SETA - Testing Document

SETA Smart Expense Tracker Application

Document Version:

Version 4.0

Revision: Initial Release based on Course Project Specification

Printing Date:

May 3, 2025

Group Information:

Group ID: D2

Members:

Tang Bok Hei (1155193297)
Siu Wing Yiu (1155192075)
Leung Ho Chun (1155193014)
So Kin Pui (1155193506)

Department:

Department of Computer Science and Engineering

Contents

1 Introduction **2**

2 Testing Strategy **2**

2.1 Testing Levels 2

2.2 Testing Types 2

2.3 Tools and Environment 2

3 Test Plan **3**

3.1 Scope and Objectives 3

3.2 Test Execution and Tracking 3

3.3 Test Data Preparation 4

3.4 Covered Components/Features 4

3.5 Uncovered / Lightly Tested Components 5

4 Representative Test Cases **5**

5 Test Summary and Results **11**

6 Conclusion **12**

1 Introduction

This document details the testing strategy, plan, and representative test cases executed for the Smart Expense Tracker Application (SETA) project (Group D2). Its purpose is to verify that the application meets the functional and non-functional requirements outlined in the Project Requirements Specification (SRS) and the Design and Implementation document, ensuring software quality and readiness. This document serves as a guide for current and future developers regarding the testing process.

2 Testing Strategy

A multi-layered testing strategy was employed to ensure comprehensive coverage and identify defects early in the development cycle.

2.1 Testing Levels

- **Unit Testing:** Focused on verifying individual functions and components, particularly in the back-end API logic (e.g., helper functions, model interactions). Performed primarily during development.
- **Integration Testing:** Focused on testing the interaction between components, primarily between the React frontend and the FastAPI backend API endpoints using tools like Axios within the frontend and potentially Postman during development. Ensured data flowed correctly and API contracts were met.
- **System Testing:** End-to-end testing performed via the Electron application interface, simulating real user workflows. This was the primary focus to validate functional requirements from a user's perspective.
- **User Acceptance Testing (UAT):** Manual testing performed by team members acting as end-users to ensure the application is intuitive, meets requirements, and is free of critical usability issues.

2.2 Testing Types

- **Functional Testing:** Validated that each feature behaves as specified in the requirements (e.g., adding expenses, generating reports, user authentication). This constituted the majority of the testing effort.
- **Usability Testing:** Assessed the ease of use, clarity of the UI, and overall user experience through manual exploration and feedback sessions.
- **Compatibility Testing:** Ensured the packaged application runs correctly on the target operating systems (Windows, macOS, Linux - specify which were tested).
- **Performance Testing (Basic):** Included informal checks for UI responsiveness and data loading times under typical usage scenarios, particularly with data generated by test scripts. No formal load testing was performed.
- **Security Testing (Basic):** Focused on verifying authentication/authorization mechanisms (password hashing, email verification, session management, ensuring users can only access their own data via API calls). No penetration testing was performed.

2.3 Tools and Environment

- **Manual Testing:** Primary method for System and UAT testing, using checklists derived from requirements.

- **Test Data Generation Scripts:** Python scripts (`generate_test_data.py`, `parallel_generate_test_`
were used to populate the database for the 'test' user with a large volume of sample data (expenses, income, accounts, etc.) to enable more realistic testing of features like pagination, sorting, filtering, and dashboard performance.
- **Browser Developer Tools:** Used for inspecting frontend state, network requests, and console logs during debugging.
- **Backend API Tools (Development):** Tools like Postman or direct HTTP requests were used during development to test API endpoints independently.
- **Version Control:** GitHub was used for code management and issue tracking, facilitating collaborative testing and bug fixing.
- **Test Environment:** Testing was primarily conducted on [Specify OS tested, e.g., Windows 11, macOS Sonoma, Ubuntu 22.04] using the packaged Electron application and the default SQLite database configuration, often populated with data from the generation scripts.

3 Test Plan

This plan outlines the scope, objectives, and execution details for testing SETA, fulfilling the requirements of PRS section 7.1.3.

3.1 Scope and Objectives

- **Scope:** To test the core functional requirements of the SETA application as defined in the SRS, including user authentication, data management (expenses, income, accounts, recurring, planning), data import/export, reporting, licence management, and the dynamic dashboard functionality within the Electron desktop environment, using both manually entered and script-generated test data.
- **Objectives:**
  - Verify that all specified functional requirements are implemented correctly.
  - Identify and document defects, inconsistencies, or usability issues.
  - Ensure data integrity is maintained during CRUD operations and import/export.
  - Validate basic security mechanisms (authentication, authorization).
  - Confirm the application runs stably on target platforms with varying amounts of data.
  - Provide confidence in the application's readiness for its intended use.

3.2 Test Execution and Tracking

- **Testers:** Testing was performed by all members of Group D2.
- **Execution:** Primarily manual system testing based on test cases derived from requirements, executed against both clean and pre-populated (via script) database states. Exploratory testing was also conducted.
- **Defect Tracking:** Issues and bugs identified during testing were logged and tracked using GitHub Issues.
- **Status Reporting:** Test progress and results were discussed during team meetings.

3.3 Test Data Preparation

To ensure testing reflects realistic usage scenarios with non-trivial data volumes, Python scripts were developed:

- **Scripts:** `generate_test_data.py` and `parallel_generate_test_data.py`.
- **Purpose:** These scripts interact with the running backend API to automatically create a large number of sample records (accounts, income, expenses, recurring rules, budgets, goals) for a pre-defined test user (`test`). The parallel version uses threading to speed up the generation process.
- **Benefits:** Allowed for effective testing of features like list pagination, sorting, dashboard widget calculations (averages, trends, breakdowns), filtering performance, and reporting with representative data quantities.
- **Usage:** The scripts were run as needed during the testing phase to populate the SQLite database connected to the backend. A summary file (`test_user_data_summary.json`) confirms the number of records generated.
- **Impact:** Many of the system and functional tests were executed against this pre-populated dataset to validate behavior beyond simple, manually entered data.

3.4 Covered Components/Features

The following core features and functional requirements (FRs) were tested, utilizing both manual input and script-generated data where applicable:

- FR-AUTH: User Signup, Email Verification, Login, Logout, Password Reset flow.
- FR-USER: Profile Viewing/Editing, In-App Password Change, Licence Status Viewing/Updating, JSON Data Export/Import.
- FR-EXP: Adding (incl. custom category), Viewing, Sorting, Paginating, Deleting (Single/Bulk) Expenses.
- FR-INC: Adding, Viewing, Sorting, Paginating, Deleting (Single/Bulk) Income.
- FR-ACC: Adding, Viewing, Deleting Accounts (including conflict check for linked items).
- FR-REC: Adding, Viewing, Deleting Recurring Transaction Rules.
- FR-PLAN: Adding, Viewing, Deleting Budgets and Goals (including progress visualization).
- FR-DASH: Adding/Removing Widgets, Rearranging/Resizing Layout, Time Period Selection, Filtering (Category/Amount). Persistence of layout/filters. Functionality and responsiveness of widgets with generated data.
- FR-IMPORT: CSV Import for Expenses and Income (validation, error handling).
- FR-REPORT: Standard Report generation (Excel, PDF, CSVs) with generated data. Custom Report generation based on selections.
- FR-LICENCE: Licence key validation, UI indication/restriction for licensed features.
- UI Requirements: Basic responsiveness, feedback messages, theme/language switching.
- Database Requirements: Data persistence, schema creation (SQLite).

3.5 Uncovered / Lightly Tested Components

Despite the use of generated data, the following areas still have limitations similar to those previously identified:

- **Formal Performance/Load Testing:** While tested with hundreds of generated records, formal stress testing with significantly larger datasets (e.g., 100k+ records) was not performed.
- **Advanced Security Testing:** No formal penetration testing performed.
- **Cross-Database Compatibility (PostgreSQL):** Testing focused on SQLite.
- **Complex Edge Cases:** Focus remained on core functionality validation.
- **Accessibility Testing:** Not formally tested.
- **Automated UI Testing:** Not implemented.
- **Long-Term Stability:** Not assessed over extended periods.

4 Representative Test Cases

The following sections detail representative test cases executed to verify key functionalities. Each test case is presented within a styled box.

TC-AUTH-01: User Signup

Steps:	1. Navigate to Signup page. 2. Enter valid, unique user details (NOT ’test’). 3. Submit form.
Preconditions:	New user, valid email address accessible.
Expected:	Account created, success message shown, verification email sent.
Actual:	Pass
Rationale:	Verify core registration flow (FR-AUTH).

TC-AUTH-02: Email Verification

Steps:	1. Click verification link from email (for user from TC-AUTH-01).
Preconditions:	User registered but not verified (TC-AUTH-01). Link is valid.
Expected:	User redirected to app/confirmation page. Account activated. User can now log in.
Actual:	Pass
Rationale:	Validate email verification mechanism (FR-AUTH).

TC-AUTH-03: User Login (Success - Test User)

Steps:	1. Navigate to Login page. 2. Enter credentials for ’test’ user. 3. Submit.
Preconditions:	’test’ user exists, verified, active. DB potentially populated by script.

Expected:	User logged in, redirected to dashboard. Session persisted (localStorage).
Actual:	Pass
Rationale:	Verify login for pre-defined/script-populated user (FR-AUTH).

TC-AUTH-04: User Login (Fail - Invalid Pwd)

Steps:	1. Navigate to Login page. 2. Enter 'test' username, incorrect password. 3. Submit.
Preconditions:	'test' user exists.
Expected:	Error message shown ("Invalid username or password"). User remains on login page.
Actual:	Pass
Rationale:	Test incorrect password handling (FR-AUTH).

TC-AUTH-05: User Login (Fail - Unverified)

Steps:	1. Register new user (e.g., 'unverified_user'). 2. DO NOT verify email. 3. Try to log in as 'unverified_user'.
Preconditions:	User exists but not verified.
Expected:	Error message shown ("Email not verified..."). Login prevented.
Actual:	Pass
Rationale:	Test login restriction for unverified accounts (FR-AUTH).

TC-USER-01: Update Profile ('test' user)

Steps:	1. Log in as 'test'. 2. Go to Settings -> Profile. 3. Change First Name. 4. Save. 5. Refresh/Re-login.
Preconditions:	'test' user logged in.
Expected:	Profile updated successfully. Changes persist after refresh/re-login.
Actual:	Pass
Rationale:	Verify profile update functionality (FR-USER).

TC-USER-02: Change Password (In-App 'test' user)

Steps:	1. Log in as 'test'. 2. Go to Settings -> Change Password. 3. Enter correct current pwd, valid new pwd (matching). 4. Submit. 5. Log out, log in with new pwd.

Preconditions:	’test’ user logged in. Knows current password.
Expected:	Password changed successfully. User can log in with new password.
Actual:	Pass
Rationale:	Verify in-app password change (FR-USER).

TC-EXP-01: Add Expense

Steps:	1. Log in as ’test’. 2. Go to Expenses. 3. Add valid expense. 4. Submit.
Preconditions:	’test’ user logged in.
Expected:	Expense added to list (potentially long list). Success notification. Total expenses summary updates.
Actual:	Pass
Rationale:	Verify core expense creation (FR-EXP).

TC-EXP-02: Expense List Pagination/Sort

Steps:	1. Log in as ’test’. 2. Go to Expenses. 3. Verify pagination controls are present and functional. 4. Click column headers (Date, Amount, Category) to sort.
Preconditions:	’test’ user logged in. DB populated with many expenses via script.
Expected:	List displays correctly paginated. Sorting works as expected for each column.
Actual:	Pass
Rationale:	Test list handling with volume (FR-EXP).

TC-EXP-03: Bulk Delete Expenses

Steps:	1. Log in as ’test’. 2. Go to Expenses. 3. Select multiple expenses (e.g., across pages). 4. Click "Delete Selected". 6. Confirm.
Preconditions:	’test’ user logged in. Multiple expenses exist.
Expected:	Selected expenses removed. List updates correctly (pagination may change). Success notification.
Actual:	Pass
Rationale:	Validate bulk delete with volume (FR-EXP).

TC-INC-01: Add Income

Steps:	1. Log in as ’test’. 2. Go to Income. 3. Add valid income record. 4. Submit.
Preconditions:	’test’ user logged in.
Expected:	Income record added successfully.
Actual:	Pass
Rationale:	Verify core income creation (FR-INC).

TC-ACC-01: Add Account

Steps:	1. Log in as ’test’. 2. Go to Accounts. 3. Add valid account. 4. Submit.
Preconditions:	’test’ user logged in.
Expected:	Account added successfully.
Actual:	Pass
Rationale:	Verify account creation (FR-ACC).

TC-ACC-02: Delete Account (Conflict)

Steps:	1. Log in as ’test’. 2. Ensure an account exists linked to script-generated income/expenses. 3. Go to Accounts. 4. Try to delete the linked account.
Preconditions:	’test’ user logged in. Linked account exists.
Expected:	Deletion fails. Error message displayed ("Cannot delete account... linked to existing...").
Actual:	Pass
Rationale:	Test dependency check (FR-ACC).

TC-PLAN-01: Add Budget

Steps:	1. Log in as ’test’. 2. Go to Planning -> Budgets. 3. Add valid budget rule. 4. Submit.
Preconditions:	’test’ user logged in.
Expected:	Budget rule added successfully.
Actual:	Pass
Rationale:	Verify budget creation (FR-PLAN).

TC-PLAN-02: Add Goal

Steps:	1. Log in as 'test'. 2. Go to Planning -> Goals. 3. Add valid goal. 4. Submit.
Preconditions:	'test' user logged in.
Expected:	Goal added successfully. Progress bar visible.
Actual:	Pass
Rationale:	Verify goal creation (FR-PLAN).

TC-IMP-01: CSV Expense Import (Success)

Steps:	1. Log in as 'test'. 2. Go to Import Data. 3. Select valid expense CSV. 4. Upload.
Preconditions:	'test' user logged in. Valid CSV.
Expected:	Success message. Imported count correct. New expenses visible in Expense Manager.
Actual:	Pass
Rationale:	Test successful CSV import (FR-IMPORT).

TC-IMP-02: CSV Expense Import (Error)

Steps:	1. Log in as 'test'. 2. Go to Import Data. 3. Select invalid expense CSV. 4. Upload.
Preconditions:	'test' user logged in. Invalid CSV.
Expected:	Error message, skipped rows indicated. Invalid rows not imported.
Actual:	Pass
Rationale:	Test CSV import error handling (FR-IMPORT).

TC-IO-01: JSON Export

Steps:	1. Log in as 'test'. 2. Ensure DB populated by script. 3. Go to Settings -> Data. 4. Click Export.
Preconditions:	'test' user logged in. Data exists.
Expected:	JSON file download starts, contains expected data structures.
Actual:	Pass
Rationale:	Verify export with volume (FR-USER).

TC-IO-02: JSON Import

Steps:	1. Log in as ’test’. 2. Export data. 3. Delete some data. 4. Go to Settings -> Data. 5. Import the exported file, confirm warning.
Preconditions:	’test’ user logged in. Valid JSON backup.
Expected:	Existing data replaced by backup. Success message. Data restored correctly.
Actual:	Pass
Rationale:	Verify import/restore (FR-USER).

TC-DASH-01: Add/Remove Widget

Steps:	1. Log in as ’test’. 2. Go to Dashboard. 3. Add/Remove widgets via Manage dialog.
Preconditions:	’test’ user logged in.
Expected:	Widgets appear/disappear. Layout persists after refresh.
Actual:	Pass
Rationale:	Test dashboard customization (FR-DASH).

TC-DASH-02: Time Period Filter

Steps:	1. Log in as ’test’. 2. Go to Dashboard. 3. Ensure widgets like Trends/Summaries are present. 4. Change time period (Last 7 Days, Current Month, All Time).
Preconditions:	’test’ user logged in. DB populated. Relevant widgets added.
Expected:	Dashboard widgets update dynamically to reflect data ONLY from the selected period. Calculations (e.g., totals, averages) are correct for the period.
Actual:	Pass
Rationale:	Verify dashboard time filtering with data (FR-DASH).

TC-DASH-03: Category/Amount Filter

Steps:	1. Log in as ’test’. 2. Go to Dashboard. 3. Add Filter Widget and Category Breakdown/Trend widgets. 4. Select specific categories in Filter Widget. 5. Adjust amount slider.
Preconditions:	’test’ user logged in. DB populated. Relevant widgets added.
Expected:	Other dashboard widgets (Breakdown, Trend, Summary) update to show data ONLY matching the selected categories and amount range.
Actual:	Pass

Rationale:	Verify dashboard content filtering (FR-DASH).

TC-LIC-01: Update Licence (Valid)

Steps:	1. Log in as ’test’. 2. Go to Settings -> Licence. 3. Enter valid key. 4. Update.
Preconditions:	’test’ user logged in.
Expected:	Success message. Status "Active". Custom Reports enabled in sidebar.
Actual:	Pass
Rationale:	Test valid licence activation (FR-LICENCE, FR-USER).

TC-LIC-02: Update Licence (Invalid)

Steps:	1. Log in as ’test’. 2. Go to Settings -> Licence. 3. Enter invalid key. 4. Update.
Preconditions:	’test’ user logged in.
Expected:	Error message. Status unchanged or "Inactive".
Actual:	Pass
Rationale:	Test invalid licence key (FR-LICENCE, FR-USER).

TC-LIC-03: Custom Report Access

Steps:	1. Log in with NO active licence. 2. Check Custom Reports in sidebar. 3. Activate licence. 4. Check Custom Reports again.
Preconditions:	’test’ user logged in.
Expected:	Access denied/locked initially. Access granted/unlocked after activation.
Actual:	Pass
Rationale:	Verify licence feature restriction (FR-LICENCE).

5 Test Summary and Results

Overall, the core functionalities of the SETA application were tested against the specified requirements using both manually entered data and data generated via the test scripts. The majority of the representative test cases passed, indicating that the primary features such as authentication, data management (CRUD, lists, pagination, sorting), import/export, reporting, licence management, and dashboard functionality (widget management, filtering) are functioning as expected within the tested environments and with representative data volumes.

[Optional: Add more details here, e.g., specific pass/fail counts, summary of critical bugs found and fixed related to data volume or filtering if any.]

Defects identified during testing were logged in GitHub Issues and prioritized for resolution. The use of generated data was particularly helpful in validating list handling, dashboard calculations, and filtering responsiveness.

6 Conclusion

The testing activities performed, including those utilizing script-generated data, provide reasonable confidence in the functionality and stability of the SETA application for its core use cases. Manual system testing and UAT confirmed that the application generally meets the user requirements outlined in the SRS, even when handling a moderate volume of data.

However, limitations previously noted still exist (formal performance/load testing, advanced security, PostgreSQL compatibility, accessibility). These remain areas for future testing or enhancement. Based on the testing conducted with both manual and generated data, the current version is deemed suitable for its intended purpose.