Software Requirements Document

Smart Budget Tracker

Course: BIL482 - Design Patterns
Project Title: Smart Budget Tracker
Date: June 2025
Document Prepared By:
Emirhan Yavuz
Berşan Kayra Korkmaz
Institution: TOBB ETU
Academic Year: 2024-2025

Contents

1 Product Perspective **2**

2 Product Functions **2**

3 User Characteristics **2**

4 Constraints **2**

5 System Features (Use Case Based) **3**
5.1 Use Case 1: Add / Edit / Delete Expense 3
5.2 Use Case 2: Set Budget & Receive Notifications 3
5.3 Use Case 3: View Analytics Dashboard 4
5.4 Use Case 4: Manage Spending Categories 5

6 Non-Functional Requirements **6**

7 External Interface Requirements **6**

8 User Interfaces **6**

9 Software Interfaces **6**

10 Software Architecture Overview **7**

1. Product Perspective

The **Smart Budget Tracker** is a stand-alone web-based application developed as part of the BIL482 Design Patterns course. It aims to provide users with a comprehensive solution for managing personal finances. It is not dependent on any other system but is designed with modularity in mind for potential future integrations such as multi-account sync or cloud backup services.

2. Product Functions

The key functions of the system include:

- Adding, editing, and deleting expenses with categories, amounts, dates, and descriptions.
- Setting monthly or weekly budget limits per category.
- Visualizing spending habits via dashboards with charts.
- Generating smart notifications when approaching or exceeding budgets.
- Exporting financial data in formats such as CSV and PDF.
- Viewing analytical summaries and expense trends over time.

3. User Characteristics

The system is primarily designed for:

- **Students and young professionals**: Basic financial tracking needs and moderate technical skills.
- **Instructors and project evaluators**: Reviewing design pattern usage and functionality.

No advanced financial or technical knowledge is required to use the system. The UI is designed to be intuitive and user-friendly.

4. Constraints

- **Platform:** Web-based (accessible via browser)
- **Frontend:** React.js
- **Backend:** Spring Boot
- **Database:** SQLite (MySQL optional)
- **Graphics:** Chart.js or D3.js
- **Notification API:** Web Notifications
- **Deadline:** Final delivery by July 14, 2025 (MVP by June 30, 2025)

5. System Features (Use Case Based)

5.1 Use Case 1: Add / Edit / Delete Expense

- **Actor:** User
- **Goal:** Manage expenses by adding, editing, or deleting records.
- **Preconditions:**
  - User is authenticated
  - Expense categories are available
- **Main Flow:**
  1. Navigate to Add Expense
  2. Enter details
  3. Save entry
  4. View updated list
- **Postconditions:** Expense data is updated in the system.
- **Design Pattern:** Command Pattern, Factory Pattern

Design Pattern Explanation:
Command Pattern encapsulates each user action (add/edit/delete) as an object, enabling undo/redo features.
Factory Pattern is used to create standardized expense objects depending on type, ensuring consistency and modular creation logic. These are implemented in the backend service layer and DAL.

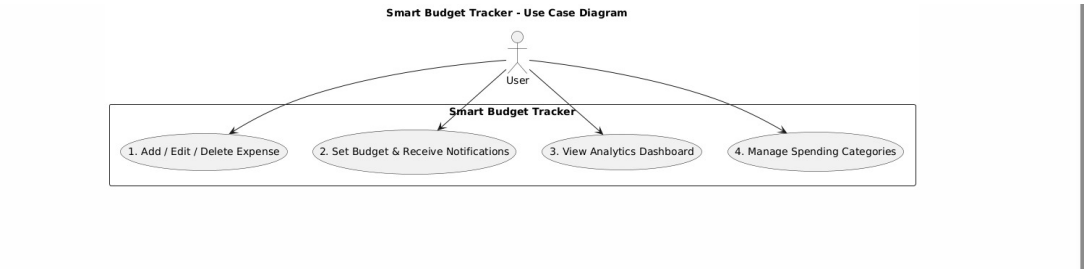

Figure 1: Use Case Diagram - Add / Edit / Delete Expense

5.2 Use Case 2: Set Budget & Receive Notifications

- **Actor:** User
- **Goal:** Set category budget limits and get notified
- **Preconditions:** Authenticated user, existing categories
- **Main Flow:**

1. Open Budget Settings
2. Set thresholds
3. Save and monitor

- **Postconditions:** Thresholds saved, alerts active
- **Design Pattern:** Strategy Pattern, Observer Pattern

Design Pattern Explanation:
Observer Pattern allows the notification component to subscribe to budget events and trigger alerts automatically.
Strategy Pattern supports dynamic switching between different evaluation strategies (daily, weekly, monthly). Both patterns increase modularity and flexibility in the backend.

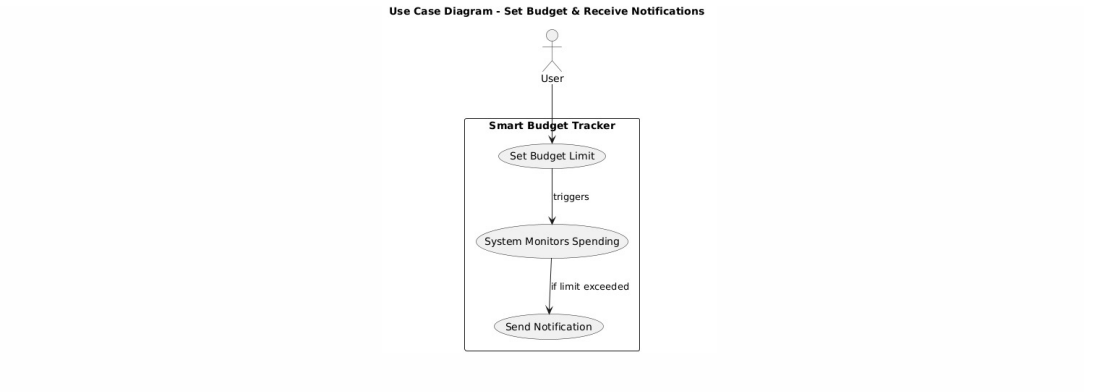

Figure 2: Use Case Diagram - Set Budget & Receive Notifications

5.3 Use Case 3: View Analytics Dashboard

- **Actor:** User
- **Goal:** Visualize budget summaries and charts
- **Preconditions:** Authenticated user, existing data
- **Main Flow:**

1. Open dashboard
2. Fetch and process data
3. Render visual charts

- **Postconditions:** User sees updated insights
- **Design Pattern:** Builder Pattern, Facade Pattern

Design Pattern Explanation:
Builder Pattern is used to incrementally generate analytics reports from multiple data sources.
Facade Pattern simplifies complex backend calls into a unified dashboard API, improving readability and reusability.

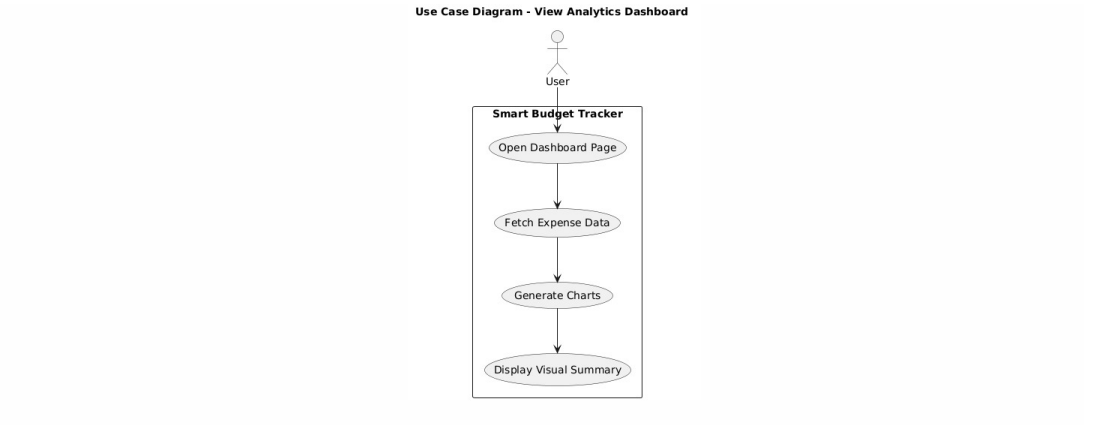

Figure 3: Use Case Diagram - View Analytics Dashboard

5.4 Use Case 4: Manage Spending Categories

- **Actor:** User
- **Goal:** Add/edit/delete expense categories
- **Preconditions:** Authenticated user
- **Main Flow:**
  1. Open category settings
  2. Make changes
  3. Save and update DB
- **Postconditions:** Updated category list visible
- **Design Pattern:** Factory Pattern

Design Pattern Explanation:
Factory Pattern encapsulates category creation logic, ensuring consistent data structure and enabling easy extension of category types. It enhances code maintainability by decoupling creation logic from controller components.

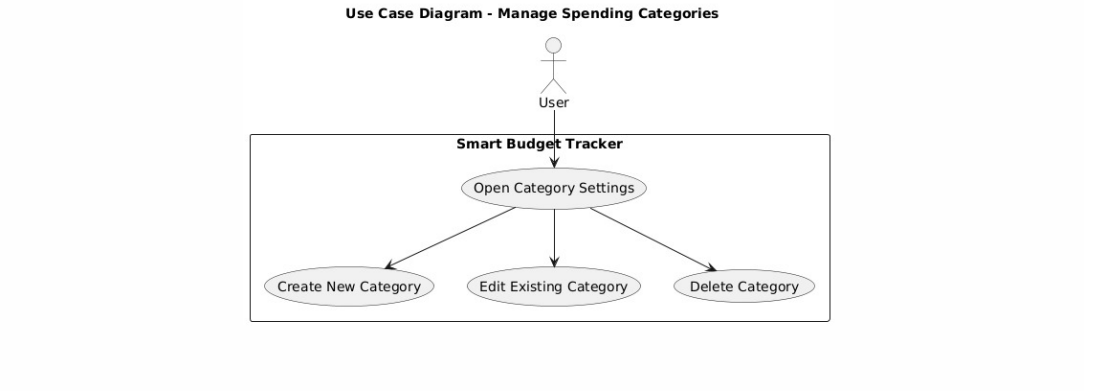

Figure 4: Use Case Diagram - Manage Spending Categories

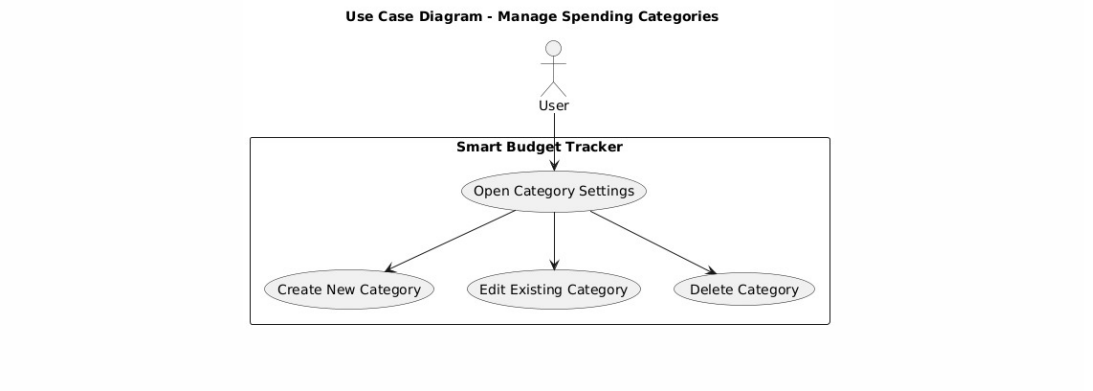

Figure 5: Use Case Diagram - Manage Spending Categories

6. Non-Functional Requirements

- **Usability:** The interface is designed to be intuitive and user-friendly, allowing non-technical users to navigate, enter expenses, and view dashboards with ease.
- **Performance:** The system is expected to handle basic operations (expense entry, budget checks) with under 500ms response time under normal conditions.
- **Portability:** As a web-based application, the system can be accessed from any platform with a modern browser including Windows, macOS, and Linux.
- **Reliability:** Core functionality such as data saving, alert triggering, and dashboard generation should function with 99% uptime during operation.

7. External Interface Requirements

There are no mandatory external interfaces. However, optional integrations such as email services or cloud storage may be considered in future versions.

8. User Interfaces

The system includes the following major screens:

- **Expense Form:** Input screen for new or updated expenses
- **Dashboard UI:** Displays summaries and charts of user data
- **Notification UI:** Displays alerts for budget limit events

9. Software Interfaces

- React.js is used for frontend development.
- Spring Boot handle API routing and backend logic.

- SQLite(or MySql) provides the local persistent data layer.
- Chart.js or D3.js is used for visual analytics components.

10. Software Architecture Overview

The Smart Budget Tracker follows a layered architecture based on a client-server model. The architecture includes a React-based frontend, a Spring Boot backend, and a SQLite(or Mysql) database. The system is organized into logical layers to separate concerns and improve maintainability.

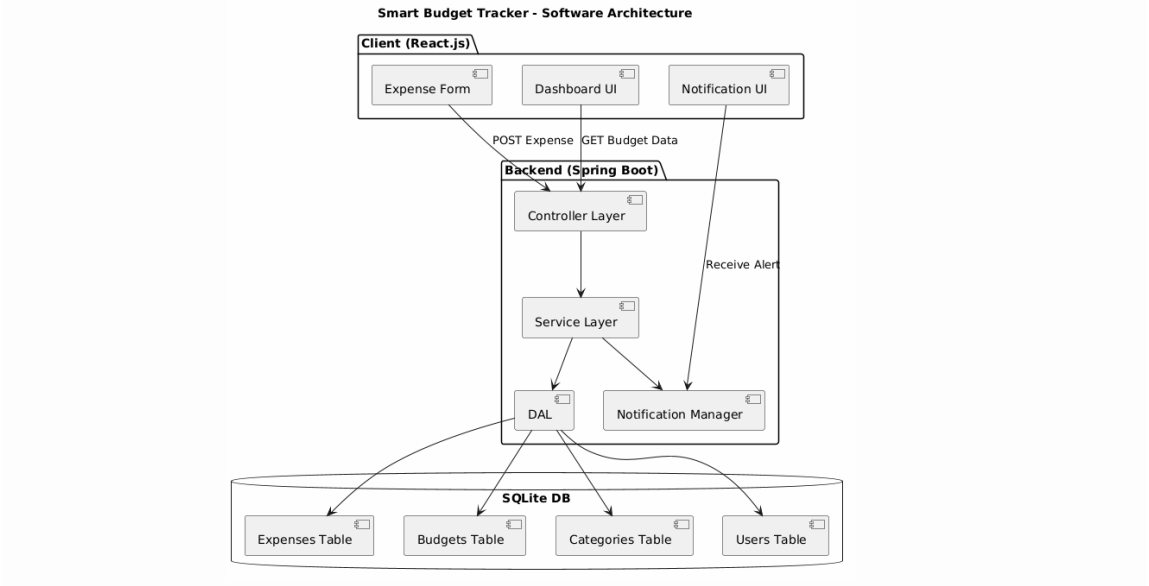

Figure 6: Software Architecture Overview