Software Requirements Specification (SRS)

System Overview

The Personal Budget and Expense Management System is a mobile application designed to provide users with a simple, intuitive, and private tool for managing their personal finances. The system's primary goal is to empower individuals to take control of their financial health by offering clear insight into their income, spending habits, and savings progress without unnecessary complexity.

At its core, the system allows users to record their monthly income and categorize their daily expenses. Using this data, it automatically calculates key financial metrics—such as total income, total expenses, and remaining budget—and presents them in a central dashboard. Beyond tracking, the system assists in forward-looking financial planning by enabling users to set specific savings goals and monitor their progress over time.

Key functionalities include the ability to log financial transactions, view summarized reports and charts, manage savings targets, and maintain a searchable history of all records. The system is designed with a focus on user experience, ensuring that all features are accessible through a clean, uncluttered interface that requires no prior technical or financial expertise.

Developed as a student project prototype, this system operates under specific constraints. It is a self-contained application that does not connect to external financial institutions or services. All user data is stored locally on the device, prioritizing privacy and simplicity over advanced features like bank synchronization. The system assumes that users will input their financial information accurately, as it employs only basic validation mechanisms.

Ultimately, this system serves as a foundational tool for financial awareness, transforming raw transaction data into an organized visual overview. It supports better personal decision-making by answering essential questions:

Q. Where is my money going?
Q.How much can I save?
Q.Am I on track to meet my goals?

By providing these insights in a secure and straightforward manner, the application aims to reduce the stress associated with personal budget management.

System Features

The Personal Budget and Expense Management System provides the following core features to empower users in managing their personal finances effectively:

1. Core Financial Tracking

· Income Recording: Log and categorize regular income sources with date and optional notes
· Expense Management: Record daily expenses with detailed categorization (e.g., Food, Transportation, Entertainment)
· Transaction History: Maintain a complete, timestamped ledger of all financial activities

2. Dashboard & Overview

· Financial Summary: Real-time display of total income, total expenses, and remaining budget
· Visual Analytics: Graphical representation of income vs. expenses using charts and graphs
· Quick Actions: One-tap access to frequently used functions like adding income or expenses

3. Savings & Goal Management

· Goal Setting: Define specific savings targets with amount and target date
· Progress Tracking: Visual indicators showing progress toward savings goals

4. Reporting & Insights

· Monthly Summaries: Automated reports showing expenses by category for each month
· Spending Patterns: Identify trends in spending behavior over time
· Budget Analysis: Compare planned budget against actual spending

5. Data Management

· Transaction Editing: Modify or delete previously recorded transactions
· Data Filtering: View transactions filtered by date range, category, or amount
· Search Functionality: Quickly locate specific transactions using search terms

6. User Experience Features

· Intuitive Interface: Clean, uncluttered design requiring no technical expertise
· Mobile Optimization: Responsive design adapting to various smartphone screen sizes
· Offline Functionality: Complete system operation without internet connection
· Local Data Storage: All financial data stored securely on the user's device

7. Alert & Notification System

· Budget Warnings: Alerts when spending approaches or exceeds budget limits
· Goal Reminders: Notifications about savings goal progress and deadlines
· Income/Expense Confirmations: Feedback for successful transaction recordings

Feature Categorization by Priority

High Priority (MVP Features)
· Income and expense recording
· Basic dashboard with totals
· Savings goal setting
· Transaction history view

Medium Priority
· Advanced filtering and search
· Detailed monthly reports
· Multiple savings goals
· Export functionality

Low Priority (Future Enhancements)
· Recurring transactions setup
· Budget forecasting
· Multi-currency support
· Cloud backup option

Technical Feature Specifications

FEATURE	INPUT REQUIREMENTS	OUTPUT/RESULT	VALIDATION RULES
ADD INCOME	*Amount (required), Date (default: today), Notes (optional)*	*Updated dashboard totals, Added to transaction history*	*Amount > 0, Date not future-dated*
OTHER BOUTIQUE AGENCIES	*Amount, Category, Date, Notes*	*Updated dashboard, Expense categorization, Budget check*	*Amount > 0, Category selection required*
SET SAVINGS GOAL	*Target Amount, Target Date, Goal Name*	*Goal progress tracker, Dashboard widget*	*Amount > 0, Future target date*
VIEW REPORT	*Month/Year selection*	*Categorized expense breakdown, Charts*	*Valid date range*

User Workflow Features

For Daily Use:

1. Quick expense entry with category selection
2. Instant budget balance update
3. Recent transactions glance view

For Weekly Review:

1. Category-wise spending summary
2. Progress toward weekly budget
3. Savings goal status check

For Monthly Planning:

1. Income planning for upcoming month
2. Budget allocation by category
3. Savings goal adjustment based on income

Accessibility Features

- Clear visual hierarchy and contrast
- Logical tab order for navigation
- Readable font sizes and spacing
- Color-blind friendly chart palettes
- Simple language throughout interface

Security & Privacy Features

- Local data storage (no cloud transmission)
- No financial account linking
- No personal identification collection
- Device-based access only
- Optional PIN protection (future enhancement)

Requirements Engineering

This section details the systematic approach undertaken to discover, analyze, document, and validate the requirements for the Personal Budget and Expense Management System. Our process followed established requirements engineering principles to ensure the final specifications were complete, consistent, and aligned with stakeholder needs.

Requirements Elicitation Methods

Elicitation involved actively engaging with potential stakeholders to gather raw needs and expectations. We employed three complementary techniques to gain a comprehensive understanding from multiple perspectives.

1. Stakeholder Interviews

We selected different and varied groups so that the group requirements would be natural and expressed by each of them without prior restrictions. For example, we chose (students, employees, high-spending people). The goal of the interview was to understand the required features and the difficulties in managing their expenses.

2. Scenario Development

After considering how we understand the sequence of tasks, their arrangement, missing requirements, expected errors, and alternatives, we developed a scenario that reflects the daily reality of using the system, such as: adding income, recording expenses, displaying summaries, and tracking savings goals.

Use Case Modeling

To formalize user-system interactions, we identified primary actors and defined core use cases, forming the basis for the UML Use Case Diagram.

· Primary Actor: The User.

· Identified Core Use Cases:

1. Add Monthly Income
2. Record Expense with Category
3. View Financial Dashboard
4. Generate Monthly Expense Summary
5. Set and Manage Savings Goal

· Outcome: This modeling clarified the system's boundaries, distinguishing between user goals (e.g., "manage budget") and system responsibilities (e.g., "calculate remaining budget").

Requirements Analysis and Classification

Following elicitation, the gathered information was analyzed to remove ambiguities, resolve conflicts, and organize it into structured categories.

1. Consolidation and Clarification: Similar needs from different interviewees were merged. Vague wishes (e.g., "make it easy") were translated into specific, actionable requirements (e.g., "complete expense entry in under 3 taps").

2. Categorization: Requirements were classified into:

· Functional Requirements (FRs): Specific behaviors or functions the system must perform (e.g., "The system shall allow the user to delete a recorded expense").

Requirements Analysis and Classification

· Non-Functional Requirements (NFRs): Qualities or constraints the system must exhibit (e.g., "The system shall respond to user inputs within 3 seconds").

· Constraints: Pre-defined limits on the project, such as "The system shall operate as a standalone mobile app without external API integration," as outlined in the project scope and assumptions.

Requirements Prioritization and Negotiation

Given the project's time constraints, requirements were prioritized to ensure the Minimum Viable Product (MVP) would deliver core value.

· Prioritization Framework: We used the MoSCoW Method (Must have, Should have, Could have, Won't have) in collaboration with our stakeholder insights.

· Prioritized List:
  · MUST HAVE (High Priority): Expense tracking, income entry, real-time dashboard display. Without these, the system fails its primary purpose.
  · SHOULD HAVE (Medium Priority): Monthly summaries, basic savings goal tracking. Critical for enhanced usability and core planning functions.
  · COULD HAVE (Low Priority): Advanced reports with custom filters, graphical trends over multiple months. Useful enhancements for future iterations.

· Negotiation: The constraint of being a student prototype (no bank integration) was explicitly agreed upon as a boundary condition, focusing efforts on core logic and user experience.

Requirements Documentation

All agreed-upon requirements were formally documented in this Software Requirements Specification (SRS).

· Standard Template: We adhered to a standard SRS structure to ensure completeness and ease of reference.

· Traceability: Each requirement is uniquely stated and traceable to its origin in elicitation. This document serves as the single source of truth and the primary input for the subsequent System Design Document (SDD) and all UML modeling activities.

· Language: Requirements are written in clear, concise, and testable language, often using "shall" to denote mandatory conditions.

Requirements Validation

To ensure the requirements were correct, complete, and feasible, we performed the following validation activities:

1. Requirements Review

We have reviewed all the requirements and made sure that they are clear and comprehensive.

2. Scenario Evaluation

We applied the previous scenario to ensure that all steps supported the requirements.

3. Testability Check

We verified each requirement by creating a simple test during system implementation. After testing, we modified any requirement that was unclear to the user or not measurable. For this verification, we adopted the methods recommended in the lecture, such as: reviews, building test cases, and using mental prototypes.

Functional Requirements ✻

Functional requirements: explain the actions and tasks that the system should be capable of doing in regard to its users or other systems. They are directly connected with the business purpose of the system:

1. The system shall allow users to create a new account: The system must provide a registration interface where new users can establish their account by providing required credentials and personal information to access the application's features.

2. The system shall validate password length (minimum 8 characters): During account creation or password change, the system must check that the user's chosen password contains at least eight characters to ensure basic security strength and reject passwords that are shorter than this requirement.

3. The system shall verify that password and confirm password match: When users create or change their password, the system must compare the entries in both password fields and prevent account creation or password update if the two entries do not match exactly, ensuring users do not accidentally set an incorrect password.

4. Add Monthly Income: The system should have an option of letting the user key in and record the form of money that he/she earns regularly.

5. Expenses with Classification: It must have a functionality to record separate purchases and classify each purchase by a certain category (such as Groceries or Utilities) to organize them.

6. Display Control Panel (Total Income & Expenses): The primary display should present the summary or dashboard view with the things that have been computed, total income, and the total expenses during a specific period.

7. Show Monthly Expenses Summary: The system should create a report that would contain the expenses grouped by their types within a month.

8. Create a Saving Target: It should enable the user to set a particular financial target that they are intending to reach such as the amount of money they want to save.

9. Keep Records of the Progress to Saving Goal: The system should automatically keep track and show the user the distance between him/her and the target set in requirement 8.

10. Show List of All Recorded Expenses: The system has to show a detailed, frequently searchable or filterable, ledger of all the transactions recorded.

11. Deletion or Modification of Past Expense: It should enable one to edit or delete an expense he/she had recorded in the past to fix errors.

Non- Functional Requirements

Non-functional requirements define conditions under which the functioning of a system should be assessed, but not an actual behavior. They place restrictions upon the design and implementation and are commonly referred to as quality attributes or "ilities". They make the system reliable, usable and efficient:

1. Usability: The system should be user friendly and need no technical skills. The interface and navigation should be user friendly, that is, a non technical user should be able to use the software immediately.

2. Security/Privacy: The system should not compromise the privacy of the user and disclose his data. The application should use security provisions to ensure that sensitive financial information is not accessed or disclosed by unauthorized people.

3. Performance: Response time must be fast and the time to respond to each operation must not take more than 3 seconds. All major operations (such as saving an expense or loading a report) are expected to be completed speedily with a defined time limit that can be measured to guarantee an enjoyable experience.

4. Aesthetics/Clarity: The system interface should be straightforward and easy to navigate and display the information without crowding. The visual side should be well-structured and clear and display information in a logical manner without cluttering or overloading the visual memory.

External Interface Requirements

USER INTERFACES:

- The system shall maintain a consistent layout, colors, fonts, and icon styles.
- ·The interface shall be designed for mobile use and adapt to different smartphone screen sizes.
- The system shall avoid visual clutter and present only essential information .

Signup Screen

- The system shall provide fields for username, email, password, and confirm password.
- The system shall validate password length (min 8 characters).
- The system shall verify password confirmation.

Login Screen

- The system shall provide a login screen containing fields for email and password, along with a "Login" button.
- The system shall verify the password when login and display an error message if it is wrong.

Main Dashboard

The system shall provide a main dashboard that includes:

- A summary of total income.
- A summary of total expenses.
- Remaining budget.
- A simple chart showing income and expenses.
- Quick access buttons for adding income and adding expense.
- A widget showing recent transactions.
- A widget showing the savings goal.

Add Expense Screen

- Users shall be able to enter an expense with amount, category, date, and optional notes.
- The system shall provide basic input feedback if required fields are empty or values are negative .
- The system shall warn users if the expense exceeds the remaining budget.
- Users shall select a category for each expense (e.g., food, shopping).

Add Income Screen

- Users shall enter income amount and optional notes.
- The system shall check that the amount is not empty or negative.
- Users can assign the income to a time period (e.g., monthly, weekly).

Transaction History Screen

- Users can view all transactions, with the ability to filter by date or category.
- Users should be able to edit or delete transactions.

Reports Screen

- Users can view reports by month or category.
- Reports shall include charts and summaries.

Savings Goal Screens

- Users can set a saving goal by defining the amount and target date.
- The system shall provide basic input feedback to ensure values are non-empty .
- users shall be able to delete an existing goal saving goal .

HARDWARE INTERFACES:

As stated in the project constraints and assumptions, the system is designed as a simple student prototype, therefore, the hardware requirements are minimal.

- The system shall operate on mobile devices such as smartphones.
- The system shall support different screen sizes without breaking the layout.
- The system shall require only basic mobile hardware features such as touchscreen interaction and internal storage, No additional hardware capabilities are needed.

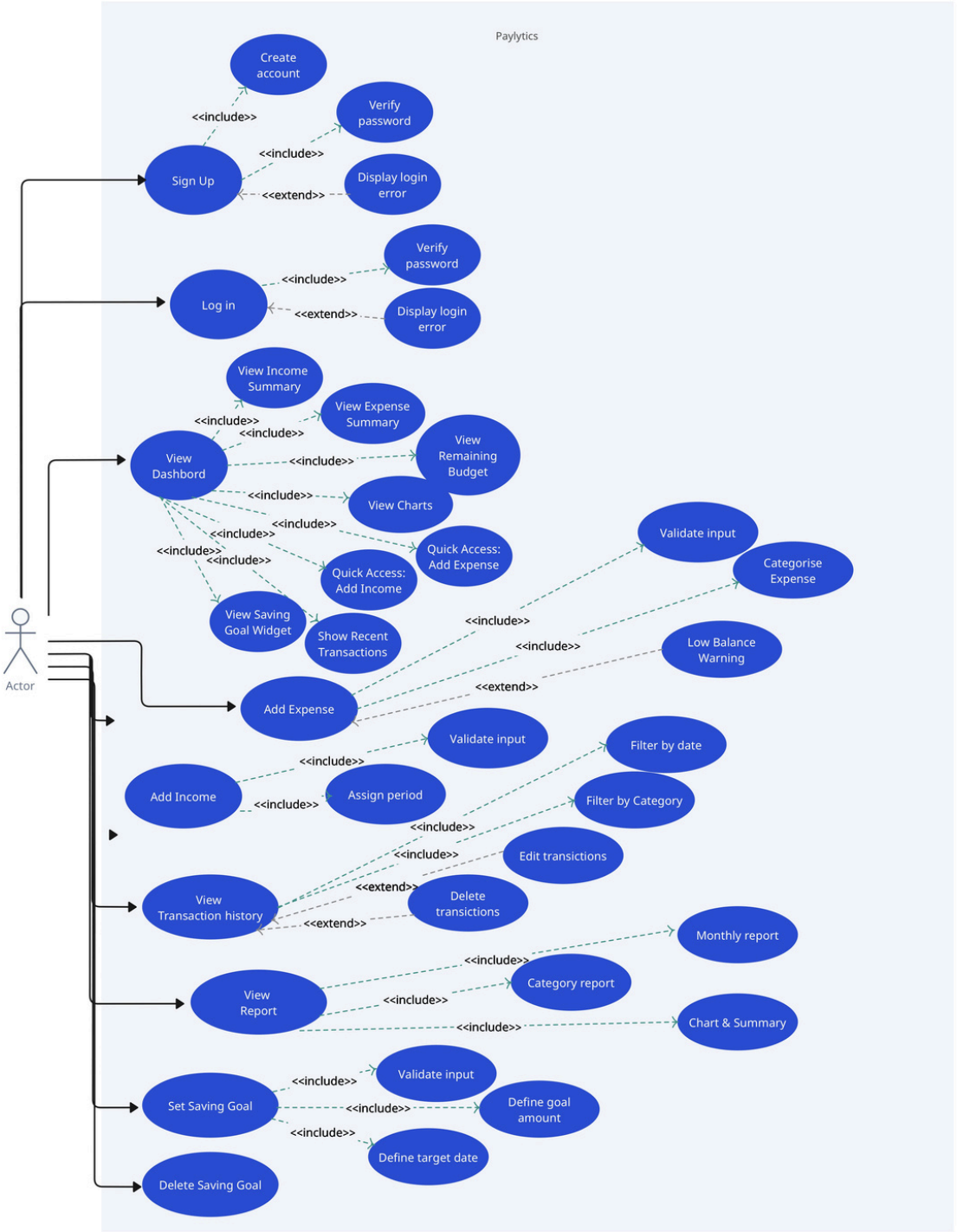
SOFTWARE INTERFACES:

According to the project constraints, the system does not integrate with any external financial services and all user data is stored locally within the application.

- The system shall use local storage to save income, expenses, and budget information.
- The system shall use basic graphical components to display summarise and charts.
- ·The system shall not require connection to external databases or web services.
- ·The system shall support operation on standard mobile operating systems (iOS/Android).

UML:

Use Case Diagram of Expense Calculator System



[Clear diagram]

SE Ethics and Responsibilities:

In line with our project requirements, our team identified six ethical principles from the *ACM/IEEE-CS* Software Engineering Code of Ethics to guide the design and development of our financial management application. These principles ensure that our work remains responsible, transparent, and aligned with professional standards.

Principle 1: PUBLIC

Software engineers should act consistently with the public interest [1]. For our project, this principle is crucial because we are dealing with personal and financial information such as income, expenses, and savings goals. We have an ethical responsibility to protect this data and maintain user trust. We plan to achieve this by using encryption, data validation, and privacy protection measures. Ensuring accuracy in all calculations also supports the public good by preventing misleading or harmful results.

Principle 2: CLIENT AND EMPLOYER

Software engineers should act in the best interests of their clients while also considering the *public interest* . In our case, the client is the user of the app. Their main concern is the confidentiality of their financial data. To protect this, all personal information will remain private and will only be used for the main purpose of budgeting and expense tracking. No external sharing or hidden data collection will occur without clear user consent.

Principle 3: PRODUCT

Software engineers should ensure that their products meet the highest professional standards . Although our project focuses on system analysis and design rather than coding, quality remains a key ethical concern. Any errors in our design, data structure, or process flow could lead to inaccurate financial calculations when implemented. Therefore, we will carefully verify and review all design components, diagrams, and logical processes to ensure accuracy, reliability, and clarity in the system documentation.

Principle 4: JUDGMENT

Software engineers should maintain integrity and independence in their professional judgment . The system provides automated financial recommendations. To maintain fairness, all algorithms and suggestions will be unbiased and focused only on helping users manage their finances effectively. No data manipulation or misleading recommendations will be allowed, ensuring honesty and independence in our work.

Principle 5: MANAGEMENT

Software engineering managers and leaders should promote ethical approaches to software development and maintenance . Even though this is a student project, ethical management still applies. We will ensure that all tasks are divided fairly and that any problems or potential risks are reported immediately. Team communication will remain open and respectful, and every member will take responsibility for their part. These practices strengthen trust and teamwork while maintaining ethical leadership within the group.

Principle 6: PROFESSION

Software engineers should advance the integrity and reputation of the profession consistent with the public interest . Our team aims to reflect professionalism and honesty in every stage of this project. We ensure that all content, designs, and ideas are original and that any external resources or frameworks used are properly cited. Maintaining transparency and academic integrity demonstrates respect for the software engineering profession and contributes to its positive reputation.

Summary

By following these six ethical principles, our team aims to build a secure, accurate, and trustworthy application. Protecting user privacy, ensuring fairness, maintaining transparency, and committing to professional quality all demonstrate that our project meets both the technical and moral standards expected in software engineering [1], [2].

References

[1] ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices, "Software Engineering Code of Ethics and Professional Practice," Version 5.2, 1999. [Online]. Available: https://www.acm.org/code-of-ethics/software-engineering-code.

[2] D. Gotterbarn, K. Miller, and S. Rogerson, "Software Engineering Code of Ethics and Professional Practice," IEEE Computer, vol. 32, no. 12, pp. 88--90, 1999.