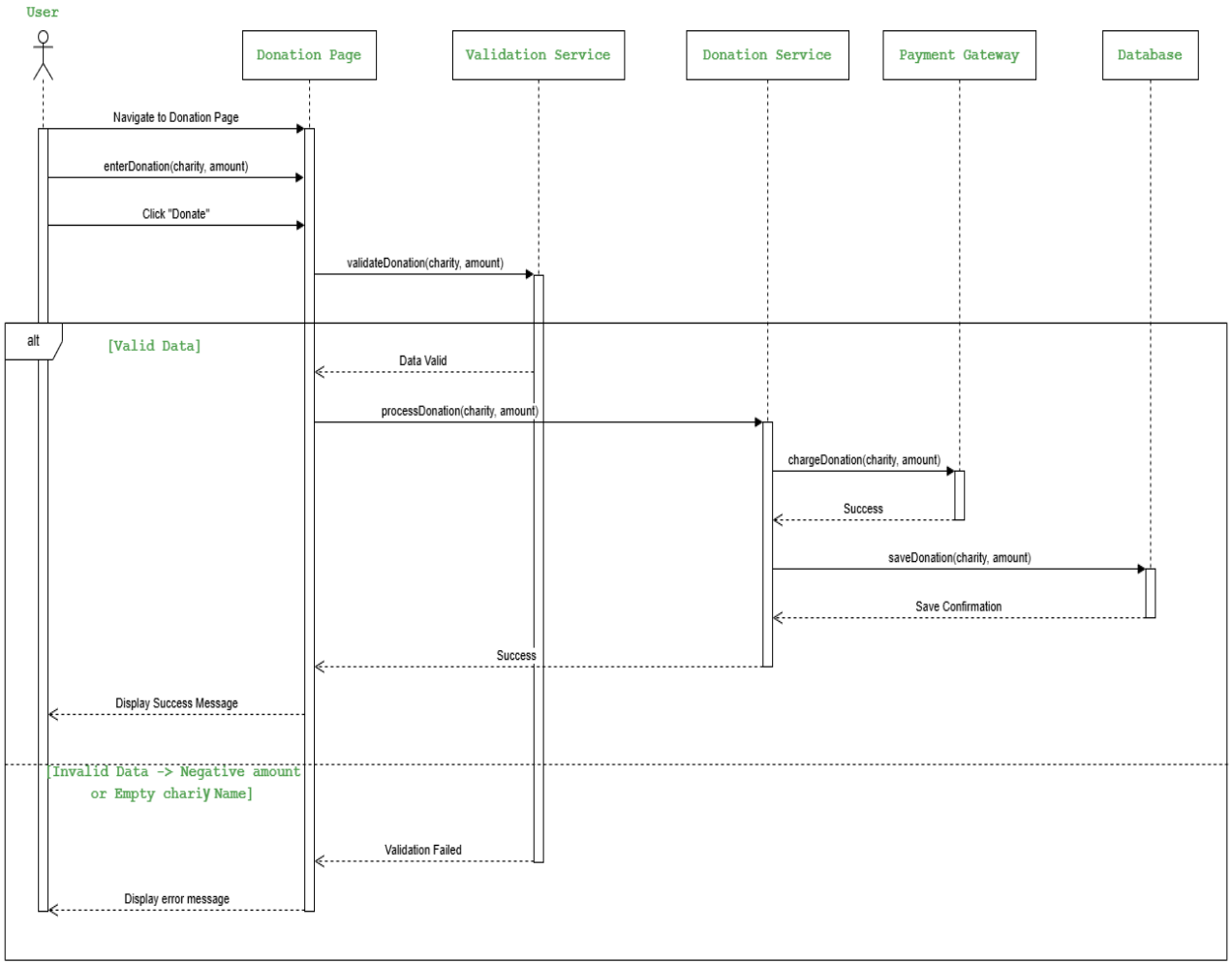
Software Design Specification

8. Debt Repayment



Software Design Specification

9. Donate



Class - Sequence Usage Table

Sequence Diagram	Classes Used	All Methods Used

Software Design Specification

Sequence Diagram	Classes Used	All Methods Used
Sign-Up	Sign-Up Page Authentication Service OTP Service Database	validate(email, username, password) sendOTP(phoneNumber) verifyOTP(OTP) createUser(email, username, password) loginUser(email, password)
Login	Login Page Authentication Service Database	verifyCredentials(email, password) checkUserExists() loginSuccess()
Tracking Income	Income Tracking Page Validation Service Database Dashboard	validateIncomeData(source, amount) saveIncomeData(source, amount) updateDashboardWithNewIncome(source, amount) displayUpdatedIncomeData()
Budgeting & Analysis	Budgeting Page Validation Service Budget Service Database Dashboard Analysis Engine	validateBudget(category, amount) createBudget(category, amount) saveBudget(category, amount) generateSpendingAnalysis() updateAnalytics() displayNewBudgetAndAnalytics()
Reminders	Reminders Page Validation Service Reminder Service Database Notification Service	validateReminder(date/time) createReminder(bill, date/time) saveReminder(bill, date/time) scheduleNotification(date/time) success()

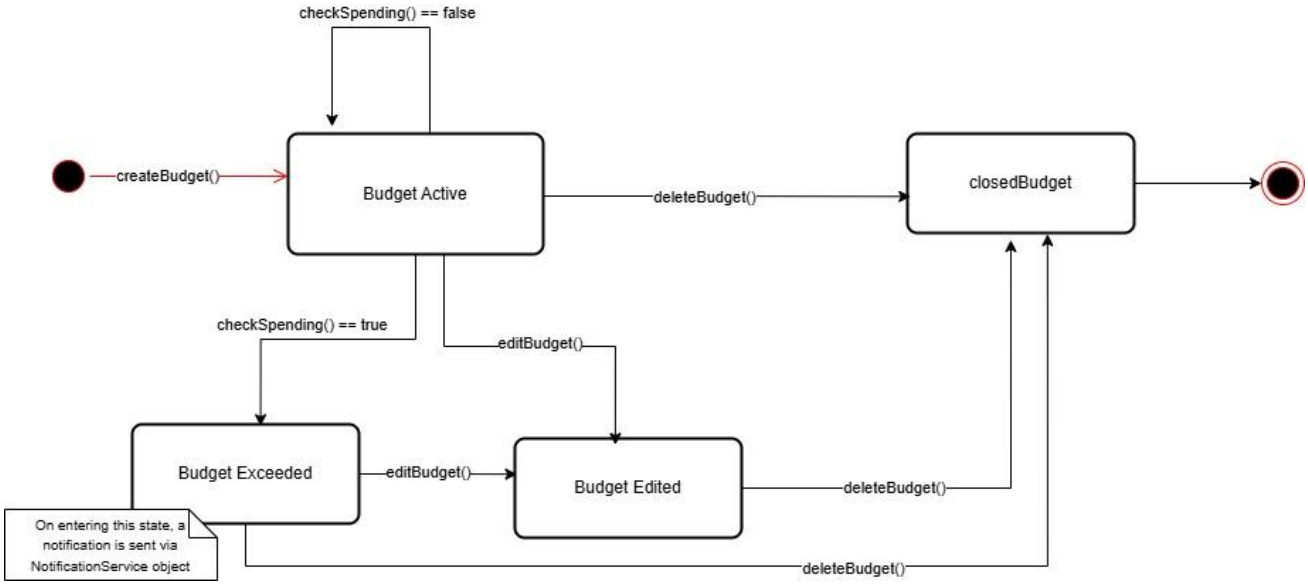
Software Design Specification

Sequence Diagram	Classes Used	All Methods Used
Savings & Goals	Savings & Goals Page Validation Service Savings Service Database Dashboard	validateGoal(amount, goal) createGoal(amount, goal) saveGoal(amount, goal) updateGoal(amount, goal) displayNewGoal()
Expense Tracking	Track Expenses Page Validation Service Expense Service Database Budget Service	validateExpense(amount, category, date) addExpense(amount, category, date) saveExpense(amount, category, date) updateBudget(category, amount) confirmation() expenseAddedSuccessfully() reloadUpdatedExpenses() displayUpdatedExpenseList(amount, category, date)
Debt Repayment	Debt Repayment Page Validation Service Debt Service Database	getDebts(userId) validateDebt(amount, APR, dueDate) addDebt(amount, APR, dueDate) saveDebt(amount, APR, dueDate) UpdateTotalDebtBalance() DisplayUpdatedDebt()
Donate	Donation Page Validation Service Donation Service Payment Gateway Database	validateDonation(charity, amount) chargeDonation(charity, amount) saveDonation(charity, amount) success()

Software Design Specification

V. State Diagram

State Diagram for the Budget Object.



VI. SOLID Principles:

1. Single Responsibility Principle (SRP):

- Each part of the system should focus on doing one thing well, so it only needs to change for one specific reason.
- **How It's Applied:**
  - The AuthenticationService class in the Service Layer is built to handle only user authentication duties. It contains methods such as validateCredentials(email, username, password) to check if user details are correct and loginUser(email, password) to manage the login process.
  - It doesn't get involved in other tasks like creating budgets, scheduling reminders, or analyzing spending, which are responsibilities of classes like BudgetService, ReminderService, and AnalysisEngine.

2. Open/Closed Principle (OCP)

- A class should be designed so you can add new features by extending it, without needing to modify its existing code.

- **How It's Applied:**
  - The Transaction class in the Model Classes is set up to allow new types of transactions to be added without changing its core structure. It acts as a parent class with attributes like amount, date, and source, and is extended by Income and Expense classes through inheritance.

  - If We need to introduce a new transaction type, such as a Refund for returned purchases, We can create a new class that inherits from Transaction without altering the existing Transaction, Income, or Expense classes.

3. Interface Segregation Principle (ISP)

- A class shouldn't be forced to depend on methods or features it doesn't need; it should only interact with the specific functionality it requires.
- **How It's Applied**:
  - The ValidationService class in the Service Layer provides distinct validation methods for different purposes, such as validateReminder(date, time) for checking reminder details and validateBudget(category, amount) for verifying budget information.
  - The ReminderService uses validateReminder(date, time) to ensure a reminder's date and time are valid, while the BudgetService uses validateBudget(category, amount) to check budget details. Neither class is required to interact with validation methods that are irrelevant to its purpose.

Software Design Specification

VII. Design Patterns:

1. **MVC Pattern**: Applied in the [Transaction Service], with Transaction and Expense as the Model, ExpenseService as the Controller, and ExpenseTrackingPage (in [User (Mobile)]) as the View. This separation improves maintainability and scalability by isolating data, logic, and presentation.

2. **Observer Pattern**: Applied between the Budget class (subject) and Dashboard class (observer), allowing the dashboard to update automatically when budget data changes, enhancing user experience and decoupling the budget logic from the UI.

3. **Singleton Pattern:** Applied to the AuthenticationService and UserDatabase classes, ensuring a single instance manages all authentication operations and user data across the application. This centralizes authentication logic and data access, prevents inconsistencies from multiple instances,

Tools

- **Draw.io**

Software Design Specification

Ownership Report

Item	Owners
Salsabil Bahaa Eldin	Class Diagram, Class Responsibility table, Document purpose, and audience, state diagram. Implementation of Sign Up, Income tracking user stories.
Ola Ghoneim	Architecture diagram, Solid Principles, Design Patterns, state diagram. Implementation of Login, Budget tracking user stories.
Mariem Refaey	Main task: 6 Sequence Diagrams, Implementation of Reminders, Expense Tracking **Bonus task:** 3 more sequence Diagrams for 3 user stories with their implementation (Debt, Donation, SavingGoals).