Deliverable 1: Budget Monitoring System

1 Project Specification

The **Budget Monitoring System** is a desktop application that enables users to:

- Create accounts
- Manage budgets linked to various cards
- Track expenses by category

It is built using **JavaFX** for the user interface and **Spring Boot** for the back-end services. It leverages **Spring Data JPA** for persistence and uses a relational database (e.g., PostgreSQL) for storage.

Key Features:

- User registration and login with secure password hashing
- Creating and linking budgets (cards) to user accounts
- Adding, editing, and deleting expense records (products) while automatically updating the available budget
- Filtering expense history by category and date
- Generating expense reports with graphical representations (e.g., pie charts) to show percentage spending per category

2 Functional Requirements

1. **User Account Management**
   - Users can register with a unique username and password.
   - Passwords are stored in a hashed format for security.
   - Users can log in and log out.
2. **Budget and Card Management**

- Users can add one or more budgets (cards) by entering a card number and an initial amount.
- The system maintains a link between the user and each budget.

3. **Expense Management**
   - Users can add a new expense (product) by entering the name, price, date, and selecting a category.
   - When an expense is added, the system deducts its price from the corresponding card's budget.
   - Users can edit an existing expense by providing its ID and new details.
   - Users can delete an expense by providing its ID; upon deletion, the expense's price is restored to the corresponding budget.
4. **Expense History and Reporting**
   - Users can view their expense history.
   - The system provides filtering options by category and by date.
   - Reports (such as pie charts) display how much was spent in each category (e.g., 20% Food, 10% Shoes, etc.).
5. **Navigation and UI**
   - The system provides a menu-driven UI for navigating between screens (Home, Manage Expenses, Manage Cards, History, and Expense Charts).

3 Use Case Models

Use Case Model 1: Manage Expenses

Use Case: Manage Expenses
Level: User Goal
Primary Actor: Registered User

Main Success Scenario

1. **User Logs In**
   The registered user logs into the system with valid credentials.
2. **Navigate to "Manage Expenses"**
   The user selects the "Manage Expenses" option from the main screen.
3. **Select Operation**
   The system displays three operations:

- **Add Expense:** The system shows a form with fields for *Name, Price, Date, Category,* and *Card Selection* (ID is auto-generated).
- **Edit Expense:** The system shows a form with fields for *Expense ID, Name, Price, Date, Category,* and *Card Selection*.
- **Delete Expense:** The system shows a form with fields for *Expense ID* and *Card Selection* only.

4. **Fill in Required Information**
   The user enters the required details and presses the *Save* button.

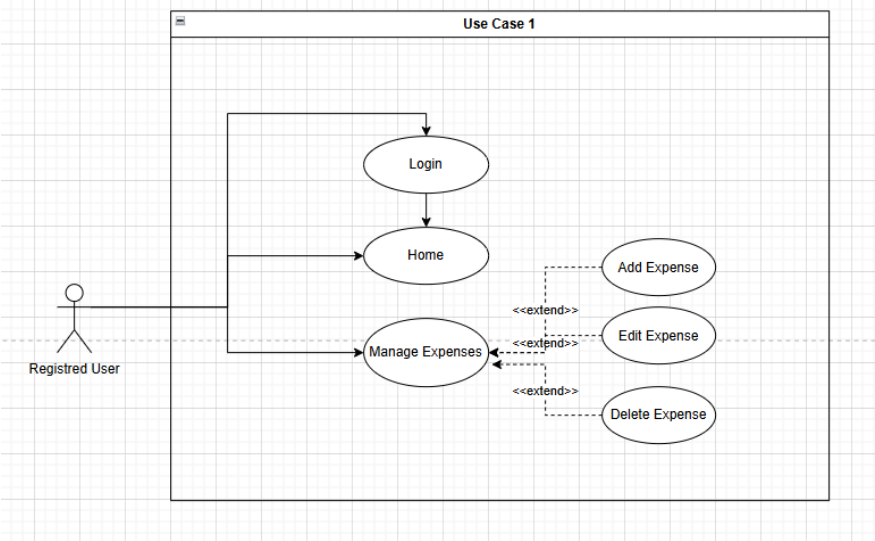
5. **System Validates and Performs Operation**

   - The system validates the inputs.
   - If valid, it updates the expense records and adjusts the budget:
     * **Add:** Creates a new expense and subtracts its price from the card's budget.
     * **Edit:** Updates the existing expense and adjusts the budget if necessary.
     * **Delete:** Removes the expense and adds its price back to the card's budget.

6. **Confirm and Update Display**
   The system confirms the operation (e.g., "Expense added successfully") and refreshes the display.

Extensions

- **Invalid Data:** Displays an error if non-numeric price or improperly formatted date is entered.
- **Nonexistent Expense ID:** Notifies the user if the provided expense ID does not exist (for edit or delete).
- **New Category:** Automatically creates a new category if the entered category does not exist.



Use Case Model 2: User Account Management (Sign In / Create Account)

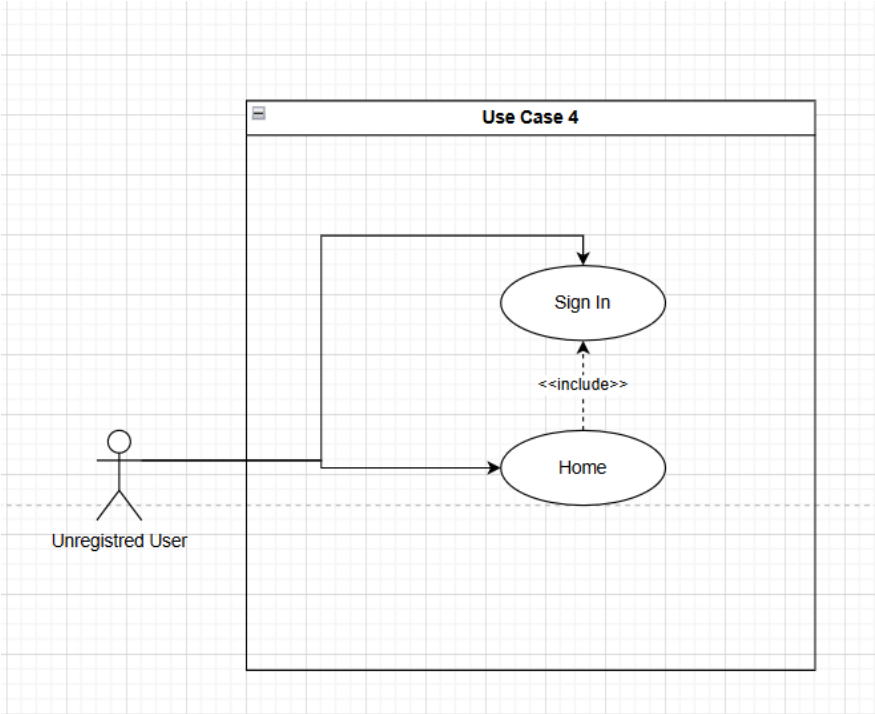
Use Case: User Sign In / Create Account
Level: User Goal
Primary Actor: Prospective or Registered User

Main Success Scenario

1. **Sign Up:**
   - The prospective user selects *Create Account* and enters a unique username, password, and any required details.
   - The system validates the data, hashes the password, and creates the account.
2. **Navigation:** The user is directed to the main screen upon successful sign in or account creation.

Extensions

- If the username is already taken, prompt for a different username.
- If the passwords do not match or credentials are incorrect, display an error message.



Use Case Model 3: Admin Operations

Use Case: Admin Login and User Management
Level: System Management
Primary Actor: Admin

Main Success Scenario

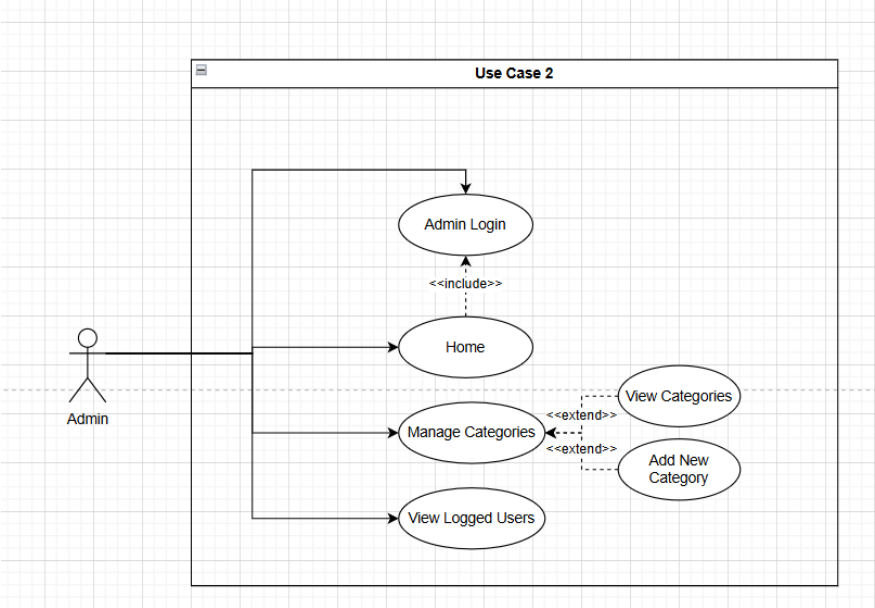
1. **Admin Logs In:**
   The Admin enters predefined credentials (username: `admin`, password: `admin`). The system validates these credentials.

2. **Access Admin Dashboard:**
   The Admin is directed to the Admin Dashboard.

3. **User Management:**
   From the dashboard, the Admin can view all registered users and manage categories.

Extensions

– If admin credentials are invalid, display an error message.



Use Case Model 4: Manage Cards

Use Case: Manage Cards
Level: User Goal
Primary Actor: Registered User

Main Success Scenario

1. **User Logs In and Navigates to Manage Cards:**
   The user logs in and selects the *Manage Cards* option.

2. **Add New Card:**
   The user clicks the *Add New Card* button.

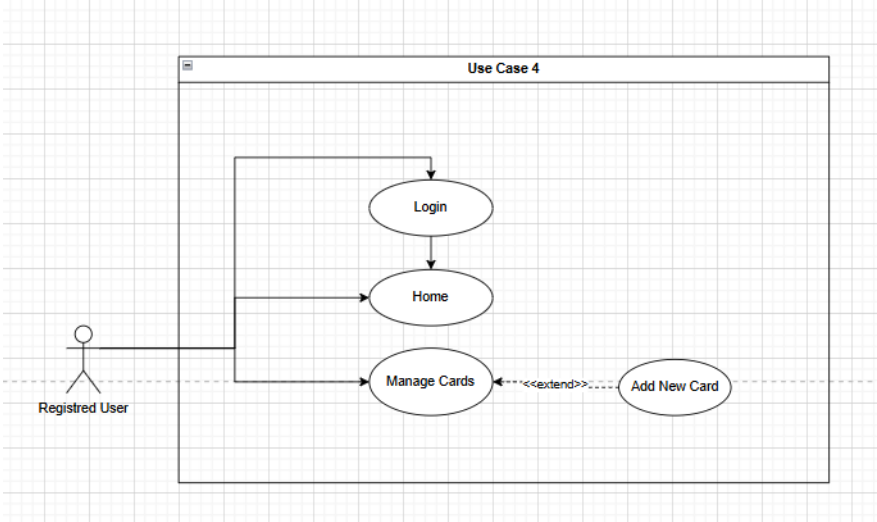
3. **Fill in Card Details:**
   The system displays a form with fields for *Card Number* and *Initial Amount*.

4. **Save Card:**
   The user enters the card details and clicks *Save*.

5. **System Validates and Updates:**
   The system validates the data, creates a new budget entry, links it to the user, and updates the display.

Extensions

– Displays an error message if the card number already exists or the data is invalid.



Use Case Model 5: Expense History

Use Case: View Expense History
Level: User Goal
Primary Actor: Registered User

Main Success Scenario

1. **User Logs In and Navigates to Expense History:**
   The user logs in and selects the *Expense History* option.

2. **Display Expense List:**
   The system displays all recorded expenses.

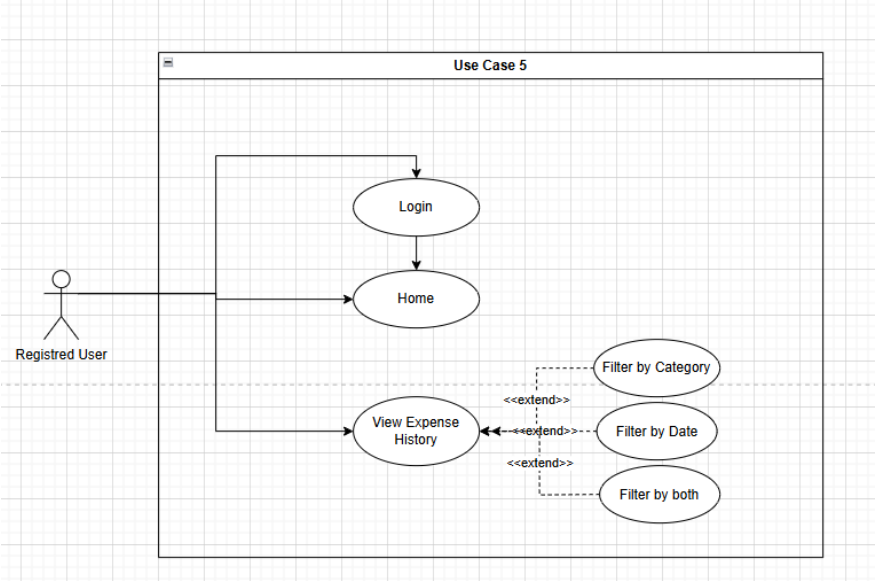
3. **Select Filters:**
   The user selects a category filter and/or picks a date (or both).

4. **Apply Filter:**
   The user clicks the *Start Filter* button.

5. **System Updates Display:**
   The system retrieves and displays the expenses that match the selected filter criteria. If no filters are applied, all expenses are shown.

Extensions

– If no matching expenses are found, the system notifies the user.



Use Case Model 6: Chart Expenses

Use Case: Generate Expense Report
Level: User Goal
Primary Actor: Registered User

Main Success Scenario

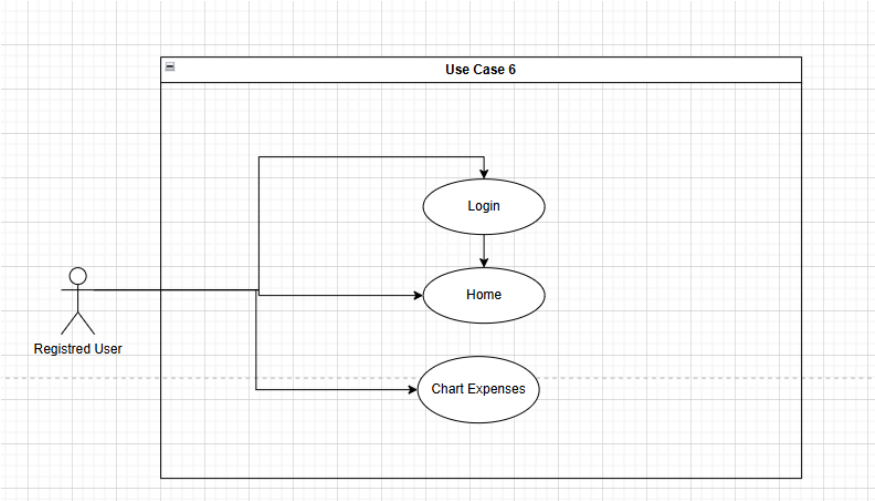
1. **User Logs In and Navigates to Expense Charts:**
   The user logs in and selects the *Expense Charts* option.

2. **Display Report:**
   The system displays a pie chart showing the percentage of spending per category.

3. **Refresh or Adjust:**
   The user may refresh the chart to see updated data.

Extensions

– If there are no expenses, the system displays an appropriate message.



4 Supplementary Specification

4.1 Non-functional Requirements

1. **Usability**
   - **Requirement:** The user interface must be intuitive and easy to navigate.
   - **Rationale:** Users should be able to manage budgets and expenses with minimal training.
2. **Security**
   - **Requirement:** User passwords must be hashed (using SHA-256) and stored securely.
   - **Rationale:** Secure storage of credentials is vital to protect user data.
3. **Performance**
   - **Requirement:** Each screen should load within 2 seconds under normal conditions.
   - **Rationale:** Fast screen loads ensure a smooth user experience.
4. **Scalability**
   - **Requirement:** The system must support future expansion (e.g., additional reporting features).
   - **Rationale:** A scalable design accommodates increasing user demands and feature additions.

5 Design Constraints

- **Programming Language:** Java (version 17)
- **UI Framework:** JavaFX
- **Back-end Framework:** Spring Boot
- **Persistence:** Spring Data JPA with a relational database (e.g., PostgreSQL)
- **Architectural Pattern:** MVC (Model-View-Controller) and a three-layered architecture.
- **Security Tools:** Spring Security (if further enhancements are needed)
- **Development Tools:** Maven for dependency management and build automation

6 Glossary

- **Budget:** A financial limit assigned to a particular card, representing the amount available for spending.
- **Expense/Product:** A record of spending that decreases the available budget.
- **Category:** A classification for an expense (e.g., Food, Shoes, Entertainment).
- **User:** An individual who registers and uses the system to manage budgets and track expenses.
- **Card:** A financial instrument or account to which a budget is linked.
- **REST API:** A web service that follows the Representational State Transfer (REST) architectural style.
- **JavaFX:** A Java library used to build graphical user interfaces (GUIs).
- **Spring Boot:** A framework for building stand-alone, production-grade Spring based applications.