Project Requirements Specification

Smart Expense Tracker Application (SETA)

Document Version:
Version 6.0
Revision: Initial Draft 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

1.1 Purpose

This document specifies the functional and non-functional requirements for the Smart Expense Tracker Application (SETA). It serves as the primary reference for project development, design, testing, and validation, ensuring alignment between the development team and project objectives.

1.2 Scope

SETA is a desktop application designed for personal finance management. It enables users to track income and expenses, manage financial accounts, set budgets and goals, generate reports, and visualize financial data through a customizable dashboard. The system includes user authentication, data persistence, and basic licence management. The application consists of a frontend user interface built with React (packaged with Electron) and a backend API built with Python (FastAPI).

1.3 Definitions, Acronyms, and Abbreviations

- **SETA:** Smart Expense Tracker Application
- **UI:** User Interface
- **API:** Application Programming Interface
- **CRUD:** Create, Read, Update, Delete (basic data operations)
- **CSV:** Comma Separated Values
- **JSON:** JavaScript Object Notation
- **MUI:** Material-UI (React component library)
- **SQLite:** Relational database management system (default local DB)
- **PostgreSQL:** Relational database management system (optional cloud/custom DB)
- **JWT:** JSON Web Token (Implicitly used for authentication session, though not explicitly detailed in provided context)
- **OS:** Operating System
- **HOC:** Higher-Order Component (React pattern)
- **SRS:** Software Requirements Specification

2 Overall Description

2.1 Project Overview

SETA aims to provide users with a comprehensive tool for monitoring and managing their personal finances. By consolidating expense tracking, income logging, account management, and planning features into a single desktop application, SETA empowers users to gain insights into their spending habits, track progress towards financial goals, and make more informed financial decisions. The application emphasizes a clear user interface and data visualization through a dynamic dashboard.

2.2 Assumptions and Dependencies

- Users possess basic computer literacy and familiarity with desktop applications.
- The target operating systems (Windows, macOS, Linux) are relatively recent (within 5 years as per project spec) and function correctly.
- Users have sufficient permissions to install and run the application and store data in their user directory.
- The default local database (SQLite) requires no external setup. Use of cloud or custom databases requires correct configuration and network access.

- Email verification and password reset features require a working internet connection and valid user email addresses.
- The application relies on external libraries (e.g., React, MUI, FastAPI, SQLAlchemy, date-fns, Recharts, react-grid-layout, libraries for PDF/Excel generation) which must be correctly installed.
- Licence validation depends on the correct implementation and availability of the backend validation logic against a predefined list.

3 Specific Requirements

This section details the specific functional and non-functional requirements for SETA.

3.1 Functional Requirements

Functional requirements define the specific behaviors and features of the system.

3.1.1 FR-AUTH: Authentication

- The system shall allow new users to sign up by providing a username, first name, last name, email, contact number, and password.
- The system shall validate user input during signup (e.g., required fields, email format, password complexity, password match, username constraints).
- The system shall prevent registration with duplicate usernames or emails.
- The system shall send a verification email containing a unique activation link upon successful signup.
- The system shall verify a user's email address and activate their account when the user clicks the activation link.
- The system shall allow registered and verified users to log in using their username and password.
- The system shall validate login credentials against stored user data.
- The system shall prevent login for inactive or unverified accounts.
- The system shall persist user login session (e.g., using `localStorage` to store user ID, username).
- The system shall provide a mechanism for users to log out, clearing the session state.
- The system shall allow users to request a password reset via their registered email address (triggered from Settings).
- The system shall allow users to set a new password by accessing the reset button in setting pages and providing/confirming the new password.
- The system shall protect application routes, ensuring only authenticated users can access non-public modules.
- The system shall redirect unauthenticated users attempting to access protected routes to the login page.
- The system shall redirect authenticated users attempting to access login or signup pages to the main dashboard.

3.1.2 FR-USER: User Profile and Settings

- The system shall allow authenticated users to view their profile information (first name, last name, username, email, contact number).
- The system shall allow authenticated users to update their profile information.
- The system shall allow authenticated users to change their password within the application by providing their current password and a new password (with confirmation).
- The system shall validate password change input (current password match, new password match, complexity).
- The system shall allow users to view their current application licence status (Active, Inactive, Not Set) and a masked key prefix.
- The system shall allow users to input and submit a new licence key.
- The system shall validate the submitted licence key format and check its validity against a predefined list on the backend.
- The system shall update and display the new licence status upon successful validation.
- The system shall provide functionality to export all user data (expenses, income, accounts, recurring, budgets, goals) into a single JSON backup file.
- The system shall provide functionality to import user data from a previously exported JSON backup file, replacing all existing data for the user after explicit confirmation.

3.1.3 FR-EXP: Expense Management

- The system shall allow users to add new expense records, including amount, date, category, and an optional description.
- The system shall provide a predefined list of expense categories (e.g., Food, Transportation, Housing) for selection.
- The system shall allow users to specify a custom category name when selecting an "Others (Specify)" option.
- The system shall display a list of all user expenses, including date, category, amount, and description.
- The system shall support sorting and pagination for the expense list.
- The system shall allow users to delete individual expense records after confirmation.
- The system shall allow users to select multiple expense records from the list.
- The system shall allow users to delete selected expense records in bulk after confirmation.
- The system shall display summary information, including the total amount of expenses and the total number of expense entries shown in the current view/filter.

3.1.4 FR-INC: Income Management

- The system shall allow users to add new income records, including amount, date, source, optional description, and an optional linked account.
- The system shall display a list of all user income records, including date, source, amount, description, and linked account name (if applicable).
- The system shall support sorting and pagination for the income list.
- The system shall allow users to delete individual income records after confirmation.
- The system shall allow users to select multiple income records from the list.
- The system shall allow users to delete selected income records in bulk after confirmation.
- The system shall display summary information for income (total amount, total entries).

3.1.5 FR-ACC: Account Management

- The system shall allow users to define and manage financial accounts (e.g., Checking, Savings, Credit Card, Cash).
- The system shall allow users to add a new account by specifying a name, account type, starting balance, and the date the starting balance was accurate ("Balance As Of Date").
- The system shall display a list of all user-defined accounts, showing name, type, starting balance, and balance date.
- The system shall allow users to delete individual accounts after confirmation.
- The system shall allow users to select multiple accounts from the list.
- The system shall allow users to delete selected accounts in bulk after confirmation.
- The system shall prevent the deletion of an account if it is currently linked to any existing expense, income, or recurring transaction records. The system shall display an error if deletion is blocked due to dependencies.

3.1.6 FR-REC: Recurring Transactions

- The system shall allow users to define recurring transaction rules (typically expenses, but could be income).
- The system shall allow users to add a new recurring rule specifying a name, amount, category, frequency (e.g., Daily, Weekly, Monthly, Quarterly, Yearly), start date, optional end date, optional description, and optional linked account.
- The system shall display a list of all defined recurring rules, showing relevant details.
- The system shall allow users to delete individual recurring rules after confirmation.
- The system shall allow users to select multiple recurring rules.
- The system shall allow users to delete selected recurring rules in bulk after confirmation.

3.1.7 FR-PLAN: Planning (Budgets and Goals)

- The system shall provide a dedicated section for managing budgets and goals, likely using a tabbed interface.
- **Budgets:**
  - The system shall allow users to define budget rules for specific expense categories.
  - The system shall allow users to set an amount limit, a period (Monthly, Quarterly, Yearly), a start date, and an optional end date for each budget rule.
  - The system shall display a list of all defined budget rules.
  - The system shall allow users to delete individual budget rules after confirmation.
  - The system shall allow users to select multiple budget rules.
  - The system shall allow users to delete selected budget rules in bulk after confirmation.
- **Goals:**
  - The system shall allow users to define financial savings goals.
  - The system shall allow users to set a name, target amount, current amount saved (defaulting to 0), and an optional target date for each goal.
  - The system shall display a list of all defined financial goals.
  - The system shall visualize the progress towards each goal (e.g., using a progress bar).
  - The system shall allow users to delete individual goals after confirmation.
  - The system shall allow users to select multiple goals.
  - The system shall allow users to delete selected goals in bulk after confirmation.

3.1.8 FR-DASH: Dynamic Dashboard

- The system shall provide a customizable dashboard displaying various financial widgets.
- The system shall allow users to add widgets from a predefined list to the dashboard.
- The system shall allow users to remove widgets from the dashboard.
- The system shall allow users to rearrange and resize widgets on a grid layout.
- The system shall persist the user's dashboard layout (widget selection, positions, sizes) across sessions (e.g., using `localStorage`).
- The system shall provide a mechanism (widget or global control) for users to select a time period (e.g., Last 7 Days, Current Month, Custom Range) for the data displayed on the dashboard.
- The system shall persist the selected time period across sessions.
- The system shall provide a filtering mechanism (widget) allowing users to filter dashboard data by expense category/income source and transaction amount range.
- The system shall persist the selected filters across sessions.
- The system shall provide a variety of widgets, including but not limited to:

  - Overview Summary (Total Income, Expenses, Net Flow)
  - Account Balances / Details
  - Expense/Income Category Breakdown (Pie Chart)
  - Expense/Income Trend (Line Chart)
  - Recent Transactions List
  - Budget Overview / Comparison
  - Goal Progress / Estimate
  - Top Spending Categories / Income Sources
  - Upcoming Bills (from Recurring Items)
  - Average Daily Spend/Income/Net Flow
  - Quick Add Transaction Form
- The system shall ensure widgets dynamically update based on the selected time period and active filters.

3.1.9 FR-IMPORT: Data Import

- The system shall provide a dedicated interface for importing expense data from a CSV file.
- The system shall provide a dedicated interface for importing income data from a CSV file.
- The system shall validate the uploaded CSV file format (e.g., check required headers: `date`, `amount`, `category_name` for expenses; `date`, `amount`, `source` for income).
- The system shall process the CSV file, creating new expense/income records for valid rows.
- The system shall provide feedback to the user upon import completion, indicating the number of records successfully imported, the number of rows skipped due to errors, and details of any errors encountered.

3.1.10 FR-REPORT: Reporting

- **Standard Report:**
  - The system shall provide a feature to generate a standard, comprehensive report containing all user data (expenses, income, accounts, recurring, budgets, goals).
  - The system shall offer the standard report download in multiple formats: Excel (.xlsx), PDF (.pdf), and individual CSV files per data type.
  - The system shall format the data appropriately within the generated report files (e.g., separate sheets/tables per data type, formatted dates and currency).
- **Custom Report (Licence Required):**
  - The system shall provide an interface for generating custom reports, accessible only to users with an active licence key.
  - The system shall allow users to select specific data types to include in the custom report.
  - The system shall allow users to filter the data for the custom report based on a selected time period.
  - The system shall allow users to choose the output format (CSV, Excel, PDF) for the custom report.
  - The system shall generate and provide the custom report file for download based on user selections.

3.1.11 FR-LICENCE: Licence Management Integration

- The system shall integrate licence status checks within the application.
- The system shall restrict access to specific features (e.g., Custom Reports) based on the user's licence status (must be "Active").
- The system shall visually indicate in the UI (e.g., sidebar) which features require an active licence, possibly disabling them or showing a lock icon if the licence is not active.

3.2 User Interface (UI) Requirements

- The system shall provide a clear, consistent, and intuitive graphical user interface (GUI) based on the Material-UI design system.
- The system shall utilize a persistent sidebar for primary navigation between application modules for authenticated users.
- The system shall ensure the layout is responsive and functions correctly within the Electron desktop application window.
- The system shall provide clear visual feedback for user actions, loading states (e.g., spinners), and success/error messages (e.g., snackbars, alerts).
- The system shall support user selection of application themes (e.g., Light, Dark, System Default) and persist the choice.
- The system shall support user selection of application language (e.g., English, Chinese) and persist the choice, applying translations throughout the UI.

3.3 Database Requirements

- The system shall utilize a single, global database instance to store all application data for a given user session.
- The system shall default to using a local SQLite database file stored within a dedicated user data directory (`seta_user_data`).
- The system shall automatically create the necessary database schema (tables, columns) if the local SQLite file does not exist on first run.
- The system shall ensure data persistence across application sessions.

3.4 Non-Functional Requirements

3.4.1 NFR-PERF: Performance

- The system shall ensure the UI remains responsive during typical user interactions.
- The system shall ensure data loading times for lists and dashboards are acceptable (e.g., within a few seconds for moderate data volumes).
- The system shall ensure report generation times are reasonable for the amount of data being processed.

3.4.2 NFR-REL: Reliability

- The system shall maintain data integrity during CRUD operations.
- The system shall provide data export/import functionality to mitigate data loss risk.
- The system shall handle potential API errors gracefully, providing informative feedback to the user.

3.4.3 NFR-USAB: Usability

- The system shall provide an intuitive workflow for common tasks (adding expenses/income, viewing data).
- The system shall ensure UI elements and terminology are consistent and easy to understand.
- The system shall minimize the steps required to perform core actions.

3.4.4 NFR-SEC: Security

- The system shall store user passwords securely using strong hashing algorithms (e.g., SHA-256). Passwords shall never be stored in plain text.
- The system shall implement email verification to confirm user identity during signup.
- The system shall utilize secure tokens for password reset links with limited expiry times.

3.4.5 NFR-MAINT: Maintainability

- The system shall ensure the codebase is well-documented, following consistent coding standards.
- The system shall structure the code logically into modules (frontend and backend).

3.4.6 NFR-COMP: Compatibility

- The system shall ensure the desktop application runs on recent versions (within 5 years) of Windows, macOS, and Linux operating systems.

4 High-Level Architecture

SETA follows a client-server architecture adapted for a desktop application:

- **Frontend (Client):** A React single-page application responsible for the user interface and user interactions. It is packaged within an Electron container to create a cross-platform desktop application. It communicates with the backend via HTTP requests to the API. Material-UI is used for UI components.
- **Backend (Server):** A Python application built using the FastAPI framework. It exposes a RESTful API for all data operations (CRUD, import/export, reporting, authentication, settings). It handles business logic, data validation, and interaction with the database using SQLAlchemy ORM.
- **Database:** A relational database (defaulting to SQLite locally) used for persistent storage of all user and application data.
- **Communication:** The frontend interacts with the backend exclusively through the defined REST API endpoints.