DIGT2107: Practice of Software Development

Project Iteration 1.2: Detailed User Stories and Requirements
Course: DIGT2107 – Fall Term 2025
Instructor: Dr. May Haidar
Due Date: October 05, 2025

Project Name: SmartBudget – Personal Finance Tracker
Team Number: Team #3
Team Members: Muhammed Ahmed, Mohsen Pasdar, Alia Hagi-Dhaffe

1. Introduction

This document defines the **requirements**, **use cases**, and **user stories** for SmartBudget, a desktop, offline-first personal finance tracker. It focuses on a realistic MVP for two semesters, written in clear, user-centric language and aligned with the object-oriented design practices covered in class.

2. Requirements

2.1 Functional Requirements

- **FR1 – Transactions:** The system must let the user record, edit, and delete **income and expense** transactions (amount, date, category, optional note), and persist them locally so they remain available across sessions.

- **FR2 – Browsing & Filters:** The system must let the user **view transactions** and apply **filters** (e.g., by date range and category) to review specific spending.

- **FR3 – Budgets & Alerts:** The system must let the user **set a monthly spending limit per category** and **notify** the user when spending **reaches or exceeds** that limit.

- **FR4 – Summaries & Charts:** The system must provide a **period summary** (income, expenses, balance) and at least one **visual breakdown** of spending by category.

2.2 Non-Functional Requirements

- **Usability:** The system should present a **simple, consistent JavaFX interface** with clear labels, readable fonts, and sensible navigation between Transactions, Budgets, and Reports.

- **Performance:** The system should **load typical datasets quickly** and remain responsive during common operations.

- **Reliability:** The system should **save changes immediately** to local storage to minimize data loss and **handle invalid inputs gracefully** with clear feedback.

- **Privacy/Offline:** The system should **operate fully offline** and **avoid sending data over a network**.

- **Portability:** The system should run on **Windows, macOS, and Linux** with a supported Java runtime.

- **Maintainability:** The system should adopt a **modular structure (e.g., MVC)** and include **unit tests** for core business logic to enable straightforward updates.

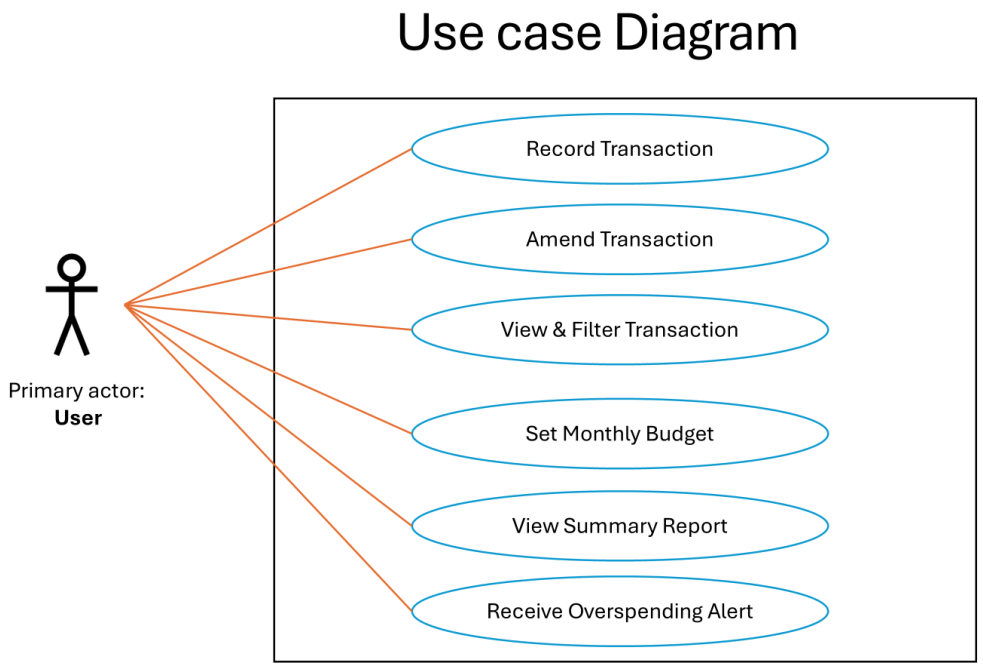
Traceability note: FR1→UC-1/UC-2, FR2→UC-3, FR3→UC-4/UC-6, FR4→UC-5.

3. Use Cases

3.1 Actors

- **User** – an individual tracking personal finances

3.2 Use-Case Diagram



3.3 Use-Case Descriptions

UC-1: Record Transaction

- **Actor:** User
- **Goal:** Add a new income or expense.
- **Preconditions:** App is running.
- **Main Flow:** (1) User initiates adding a transaction. (2) System asks for type, amount, date, category, optional note. (3) User confirms. (4) System saves and updates totals.
- **Alternates:** Missing/invalid fields → system requests correction.
- **Postconditions:** Transaction is persisted; summaries reflect it.

UC-2: Amend Transaction

- **Actor:** User
- **Goal:** Modify or delete an existing entry.
- **Preconditions:** At least one transaction exists.

- **Main Flow:** (1) User selects a transaction. (2) System shows details. (3) User edits or deletes. (4) System saves and refreshes totals.

- **Alternates:** Entry not found → system informs user.

- **Postconditions:** History and aggregates reflect the change.

UC-3: View & Filter Transactions

- **Actor:** User

- **Goal:** Review past entries by period/category.

- **Main Flow:** (1) User requests transaction list. (2) System displays entries. (3) User applies filters. (4) System narrows results and updates totals for the selection.

- **Alternates:** No matches → show empty results and zero totals.

UC-4: Set Monthly Budget

- **Actor:** User

- **Goal:** Define or adjust a per-category spending limit for the current month.

- **Main Flow:** (1) User selects category. (2) System requests limit. (3) User confirms. (4) System stores limit and shows remaining allowance.

- **Alternates:** Invalid limit → system asks for a valid amount.

- **Postconditions:** Budgets available to alerts and reports.

UC-5: View Summary Report

- **Actor:** User

- **Goal:** See totals and category breakdowns for a period.

- **Main Flow:** (1) User selects a period. (2) System computes totals and breakdown. (3) System presents figures and at least one chart.

- **Alternates:** No data in period → system shows zeros and a note.

UC-6: Receive Overspending Alert

- **Actor:** User
- **Goal:** Be notified when a category exceeds its limit.
- **Preconditions:** At least one budget is defined.
- **Main Flow:** (1) System compares spending to limits as transactions are added/viewed. (2) System signals when a limit is reached or exceeded.
- **Alternates:** No budgets defined → no alert.
- **Postconditions:** User is aware of budget status.

4. User Stories

- **US1**
  As a user, I want to record an expense or income so that my daily finances are tracked.
  Acceptance Criteria: Requires type, amount, date, and category; data persists; appears in history and totals.
  Priority: High

- **US2**
  As a user, I want to edit or delete a transaction so that my records stay accurate.
  Acceptance Criteria: Updates or removals reflect immediately; totals recalculate.
  Priority: High

- **US3**
  As a user, I want to view and filter transactions by date and category so that I can review specific spending.
  Acceptance Criteria: Filters by category and date range; filtered totals update dynamically.
  Priority: Medium

- **US4**
  As a user, I want to set a monthly budget per category so that I can control my spending.
  Acceptance Criteria: Limit per category stored for the current month; remaining allowance shown.
  Priority: Medium

- **US5**
  As a user, I want to be alerted when spending exceeds a budget so that I can adjust early.
  Acceptance Criteria: Alert triggers at or above the limit; visual indicator appears in the app.

Priority: Medium

- **US6**
  As a user, I want a monthly summary of income, expenses, and balance so that I can track my financial status.
  Acceptance Criteria: Totals computed for selected period; category breakdown displayed.
  Priority: High

- **US7**
  As a user, I want a simple chart of spending by category so that I can quickly spot patterns.
  Acceptance Criteria: At least one visual (pie or bar chart) shows category distribution.
  Priority: Medium

- **US8**
  As a user, I want my saved data available when I reopen the app so that I can continue seamlessly.
  Acceptance Criteria: Transactions and budgets reload automatically when the application starts.
  Priority: High

5. Iteration and Backlog Plan

The SmartBudget project will be developed through multiple iterations across the **Fall 2025** and **Winter 2026** terms. Each iteration builds upon previous work, aligning with Agile and incremental development principles.

Iteration 1 (Weeks 1–2 – Vision and Planning)

- Deliverable 1.1 – Vision Document: Submitted.

Iteration 2 (Weeks 3–5 – Requirements and Use Cases)

- Deliverable 1.2 – Requirements, Use Cases, and User Stories: **Current deliverable.**

Iteration 3 (Weeks 6–9 – Initial Development and Testing)

- Deliverable 2.1 – Testing Document: Unit testing setup, prototype development, and initial test results.

Iteration 4 (Weeks 10–12 – Mid-Project Peer Review and Iteration)

- Deliverable 2.2 – Full Documentation, Prototype, and Presentation

Iteration 5 (Weeks 1–2 – Advanced Software Design)

- Deliverable 3.1 – Design Documentation: Class, sequence, and activity diagrams.

Iteration 6 (Weeks 3–4 – GUI Development and Testing)

- Deliverable 3.1 (Update): GUI implementation and initial testing.

Iteration 7 (Weeks 6–9 – Refactoring and Implementation)

- Deliverable 3.2 – Completed Implementation: Refactored, optimized codebase with all core features finalized.

Iteration 8 (Weeks 10–11 – Final Testing and Documentation)

- Deliverable 3.3 – Final Documentation Preparation

Iteration 9 (Week 12 – Block Week – Final Presentation and Submission)

- Deliverable 3.3 – Final Project Submission

6. Conclusion

This deliverable defines a **clear, achievable MVP** for *SmartBudget* that emphasizes user-centred design, minimal complexity, and iterative refinement.

The scope—transactions, filtering, budgeting, alerts, and summaries—is well balanced for two semesters and aligns with Agile best practices and object-oriented design foundations.