System Documentation Individual Report

for

Personal Budget Tracker

Version <3.0>

Tutorial Section: **<TT1L>**

Group No.: **< Group 1>**

<CHIN ZHEN HO> **<1221102540>**

Date: **<12.2.2025>**

Contents

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
SRS in Part 1(as Ver 1.0) SDS in Part 2(as Ver 2.0.X) *System Documentation in Part 3 (as Ver 3.0) Draft Type and Number	CHIN ZHEN HO	Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded.	12/02/25

1 System Overview

1.1 Description

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, will be effective.

TO DO: Describe the major processes to be performed by the system and the the actors involved in each process.>

This web application is designed to help users worldwide track and manage their personal finances, aiming to improve their financial situations. The primary functions of the system are structured around various processes that involve different actors: users, admins, financial advisors, and guests.

Users of the system can create an account, log in, and manage their financial data. They can input and categorize their income, expenses, and savings, track their spending, and set financial goals. The system provides tools for users to create budgets and monitor their progress towards these goals. Additionally, users can access a financial history to observe spending trends and adjust their budgeting strategies accordingly. For more personalized support, users can also seek guidance from financial advisors who can review their data and offer tailored advice.

Admins have full control over the system and all user accounts. They can create, modify, or delete user profiles and oversee the overall system's functionality. Admins are responsible for managing the integrity of financial data and ensuring the proper functioning of the system, including generating reports on user activity and system performance. In addition to their oversight roles, admins have the capability to monitor budget progress and provide universal support to users.

Financial advisors have access to user data to provide personalized financial advice. They can assist users in setting up budgets, suggest better financial practices, and guide them toward achieving their financial goals. Advisors can view user spending patterns, offer recommendations, and help users optimize their financial decisions.

Guests, who do not have full accounts, are granted limited access to the system. They can explore basic features such as inputting financial data and tracking budgets but cannot save their entries or access advanced features. This allows potential users to get a feel for the application before deciding to sign up for full access.

The interactions between these actors and processes can be effectively represented through a data flow diagram or object class diagram, illustrating how each role contributes to the overall functionality of the web application.

1.2 Use Cases

<TO DO: List the actor and use cases required in a table as shown below.>

Actor	Use Cases
User	View transaction history(edit transaction history function)
Admin	Login as admin
	Manage Categories
	Financial Category Statistics
	Analysis of "Others" Category Descriptions
	View Users Details

1.3 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.

TO DO: Provide a short list of some major assumptions that might significantly affect your design. For example, you can assume that your client will have 1, 2 or at most 50 Automated Banking Machines. Every number has a significant effect on the design of your system. >

Assumptions:

User Technical Proficiency:

It assumes that users have basic technical literacy, meaning they can navigate web applications, input financial data, and interpret visualizations like charts and graphs without extensive technical support or training.

Data Volume and Storage:

It assumes that the platform is designed to scale efficiently and can support thousands of users, ensuring that performance does not degrade as the user base grows and data accumulates.

Development Resources:

It assumes that the development team has access to the necessary tools, frameworks, and libraries required to implement the planned features, including those related to security, user interfaces, and data management.

Dependencies:

Third-Party APIs:

The application will rely on third-party APIs for optional features, such as currency conversion and financial data syncing. This means the functionality of these features is dependent on the availability, stability, and reliability of the third-party services.

Database and Hosting:

The application depends on a scalable and reliable database solution to store and retrieve user data. The choice of database and hosting infrastructure is critical to ensuring smooth performance, especially as the volume of user data grows.

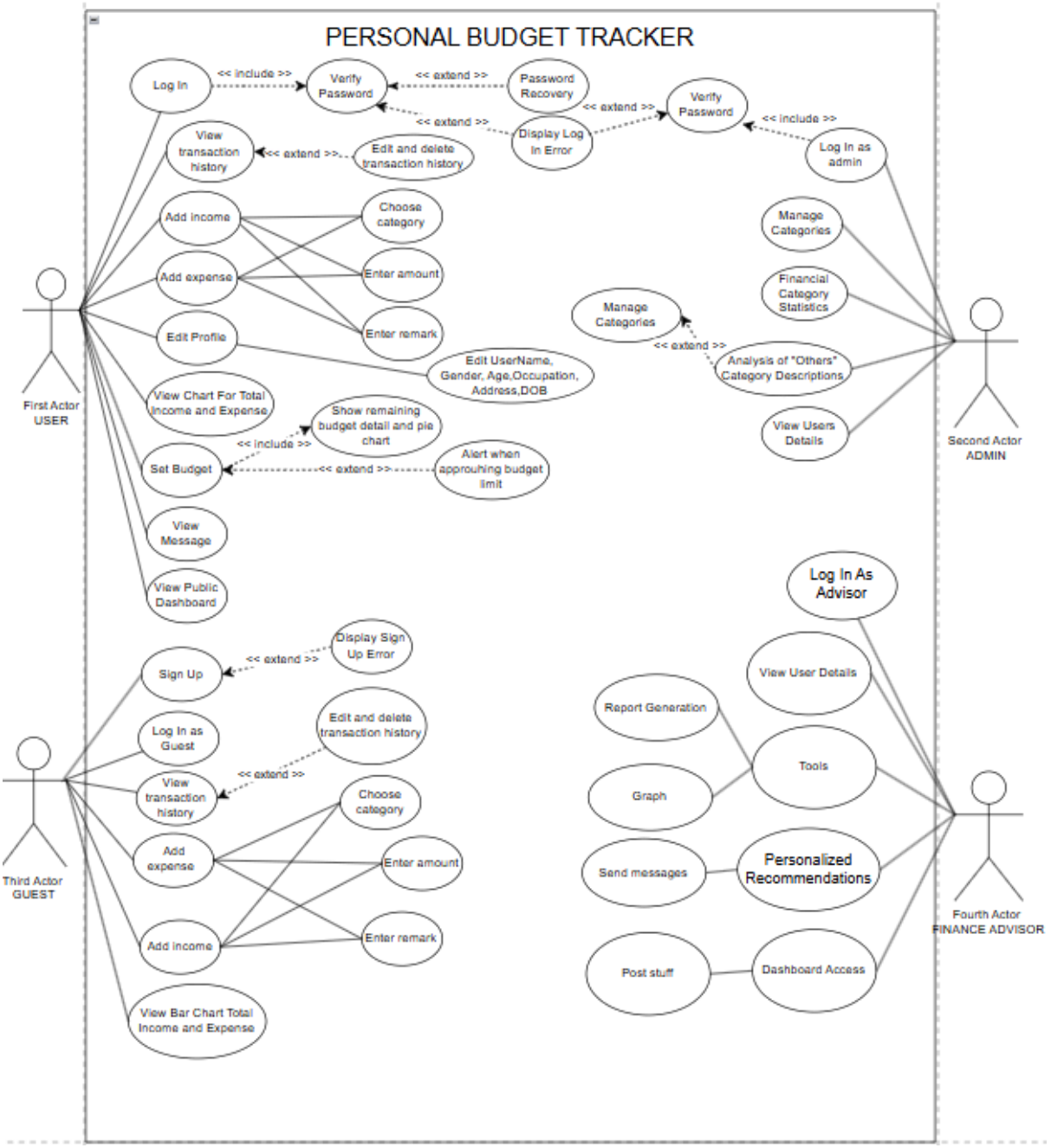
Browser Compatibility:

The system depends on modern browser standards and ongoing support for critical web APIs. This assumption ensures the application can function across various browsers, but it also relies on continuous updates and support for these standards.

2 Requirements

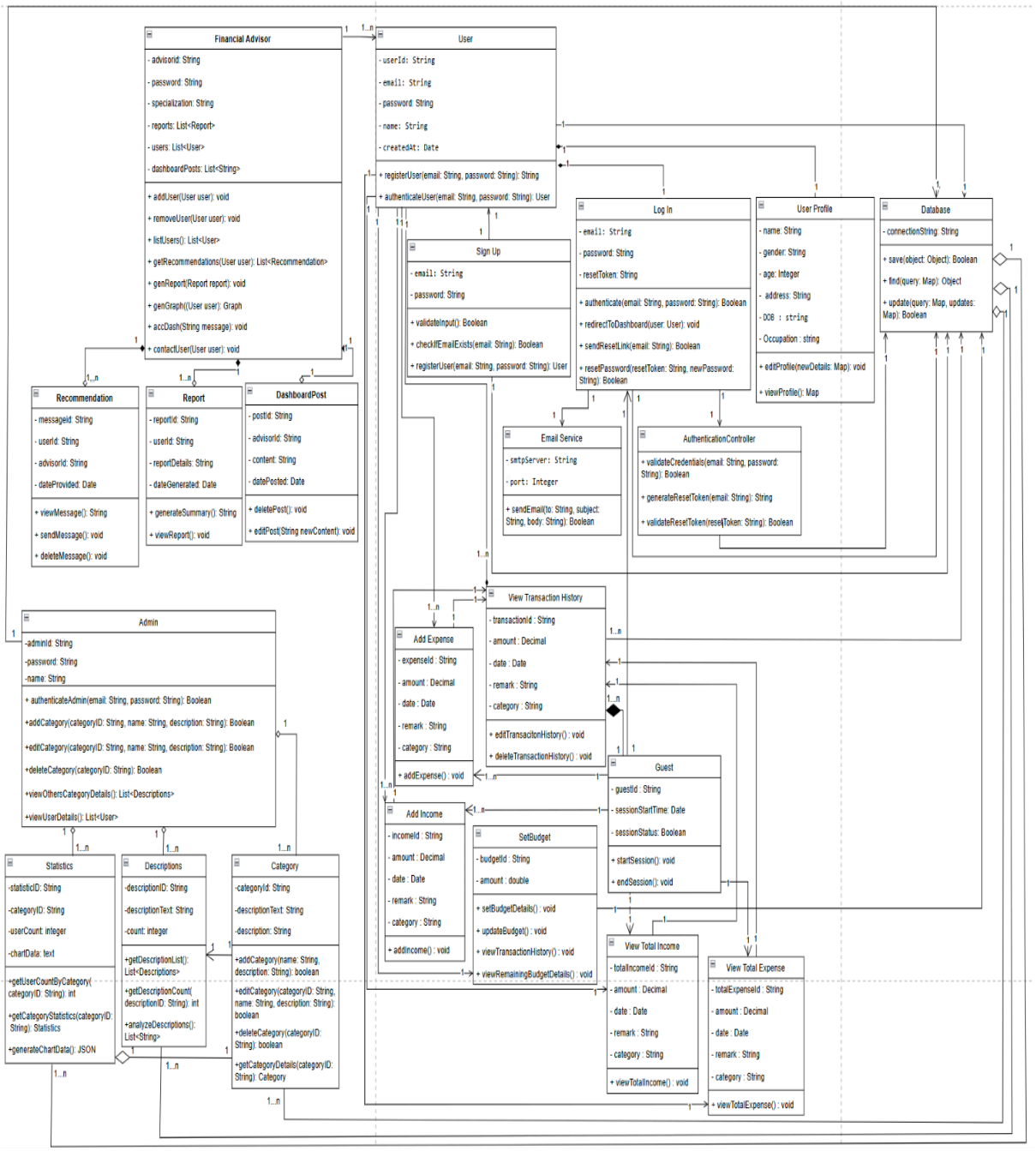
2.1 Use Case Diagram

<TO DO: Place the actor's use case diagram here.>



2.2 Class Diagrams / ERD

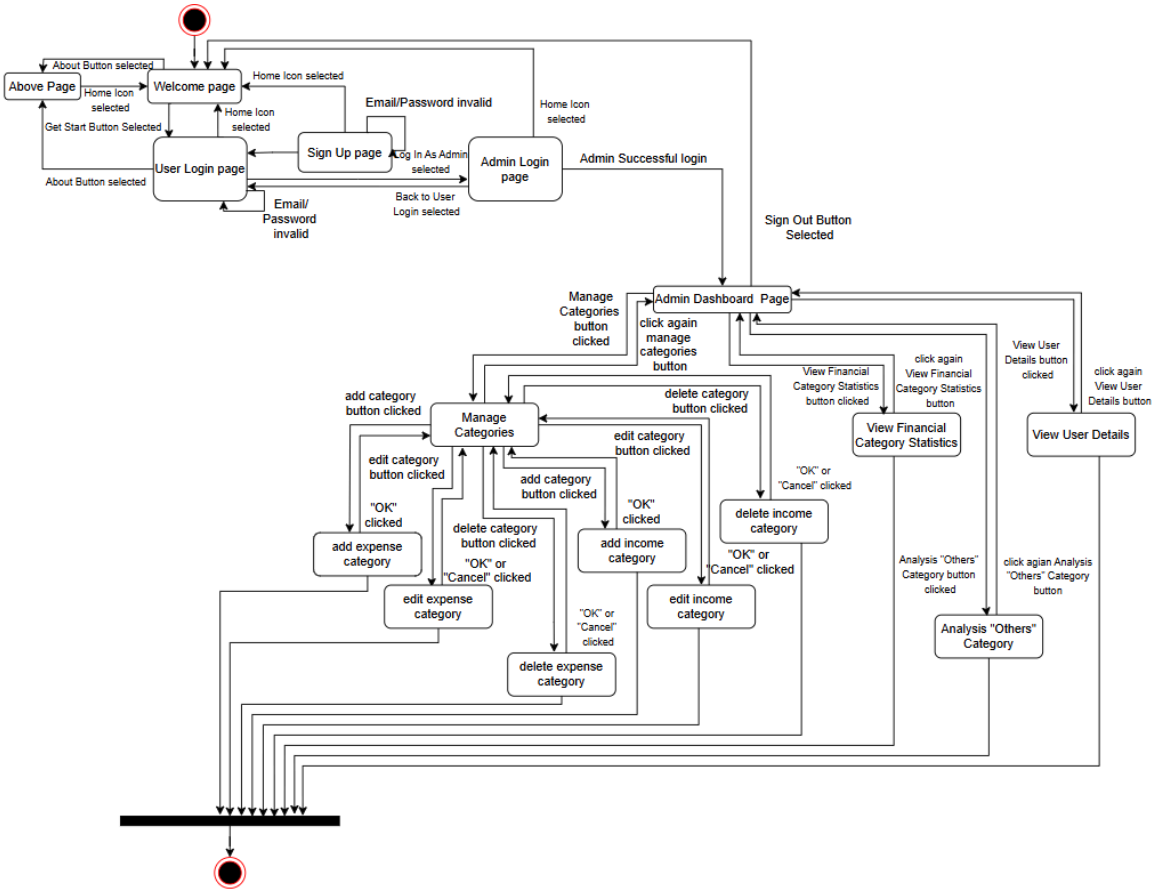
<TO DO: Place the class diagram / ER diagram here.>



2.3 State Diagrams

<TO DO: Place only the actor's state diagram here.>

Admin State Diagram illustrates the flow of actions and states an administrator can navigate through in the application. Starting from the Welcome Page, the admin logs in via the Admin Login Page and, upon successful authentication, accesses the Admin Dashboard Page. From here, the admin can manage categories by adding, editing, or deleting income and expense categories, view financial category statistics , analyze "Others" Category Description for reviewing uncategorized data, and access user details. The admin can log out or return to the dashboard at any time, providing a streamlined and efficient process for managing application functionalities.



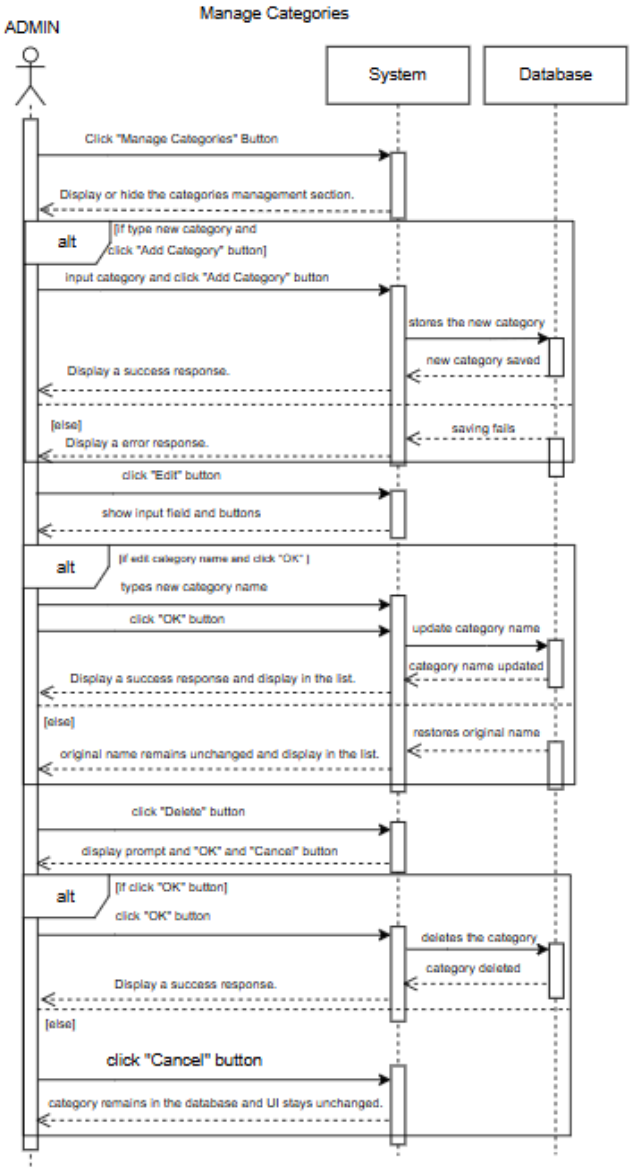
3 Design

3.1 Use Cases

3.1.1 Actor Admin

3.1.1.1 Manage Categories

<TO DO: Describe the use case and place the sequence diagram.>

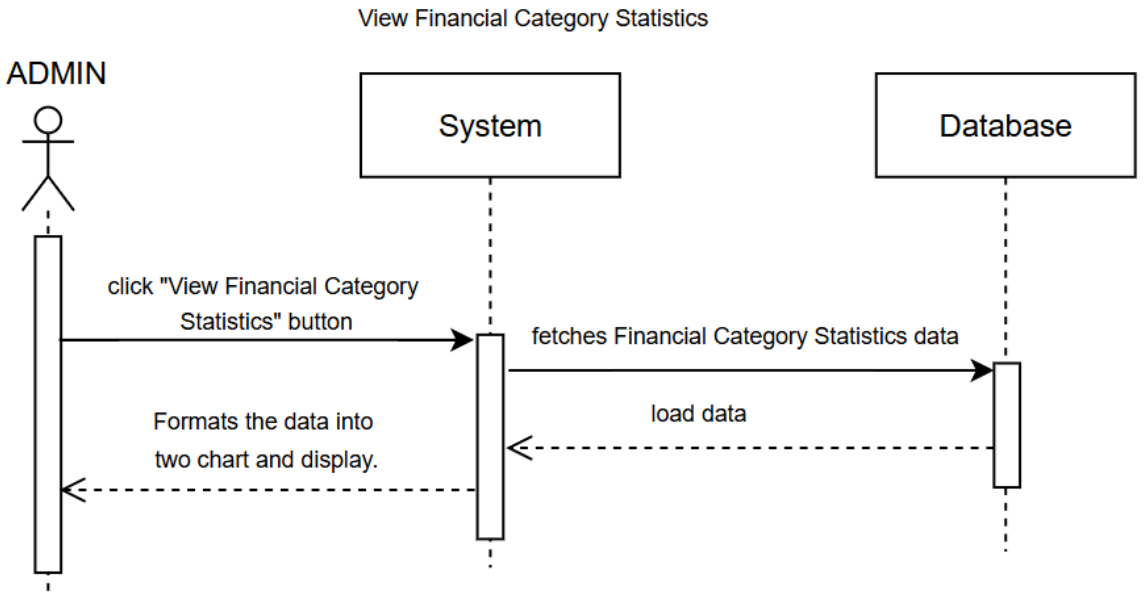


Use Case Name:	Manage Categories
Description:	The "Manage Categories" use case enables the admin to efficiently manage income and expense categories in the system. The admin can add, edit, or delete categories. The system validates input, communicates with the database, and updates the category list accordingly.
Primary Actor:	Admin
Precondition:	1. The admin has access to the "Manage Categories" button and category management section. 2. The database connection is established and functioning.
Postcondition:	1. The category list is updated based on the admin's actions. 2. Any additions, edits, deletions are successfully stored in the database.
Main Success Scenario:	1. The admin clicks the "Manage Categories" button to access the category management section. 2. To add a category, the admin enters a new category name and clicks the "Add Category" button. 3. The system validates the input and sends a request to the database to store the category. 4. If successful, the system updates the category list and displays a success message. 5. To edit a category, the admin clicks the "Edit" button, modifies the name, and clicks "OK." 6. The system updates the category name in the database, refreshes the list, and displays a success message. 7. To delete a category, the admin clicks the "Delete" button and confirms the action. 8. The system removes the category from the database, updates the list, and displays a success message.
Alternative Scenario:	1. Error Adding Category: ● The admin enters an invalid category name or the database fails to store the category. ● The system displays an error message indicating the issue. 2. Error Editing Category: ● The admin clicks "Edit" but enters invalid input or the database fails to update the category.

	● The system displays an error message and restores the original name. 3. Error Deleting Category: ● The admin clicks "Delete," but the database fails to remove the category. ● The system displays an error message, and the category remains unchanged. 4. Cancel Actions: ● The admin cancels any edit or delete action. ● The system makes no changes, and the UI remains unchanged.

3.1.1.2 View Financial Category Statistics

<TO DO: Describe the use case and place the sequence diagram.>

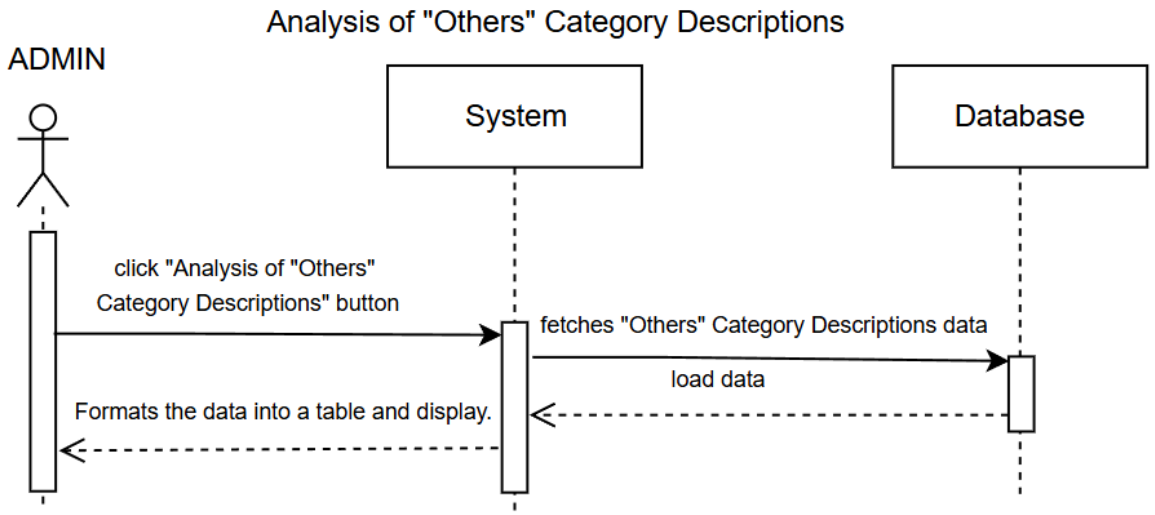


Use Case Name:	View Financial Category Statistics
Description:	The "View Financial Category Statistics" use case allows the admin to view detailed statistics about user interactions with income and expense categories. The system retrieves real-time data from the database and visualizes it as two charts,one for expense categories and one for income categories. These insights help the admin understand user behavior and trends for improving the system's usability and optimizing category offerings.
Primary Actor:	Admin
Precondition:	1. The "View Financial Category Statistics" button is accessible in the system. 2. The database contains financial category usage data.
Postcondition:	1. The system displays two charts with the financial category statistics. 2. If there is no data, an appropriate message is displayed.
Main Success Scenario:	1. The admin clicks the "View Financial Category Statistics" button.

	2. The system sends a request to the database to retrieve financial category statistics data. 3. The database processes the request and returns the data. 4. The system formats the data into two charts: ● An Expense Chart shows the number of users selecting each expense category. ● An Income Chart shows the number of users selecting each income category. 5. The system displays the charts to the admin.
Alternative Scenario:	1. No Data Available: ● The database contains no financial category statistics. ● The system displays empty charts with a message indicating "No available data." 2. Database Connection Error: ● The system fails to connect to the database or retrieve data. ● An error message is displayed to the admin, indicating the issue.

3.1.1.3 Analysis of “Others” Category Description

<TO DO: Describe the use case and place the sequence diagram.>

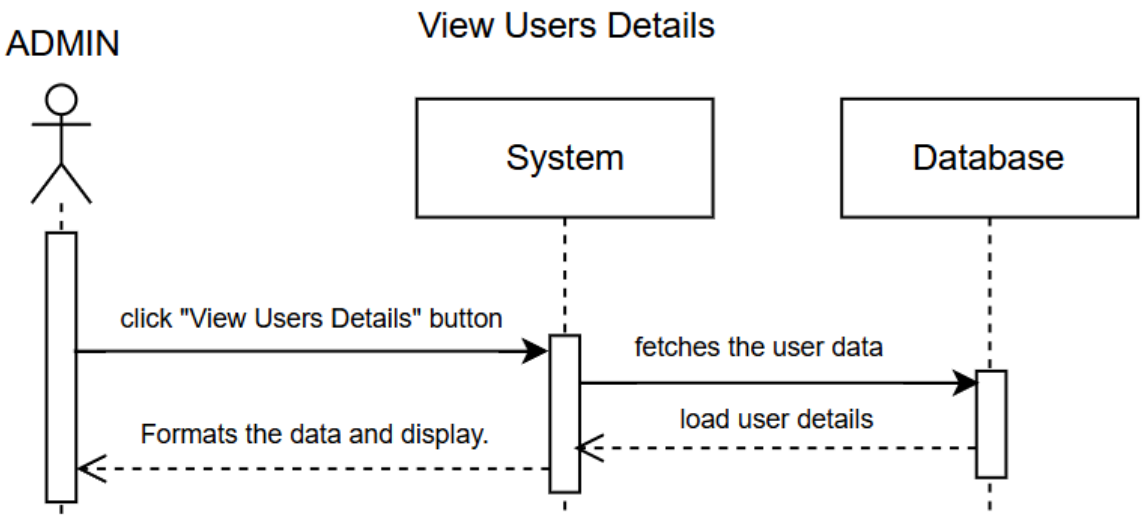


Use Case Name:	Analysis of "Others" Category Descriptions
Description:	This use case allows the admin to analyze user-defined descriptions for the "Others" category and identify potential new categories to add. The system retrieves and processes data about user interactions with the "Others" category, including the frequency of each user-provided description. The data is formatted into a table and displayed to help the admin make informed decisions about optimizing the category structure.
Primary Actor:	Admin
Precondition:	1. The "Analysis of 'Others' Category Descriptions" button is accessible in the system. 2. The database contains data on user-provided descriptions for the "Others" category.
Postcondition:	1. A table displaying user-provided descriptions and their frequencies is shown to the admin. 2. The admin gains insights into trends and potential new categories.
Main Success Scenario:	1. The admin clicks the "Analysis of 'Others' Category Descriptions" button. 2. The system sends a request to the database to retrieve data about the "Others" category. 3. The database processes the request and returns

	the relevant data, including a list of user-provided descriptions and their frequencies. 4. The system processes and formats the data into a table. 5. The table is displayed below the button for the admin to analyze.
Alternative Scenario:	1. No Data Available: ● The database contains no data on the "Others" category. ● The system displays a message indicating "No data available." 2. Database Connection Error: ● The system fails to connect to the database or retrieve data. ● An error message is displayed to the admin, indicating the issue.

3.1.1.4 View Users Details

<TO DO: Describe the use case and place the sequence diagram.>



Use Case Name:	View Users Details
Description:	The "View Users Details" use case allows the admin to access detailed information about all users in the system. The system retrieves and displays user data, such as email addresses, name, age, date of birth, gender, occupation, and address in a clear table format. This helps the admin monitor user activity, manage user accounts, and gain insights into system usage patterns.
Primary Actor:	Admin
Precondition:	1. The "View Users Details" button is accessible in the system. 2. The database contains user data, including email addresses, name, age, date of birth, gender, occupation, and address.
Postcondition:	1. The system displays a table containing user data for the admin to review. 2. The admin gains insights into user activity and system usage.
Main Success Scenario:	1. The admin clicks the "View Users Details" button. 2. The system sends a request to the database to fetch user data, including email addresses, name, age, date of birth, gender, occupation, and address.

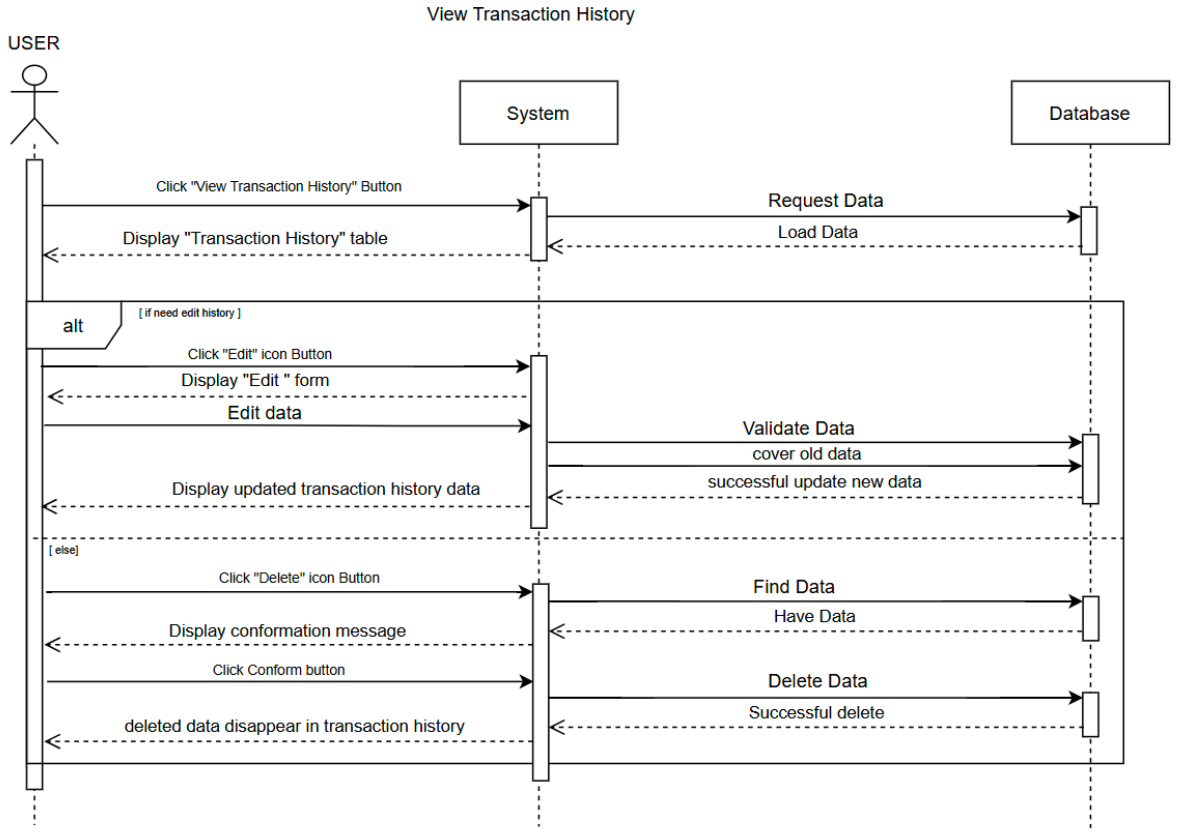
	3. The database processes the request and returns the user data to the system. 4. The system processes and formats the retrieved data into a table. 5. The table is displayed below the button for the admin to review.
Alternative Scenario:	1. No User Data Available: ● The database contains no user data. ● The system displays a message indicating "No user data available." 2. Database Connection Error: ● The system fails to connect to the database or retrieve data. ● An error message is displayed to the admin, indicating the issue.

3.1.2 Additional Task

3.1.2.1 Actor 1 User

3.1.2.1.1 View Transaction History(Edit Transaction History function)

<TO DO: Describe the use case and place the sequence diagram.>



Use Case Name:	View and Manage Transaction History
Description:	This use case allows users to view their transaction history in a tabular format. It also provides functionality to edit or delete individual transactions. Edited data is validated and updated in the database, while deleted transactions are permanently removed upon user confirmation.
Primary Actor:	User
Precondition:	1. The user must be logged into their account. 2. Transaction history must exist in the database.
Postcondition:	1. Transaction data is displayed to the user in a table format.

	2. Updated transaction data is reflected in the table after successful editing. 3. Deleted transactions are removed from the table and database.
Main Success Scenario:	1. The user clicks the "View Transaction History" button. 2. The system retrieves transaction data from the database and displays it in a table. 3. If the user clicks the "Edit" icon button: ● The system displays an edit form for the selected transaction. ● The user edits the data and submits it. ● The system validates the data and updates the transaction in the database. ● The updated transaction history is displayed. 4. If the user clicks the "Delete" icon button: ● The system displays a confirmation message. ● If the user confirms the deletion, the transaction is removed from the database. ● The system updates the table to reflect the deleted transaction.
Alternative Scenario:	1. Invalid Data During Editing: ● The user submits invalid data while editing a transaction (e.g., incorrect format or missing fields). ● The system displays an error message, prompting the user to correct the data. 2. Transaction Not Found (During Deletion): ● The system is unable to locate the transaction in the database (e.g., due to deletion by another process). ● The system displays an error message indicating the transaction could not be deleted. 3. User Cancels Deletion: ● The user clicks the "Cancel" button on the confirmation dialog. ● The system aborts the deletion process, and the transaction history remains unchanged.

	4. No Transactions Found: ● If no transaction data is available for the user, the system displays a message indicating no transactions are available.

3.2 Data Dictionary

<TO DO: Describe the data dictionary and place the table with the details here>

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
email	string	50	-	Yes	Not Null	Admin's email address
password	string	50	-	Yes	Not Null	Admin's password

3.2.1 Admin

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
categoryId	string	30	PK	Yes	Not Null	Unique ID for each category
name	string	50	-	Yes	Not Null	Name of the category
type	string	10	-	Yes	Not Null	Specifies if the category is income or expense

3.2.2 Category

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
description	string	255	-	Yes	Not Null	User-defined description for the "Others" category
expensesCount	int	-	-	Yes	Not Null	Total number of times this description was used in expenses
incomesCount	int	-	-	Yes	Not Null	Total number of times this description was used in incomes

3.2.3 Others Category Analysis

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
expenseStatistics	Map<String, Int>	-	-	Yes	Not Null	Stores expense categories and their usage counts
incomeStatistics	Map<String, Int>	-	-	Yes	Not Null	Stores income categories and their usage counts

3.2.4 Financial Statistics

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
categoryId	string	30	PK	Yes	Not Null	Unique identifier for the category
userId	string	30	FK	Yes	Not Null	Foreign key referencing the User table
amount	int	-	-	Yes	Not Null	Amount associated with the category
date	date	-	-	Yes	Not Null	Date the category is used
remark	string	255	-	Yes	Null	Remark provided for the category

3.2.5 Income and Expense Categories

3.2.1.1Additional Task

Field Name	Data Type	Length	PK/FK	Required	Null / Not Null	Description
transactionId	string	30	PK	Yes	Not Null	Transaction history unique ID

userId	string	30	FK	Yes	Not Null	User's unique ID
ExpenseId	string	30	FK	Yes	Not Null	Expense's unique ID
IncomeId	string	30	FK	Yes	Not Null	Income's unique ID
amount	int	10	-	Yes	Not Null	Amount that user enter for Expense
date	Date	10	-	Yes	Not Null	Date that user select to save the Expense
remark	string	30	-	Yes	Not Null	Remark that user enter for expense
category	string	30	-	Yes	Not Null	Category that user choose for expense

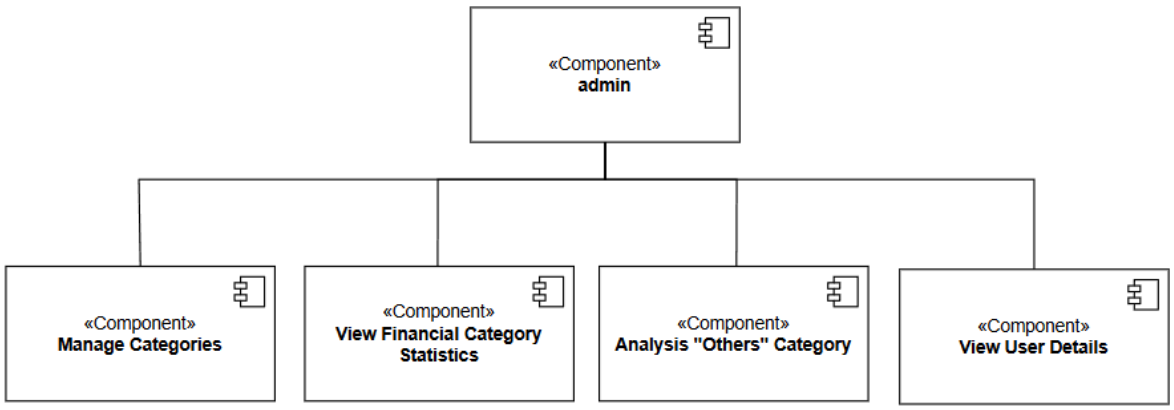
3.2.1.1.1 View Transaction History(Edit Transaction History function)

3.3 Subsystem Architecture

<TO DO: Describe the subsystem and place the architecture diagram here.>

3.3.1 Admin Subsystem

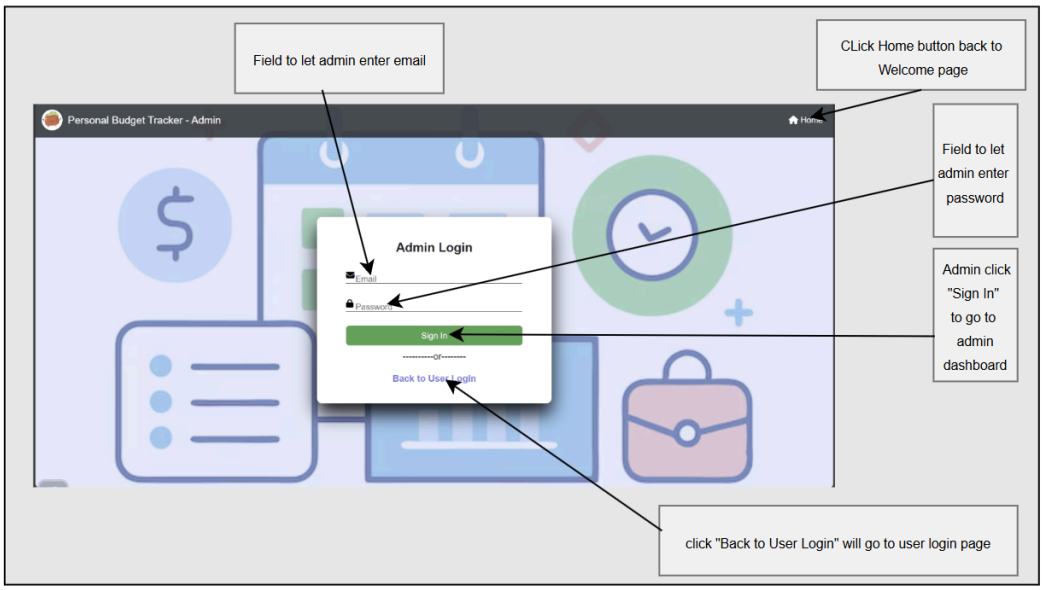
The Admin Subsystem consists of four main functions, which are Manage Categories, View Financial Category Statistics, Analysis of "Others" Category, and View User Details. The Manage Categories function allows administrators to create, edit, or delete financial categories to ensure proper organization. The View Financial Category Statistics function provides insights into categorized data, helping to monitor trends and financial performance. The Analysis of "Others" Category function focuses on reviewing uncategorized data. Lastly, the View User Details function enables administrators to access and manage user profiles, monitor activities, and assist users as needed.



3.4 Subsystem Screens

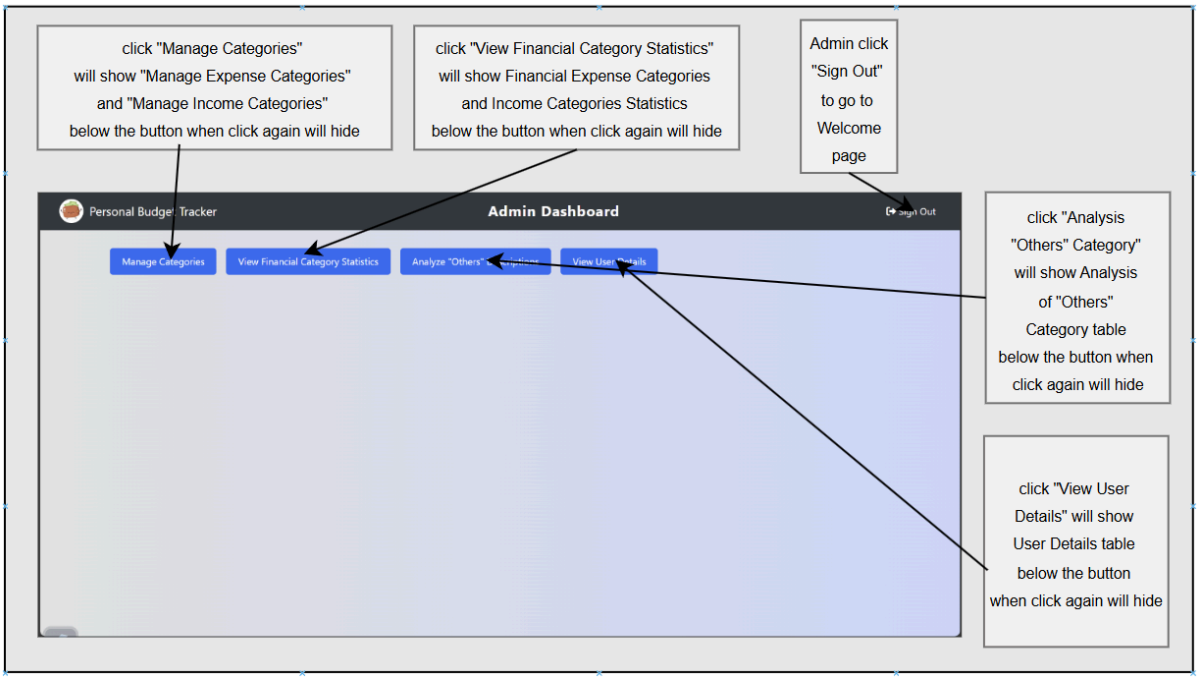
<TO DO: Describe the screens of subsystem 1 and place the screen designs here.>

3.4.1 Admin Login Page



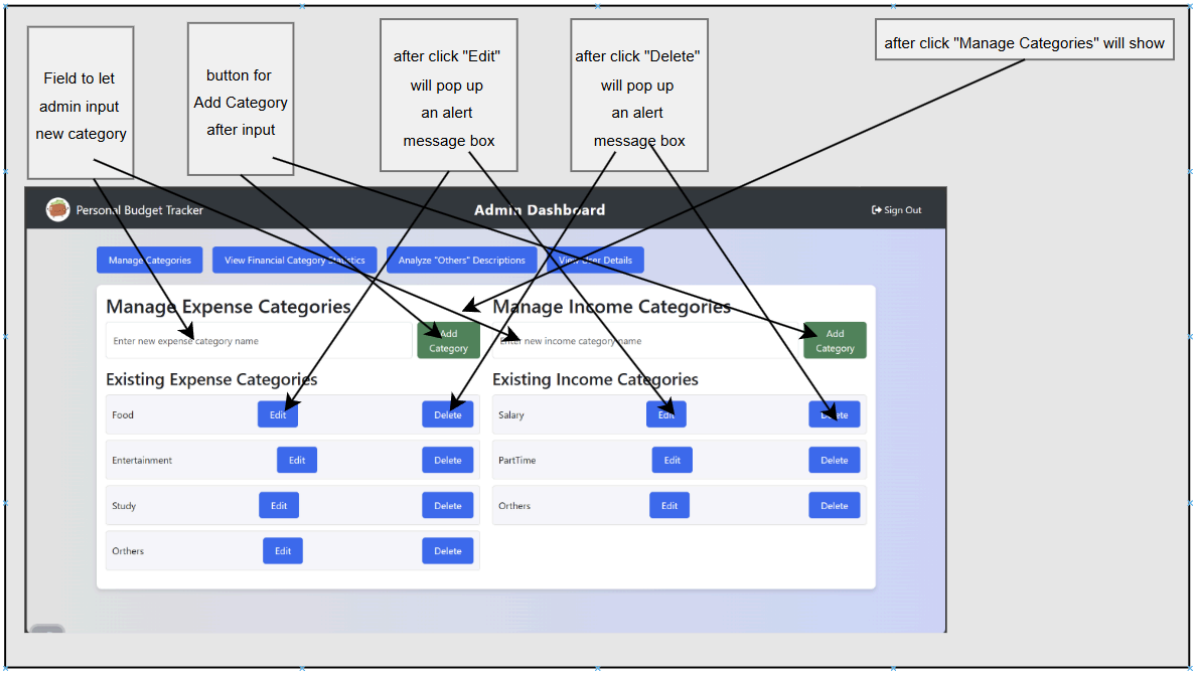
The **Admin Login Page** of the Personal Budget Tracker is a secure interface designed for administrators to access the system. It features a clear and simple design with an input field for the admin's email, marked with an envelope icon, and a password field, marked with a lock icon, to ensure security. A green **Sign In** button allows administrators to authenticate their credentials and proceed to the Admin Dashboard. For additional navigation, a **Back to User Login** link is provided, enabling users to return to the user login page. Additionally, a **Home** button in the top-right corner allows users to return to the main welcome page.

3.4.2 Admin Main Page

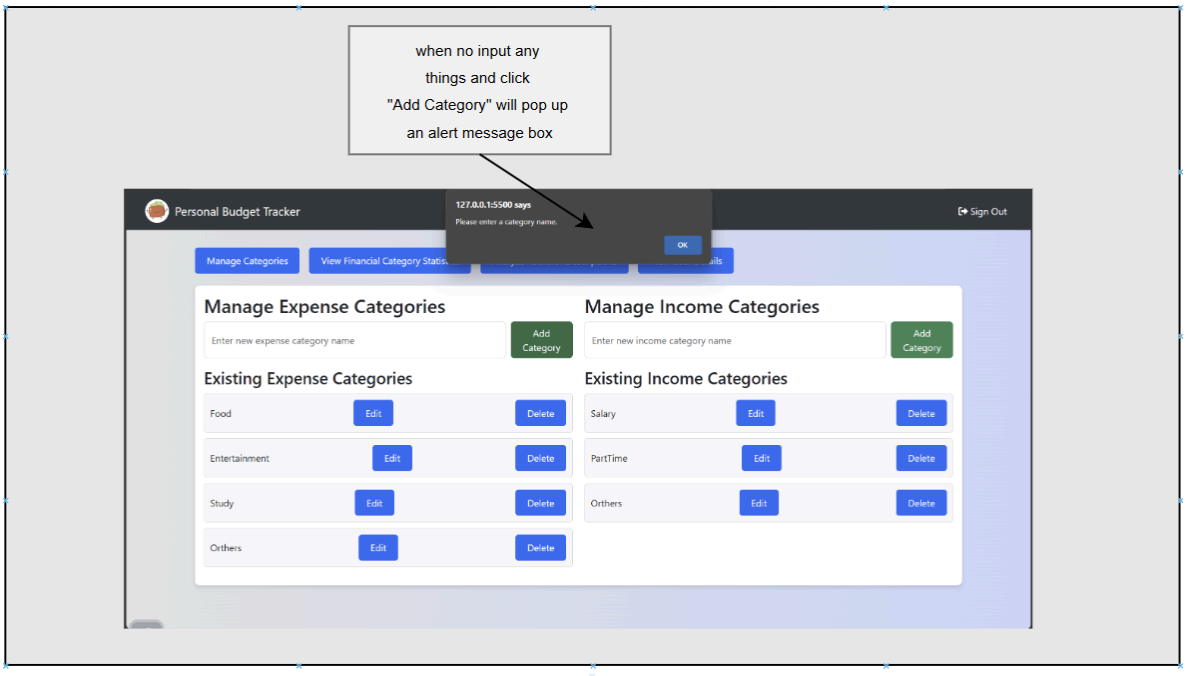


The **Admin Dashboard** of the Personal Budget Tracker provides essential tools for managing and analyzing financial data. It includes buttons like **Manage Categories** to handle expense and income categories, **View Financial Category Statistics** to display income and expense statistics, and **Analyze "Others" Category Description** for reviewing uncategorized data. Additionally, the **View User Details** button shows user information in a table format. All these buttons feature toggle functionality to show or hide their respective details. A **Sign Out** button in the top-right corner allows secure logout. The dashboard's clean layout ensures easy navigation and efficient management.

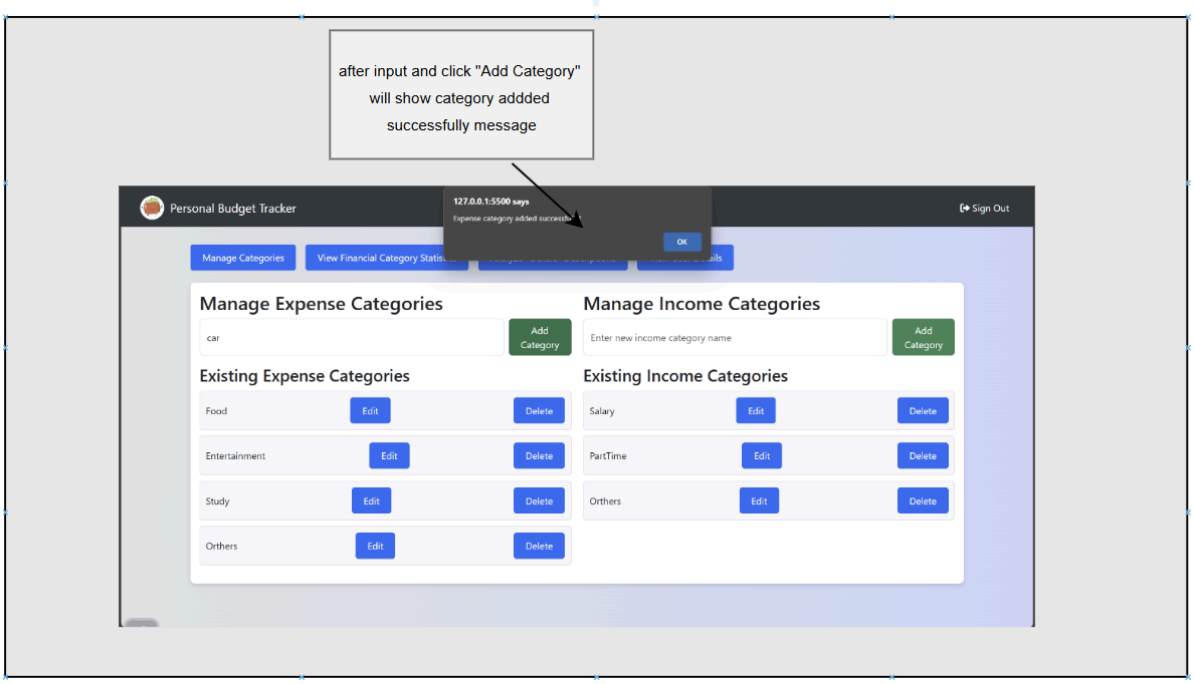
3.4.3 Manage Categories Page



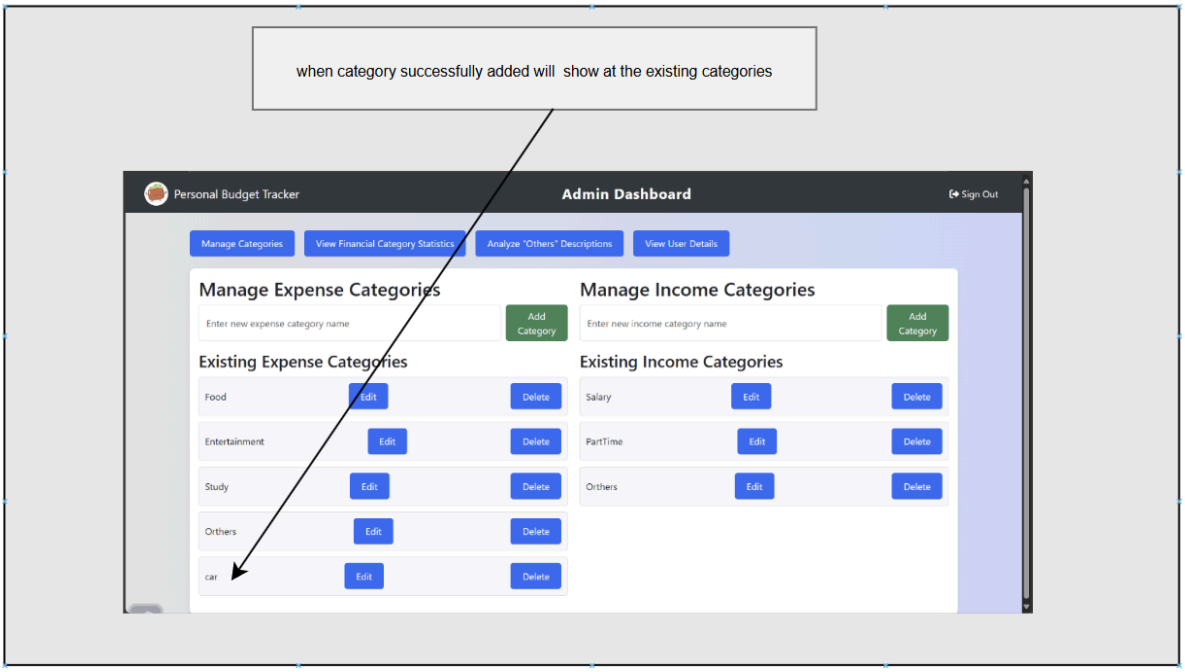
The **Manage Categories** section of the Admin Dashboard allows administrators to manage both expense and income categories. It includes input fields for adding new categories, accompanied by a green **Add Category** button to save the entries. Existing categories are listed with **Edit** and **Delete** buttons for modification or removal. Clicking the **Edit** or **Delete** buttons triggers an alert message for confirmation. This section streamlines the process of organizing financial data, ensuring that categories for income and expenses can be easily customized and maintained. The interface is simple and user-friendly, enhancing efficiency for administrators.



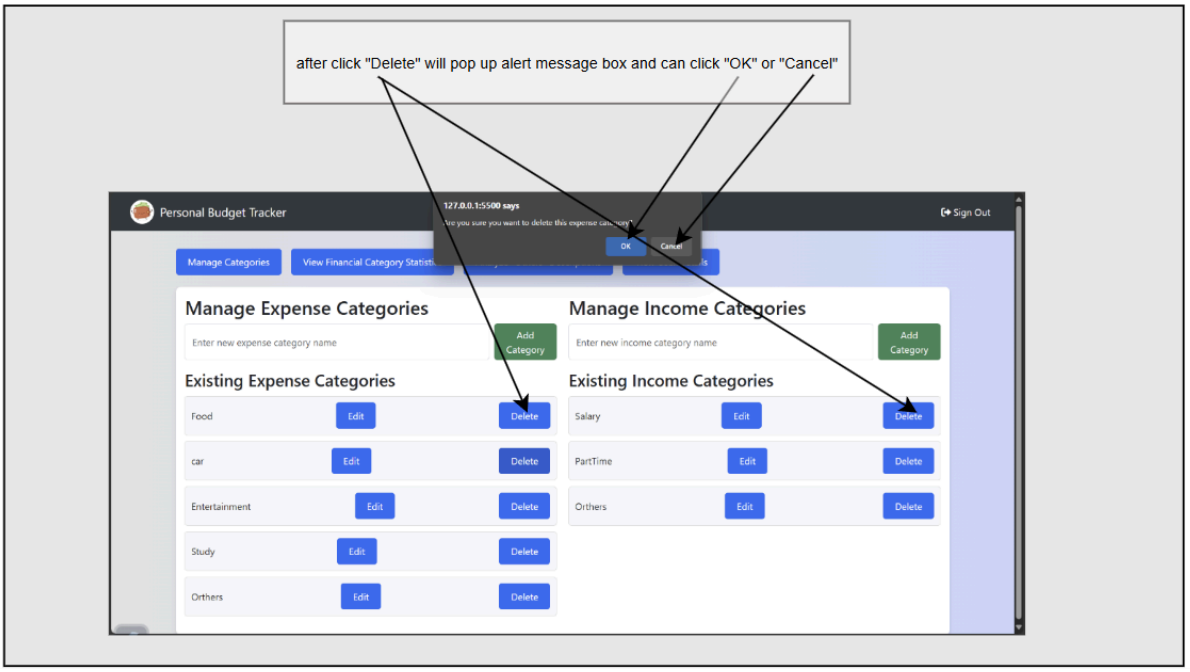
The **Manage Categories** section includes a validation feature for adding new categories. If the **Add Category** button is clicked without entering any input in the category name field, an alert message box appears with a prompt stating, "Please enter a category name." This ensures that users cannot add empty or invalid entries, maintaining the integrity of the category data.



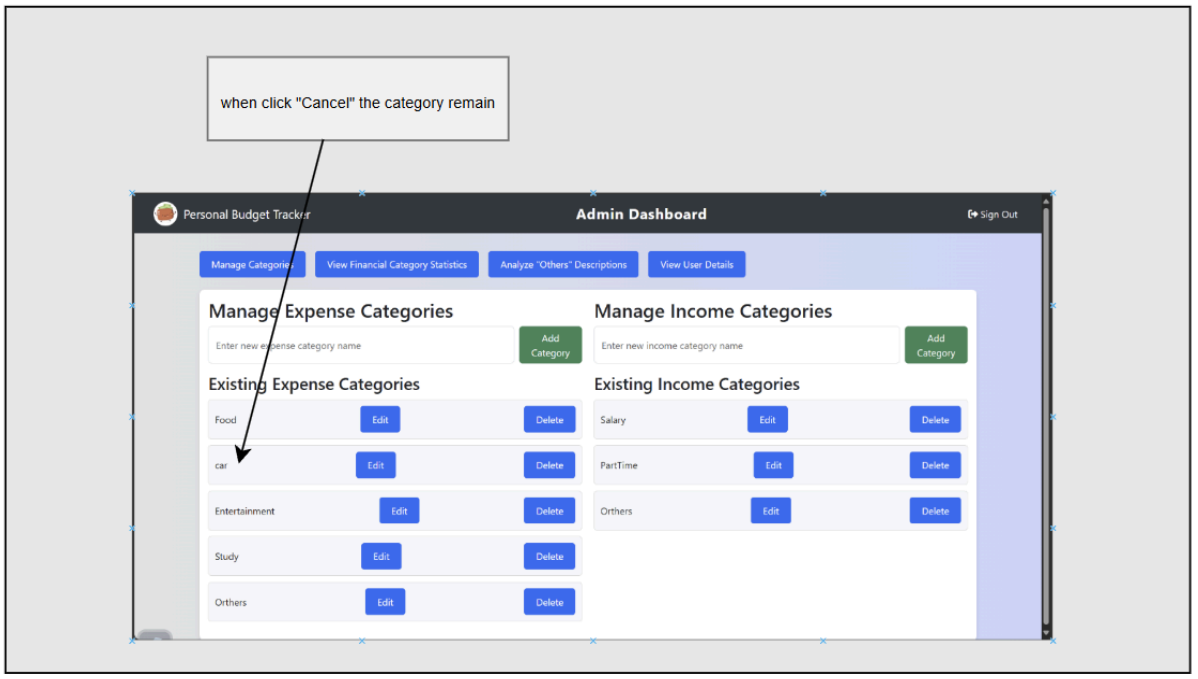
When a new category is successfully added in the **Manage Categories** section, an alert message box confirms the action with a message like "Expense category added successfully."



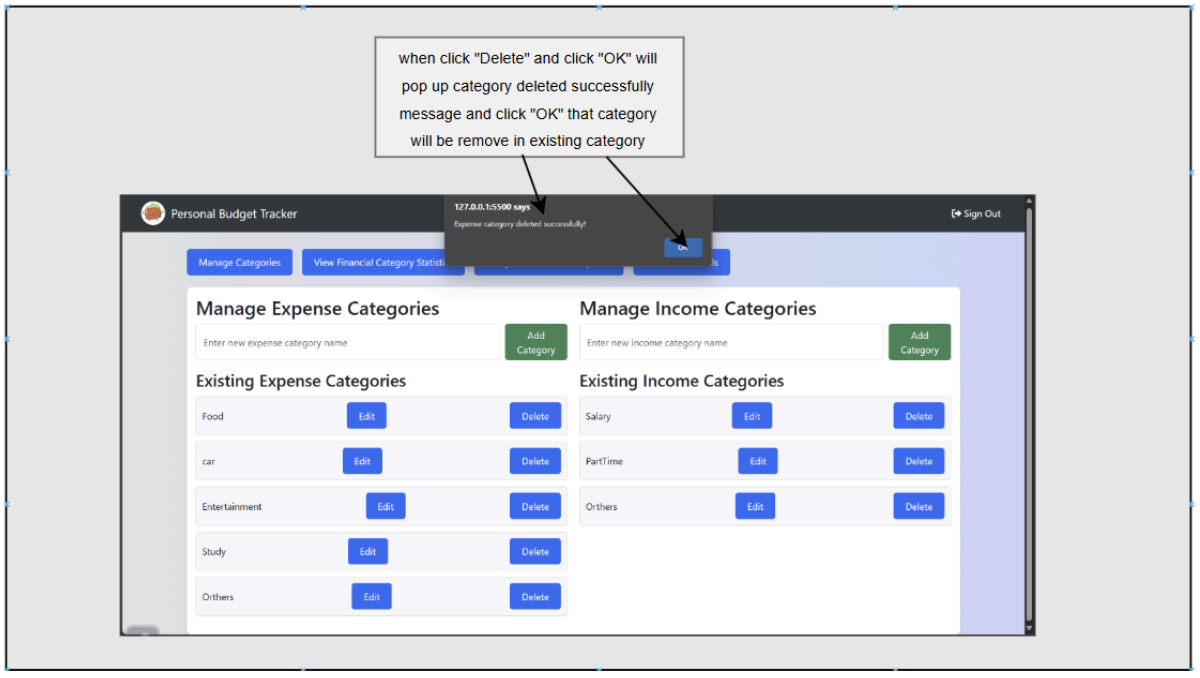
The newly added category is then displayed under the **Existing Categories** list, along with options to **Edit** or **Delete** it. This ensures that the admin can immediately see and manage the added category, maintaining a smooth workflow and clear organization of expense and income categories.



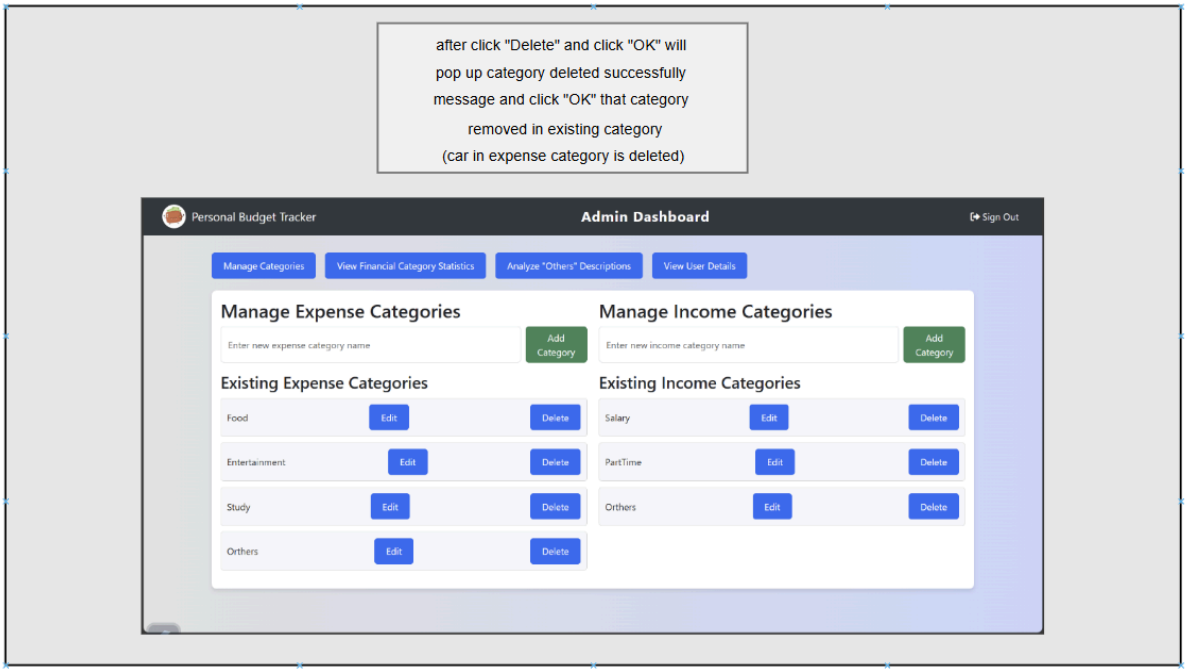
When the **Delete** button is clicked in the **Manage Categories** section, a confirmation alert message box appears, asking the admin to confirm the deletion by selecting either "OK" or "Cancel."



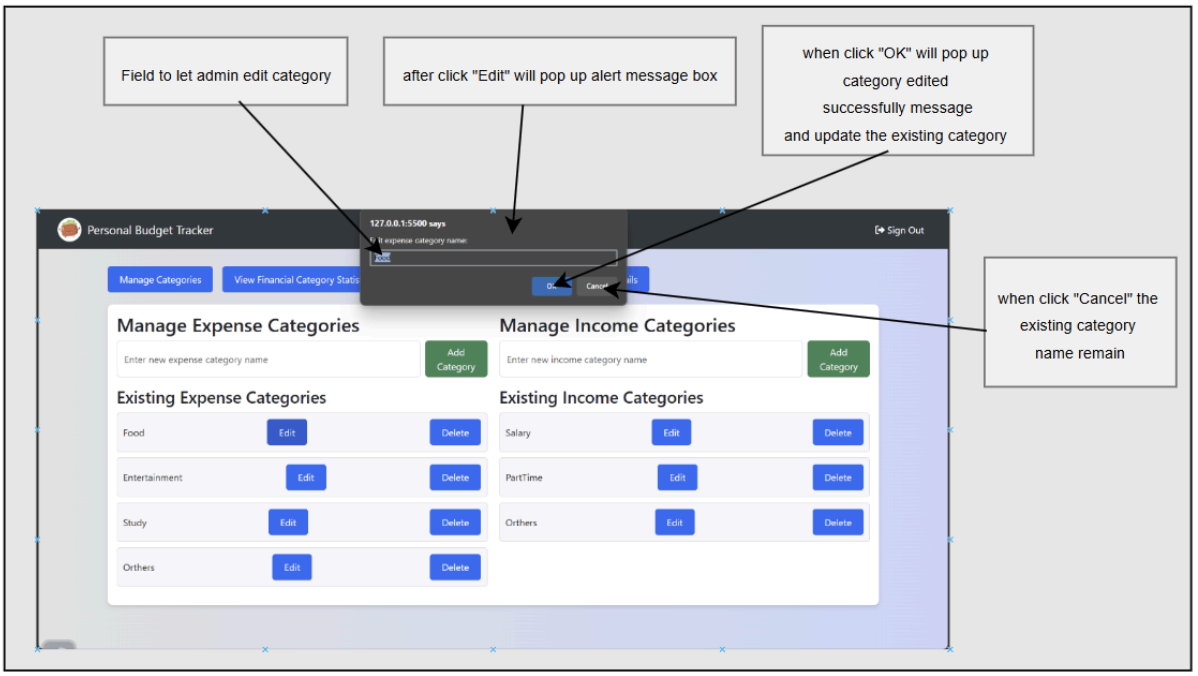
If "Cancel" is selected, the category remains unchanged.



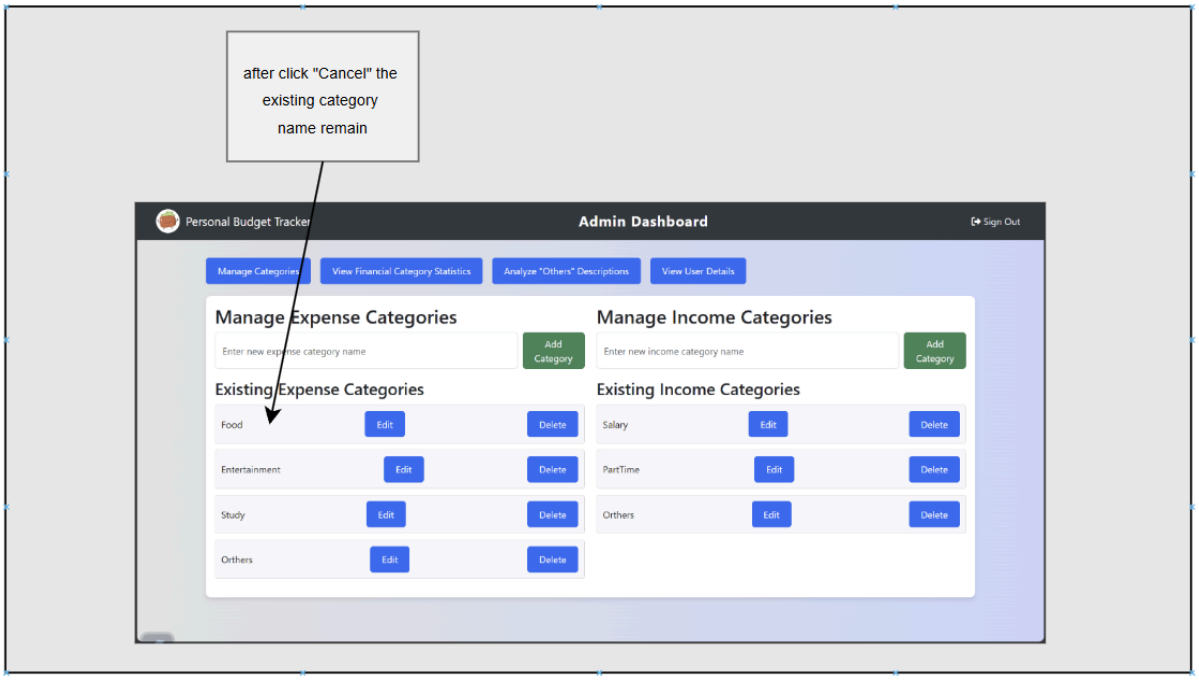
If "OK" is clicked, a success message confirms that the category has been deleted.



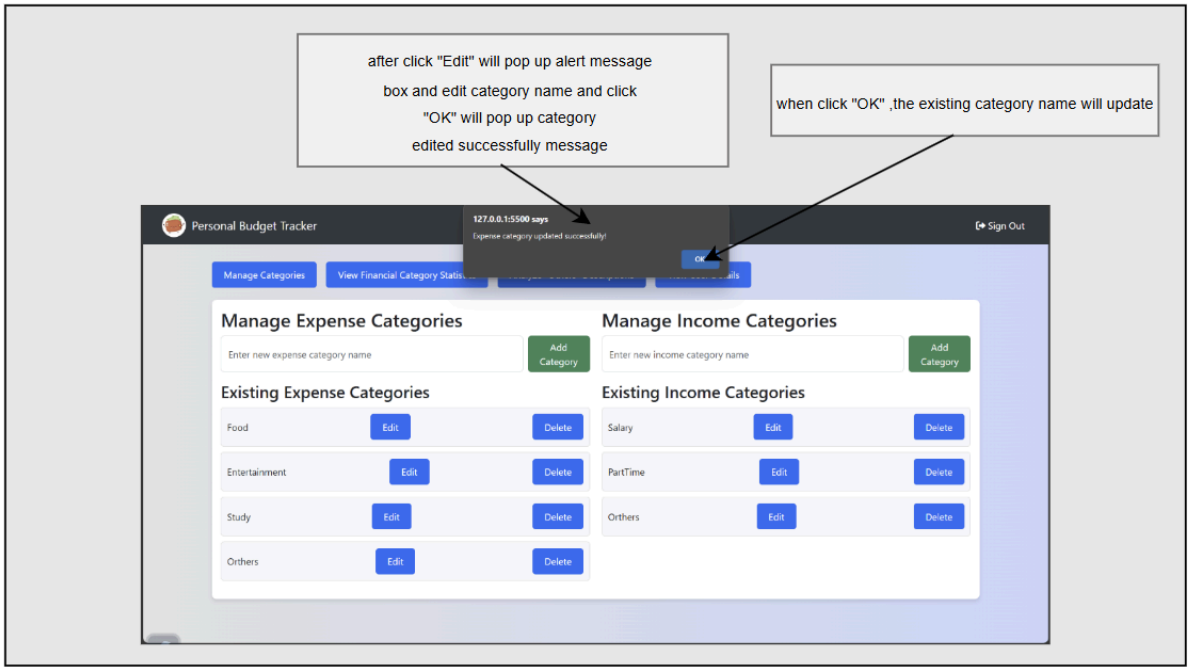
The selected category is removed from the **Existing Categories** list. This functionality ensures that administrators have control over category deletion while minimizing accidental changes through confirmation prompts.



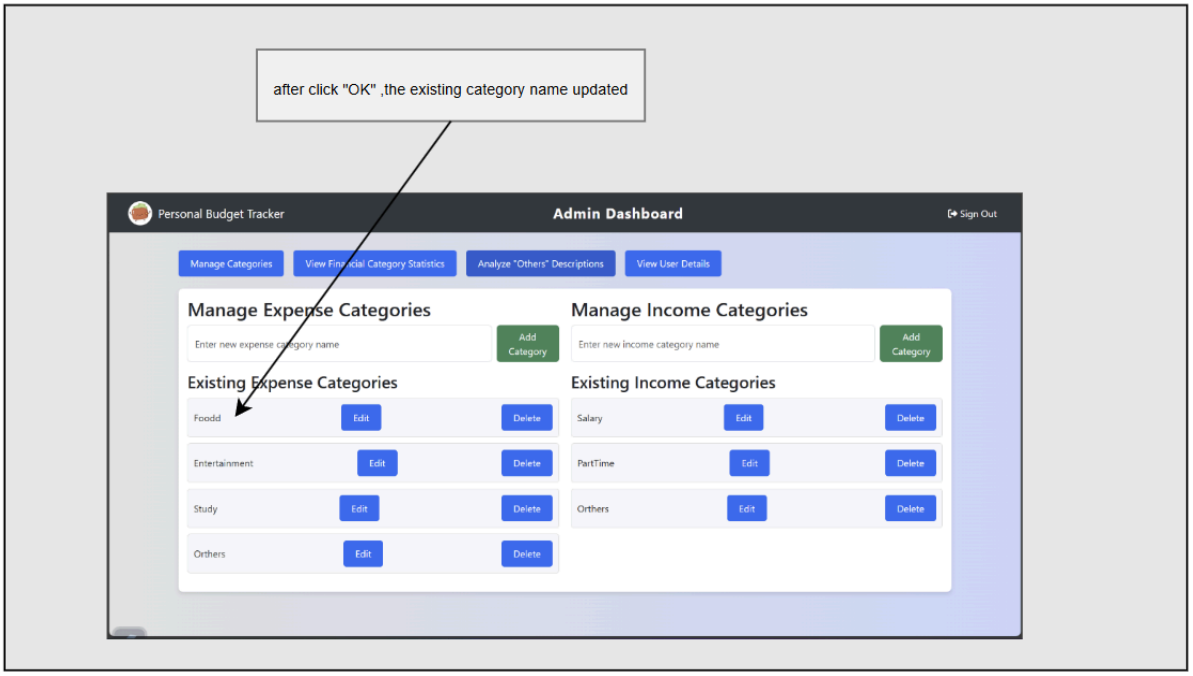
In the **Edit Category** process, clicking the **Edit** button opens an alert message box that allows the admin to input a new name for the selected category.



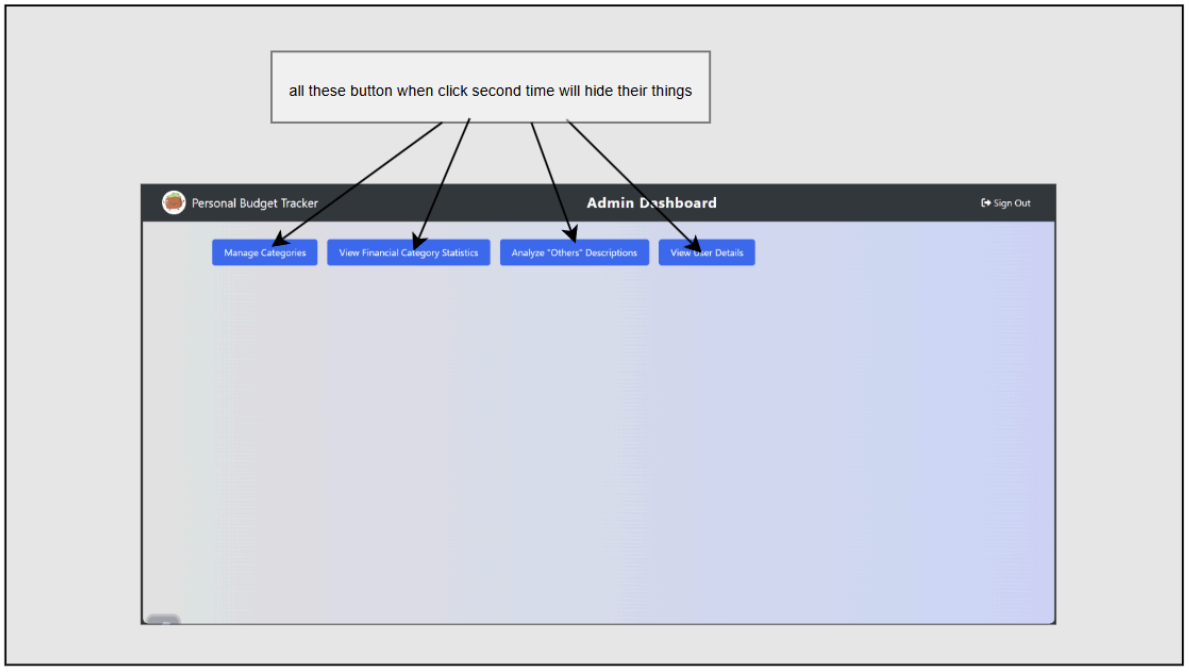
If the admin clicks **Cancel**, the category remains unchanged.



If the admin clicks **OK**, a success message confirms the update.

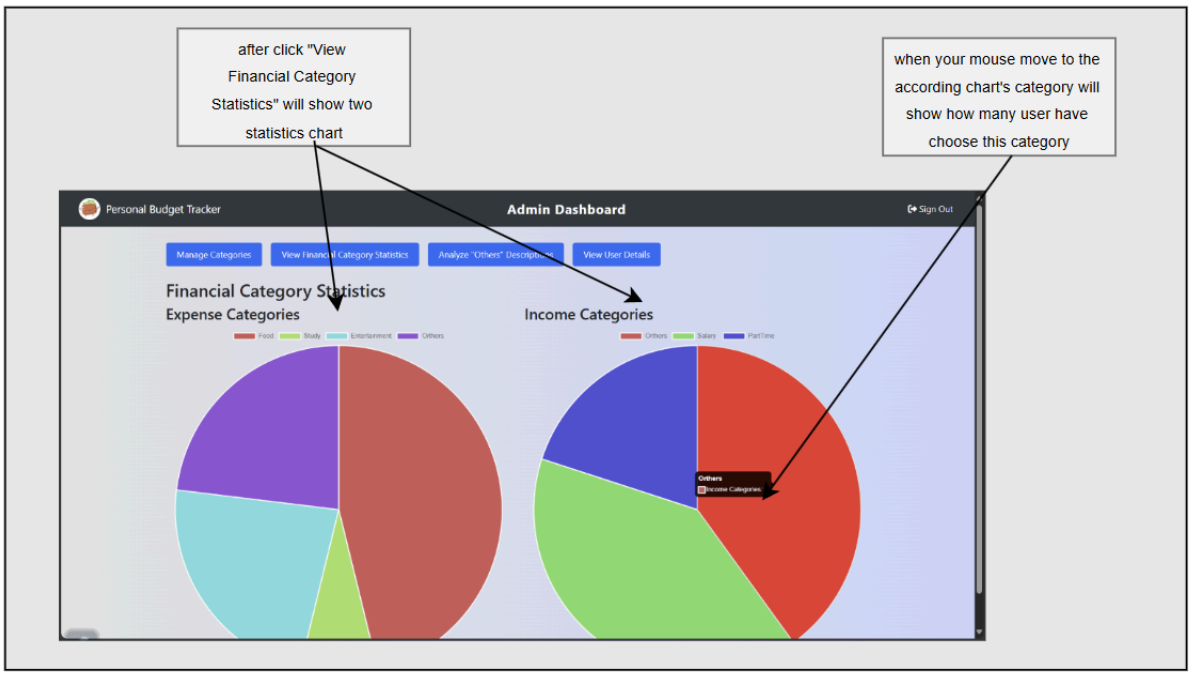


The category name is immediately updated in the **Existing Categories** list.This feature ensures flexibility and control in managing category names while providing clear confirmation for every action.



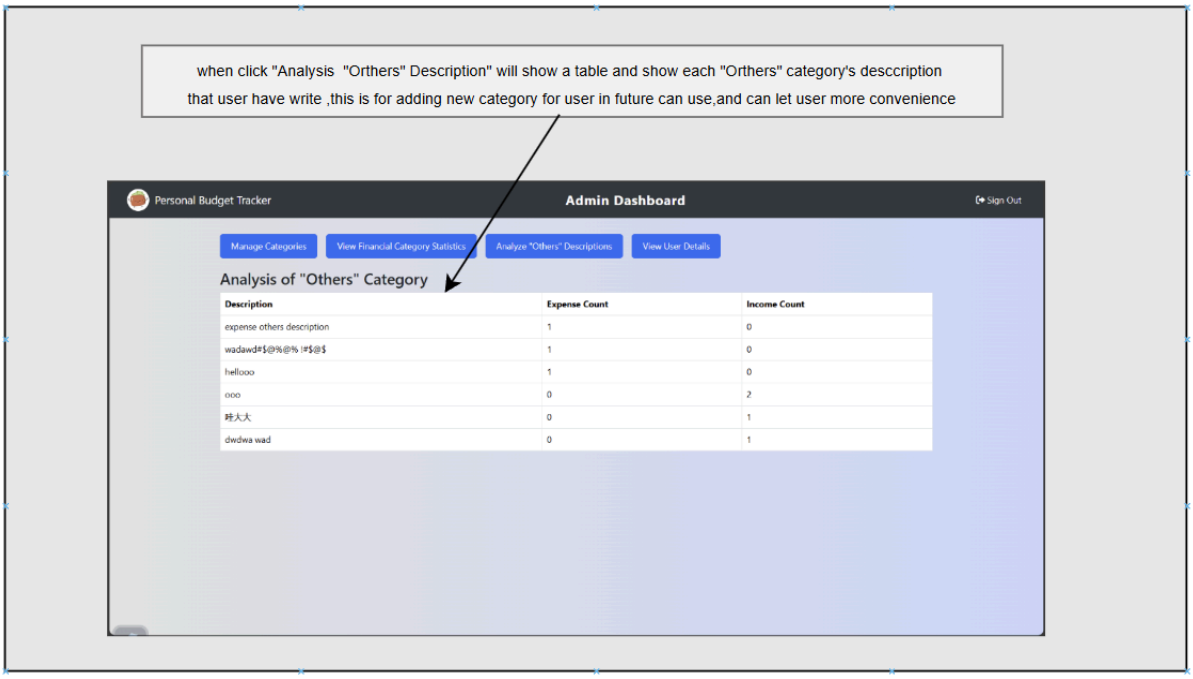
The buttons on the **Admin Dashboard** (such as **Manage Categories**, **View Financial Category Statistics**, **Analyze "Others" Descriptions**, and **View User Details**) are toggle buttons. When clicked the first time, they reveal their respective content or functionality. Clicking them a second time hides the displayed content, allowing for a clean and uncluttered interface. This toggle functionality ensures better usability and efficient navigation for administrators.

3.4.4 View Financial Category Statistics Page



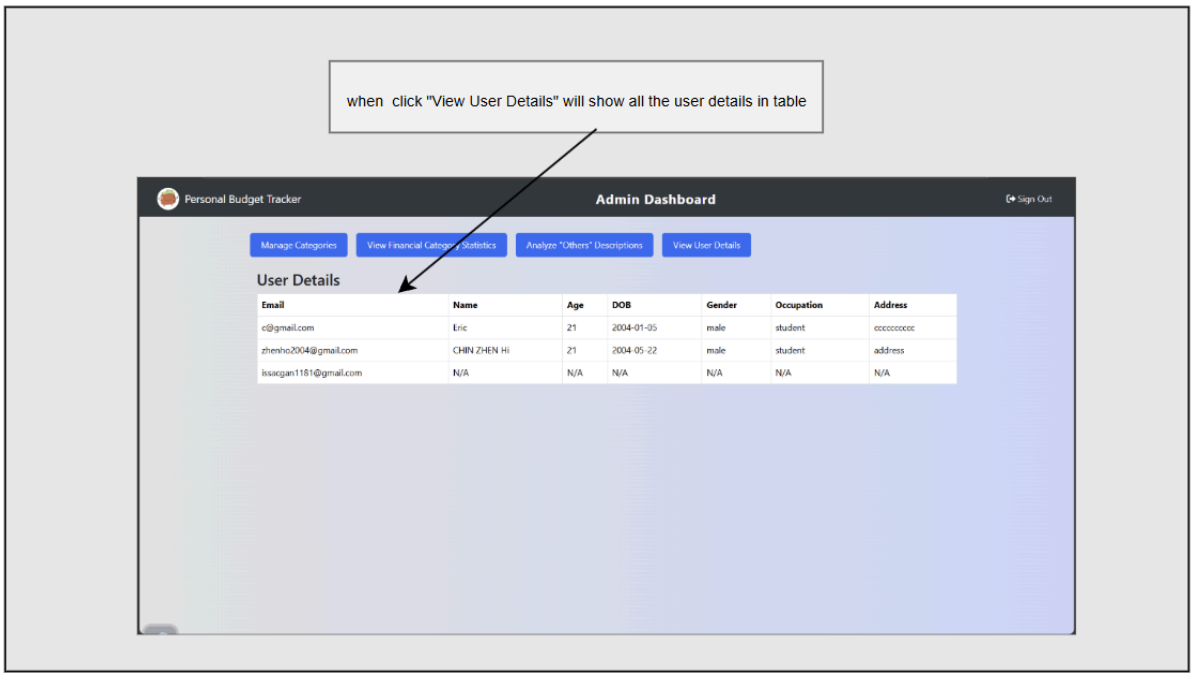
When the **View Financial Category Statistics** button is clicked, two pie charts are displayed, representing **Expense Categories** and **Income Categories**. Each chart visually breaks down the categories, such as food, study, salary, and others. Hovering over a category in the chart reveals a tooltip showing how many users have selected that category. This feature provides an intuitive and interactive way for administrators to analyze category usage statistics.

3.4.5 Analyze "Others" Descriptions Page



When the **Analyze "Others" Descriptions** button is clicked, a table appears displaying the descriptions under the "Others" category along with their corresponding **Expense Count** and **Income Count**. This feature allows administrators to review user-provided descriptions, helping them identify patterns or common entries that could be used to create new categories. It enhances user convenience by refining the category system based on actual usage and feedback.

3.4.6 View User Details Page

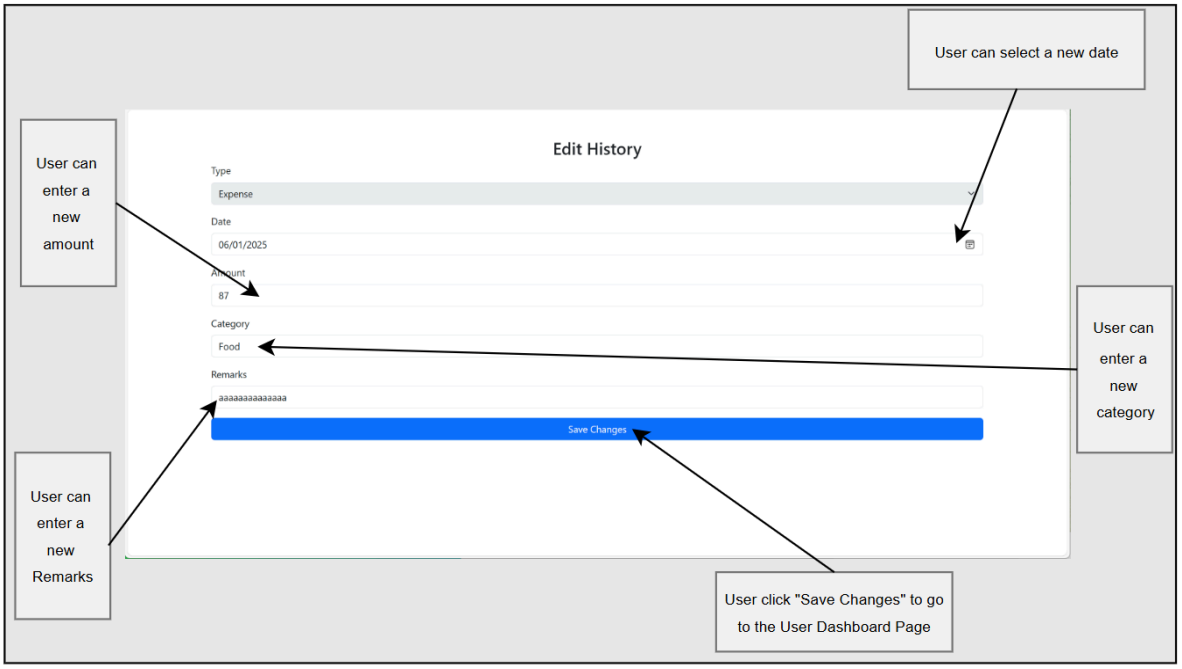


When the **View User Details** button is clicked, a table is displayed showing all user information, including fields such as **Email**, **Name**, **Age**, **Date of Birth (DOB)**, **Gender**, **Occupation**, and **Address**. This feature allows administrators to view and manage user data effectively, providing a clear overview of user details within the system.

3.4.1.1 Additional Task

3.4.1.1.1 Edit History Page

The **Edit History Page** provides a form for users to modify specific transaction details. Users can select a new date, change the amount, edit the category, and add or revise remarks. Once the necessary changes are made, clicking the **Save Changes** button updates the transaction and redirects the user back to the dashboard. This page offers a straightforward and user-friendly interface for editing transaction records, ensuring data accuracy and flexibility.



3.5 Subsystem Components

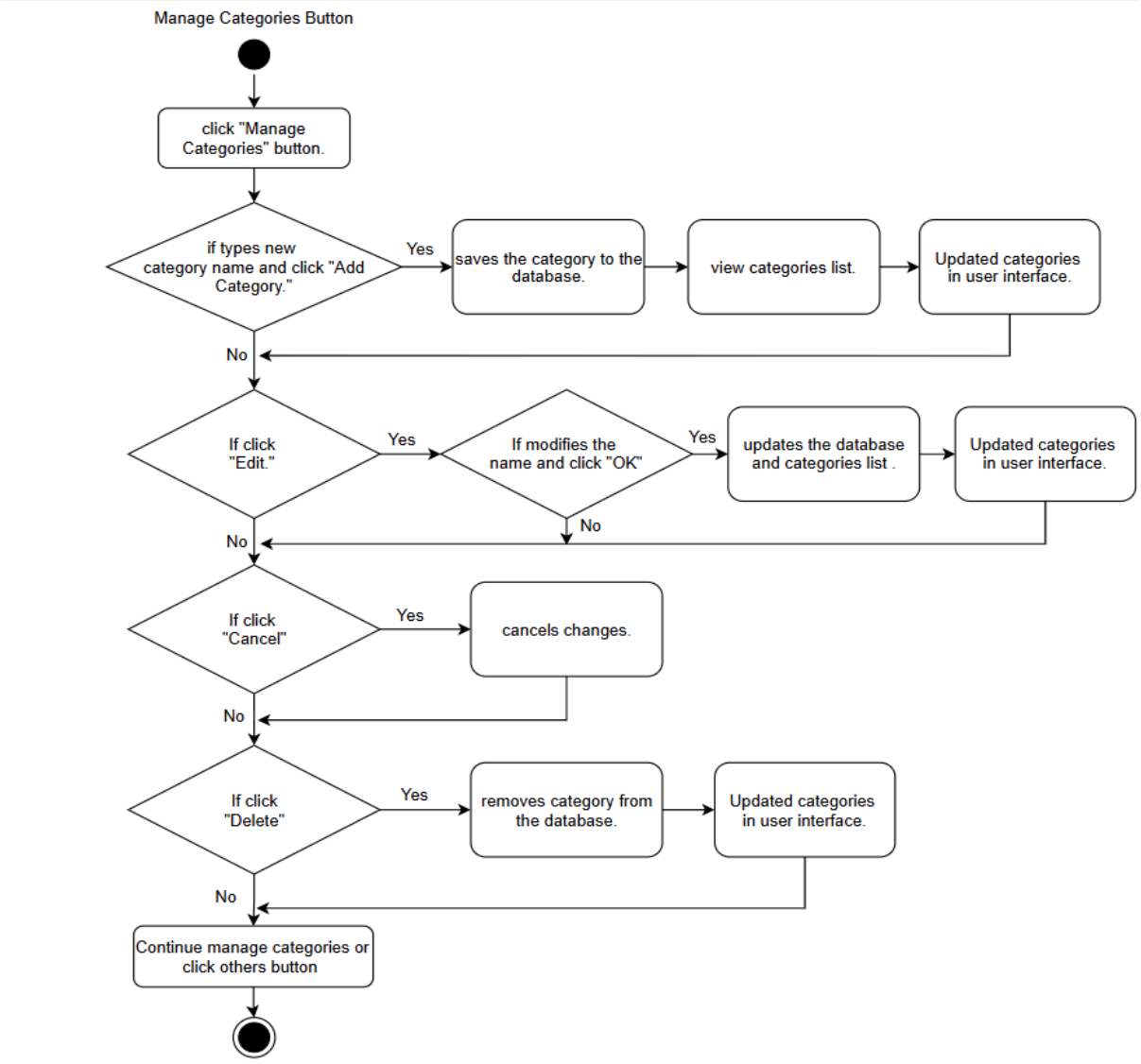
<TO DO: Describe the subsystem components (modules, classes, packages, etc.) and the table with the subsystem components here.>

Actor	Component
User	View Transaction History(Edit transaction History function)
Admin	Manage Categories
	View Financial Category Statistics
	Analysis "Others" Descriptions
	View User Details
	Login as Admin

3.5.1 Admin Component

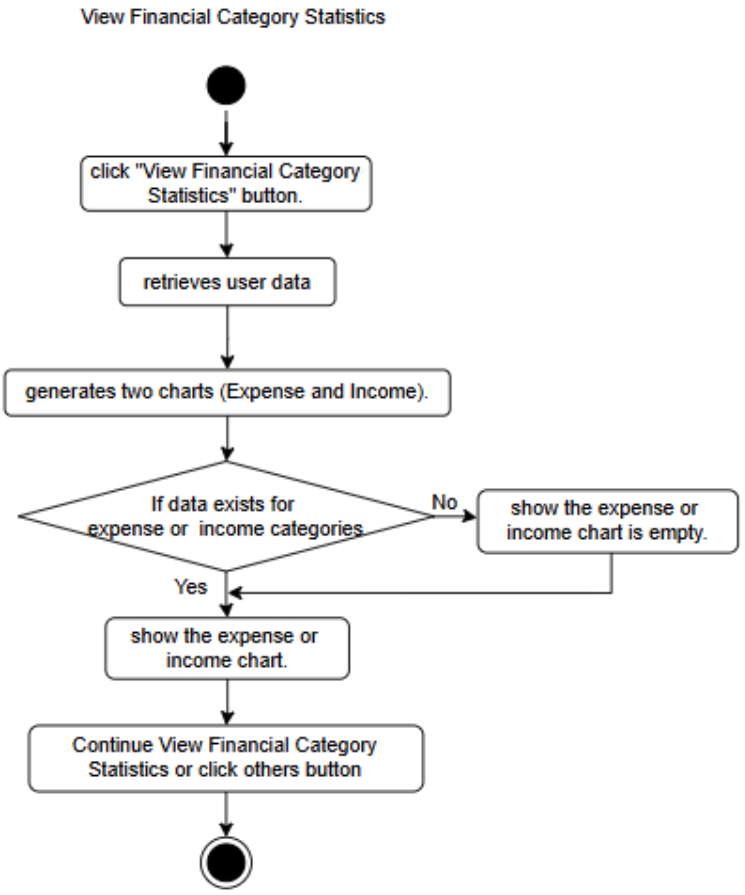
3.5.1.1 Manage Categories

The **Manage Categories** feature allows users to add, edit, delete, or cancel modifications to categories. Users can type a new category name and click **"Add Category"**, which validates and saves the data to the database, updating the category list in the interface. For editing, admin click **"Edit"**, modify the name, and confirm with **"OK"**, after which the system validates and updates the database. If users cancel during editing, changes are discarded. For deletion, users click **"Delete"**, confirm the action, and the category is removed from the database, with updates reflected in the interface. Throughout, error handling ensures invalid inputs are flagged, and confirmations are required for critical actions like deletion, ensuring a smooth and user-friendly process.



3.5.1.2 View Financial Category Statistics

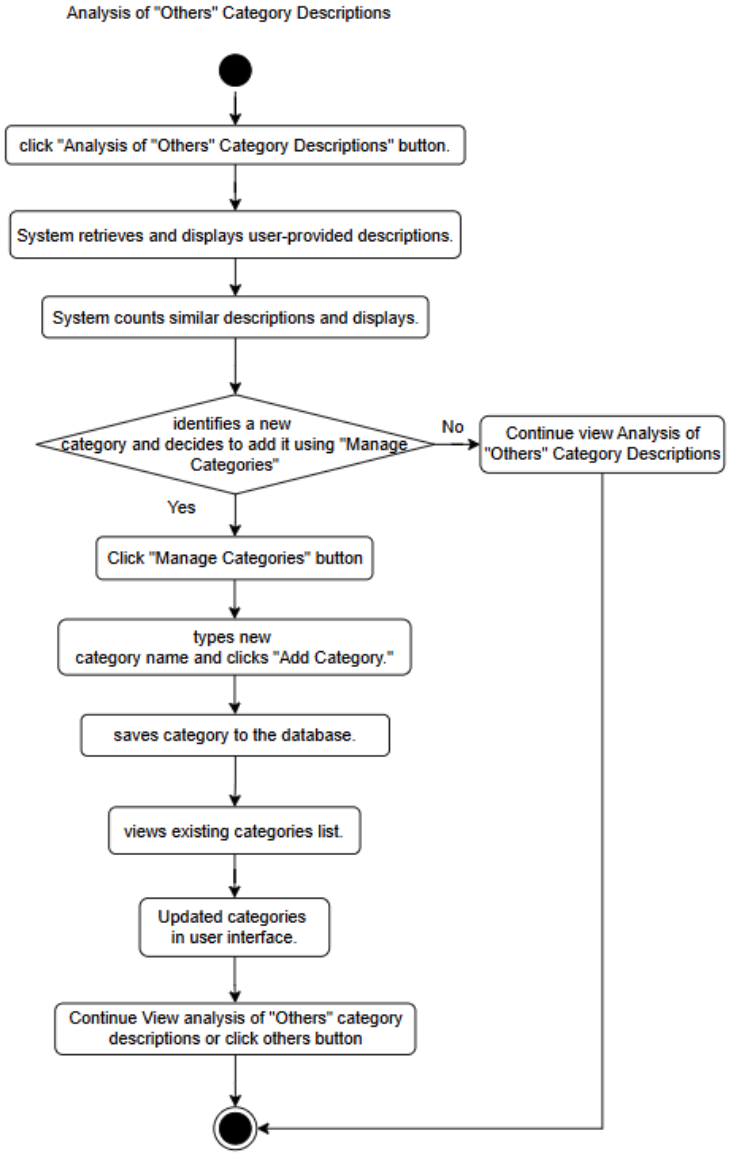
The **View Financial Category Statistics** feature allows users to visualize their financial data through charts. When the user clicks the **"View Financial Category Statistics"** button, the system retrieves the user's financial data. It generates two charts: one for expenses and one for income. If the data exists for either category, the respective chart is displayed to the admin. If no data is available, the system shows an empty chart with a message indicating the lack of data. After viewing the statistics, the user can continue exploring this feature or navigate to other sections of the application. This design ensures that users can easily understand and track their financial activities through clear visual representations.



…

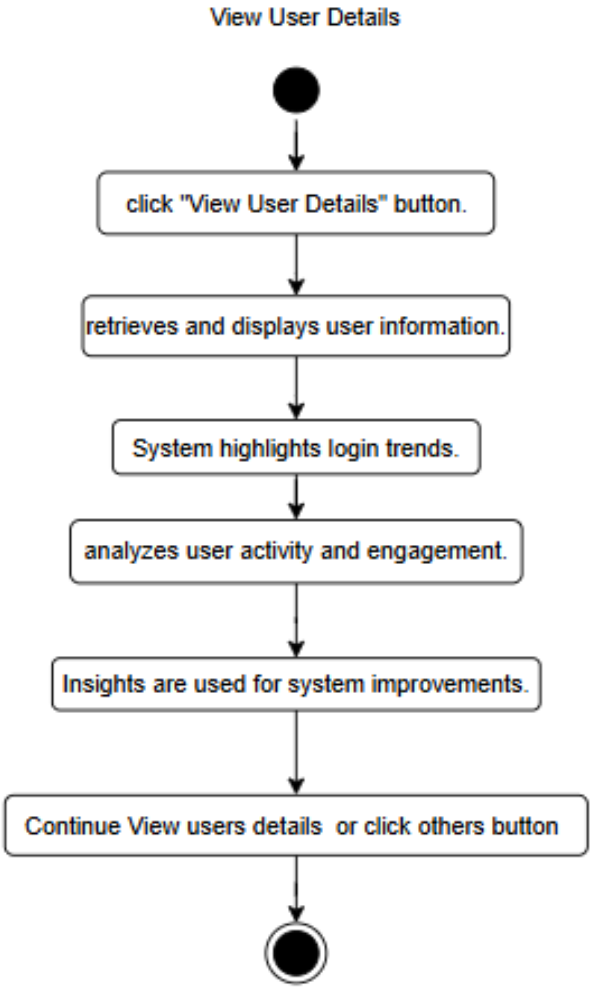
3.5.1.3 Analysis "Others" Descriptions

The **Analysis of "Others" Category Descriptions** feature allows admin to review and organize descriptions under the "Others" category. Upon clicking the **"Analysis of 'Others' Category Descriptions"** button, the system retrieves and displays user-provided descriptions, counts similar entries, and presents them for analysis. If the admin identifies a new category, admin can add it using the **"Manage Categories"** feature by entering the category name, which is then saved to the database and reflected in the updated category list.Admin can continue analyzing descriptions or navigate to other sections, ensuring better data organization and clarity.



3.5.1.4 View User Details

The **View User Details** feature allows admin to access detailed insights about user activity. When the **"View User Details"** button is clicked, the system retrieves and displays user information, including profile details and login trends. It further analyzes user activity and engagement patterns to provide meaningful insights. These insights are then utilized to identify areas for system improvements, ensuring a better user experience. After viewing the details, the admin can either continue reviewing the information or navigate to other sections of the application.



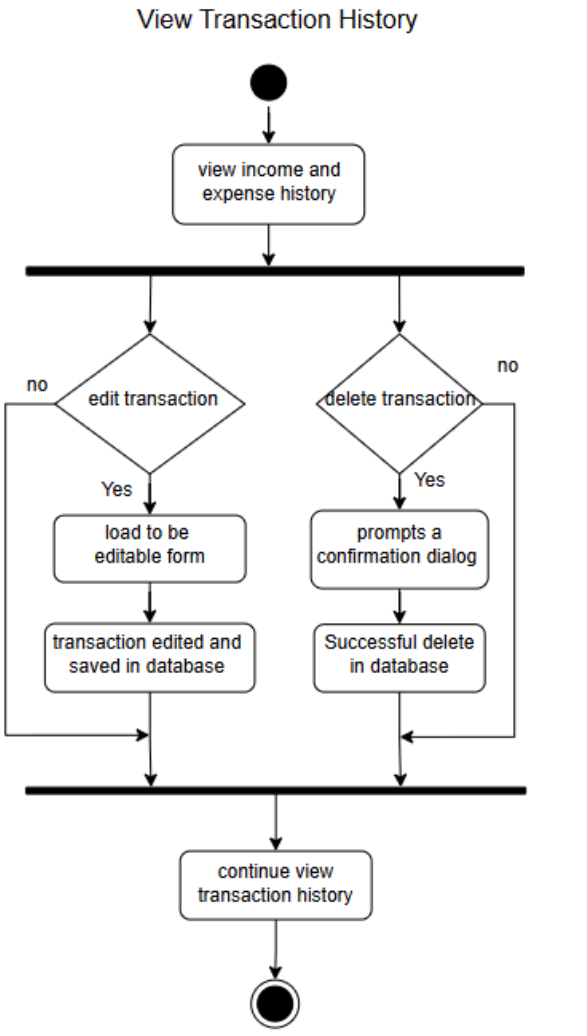
3.5.2 Additional Task

3.5.2.1User Component

3.5.2.1.1 Viewing Transaction History(Edit Transaction History function)

<TO DO: Describe the component and place the diagram here. There should be algorithm, pseudocode, flowchart, activity diagram to support the processing in the component.>

The user begins by viewing their income and expense history. They can choose to **edit a transaction** or **delete a transaction**. If the user chooses to edit, the selected transaction is loaded into an editable form, where changes are made and saved back to the database. If the user chooses to delete, a confirmation dialog appears; upon confirmation, the transaction is successfully deleted from the database. In both cases, the user returns to continue viewing the transaction history. If no action is taken, the user remains on the transaction history page.



4 Implementation

4.1 Development Environment

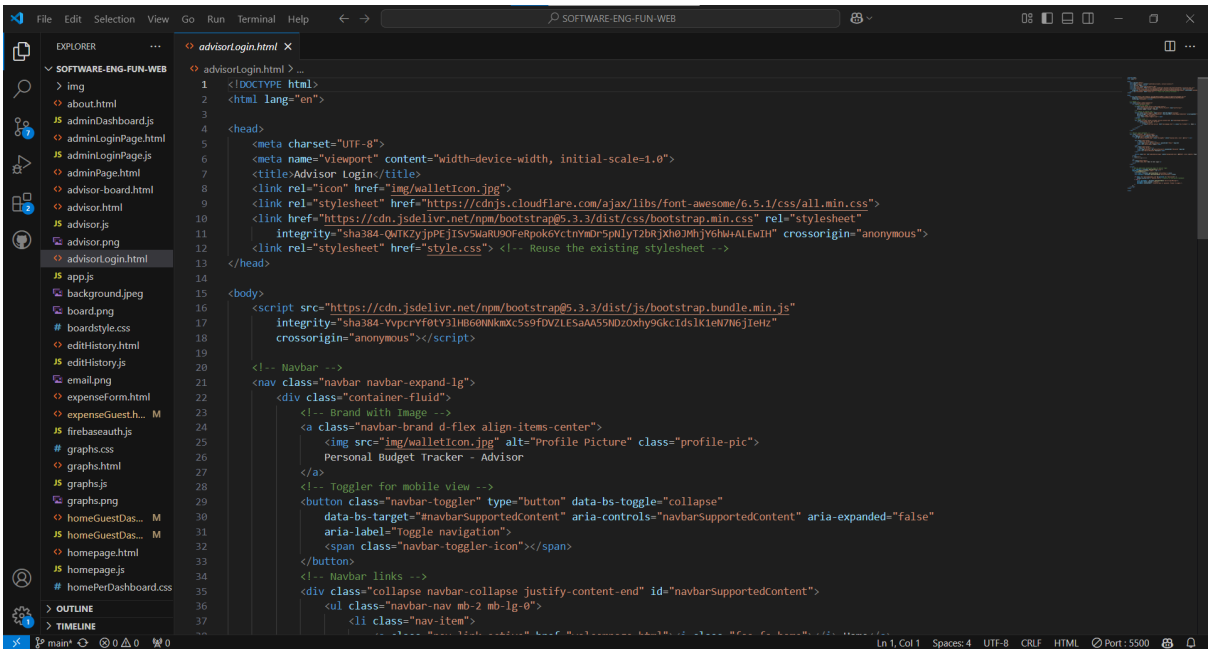
<TO DO: Describe the development environment here with relevant images that shows the code.>

In this assignment, we have used Visual Studio Code as our main IDE and used Firebase as our database.

VS Code:

For this project, Visual Studio Code (VS Code) was used as the primary development environment. VS Code is a lightweight yet powerful editor developed by Microsoft, known for its user-friendly interface, flexibility, and extensive support for multiple programming languages.

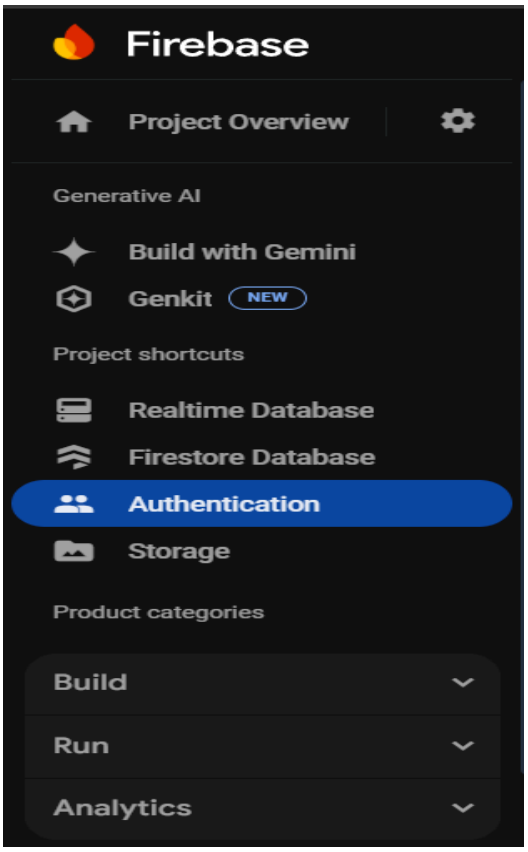
VS Code includes an integrated terminal, built-in debugging tools, and Git integration, allowing seamless coding, testing, and version control. Its extension support enables developers to enhance functionality with plugins for Python, C++, Java, and other tools. Additionally, customization options like themes and shortcuts improve workflow efficiency.



Firebase:

Firebase is a cloud-based platform by Google that provides backend services like real-time databases, authentication, cloud storage, and hosting. It simplifies app development by handling data synchronization, user authentication, and secure storage.

Key features include the Realtime Database and Firestore for live data updates, Authentication Services for secure logins, and Cloud Storage for managing user-generated content. Additionally, Cloud Functions enable serverless backend logic, while Firebase Hosting ensures fast and secure web deployment



Programming elements:

HTML:

HTML (HyperText Markup Language) is the standard language used for creating and structuring web pages. It defines the basic layout and content of a webpage using elements such as headings, paragraphs, images, links, and forms.

CSS:

CSS(**Cascading Style Sheets**) is a stylesheet language that is used in order to change the appearance and layout of HTML elements. It helps to enhance the visual presentation of these web pages by making styles such as maybe adding colors, spacing or even positioning. Every property contains one or more values that define how it looks or behave in a certain way.

Java Script:

JavaScript (**JS**) is a special programming language that helps these web pages by adding interactions and functions. It allows developers to create responsive user interactions, handle events, manipulate CSS and HTML and interact with APIs.

JavaScript runs in browsers and is used for things such as form validation, animations or event handling. It works alongside HTML for structure and CSS for styling, making this language an essential part of web development.

4.2 Main Program Codes

<TO DO: List and describe the applications required to implement the components, and the table with application and files, and place the relevant images that shows the code for the files here.>

Application	Files
• Admin Login Page	adminLoginPage.html adminLoginPage.js
1. Admin Main Page 2. Manage Categories Page 3. View Financial Category Statistics Page 4. Analyze "Others" Descriptions Page 5. View User Details Page	adminPage.html adminDashboard.js
Additional Task: • Edit History Page	editHistory.html editHistory.js

4.2.1.1 adminLoginPage.html

<TO DO: Place the relevant images that shows the code for the file here.>

```
<!DOCTYPE html>

<html lang="en">



<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Admin Login</title>

    <link rel="icon" href="img/walletIcon.jpg">

    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet"

        integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">

    <link rel="stylesheet" href="style.css"> <!-- Reuse the existing stylesheet -->



    <style>

        /* Ensure the error message is visible */

        #adminLoginMessage {

            display: none;

            /* Initially hidden */

            color: red;

            /* Error message color */

        }

    </style>

</head>
```

```
<body>

    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"

        integrity="sha384-YvpcrYf0tY3lHB60NNkmXc5s9fDVZLESaAA55NDzOxhy9GkcIdslK1eN7N6jIeHz"

        crossorigin="anonymous"></script>



    <!-- Navbar -->

    <nav class="navbar navbar-expand-lg">

        <div class="container-fluid">

            <!-- Brand with Image -->

            <a class="navbar-brand d-flex align-items-center">

                <img src="img/walletIcon.jpg" alt="Profile Picture" class="profile-pic">

                Personal Budget Tracker - Admin

            </a>

            <!-- Toggler for mobile view -->

            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"

                data-bs-target="#navbarSupportedContent"
aria-controls="navbarSupportedContent" aria-expanded="false"

                aria-label="Toggle navigation">

                <span class="navbar-toggler-icon"></span>

            </button>

            <!-- Navbar links -->

            <div class="collapse navbar-collapse justify-content-end"
id="navbarSupportedContent">

                <ul class="navbar-nav mb-2 mb-lg-0">

                    <li class="nav-item">

                        <a class="nav-link active" href="welcompage.html"><i class="fas
fa-home"></i> Home</a>

                    </li>

                </ul>
```

```
            </div>

        </div>

    </nav>



    <!-- Admin Login Container -->

    <div class="container" id="adminLogin">

        <h1 class="form-title">Admin Login</h1>

        <form method="post" action="">

            <div id="adminLoginMessage" class="messageDiv"

                style="display: none; color: red; font-weight: bold; margin-top: 10px;">

            </div>









            <div class="input-group">

                <i class="fas fa-envelope"></i>

                <input type="email" id="adminEmail" placeholder="Email" required>

                <label for="adminEmail">Email</label>

            </div>

            <div class="input-group">

                <i class="fas fa-lock"></i>

                <input type="password" id="adminPassword" placeholder="Password" required>

                <label for="adminPassword">Password</label>

            </div>



            <button class="btn" id="submitAdminLogin" style="background-color: #28a745;
color: white;">Sign In</button>

        </form>
```

```
        <p class="or">

            ----------or--------

        </p>

        <div class="guest-login">

            <a href="index.html">Back to User Login</a>

        </div>

    </div>



    <script src="adminLoginPage.js"></script>

</body>



</html>
```

4.2.1.2 adminLoginPage.js

<TO DO: Place the relevant images that shows the code for the file here.>

```
// Get elements from the HTML
```

```
const adminEmail = "admin@gmail.com";
const adminPassword = "admin123";

// Event listener for login button
submitAdminLogin.addEventListener("click", function (event) {
    event.preventDefault(); // Prevent form submission

    const enteredEmail = document.getElementById("adminEmail").value.trim();
    const enteredPassword = document.getElementById("adminPassword").value.trim();

    if (enteredEmail === "" || enteredPassword === "") {
        showErrorMessage("⚠️ Please enter both email and password.");
        return;
    }

    // Check credentials (simulated authentication)
    if (enteredEmail === adminEmail && enteredPassword === adminPassword) {
        window.location.href = "adminPage.html"; // Redirect on success
    } else {
        showErrorMessage("Invalid email or password. Please try again.");
    }
});

// Function to display the error message
function showErrorMessage(message) {
    adminLoginMessage.innerHTML = message;
    adminLoginMessage.style.display = "block"; // Make it visible
```

```
    adminLoginMessage.style.opacity = "1"; // Ensure full visibility

    adminLoginMessage.style.transition = "opacity 0.5s ease-in-out"; // Smooth fade-in
effect



    // Remove message after 3 seconds

    setTimeout(() => {

        adminLoginMessage.style.opacity = "0"; // Fade out

        setTimeout(() => {

            adminLoginMessage.style.display = "none"; // Hide completely

        }, 500);

    }, 3000);

}
```

4.2.2.1 adminPage.html

<TO DO: Place the relevant images that shows the code for the file here.>

```
<!DOCTYPE html>

<html lang="en">
```

```
<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Admin Dashboard</title>

    <link rel="icon" href="img/walletIcon.jpg">

    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet"

        integrity="sha384-QWTKZyjpPEjISv5WaRU9OFeRpok6YctnYmDr5pNlyT2bRjXh0JMhjY6hW+ALEwIH"
crossorigin="anonymous">

    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css"
rel="stylesheet">

    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

    <style>

        /* Navbar Styling */

        .navbar {

            position: fixed;

            top: 0;

            left: 0;

            width: 100%;

            background-color: #343a40;

            z-index: 1000;

            padding: 8px 20px;

            height: 65px;

        }



        .navbar .navbar-brand,

        .navbar .nav-link {

            color: white !important;
```

```
}

.navbar .nav-link:hover {
    color: #adb5bd !important;
}

.navbar-text {
    font-size: 1.5rem;
    letter-spacing: 1px;
}

.profile-pic {
    width: 40px;
    height: 40px;
    border-radius: 50%;
    margin-right: 10px;
}

/* Body Styling */
body {
    padding-top: 70px;
    background: linear-gradient(to right, #e2e2e2, #c9d6ff);
}

/* Button Group Styling */
#buttonGroup {
    display: flex;
    justify-content: flex-start;
```

```
        gap: 10px;
        margin-bottom: 20px;
    }

    /* Section Styling */
    #categoriesSection,
    #statisticsSection {
        margin-top: 20px;
        padding: 15px;
        background-color: #ffffff;
        border-radius: 8px;
        box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
    }

    .hide {
        display: none !important;
    }

    .categories-wrapper {
        display: flex;
        gap: 20px;
        flex-wrap: wrap;
    }

    .section {
        flex: 1;
        min-width: 45%;
    }
```

```
ul {
    list-style: none;
    padding: 0;
}

.category-list li {
    margin: 10px 0;
    padding: 10px;
    background: #f8f9fa;
    border: 1px solid #ddd;
    border-radius: 5px;
    display: flex;
    justify-content: space-between;
    align-items: center;
}



/* Custom Button Styling */
button {
    background-color: #0d6efd;
    /* Bootstrap primary button color */
    color: white;
    border: none;
    padding: 10px 20px;
    border-radius: 5px;
    cursor: pointer;
}
```

```
button:hover {
    background-color: #0b5ed7;
    /* Slightly darker shade for hover */
}

#analysisTableContainer {
    padding: 20px;
    background-color: #ffffff;
    border-radius: 8px;
    box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
}

#analysisTableContainer.hide {
    display: none !important;
}

#analysisOthersBtn {
    background-color: #007bff;
    color: white;
    border: none;
    padding: 10px 15px;
    border-radius: 5px;
    cursor: pointer;
}

#analysisOthersBtn:hover {
    background-color: #0056b3;
```

```
        }
    </style>
</head>

<body>
    <!-- Navbar -->
    <nav class="navbar navbar-expand-lg">
        <div class="container-fluid d-flex justify-content-between align-items-center">
            <a class="navbar-brand d-flex align-items-center">
                <img src="img/walletIcon.jpg" alt="Profile Picture" class="profile-pic">
                Personal Budget Tracker
            </a>
            <span class="navbar-text mx-auto text-white fw-bold">Admin Dashboard</span>
            <ul class="navbar-nav mb-2 mb-lg-0">
                <li class="nav-item">
                    <a class="nav-link text-white" href="#" id="logout">
                        <i class="fas fa-sign-out-alt"></i> Sign Out
                    </a>
                </li>
            </ul>
        </div>
    </nav>

    <!-- Main Content -->
    <div class="container mt-4">
        <!-- Button Group -->
        <div id="buttonGroup" class="d-flex gap-3">
            <button id="manageCategoriesBtn">Manage Categories</button>
```

```
            <button id="viewCategoryStatsBtn">View Financial Category Statistics</button>
            <button id="othersAnalysisBtn">Analyze "Others" Descriptions</button>
            <button id="viewUserDetailsBtn">View User Details</button>
        </div>



        <!-- Categories Section -->
        <div id="categoriesSection" class="hide categories-wrapper">
            <div class="section">
                <h2>Manage Expense Categories</h2>
                <div class="form-group d-flex mb-3">
                    <input type="text" id="expenseCategoryInput" class="form-control me-2"
                        placeholder="Enter new expense category name">
                    <button id="addExpenseCategoryBtn" class="btn btn-success">Add
Category</button>
                </div>
                <h3>Existing Expense Categories</h3>
                <ul id="expenseCategoryList" class="category-list"></ul>
            </div>


            <div class="section">
                <h2>Manage Income Categories</h2>
                <div class="form-group d-flex mb-3">
                    <input type="text" id="incomeCategoryInput" class="form-control me-2"
                        placeholder="Enter new income category name">
                    <button id="addIncomeCategoryBtn" class="btn btn-success">Add
Category</button>
                </div>
                <h3>Existing Income Categories</h3>
```

```
            <ul id="incomeCategoryList" class="category-list"></ul>
        </div>
    </div>



    <!-- Financial Statistics Section -->
    <div id="categoryStatsSection" class="hide mt-4">
        <h2>Financial Category Statistics</h2>
        <div class="row">
            <div class="col-md-6">
                <h3>Expense Categories</h3>
                <canvas id="expenseChart"></canvas>
            </div>
            <div class="col-md-6">
                <h3>Income Categories</h3>
                <canvas id="incomeChart"></canvas>
            </div>
        </div>
    </div>


    <!-- Analysis Section -->
    <div id="othersAnalysisSection" class="hide">
        <h3>Analysis of "Others" Category</h3>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Description</th>
                    <th>Expense Count</th>
```

```
                    <th>Income Count</th>
                </tr>
            </thead>
            <tbody id="othersAnalysisTable"></tbody>
        </table>
    </div>



    <!-- User Details Section -->
    <div id="userDetailsSection" class="container mt-4" style="display: none;">
        <h3>User Details</h3>
        <table id="userDetailsTable" class="table table-bordered">
            <thead>
                <tr>
                    <th>Email</th>
                    <th>Name</th>
                    <th>Age</th>
                    <th>DOB</th>
                    <th>Gender</th>
                    <th>Occupation</th>
                    <th>Address</th>
                </tr>
            </thead>
            <tbody>
                <!-- User details will be dynamically populated here -->
            </tbody>
        </table>
    </div>
```

```
    </div>



    <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script
>
    <script type="module" src="adminDashboard.js"></script>
</body>


</html>
```

4.2.2.2 adminDashboard.js

<TO DO: Place the relevant images that shows the code for the file here.>

```
// Import Firebase modules

import { initializeApp } from "https://www.gstatic.com/firebasejs/11.0.2/firebase-app.js";

import { getFirestore, collection, addDoc, getDocs, updateDoc, deleteDoc, doc,
collectionGroup, getDoc } from
"https://www.gstatic.com/firebasejs/11.0.2/firebase-firestore.js";
```

```
import { getAuth, onAuthStateChanged, signOut } from
"https://www.gstatic.com/firebasejs/11.0.2/firebase-auth.js";



// Firebase configuration

const firebaseConfig = {

    apiKey: "AIzaSyDlrbumKo34d3Hel__bA1Xj6TG_ClFOF0g",

    authDomain: "login-form-4c66d.firebaseapp.com",

    projectId: "login-form-4c66d",

    storageBucket: "login-form-4c66d.firebasestorage.app",

    messagingSenderId: "9677626952",

    appId: "1:9677626952:web:4f55f083e1cdf36f77c27d"

};



// Initialize Firebase

const app = initializeApp(firebaseConfig);

const db = getFirestore(app);




// HTML Elements for Expense Categories

const manageCategoriesBtn = document.getElementById('manageCategoriesBtn');

const categoriesSection = document.getElementById('categoriesSection');

const expenseCategoryInput = document.getElementById('expenseCategoryInput');

const addExpenseCategoryBtn = document.getElementById('addExpenseCategoryBtn');

const expenseCategoryList = document.getElementById('expenseCategoryList');



// HTML Elements for Income Categories

const incomeCategoryInput = document.getElementById('incomeCategoryInput');

const addIncomeCategoryBtn = document.getElementById('addIncomeCategoryBtn');
```

```
const incomeCategoryList = document.getElementById('incomeCategoryList');



// Elements

const viewCategoryStatsBtn = document.getElementById('viewCategoryStatsBtn');

const categoryStatsSection = document.getElementById('categoryStatsSection');

const expenseChartCanvas = document.getElementById('expenseChart').getContext('2d');

const incomeChartCanvas = document.getElementById('incomeChart').getContext('2d');




// Chart Instances (to avoid recreating)

let expenseChart;

let incomeChart;




// Sign Out Button

const logoutBtn = document.getElementById('logout');



logoutBtn.addEventListener('click', (e) => {

    e.preventDefault(); // Prevent the default action of the link

    window.location.href = 'welcompage.html'; // Redirect to the welcome page

});



// Toggle Categories Section Visibility

manageCategoriesBtn.addEventListener('click', () => {

    if (categoriesSection.classList.contains('hide')) {

        categoriesSection.classList.remove('hide'); // Show the section

    } else {

        categoriesSection.classList.add('hide'); // Hide the section
```

```
    }
});









// Fetch and Display Categories
async function displayExpenseCategories() {
    expenseCategoryList.innerHTML = ''; // Clear the list
    const querySnapshot = await getDocs(collection(db, 'expenseCategories'));
    querySnapshot.forEach((docSnapshot) => {
        const category = docSnapshot.data();
        const li = document.createElement('li');
        li.textContent = category.name;


        // Create Edit Button
        const editButton = document.createElement('button');
        editButton.textContent = 'Edit';
        editButton.onclick = () => editExpenseCategory(docSnapshot.id, category.name);


        // Create Delete Button
        const deleteButton = document.createElement('button');
        deleteButton.textContent = 'Delete';
        deleteButton.onclick = () => deleteExpenseCategory(docSnapshot.id);


        li.appendChild(editButton);
        li.appendChild(deleteButton);
```

```
        expenseCategoryList.appendChild(li);
    });
}

async function displayIncomeCategories() {
    incomeCategoryList.innerHTML = ''; // Clear the list
    const querySnapshot = await getDocs(collection(db, 'incomeCategories'));
    querySnapshot.forEach((docSnapshot) => {
        const category = docSnapshot.data();
        const li = document.createElement('li');
        li.textContent = category.name;

        // Create Edit Button
        const editButton = document.createElement('button');
        editButton.textContent = 'Edit';
        editButton.onclick = () => editIncomeCategory(docSnapshot.id, category.name);

        // Create Delete Button
        const deleteButton = document.createElement('button');
        deleteButton.textContent = 'Delete';
        deleteButton.onclick = () => deleteIncomeCategory(docSnapshot.id);

        li.appendChild(editButton);
        li.appendChild(deleteButton);
        incomeCategoryList.appendChild(li);
    });
}
```

```
// Add New Expense Category
addExpenseCategoryBtn.addEventListener('click', async () => {
    const categoryName = expenseCategoryInput.value.trim();
    if (categoryName) {
        try {
            await addDoc(collection(db, 'expenseCategories'), { name: categoryName });
            expenseCategoryInput.value = '';
            displayExpenseCategories(); // Refresh the list
            alert('Expense category added successfully!');
        } catch (error) {
            console.error('Error adding expense category:', error);
            alert('Failed to add expense category. Error: ' + error.message);
        }
    } else {
        alert('Please enter a category name.');
    }
});


// Add New Income Category
addIncomeCategoryBtn.addEventListener('click', async () => {
    const categoryName = incomeCategoryInput.value.trim();
    if (categoryName) {
        try {
            await addDoc(collection(db, 'incomeCategories'), { name: categoryName });
            incomeCategoryInput.value = '';
            displayIncomeCategories(); // Refresh the list
            alert('Income category added successfully!');
        } catch (error) {
```

```
            console.error('Error adding income category:', error);
            alert('Failed to add income category. Error: ' + error.message);
        }
    } else {
        alert('Please enter a category name.');
    }
});

logoutBtn.addEventListener('click', async (e) => {
    e.preventDefault();
    const auth = getAuth();
    try {
        await signOut(auth); // Sign out the user
        window.location.href = 'welcompage.html'; // Redirect to the welcome page
    } catch (error) {
        console.error("Error signing out:", error);
        alert("Failed to sign out. Try again.");
    }
});

// Edit Expense Category
async function editExpenseCategory(categoryId, currentName) {
    const newName = prompt('Edit expense category name:', currentName);
    if (newName && newName.trim() !== '') {
        try {
            const docRef = doc(db, 'expenseCategories', categoryId);
            await updateDoc(docRef, { name: newName });
            displayExpenseCategories(); // Refresh the list
```

```
            alert('Expense category updated successfully!');
        } catch (error) {
            console.error('Error updating expense category:', error);
            alert('Failed to update expense category. Error: ' + error.message);
        }
    }
}

// Edit Income Category
async function editIncomeCategory(categoryId, currentName) {
    const newName = prompt('Edit income category name:', currentName);
    if (newName && newName.trim() !== '') {
        try {
            const docRef = doc(db, 'incomeCategories', categoryId);
            await updateDoc(docRef, { name: newName });
            displayIncomeCategories(); // Refresh the list
            alert('Income category updated successfully!');
        } catch (error) {
            console.error('Error updating income category:', error);
            alert('Failed to update income category. Error: ' + error.message);
        }
    }
}

// Delete Expense Category
async function deleteExpenseCategory(categoryId) {
    const confirmDelete = confirm('Are you sure you want to delete this expense
category?');
```

```
    if (confirmDelete) {
        try {
            const docRef = doc(db, 'expenseCategories', categoryId);
            await deleteDoc(docRef);
            displayExpenseCategories(); // Refresh the list
            alert('Expense category deleted successfully!');
        } catch (error) {
            console.error('Error deleting expense category:', error);
            alert('Failed to delete expense category. Error: ' + error.message);
        }
    }
}


// Delete Income Category
async function deleteIncomeCategory(categoryId) {
    const confirmDelete = confirm('Are you sure you want to delete this income category?');
    if (confirmDelete) {
        try {
            const docRef = doc(db, 'incomeCategories', categoryId);
            await deleteDoc(docRef);
            displayIncomeCategories(); // Refresh the list
            alert('Income category deleted successfully!');
        } catch (error) {
            console.error('Error deleting income category:', error);
            alert('Failed to delete income category. Error: ' + error.message);
        }
    }
}
```

```
// Show Category Statistics Section
viewCategoryStatsBtn.addEventListener('click', async () => {
    if (categoryStatsSection.classList.contains('hide')) {
        categoryStatsSection.classList.remove('hide');
        await loadCategoryStatistics(); // Load data when the section is displayed
    } else {
        categoryStatsSection.classList.add('hide');
    }
});

// Fetch and Load Statistics
async function loadCategoryStatistics() {
    const expenseData = {};
    const incomeData = {};

    // Fetch Expense Records from All Users
    const expenseQuerySnapshot = await getDocs(collection(db, "users"));
    expenseQuerySnapshot.forEach((userDoc) => {
        const expensesCollection = collection(db, "users", userDoc.id, "expenses");
        getDocs(expensesCollection).then((expensesSnapshot) => {
            expensesSnapshot.forEach((expense) => {
                const category = expense.data().category;
                if (category) {
                    expenseData[category] = (expenseData[category] || 0) + 1;
                }
            });
```

```
            // Update the chart with new data
            renderExpenseChart(expenseData);
        });
    });

    // Fetch Income Records from All Users
    const incomeQuerySnapshot = await getDocs(collection(db, "users"));
    incomeQuerySnapshot.forEach((userDoc) => {
        const incomesCollection = collection(db, "users", userDoc.id, "incomes");
        getDocs(incomesCollection).then((incomesSnapshot) => {
            incomesSnapshot.forEach((income) => {
                const category = income.data().category;
                if (category) {
                    incomeData[category] = (incomeData[category] || 0) + 1;
                }
            });

            // Update the chart with new data
            renderIncomeChart(incomeData);
        });
    });
}

// Render Expense Chart
function renderExpenseChart(data) {
    const labels = Object.keys(data);
    const values = Object.values(data);
```

```
    if (expenseChart) {
        expenseChart.destroy(); // Destroy existing chart to prevent duplication
    }

    expenseChart = new Chart(expenseChartCanvas, {
        type: 'pie',
        data: {
            labels: labels,
            datasets: [{
                label: 'Expense Categories',
                data: values,
                backgroundColor: generateColors(labels.length)
            }]
        },
        options: {
            responsive: true,
            plugins: {
                legend: { position: 'top' },
                tooltip: { enabled: true }
            }
        }
    });
}

// Render Income Chart
function renderIncomeChart(data) {
    const labels = Object.keys(data);
```

```
    const values = Object.values(data);



    if (incomeChart) {

        incomeChart.destroy(); // Destroy existing chart to prevent duplication

    }



    incomeChart = new Chart(incomeChartCanvas, {

        type: 'pie',

        data: {

            labels: labels,

            datasets: [{

                label: 'Income Categories',

                data: values,

                backgroundColor: generateColors(labels.length)

            }]

        },

        options: {

            responsive: true,

            plugins: {

                legend: { position: 'top' },

                tooltip: { enabled: true }

            }

        }

    });

}




// Generate Unique Colors for Categories
```

```
function generateColors(count) {

    const colors = [];

    const hueStep = Math.floor(360 / count); // Divide the hue range by the number of
categories

    for (let i = 0; i < count; i++) {

        const hue = i * hueStep; // Assign a distinct hue for each category

        const saturation = 70; // Keep saturation fixed for consistency

        const lightness = 60; // Keep lightness fixed for consistency

        colors.push(`hsl(${hue}, ${saturation}%, ${lightness}%)`); // Generate color

    }

    return colors;

}







//改了这里

document.addEventListener('DOMContentLoaded', () => {

    // HTML elements for 'Others' analysis

    const othersAnalysisBtn = document.getElementById('othersAnalysisBtn');

    const othersAnalysisSection = document.getElementById('othersAnalysisSection');

    const othersAnalysisTable = document.getElementById('othersAnalysisTable');



    // Event listener for the "Others Analysis" button

    // Event listener for the "Others Analysis" button

    othersAnalysisBtn.addEventListener('click', async () => {

        // Toggle visibility of the analysis section

        othersAnalysisSection.classList.toggle('hide');
```

```
        if (!othersAnalysisSection.classList.contains('hide')) {

            othersAnalysisTable.innerHTML = '<tr><td colspan="3">Loading...</td></tr>'; //
Show loading state



            try {

                const othersData = {}; // To store aggregated descriptions and their counts



                // Fetch "Others" from Expenses

                const expenseQuerySnapshot = await getDocs(collectionGroup(db,
'expenses'));

                expenseQuerySnapshot.forEach((docSnapshot) => {

                    const expense = docSnapshot.data();

                    if (expense.category === 'Others' || expense.category === 'Orthers') {

                        const description = expense.remarks || 'No description provided';

                        othersData[description] = othersData[description] || { expenses: 0,
incomes: 0 };

                        othersData[description].expenses += 1; // Increment count for
expenses

                    }

                });



                // Fetch "Others" from Incomes

                const incomeQuerySnapshot = await getDocs(collectionGroup(db, 'incomes'));

                incomeQuerySnapshot.forEach((docSnapshot) => {

                    const income = docSnapshot.data();

                    if (income.category === 'Others' || income.category === 'Orthers') {

                        const description = income.remarks || 'No description provided';

                        othersData[description] = othersData[description] || { expenses: 0,
incomes: 0 };

                        othersData[description].incomes += 1; // Increment count for
incomes
```

```
                    }

                });



                console.log('Fetched "Others" Data:', othersData); // Debug the fetched
data



                // Build table rows with aggregated data

                const tableRows = Object.entries(othersData)

                    .map(

                        ([description, counts]) =>

                            `<tr>

                            <td>${description}</td>

                            <td>${counts.expenses}</td>

                            <td>${counts.incomes}</td>

                        </tr>`

                    )

                    .join('');



                // Populate table with results or show "No data available"

                othersAnalysisTable.innerHTML =

                    tableRows || '<tr><td colspan="3">No data available.</td></tr>';

            } catch (error) {

                console.error('Error fetching "Others" data:', error);

                othersAnalysisTable.innerHTML = '<tr><td colspan="3">Failed to load data.
Please try again.</td></tr>';

            }

        }

    });
```

```
});









document.getElementById("viewUserDetailsBtn").addEventListener("click", async () => {
    const userDetailsSection = document.getElementById("userDetailsSection");
    const userDetailsTableBody = document.querySelector("#userDetailsTable tbody");

    // Check if the user details section is already visible
    if (userDetailsSection.style.display === "block") {
        // Hide the section
        userDetailsSection.style.display = "none";
        return; // Exit the function
    }

    try {
        // Show the user details section (unhide it)
        userDetailsSection.style.display = "block";

        // Clear previous user details
        userDetailsTableBody.innerHTML = "<tr><td colspan='7'>Loading...</td></tr>";

        // Fetch user documents from the "users" collection
        const usersCollection = collection(db, "users");
        const usersSnapshot = await getDocs(usersCollection);
```

```
        if (!usersSnapshot.empty) {

            // Fetch details for all users asynchronously

            const userDetailsPromises = usersSnapshot.docs.map(async (userDoc) => {

                const userId = userDoc.id;

                const userData = userDoc.data(); // Get data directly from the root
document



                // Fetch additional details (personal-info) if available

                const userDetailsRef = doc(db, "users", userId, "details",
"personal-info");

                const userDetailsSnap = await getDoc(userDetailsRef);



                const userDetails = userDetailsSnap.exists() ? userDetailsSnap.data() : {};



                return `

                    <tr>

                        <td>${userData.email || "N/A"}</td> <!-- Email fetched from root
document -->

                        <td>${userDetails.name || "N/A"}</td>

                        <td>${userDetails.age || "N/A"}</td>

                        <td>${userDetails.dob || "N/A"}</td>

                        <td>${userDetails.gender || "N/A"}</td>

                        <td>${userDetails.occupation || "N/A"}</td>

                        <td>${userDetails.address || "N/A"}</td>

                    </tr>

                `;

            });



            // Wait for all user details to be fetched
```

```
            const userDetailsRows = await Promise.all(userDetailsPromises);



            // Populate the table

            userDetailsTableBody.innerHTML = userDetailsRows.join("") || "<tr><td
colspan='7'>No users found.</td></tr>";

        } else {

            userDetailsTableBody.innerHTML = "<tr><td colspan='7'>No users
found.</td></tr>";

        }

    } catch (error) {

        console.error("Error fetching user details:", error);

        userDetailsTableBody.innerHTML = `<tr><td colspan='7'>Error fetching user details:
${error.message}</td></tr>`;

    }

});




// Display categories on page load

displayExpenseCategories();

displayIncomeCategories();
```

4.2.3.1 editHistory.html

<TO DO: Place the relevant images that shows the code for the file here.>

```
<!DOCTYPE html>

<html lang="en">



<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Edit History</title>
```

```
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css"
rel="stylesheet">

    <script type="module" src="editHistory.js"></script>

</head>



<body>

    <div class="container mt-5">

        <h3 class="text-center">Edit History</h3>

        <form id="editHistoryForm">

            <div class="mb-3">

                <label for="type" class="form-label">Type</label>

                <select id="type" class="form-select" disabled>

                    <option value="Income">Income</option>

                    <option value="Expense">Expense</option>

                </select>

            </div>

            <div class="mb-3">

                <label for="date" class="form-label">Date</label>

                <input type="date" id="date" class="form-control" required>

            </div>

            <div class="mb-3">

                <label for="amount" class="form-label">Amount</label>

                <input type="number" id="amount" class="form-control" required>

            </div>

            <div class="mb-3">

                <label for="category" class="form-label">Category</label>

                <input type="text" id="category" class="form-control" required>

            </div>
```

```
            <div class="mb-3">
                <label for="remarks" class="form-label">Remarks</label>
                <input type="text" id="remarks" class="form-control">
            </div>
            <button type="submit" class="btn btn-primary w-100">Save Changes</button>
        </form>
    </div>
</body>


</html>
```

4.2.3.2 editHistory.js

<TO DO: Place the relevant images that shows the code for the file here.>

```
import { initializeApp } from "https://www.gstatic.com/firebasejs/11.0.2/firebase-app.js";
```

```
import { getFirestore, doc, getDoc, updateDoc } from
"https://www.gstatic.com/firebasejs/11.0.2/firebase-firestore.js";

import { getAuth } from "https://www.gstatic.com/firebasejs/11.0.2/firebase-auth.js";



// Firebase configuration
```

```
const firebaseConfig = {

    apiKey: "AIzaSyDlrbumKo34d3Hel__bA1Xj6TG_ClFOF0g",

    authDomain: "login-form-4c66d.firebaseapp.com",

    projectId: "login-form-4c66d",

    storageBucket: "login-form-4c66d.appspot.com",

    messagingSenderId: "9677626952",

    appId: "1:9677626952:web:4f55f083e1cdf36f77c27d"

};



// Initialize Firebase

const app = initializeApp(firebaseConfig);

const db = getFirestore(app);

const auth = getAuth();



const form = document.getElementById("editHistoryForm");

const typeInput = document.getElementById("type");

const dateInput = document.getElementById("date");

const amountInput = document.getElementById("amount");

const categoryInput = document.getElementById("category");

const remarksInput = document.getElementById("remarks");



// Get URL parameters

const urlParams = new URLSearchParams(window.location.search);

const id = urlParams.get("id");



if (!id) {

    alert("No entry ID provided. Redirecting back.");

    window.location.href = "homePerDashboard.html";
```

```
}



// Fetch the history entry

auth.onAuthStateChanged(async (user) => {

    if (user) {

        const userId = user.uid;

        const incomeDocRef = doc(db, "users", userId, "incomes", id);

        const expenseDocRef = doc(db, "users", userId, "expenses", id);



        try {

            const incomeDoc = await getDoc(incomeDocRef);

            const expenseDoc = await getDoc(expenseDocRef);



            let data = null;

            if (incomeDoc.exists()) {

                data = incomeDoc.data();

                typeInput.value = "Income";

            } else if (expenseDoc.exists()) {

                data = expenseDoc.data();

                typeInput.value = "Expense";

            } else {

                throw new Error("Document not found.");

            }



            // Populate the form with the data

            dateInput.value = new Date(data.timestamp.seconds *
1000).toISOString().split("T")[0];

            amountInput.value = data.amount;
```

```
            categoryInput.value = data.category;
            remarksInput.value = data.remarks;

        } catch (error) {
            console.error("Error fetching document:", error);
            alert("Failed to fetch entry. Redirecting back.");
            window.location.href = "homePerDashboard.html";
        }
    } else {
        alert("User not signed in. Redirecting to login.");
        window.location.href = "login.html";
    }
});

// Handle form submission
form.addEventListener("submit", async (event) => {
    event.preventDefault();

    const user = auth.currentUser;
    if (!user) {
        alert("User not signed in. Redirecting to login.");
        window.location.href = "login.html";
        return;
    }

    const userId = user.uid;
    const collectionName = typeInput.value === "Income" ? "incomes" : "expenses";
    const docRef = doc(db, "users", userId, collectionName, id);
```

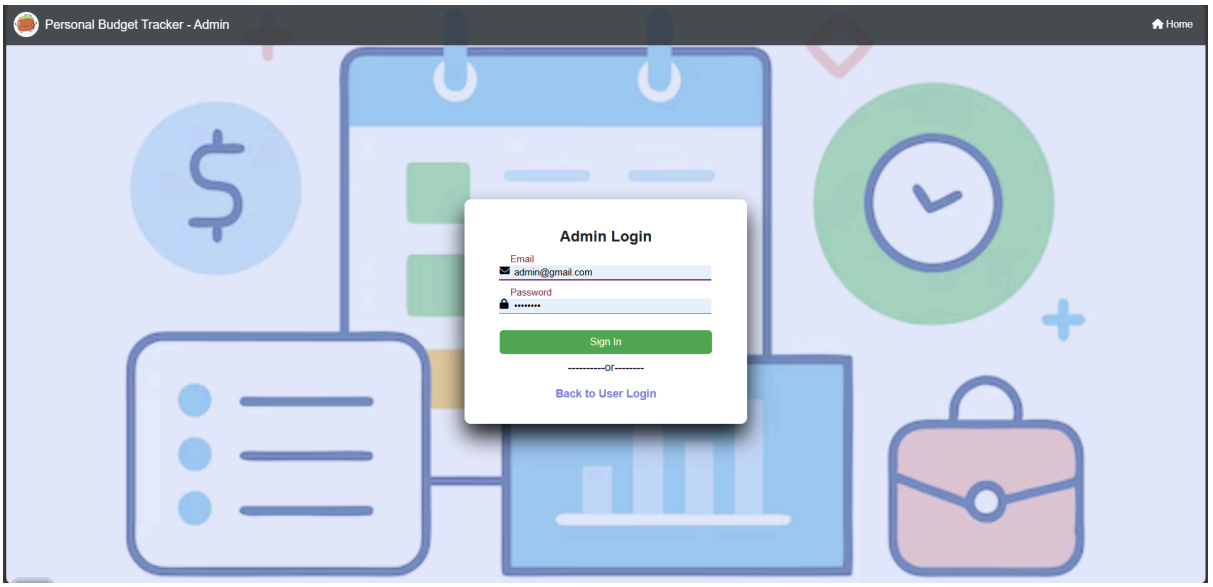
```
    // Prepare the updated data
    const updatedData = {
        amount: parseFloat(amountInput.value),
        category: categoryInput.value.trim(),
        remarks: remarksInput.value.trim(),
        timestamp: new Date(dateInput.value), // Use a Date object for Firestore to handle
properly
    };

    try {
        // Use updateDoc to only update the specific fields without overwriting the entire
document
        await updateDoc(docRef, updatedData);
        alert("History updated successfully!");
        window.location.href = "homePerDashboard.html"; // Redirect back to the dashboard
    } catch (error) {
        console.error("Error updating document:", error);
        alert("Failed to update entry. Please try again.");
    }
});
```

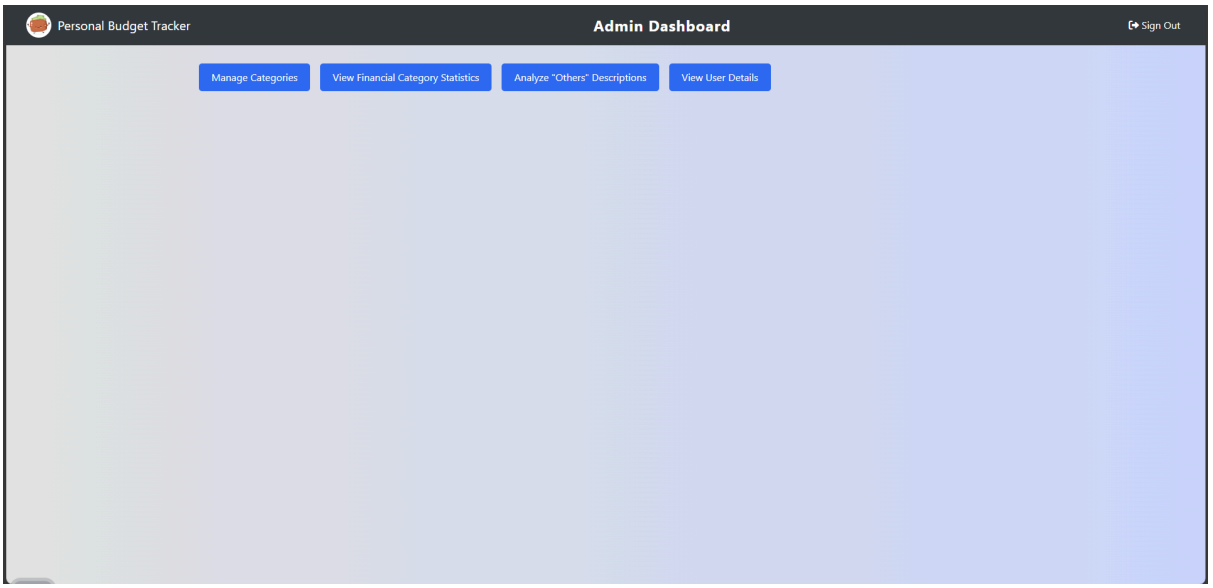
4.3 Sample Screens

<TO DO: Describe and place the screens of subsystem 1 here.>

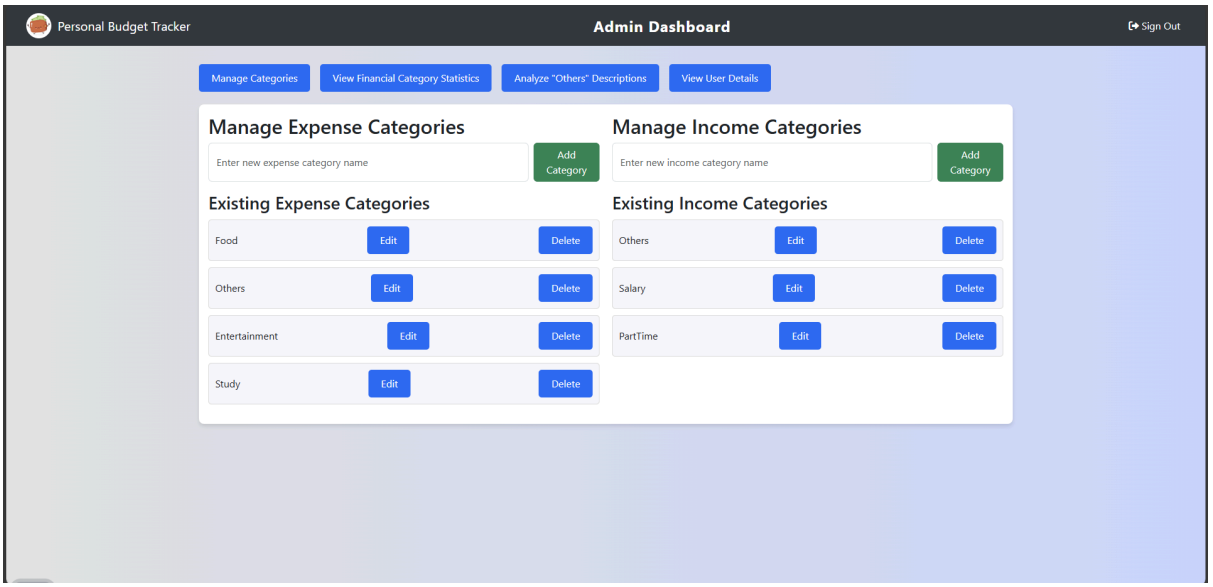
Admin Login Page



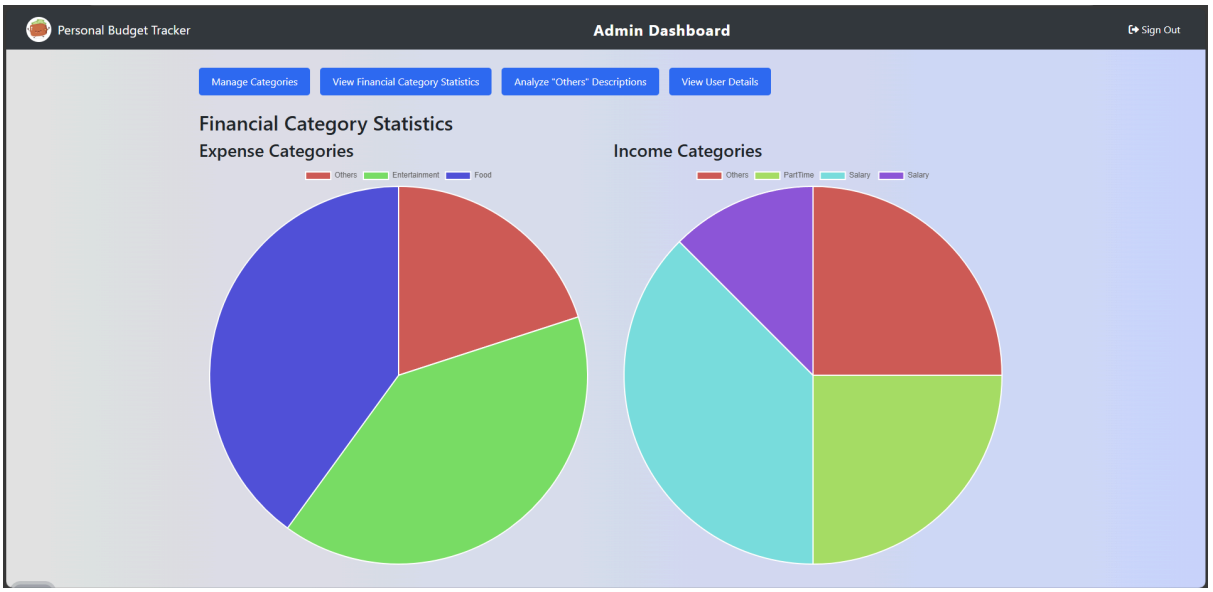
Admin Main Page



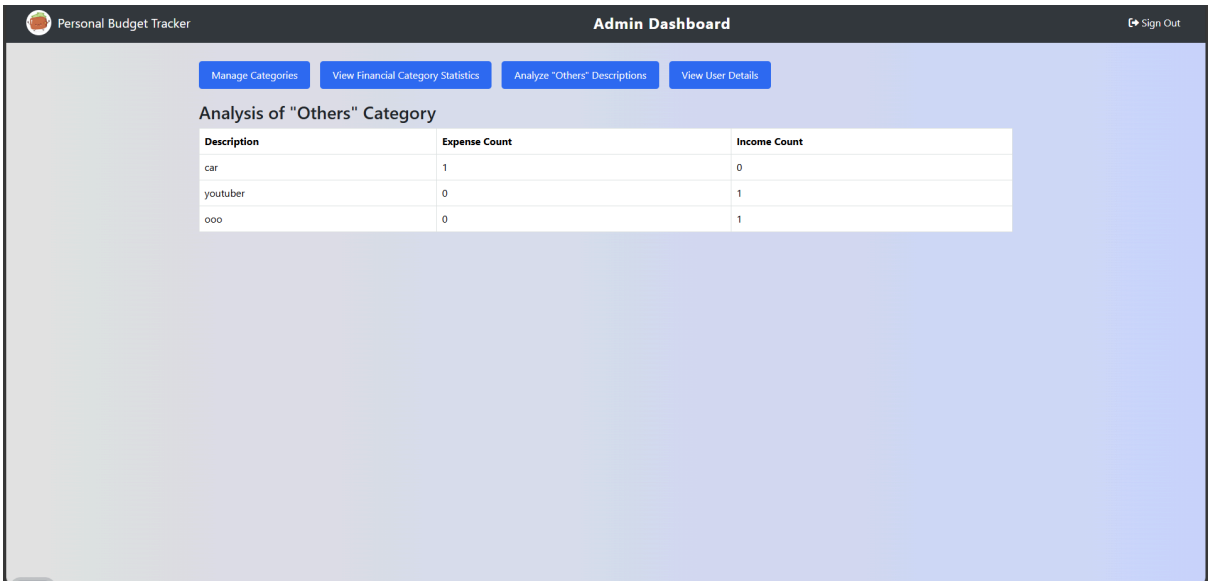
Manage Categories Page



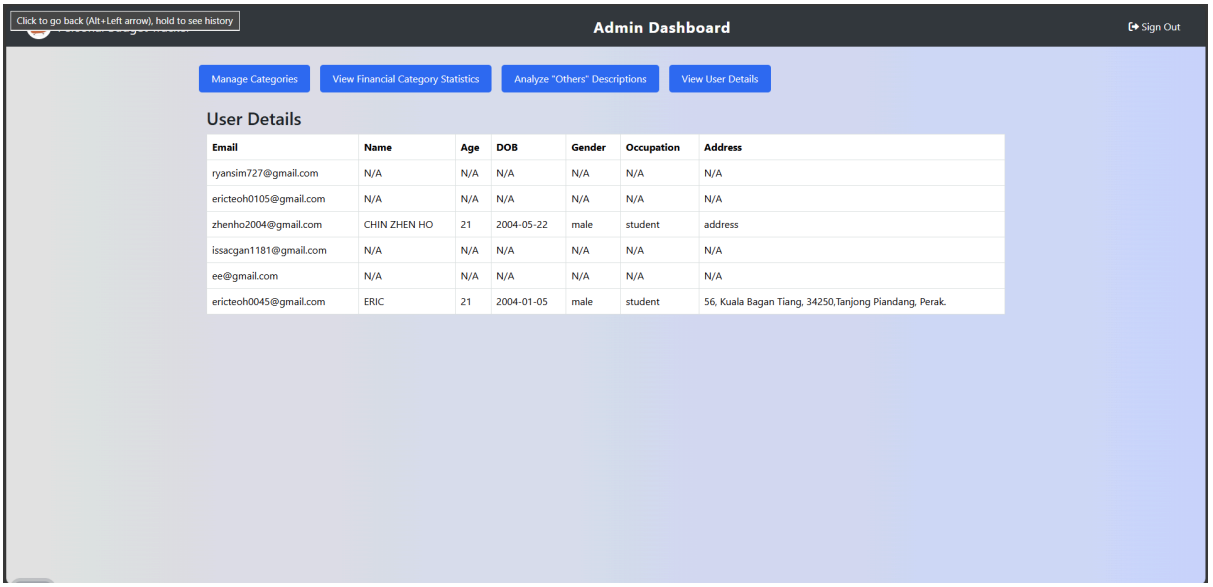
View Financial Category Statistics Page



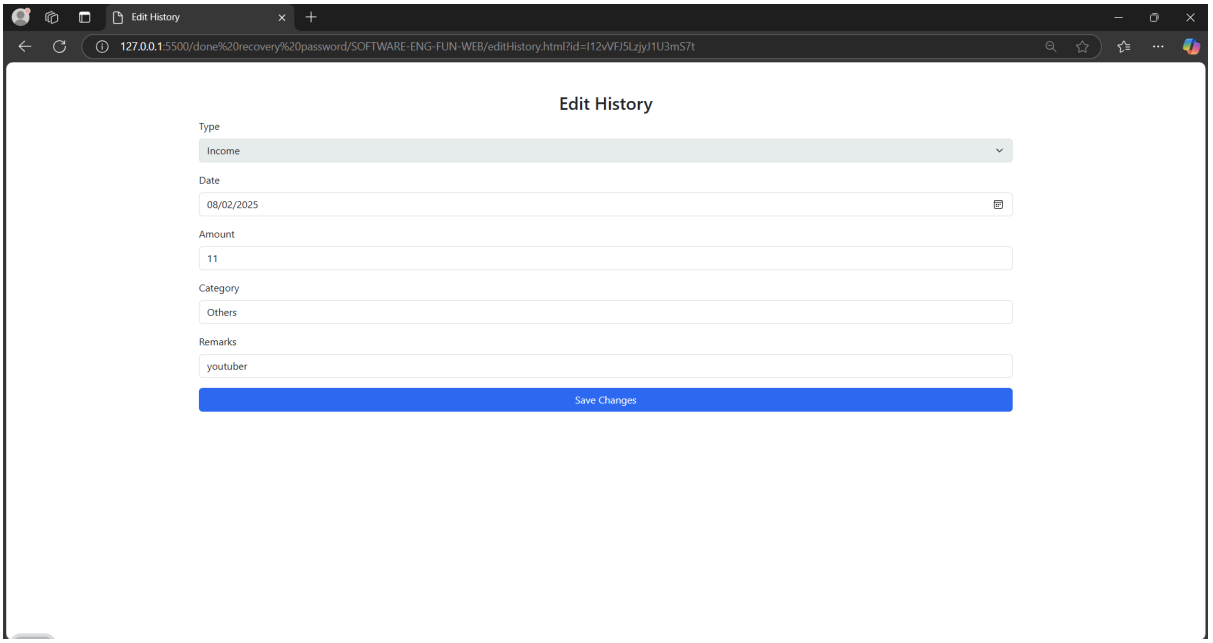
Analyze "Others" Descriptions Page



View User Details Page



Additional Task:Edit History Page


Edit History
127.0.0.1:5500/done%20recovery%20password/SOFTWARE-ENG-FUN-WEB/editHistory.html?id=I12vVFJ5Lzjy/1U3mS7t
Edit History
Type
Income
Date
08/02/2025
Amount
11
Category
Others
Remarks
youtuber
Save Changes

5 Testing

5.1 Test Data

<TO DO: Provide a detailed description of the test data used to verify the subsystem here.>

5.1.1 Admin Test Data

5.1.1.1 Login as admin

Actions	Input	Expected Output	Pass/Fail
Admin login with valid credentials	Correct Email and Password	Go to admin dashboard	Pass
Admin login with invalid credentials	Incorrect Email and Password	“Invalid email or password. Please try again.”	Pass

5.1.1.2 Manage expense categories

Actions	Input	Expected Output	Pass/Fail
Admin input new expense category name and click add category	New expense category name	“Expense category added successfully!”	Pass
Admin did not input new expense category name and click add category	Did not input new expense category name	“Please enter a category name.”	Pass
Admin edit expense category name	New expense category name	“Expense category updated successfully!”	Pass
Admin delete expense category	click “OK”	“Expense category deleted successfully!”	Pass

5.1.1.3 Manage income categories

Actions	Input	Expected Output	Pass/Fail
Admin input new income category name and click add category	New income category name	“Expense category added successfully!”	Pass
Admin did not input new income category name and click add category	Did not input new income category name	“Please enter a category name.”	Pass
Admin edit income category name	New expense category name	“Income category updated successfully!”	Pass
Admin delete Income category	click “OK”	“Income category deleted successfully!”	Pass

5.1.2 Additional Task:View Transaction History(Edit transaction History function)

5.1.2.1 Edit Income or Expense transaction history

Actions	Input	Expected Output	Pass/Fail
User edit income or expense	Enter correct Income or expense detail	Display “History updated successfully!” and History for Income and Expense will be updated.	Pass
User edit income or expense	Missing enter some income or expense detail	Please fill out this field	Pass
User edit income or expense	Enter Incorrect income or expense detail	Please enter a valid value	Pass

5.2 Acceptance Testing

<TO DO: Prepare the acceptance test for the subsystem>

5.2.1 Admin Acceptance Testing

Criteria	Fulfilled?	Remarks
Admin login with valid credentials	Yes	
Admin login with invalid credentials	Yes	
Add,edit,delete expense categories	Yes	
Add,edit,delete income categories	Yes	
View financial expense and income category statistics by chart	Yes	
View expense and income of "Others" category description	Yes	
View user details	Yes	

Date tested : 8/2/2025

% Complete : 100%

Tested by : Gan Shao Yang

Verified by :Chin Zhen Ho

5.2.2 Additional Task:View Transaction History(Edit transaction History function)

Criteria	Fulfilled?	Remarks
User can edit the Transaction History	Yes	Can improve the category load back as drop down menu like add income or expense

Date tested : 8/2/2025

% Complete : 100%

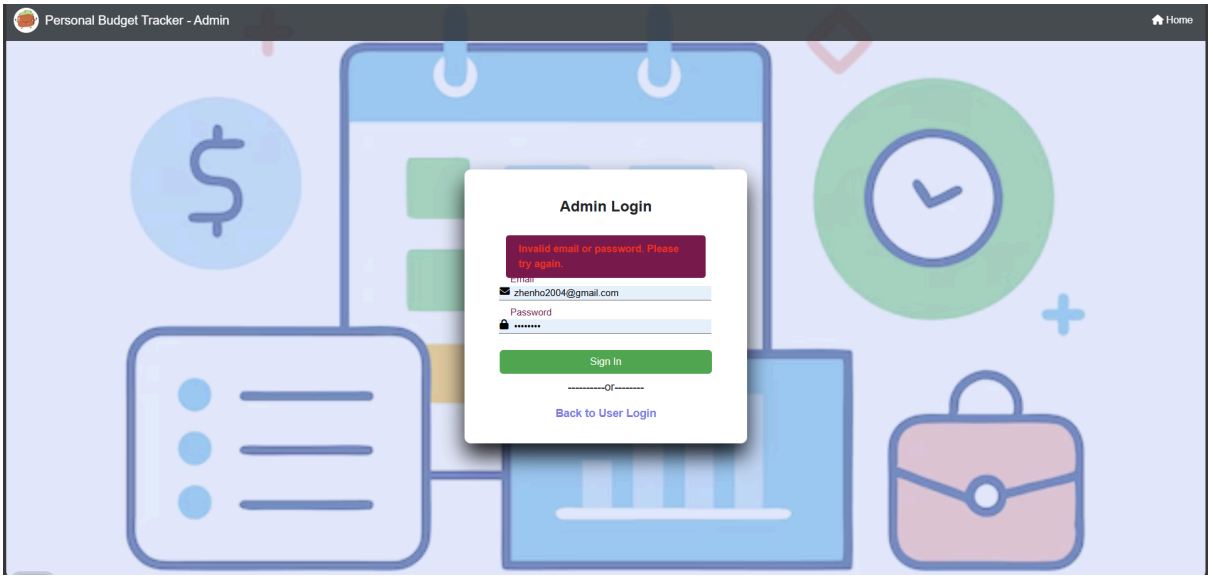
Tested by : BERNARD RYAN SIM KANG XUAN

Verified by : ERIC TEOH WEI XIANG

5.3 Test Results

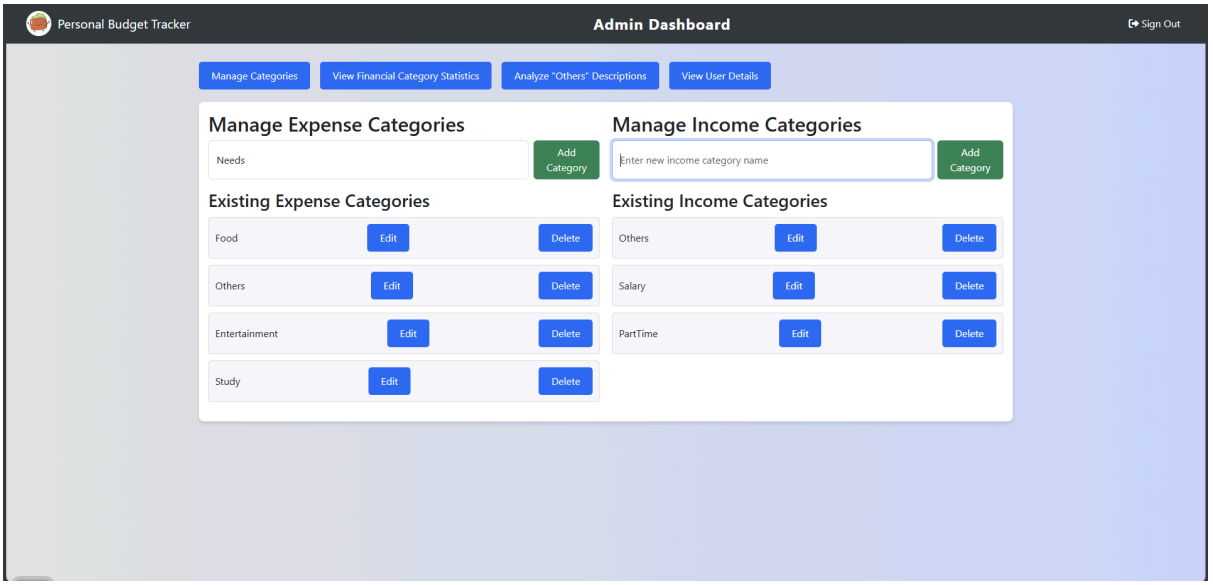
<TO DO: Place the subsystem/application screens here.>

5.3.1 Login as admin

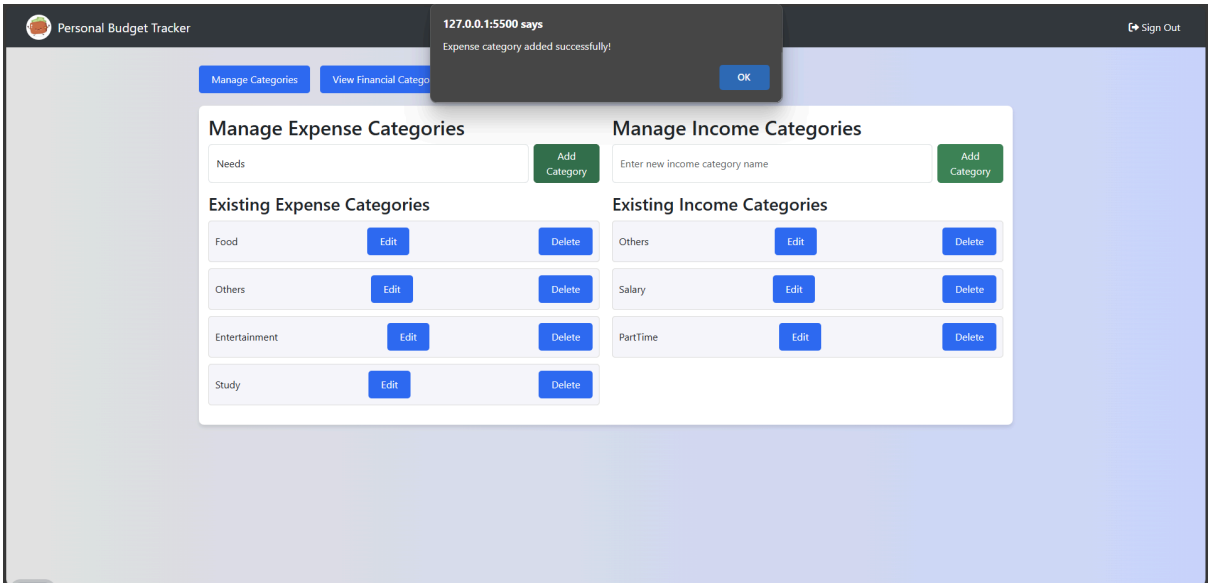


When you enter the correct email and password and click the “Sign In’ button will directly go to the admin dashboard.But when you enter the wrong email or password will show an error message to call you to try again.

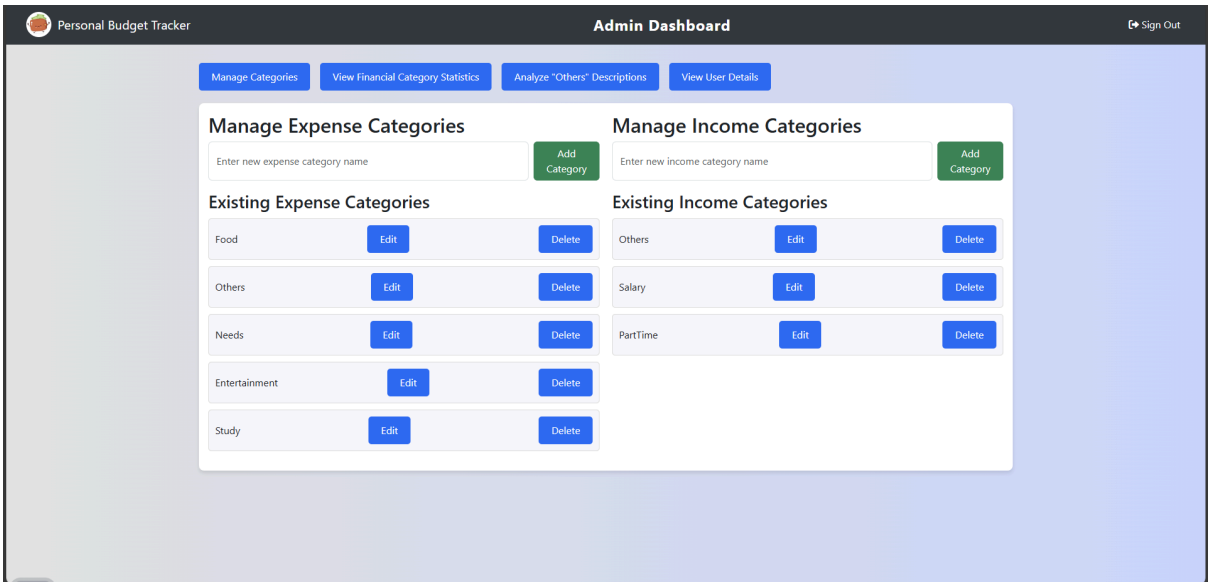
5.3.2 Manage expense categories



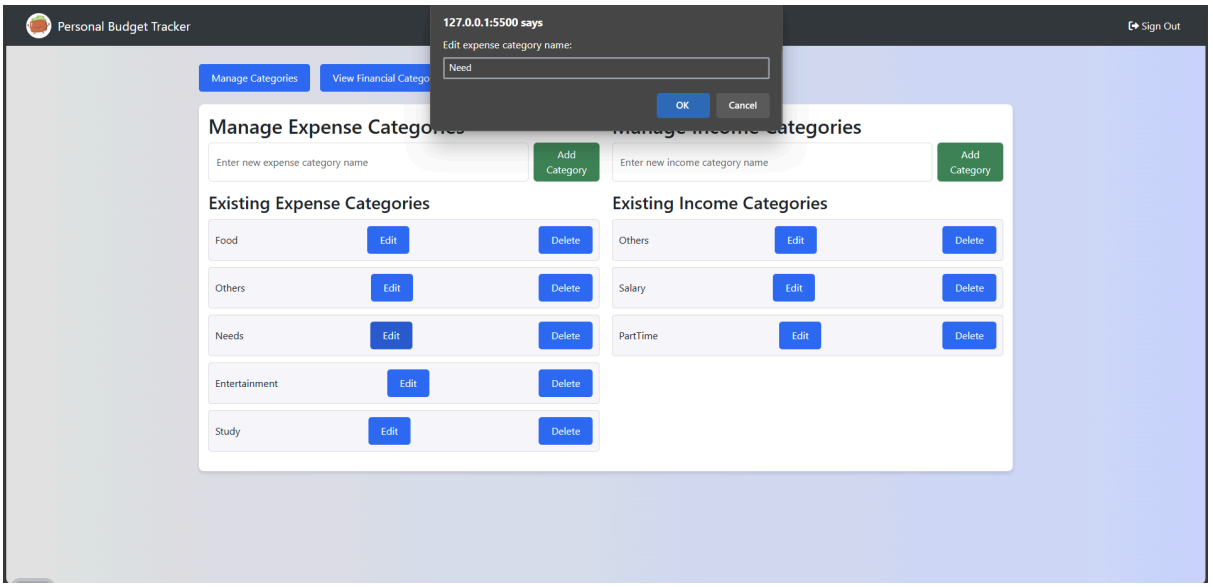
Now, we test the add expense category first,we type the new category name call “Needs” and click the add category button.



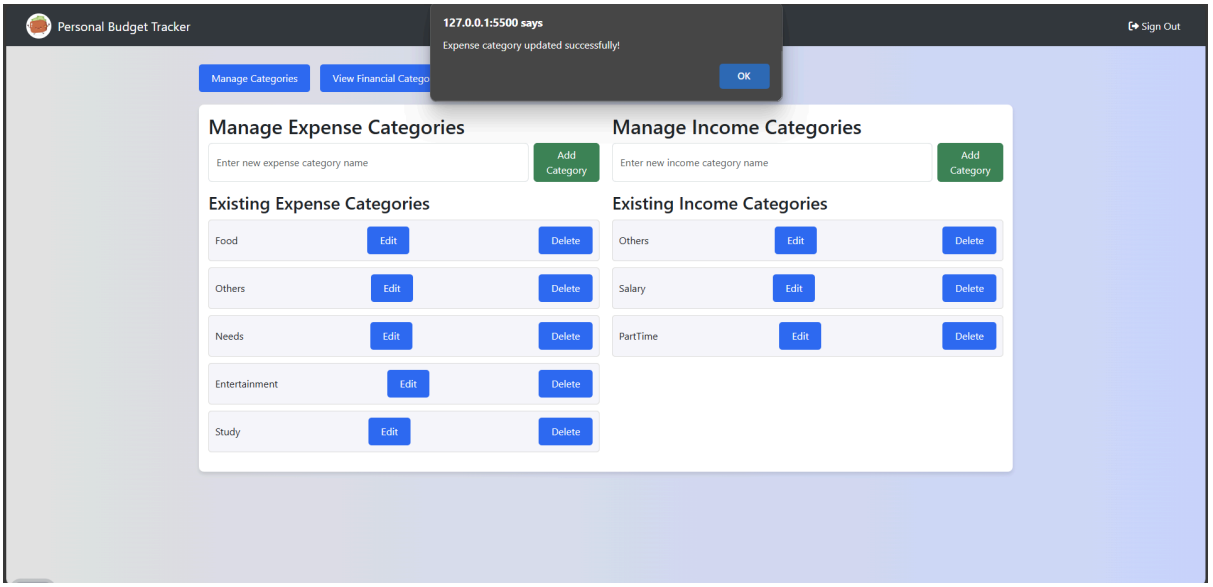
When a category is added successfully,it will pop up a message saying “Expense category added successfully!” and click the “OK” button,the category you added will show at existing expense categories below.



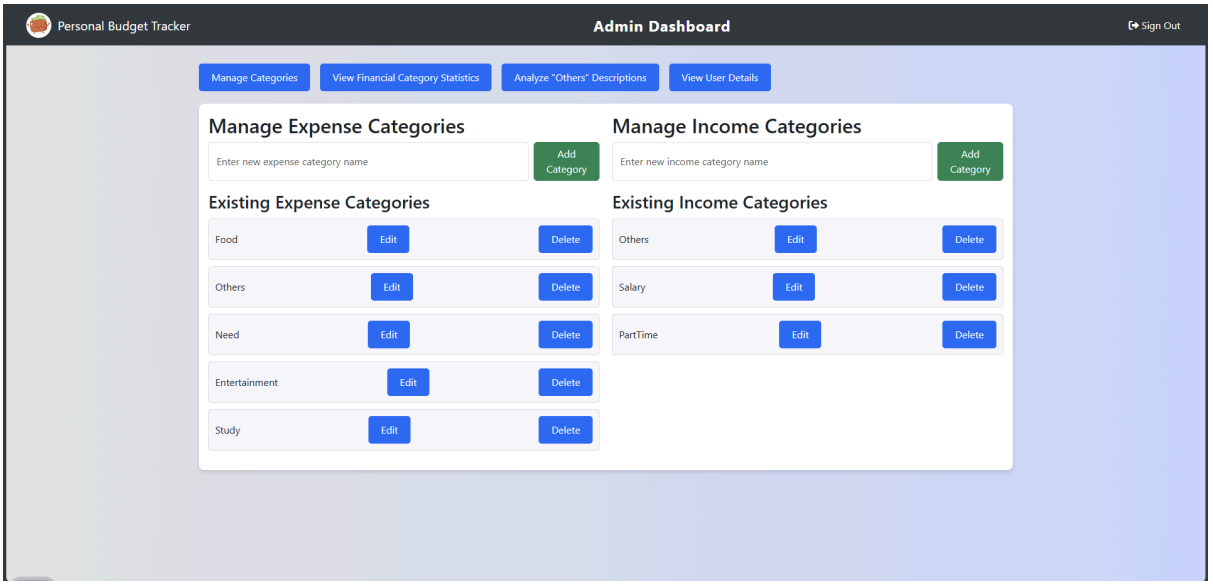
The category you added is shown at existing expense categories below.



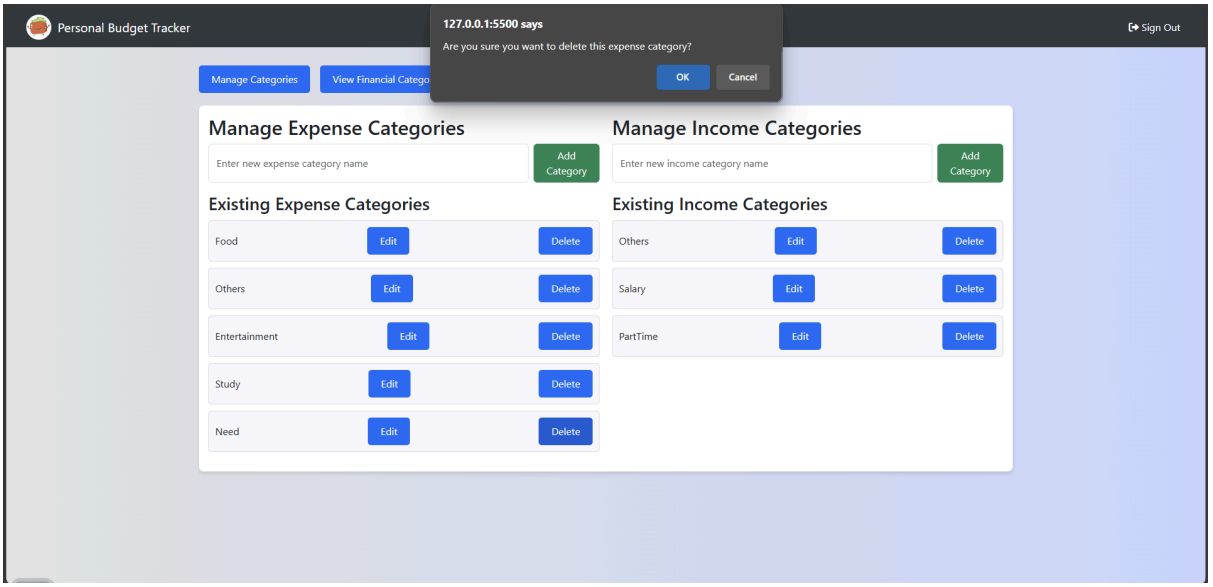
Now,we test the edit expense category,when you click the “Edit” button it will pop up a input field to let you edit category name,now we test to edit it from “Needs” to “Need”.So we need to type “Need” in the input field and click “OK” button.



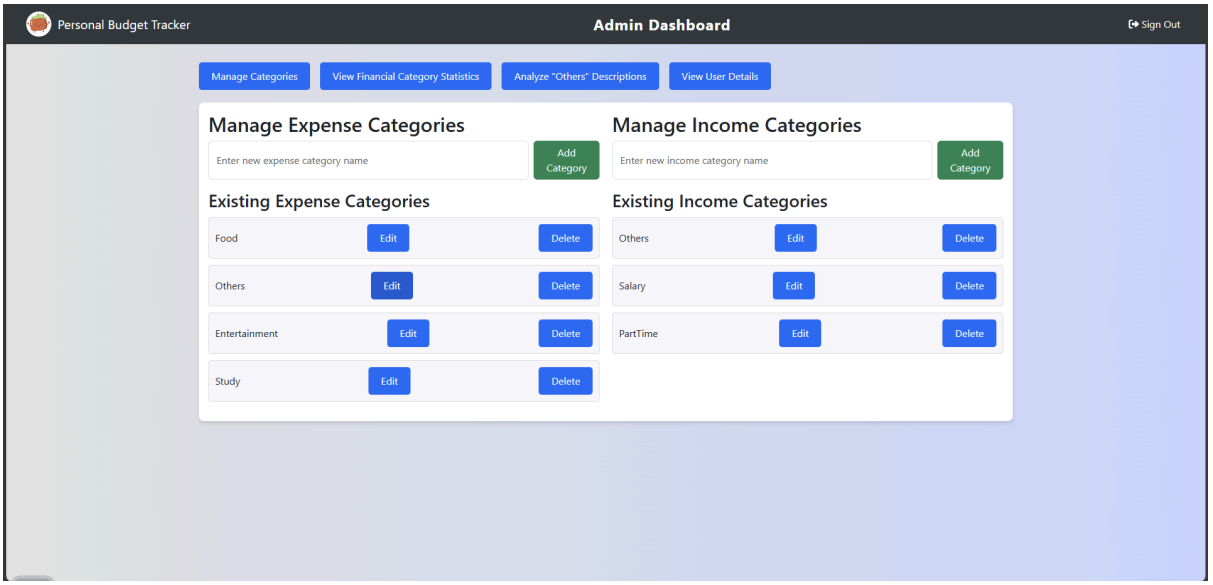
After editing successfully it will show a successful edit message,and when you click the “OK” button,the updated category will be directly shown at existing expense categories below.



After you click the “OK” button,you can see the “Needs” category is updated to “Need”.

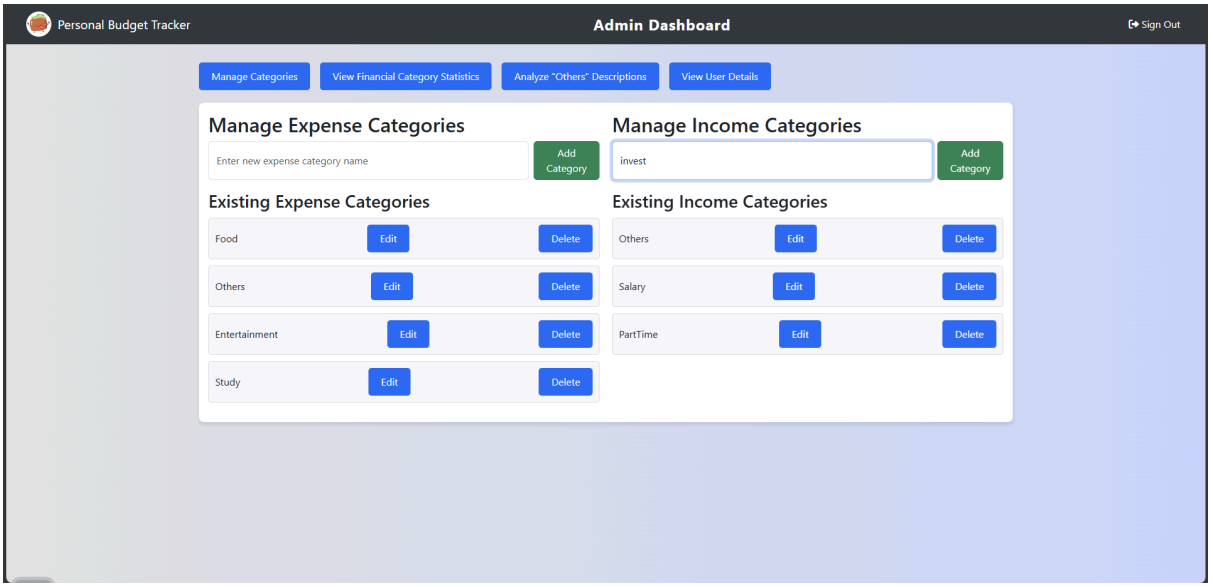


Now we test to delete the expense category just now we update which is “Need”,we click the delete button and it will show a message to ask you “Are you sure you want to delete this expense category?”,and then you click “OK”,it will show a message saying “Expense category deleted successfully!”,and then you click the “OK” button again,the “Need” category in existing expense categories will be removed.

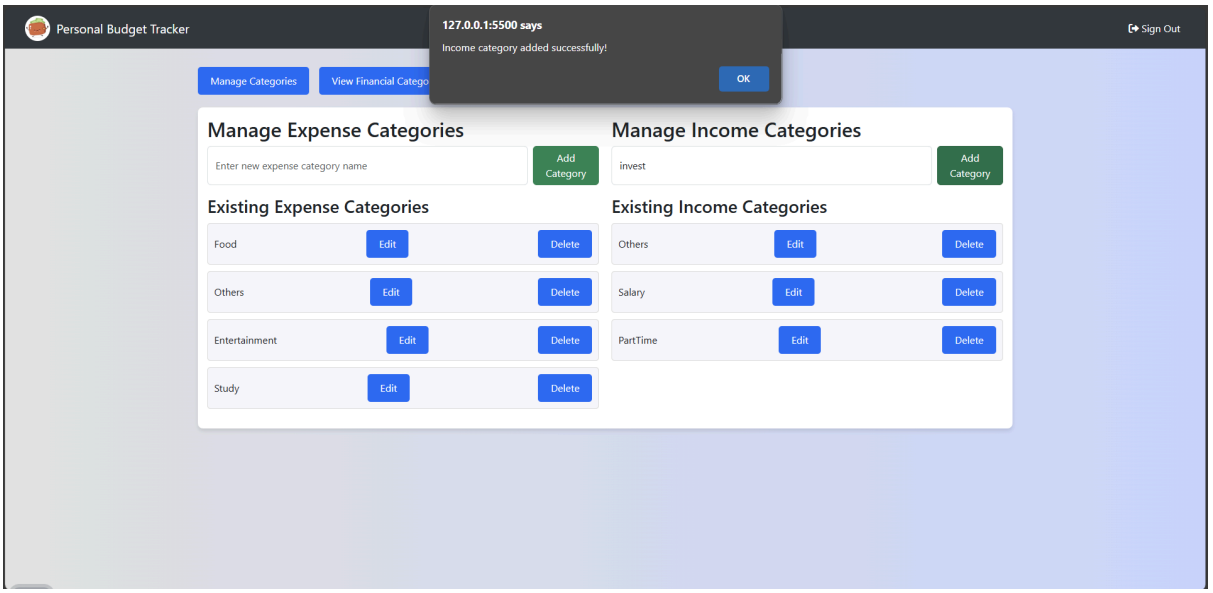


Now the “Need” category is removed.

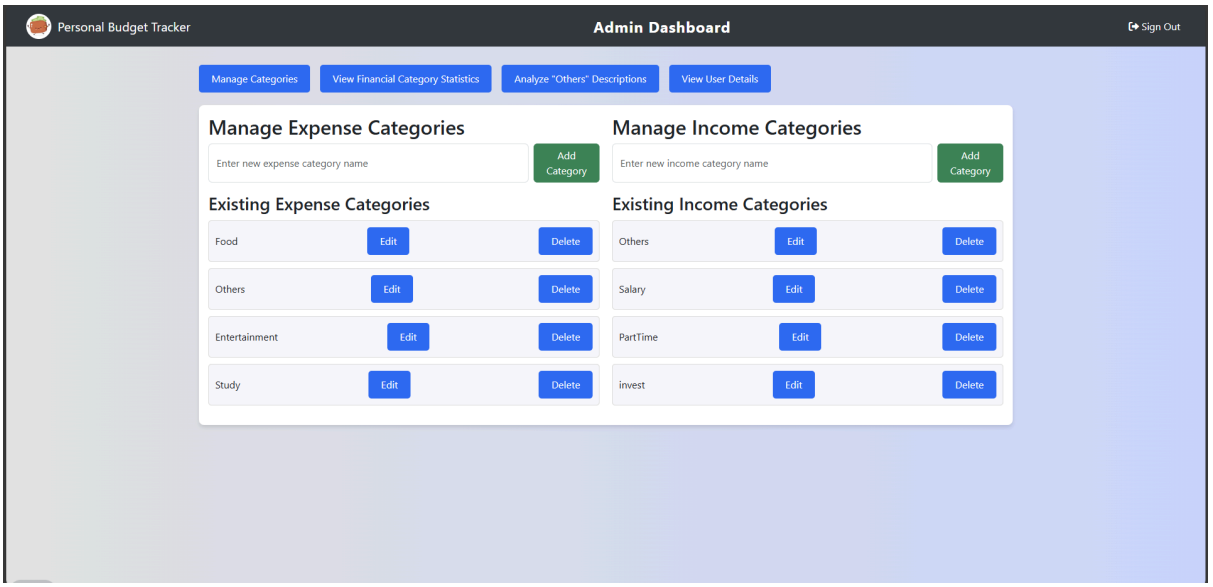
5.3.3 Manage income categories



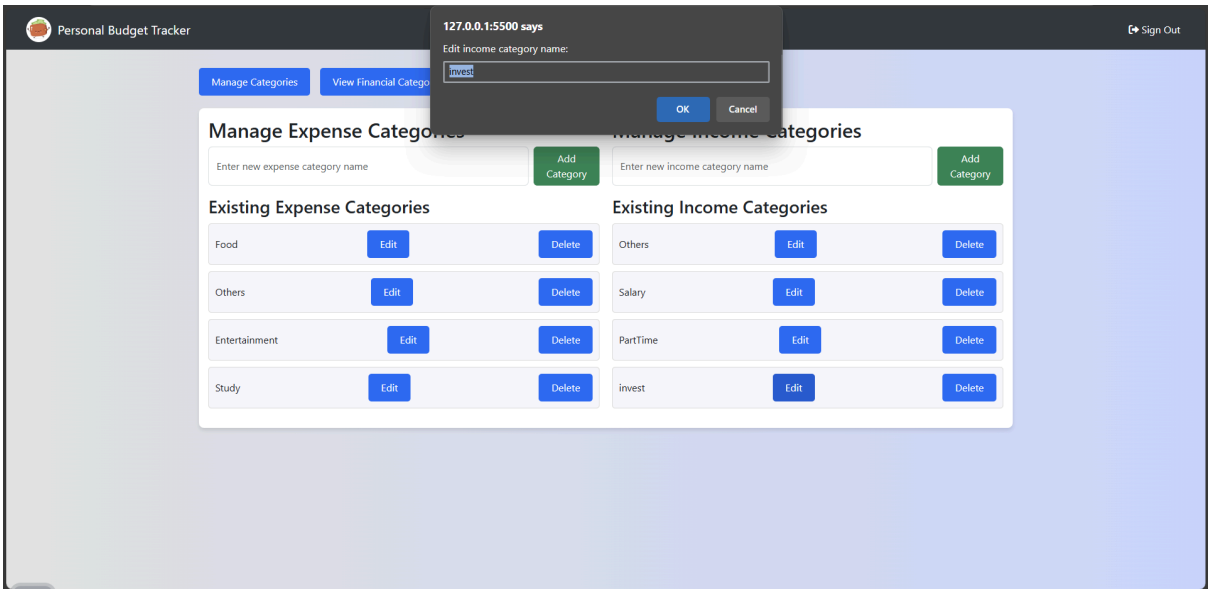
Now we test the add income category,we can input the new category name called “invest” into the input field and click the add category button.



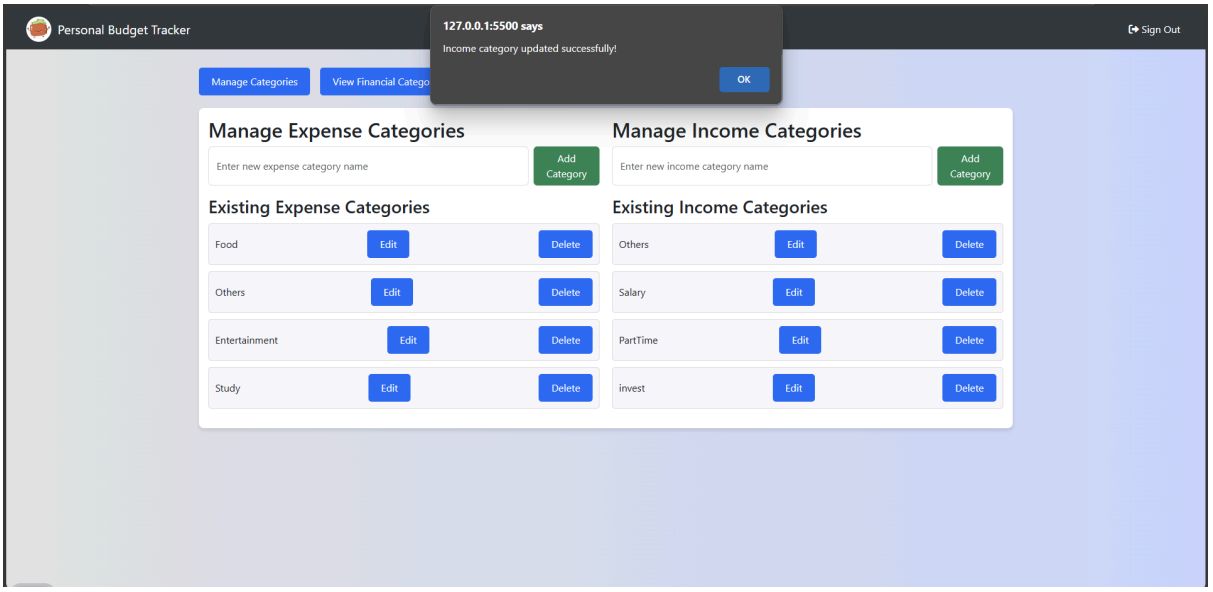
After adding successfully,it will show a message saying “Income category added successfully!”,and you click the “OK” button,it will update directly at existing income categories below.



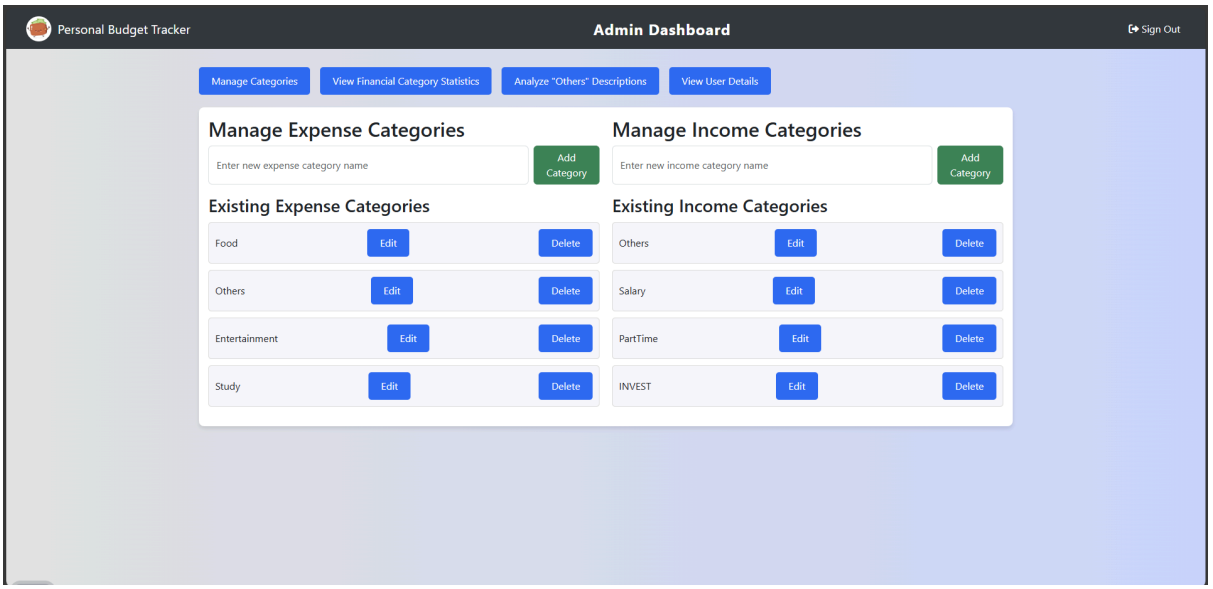
Now the income category just now we added is shown in the existing income categories.



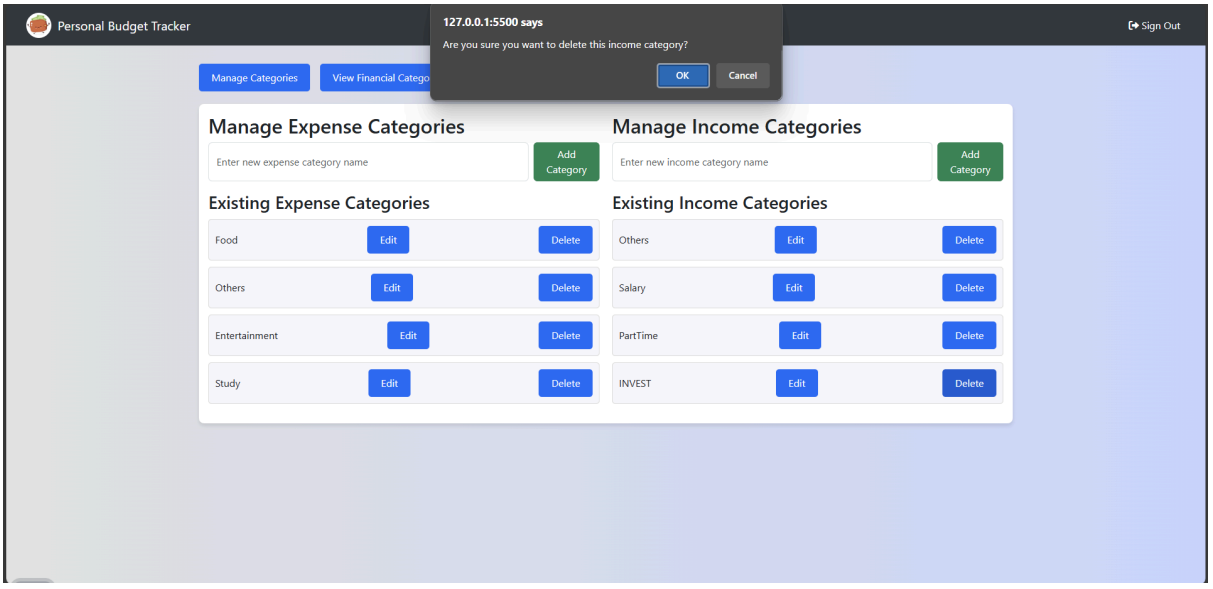
Now we test to edit category call “invest”,we click the “Edit” button, it will pop up a input field and you can edit it,now we edit it from “invest” to “INVEST” and click “OK”button.



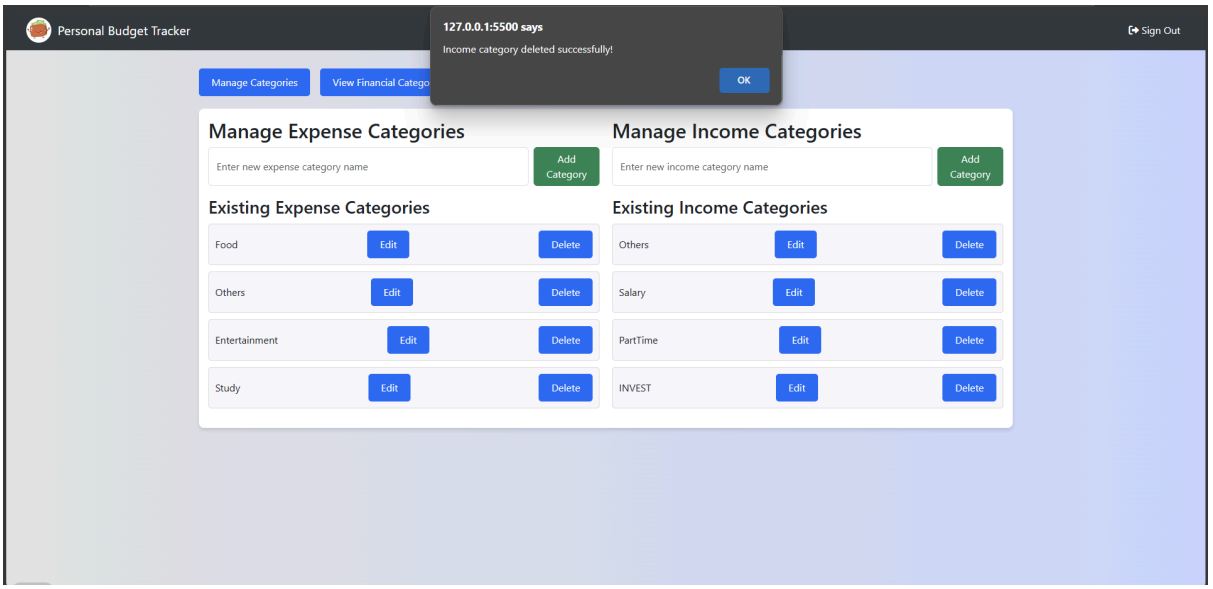
After editing successfully it will show a successfully edited message,and you click “OK” button.



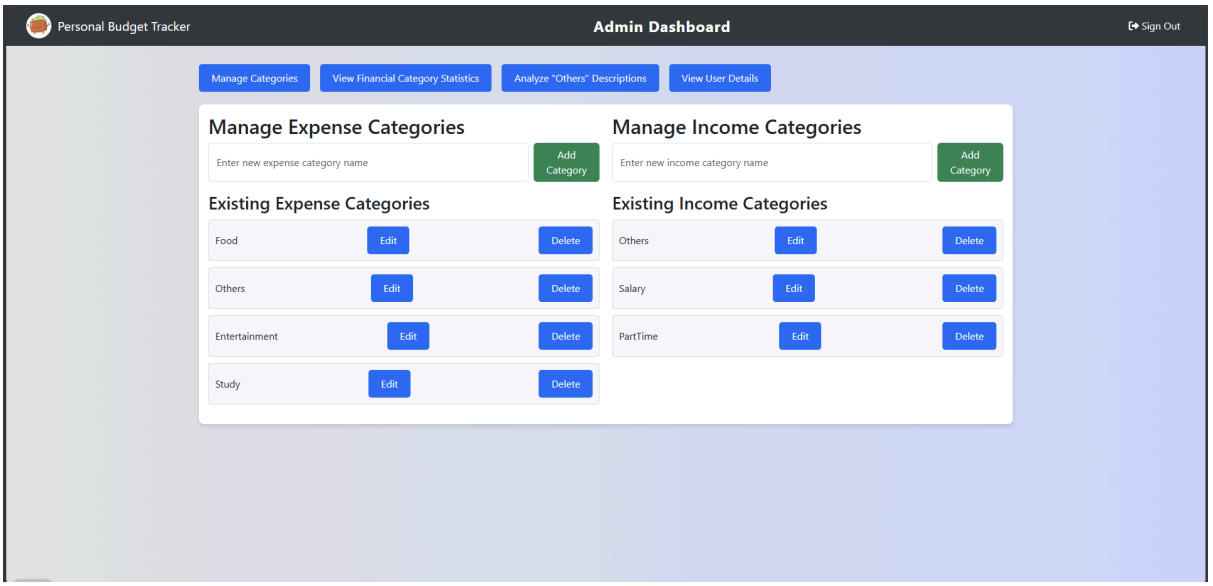
After you click “OK”button ,the category “INVEST” just now you edited is updated in existing income categories.



Now we test to delete the “INVEST” category,we click the “Delete” button and it will pop up a message to ask you,“Are you sure you want to delete this income category?”,and then you click the “OK” button.



Now the “INVEST” category is deleted,it will show a successful deleted message.



After that,you click the “OK” button,the category called “INVEST” will be removed in the existing income categories.

6 Conclusion

<This section should include details of completion of software, software quality assurance, group collaboration, problems encountered>

6.1 Project Achievements

This project has achieved most of its objectives and demonstrated the successful integration of key technologies. It has effectively implemented the functions and interactive systems we aimed for, utilizing HTML, CSS, and JavaScript as the core programming languages to ensure a well-structured, visually appealing, and responsive user interface. Additionally, Firebase served as the main database, providing real-time storage, authentication, and other essential services, enhancing both performance and user experience. The use of VS Code facilitated efficient debugging and version control, streamlining the development process. While some features, such as real-time messaging, were not fully implemented, the project successfully combined front-end development with backend services, resulting in a scalable and user-friendly web application.

6.2 Software quality assurance

Ensuring software quality was a crucial aspect of this project to guarantee reliability, performance, and usability. This was achieved through extensive testing and debugging at various stages of development. The project underwent functional testing to verify that all implemented features, such as user authentication and UI responsiveness, worked as intended. Additionally, usability testing was conducted to ensure a seamless and intuitive user experience. To maintain code quality, VS Code's debugging tools were used extensively, along with Firebase's built-in analytics and error-tracking capabilities. Version control with Git allowed for systematic updates, minimizing errors and ensuring a stable development process. Although some planned features were not fully implemented, the project maintained a high standard of quality, resulting in a functional, scalable, and user-friendly web application.

6.3 Group Collaboration

Our leader, Chin Zhen Ho, played a crucial role in this assignment by keeping the team organized, assigning tasks, and ensuring smooth coordination. He was also responsible for reviewing the final code and report, making sure everything was in order before submission.
Eric Teoh contributed significantly to the project by writing a large portion of the code and solving complex issues that others encountered. In addition, he kept track of the team's progress to ensure that all tasks were completed on schedule.
Our other team members, Gan Shao Yang and Bernard, also played important roles in coding, consistently completing assigned tasks to keep the project on track. They actively participated in discussions, offering valuable suggestions during meetings.
The documentation was a shared responsibility among all team members, including drawing diagrams, writing reports, and documenting progress. Through this collective effort, the project was successfully completed.

6.4 Problems Encountered

Of course, during development, we encountered several challenges that we were unable to fully overcome. One major issue was real-time messaging, which was intended to allow advisors or admins to communicate directly with users instead of relying on emails or messages. This feature was meant to facilitate easier access to user data for advisors. However, implementing this functionality required access to a paid service, which was beyond the project's limited budget, preventing us from integrating it.

Another set of challenges arose in the transaction history feature. The first issue occurred when users attempted to edit transaction details—the category selection, which was originally a drop-down menu, unexpectedly changed to a text-based input. Additionally, another problem surfaced when users added too many income or expense entries. If they did not click the history button again, it sometimes led to unintended duplication of records.

Despite these challenges, we managed to implement alternative solutions and ensured that the core functionalities of the project remained intact.