Cairo University

Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

Personal Budgeting

Software Design Specifications

Version 1.0

Team Names and Emails

ID	Name	Email	Mobile
20231090	Anoud Mohamed Ahmed	anoudmohammed65@gmail.com	01097211041
20231196	Nayera Shabaan Rashad	Nayerashaaban54@gmail.com	0115 299 2141
20231119	Fatima Mossad Eid	fatimamosaad1@gmail.com	0112 310 2709

April - 2025

CS251:

Project: Budget Management App

Software Design Specification

Contents

CS251:

Project: Budget Management App

Software Design Specification

Document Purpose and Audience

1. **Document Purpose**

   This (SDS) establishes the architectural framework and detailed technical design for the Personal Budget Application. It translates business requirements into a comprehensive technical blueprint that will guide the development team throughout the implementation process. The document specifies the system architecture, component interactions, user interfaces, and security measures necessary to deliver a robust and user-friendly budget management solution. By documenting these specifications, we ensure alignment across all stakeholders, provide clear direction for developers, and establish verifiable criteria for testing and validation. This SDS serves as the authoritative reference for technical decisions during development and will support future maintenance and enhancement efforts.

2. **Target Audience**
   - **Project Manager**
   - **Software Development Team**
   - **Product Stakeholders**
   - **Client Representative**

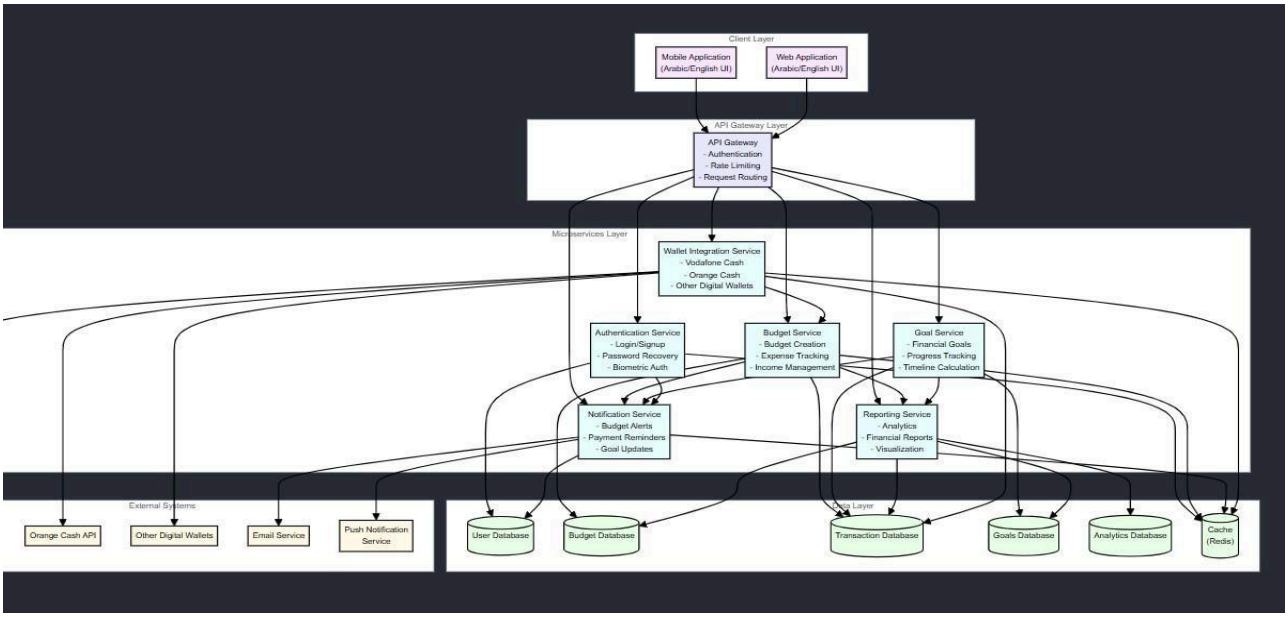
CS251:

Project: Budget Management App

Software Design Specification

System Models

I. Architecture Diagram

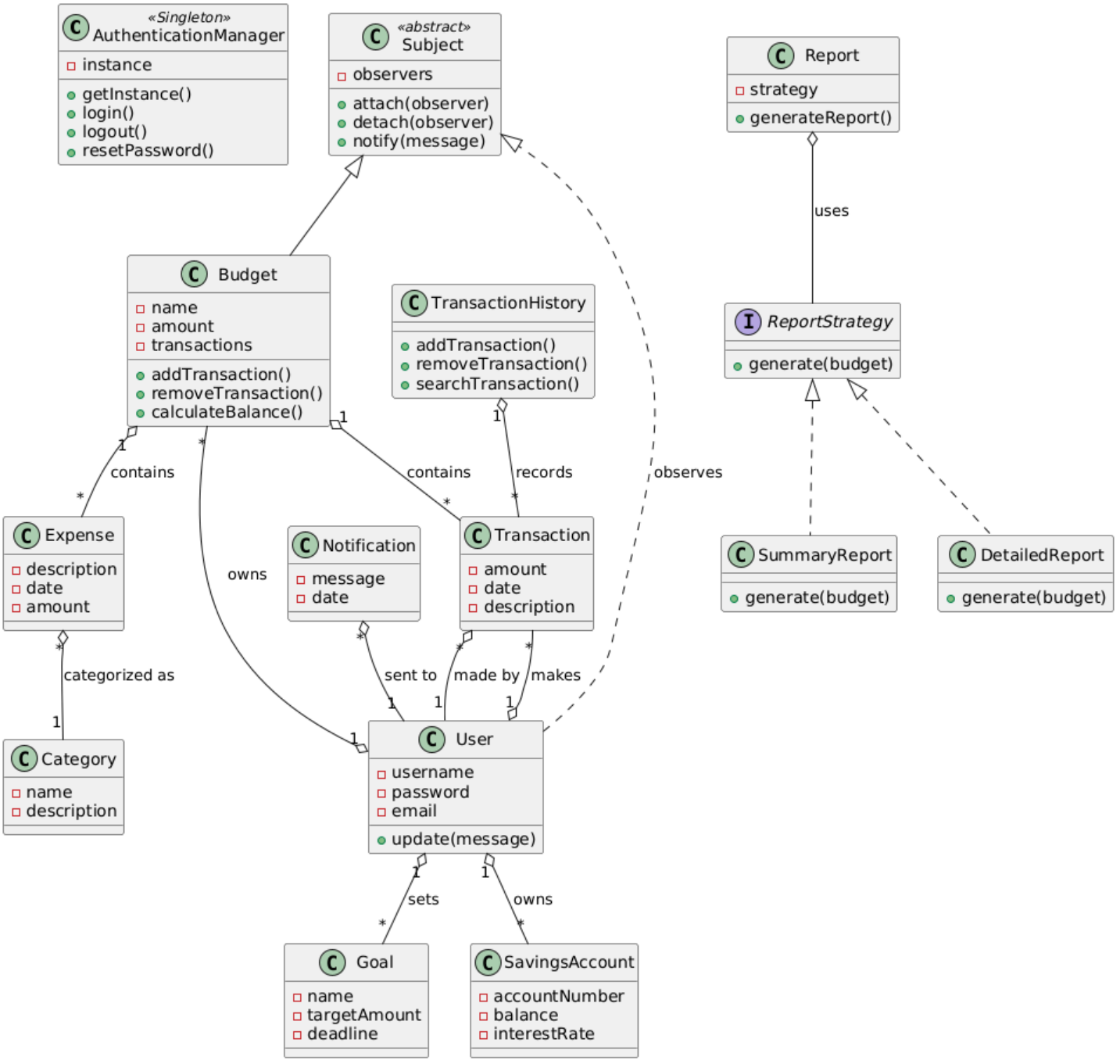


CS251:

Project: Budget Management App

Software Design Specification

II. Class Diagram(s)



CS251:

Project: Budget Management App

Software Design Specification

III. Classes Description

Class Name	Responsibility
AuthenticationManager	Handles user login, logout, and password reset. (Singleton for central access)
User	Represents a user, manages personal data, receives notifications, and owns transactions and goals.
Transaction	Represents a financial transaction with amount, date, and description.
TransactionHistory	Manages transaction records: add, remove, and search.
Budget	Represents a budget, tracks associated transactions and expenses, calculates balance.
Expense	Represents a financial expense with description, amount, and date.
Category	Categorizes expenses for organization and reporting.
Notification	Sends alert messages with a date to the user.
Goal	Represents a savings or financial goal with a target amount and deadline.
SavingsAccount	Represents a savings account with number, balance, and interest rate.
Report	Generates a report using a chosen strategy (ReportStrategy).
ReportStrategy	Interface for report generation strategies.
SummaryReport	Implements a simple, summarized report format.
DetailedReport	Implements a detailed report format.
Subject (abstract)	Observer pattern component to manage and notify observers of state changes.

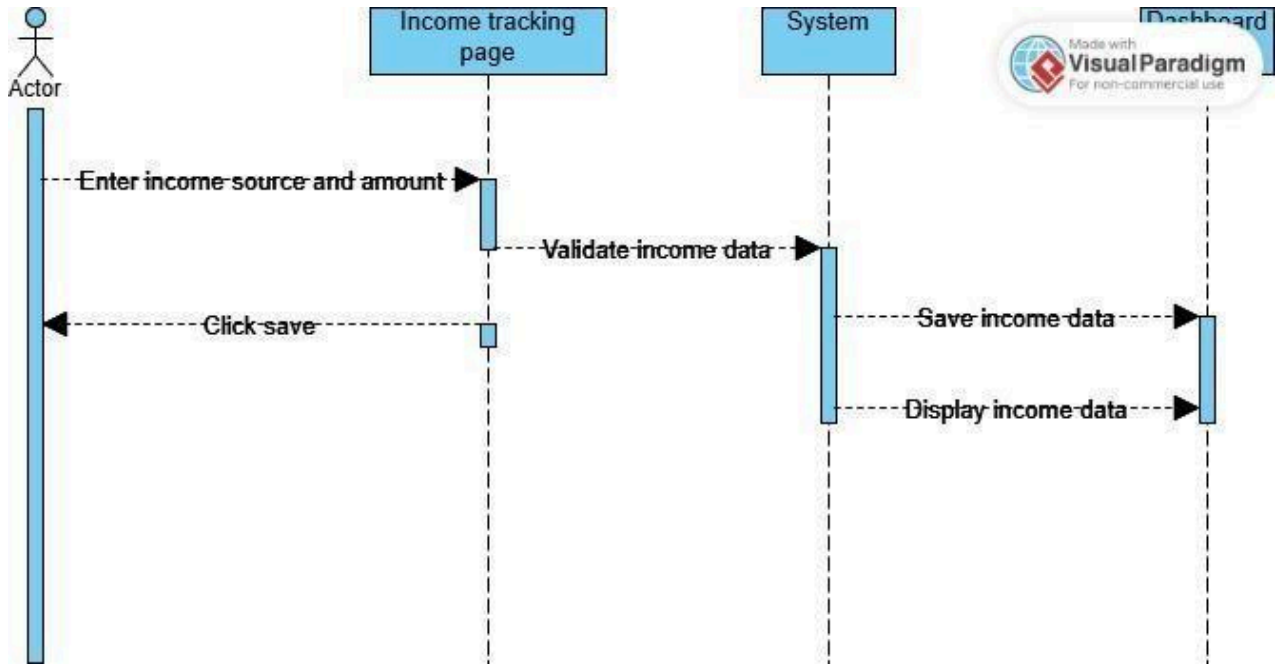
CS251:

Project: Budget Management App

Software Design Specification

IV. Sequence diagrams

1.



Sequence diagrams Class – Sequence Usage Table

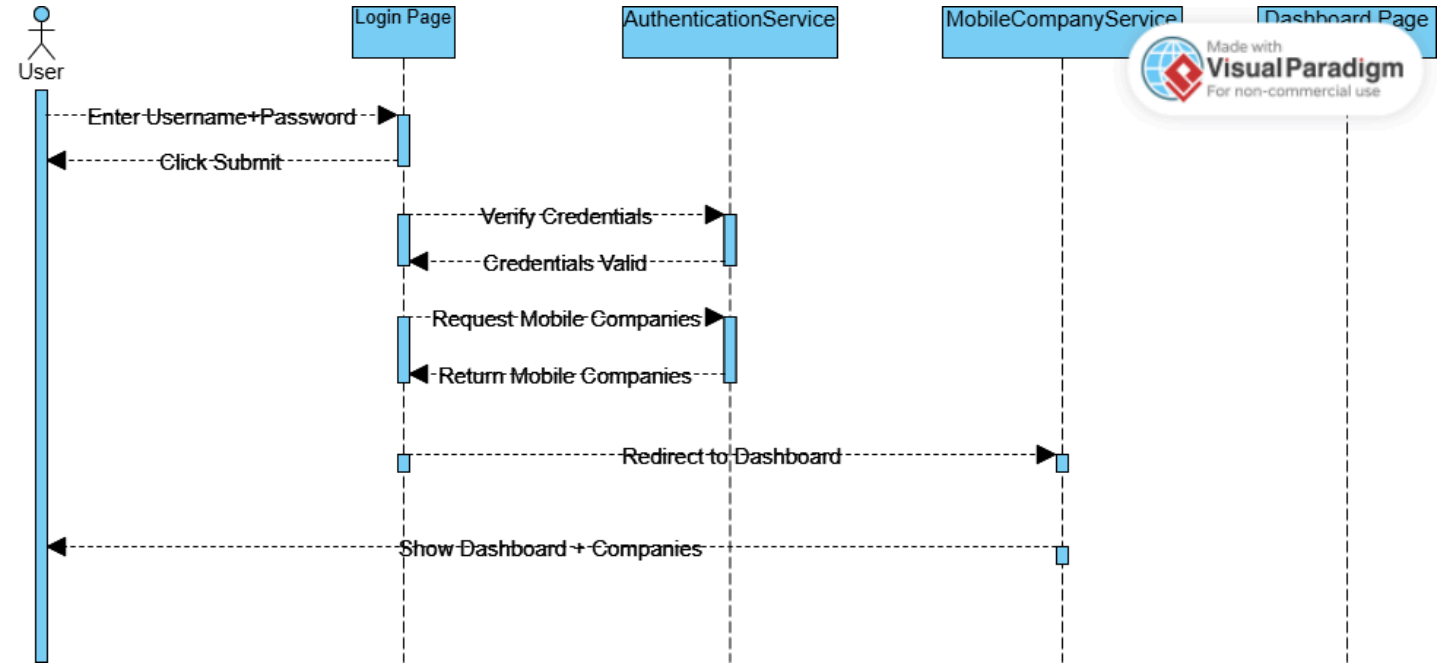
Class	Methods Used
SignUpPage	enterInfo(); clickSignUp(); enterOTP()
ValidationService	validateInfo(); sendOTP(); verifyOTP()
OTPService	sendOTP(); receiveOTPSMS()
AccountService	createAccount()
DashboardPage	redirect()
User	---

CS251:

Project: Budget Management App

Software Design Specification

2-



Sequence diagrams Class – Sequence Usage Table

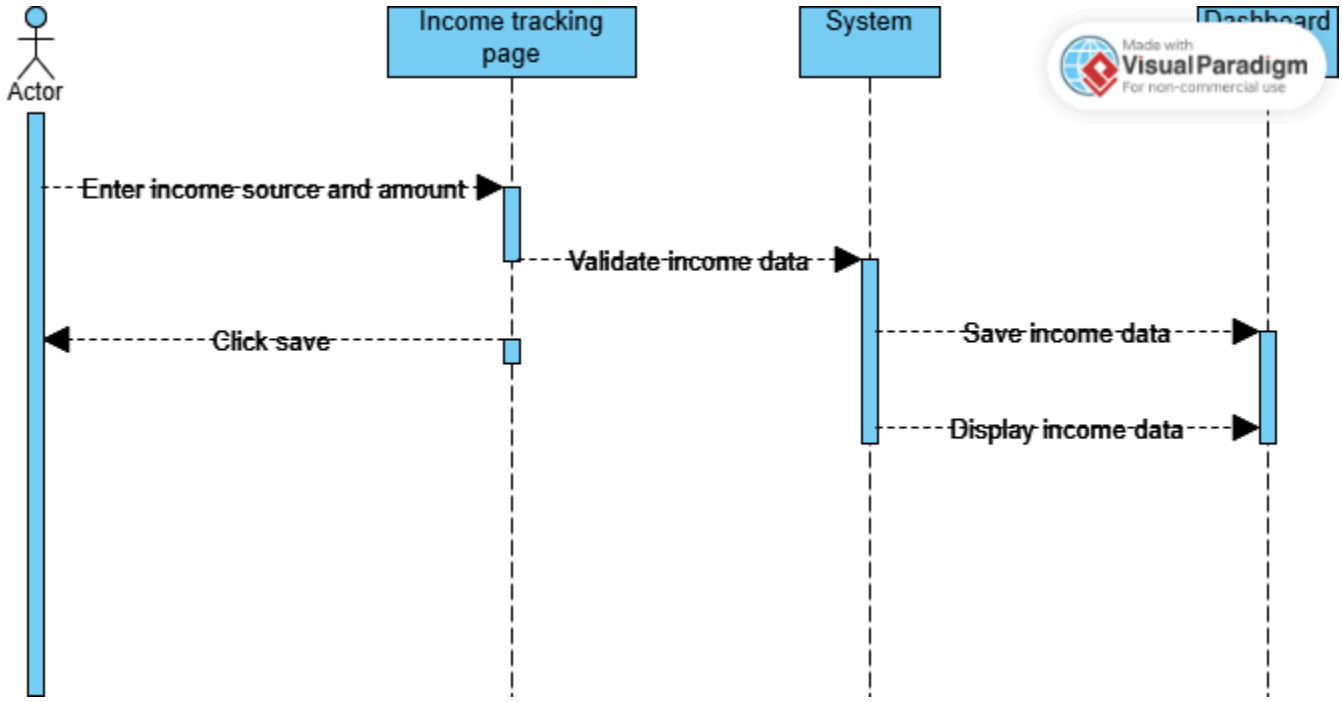
Class	Methods Used
LoginPage	enterCredentials(); clickSubmit()
AuthenticationServices	verifyCredentials(); requestMobileCompanies()
MobileCampanyServices	returnMobileCampanies()
DashbardPage	redirectToDashboard(), showDashboardAndCampanies()
User	---

CS251:

Project: Budget Management App

Software Design Specification

3-



Sequence diagrams Class – Sequence Usage Table

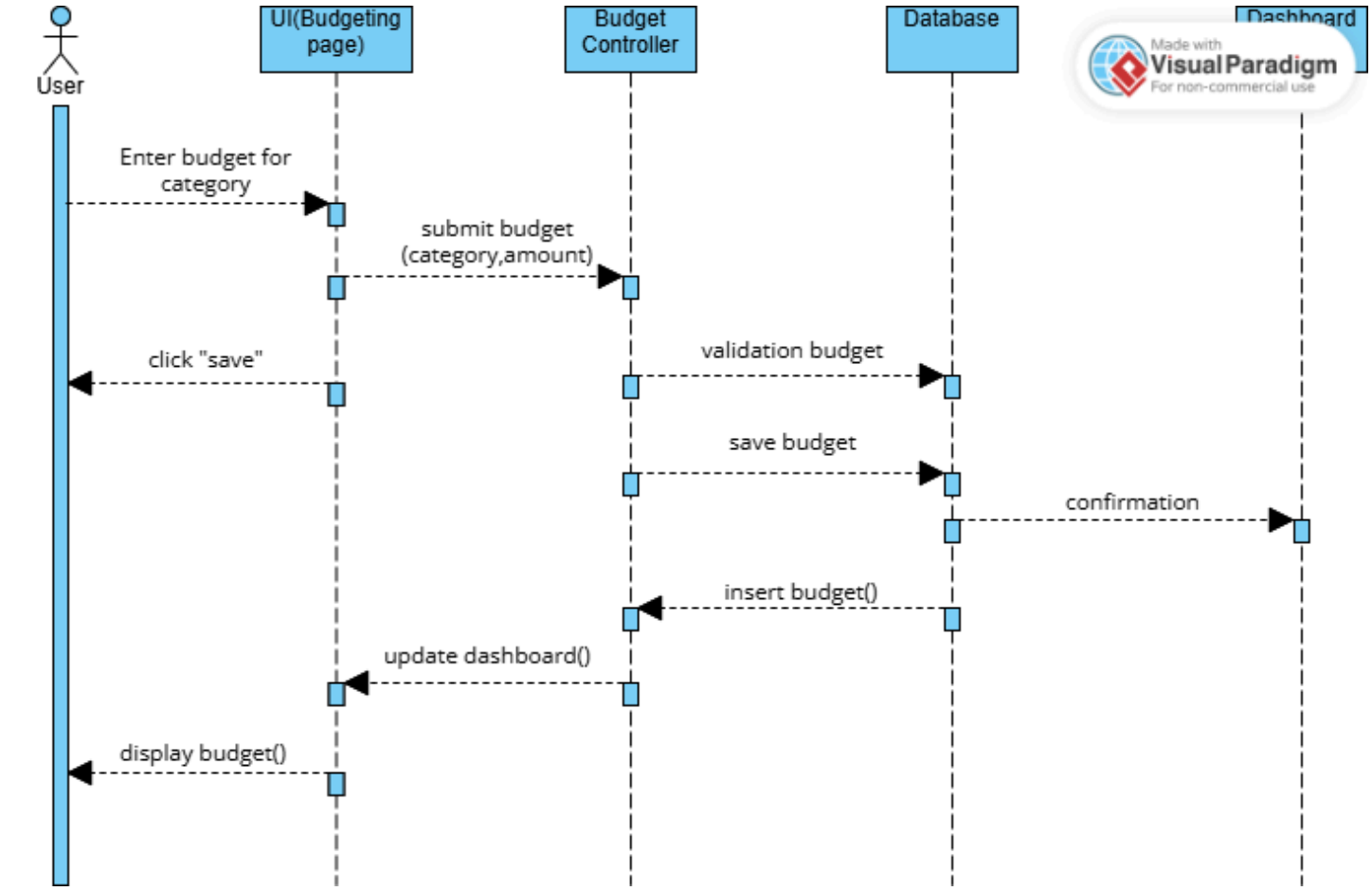
Class	Methods Used
IncomeTrackingPage	enterIncome(source, amount), clickSave()
System	validateIncomeData(), saveIncomeData(), displayIncomeData()
Dashboard	displayIncomeData()
User	---

CS251:

Project: Budget Management App

Software Design Specification

4-



Sequence diagrams Class – Sequence Usage Table

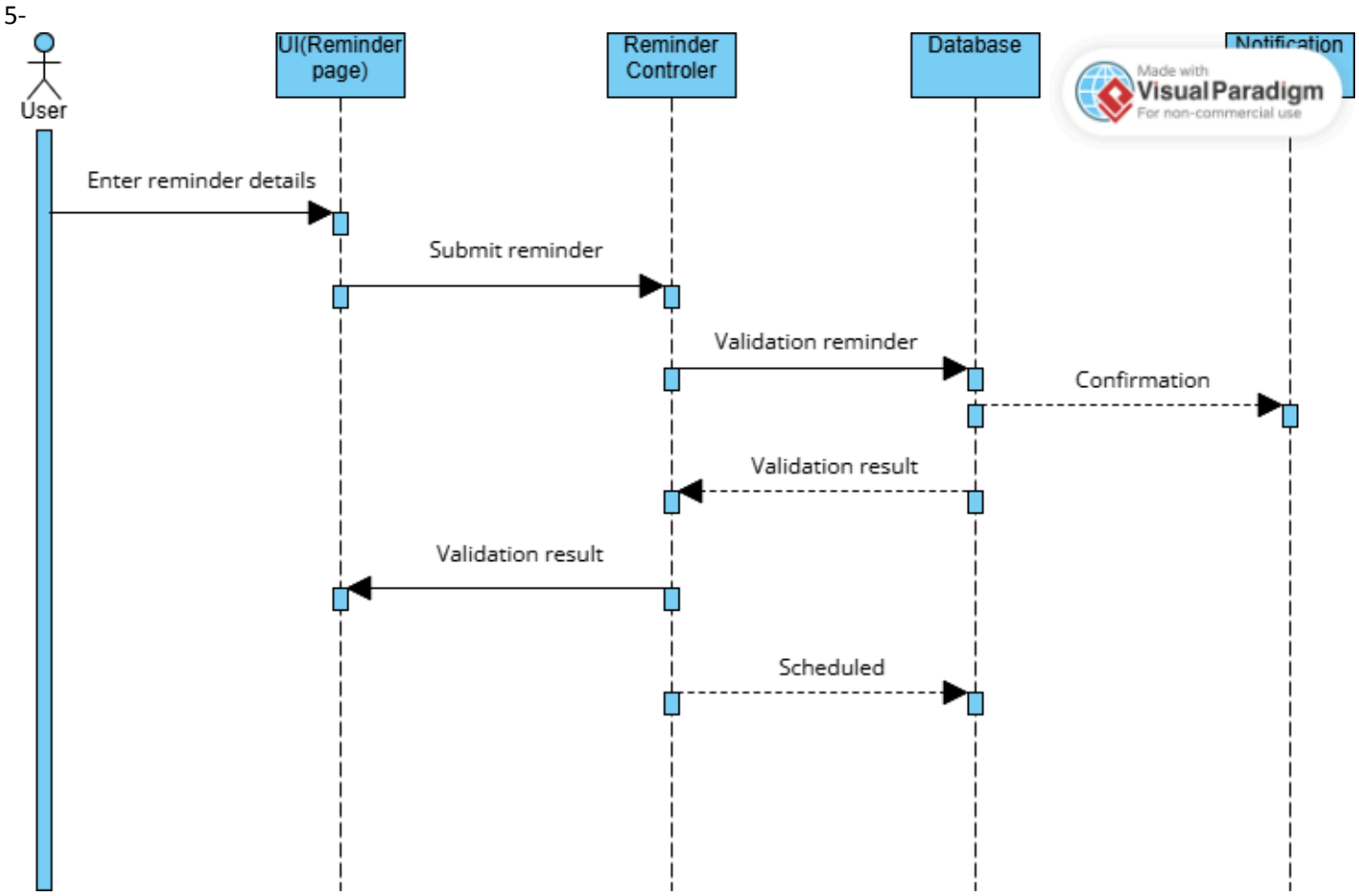
Class	Methods Used
BudgetController	submitBudget(category, amount)
BudgetService	validateBudget(category, amount); saveBudget(userID, category, amount)
Dashboard	displayBudget(category, amount)
Database	insertBudget(userID, category, amount)
UI	---

CS251:

Project: Budget Management App

Software Design Specification

5-



Sequence diagrams Class – Sequence Usage Table

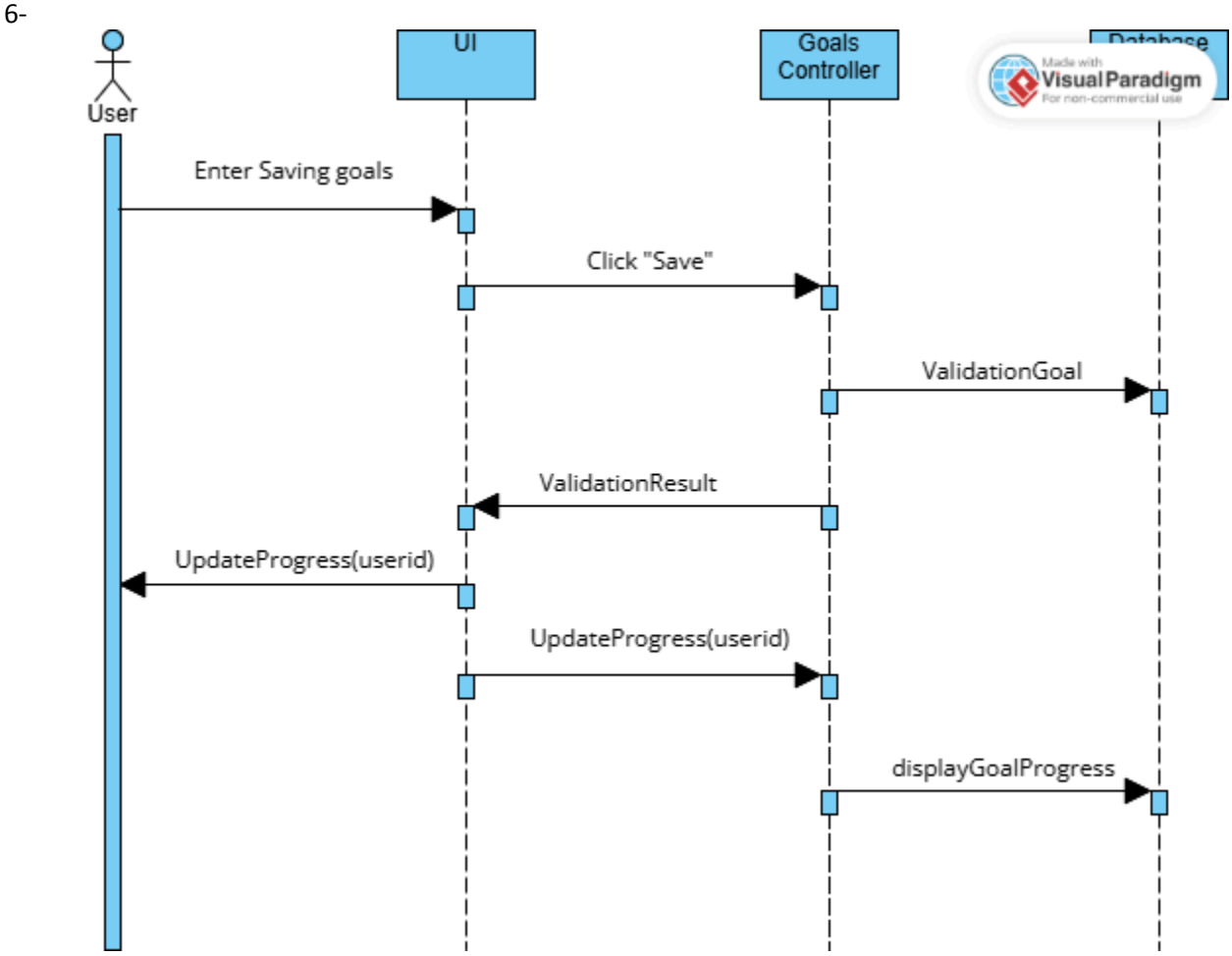
Class	Methods Used
ReminderController	submitReminder(billName, date, time)
ReminderService	validateReminder(billName, date, time); saveReminder(userID, billName, date, time)
NotificationService	scheduleNotification(userID, billName, date, time)
Database	insertReminder(userID, billName, date, time)
UI	---

CS251:

Project: Budget Management App

Software Design Specification

6-



Sequence diagrams Class – Sequence Usage Table

Class	Methods Used
GoalsController	submitGoal(goalName, targetAmount)
GoalsService	validateGoal(goalName, targetAmount); saveGoal(userID, goalName, targetAmount); updateProgress(userID)
Dashboard	displayGoalProgress(goalName, targetAmount, currentAmount)
Database	insertGoal(userID, goalName, targetAmount)
UI	---

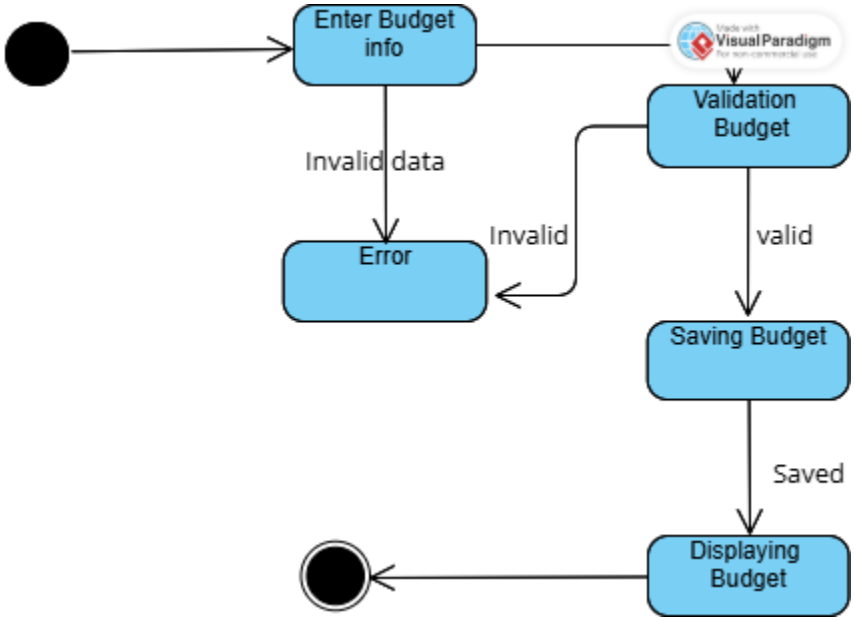
CS251:

Project: Budget Management App

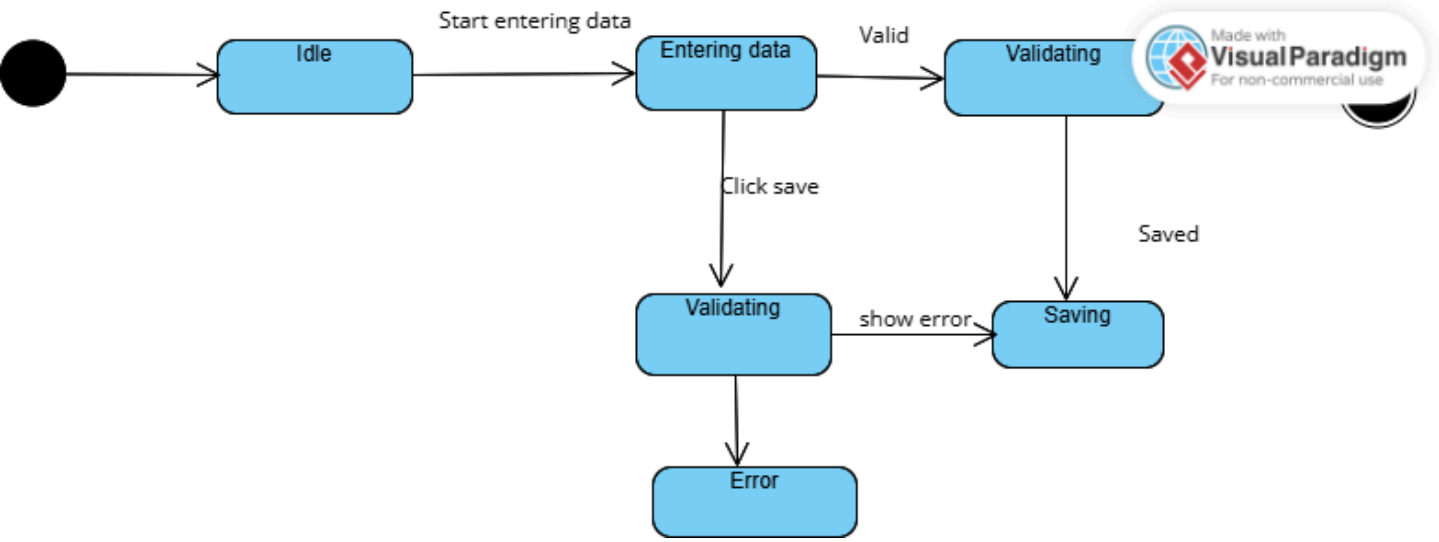
Software Design Specification

V.State Diagram

1-Budget & Analysis



2- Income tracking



CS251:

Project: Budget Management App

Software Design Specification

VI. SOLID Principles

Principle	Description	How It's Applied in the Project
S - Single Responsibility	A class should have only one reason to change.	**Expense** handles expense data only, **Budget** tracks budget allocation, and **FinancialReport** generates reports.
O - Open/Closed Principle	Software entities should be open for extension but closed for modification.	Report uses a **strategy pattern** (ReportStrategy), so we can add new types of reports like GraphicalReport **without modifying existing classes**.
D - Dependency Inversion	High-level modules should not depend on low-level modules, both should depend on abstractions.	The Report class depends on the ReportStrategy **interface**, not on SummaryReport or DetailedReport directly.

Design Patterns

1. Singleton Pattern Implementation

Implemented Class: `AuthenticationManager`

Implementation Details

- **Private static instance attribute**
- **Public static `getInstance()` method for accessing the single instance**
- **Private constructor to prevent direct instantiation**
- **Public methods (`login()`, `logout()`, `resetPassword()`) operate on this single instance**

How It Works

1. **When authentication is needed, the system calls `AuthenticationManager.getInstance()`**
2. **If the instance doesn't exist, it's created; otherwise, the existing instance is returned**
3. **This ensures only one authentication service exists throughout the application**
4. **All authentication requests go through this single point**

Benefits: Centralized authentication management provides consistent security and session

management across the budgeting application.

2. Strategy Pattern Implementation

Implemented Classes: `Report`, `ReportStrategy`, `SummaryReport`, `DetailedReport`

Implementation Details

- `Report` class contains a strategy attribute of type `ReportStrategy`
- `ReportStrategy` is an interface with a `generate(budget)` method
- `SummaryReport` and `DetailedReport` are concrete implementations of `ReportStrategy`
- Each report type implements its own version of `generate(budget)`

How It Works

1. The `Report` class defers the actual report generation to its strategy object
2. Client code creates a `Report` and sets its strategy to either `SummaryReport` or `DetailedReport`
3. When `generateReport()` is called, it delegates to the strategy's `generate(budget)` method
4. Different reporting behaviors are possible without modifying the `Report` class

Benefits: Easily add new report types (like `CategoryReport` or `TrendReport`) without changing existing code.

3. Observer Pattern Implementation

Implemented Classes: `Subject` (abstract), `Budget`, `User`, `Notification`

Implementation Details

- Abstract `Subject` class defines the observer management interface:
  - `attach(observer)`
  - `detach(observer)`
  - `notify(message)`
- `Budget` class inherits from `Subject`, acting as a concrete subject
- `User` class acts as the observer with an `update(message)` method

CS251:

Project: Budget Management App

Software Design Specification

- `Notification` class represents the message sent from subject to observer

How It Works

1. When a user sets up budget monitoring, their `User` object is attached as an observer to the `Budget`
2. When budget thresholds are reached or other significant events occur, the `Budget` calls `notify(message)`
3. The notification system then calls `update(message)` on all attached `User` objects
4. The `User` receives the notification about budget status
5. The `TransactionHistory` also appears to observe the `User` for transaction-related notifications

Benefits: Users are kept informed about their budget status without constant manual checking.

CS251:

Project: Budget Management App

Software Design Specification

Tools

1. **draw.io**
2. **Visual Paradigm**

Ownership Report

Item	Owners
Anoud Mohamed	Architecture Diagram Design Patterns
Nayera Shabaan	Sequence Diagrams State Diagram
Fatima Mossad	Class Diagram & Class Description SOLID Principles