Paylytics

SOFTWARE ENGINEERING (CS383)

Team members*

Bana Altuwayjiri
441203227

Deem AlBassm
431200957

Reema alrsheed
441203493

Renad Almohsen
441203253

Yasmin Altwaijiry
441212947

Nourah ALAwadh
441203248

Ramah Alhamdan
441203623

Haydab Alkhalifah
441203272

Joudi abazai
432205391

Fajr Alghwafeli
441203515

Table of Contents

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

INTRODUCTION 6

SYSTEM OVERVIEW 9

SYSTEM FEATURES 11

REQUIREMENTS ENGINEERING 16

FUNCTIONAL REQUIREMENTS 20

NON-FUNCTIONAL REQUIREMENTS 22

EXTERNAL INTERFACE REQUIREMENTS 23

UML - USE CASE DIAGRAM 26

ETHICS AND RESPONSIBILITIES 27

Table of Contents

SOFTWARE ARCHITECTURE DESIGN (SDD)

SOFTWARE ARCHITECTURE DESIGN 31

PROCESS MODEL 40

INTERFACES 43

ARCHITECTURE ILLUSTRATES 49

ACTIVITY DIAGRAMS 50

SEQUENCE DIAGRAM 54

STATE DIAGRAMS 55

CLASS DIAGRAM 57

CONTRIBUTIONS TABLE 59

Table of Contents

MEETING

FIRST MEETING 60

SECOND MEETING 61

THIRD MEETING 62

FOURTH MEETING 63

FIFTH MEETING 64

SIXTH MEETING 65

SEVENTH MEETING 66

EIGHTH MEETING 67

Introduction

Paylytics is an intelligent personal financial management system designed to simplify budgeting, automate expense tracking, and provide practical financial insights. It addresses the growing need for accessible financial literacy tools by turning raw spending data into clear, actionable guidance. The system empowers users to make informed decisions through personalized budgets, savings recommendations, and intuitive financial analysis. This SRS outlines the essential requirements, functionalities, and design considerations needed to guide the development of Paylytics as a multi-platform, user-centric financial wellness application.

Core System Components

1. Expense Management Module

- Automated categorization of expenditures (e.g., essential, discretionary, emergency)
- Real-time tracking of spending with customizable category definitions
- Historical spending analysis and trend detection
- Income-based budgeting guided by financial principles such as the 50/30/20 rule and zero-based budgeting
- Dynamic budget adjustment based on spending behavior and life events
- Predictive budgeting for seasonal or recurring expenses
- Multi-category budget limits with intelligent alerts and warning systems

2. Savings Optimization System

- Goal-oriented savings planning with milestone tracking
- Automated savings recommendations aligned with income patterns
- Progress visualization with positive reinforcement features
- Emergency fund estimation and monitoring tools

3. Analytical Reporting Dashboard

- Comprehensive financial health scoring system
- Interactive data visualization of spending patterns and trends
- Comparative analysis against demographic or custom user benchmarks
- Exportable financial reports for professional or personal review

4. Security and Privacy Framework

- Bank-level encryption for all sensitive financial data
- Biometric and multi-factor authentication options
- Local data processing with optional cloud synchronization
- Transparent data usage policies with granular consent management

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.

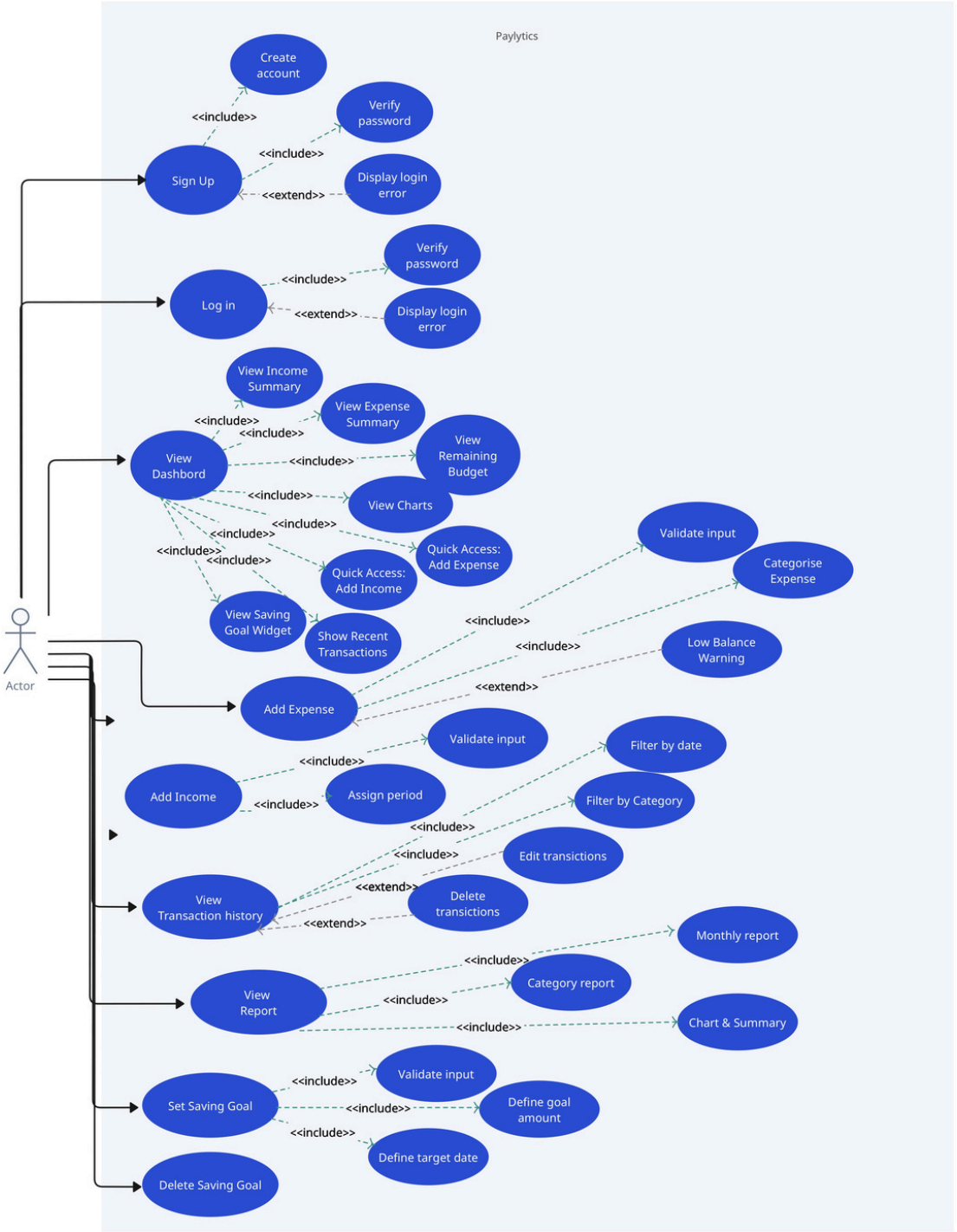
SOFTWARE INTERFACES:

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.

Software Design Document (SDD)

Software Architecture Design

Introduction

The System Design Document (SDD) is created to explain the general design of the system and how all the main parts are arranged together. This document is based on the SRS we already finished, and it helps us move from understanding the requirements to actually planning how the system will work in a clear and organized way. The idea of the SDD is not to go into too much detail, but to give a complete picture of the structure of the system before starting any development.

In this document, we describe how the system is divided into different components and how these components will interact with each other during the system's operation. This includes showing the architecture model, describing the main diagrams, and explaining how the components fit together to support the features we need in the project. The SDD helps the team have the same understanding about the system and reduces confusion later during design or implementation.

Since our project uses the Agile (Scrum) method, the design is not fixed from the beginning. It can be improved or changed while we work on the sprints. This makes the design more flexible, because sometimes new things appear while working, or we find better ways to design some parts of the system. So, the SDD is not a final version, but more like a guide that we can update whenever the sprint reviews show something important.

The SDD contains several main sections. One of them is the system architecture model, which explains the layers of the system and how the application will be divided (for example, the user interface layer, the logic layer, and the storage layer). It also includes the main UML diagrams that show how the user interacts with the system and how the system handles different processes, such as adding income or recording expenses. Another important part of the SDD is the class diagram, which shows the main classes in our system, their attributes, methods, and the relationships between them.

The main purpose of this document is to provide a clear understanding of how the system is structured, so that the team can move smoothly into the design and implementation stages. It also helps make sure the system design matches the requirements we wrote earlier and supports the functions we want to include in our budget and expense management application.

Software Architecture Design

Requirement 1: Add Monthly Income

Description:
The system shall allow users to record and store their monthly income entries with details such as amount, source, and date.

Input:
· Income amount (decimal)
· Source description (optional)
· Date received
· Frequency (monthly, weekly, one-time)

Output:
· Updated total income calculation
· Updated dashboard display
· Stored income record in database

Pre-conditions:
· User must be logged into the system
· Valid positive amount must be entered

Post-conditions:
· Income record is saved to local storage
· Dashboard reflects new total income
· Available budget is recalculated

Operational Considerations:
Income entries are validated for positive values and reasonable amounts to prevent data entry errors.

Requirement 2: Record and Classify Expenses

Description:
The system shall allow users to record expenses with categorization and detailed information.

Input:
· Expense amount
· Category selection
· Date of expense
· Optional notes
· Payment method

Output:
· Updated expense total
· Category-wise expense tracking
· Budget balance update

Pre-conditions:
· User must be logged in
· Valid category must be selected
· Amount must be positive

Post-conditions:
· Expense record is stored
· Remaining budget is updated
· Category totals are recalculated

Side Effects:
· May trigger budget warning if expense exceeds remaining budget
· May affect savings goal progress calculations

Requirement 3: Display Financial Dashboard

Description:
The system shall present a comprehensive dashboard showing total income, total expenses, and remaining budget.

Input:
· Aggregated income data
· Aggregated expense data
· Time period selection

Output:
· Visual summary of finances
· Charts/graphs representation
· Quick financial status overview

Pre-conditions:
· User must be logged in
· System must have financial data to display

Post-conditions:
· User can view current financial status
· Dashboard is updated with latest calculations

Operational Considerations:
Dashboard calculations must be efficient to provide instant feedback without noticeable delay.

Requirement 5: Create and Manage Savings Goals

Description:
The system shall allow users to set, track, and manage savings goals with target amounts and deadlines.

Input:
· Goal name and description
· Target amount
· Deadline date
· Current saved amount

Output:
· Goal progress visualization
· Time-based tracking
· Achievement notifications

Pre-conditions:
· Target amount must be positive
· Deadline must be in the future

Post-conditions:
· Goal is saved and tracked
· Progress is calculated automatically

Operational Considerations:
Goal tracking must update automatically with each income/expense transaction.

Requirement 6: Transaction History Management

Description:
The system shall maintain and display a complete history of all financial transactions with filtering and search capabilities.

Input:
· Date range filters
· Category filters
· Search keywords

Output:
· Filtered transaction list
· Detailed transaction view
· Edit/delete options

Pre-conditions:
· User must have transaction history
· Valid filter criteria

Post-conditions:
· User can review and manage past transactions
· Changes are reflected in financial totals

Requirement 7: Edit/Delete Past Transactions

Description:
The system shall allow users to modify or remove previously recorded transactions.

Input:
· Selected transaction for editing
· Updated transaction details
· Delete confirmation

Output:
· Updated transaction record
· Recalculated financial totals
· Confirmation message

Pre-conditions:
· Transaction must exist
· User must have permission to modify

Post-conditions:
· Financial totals are updated
· All related calculations are refreshed

Side Effects:
· May affect historical reports and trend analysis

Requirement 8: Non-Functional Requirements Implementation

Description:
The system shall meet all specified non-functional requirements including usability, performance, security, and aesthetics.

Implementation Strategy:
· Usability: Intuitive interface with minimal learning curve
· Performance: Response time under 3 seconds for all operations
· Security: Local data storage with encryption where applicable
· Aesthetics: Clean, uncluttered interface with consistent design

Operational Considerations:
All non-functional requirements are treated as critical success factors and are measured throughout development.

Process Model And Activities

Selected Software Process Activity Method and Rationale

Selected Method: Agile (Scrum Methodology)
The Agile Methodology, through (Scrum) is selected as the process framework for developing the personal budget and expense calculator. The method is selected because of the project need for frequent refinements of the interface, as well as the evolving nature of the project.

Rationale for Selecting the Method:

1- Frequent User Input and Feedback: The project requires frequent input from users to help ensure complex features (like budget management and saving goals) are useful and responsive to actual user needs; the Scrum framework supports frequent interaction each sprint that enables changes immediately.

2-Flexibility and Ability to Adapt When Requirements Change: In an ongoing project, user requirements are subject to change (for example, the addition of new categories or features, and refinement of the budgeting logic). The Scrum methodology supports changing requirements without a disruption to the core development activity.

3-Incremental/Incremental Delivery: Scrum organizes project activities to allow delivery of working portions of the system in short delivery cycles (sprints). This supports early delivery of key features, like expense tracking, though more complex feature will be inclementally delivered, in lo sectinos to avoid risk.

4-Transparency and Ownership: The Scrum process - framework idea and transparency is alive through Sprint Review and Retrospective events.

Scrum Implementation and Documentation Information In order to illustrate the use of this model, the System Design Document (SDD) will address the implementation of Agile (Scrum):

1- Software Process Activities:

Development will occur as described around the following scrum ceremonies:

- Sprint Planning: Determine which features (e.g., expense tracking and savings goals) will get prioritized and then broken into smaller tasks for development within that sprint.
- Daily Stand-Ups: Daily short meetings to continually communicate about progress being made as well as quickly solve any issues.
- Sprint Review: Once part of the work is completed, the work will be demonstrated to elicit feedback for improvements in the future.
- Sprint Retrospective: The team will reflect on workflow and process to enhance efficiency as well as strategy in the next sprint.

2- SDD Documentation Considerations:

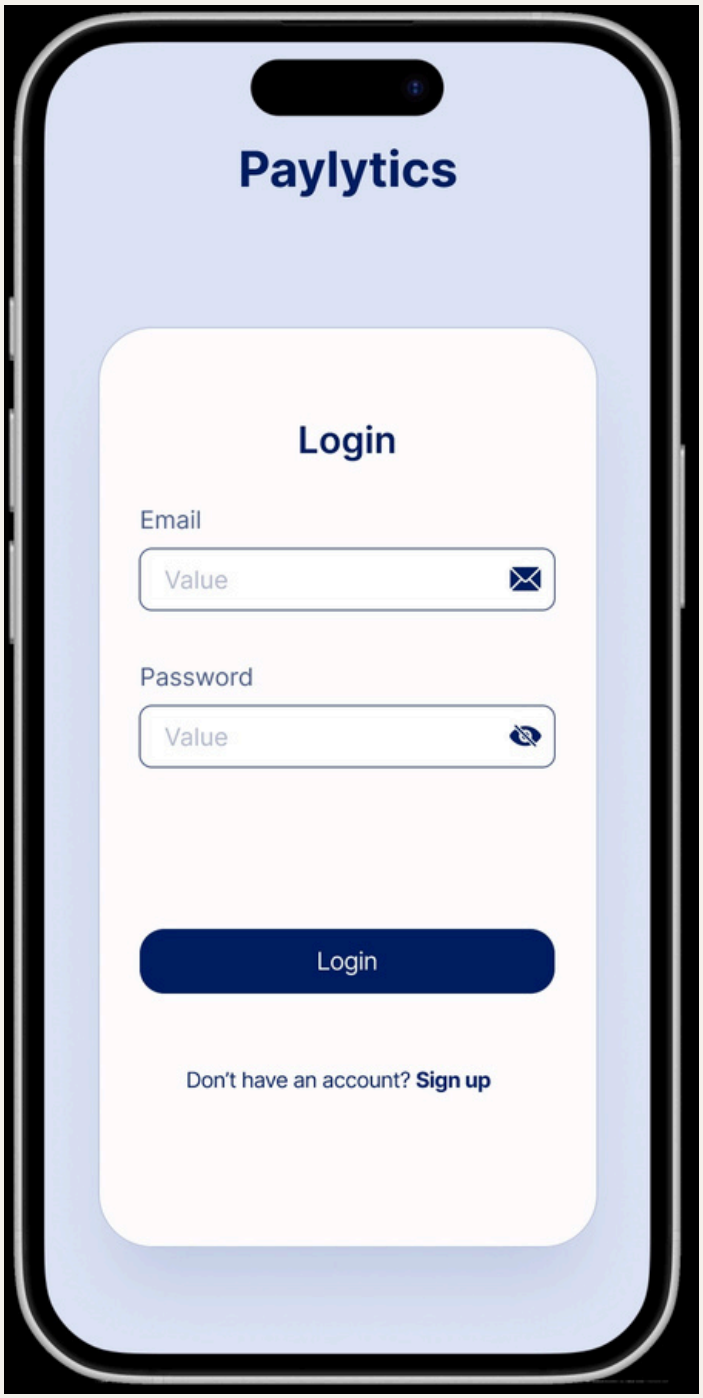
The SDD will focus on describing the Scrum practice more specifically:

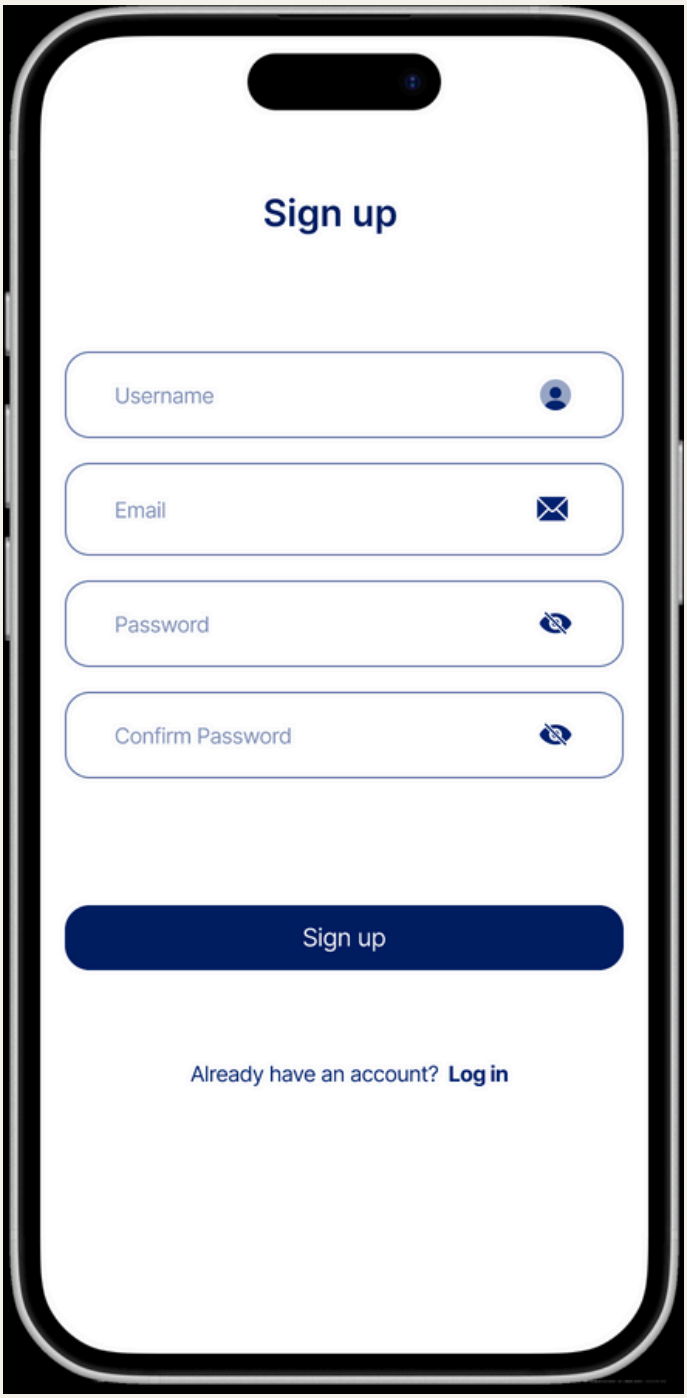
- Breakdown of Sprint: How the project is organized into sprints, each part addressing specific and deliverable feature sets.
- Task Priorization: Criteria for prioritizing features and tasks for each sprint iterations.
- User Input: The way user input will be formally collected and addressed from the previous iteration into the Agile type for the next iteration reflecting the fact that it is an iterative process, as well as continuous improvement.

1- Software Process Activities:

SPRINT	CONTRIBUTION
SPRINT 1	*Project setup, requirements gathering, basic Ul design*
SPRINT 2	*Core architecture, database design, basic navigation*
SPRINT 3	*Income/Expense recording, dashboard implementation*
SPRINT 4	*Reporting features, savings goals functionality*
SPRINT 5	*Advanced features, testing, documentation*
SPRINT 6	*Final integration, bug fixes, presentation preparation*

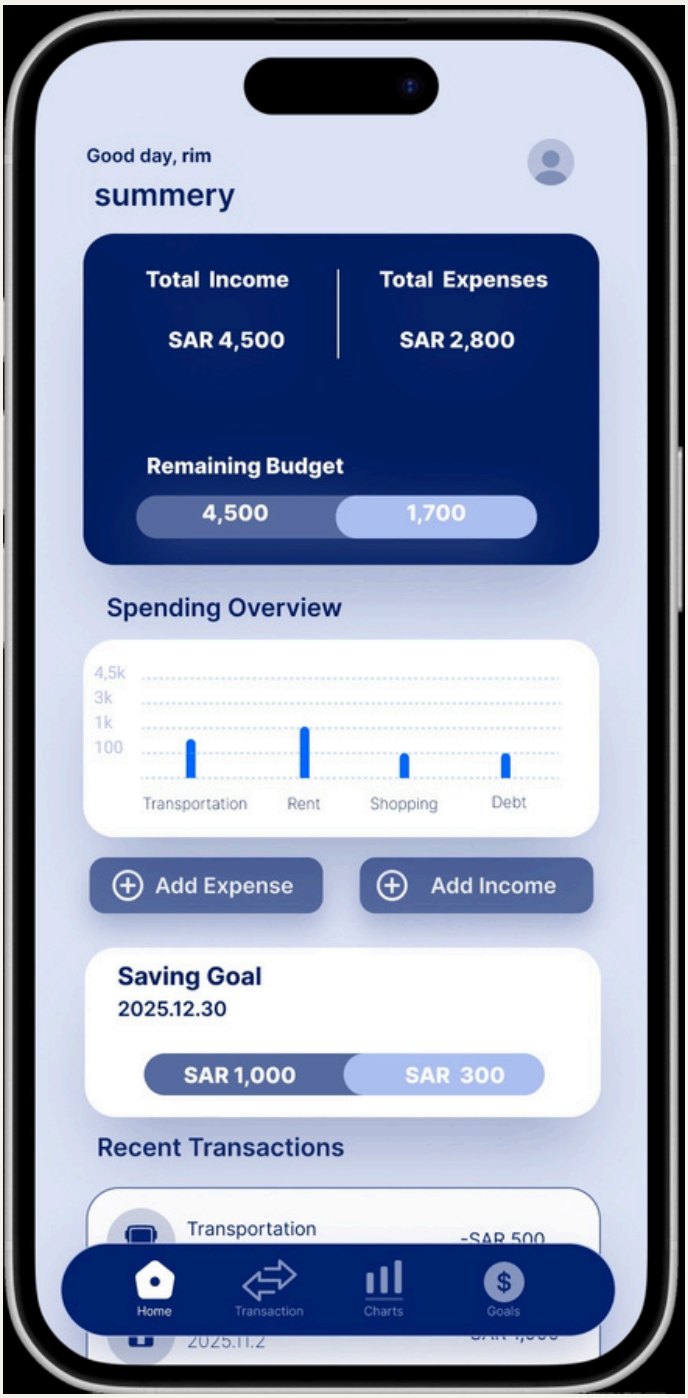
Interface application

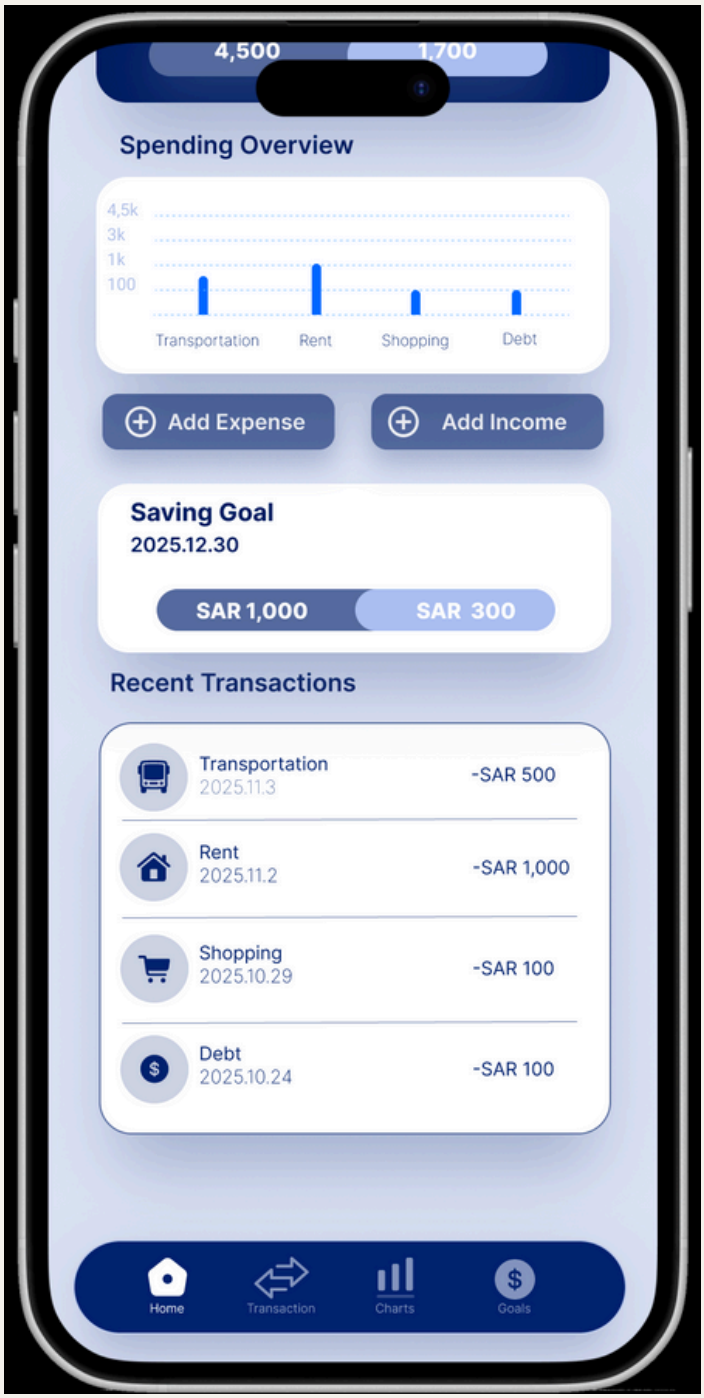




Clear pictures here :

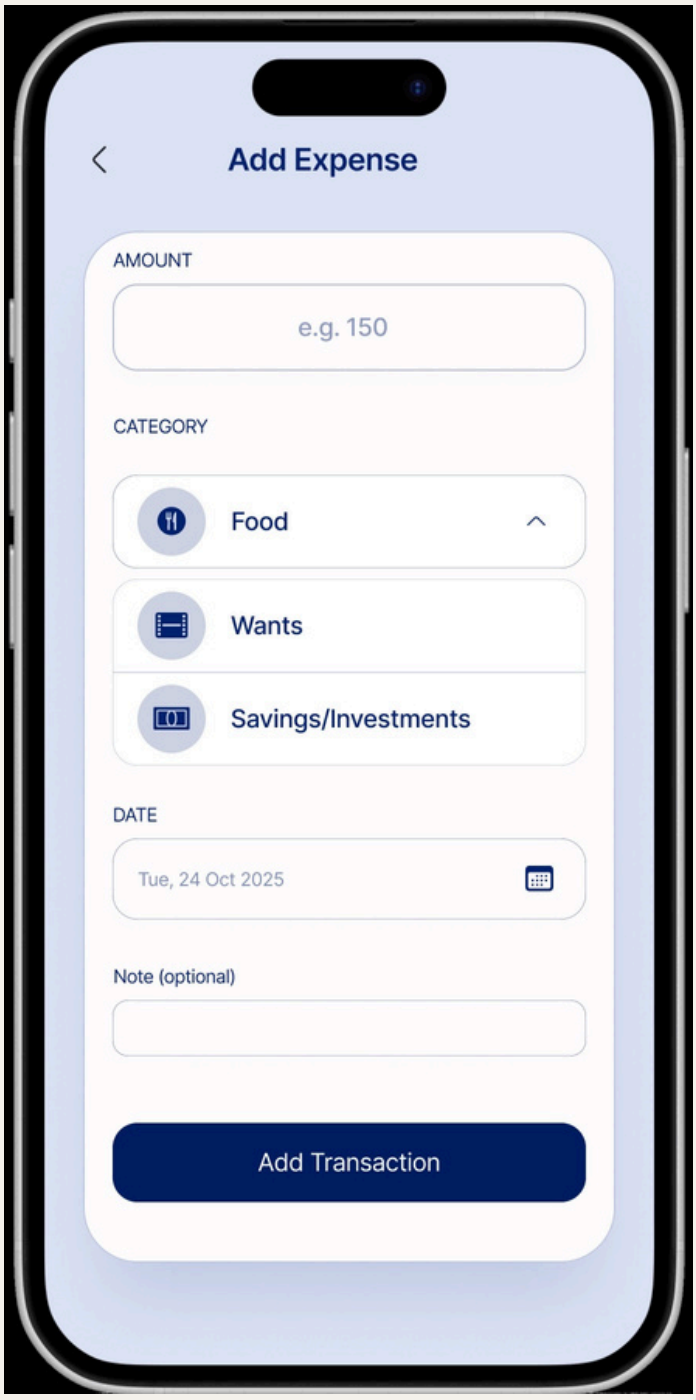
Interface application

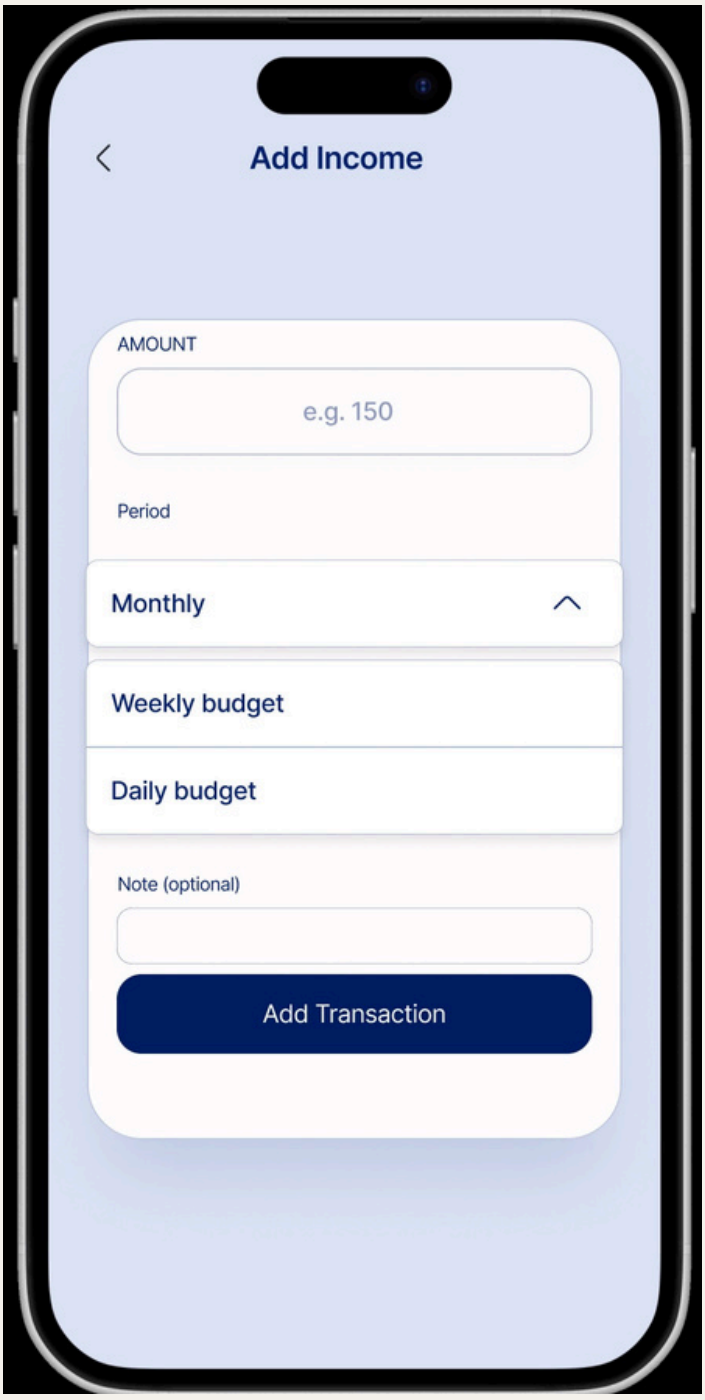




Clear pictures here :

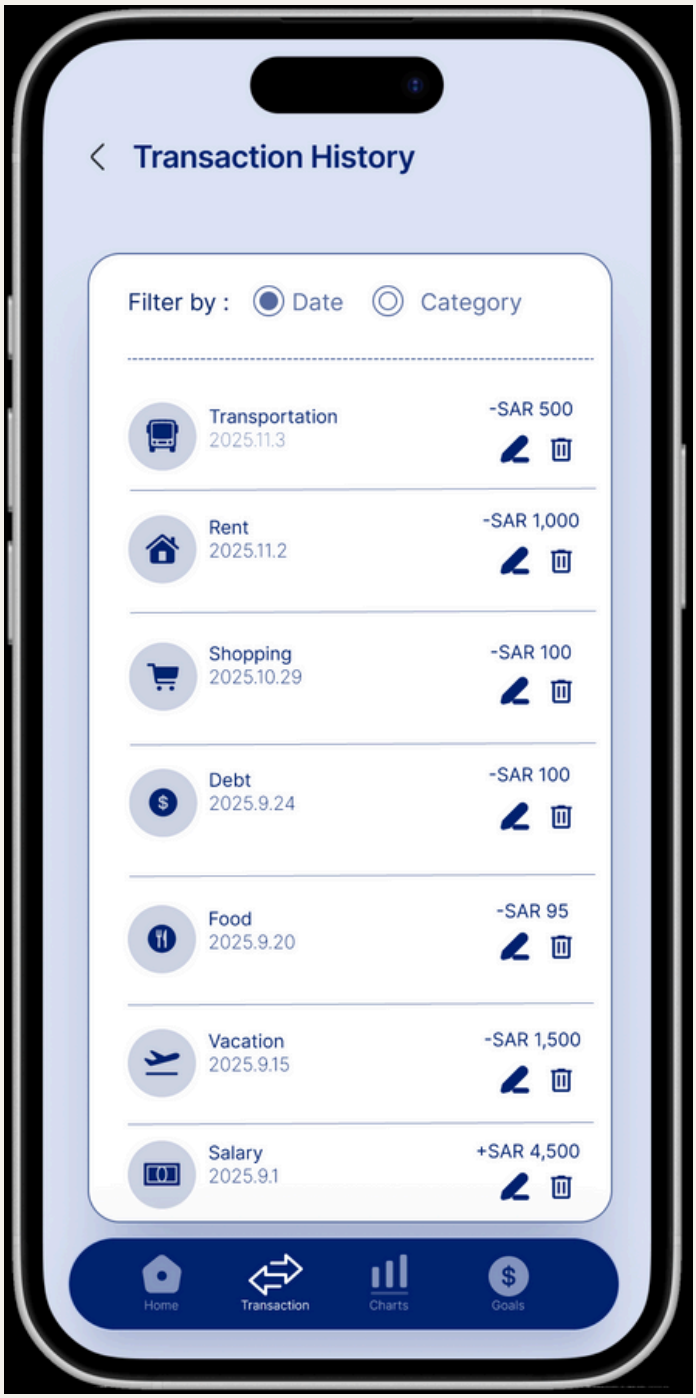
Interface application

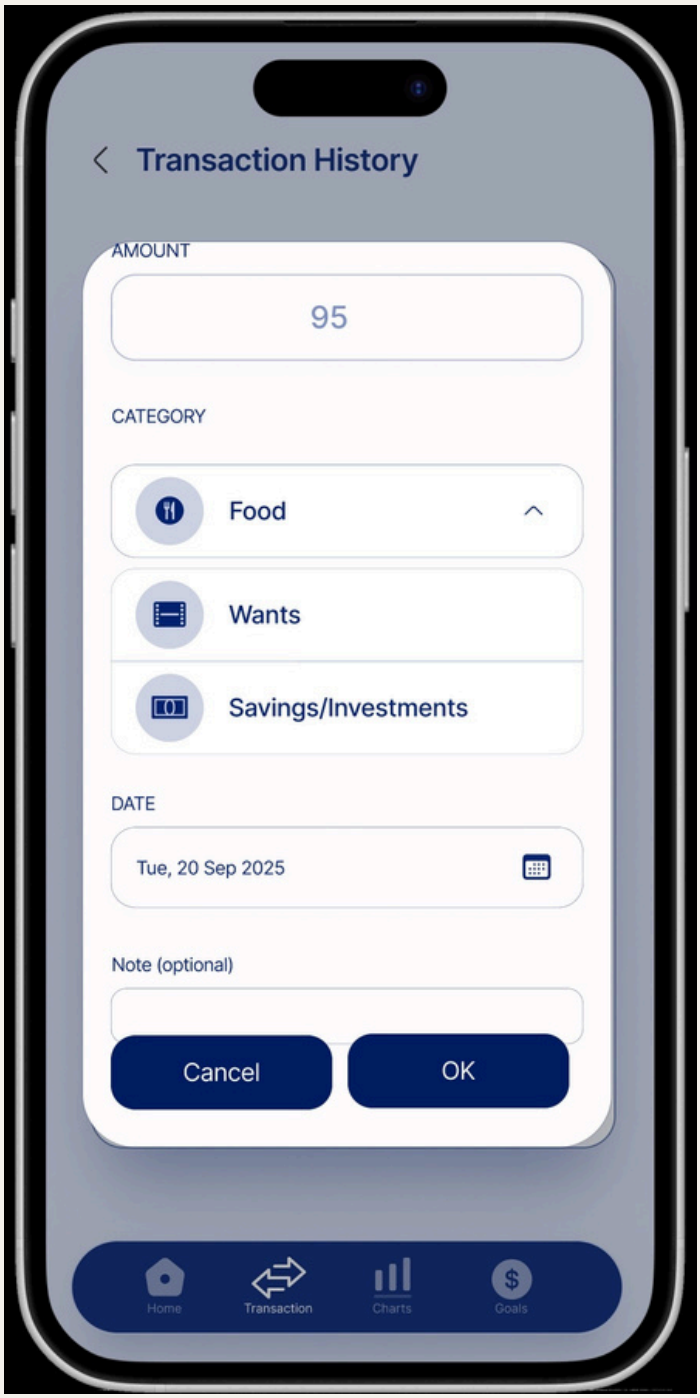




Clear pictures here :

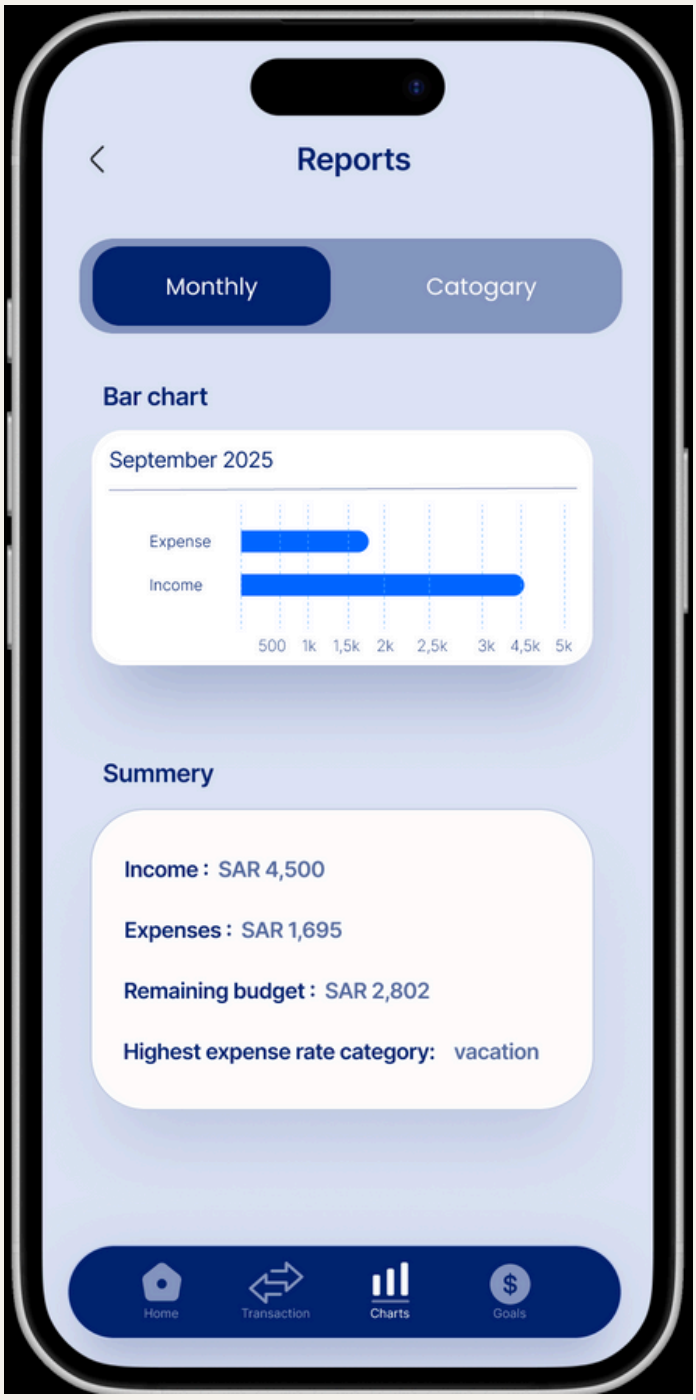
Interface application





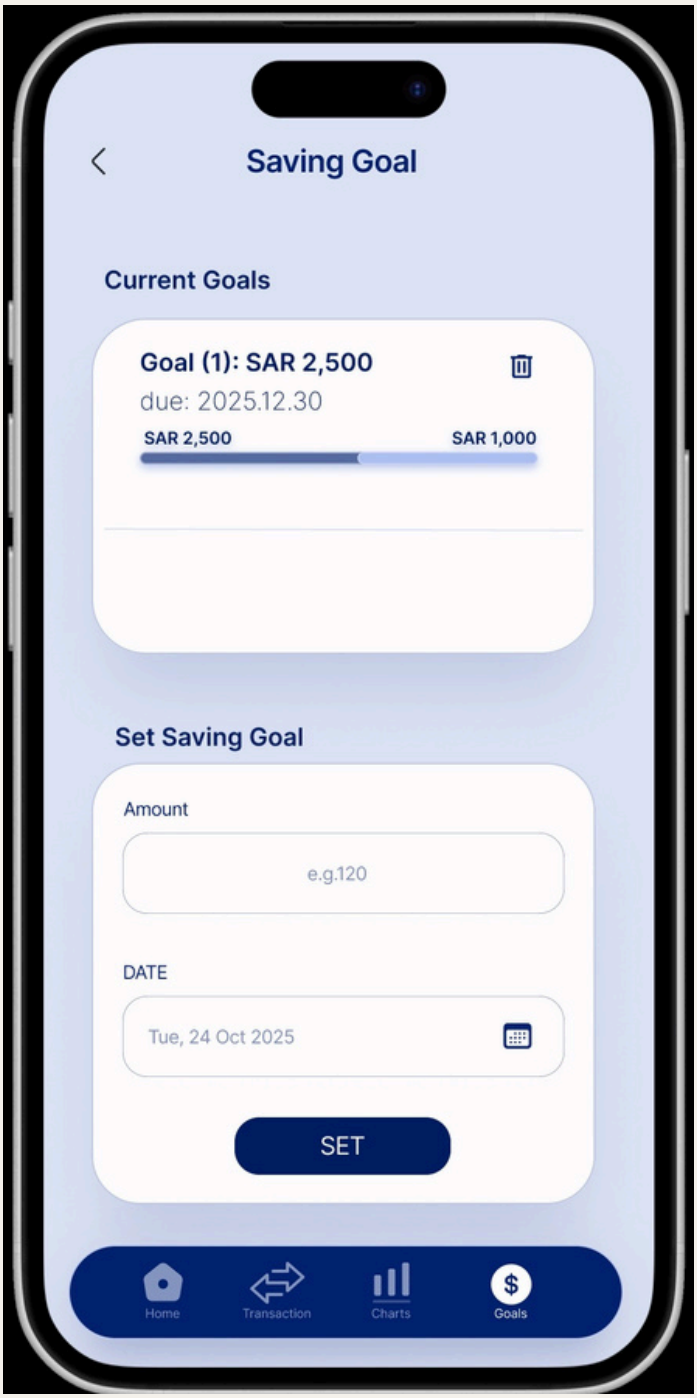
Clear pictures here :

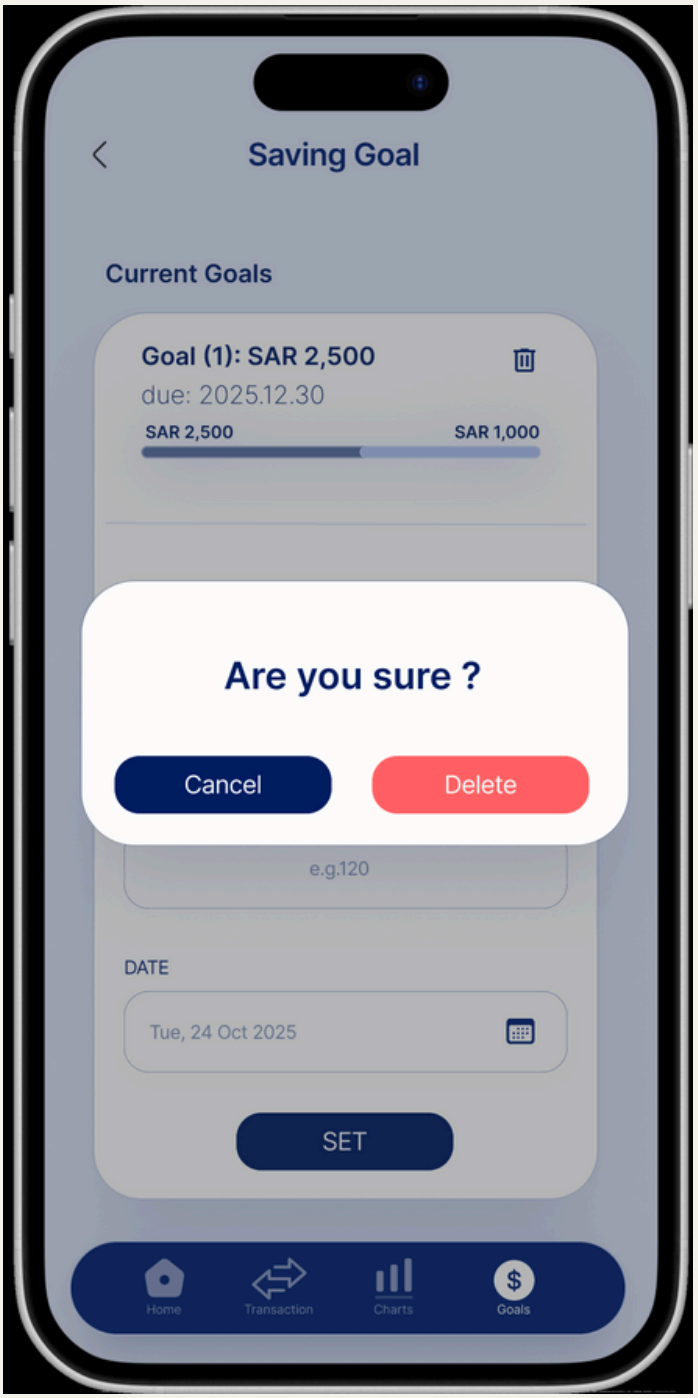
Interface application



Clear pictures here :

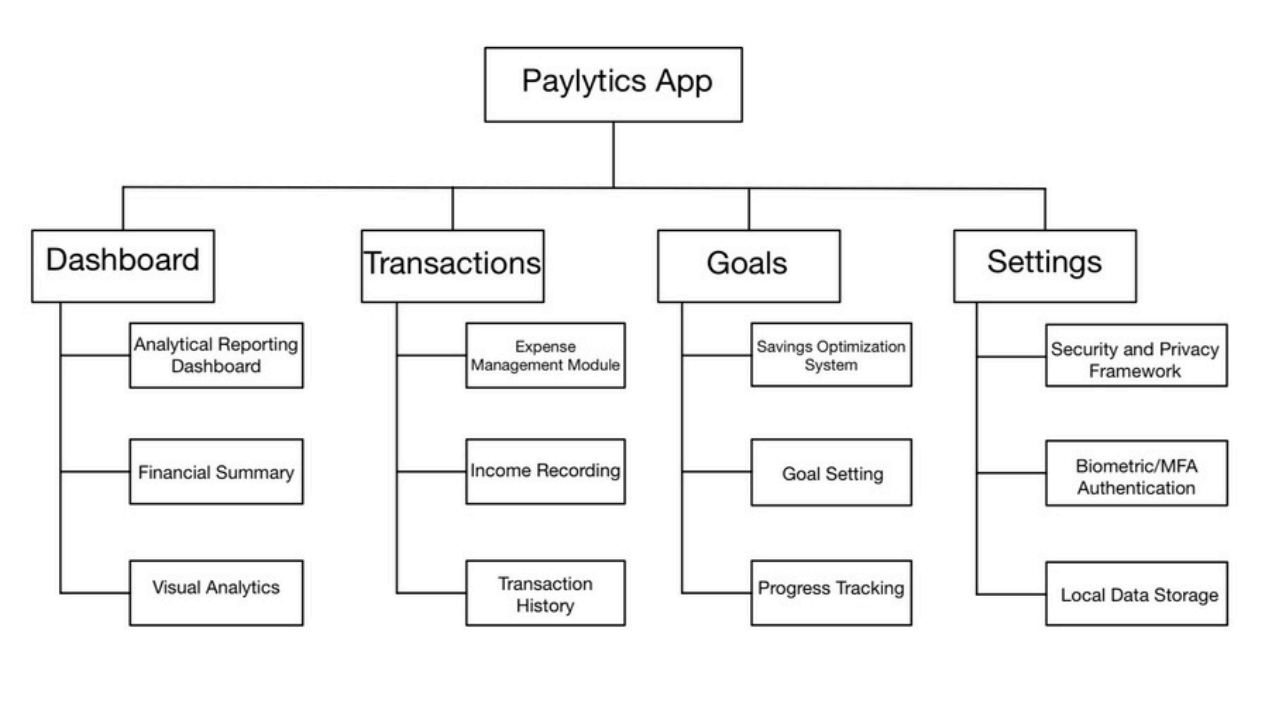
Interface application





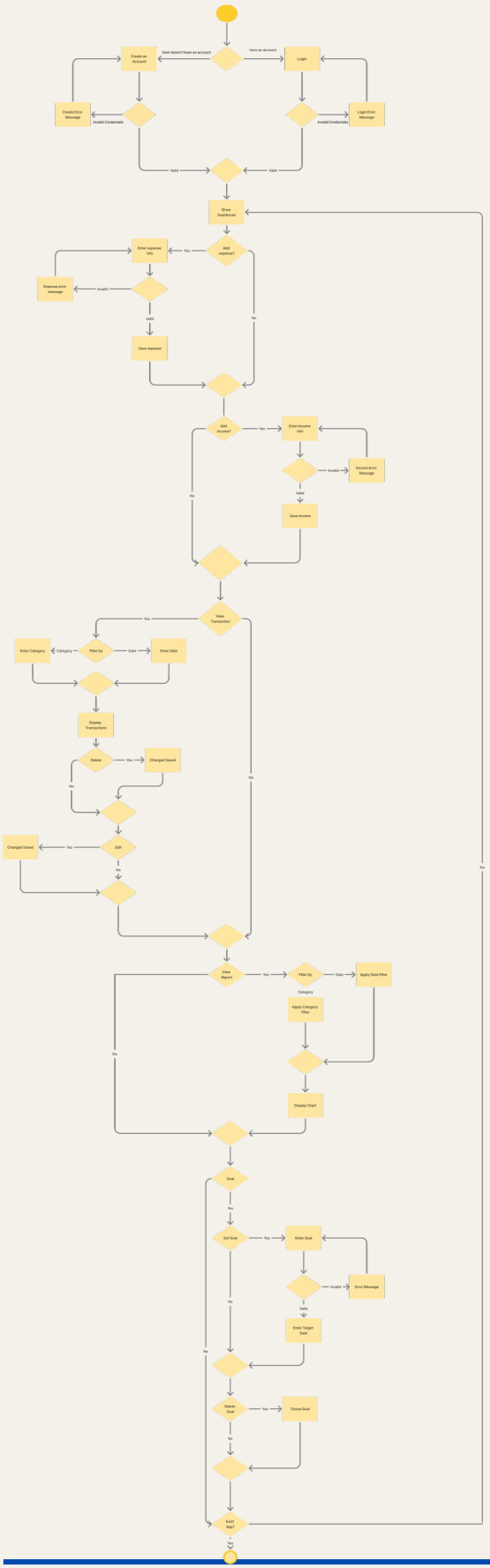
Clear pictures here :

architecture illustrate



The diagram illustrates the main architecture of the Paylytics App by dividing it into four fundamental functional domains. These primary sections include the Dashboard for displaying analytics, Transactions for recording the movement of funds, Goals for managing savings and financial planning, and finally Settings for managing security and local data. Specialized working units branch off from each section to execute the system's full range of functions.

Activity diagram

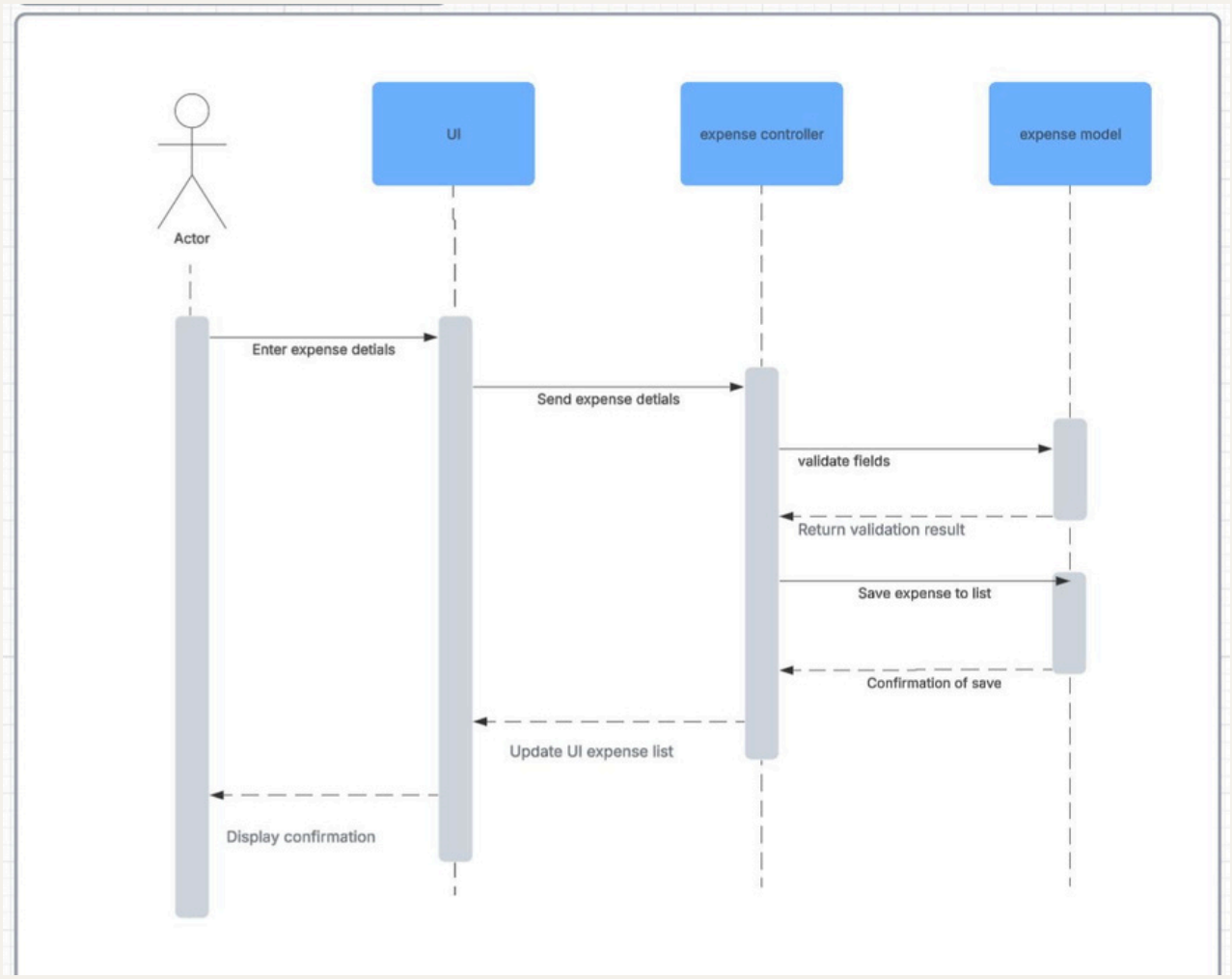


This activity diagram shows how user interact with the app. The user start by logging in or creating an account, and the system guides them if something goes wrong. Once in, users can add expenses or income, review transactions and reports, filter transactions, and set or delete goals.

Clear pictures here :

Sequence diagram

Sequence Diagrams

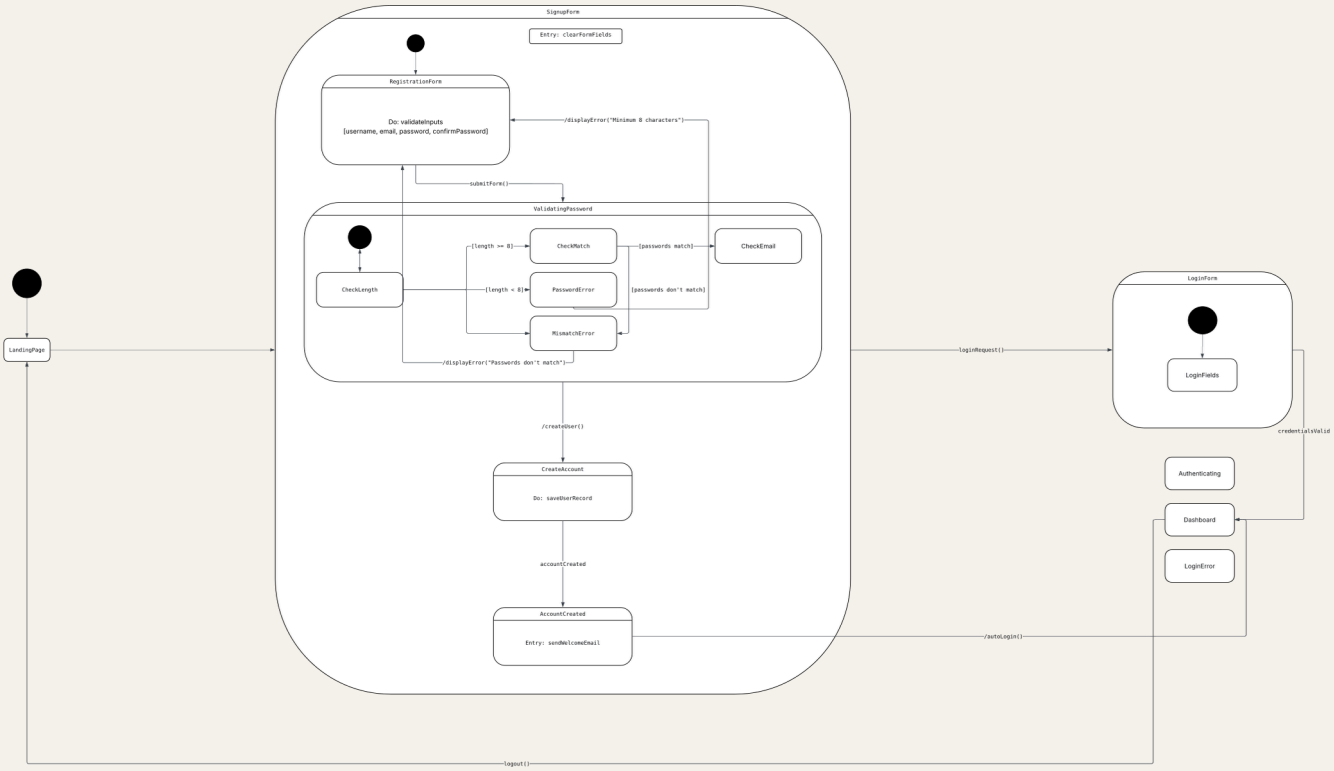


This sequence diagram describes the process of adding a new expense to the system.
The user begins by entering the expense details through the UI and UI sends these details to the Expense Controller, which forwards the data to the Expense Model for field validation.
The model validates the fields and returns the validation result to the controller, the controller instructs the model to save the expense to the list. After saving, the model returns a confirmation message to the controller

Clear pictures here :

State Diagram

State Diagram: Authentication State Diagram

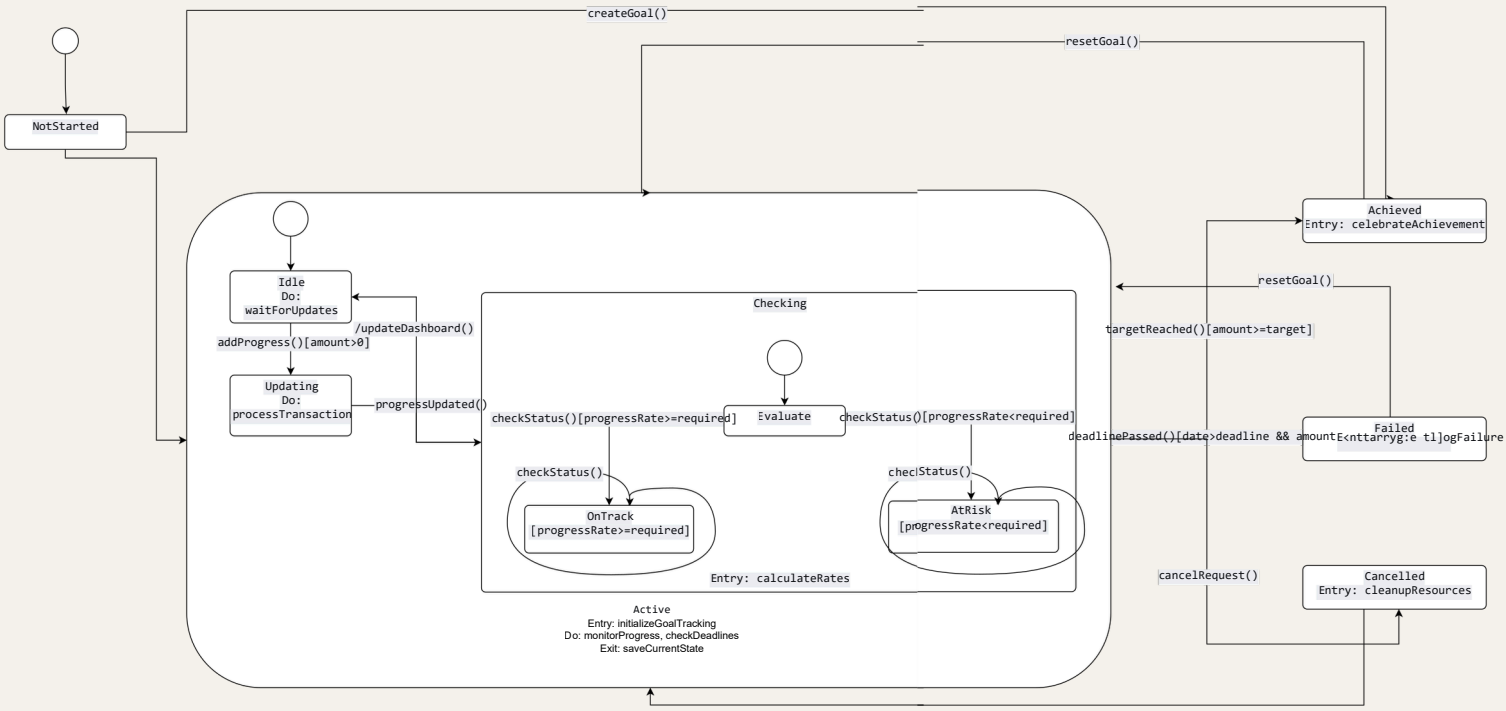


This state diagram describes the authentication flow for account creation and login. The process begins on the LandingPage, where users choose to sign up or log in. New users move to the SignupForm, which validates password length and confirmation before creating an account. Existing users go to the LoginForm, which handles authentication. After successful signup or login, the user enters the Dashboard, the main application interface.

During signup, the system checks password length through CheckLength and ensures both password fields match through CheckMatch, with error states providing clear feedback when needed. A successful registration automatically logs the user in. The diagram supports FR-1 by enabling account creation, FR-2 by validating password length, and FR-3 by verifying password matching.

State Diagram

State Diagram: Savings Goal State Diagram

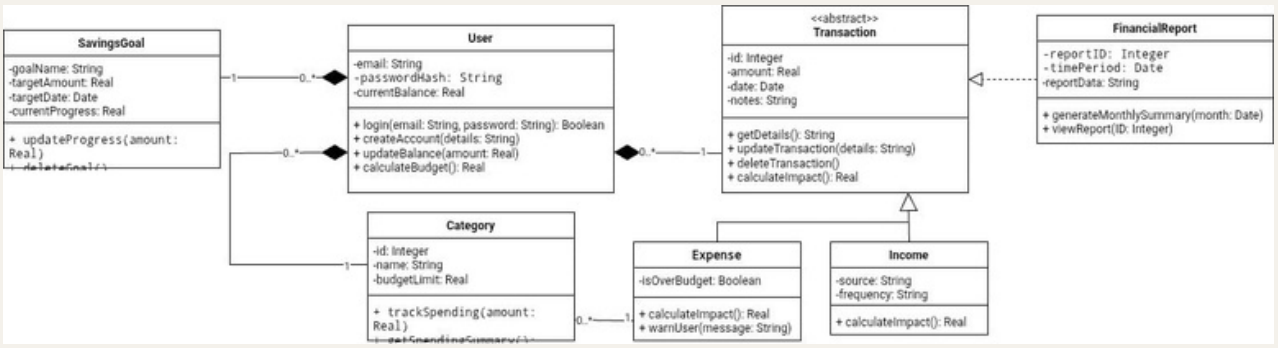


This state diagram shows how the system manages a savings goal from creation to completion or failure. It begins in NotStarted before the goal is created, then moves to Active, where progress is tracked. Within Active, the goal may be Idle while waiting for updates, Updating when new progress is added, or Checking when the system evaluates performance.

During checking, the system calculates progressRate = currentAmount / daysElapsed and compares it to requiredRate = targetAmount / totalDays. If the progressRate meets or exceeds the requiredRate, the goal is OnTrack; otherwise, it becomes AtRisk.

The goal enters Achieved when targetReached() confirms the amount has reached the target, or Failed when deadlinePassed() occurs and the amount is still below the target. The user may also trigger cancelRequest() to move it to Cancelled. From any outcome, resetGoal() can return the system to Active to start again.

Class Diagram



The Class Diagram above models the static structure of the Paylytics application, defining the system's core entities, their properties (attributes and operations), and the relationships that govern data integrity and system behavior.

1. Core Entities and Traceability

The system is organized around seven main classes:

- User: The central class, modeling the client application user account and maintaining the overall balance.
- Transaction: An abstract superclass designed to hold common data (amount, date) shared by all financial movements. This enforces structural consistency between Income and Expense records.
- Income and Expense: Concrete subclasses of Transaction that implement specific functionality for earning and spending money.
- Category and SavingsGoal: Classes modeling key user-defined financial structures, fulfilling the budget classification (FR.5) and goal-tracking (FR.8, FR.9) requirements.
- FinancialReport: The class responsible for generating and summarizing complex data structures (FR.7, FR.6) based on the recorded transactions.

Clear pictures here :

2. Key Relationships and Multiplicity

The design uses three critical types of relationships to ensure data integrity and traceability:

A. Generalization (Inheritance)

- The Income and Expense classes inherit all properties from the Transaction superclass. This is represented by the solid line with an unfilled triangle.

B. Composition (Strong Ownership)

- The Composition relationship (represented by the filled diamond) is used to model the system's data architecture, where the User is the primary owner.
- This relationship signifies that the data entities (Transaction, SavingsGoal, Category) cannot exist outside of the User object. If a user record is deleted, all associated financial data must be deleted instantly.
- The multiplicity is One-to-Many: One user owns zero or more of these data records.

C. Association (Classification)

- A simple solid line connects Expense and Category to model the requirement that every expense must be classified.
- The multiplicity is enforcing the rule that every expense belongs to exactly one category, and a category can contain many expenses.

D. Dependency (Usage)

- A dashed line with an open arrow is drawn from the FinancialReport class to the Transaction class. This signifies that the report component relies on the transaction data to execute its functions, but it does not own the data.

Contributions Table

MEMBER	CONTRIBUTION
DEEM ALBASSAM	*Project Coordination, Final Review and Integration , uploading documents in repository*
HAYDAB ALKHALIFAH	*Ethics and Responsibilities*
JOUDI ABAZAI	*process activities and models*
NOURAH ALAWADH	*Requirements Engineering , System Overview and Use Case Diagram , UML Diagrams*
REEMA ALRSHEED	*External Interface Requirements , interface applications*
RAMAH ALHAMDAN	*Functional Requirements and Non-Functional Requirements*
FAJR ALGHWAFELI	*Complete class diagrams*
YASMIN ALTWAIJIRY	*illustrate architecture model*
RENAD ALMOHSEN	*System/Software Design ,sequence diagram*
BANA ALTUWAYJIRI	*activity diagram*

Meeting Notes

DETAILS

Date: 27.10.2025
Time: 6:00pm
Location: WhatsApp
Topic: Project Kickoff Meeting

ATTENDEES

Notetaker: Deem Al-Bassam
Attendees: All 10 members
Absence: -

AGENDA

1. *Project Introduction and Topic Brainstorming*
2. *Final Topic Selection: Expense Calculation App*
3. *Initial Task Distribution Among Members*
4. *Set Timeline for SRS Document Drafting*

ACTION POINTS

1. *Discussed and finalized "Expense Calculation App" as the project topic.*
2. *Broke down initial tasks and distributed responsibilities among all 10 team members.*
3. *Agreed to start working on the Software Requirements Specification (SRS) document.*
4. *Set deadline for initial SRS contributions from each member.*

Meeting Notes

DETAILS

Date: 31.10.2025
Time: 1.5 hours
Location: WhatsApp
Topic: Detailed Task Assignment & Timeline Planning

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Review project requirements from guidelines

2. Break down SRS document sections

3. Assign specific tasks to each member

4. Set deadlines for SRS completion

5. Discuss tools for UML diagrams

ACTION POINTS

- *Create GitHub repository and set up folder structure*
- *Write System Overview and Requirements Engineering sections*
- *Draft Functional Requirements*
- *Draft Non-Functional Requirements*
- *Create Use Case Diagram and External Interface Requirements*
- *Review assigned sections and complete within 3 day*

Meeting Notes

DETAILS

Date: 5.11.2025
Time: 1 hours
Location: WhatsApp
Topic: SRS Draft Review & Alignment

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Individual progress updates on SRS sections

2. Review completed functional requirements

3. Discuss any challenges or questions

4. Ensure consistency across all sections

5. Plan for final SRS compilation

ACTION POINTS

- *Add 2 more functional requirements to reach 8 total*
- *Clarify security and performance requirements*
- *Finalize use case diagram using Lucidchart*
- *Begin compiling all sections into one SRS document*
- *Submit individual sections to DEEEM by tomorrow*

Meeting Notes

DETAILS

Date: 10.11.2025
Time: 2 hours
Location: WhatsApp
Topic: Complete SRS Document Finalization

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Walk through complete SRS document section by section

2. Verify all requirements are clear and testable

3. Ensure use case diagram is included and clear

4. Confirm ethics section addresses 4+ principles

ACTION POINTS

- *Make final formatting adjustments to SRS*
- *Add missing details to requirements engineering section*
- *Export high-quality use case diagram as PNG/PDF*
- *Review final SRS version and approve for submission*
- *Upload SRS to GitHub repository*

Meeting Notes

DETAILS

Date: 15.11.2025
Time: 1.5 hours
Location: WhatsApp
Topic: Starting Software Design Docume

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Transition from SRS to SDD phase

2. Review required UML diagrams from guidelines

3. Assign diagram creation tasks

4. Discuss software architecture design

5. Set SDD completion timeline

ACTION POINTS

- *Work on Software Architecture Design section*
- *Create Class Diagram and Sequence Diagrams*
- *Develop State Diagram and Activity Diagram*
- *Design Data Flow Diagrams (Level 0 & 1)*
- *Document Process Model (Agile/Scrum implementation)*
- *Complete assigned diagrams within 5 days using Lucidchart*

Meeting Notes

DETAILS

Date: 20.11.2025
Time: 2 hours
Location: WhatsApp
Topic: UML Diagrams
Review & Feedback

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Review all completed UML diagrams

2. Provide feedback on clarity and accuracy

3. Ensure diagrams align with SRS requirements

4. Identify any missing components

5. Discuss integration of all diagrams into SDD

ACTION POINTS

- *Adjust class diagram to include all entity relationships*
- *Simplify state diagram for better readability*
- *Ensure DFD levels are consistent and logical*
- *Begin compiling SDD document with all diagrams*
- *Make final adjustments to diagrams within 2 days*

Meeting Notes

DETAILS

Date: 25.11.2025
Time: 1 hours
Location: WhatsApp
Topic: Complete SDD Finalization

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Review complete Software Design Document

2. Verify all required diagrams are included and clear

3. Check architecture model and process model sections

4. Ensure contributions table is completed

5. Confirm document formatting meets guidelines

ACTION POINTS

- *Finalize SDD formatting and organization*
- *Review assigned sections in final SDD*
- *Verify architecture design accurately represents system*
- *Upload final SDD to GitHub repository*
- *Prepare for presentation development in next meeting*

Meeting Notes

DETAILS

Date: 2.12.2025
Time: 1 hours
Location: WhatsApp
Topic: Project Completion & Presentation Development

ATTENDEES

Attendees: All members
Absence: -

AGENDA

1. Review complete project (SRS + SDD + Meetings)

2. Develop presentation slides structure

3. Assign presentation speaking parts

4. Practice presentation timing

5. Final submission checklist

ACTION POINTS

- *Create presentation slides using the structure we discussed*
- *Prepare to explain requirements engineering section*
- *Practice explaining key UML diagrams*
- *Ready to discuss ethics and responsibilities*
- *Prepare to demonstrate system features*
- *Individual submission of final project files*
- *Ensure GitHub repository is complete and organized*

References and links

ALL LINKS AND REFERNCES THROUGHOUT THE REPORT

Repostrry

https://github.com/Suioii/Paylytics

UML DIGRAM

https://app.creately.com/d/Ue7dAnhTZrX/edit

Interface application

https://www.figma.com/proto/iwHNW1NyQZrFhDpgHP7SKt/Paylytics?node-id=0-1&t=MijjGyzRWmGyyeQI-1

Activity diagram

https://www.figma.com/board/3FgL9ZUVrbyqvocG6M1ZqG/UML-Activity-Diagram--Community-?node-id=403-3954&t=nw9z7haPuWyGyd2N-1

Sequence diagram

https://lucid.app/lucidchart/dd5366d0-c4f6-46ed-ae90-80cf3d5eb1d7/edit?invitationId=inv_67f60943-de61-474c-91a7-7bde108d8f50

State Diagram

https://lucid.app/lucidchart/8d87aa7d-c107-488f-8cb8-b9f753340ac7/edit?invitationId=inv_c26e2a16-89e1-4bfe-9698-d36300c16b9b

Class Diagram

https://drive.google.com/file/d/1h2t-_WPMU6ace7cJhDJuiH1HWXPMZfEW/view?usp=sharing