**MINISTRY OF EDUCATION AND TRAINING**

**CMC UNIVERSITY**



**ASSIGNMENT REPORT**

**COURSE: Mobile Application Development**

**Project Title : FinGuard: A Personal Financial Management Tool**

**Group Member :**

**Trần Quỳnh Anh - BIT230008**

**Nguyễn Hải Anh - BIT230500**

**Đoàn Anh Vũ - BIT230458**

**Hoàng Nguyên Giáp - BIT230135**

**Hà Nội, tháng năm 2024**

# Contents

[Contents 2](#_Toc13285)

[INTRODUCTION 2](#_Toc18788)

[LIST OF GROUP MEMBERS 3](#_Toc23406)

[CHAPTER I: SYSTEM INTRODUCTION 4](#_Toc14509)

[1. Introduction 4](#_Toc1542)

[2. System Survey and Research 4](#_Toc20779)

[2.1 System Objectives 4](#_Toc28383)

[2.2 Identifying Surveyed Subjects 4](#_Toc3654)

[3. Define System Requirements Specification 4](#_Toc9405)

[3.1 Similar application 4](#_Toc10440)

[3.2 Functional Requirements 4](#_Toc5637)

[3.3 Non-functional Requirements 4](#_Toc13088)

[CHAPTER II: TECHNOLOGY USED 5](#_Toc31372)

[1. Describe the technology used in the system 5](#_Toc8325)

[2. Describe the methodology used in the system 5](#_Toc28816)

[CHAPTER III: ANALYSIS OF THE SYSTEM 5](#_Toc22141)

[CHAPTER III: DATABASE DESIGN 5](#_Toc3773)

[CHAPTER V: INTERFACE DESIGN 5](#_Toc9044)

[PART VI: LIMITATIONS AND FUTURE IMPROVEMENTS 6](#_Toc31544)

[PART VII: CONCLUSION 6](#_Toc2433)

# INTRODUCTION

Managing personal finances is a crucial skill in today’s fast-paced and unpredictable world. Effective financial management allows individuals to prepare for emergencies, save for future goals, and make informed spending decisions. However, many people still lack the habit of tracking their income and expenses, which often leads to overspending, inadequate savings, and poor financial planning.

In response to this problem, our team has developed a mobile application that helps users manage their personal finances in a simple and efficient way. The application is built using the Flutter framework, ensuring cross-platform compatibility and a smooth user experience.

The app provides key features such as daily income and expense tracking, budget management, and financial goal setting. Through an intuitive and user-friendly interface, users can easily monitor their spending habits, control their budgets, and improve their overall financial health.

This project aims not only to offer a practical solution for individual users but also to help raise awareness about the importance of financial discipline and planning in daily life.

# IMPLEMENTATION PLAN

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Task** | **Description** | **Main Responsible Member** | **Time (Week)** | **Deliverables** |
| 1 | Requirement Analysis | Identify core features and design user flow | Trần Quỳnh Anh | Week 1 | List of planned features, use-case diagram |
| 2 | Wireframe Design | Create low-fidelity screen sketches | Đoàn Anh Vũ | Complete wireframes for all main screens |
| 3 | Database Design | Define tables, data models, and relationships | Nguyễn Hải Anh | Week 2 | ERD diagram, SQL schema or model classes |
| 4 | Flutter Project Setup | Initialize project structure and dependencies | Nguyễn Hải Anh | Flutter project on Git, with folder structure and dependencies set |
| 5 | Navigation Tree Setup | Configure screen navigation using Flutter routes | Nguyễn Hải Anh | Working route configuration for all screens |
| **Screens** | | | | | |
| 6 | Register/Login Screen | Design UI | Hoàng Nguyên Giáp | Week 3 | Register/Login Screen |
| 7 | Add Transaction (Income/Expense) Screen | Design UI | Trần Quỳnh Anh | Add Transaction Screen |
| 8 | Category Management Screen | Manage categories (add/edit/delete) | Hoàng Nguyên Giáp | Category management screen |
| 9 | Transaction History Screen | Display transactions by date/category | Đoàn Anh Vũ | Transaction History Screen |
| 10 | Budget Planning Overview Screen | All the current budget plans | Nguyễn Hải Anh | Budget planning overview screen |
| 11 | Set Budget Planning Screen | Add budget plans | Nguyễn Hải Anh | Set budget plans screen |
| 12 | Dashboard Screen | Show charts and summary (income, expenses) | Trần Quỳnh Anh | Dashboard screen with charts and totals |
| **API** | | | | | |
| 13 | Transaction API | Implement CRUD API for transactions | Trần Quỳnh Anh | Week 4 | TransactionController with endpoints tested |
| 14 | Categories API | Create API to manage categories (CRUD) | Hoàng Nguyên Giáp | CategoriesController with endpoints tested |
| 15 | Budget Planning API | API for setting and retrieving budget plans | Nguyễn Hải Anh | BudgetPlanningController with endpoints tested |
| 16 | Function Testing | Test features, find and fix bugs | Hoàng Nguyên Giáp | Week 6 | Test report with fixed bugs and screenshots |
| 17 | Report Writing | Document system overview, features, and development | Trần Quỳnh Anh | Week 7 | Draft of group report in Word/PDF |
| 18 | Final Report Compilation | Format and finalize the full group report | Trần Quỳnh Anh | Week 8 | Finalized PDF report ready for submission |

# CHAPTER I: PROJECT OVERVIEW

## Project Introduction

In the modern era, where economic instability and rapid lifestyle changes are common, managing personal finances has become a critical life skill. The ability to track income, monitor expenses, and set realistic financial goals plays a key role in achieving long-term financial stability. Despite the importance of these habits, many individuals still struggle to manage their finances effectively, often due to a lack of suitable tools or knowledge.

To address this issue, our team has developed a mobile application designed to support users in managing their personal finances more efficiently. Built using the Flutter framework, the application offers cross-platform compatibility and a smooth user experience across devices. The app integrates core functionalities such as transaction tracking, budget planning, and financial summary visualization through an intuitive interface. This project aims to contribute to the financial well-being of users by providing them with the tools and awareness necessary to take control of their personal finances.

## Problem Statement & Objectives

### 2.1 Objectives of the application

The main objectives of this application are:

* To provide a simple and user-friendly platform for tracking daily income and expenses.
* To enable users to create and manage budgets, helping them set spending limits and stay within them.
* To visualize financial data through charts and summaries, offering insights into spending habits.
* To raise users' awareness about the importance of personal financial discipline and long-term planning.
* To support financial goal-setting, encouraging users to save and spend more consciously.

### 2.2 Target Users

The application is designed for a broad range of users, including:

* Students learning to manage their personal finances independently.
* Young professionals seeking to monitor their income and spending habits.
* Individuals or families who want to plan budgets and save for future goals.
* Anyone looking for a simple and efficient tool to improve their financial awareness and discipline.

# CHAPTER II: REQUIREMENTS ANALYSIS

1. System Requirements

## Functional Requirements

The core functional requirements of the application include:

* **User Registration and Login:**
  + Users can securely create an account, log in, and log out.
* **Transaction Management:**
  + Add, edit, or delete income and expense transactions.
  + Assign categories to each transaction.
  + Include notes and select the transaction date.
* **Category Management:**
  + Create, update, and delete income or expense categories.
  + Clearly separate income and expense category types.
* **Budget Planning:**
  + Allow users to set monthly budget limits for each category.
  + Track expenses against the defined budget.
* **Transaction History:**
  + Display a list of transactions filtered by date, category, or type.
  + Support searching and filtering for easier tracking.
* **Dashboard Overview:**
  + Provide charts and summaries for income, expenses, and budget status.
  + Show real-time updates of financial activities.
* **Data Security and Storage:**
  + Ensure secure data storage via backend services.
  + Maintain personalized data per user account.

## Non-functional Requirements

* **Platform Support:** The application is developed specifically for Android devices using the Flutter framework. It is optimized for performance and usability on a wide range of Android smartphones and tablets.
* **Performance:** The app should respond to user interactions within 300 milliseconds to ensure a smooth and seamless experience.
* **Security:**
* User authentication must be implemented securely
* All communication between the client and the backend server should be protected (e.g., HTTPS).
* User data must be stored and handled in a way that ensures confidentiality and integrity.
* **Usability:** The user interface must be intuitive and user-friendly, allowing individuals with little or no technical background to use the app effectively.
* **Scalability:** While the app is initially intended for Android and personal use, the architecture should allow for future enhancements such as cloud data backup, multi-user support, push notifications, and expanded reporting features.

# CHAPTER III: TECHNOLOGY USED

1. Technologies Used
   1. Mobile Platform: **Flutter (Dart)**

## Backend / API / Database Services

* Spring Boot (Java): The backend RESTful API is developed using the Spring Boot framework. It provides a robust and scalable environment for handling user authentication, transaction processing, and data management.
* SQL Server: SQL Server is used as the relational database system to store persistent data such as:
* User accounts and login credentials
* Transactions (income/expense)
* Categories and budget plans
* Postman: Postman is used to test and document all API endpoints during development.

1. Development Tools & Frameworks

|  |  |
| --- | --- |
| **Tool/Framework** | **Purpose** |
| Android Studio | Main IDE for Flutter development on Android |
| Visual Studio Code (Or Apache Netbeans) | IDE for backend (Spring Boot) development |
| Apache Tomcat + ngrok | Server to deploy backend and temporally exposed online |
| Git & GitHub | Version control and collaborative coding |
| Postman | API testing and documentation |

1. Development Methodology

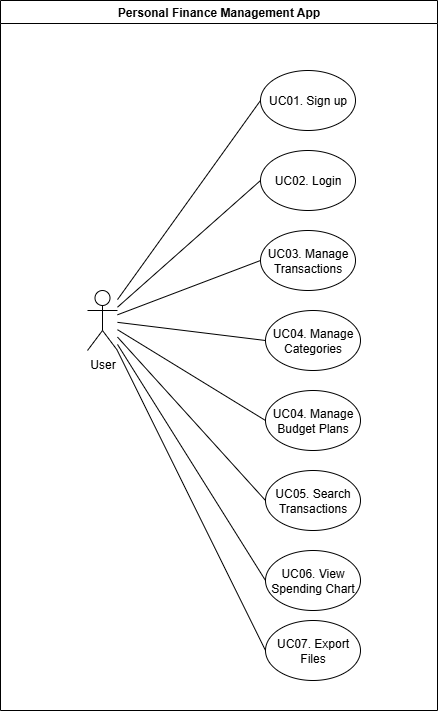
The team followed an Agile development methodology, allowing for iterative and incremental development through clearly defined weekly tasks and deliverables. The development process involved:

* **Requirement Analysis (Week 1):**  
  Identifying key features, target users, and technical feasibility.
* **UI/UX Design (Weeks 1–2):**  
  Creating wireframes and interface prototypes based on user flows.
* **Backend & API Development (Weeks 2–5):**  
  Building a secure and well-structured API to support data flow.
* **Frontend Integration (Weeks 3–6):**  
  Connecting UI screens with API logic, ensuring smooth user experience.
* **Testing & Bug Fixing (Week 6):**  
  Comprehensive testing across functions to ensure reliability.
* **Documentation (Weeks 7–8):**  
  Preparing technical and user documentation, finalizing the group report.

This structured approach ensured that the team could efficiently handle tasks, respond to feedback, and produce a stable, feature-complete application within the scheduled timeline.

# CHAPTER IV: SYSTEM DESIGN

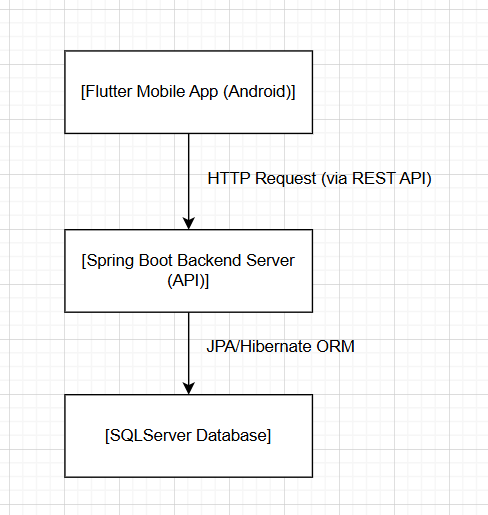
1. Use Case Diagrams



1. System Architecture

The system is designed using a client-server architecture with clear separation between the frontend and backend. It follows the MVC (Model–View–Controller) and RESTful service design principles.

**Architecture Overview:**



1. Database Design

|  |  |
| --- | --- |
| **Table Name** | **Description** |
| Users | Containing user’s account basic information |
| Categories | Containing categories of each user, each category belongs to only one user. Can be set to Expense or Income. |
| Transactions | Containing transactions of each user for a single category of them. |
| Budgets | Containing budget limit for each month of a user. Each category can only have 1 budget set for each month |
| Prefixes | Containing budget’s template created by user, can be apply for a selected month. |

**Entity Relationship Diagram (ERD):**

**A diagram of a computer

AI-generated content may be incorrect.**

**Database Diagram:**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. API Design

Budget API:

Budget mostly require simple API method that can be called via Repository – which has already got implemented by SpringBoot. For example, adding and updating method is fully supported by Repository with the “save” method, it simply requires the correct input from front-end, and SpringBoot will process them easily.

However, the difficulty come from showing the budget, where it requires a lot more than just those fields of the table. For example, total amount spent in that month require some further calculations that involve in other tables. Although Repository provide ability to create custom method, it doesn’t give us enough control over what we get and what it does.

This is where we need more control over what the API do, hence the appearance of DTO (Data Transfer Object) and DAO (Data Access Object). With DTO to control what data we send and DAO to control what the API do manually, we can achieve the expected result that we needed to send to front-end.

First, we need to have a way to get those fields that we need. Using Store Procedure created in SQL Server, we can achieve fields even from other tables easily. This also the fastest way, as any calculations is done on Server side. Here is the API we use for retrieving a budget list that also contains some extra fields like Category’s name and Total spent amount in that month:

A screenshot of a computer program

AI-generated content may be incorrect.

Store Procedure is executed in the following flow  
Create filter transaction to contain only those in the given month →

Select records that contain expected userID, Month and Year → Only select budget have category with type “Expense” → Apply transaction’s filter → Group up to calculate total Spent by summing transaction amount → Select all needed fields to show.

BudgetDTO class is created with the same fields receive from the SP, and DAO class containing a method that call the SP and map the received result to a list of BudgetDTO objects. Service and Controller class will then process and create a Get method that can be requested from front-end with the given userID, month and year parameters.

Prefix API:

In case you still don’t understand what prefixes do yet, it basically the template that created by the user for creating budget each month quickly. It is especially useful when the user must set ten budget that reset each month. Most of the prefix’s method can be done by Repository, leaving only applying them the main difficulty here.

We will still use the combination of DTO, DAO and Store Procedure to efficiently do this. One of the is calling add new budget API for every instance of prefix of the user. However, doing it this way will make the request take very long to finish, especially if the user have a lot of prefixes. So instead of do it with backend API logic, we do it with in Server side with a different approach. We will have a Store Procedure that apply all the user’s prefix to budget instead of calling adding budget API. This way, since the adding logic is done inside SQL Server, the response speed will be improved significantly. A screenshot of a computer program

AI-generated content may be incorrect.

The SP will do the following: Find all prefixes set by the user through userID, then compare it with Budgets table base on the condition: same categoryID, month and year. If the condition is satisfied, it count as matched and the budget’s amount will be update. If not matched, it will insert new budget.

The SP handle the adding logic even in edge case, where user already have the budget set, or when user have budget that not in their prefix list. With this, it take only a few seconds to finish applying all the prefixes.

# CHAPTER V: USER INTERFACE DESIGN

1. Design Process Overview

The user interface of FinGuard was designed following a user-centered approach. The process began with identifying core user tasks and mapping out typical usage scenarios. Based on these, wireframes were developed to visualize screen layouts and user interactions before implementation.

The design focused on the following goals:

* Simplicity: ensuring an intuitive interface for all users
* Consistency: applying a uniform color palette and font system
* Responsiveness: optimizing for various screen sizes
* Visual clarity: presenting financial data in a clean and informative way
* Low-fidelity wireframes were first created using Draw.io, then converted into high-fidelity, interactive screens using Flutter.

1. Wireframe Design

The wireframe system includes all main screens and demonstrates a complete user journey from authentication to transaction management, budget planning, and data export. Key wireframe components include:

* **Login & Register Screens**
  + Contain input fields for email and password.
  + “Sign in” and “Sign up” buttons are clearly distinguished.
* **Dashboard Screen**
  + Displays core balances: total income, total expense, and net balance.
  + Provides visual pie charts of expenses by category.
  + Features recent transactions and navigation buttons for History, Budget, Categories, and Add Transaction.
  + Offers data export options (CSV, Excel) and optional integration with an AI agent.
* **Add Transaction Screen**
  + Allows users to input transaction type (Income or Expense), amount, category, and date.
  + Includes validation for required fields.
* **Transaction History Screen**
  + Lists transactions grouped by date.
  + Includes search and filter functionality by type, category, and amount/date range.
* **Category Management Screen**
  + Displays existing categories with corresponding icons.
  + Allows adding new categories, with icon selection functionality.
* **Budget Planning Screen**
  + Enables users to set monthly budgets for specific categories.
  + Shows total budget, total spent, and remaining budget.
  + Budget items can be edited individually.

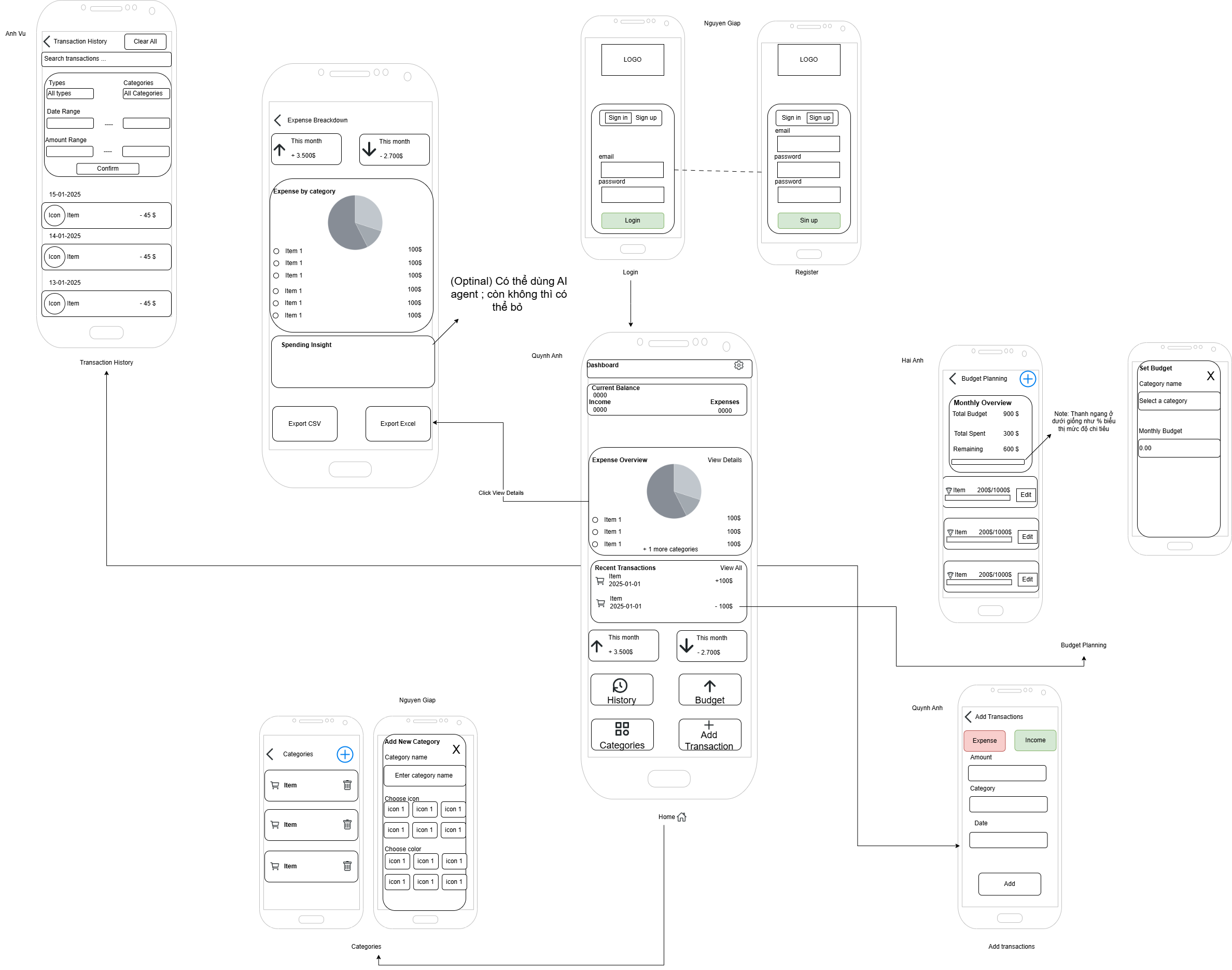
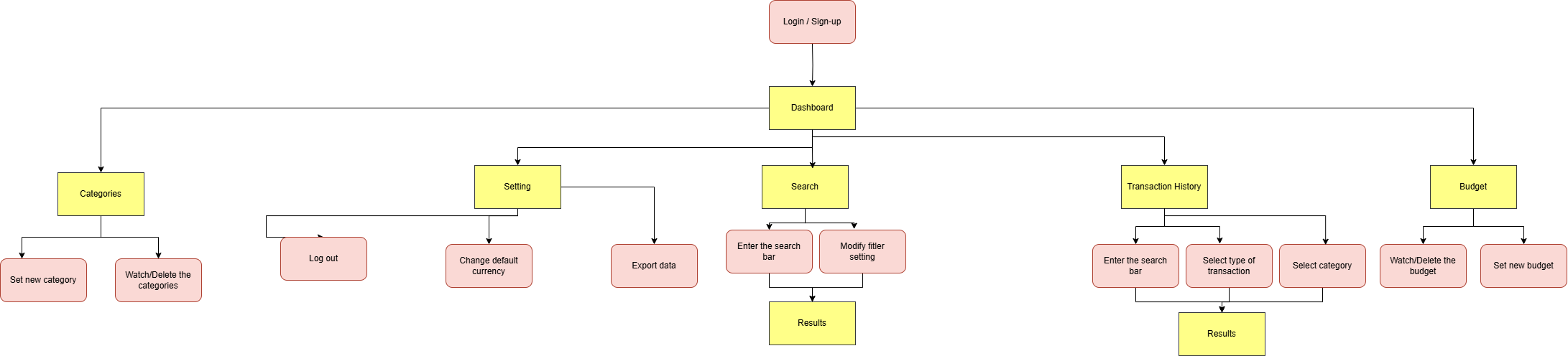


Figure 1: Wireframe System Overview

1. Sitemap

To visualize the navigation logic of FinGuard, a sitemap diagram has been created, illustrating the hierarchical structure of screens and actions. This sitemap replaces the traditional navigation flow diagram by presenting a clearer overview of how users traverse the application.



1. Final UI Design

A screenshot of a login form

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Login And Dashboard screens

A screenshot of a phone

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Transaction History and Add New Transaction screens

A screenshot of a phone

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Budget and Adding budget for month screens

A screenshot of a cell phone

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Prefix and Add New Prefix screens

A screenshot of a phone

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Category Management and Add New Category Modal screens

A screenshot of a tablet

AI-generated content may be incorrect. A screenshot of a phone

AI-generated content may be incorrect.

Detailed Expense screens and dashboard with dark theme toggle

**CHAPTER VI: IMPLEMENTATION**

FinGuard was implemented using Flutter, along with Android Studio, which is used for testing efficiently. The project follows a modular architecture separating models, views, and services controller (or MVC for short) for better maintainability and scalability. The frontend communicates with a backend REST API to retrieve and update data in real time. Key functionalities such as basic user login, transaction management, and data visualization were successfully implemented.

***1. Implemented Project Structure***

The general structure of the project are screens, model, services and global variable. Screens package – just like the name mentioned – containing all screens of the app. The screens can be navigated properly from another using Navigator.

Connection to backend is successfully implemented through service classes, which containing http method. The result receive are in JSON form, is then mapped to a List of object model, help visualize data can be done easily.

Till this point of the project, global variable package contains 2 important variables: ip\_address and currentTheme. Ip\_address is used to correctly defined url mapping to http method to call API. The current theme of the app is also saved in the variable, allow user to alternate between light and dark mode.

1. ***Used Library***

The core package we used for the project is http, which allowing API call method, letting actual data flow between frontend and backend. We also use month picker and intl for better month-based workflow. We also experiment with share\_preference package, which can be compared to a mini local storage on phone to save user’s information without using global variable.

1. ***Implemented Function***

|  |  |
| --- | --- |
| Function | Workflow |
| Log in – log out | Send POST request to backend, find a record having the same email and password. If success, return a user’s data (beside password) back. |
| Register | Send POST request to backend, check if email or username already exist. If success, add the sent data to Users table, navigate back to log in. |
| Dashboard overview (Summary) | Send multiple GET request, return data of recent transaction, balance information and expense summary. |
| Category Management | Send requests to get category list, allow user send request for changing, updating or adding new category of 2 different type: Expense or Income. |
| Transaction Management | Send request to get all transaction history of the user, allow deleting and creating new transaction for a chosen category. |
| Budget Management | Send request to get all budgets of the chosen month, allow editing, inserting and deleting. Also having Prefix as an addition function, allow creating template and adding multiple budgets at once. |
| Theme changing | Theme class is created inside project, allow user to toggle between light and dark mode. |

# CHAPTER VII: LIMITATIONS AND FUTURE WORK

# PART VIII: CONCLUSION