Smart Budget Tracker - Pattern–Use Case Mapping

Emre Karadogan
Student ID: 211101068
Course: BIL482 - Design Patterns
Date: June 15, 2025

This document contains a list of cases that will be used in the **Smart Budget Tracker** project and the software design patterns that could be applied.

Core Case 1: Add / Edit / Delete Expense

Command Pattern: Encapsulates each user action (add/edit/delete) as an object, enabling features like undo/redo and improving action traceability.
Factory Pattern: Used to create standardized expense objects based on type (e.g., fixed vs variable expenses), improving modularity and scalability.

Core Use Case 2: Set Budget & Receive Notifications

Strategy Pattern: Enables flexible budget evaluation logic by allowing interchangeable strategies (e.g., weekly, monthly, custom thresholds).
Observer Pattern: Automatically triggers notifications when spending nears or exceeds defined limits, by observing relevant budget state changes.

Core Use Case 3: View Analytics Dashboard

Builder Pattern: Builds complex analytics reports from various data points in a stepwise manner, supporting extensibility and modular report generation.
Facade Pattern: Simplifies complex backend interactions (data fetch, calculations, formatting) behind a unified dashboard interface.

Core Use Case 4: Manage Spending Categories

Factory Pattern: Centralizes creation of user-defined or predefined spending categories, ensuring consistent structure and easing future additions.

Dashboard and Expense Analysis/Reporting

Builder Pattern: Can be used to enable the step-by-step construction of complex expense reports and analyses.
Facade Pattern: Can be used to simplify complex financial calculations and reporting operations. The dashboard allows the user to easily view financial summaries without needing to know the complex backend processes.

Application Configuration and Settings Management

Singleton Pattern: Can be used to provide a single, central access point for managing application-wide configuration settings and preferences. This pattern ensures that settings are accessed and modified consistently throughout the application.

Integration with Different Data Sources or External APIs

Adapter Pattern: When the application needs to retrieve data from different data sources or external services, this pattern can be used to adapt these external interfaces to the application's internal data format. This increases the system's flexibility and integration capability.

Adding Additional Features to Expense Entries

Decorator Pattern: Can be used to add additional features (e.g., tags, payment methods, etc.) to basic expense entries. This pattern allows for dynamically adding new functionalities without modifying the core expense object.

Managing Application States (Viewing, Editing, Analyzing)

State Pattern: Can be used to manage different user interface states of the application. This encapsulates each state's behavior, making the code more organized and sustainable.

Pattern–Use Case Mapping Matrix Image

	A	B	C
1	**Use Case**	**Design Pattern(s)**	**Explanation \& Justification**
2	Add / Edit / Delete Expense	Command Pattern	Encapsulates each action as an object. Enables undo/redo operations.
3	Add / Edit / Delete Expense	Factory Pattern	Creates standardized expense objects, allows adding new types easily.
4	Set Budget \& Receive Notifications	Strategy Pattern	Enables flexible evaluation strategies (e.g., weekly, monthly).
5	Set Budget \& Receive Notifications	Observer Pattern	Sends alerts when spending exceeds limits using observer-subscriber logic.
6	View Analytics Dashboard	Builder Pattern	Builds analytics step-by-step from multiple sources.
7	View Analytics Dashboard	Facade Pattern	Hides backend logic, offers simple dashboard API to frontend.
8	Manage Spending Categories	Factory Pattern	Consistently creates and manages predefined/custom categories.
9	Dashboard \& Expense Reporting	Builder Pattern	Builds complex financial reports dynamically.
10	Dashboard \& Expense Reporting	Facade Pattern	Aggregates multiple services into a simple interface.
11	App Config / Settings	Singleton Pattern	Central control of shared configuration across app.
12	External Data Integration	Adapter Pattern	Converts external API data to internal compatible format.
13	Extra Features for Expenses	Decorator Pattern	Adds metadata to expenses (tags, payment methods) without altering core class.
14	Managing UI States	State Pattern	Encapsulates view/edit states into self-contained logic.