



UE22CS352B - Object Oriented Analysis & Design

Mini Project Report

Budget Ease – Finance Tracker

Submitted by:

<i>PESIUG22CS419</i>	<i>Prabhat Deshmukh</i>
<i>PESIUG22CS396</i>	<i>Nishant Banakar</i>
<i>PESIUG22CS430</i>	<i>Pranav Jigalur</i>
<i>PESIUG22CS432</i>	<i>Pranav Prashant Dambal</i>

Semester – 6 Section – G

Faculty Name – Bharghavi Mokashi

January - May 2025

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement:

In today's fast-paced digital world, individuals often struggle to manage their personal finances effectively, leading to missed savings goals, overspending, and a lack of awareness around their financial habits. Existing budgeting tools are often too complex, lack personalization, or fail to integrate key features like real-time notifications, goal tracking, and receipt-based transaction entry.

This project aims to develop "BudgetEase" — a full-stack, user-friendly financial planning application that helps users create and manage budgets, track income and expenses, set financial goals, and receive real-time notifications on important actions. The system also supports optional receipt uploads for expense verification and provides rewards to encourage good financial behaviour.

Key Features:

User Authentication & Profile Management –

- **JWT-based Authentication** with secure login and registration.
- **User Profile Management:**
 - View and update profile info (username, email, phone).
 - Upload profile picture.
 - Change password securely.

Budget Management –

- **Create Budget:**
 - Add title, amount, and timeframe.
- **View Budgets:**
 - See all created budgets in a card-style UI.
 - Includes "View" and "Delete" buttons.
- **Update Budget:**
 - Modify budget details using a prefilled form layout.

Goal Management –

- **Create Goal:**
 - Set financial targets with descriptions.
- **View Goals:**
 - List all user goals in a styled, responsive layout.
- **Update Goal:**
 - Edit goal details similarly to budgets.

Transaction Management –

- **Create Transaction:**
 - Add income or expense entries.
 - Dynamically link to **budgets** or **goals** depending on the type.
- **Create Transaction from Image:**
 - Upload receipt/screenshots for parsing or tracking (manual input with optional file).
- **Update Transaction:**
 - Edit transaction details (linked budget/goal retained).
- **View Transactions:**
 - Display all transactions with sorting/filter options.
- **View Transactions by Budget/Goal:**
 - Filter transactions based on selected budget or goal.

Notification System –

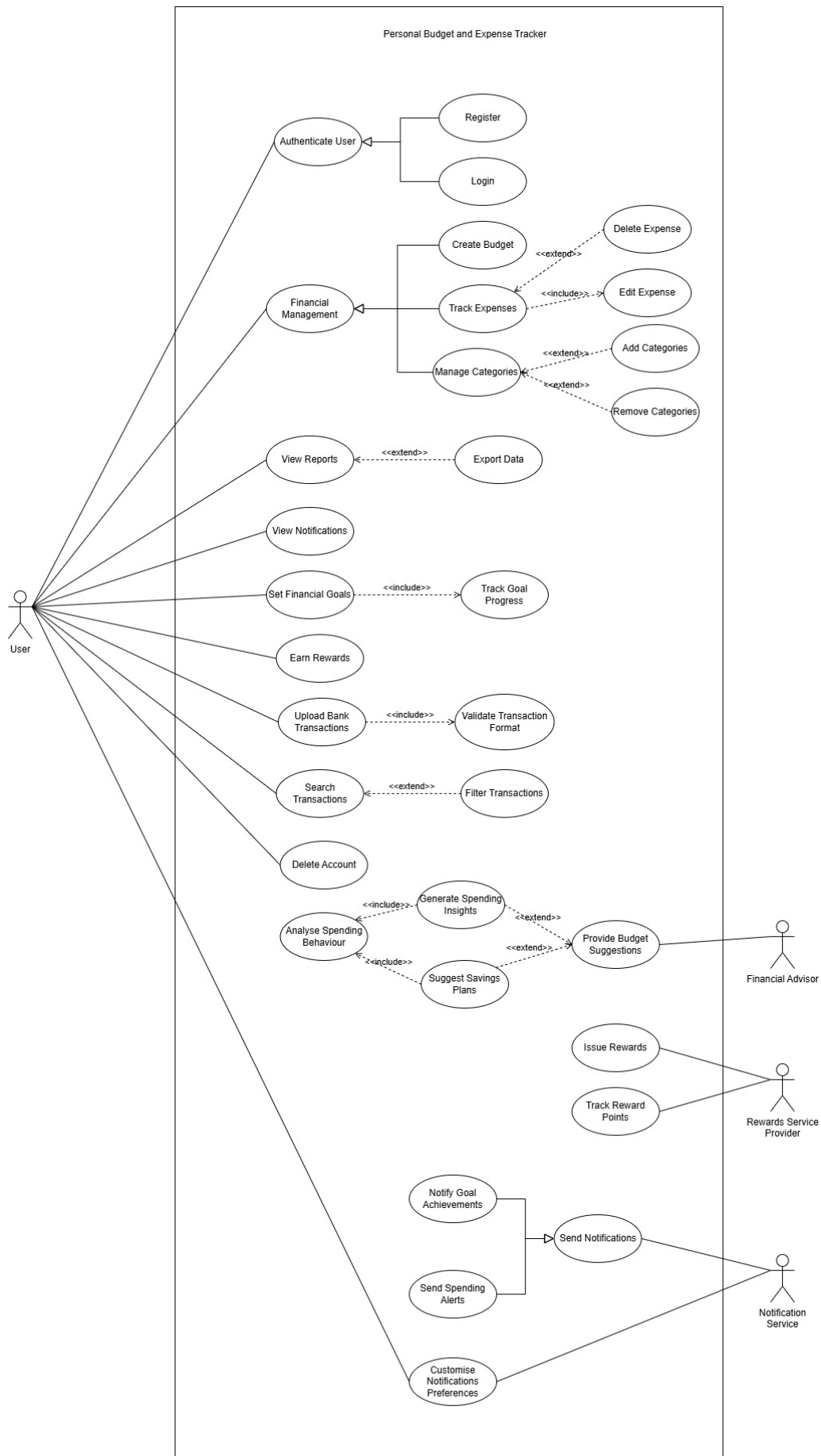
- **Real-Time Notifications** via WebSockets:
 - Alerts on events like transaction added, budget or goal set.
- **Notification Viewer Page:**
 - View all triggered notifications.

Reward System –

- View rewards for achieving financial goals (e.g., saved enough, didn't overspend).
- Encourages positive financial behavior.

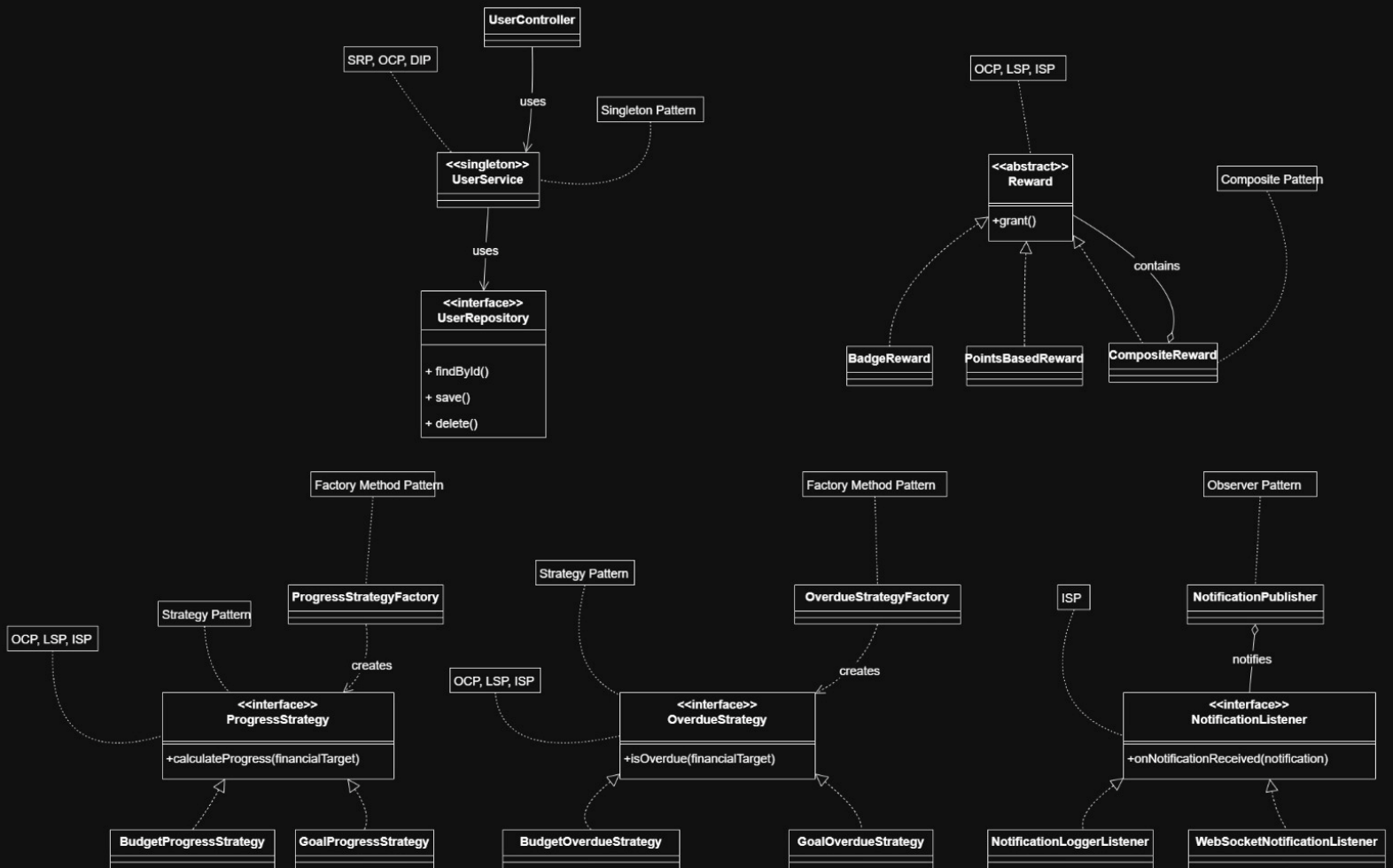
Models:

Use Case Diagram:

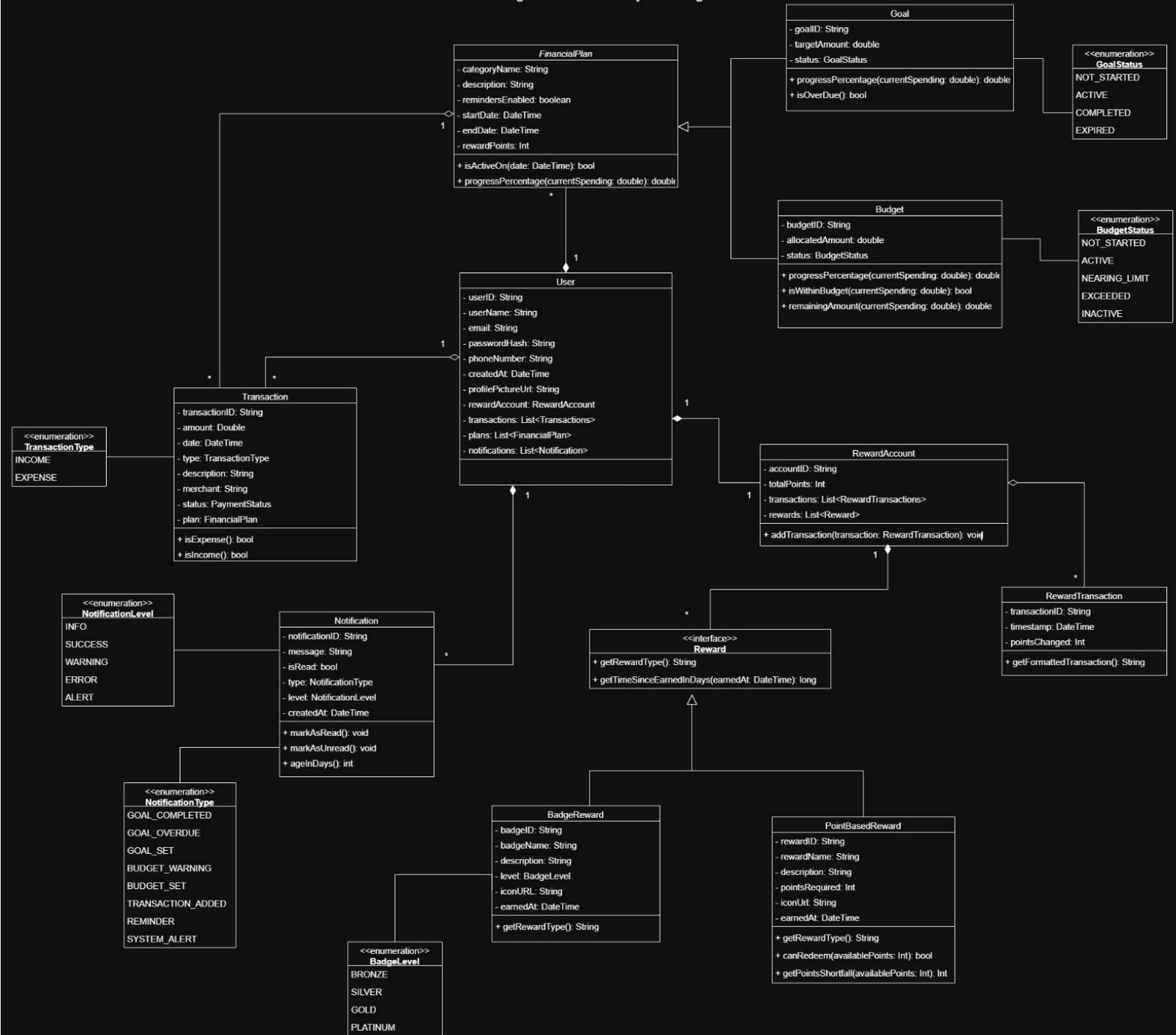


Class Diagram:

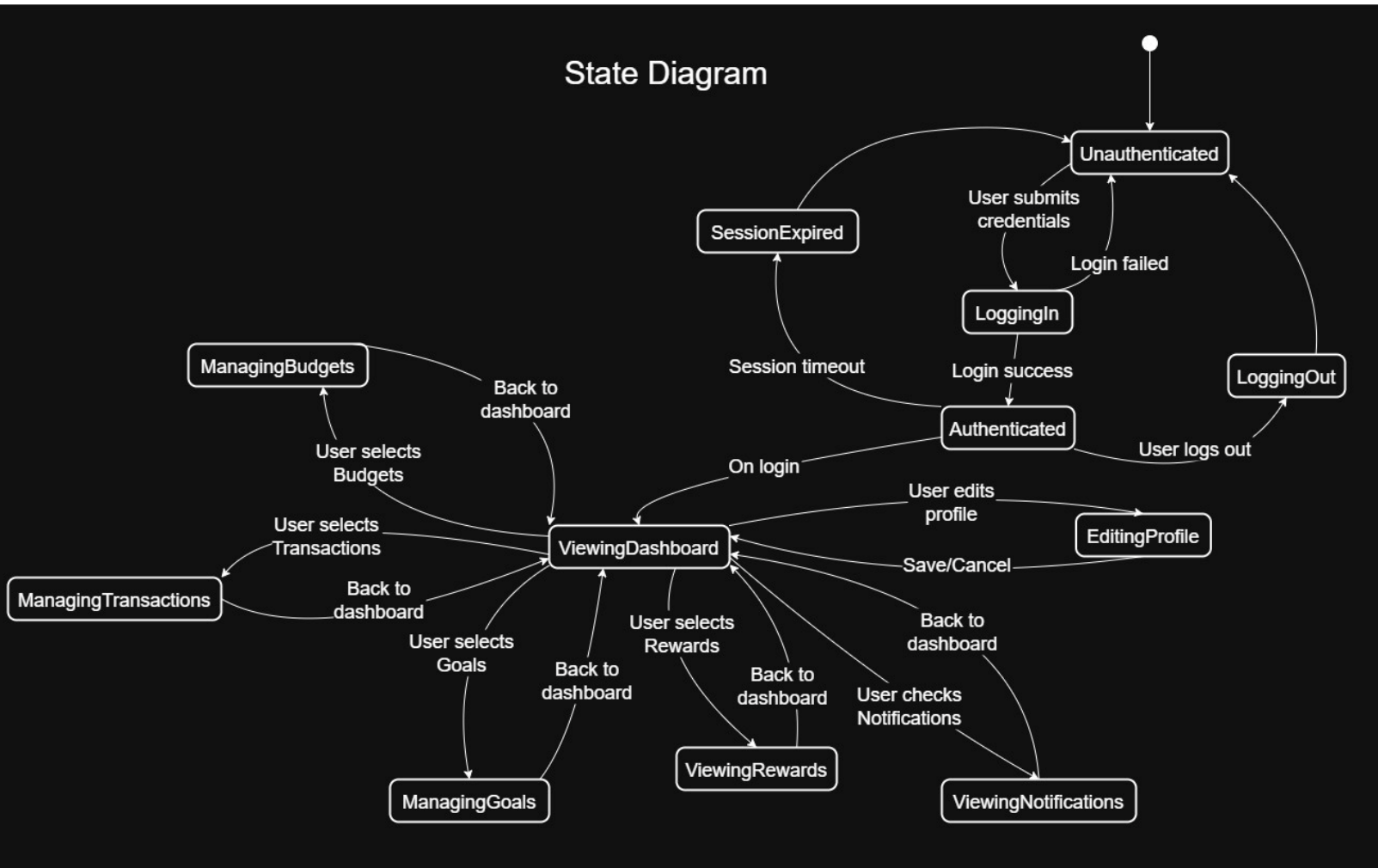
Design Patterns and Design Principles Implemented



Class Diagram for Model Layer - BudgetEase



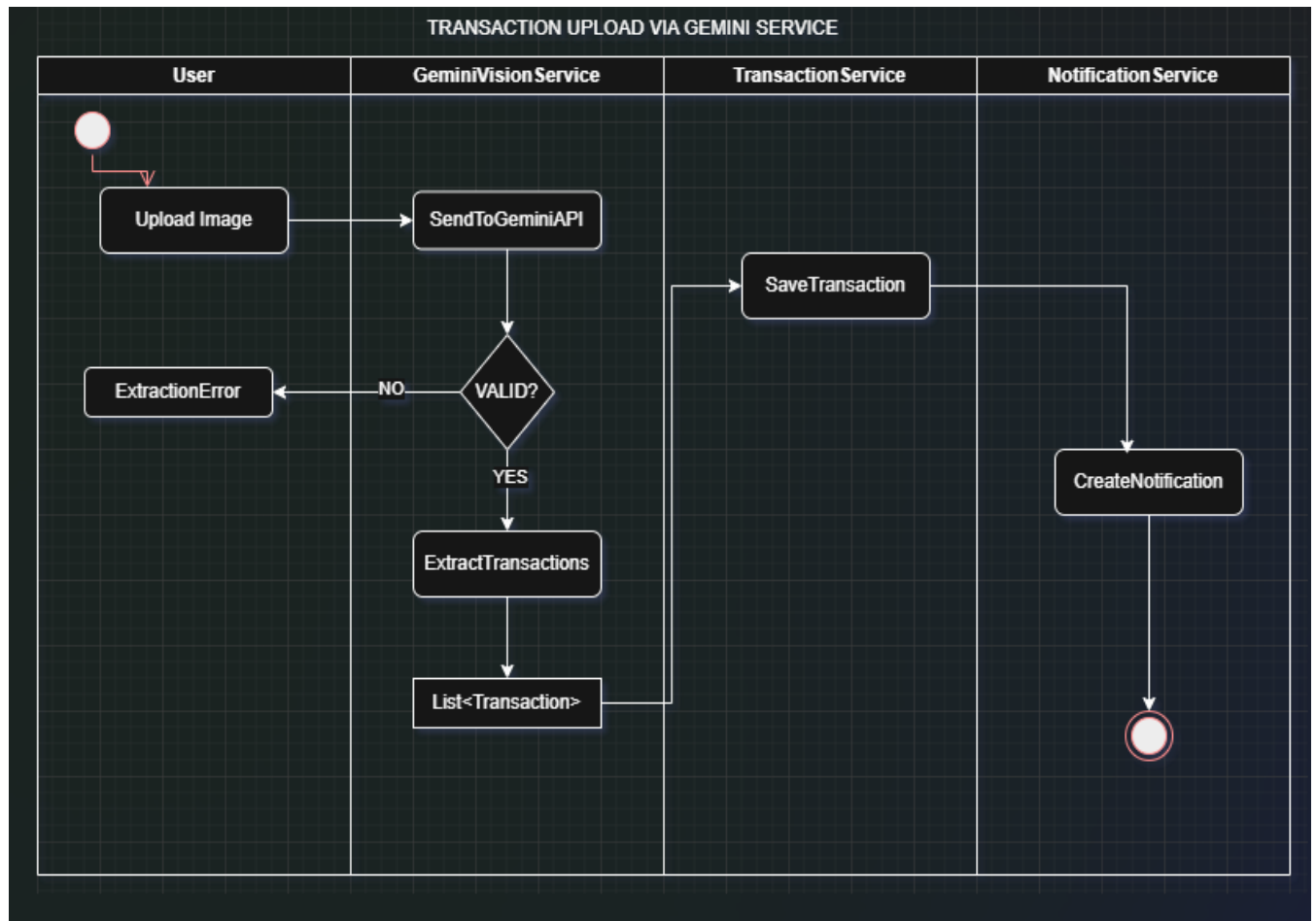
State Diagram:



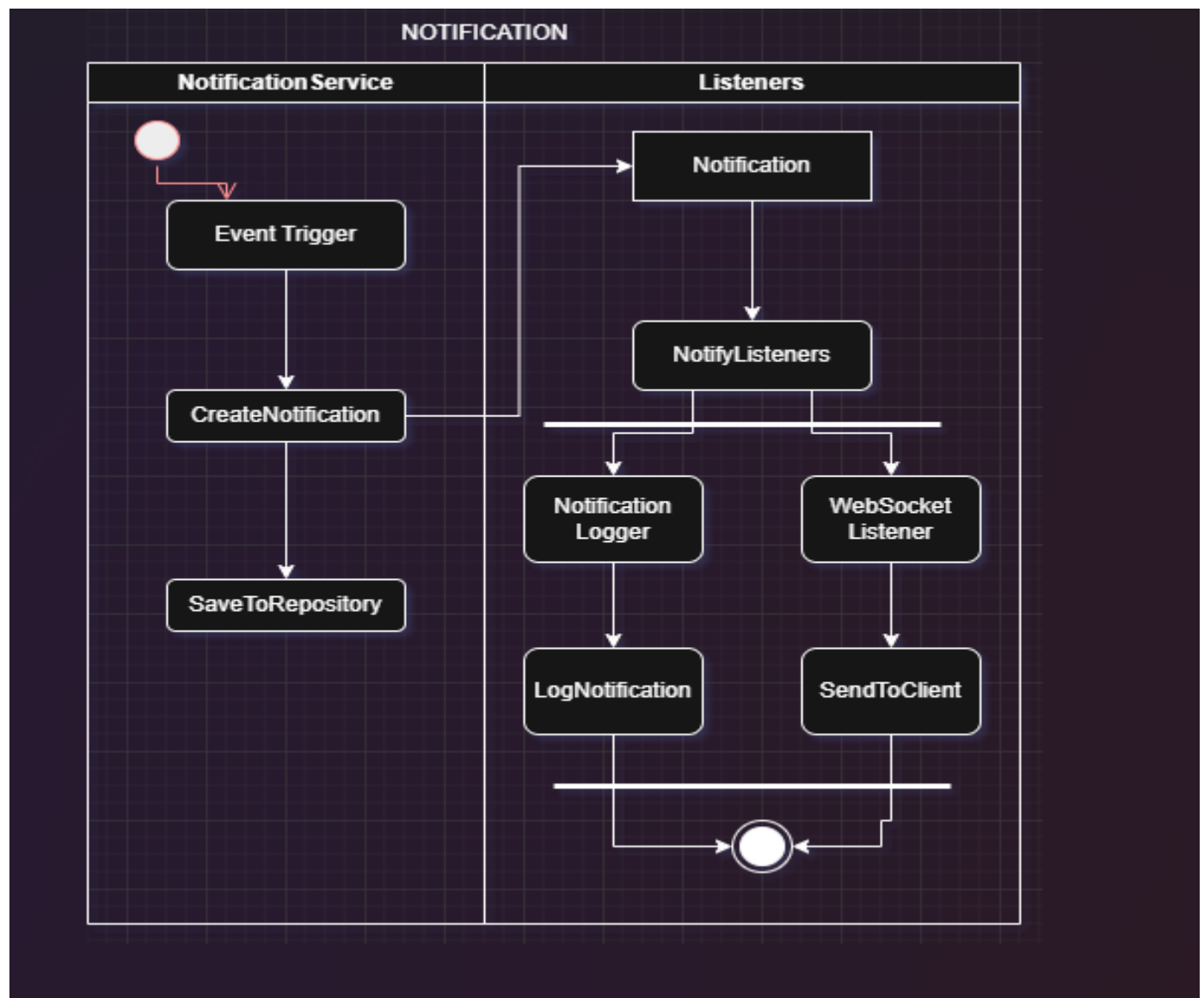
Activity Diagrams:

Major Use case –

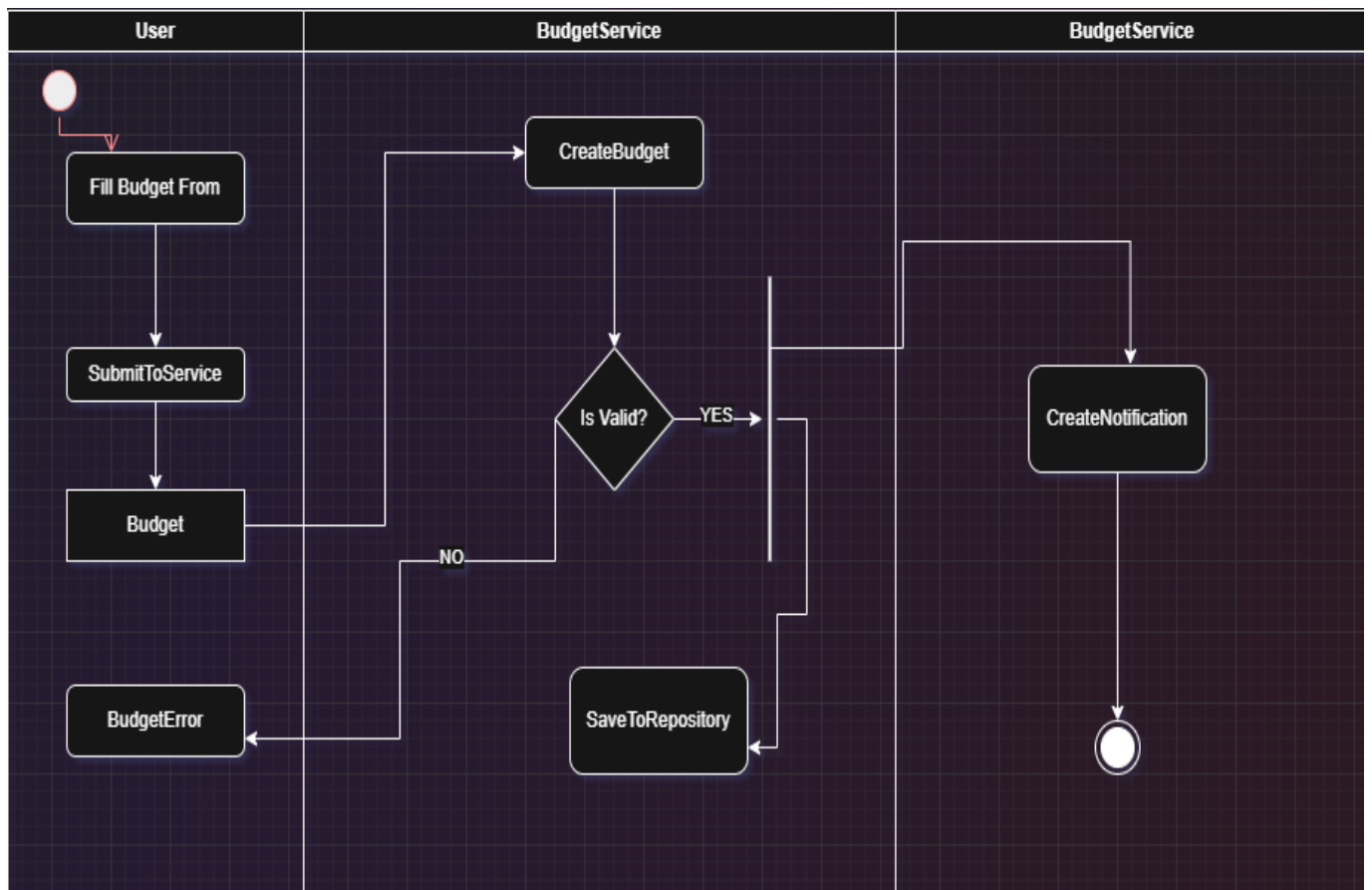
Create transaction via Gemini vision service –



Notification Listener –

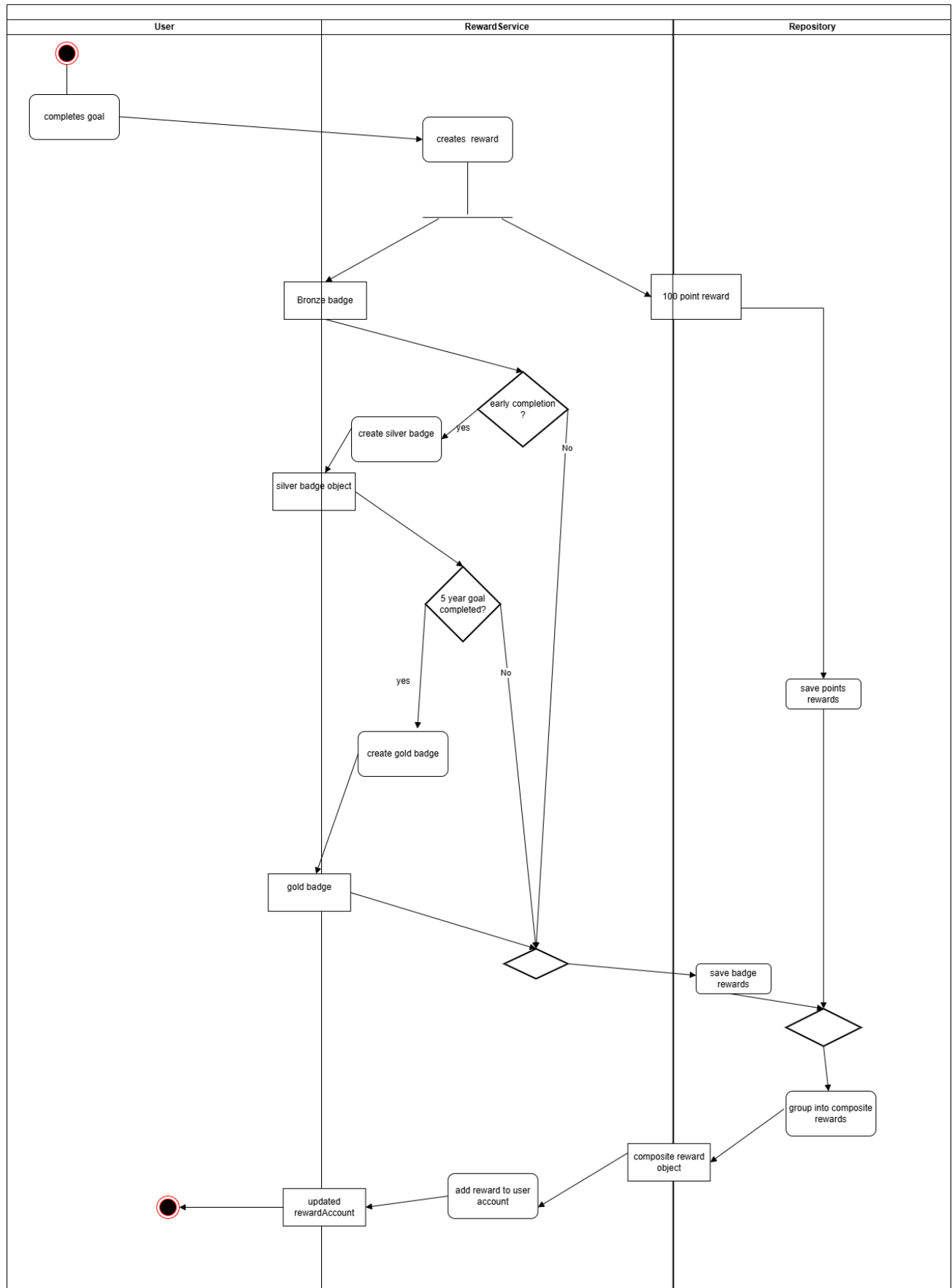


Create Budget –



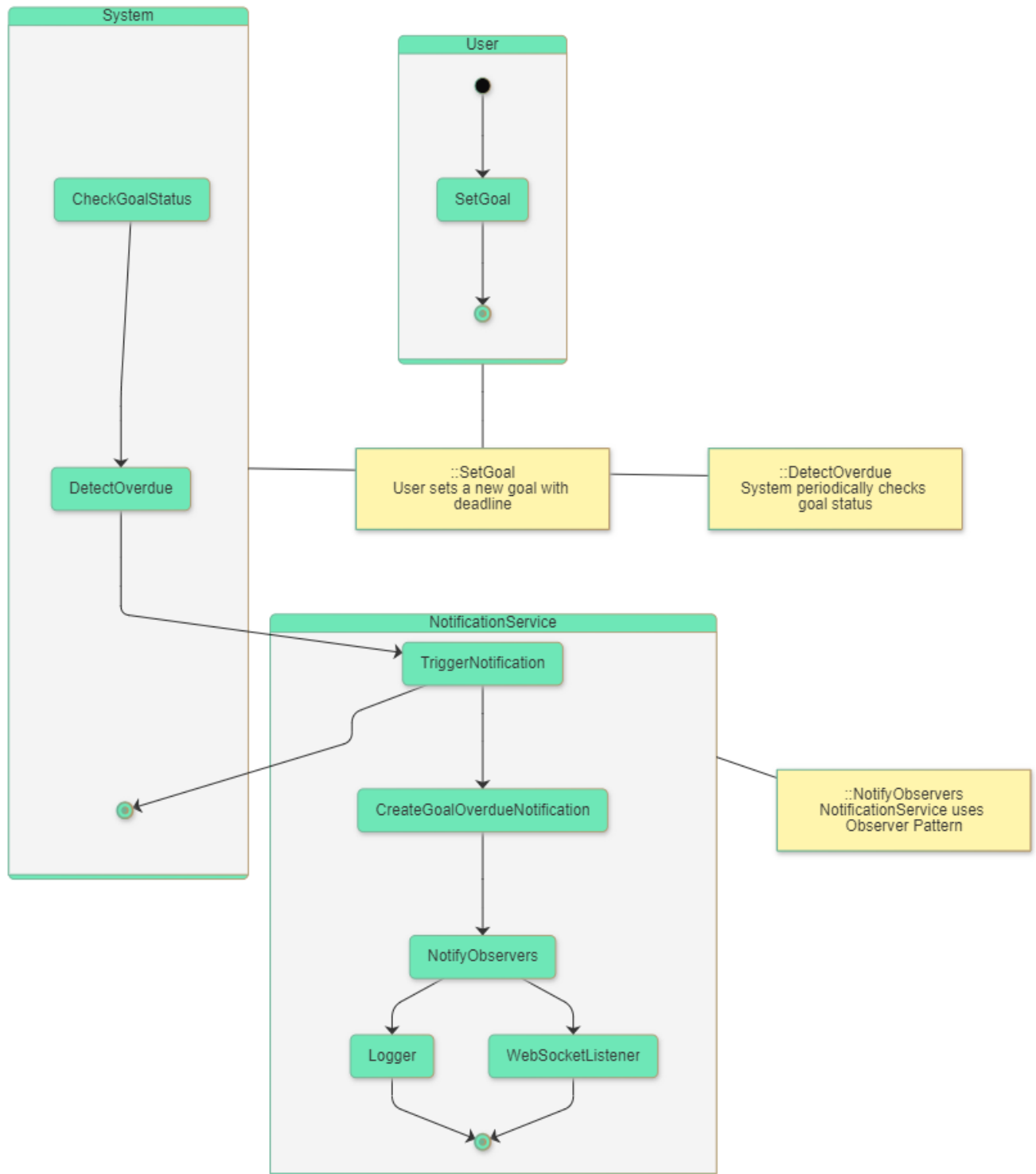
Composite rewards –

RewardService

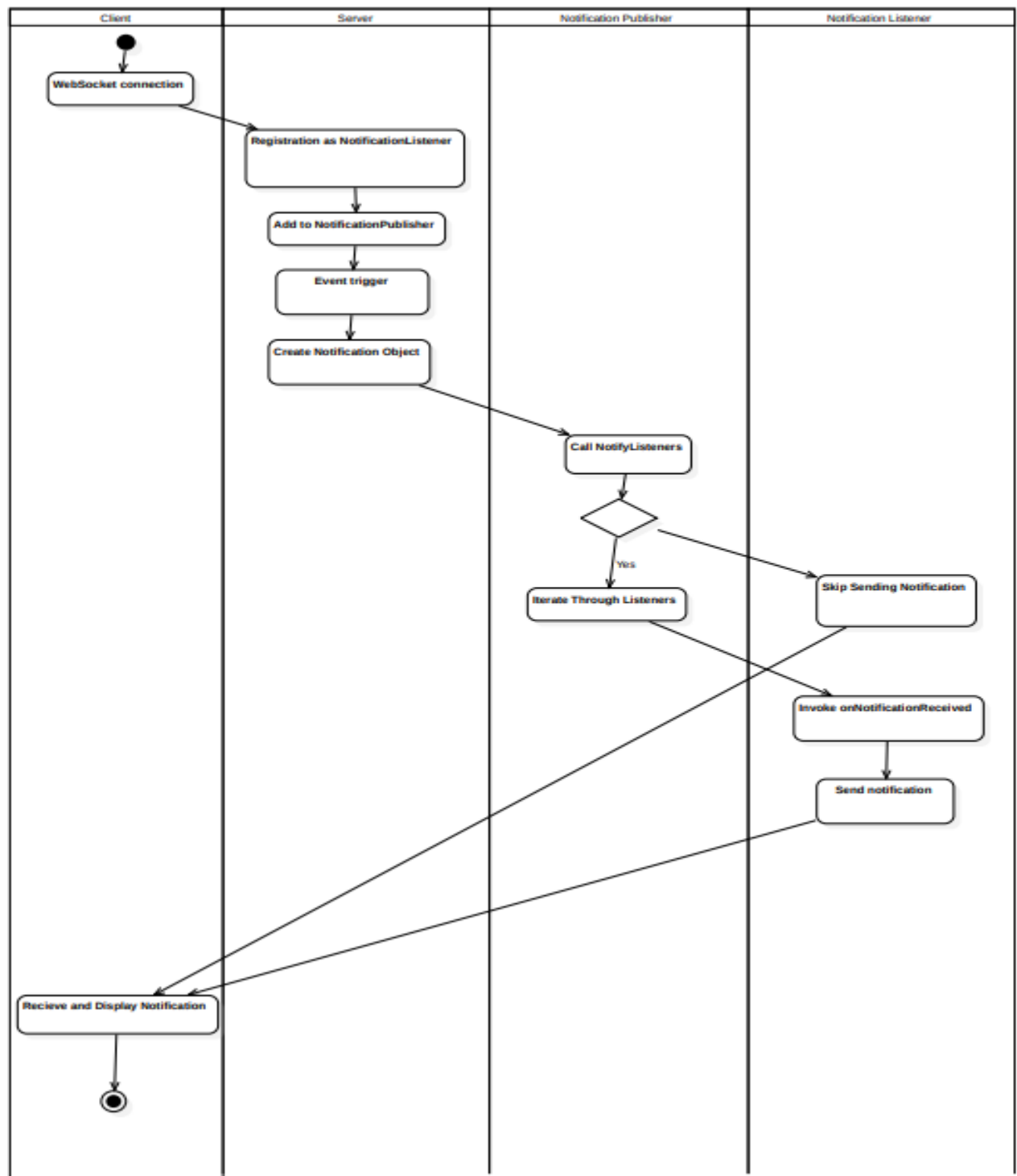


Minor Use case –

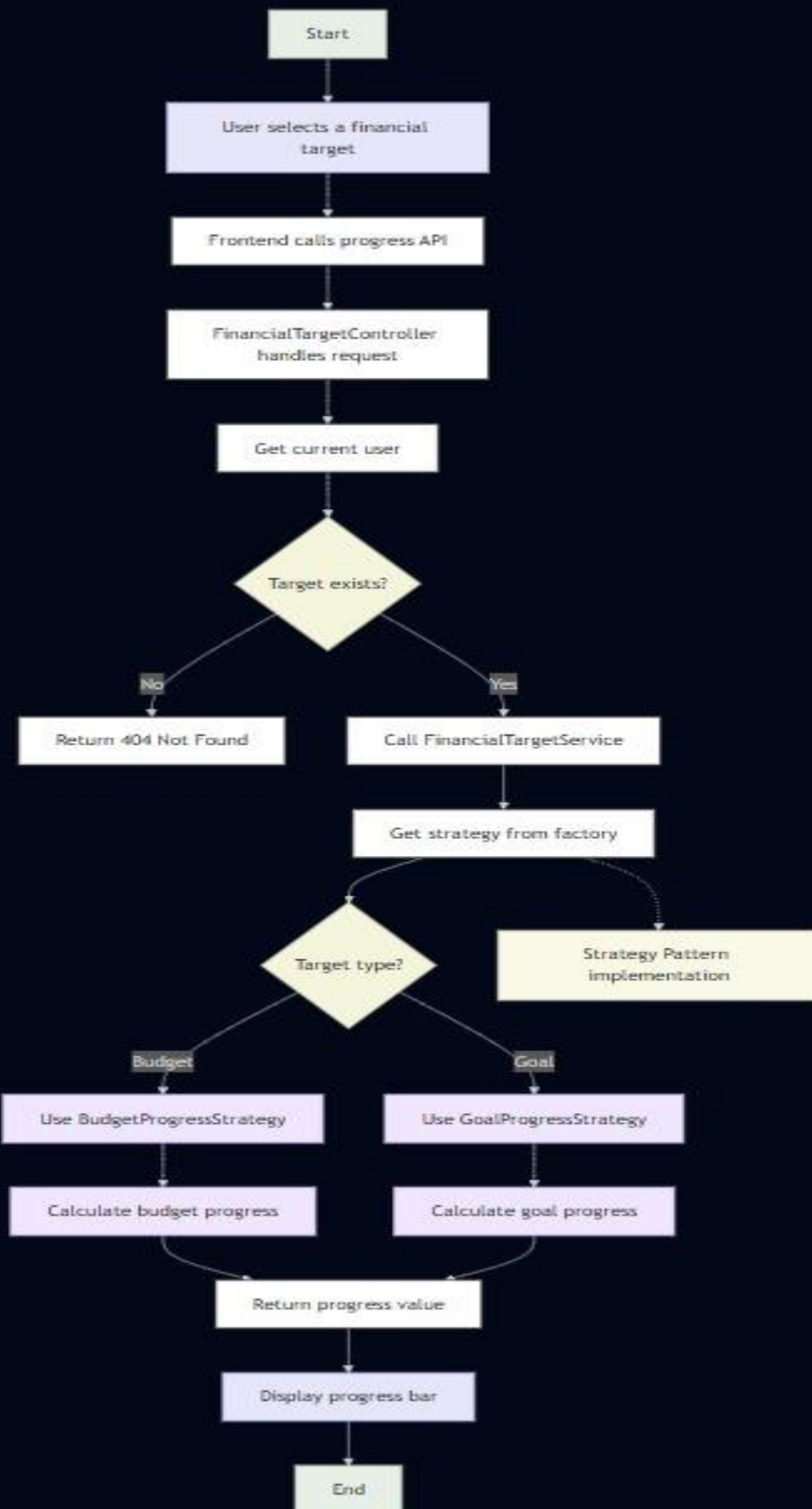
1. Goal Overdue activity:



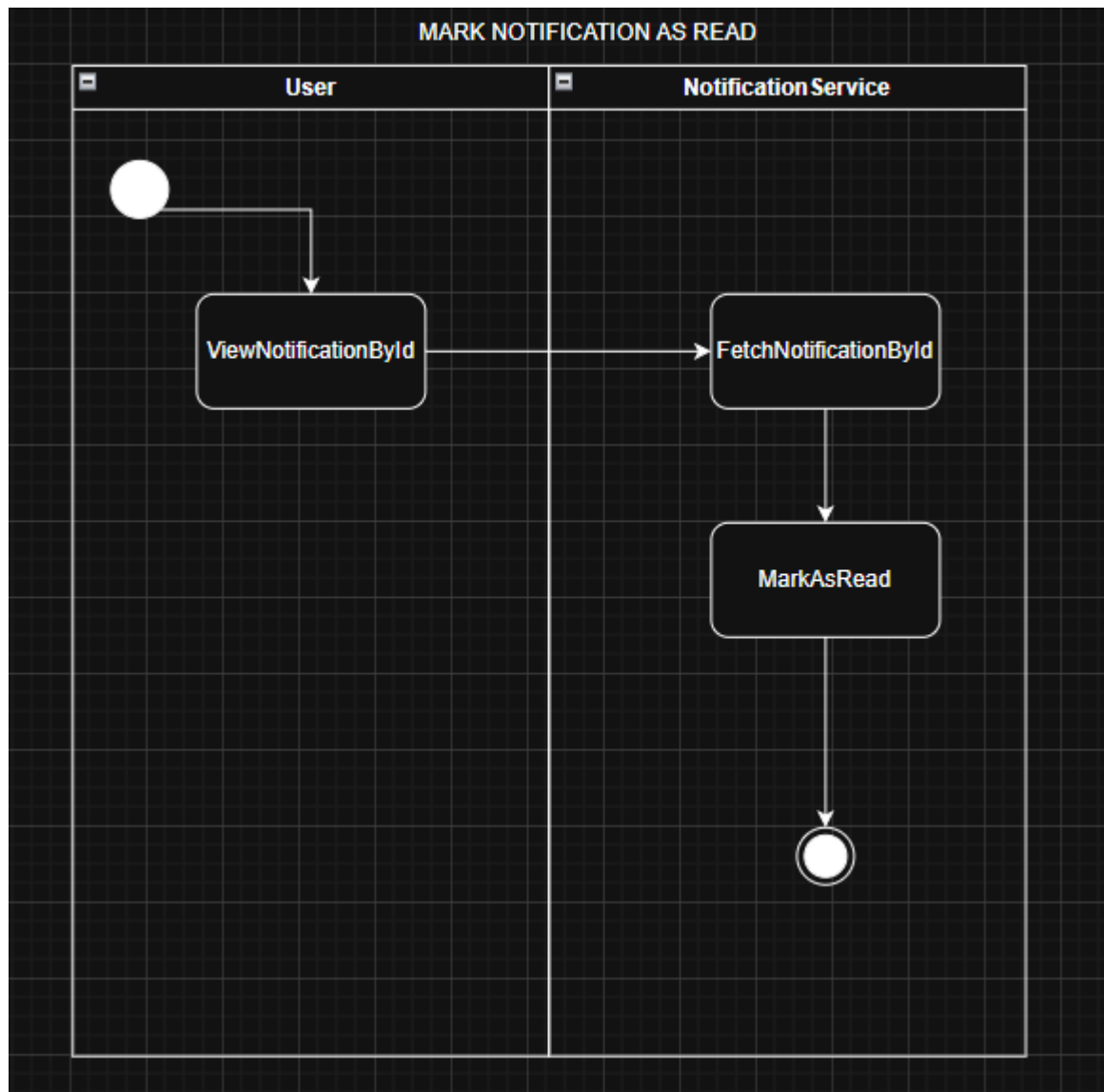
2. WebSocket notification



3. Display progress bar



4. Mark Notification as read



Architecture Patterns, Design Principles, and Design Patterns:

Architecture Patterns

For our project, we have made use of the Model-View-Controller (MVC) pattern.

Design Principles

We have implemented SOLID and GRASP principles.

Summary –

Principle	Example Classes
SRP	<code>UserService</code> , <code>BudgetService</code> , <code>GoalService</code> , <code>TransactionService</code> , <code>RewardService</code> , <code>NotificationService</code> , <code>FileUpload</code> , <code>JwtUtil</code> , <code>PasswordUtil</code> , <code>ValidateEmail</code> , <code>NotificationPublisher</code> , <code>NotificationLoggerListener</code> , <code>WebSocketNotificationListener</code>
OCP	<code>ProgressStrategy</code> , <code>BudgetProgressStrategy</code> , <code>GoalProgressStrategy</code> , <code>OverdueStrategy</code> , <code>BudgetOverdueStrategy</code> , <code>GoalOverdueStrategy</code> , <code>Reward</code> , <code>BadgeReward</code> , <code>PointBasedReward</code> , <code>CompositeReward</code> , <code>NotificationListener</code> , <code>NotificationLoggerListener</code> , <code>WebSocketNotificationListener</code>
LSP	<code>ProgressStrategy</code> and its implementations, <code>OverdueStrategy</code> and its implementations, <code>Reward</code> and its subclasses, <code>NotificationListener</code> and its implementations
ISP	<code>ProgressStrategy</code> , <code>OverdueStrategy</code> , <code>NotificationListener</code>
DIP	All services depend on interfaces (repositories, strategies, listeners)
Controller (GRASP)	All <code>*Controller</code> classes
Creator (GRASP)	<code>UserService</code> , <code>RewardService</code> , <code>GoalService</code> , <code>BudgetService</code> , <code>NotificationService</code>
Information Expert (GRASP)	All service classes, <code>FinancialTargetService</code>
Low Coupling (GRASP)	Utility classes, use of interfaces
High Cohesion (GRASP)	Service classes, strategy classes, utility classes
Polymorphism (GRASP)	<code>ProgressStrategy</code> , <code>OverdueStrategy</code> , <code>Reward</code> , <code>NotificationListener</code>
Pure Fabrication (GRASP)	<code>JwtUtil</code> , <code>PasswordUtil</code> , <code>ValidateEmail</code> , <code>FileUpload</code> , <code>ApiResponse</code>
Indirection (GRASP)	<code>ProgressStrategyFactory</code> , <code>OverdueStrategyFactory</code> , <code>NotificationPublisher</code>

Design Patterns

1. Strategy Pattern –

The Budget and Goal classes have similar attributes and behaviors, namely, behaviors like `isOverdue()` and `calculateProgress()` are both similar behaviors, but their particular implementations are different. In order to manage this, we have made use of Strategy Pattern.

This helps us to group the common behavior while giving them specific implementation based on our requirement.

2. Factory Pattern –

We use Factory pattern to delegate the instantiation of an object to a factory class, which handles the instantiation of the object based on the instance which it receives, i.e. either Budget or Goal.

In our project, strategy and factory patterns both work hand in hand, when a ‘/calculate-progress’ or ‘/overdue’ endpoint is hit, factory pattern is used to instantiate object of correct class based on whether received input is of Budget or Goal type, we then use strategy pattern implementations to actually give the output.

3. Composite Pattern –

Composite pattern is used to deal with individual objects and a composition of objects in the same way, simplifying handling of single rewards and groups of rewards as well, in our project, certain events result in granting of multiple rewards, this is where composite pattern is used. In particular, we have made use of it in Reward model.

We have a Reward abstract class that defines common attributes and behaviors. PointBasedReward and BadgeReward both inherit from Reward class. We have a CompositeReward class that has a list of Reward type objects and methods to apply rewards to individual rewards (i.e. the leaf classes) based on which instance it is.

4. Observer Pattern –

We have made use of Observer pattern for Notifications.

This pattern is used to decouple a service that creates and sends notifications/messages from the actual entities that listen to these notifications/messages.

In our project, we have a NotificationListener interface that has a method called onNotificationReceived().

We have a class called NotificationLoggerListener and WebSocketListener which both implement this interface and gives particular implementation for the onNotificationReceived() method, NotificationLoggerListener just System.out.prints the notification and WebSocketListener sends the message via websocket to our frontend.

There is a publisher class that maintains a list of listeners, making is easy to add and remove listeners. It contains a method called notifyListeners that iterates this list of listeners and calls the onNotificationReceived() method for them.

Finally, in NotificationService we add these listeners to publisher, and upon creating it, calls the notifyListeners method.

This pattern allows one to add as many listeners as they want with as different implementation as can be, but the NotificationService is not bothered with this. It also allows easy removal of listeners if need be.

Github link to the Codebase:

<https://github.com/Nish-077/BudgetEase>

Screenshots


UI:

User authentication –


Welcome Back

Sign in to access your account

Email or Username

 test

Password



☐ Remember me

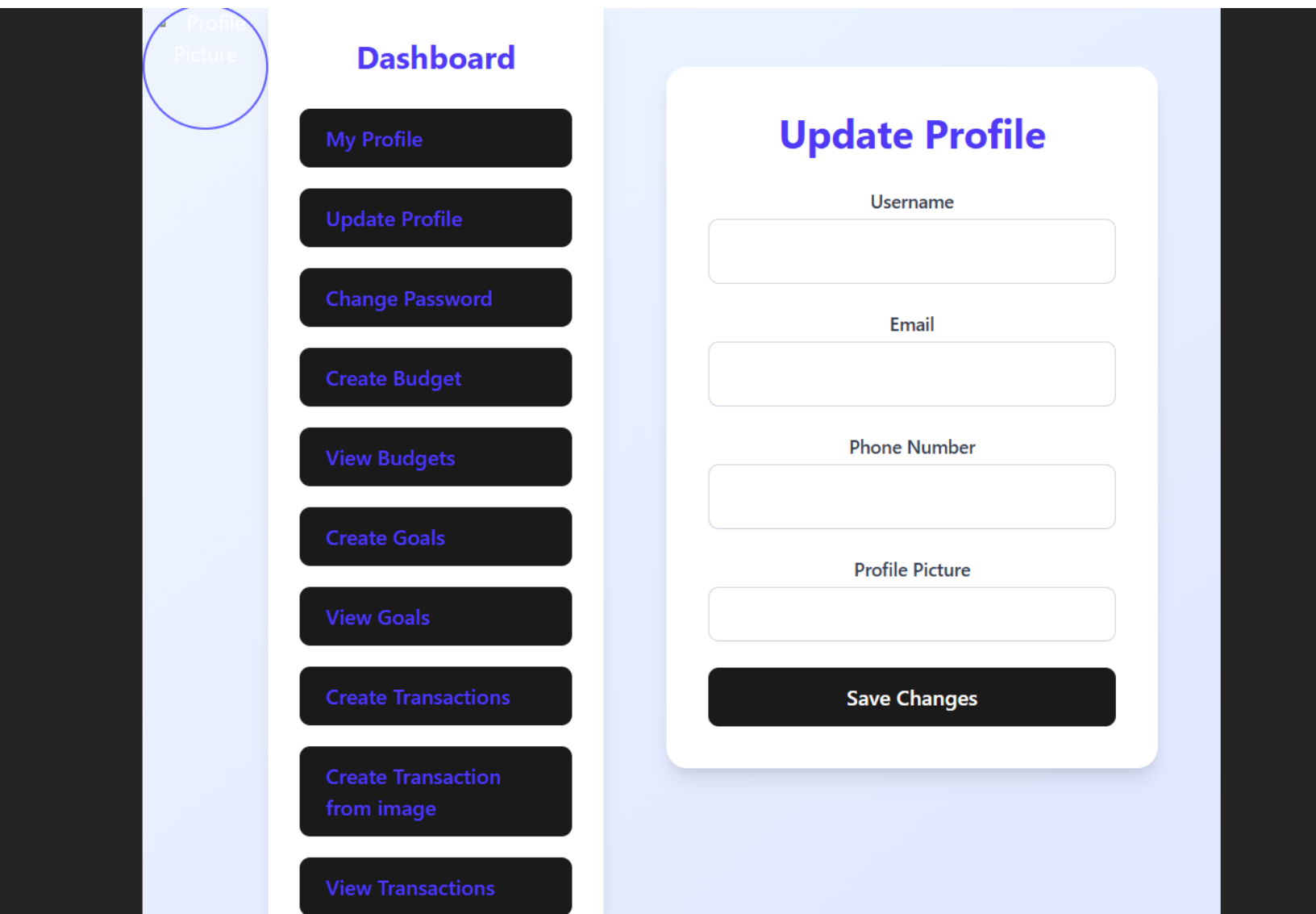
[Forgot password?](#)

Sign in

Don't have an account?

[Sign up](#)

Update profile (user can upload profile picture as well) –



Profile Picture

Dashboard

My Profile

Update Profile

Change Password

Create Budget

View Budgets

Create Goals

View Goals

Create Transactions

Create Transaction from image

View Transactions

Create a Budget

Plan ahead and track your expenses

Budget Name

January Savings

Amount

0.00

Start Date

dd-mm-yyyy --:--

End Date

dd-mm-yyyy --:--

Category Name

Groceries

Create Budget

Want to manage your budgets? [View Budgets](#)

Viewing Budgets (here we have progress and overdue calls, notice the Budget Exceeded notification) –

Profile Picture

Dashboard

My Profile

Update Profile

Change Password

Create Budget

View Budgets

Create Goals

View Goals

Create Transactions

Create Transaction from image

Your Budgets

PES1UG22CS419

\$100

Gym

PES1UG22CS419

Gym

Budget Progress

100% of budget used

0% remaining

Allocated Amount

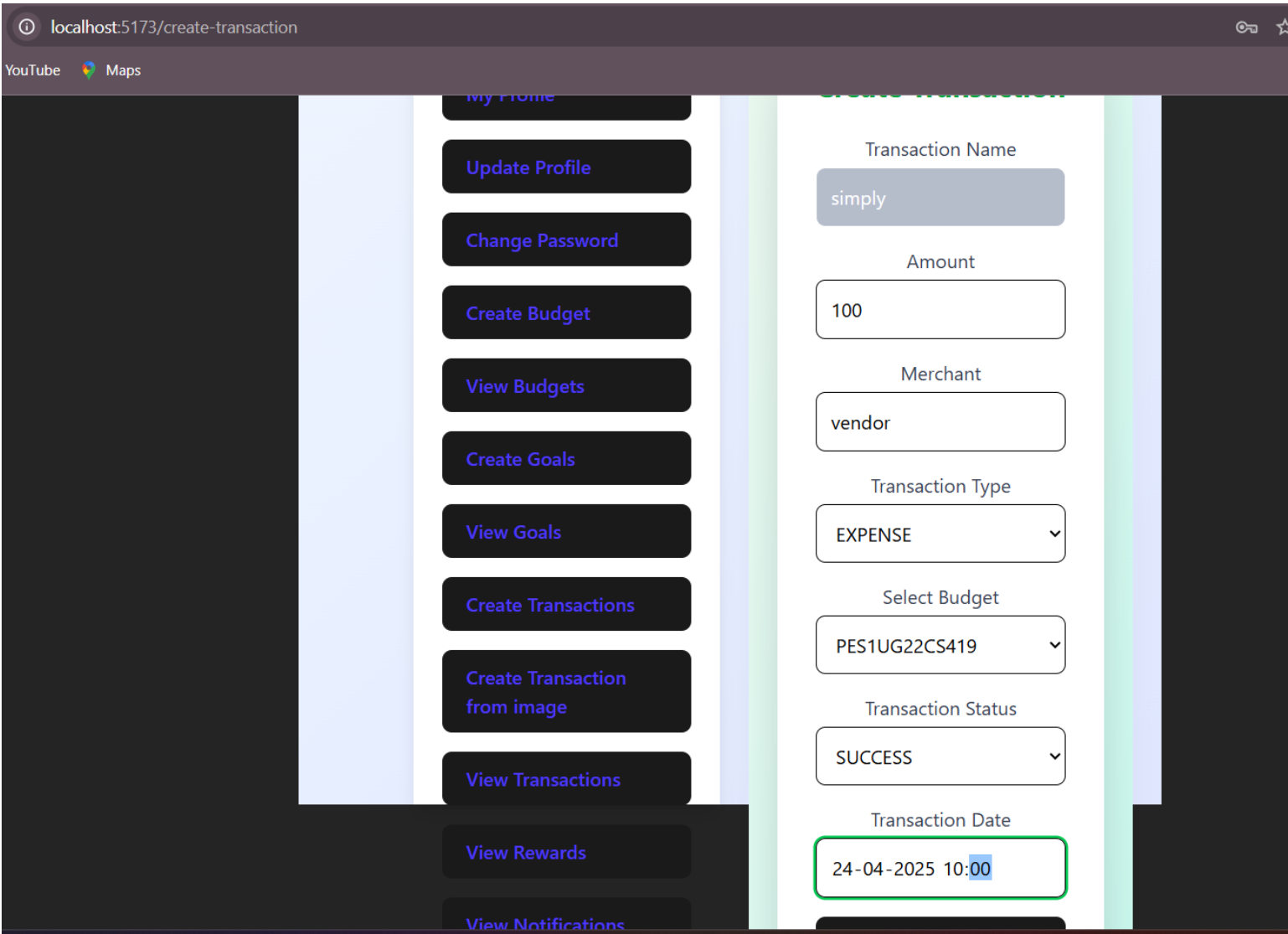
\$100

Edit Budget

View Transactions

BUDGET EXCEEDED

Create Transaction –



Create Transactions from image –

Picture

Dashboard

My Profile

Update Profile

Change Password

Create Budget

View Budgets

Create Goals

View Goals

Create Transactions

Create Transaction
from image

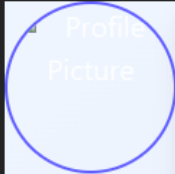
View Transactions

Extract Transactions from Image

Upload Receipt / Image

Choose File WhatsApp Imag... 12.41.53 AM.jpeg

Extract Transactions



Dashboard

My Profile

Update Profile

Change Password

Create Budget

View Budgets

Create Goals

View Goals

Create Transactions

Create Transaction
from image

View Transactions

Extract Transactions from Image

Upload Receipt / Image

Choose File WhatsApp Imag... 12.41.53 AM.jpeg

Extract Transactions

Extracted Transactions:

Name:

Amount: \$76

Type: EXPENSE

Date: 4/18/2025, 12:00:00 AM

Description: Transfers

Name:

Amount: \$0

Type: EXPENSE

Date: 4/17/2025, 12:00:00 AM

Description: Food

Viewing Transactions of a Budget –



Dashboard

My Profile

Update Profile

Change Password

Create Budget

View Budgets

Create Goals

View Goals

Create Transactions

Create Transaction
from image

View Transactions

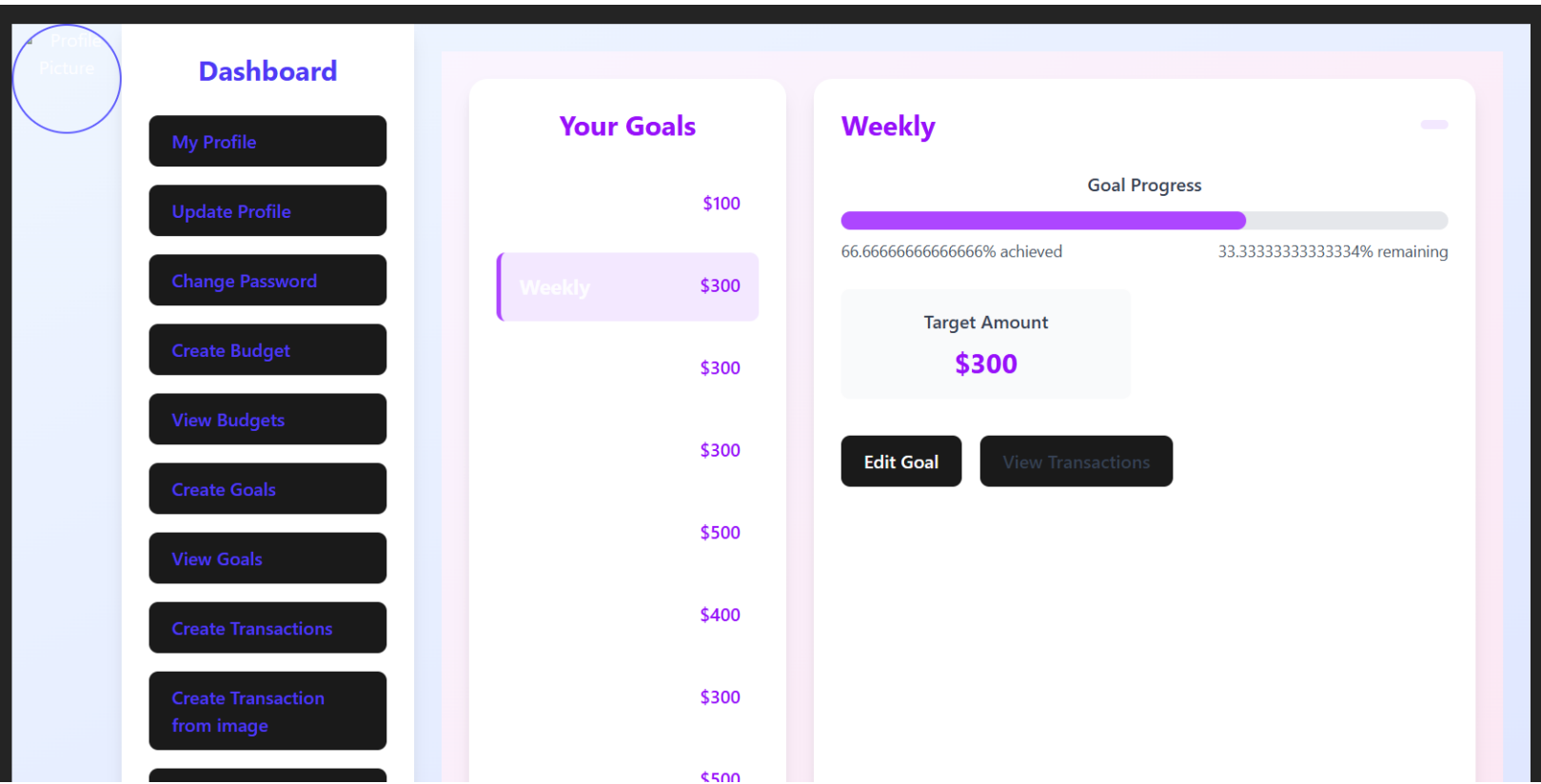
Transactions

Back to Budget

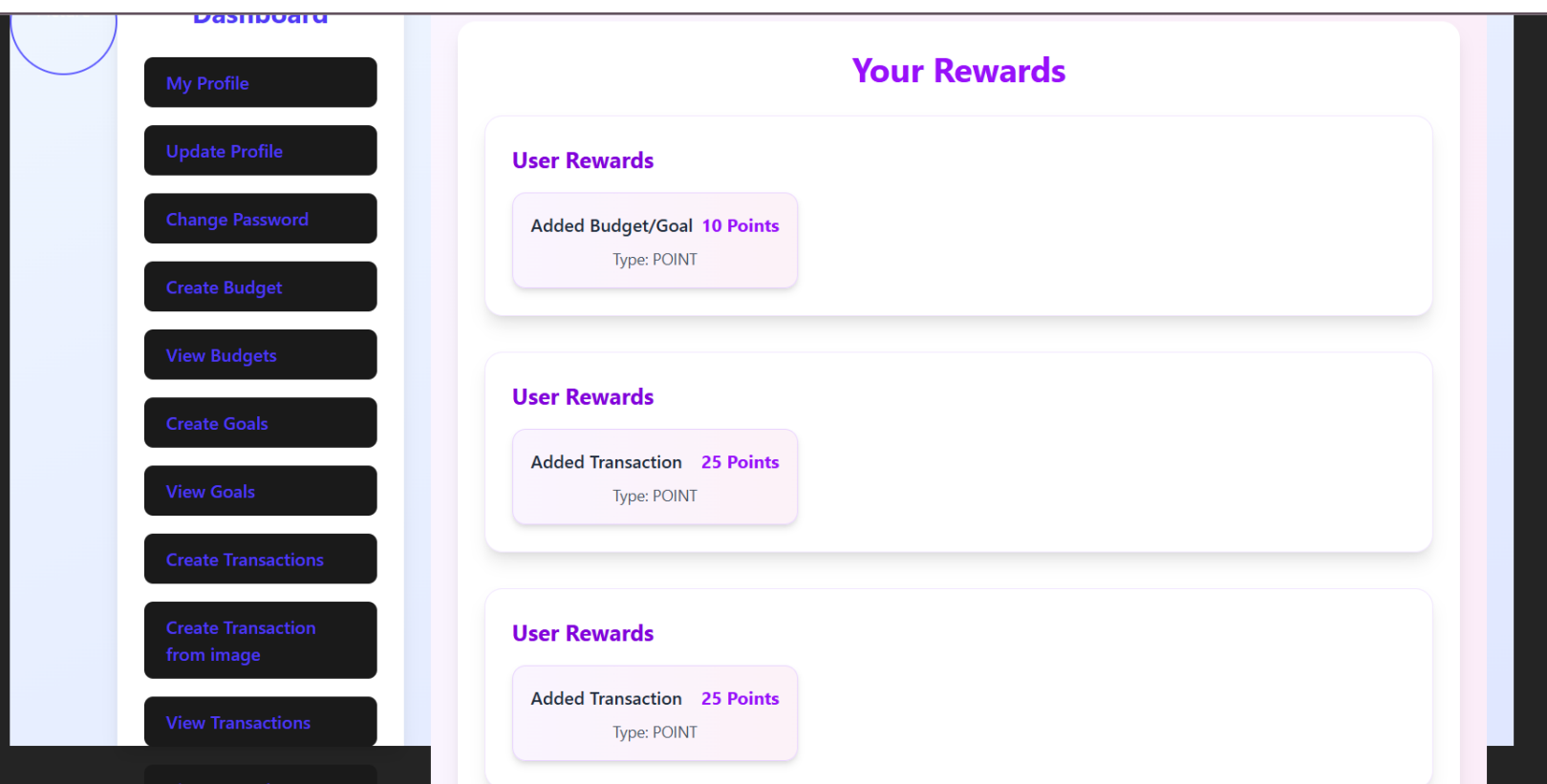
Date	Description	Amount	Attachment
4/24/2025	-	\$100	No file

Delete Budget

Viewing Goal (progress updates automatically when transactions are added to the goal/budget) –



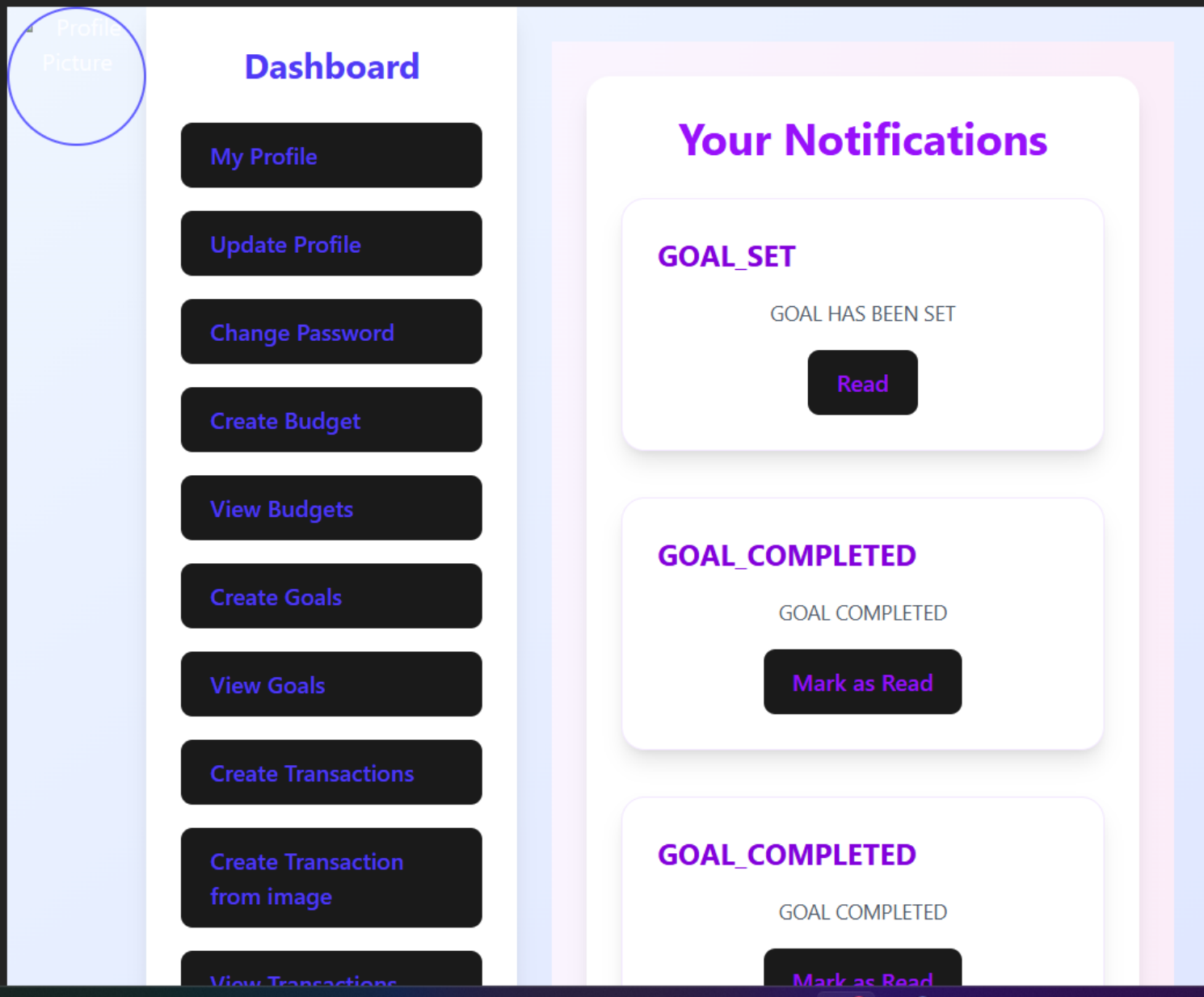
Viewing Rewards –



User Rewards

Completion Reward100 Points	CompletionPoints	Early CompletionPoints
Type: POINT	Type: BADGE	Type: BADGE
	Badge: BRONZE	Badge: SILVER

User Rewards



Individual contributions of the team members:

Name	Module worked on
Prabhat Deshmukh	Transaction
Nishant Banakar	Financial Target
Pranav Jigalur	Notification

Pranav Prashant Dambal	Rewards
-------------------------------	---------