DIGT2107: Practice of Software Development

Project Iteration 2.1:

Testing Document

Project Name:

SmartBudget – Personal Finance Tracker

Team Number:

Team #3

Team Members:

Muhammed Ahmed, Mohsen Pasdar, Alia Hagi-Dhaffe

Date:

November 02, 2025

Table of Contents

1 Introduction 1
2 Updated Requirements 1
2.1 Functional Requirements 1
2.2 Non-Functional Requirements 2
3 Updated Use Cases 2
3.1 Actors 2
3.2 Use-Case Diagram 2
3.3 Use-Case Descriptions 3
3.3.1 UC-1: Record Transaction 3
3.3.2 UC-2: Amend Transaction 3
3.3.3 UC-3: View & Filter Transactions 4
3.3.4 UC-4: Set Monthly Budget 4
3.3.5 UC-5: View Summary Report 4
3.3.6 UC-6: Receive Overspending Alert 4
4 Updated User Stories 5
4.1 User Story 1: Record a Transaction 5
4.2 User Story 2: Edit or Delete a Transaction 5
4.3 User Story 3: View and Filter Transactions 6
4.4 User Story 4: Set a Monthly Budget 6
4.5 User Story 5: Receive Overspending Alerts 6
4.6 User Story 6: View Monthly Summary 7
4.7 User Story 7: View Spending Chart 7
4.8 User Story 8: Load Saved Data on Startup 8
5 Test Plan 8
5.1 Testing Objectives and Scope 8
5.2 Testing Approach (Unit Testing Focus for Iteration 3) 9
5.3 Requirement-to-Test Case Traceability Matrix 9
5.4 Mapping to Use Cases and User Stories 12
5.5 Detailed Test Case Descriptions 13
5.5.1 TC-001 13
5.5.2 TC-002 14
5.5.3 TC-003 14
5.5.4 TC-004 15
5.5.5 TC-005 15
5.5.6 TC-006 16
5.5.7 TC-007 16
5.5.8 TC-008 16
5.5.9 TC-009 17
5.5.10 TC-010 17
5.5.11 TC-011 18

5.5.12 TC-012 18
5.5.13 TC-013 18
5.5.14 TC-014 19
5.5.15 TC-015 19
5.5.16 TC-016 19
5.5.17 TC-017 20
5.5.18 TC-018 20
5.5.19 TC-019 21
5.5.20 TC-020 21
5.5.21 TC-021 21
5.5.22 TC-022 22
5.5.23 TC-023 22
5.5.24 TC-024 22
5.5.25 TC-025 23
5.5.26 TC-026 23
5.5.27 TC-027 23
5.5.28 TC-028 24
5.5.29 TC-029 24
5.5.30 TC-030 25
5.5.31 TC-031 25
5.5.32 TC-032 25
6 Iteration Plan and Backlog 26
6.1 Iteration Plan 26
6.1.1 Iteration 1 (Weeks 1–2 – Vision and Planning) 26
6.1.2 Iteration 2 (Weeks 3–5 – Requirements and Use Cases) 26
6.1.3 Iteration 3 (Weeks 6–9 –Testing Document) 26
6.1.4 Iteration 4 (Weeks 10–12 – Mid-Project Peer Review) 26
6.1.5 Iteration 5 (Weeks 1–2 – Advanced Software Design) 26
6.1.6 Iteration 6 (Weeks 3–4 – GUI Development and Testing) 26
6.1.7 Iteration 7 (Weeks 6–9 – Refactoring and Implementation) 27
6.1.8 Iteration 8 (Weeks 10–11 – Final Testing and Documentation) 27
6.1.9 Iteration 9 (Week 12 – Block Week – Final Presentation and Submission) 27
6.2 Updated Backlog 27
7 Conclusion 27

1 Introduction

This document represents **Deliverable 2.1 – Testing Documents** for *Project Iteration 3* of the ***SmartBudget – Personal Finance Tracke***r. It builds upon the previous deliverables, which defined the vision, requirements, and use cases of the system, and focuses on developing the foundation for systematic testing of the application. The purpose of this deliverable is to outline the testing plan and documentation that will guide the validation of the system's functional and non-functional requirements in future iterations.

The primary scope of this document is to define the testing framework for the ***SmartBudget*** application by presenting the requirement-to-test case traceability matrix and the detailed test case descriptions for all identified features. Although no physical test execution or results are included in this iteration, this deliverable ensures that each functional component is mapped to corresponding test cases, supporting complete coverage and traceability. The testing effort in this iteration centers around unit testing at the component level, ensuring that the individual features are verifiable and ready for future phases.

This document is structured to first summarize any updates to the requirements and use cases based on feedback from earlier iterations. It then introduces the comprehensive test plan, which includes both the traceability matrix and detailed test case descriptions prepared for each feature. Finally, the updated iteration plan and backlog are provided to show progress and upcoming milestones for the next phase of the project. Together, these sections demonstrate the team's readiness to proceed from design and planning toward implementation and testing within an Agile development framework.

2 Updated 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

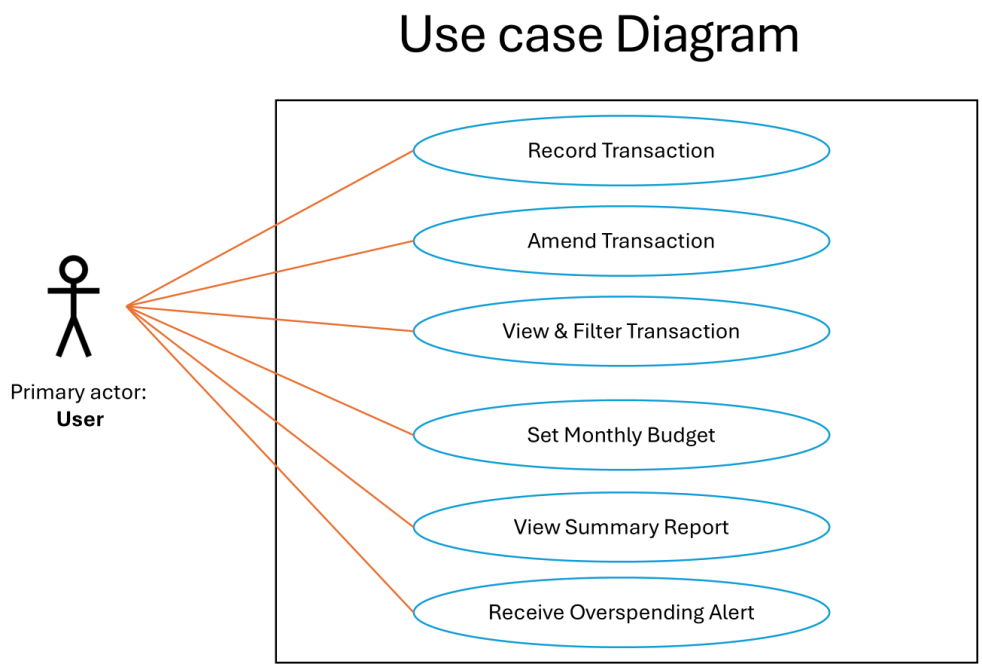
- **Usability**: The system should present a simple, consistent JavaFX interface with clear labels, readable fonts, and sensible navigation between Transactions, Budgets, and Reports.
- **Performance**: The system should load typical datasets quickly and remain responsive during common operations.
- **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.

3 Updated Use Cases

3.1 Actors

- **User** – an individual tracking personal finance.

3.2 Use-Case Diagram



3.3 Use-Case Descriptions

3.3.1 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.

3.3.2 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.

3.3.3 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.

3.3.4 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.

3.3.5 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.

3.3.6 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 Updated User Stories

4.1 User Story 1: Record a Transaction

- **Description**: *As a user, I want to record an expense or income so that my daily finances are accurately tracked.*
- **Traceability**: FR1, NFR – Reliability, Usability
- **Preconditions:** The application is open and the user has access to the "Add Transaction" feature.
- **Acceptance Criteria:**
  - The user can enter transaction details (amount, type, date, and category).
  - The transaction is saved immediately after confirmation and appears in the list of transactions.
  - The total income/expense summary updates automatically.
  - The system provides feedback confirming successful entry.
- **Postconditions:** Transaction is stored locally and persists when the application restarts.
- **Priority:** High

4.2 User Story 2: Edit or Delete a Transaction

- **Description**: *As a user, I want to edit or delete a transaction so that my financial records stay accurate.*
- **Traceability:** FR1, NFR – Reliability, Usability
- **Preconditions:** At least one transaction exists in the system.
- **Acceptance Criteria:**
  - The user can open an existing transaction and modify any editable field.
  - The user can delete an entry after a confirmation prompt.
  - Updated or deleted transactions reflect immediately in totals and summaries.
  - The system prevents accidental deletion through a confirmation step.
- **Postconditions:** Transaction list and totals are updated to reflect the change.
- **Priority:** High

4.3 User Story 3: View and Filter Transactions

- **Description**: *As a user, I want to view and filter transactions by date and category so that I can easily review my spending history.*
- **Traceability:** FR2, NFR – Usability, Performance
- **Acceptance Criteria:**
  - The user can view all past transactions in a table or list.
  - The user can filter transactions by category, date range, or keyword.
  - The filtered list updates immediately after criteria are applied.
  - The user can clear filters to return to the full view.
- **Postconditions:** Filtered or full lists remain visible until the user changes the view.
- **Priority:** High

4.4 User Story 4: Set a Monthly Budget

- **Description**: *As a user, I want to set a monthly budget for each spending category so that I can manage my expenses more effectively.*
- **Traceability:** FR3, NFR – Usability, Reliability
- **Preconditions:** User has at least one spending category defined.
- **Acceptance Criteria:**
  - The user can set or update a monthly spending limit for a selected category.
  - The budget amount is saved and displayed for the current month.
  - The system visually shows remaining budget as transactions are added.
  - Invalid inputs (e.g., negative values or empty fields) are rejected with feedback.
- **Postconditions:** Budget is stored and available for alert and reporting features.
- **Priority:** Medium

4.5 User Story 5: Receive Overspending Alerts

- **Description**: *As a user, I want to receive an alert when spending exceeds my budget so that I can adjust my spending habits early.*
- **Traceability:** FR3, NFR – Reliability, Usability

- **Preconditions:** At least one active budget is set for the month.
- **Acceptance Criteria:**
  - The system checks total spending against each category limit.
  - When spending meets or exceeds the limit, the system displays a clear visual alert.
  - The alert indicates the category and amount over budget.
  - The alert remains visible until acknowledged or budget is reset.
- **Postconditions:** User is aware of overspending status and can adjust budgets accordingly.
- **Priority:** Medium

4.6 User Story 6: View Monthly Summary

- **Description**: *As a user, I want to view a summary of my income, expenses, and remaining balance so that I can assess my financial status.*
- **Traceability:** FR4, NFR – Usability, Performance
- **Acceptance Criteria:**
  - The user can select a reporting period (e.g., month or custom range).
  - The system generates totals for income, expenses, and balance.
  - The summary is displayed in an easy-to-read format.
  - When no data is available, a clear “no records found” message is shown.
- **Postconditions:** Generated summaries can be revisited without re-entry until the user changes the period.
- **Priority:** High

4.7 User Story 7: View Spending Chart

- **Description**: *As a user, I want to see a chart of my spending by category so that I can identify trends and make informed financial decisions.*
- **Traceability:** FR4, NFR – Usability, Performance
- **Acceptance Criteria:**
  - The system presents at least one visual chart (pie or bar) showing spending by category.
  - Charts update automatically when transactions or filters change.

- The visualization is easy to interpret with clear labels and legends.

- **Postconditions:** Chart view remains synchronized with current transaction data.
- **Priority:** Low

4.8 User Story 8: Load Saved Data on Startup

- **Description**: *As a user, I want my saved transactions and budgets to load automatically when I open the app so that I can continue seamlessly.*
- **Traceability:** FR1, FR3, NFR – Reliability, Performance
- **Preconditions:** Previously saved data exists locally.
- **Acceptance Criteria:**
  - The system loads all stored transactions and budgets automatically at startup.
  - The system notifies the user if no data is found.
  - Loaded data appears accurately within corresponding views (Transactions, Budgets, Reports).
- **Postconditions:** User resumes from previous state with no manual loading required.
- **Priority:** High

5 Test Plan

5.1 Testing Objectives and Scope

The main goal of this testing phase is to make sure that each part of the SmartBudget system works as expected and meets the requirements defined in earlier deliverables. In this iteration, the focus is on preparing the groundwork for testing by defining clear test cases and linking them to each requirement. The testing will mainly cover the system's core functions such as recording transactions, filtering data, setting budgets, and viewing summaries. Although the tests are not yet executed, this plan ensures that every feature is ready for validation in later stages of development and that the testing process will be organized, consistent, and easy to track.

5.2 Testing Approach (Unit Testing Focus for Iteration 3)

The testing approach for this iteration follows a unit testing strategy, where each function or module of the SmartBudget system will be tested individually. This helps identify issues early and ensures that small parts of the program work correctly before being combined into a larger system. The testing plan focuses on designing tests using JUnit in Java to verify calculations, data handling, and interface responses. Each test case will check one small piece of functionality based on the requirements and user stories. Future iterations will expand this to include integration and GUI testing once the main components are implemented.

5.3 Requirement-to-Test Case Traceability Matrix

This section shows how each system requirement is linked to one or more test cases. The traceability matrix helps make sure that all functional and non-functional requirements are covered by at least one planned test. It provides a clear view of what will be tested, how it will be verified, and ensures that no part of the system is left untested. This also helps track future changes and maintain consistency between requirements and test documentation.

Table 1: Requirement-to-Test Case Traceability Matrix

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Coverage
FR-001	Record, edit, delete income/expense transactions; persist locally across sessions	TC-001	Add expense with valid inputs (amount, date, category, optional note)	
FR-001	— same as above —	TC-002	Add income with valid inputs	
FR-001	— same as above —	TC-003	Reject invalid amount (non-numeric / negative)	
FR-001	— same as above —	TC-004	Reject missing required fields (amount, date, type) with clear feedback	
FR-001	— same as above —	TC-005	Edit an existing transaction and verify totals update	

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Coverage
FR-001	— same as above —	TC-006	Delete transaction with confirmation prompt; totals update	
FR-001	— same as above —	TC-007	Cancel delete preserves the transaction	
FR-001	— same as above —	TC-008	Data persists across app restart (transactions remain available)	
FR-002	View transactions and apply filters (date range, category)	TC-009	View all transactions in list/table	
FR-002	— same as above —	TC-010	Filter by category (single category)	
FR-002	— same as above —	TC-011	Filter by date range (start–end)	
FR-002	— same as above —	TC-012	Combined filters (date range + category) narrow results and update totals	
FR-002	— same as above —	TC-013	No matches → empty results + zero totals shown	
FR-002	— same as above —	TC-014	Clear filters restores full transaction list	
FR-002	— same as above —	TC-015	Keyword search narrows list	
FR-003	Set monthly spending limit per category and alert on reaching/exceeding	TC-016	Set monthly budget (valid positive amount)	
FR-003	— same as above —	TC-017	Reject invalid budget inputs (negative/blank) with feedback	
FR-003	— same as above —	TC-018	Update an existing budget; remaining allowance recalculates	
FR-003	— same as above —	TC-019	Alert triggers when spending equals the limit	
FR-003	— same as above —	TC-020	Alert triggers when spending exceeds the limit; shows overage amount	

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Coverage
FR-003	— same as above —	TC-021	No budgets defined → no alert appears	
FR-004	Provide period summary (income, expenses, balance) + at least one chart	TC-022	Monthly summary totals compute correctly (income, expenses, balance)	
FR-004	— same as above —	TC-023	Custom range summary computes correctly	
FR-004	— same as above —	TC-024	No data in selected period → zeros + “no records found” note	
FR-004	— same as above —	TC-025	Category chart (pie/bar) reflects data accurately with clear labels	
FR-004	— same as above —	TC-026	Chart and summary auto-update when transactions or filters change	
NFR-Usability	Simple, consistent JavaFX UI; clear labels; readable fonts; sensible navigation	TC-027	Heuristic usability review of labels, font sizes, and navigation between Transactions/Budgets/Reports	
NFR-Performance	Responsive under typical datasets; quick loads	TC-028	Load 1–5k transactions; ensure UI remains responsive (no noticeable lag on list/filter)	
NFR-Reliability	Immediate save to local storage; graceful handling of invalid input with clear feedback	TC-029	Immediate persistence after create/edit/delete; verify storage writes and recovery on restart	
NFR-Privacy/Offline	Full offline operation; no network transmission	TC-030	Run with network disabled; verify all functions work and no network calls are made	
NFR-Portability	Runs on Windows, macOS, Linux (supported Java runtime)	TC-031	Smoke run on 3 platforms; application starts and core flows (add/view) work	

Requirement ID	Requirement Description	Test Case ID	Test Case Description	Coverage
NFR-Maintainability	Modular (e.g., MVC) structure; unit tests for core business logic	TC-032	Project structure review + presence of unit tests for core model/services	

5.4 Mapping to Use Cases and User Stories

This section connects each test case to the related use cases and user stories. It ensures that all user actions and scenarios described by the users are properly represented in the testing plan. By mapping test cases to these elements, the document confirms that the system will be tested from both technical and user perspectives, covering real-life usage and expected workflows.

Table 2: Mapping Test Cases to Use Cases

Use Case ID	Use Case Goal	Test Case ID(s)	Coverage
UC-1: Record Transaction	Add income/expense and update totals	TC-001, TC-002, TC-003, TC-004, TC-022, TC-026	
UC-2: Amend Transaction	Modify or delete an existing entry; refresh totals	TC-005, TC-006, TC-007, TC-022, TC-026	
UC-3: View & Filter Transactions	Review by period/category; empty results path	TC-009, TC-010, TC-011, TC-012, TC-013, TC-014, TC-015	
UC-4: Set Monthly Budget	Define/adjust per-category limit	TC-016, TC-017, TC-018	
UC-5: View Summary Report	Select period; show totals and chart; no-data path	TC-022, TC-023, TC-024, TC-025	
UC-6: Receive Overspending Alert	Notify on reach/exceed budget; precondition = budgets defined	TC-019, TC-020, TC-021	

Table 3: Mapping Test Cases to User Stories

Use Story ID	Story Summary	Related FR	Test Case ID(s)	Coverage
US-1: Record a Transaction	Add expense or income; immediate save; totals update; success feedback	FR-001	TC-001, TC-002, TC-004, TC-022	
US-2: Edit or Delete a Transaction	Edit/delete; confirmation; totals/summaries update	FR-001	TC-005, TC-006, TC-007, TC-022	
US-3: View & Filter Transactions	View all; filter by category/date/keyword; clear filters	FR-002	TC-009, TC-010, TC-011, TC-012, TC-014, TC-015	
US-4: Set a Monthly Budget	Set/update budget; show remaining; reject invalid	FR-003	TC-016, TC-017, TC-018	
US-5: Receive Overspending Alerts	Trigger at ≥ limit; show category and overage; remains visible	FR-003	TC-019, TC-020, TC-021	
US-6: View Monthly Summary	Select period; totals computed; clear message for no data	FR-004	TC-022, TC-024	
US-7: View Spending Chart	Present chart; auto-updates; clear labels/legend	FR-004	TC-025, TC-026	
US-8: Load Saved Data on Startup	Auto-load transactions/budgets at startup; notify if none	FR-001 / FR-003	TC-008	

5.5 Detailed Test Case Descriptions

This section provides detailed descriptions of all planned test cases that will be used to verify the ***SmartBudget*** system. Each test case includes its ID, related requirement, preconditions, steps, and expected results. These details will guide future testing activities, making it easy for developers and testers to perform and track tests systematically once it is implemented.

5.5.1 TC-001

- **Title:** Add Expense – Valid Inputs
- **Related Requirement ID(s):** FR-001; UC-1; US-1
- **Preconditions:** App is running; “Transactions” view available.

- **Test Steps:**
  1- Open Transactions; Click “Add Transaction”; Select Expense.
  2- Enter valid amount (e.g., 25.50), pick a valid date, choose category (e.g., Food), add optional note.
  3- Click “Save”.
- **Expected Results:** Transaction appears in list; totals/summary update; success feedback shown; data written to local storage.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.2 TC-002

- **Title:** Add Income – Valid Inputs
- **Related Requirement ID(s):** FR-001; UC-1; US-1
- **Preconditions:** App is running; “Transactions” view available.
- **Test Steps:**
  1- Open Transactions; Click “Add Transaction”; Select Income.
  2- Enter valid amount (e.g., 25.50), pick a valid date, choose category (e.g., Salary), add optional note.
  3- Click “Save”.
- **Expected Results:** Income row added; income and balance totals increase; success feedback; persisted.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.3 TC-003

- **Title:** Add Transaction – Reject Invalid Amount
- **Related Requirement ID(s):** FR-001; UC-1; US-1
- **Preconditions:** App is running; “Transactions” view available.
- **Test Steps:**
  1- Open Transactions; Click “Add Transaction”; Select Income/Expense.
  2- Enter non-numeric (e.g., “abc”) or negative amount.

3- Click “Save”.

- **Expected Results:** Save blocked; clear validation message; no record created; no totals change.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.4 TC-004

- **Title:** Add Transaction – Reject Missing Required Fields
- **Related Requirement ID(s):** FR-001; UC-1; US-1
- **Preconditions:** App running.
- **Test Steps:**
  1- Add Transaction.
  2- Leave amount or date or type empty.
  3- Click “Save”.
- **Expected Results:** Save blocked; specific error message; no changes persisted.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.5 TC-005

- **Title:** Edit Transaction – Totals Recalculate
- **Related Requirement ID(s):** FR-001; UC-2; US-2
- **Preconditions:** At least one existing transaction.
- **Test Steps:**
  1- Open an existing transaction.
  2- Change amount (e.g., 20 → 30).
  3- Click “Save”.
- **Expected Results:** Row updates; affected totals/summary recalculates; success feedback; persisted.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.6 TC-006

- **Title:** Delete Transaction – Confirmation and Totals Update
- **Related Requirement ID(s):** FR-001; UC-2; US-2
- **Preconditions:** At least one existing transaction.
- **Test Steps:**
  4- Open an existing transaction.
  1- Click delete.
  2- Confirm deletion.
- **Expected Results:** Row removed; totals/summary recalculates; confirmation used to prevent accidental deletion; persisted removal.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.7 TC-007

- **Title:** Delete Transaction – Cancel Preservation
- **Related Requirement ID(s):** FR-001; UC-2; US-2
- **Preconditions:** At least one existing transaction.
- **Test Steps:**
  3- Open an existing transaction.
  4- Click delete.
  1- In confirmation, click Cancel.
- **Expected Results:** No deletion; list and totals unchanged.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.8 TC-008

- **Title:** Persistence – Transactions Survive App Restart
- **Related Requirement ID(s):** FR-001; US-8
- **Preconditions:** At least one existing transaction.
- **Test Steps:**
  2- Close app.

3- Reopen app.

4- Go to Transactions.

- **Expected Results:** Previously saved transactions load automatically; no manual import required.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.9 TC-009

- **Title:** View All Transactions – Default List/Table
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** At least one existing transaction.
- **Test Steps:**

  1- Open Transactions view.
- **Expected Results:** All transactions visible in a readable table/list with key columns; totals visible.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.10 TC-010

- **Title:** Filter by Category (Single Category)
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Transactions exist across multiple categories.
- **Test Steps:**

  1- Apply category filter (e.g., Food).
- **Expected Results:** Only rows with category = Food shown; totals/summary reflect filtered subset.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.11 TC-011

- **Title:** Filter by Date Range
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Transactions exist across multiple dates.
- **Test Steps:**

  1- Set start and end date.
- **Expected Results:** Only transactions within range shown; totals reflect range.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.12 TC-012

- **Title:** Combined Filters – Date Range + Category
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Data spans categories and dates.
- **Test Steps:**

  2- Apply date range.

  1- Apply category.
- **Expected Results:** List narrows to intersection; totals update accordingly.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.13 TC-013

- **Title:** Clear Filters – No Matches
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Data set present.
- **Test Steps:**

  1- Apply filters that yield zero results.
- **Expected Results:** Empty list message + zero totals displayed clearly.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.14 TC-014

- **Title:** Clear Filters – Restore Full List
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Active filter(s) applied.
- **Test Steps:**

  1- Click “Clear Filters”.
- **Expected Results:** All transactions visible again; totals reflect full data set.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.15 TC-015

- **Title:** Keyword Search – Narrows List
- **Related Requirement ID(s):** FR-002; UC-3; US-3
- **Preconditions:** Keyword Search – Narrows List.
- **Test Steps:**

  1- Enter keyword (e.g., “coffee”) in search.
- **Expected Results:** Only matching rows shown (by configured fields: note, payee, category); totals update; clearing search restores list.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.16 TC-016

- **Title:** Set Monthly Budget – Valid Positive Amount
- **Related Requirement ID(s):** FR-003; UC-4; US-4
- **Preconditions:** Category exists (e.g., Food).
- **Test Steps:**

  1- Open Budgets.

  2- Select “Food”.

  3- Enter 200.00.

  4- Save

- **Expected Results:** Budget stored for current month; remaining allowance visible.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.17 TC-017

- **Title:** Set Monthly Budget – Reject Invalid Inputs
- **Related Requirement ID(s):** FR-003; UC-4; US-4
- **Preconditions:** Budgets view open.
- **Test Steps:**
  1- Enter negative or blank budget.
  2- Save
- **Expected Results:** Save blocked; clear error message; no budget created/changed.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.18 TC-018

- **Title:** Update Existing Budget – Recalculate Remaining
- **Related Requirement ID(s):** FR-003; UC-4; US-4
- **Preconditions:** Budget already set for category; some spending exists.
- **Test Steps:**
  1- Edit Food budget 200 → 250.
  2- Save
- **Expected Results:** New limit stored; remaining allowance recalculated and displayed.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.19 TC-019

- **Title:** Overspending Alert – Trigger at Equal to Limit
- **Related Requirement ID(s):** FR-003; UC-6; US-5
- **Preconditions:** Budget = 100 for Food; current spending = 90.
- **Test Steps:**

  1- Add Food expense of 10.
- **Expected Results:** Alert appears indicating limit reached (category shown); remains visible until acknowledged/reset.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.20 TC-020

- **Title:** Overspending Alert – Trigger when Exceeding Limit
- **Related Requirement ID(s):** FR-003; UC-6; US-5
- **Preconditions:** Budget = 100 for Food; current spending = 95.
- **Test Steps:**

  1- Add Food expense of 10 (total 105).
- **Expected Results:** Alert shows category and overage amount (e.g., +5); persists until acknowledged/reset.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.21 TC-021

- **Title:** No Budget Defined – No Alert
- **Related Requirement ID(s):** FR3; UC-6; US-5
- **Preconditions:** No budgets set.
- **Test Steps:**

  1- Add multiple expenses in any category.
- **Expected Results:** No budget alert appears.
- **Actual Result:**
- **Status:**

- **Priority:** Medium

5.5.22 TC-022

- **Title:** Monthly Summary – Correct Totals
- **Related Requirement ID(s):** FR-004; UC-5; US-6
- **Preconditions:** Month has known set of incomes/expenses.
- **Test Steps:**
  1- Open Reports.
  2- Select “This Month”.
- **Expected Results:** Income, expenses, and balance compute correctly and match dataset.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.23 TC-023

- **Title:** Custom Range Summary – Correct Totals
- **Related Requirement ID(s):** FR-004; UC-5; US-6
- **Preconditions:** Data across multiple months.
- **Test Steps:**
  1- Choose custom start/end dates across months.
- **Expected Results:** Totals reflect only the selected range; displayed clearly.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.24 TC-024

- **Title:** Summary – No Data Path
- **Related Requirement ID(s):** FR-004; UC-5; US-7
- **Preconditions:** Select a period with no transactions.
- **Test Steps:**
  1- Open Reports → empty range.
- **Expected Results:** Zeros shown for totals; clear “no records found” note.

- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.25 TC-025

- **Title:** Category Chart – Accurate Visualization
- **Related Requirement ID(s):** FR-004; UC-5; US-7
- **Preconditions:** Transactions across multiple categories.
- **Test Steps:**

  1- Open Reports with chart (pie/bar).
- **Expected Results:** Chart values/percentages match underlying totals; labels/legend are clear.
- **Actual Result:**
- **Status:**
- **Priority:** Low

5.5.26 TC-026

- **Title:** Reports/Charts Auto-Update on Data/Filter Change
- **Related Requirement ID(s):** FR-004; UC-5; US-7
- **Preconditions:** Reports open with visible chart; baseline totals noted.
- **Test Steps:**

  1- Add/edit/delete a transaction OR change filters.
- **Expected Results:** Summary and chart refresh immediately to reflect the change.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

NFR-Oriented Test Cases

5.5.27 TC-027

- **Title:** Usability Heuristics – Labels, Fonts, Navigation
- **Related Requirement ID(s):** NFR-Usability
- **Preconditions:** UI available (Transactions, Budgets, Reports).

- **Test Steps:**
  1- Inspect labels, font sizes, contrast.
  2- Walk through navigation paths between key views.
- **Expected Results:** Clear labels; readable fonts; predictable navigation; consistent UI components.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.28 TC-028

- **Title:** Performance – Typical Dataset Responsiveness
- **Related Requirement ID(s):** NFR- Performance
- **Preconditions:** Seed transactions locally.
- **Test Steps:**
  1- Launch app and open Transactions.
  2- Apply filters/search repeatedly.
- **Expected Results:** Loads and interactions complete without noticeable lag; list/filters remain responsive.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.29 TC-029

- **Title:** Reliability – Immediate Save & Graceful Invalid Handling
- **Related Requirement ID(s):** NFR- Reliability
- **Preconditions:** App running.
- **Test Steps:**
  1- Add/edit/delete; immediately kill and relaunch app.
  2- Attempt invalid inputs on add/edit.
- **Expected Results:** Most recent changes present after relaunch (immediate save); invalid inputs yield clear feedback; no data corruption.
- **Actual Result:**
- **Status:**

- **Priority:** High

5.5.30 TC-030

- **Title:** Privacy/Offline – No Network Dependency
- **Related Requirement ID(s):** Privacy/Offline
- **Preconditions:** Device offline (disable network).
- **Test Steps:**
  1- Launch app.
  2- Perform add/view/filter/report flows.
- **Expected Results:** All features function offline; no network calls attempted.
- **Actual Result:**
- **Status:**
- **Priority:** High

5.5.31 TC-031

- **Title:** Portability – Run on Windows/macOS/Linux
- **Related Requirement ID(s):** NFR-Portability
- **Preconditions:** Build packaged with supported Java runtime.
- **Test Steps:**
  1- Start app on each OS.
  2- Smoke test add/view flows.
- **Expected Results:** App launches and core flows work on all three platforms.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

5.5.32 TC-032

- **Title:** Maintainability – Modular Structure & Unit Tests Present
- **Related Requirement ID(s):** NFR-Maintainability
- **Preconditions:** Repo accessible.
- **Test Steps:**
  1- Review project structure (e.g., MVC separation.
  2- Verify presence of unit tests for core model/services.

- **Expected Results:** Clear modular separation; baseline unit tests exist; naming/packaging consistent.
- **Actual Result:**
- **Status:**
- **Priority:** Medium

6 Iteration Plan and Backlog

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.

6.1 Iteration Plan

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

- Deliverable 1.1 – Vision Document: Submitted.

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

- Deliverable 1.2 – Requirements, Use Cases, and User Stories: Submitted.

6.1.3 Iteration 3 (Weeks 6–9 –Testing Document)

- Deliverable 2.1 – Testing Document: Current deliverable

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

- Deliverable 2.2 – Full Documentation, Prototype, and Presentation

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

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

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

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

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

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

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

- Deliverable 3.3 – Final Documentation Preparation

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

- Deliverable 3.3 – Final Project Submission

6.2 Updated Backlog

- **User Story 1** – Record a Transaction (Core functionality: enables basic data entry.)
- **User Story 2** – Edit or Delete a Transaction (Ensures data accuracy and integrity.)
- **User Story 3** – View and Filter Transactions (Viewing is essential for MVP; filtering can be expanded later.)
- **User Story 8** – Load Saved Data on Startup (Allows continuity between sessions.)
- **User Story 6** – View Monthly Summary (Provides immediate feedback on totals and balances.)
- **User Story 4** – Set a Monthly Budget (Introduces budget tracking.)
- **User Story 5** – Receive Overspending Alerts (Enhances awareness and control over spending.)
- **User Story 7** – View Spending Chart (Adds visualization for deeper insights.)

7 Conclusion

This document summarizes the testing preparation for the SmartBudget – Personal Finance Tracker project. It presents the test plan, including the requirement-to-test case traceability matrix and detailed test case descriptions, to ensure that all functional and non-functional requirements are properly documented and ready for validation. The testing

documentation provides flexibility for future updates as new features are added or modified, ensuring full traceability between requirements, user stories, and test cases. In addition, minor updates have been made to refine the user stories and use cases based on the evolving project scope, keeping the documentation aligned with the planned functionality.

The next steps for the team involve moving from documentation to implementation, focusing on building the initial prototype of the **SmartBudget** system. Once the prototype is developed, the prepared test cases will be used for actual execution and coverage analysis to confirm system reliability and accuracy. The following iteration will also include the full project documentation and an initial presentation, marking the transition from the testing design phase to practical validation and demonstration of the system's core features.

GitHub Repository Link:

https://github.com/Hercules-Hawk/smart-budget-personal-finance-tracker