INTERIM REPORT

EXPENSE TRACKER

Dissertation submitted in fulfilment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

– DATA SCIENCE WITH MACHINE LEARNING

By

JERARD M FRANCIS

Registration No: 12310619

Section: K23UP

Roll No: 41

15/11/2024

Supervisor

Aman Kumar

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

October 2024

ACKNOWLEDGEMENT

I at this moment declare that the research work reported in the dissertation/dissertation proposal entitled “EXPENSE TRACKER” in partial fulfillment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under the supervision of my research supervisor Mr. Aman Kumar. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith directly complies with Lovely Professional University’s Policy on plagiarism, intellectual property rights, and the highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents an authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

JERARD M FRANCIS

REG.NO - 12310619

Table of Contents

I . INTRODUCTION

1.1 Project Overview

The **Expense Tracker** is a Python-based application designed to simplify the management of personal and household finances. Built using the Tkinter GUI library, this system enables users to effortlessly input, view, categorize, and manage their daily expenses. The application provides a comprehensive platform for tracking expenses such as groceries, utilities, transportation, and other categories, allowing users to maintain control over their spending habits.

With a clean and user-friendly interface, the system allows users to add new expense entries, view all records in a tabular format, calculate the total expenditure, and delete unnecessary or outdated entries. Additionally, the system provides optional features like saving and loading data from files, ensuring data persistence. The Expense Tracker is a practical and effective tool for individuals seeking to enhance their financial discipline and manage their budgets efficiently.

1.2 Purpose and Significance

The purpose of this project is to demonstrate an efficient system for managing personal finances, simplifying the process of recording, categorizing, and analyzing daily expenses. Effective expense tracking is essential for understanding spending habits, setting budgets, and achieving financial goals. This project illustrates how a Python-based solution, leveraging a GUI, can reduce the manual effort and inaccuracies often associated with traditional pen-and-paper or spreadsheet methods of financial management.

The significance of this system lies in its ability to automate and streamline the tracking of expenses. By centralizing all financial records in one user-friendly platform, the **Expense Tracker** ensures that users can add, review, and manage their expenses quickly and efficiently. This approach not only saves time but also enhances financial discipline and awareness, making it a valuable tool for individuals and households. Furthermore, the application provides a foundation for future expansions, such as integrating data visualization or generating detailed spending reports, making it adaptable to evolving user needs.

1.3 Registration Details

To efficiently manage expenses, this project includes the following key features and registration specifics:

1) **Expense Entity Structure**: Each expense is represented by a structure with the following attributes:

- Expense ID: A unique identifier for each expense to facilitate easy access and management of individual entries
- Expense Name: A descriptive field to identify the nature of the expense (e.g., groceries, utilities, transportation).
- Amount: The monetary value associated with the expense.
- Category: A dropdown field to specify the type of expense (e.g., Food, Transportation, Utilities, Entertainment).
- Date: The date when the expense was incurred, used for organizing expenses by time period.
- Description: An optional text field for adding additional details about the expense, such as vendor or purpose.

2 **Data Storage**: Expense records are stored in a list within the application, where each expense is an instance of the **Expense** class. Currently, the system uses an in-memory list for storing data. However, future iterations could integrate a database (like SQLite or MySQL) for persistent storage across sessions.

3 **Expense Management System**: The project includes an ExpenseTrackerApp class that facilitates the management of expenses. This class provides methods for:

- Adding New Expenses: Users can add new expense entries by entering details through a GUI form.
- Updating Expense Records: Users can modify existing expense details, including correcting information or updating amounts.
- Searching for Expenses: Users can search for expenses by category, name, or date range.
- Deleting Expenses: Specific expense records can be deleted by selecting them from the list.
- Displaying All Expenses: A table displays all recorded expenses in a structured format for easy viewing and management..

II. Objectives and Scope of the Project

2.1 Project Objectives

The primary objective of this project is to design and implement an Expense Tracker application that efficiently manages personal finances using Python and Tkinter. The system aims to provide a user-friendly interface for tracking, categorizing, updating, and analyzing daily expenses. Specific objectives include:

1. Develop a Comprehensive Expense Management System:
    - Create a system that stores and manages various aspects of financial data, including expense name, amount, category, date, and description.
    - The system should allow users to efficiently add new expenses, update existing records, search for specific expenses, and delete records when necessary.

2. Implement an Organized Data Storage Solution:
    - Use Python's in-memory list data structure to store and manage expense records.
    - Future iterations can explore integrating a database (e.g., SQLite) for persistent storage and scalable data management.

3. Demonstrate Efficient Data Management and Interaction:
    - Showcase the system's functionality by allowing users to interact with expense data through a GUI.
    - Implement features for adding, searching, updating, and deleting expense records.
    - Provide a user-friendly way to display all expense records in a table format for easy viewing.

4. Highlight the Significance of Financial Management:
    - Illustrate the importance of tracking expenses and how it helps users stay within budgets, save money, and make informed financial decisions.
    - Provide insights into how the system can reduce manual record-keeping errors and improve financial awareness.

5. Design a Scalable and Extendable Solution:
    - Create a flexible architecture that allows for future enhancements, such as adding features like budgeting, financial reports, or integration with other financial tools.

2.2 Project Scope

The scope of this project is focused on the design, development, and demonstration of an **Expense Tracker** in Python, with the following key areas of focus:

1. **Expense Entity Structure:**
    - Define an **Expense** class with attributes such as expense name, amount, category, date, and description. This class will serve as the foundational data structure for managing expense records.

2. **In-Memory Data Storage:**
    - Store expense data in a list of **Expense** objects. Each expense record is an instance of the **Expense** class, providing easy access to and management of individual financial information.
    - This approach is suitable for demonstration purposes and smaller datasets. Future improvements could involve transitioning to a database for persistent storage.

3. **Expense Record Management Operations:**
    - Implement the ability to add, update, search, delete, and display expense records.
    - Provide functionalities to retrieve expense data by category, amount, or date, update any expense details, and delete expense records when needed.

4. **Graphical User Interface (GUI):**
    - Use Tkinter to build a user-friendly GUI for interacting with the system.
    - The GUI allows users to enter expense information, display records in a table format, and perform actions like searching, updating, and deleting records.

5. **Limitations:**
    - The system currently focuses on basic expense record management and does not support advanced features like financial reports, budget tracking, or integration with bank account systems.
    - Data persistence is limited to in-memory storage, with no database integration at this stage. This means that data will be lost once the application is closed.
    - Security measures for sensitive financial data (such as data encryption) are not implemented in this version.

III. Application Tools

3.1 Software Applications

The development of the Expense Tracker project utilizes several software tools to ensure an efficient and organized development environment. These tools are as follows:

1. Python (Version 3.x):
    - Python is the core language used for implementing the system. It is a versatile, easy-to-learn, and powerful language that provides extensive libraries, including Tkinter for building the graphical user interface (GUI). Python's simplicity and readability make it an ideal choice for this project, as it allows for rapid development and easy maintenance.

2. Integrated Development Environment (IDE) - Visual Studio Code (VS Code):
    - VS Code is used for writing, debugging, and testing Python code. It provides a rich development environment with features such as syntax highlighting, code linting, version control integration, and debugging tools, which make the development process smoother and more efficient.

3. Git and GitHub:
    - Git is used for version control, while GitHub serves as a platform for code management and collaboration. Git tracks code changes, making it easy to manage different versions of the project. GitHub facilitates sharing and collaboration, allowing multiple developers to work on the project simultaneously and keeping the codebase synchronized.

4. Tkinter:
    - Tkinter is used for building the graphical user interface (GUI). It provides a simple and effective way to create windows, buttons, labels, text boxes, and other GUI elements, making the system user-friendly and accessible for users.

5. SQLite (Optional):
    - While the current version of the project uses in-memory storage for expense records, SQLite can be integrated in future versions for persistent data storage. SQLite is a lightweight, serverless relational database that is easy to use with Python through the sqlite3 module.

3.2 Programming Languages of the Project

The project is primarily implemented in Python, chosen for its simplicity, readability, and support for rapid development of desktop applications.

1. Python:
    - Python serves as the core language for implementing the Expense class and the ExpenseTrackerApp class. The Expense class encapsulates expense data, while the ExpenseTrackerApp class provides methods for managing and processing records.
    - Python's extensive standard libraries, such as Tkinter for GUI development and sqlite3 for optional database integration, make it a robust choice for this project.

2. SQLite (Optional):
    - SQLite can be used in future versions to store expense data persistently. The sqlite3 library in Python allows for database integration, enabling the system to store, retrieve, and update expense records even after the application is closed, making the system more scalable.

lV . Project Structure of the Student Management System (SMS)

The Expense Tracker is organized into several components, classes, and functions, each responsible for managing specific aspects of the system. The architecture of the system ensures modularity, maintainability, and scalability.

4.1 Main Components of the Project

1. Expense Entity (Expense Class):
    - Purpose: Represents individual expense data.
    - Responsibilities: Stores expense details such as category, amount, date, description, and payment method.

2. Expense Tracker System (ExpenseTrackerApp Class):
    - Purpose: Handles the logic for managing expense records.
    - Responsibilities: Provides methods for adding, updating, searching, deleting, and viewing expense records.

3. Graphical User Interface (GUI) (Tkinter):
    - Purpose: Provides an interface for the user (e.g., individual users, admins) to interact with the system.
    - Responsibilities: Allows the user to input, update, search, and delete expenses, as well as view summary reports.

4. Storage (In-memory or SQLite):
    - Purpose: Stores expense records for retrieval and updating.
    - Responsibilities: The system uses in-memory storage (a list of Expense objects) in the initial version. In future versions, this can be extended to use SQLite for persistent storage.

4.2 Classes and Their Functions

1. Expense Class

The Expense class is the fundamental building block of the system, representing each individual expense record.

- Attributes:
    - category: The category of the expense (e.g., groceries, entertainment, utilities).
    - amount: The monetary value of the expense.
    - date: The date when the expense occurred.
    - description: A brief description of the expense.
    - payment_method: The method used for payment (e.g., cash, credit card).

- Methods:
    - __init__(self, category, amount, date, description, payment_method): Constructor to initialize an expense's attributes.
    - __str__(self): Method to return a string representation of the expense details for easy display.

2. Expense Tracker System Class

The ExpenseTrackerApp class is responsible for managing the collection of expense records. It handles operations such as adding, updating, searching, deleting, and displaying expense records.

- Attributes:
  - expenses: A list (or a database connection if SQLite is implemented) that holds all expense objects.

- Methods:
  - add_expense(self, expense): Adds a new expense object to the expenses list.
  - update_expense(self, expense_id, new_data): Updates the details of an existing expense by its ID.
  - delete_expense(self, expense_id): Deletes an expense record by its ID.
  - search_expense(self, category): Searches for expenses by category and returns the matching records.
  - get_all_expenses(self): Returns a list of all expenses stored in the system for display or report generation.
  - generate_report(self): Generates a summary report showing the total expenses and breakdown by category.
  - save_to_database(self): (Optional) Saves the expense records to a persistent database like SQLite.

3. GUI Class (Tkinter Interface)

The GUI component is built using Python's Tkinter library. It provides the user interface for interacting with the system.

- Main Components of GUI:
  - Expense Form: A form that allows the user to input expense details (category, amount, date, description, etc.).
  - Buttons: Buttons for adding, updating, searching, deleting expenses, and generating reports.
  - Display Area: A section of the window that displays the list of expenses or their details after a search or action is performed.

- Main Functions:
  - add_expense_gui(self): Captures data from the user input form and calls the add_expense method from the ExpenseTrackerApp class to add the new expense.
  - update_expense_gui(self): Retrieves the data from the input form and updates an existing expense by calling the update_expense method.
  - search_expense_gui(self): Accepts a category or date from the user and searches for the expense using the search_expense method.
  - delete_expense_gui(self): Allows the user to delete an expense record by calling the delete_expense method.
  - generate_report_gui(self): Generates a report by calling the generate_report method from the ExpenseTrackerApp class and displays it in the GUI.

4.3 Interaction Between Components

1. Adding a New Expense:
   - The user fills out the expense form in the GUI.
   - The form data is passed to the add_expense_gui() method in the Tkinter interface.
   - The Tkinter interface then calls the add_expense() method of the ExpenseTrackerApp class, passing a newly created Expense object.
   - The expense is added to the expenses list in the ExpenseTrackerApp class, effectively storing the record.

2. Updating Expense Records:
    - The user selects an expense to update in the GUI.
    - The new data is entered in the input form and passed to update_expense_gui().
    - The Tkinter interface calls the update_expense() method of the ExpenseTrackerApp class to modify the existing expense's record based on the expense ID.

3. Searching for Expenses:
    - The user enters a category or date in the search field of the GUI and clicks the search button.
    - The Tkinter interface calls the search_expense_gui() method, which in turn calls search_expense() from the ExpenseTrackerApp class to retrieve the matching expense object.
    - The expense details are displayed on the GUI.

4. Deleting an Expense:
    - The user enters an expense ID in the GUI and clicks the delete button.
    - The Tkinter interface calls the delete_expense_gui() method, which calls delete_expense() from the ExpenseTrackerApp class to remove the expense record.

5. Generating Reports:
    - The user clicks the report button in the GUI.
    - The Tkinter interface calls the generate_report_gui() method, which invokes the generate_report() method in the ExpenseTrackerApp class to generate a report based on the expense data.
    - The report is displayed in the GUI for the user to review.

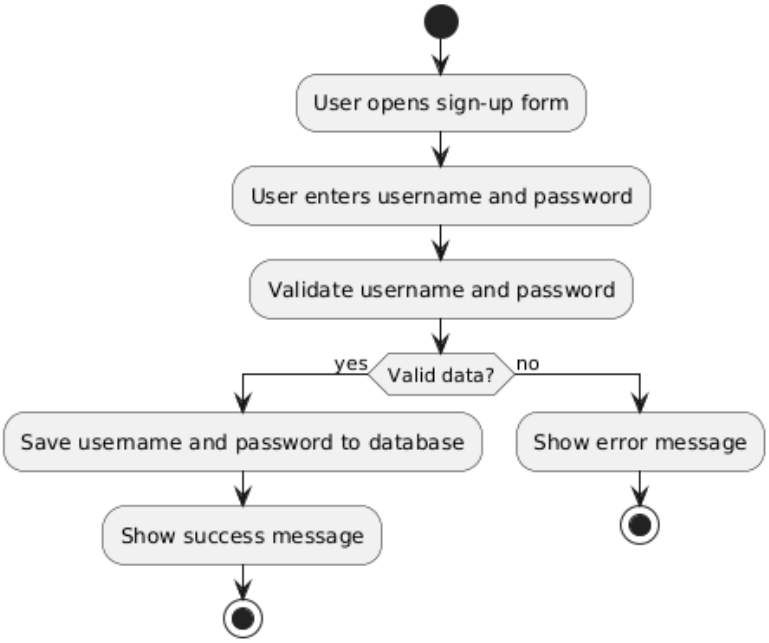
VI. Flowchart or Algorithm of the Project

Algorithm of the Expense Tracker Project:

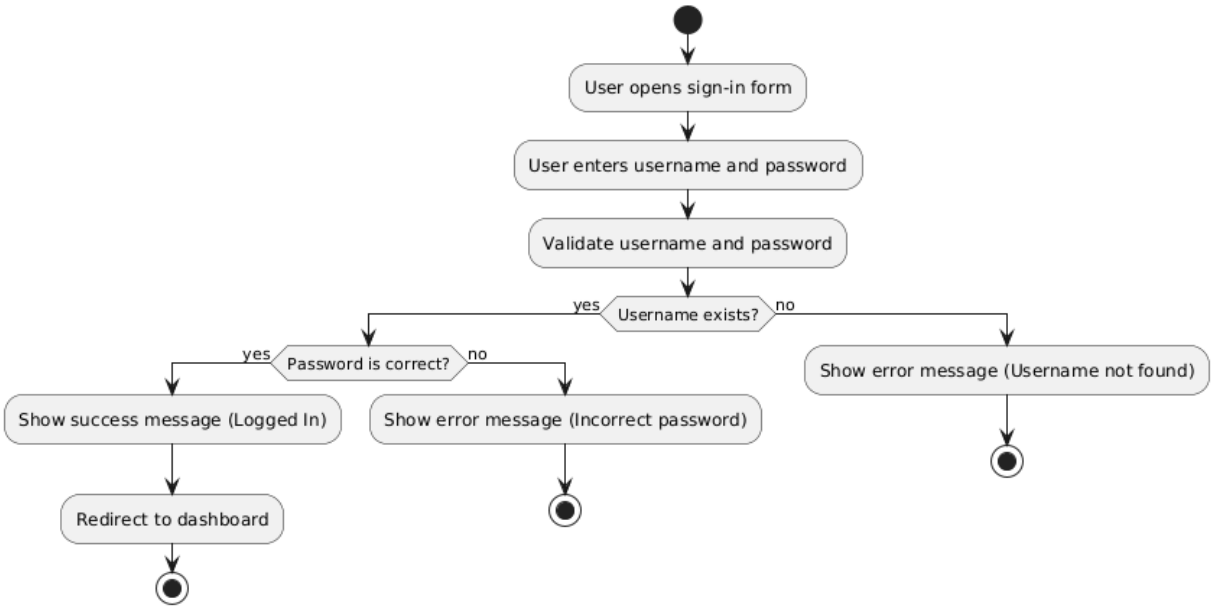
1. Start the program.
2. Display Main Menu with options:
    - Add Expense
    - Update Expense
    - Delete Expense
    - Search Expense
    - View Report
    - Exit
3. User Input: Select an option.
4. Decision Block: Based on user selection, proceed with the corresponding option:
    - Add Expense:
        1. Prompt the user to enter expense details (Category, Amount, Date, Description, Payment Method).
        2. Save the expense data.
        3. Return to the Main Menu.
    - Update Expense:
        1. Prompt the user to search for the expense by ID or category.
        2. Display the selected expense.
        3. Prompt the user to enter new details.
        4. Update the expense data.
        5. Return to the Main Menu.
    - Delete Expense:
        1. Prompt the user to search for the expense by ID.
        2. Confirm the expense deletion.
        3. Delete the expense from the list.
        4. Return to the Main Menu.
    - Search Expense:
        1. Prompt the user to enter search criteria (e.g., Category, Date, Amount).
        2. Display matching expenses.
        3. Return to the Main Menu.
    - View Report:
        1. Generate a report showing total expenses by category.
        2. Display the report.
        3. Return to the Main Menu.
    - Exit:
        1. End the program.
5. End.

Flowchart:

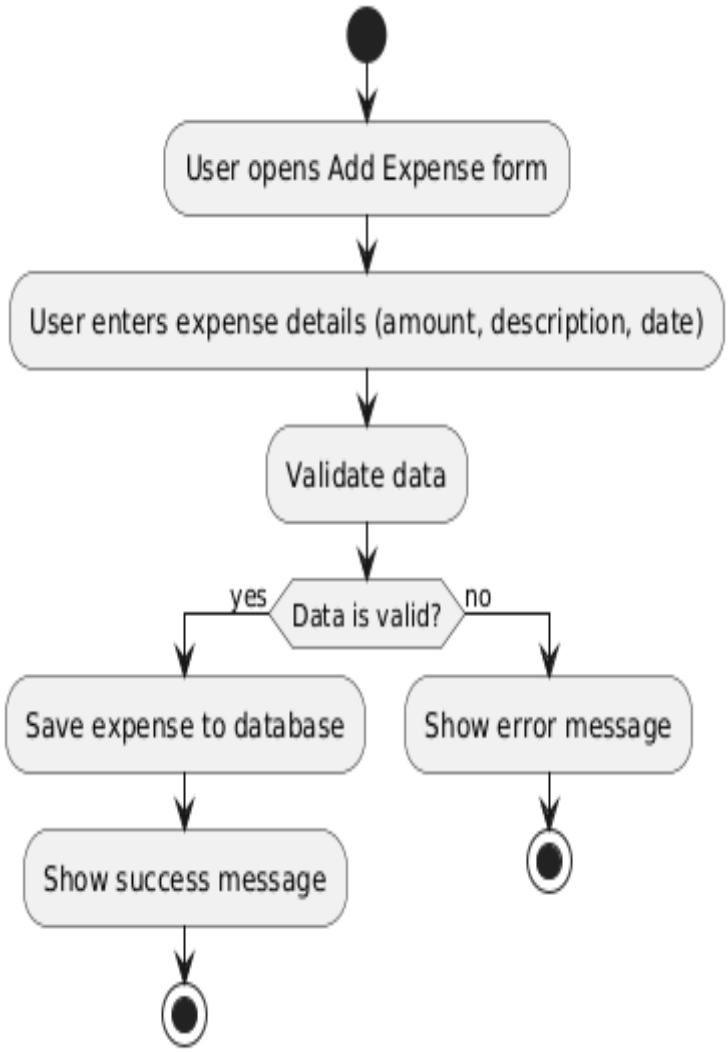
For Sign up:



For sign in:



For adding values:



5.1 Explanation of the Flowchart

Explanation of the Flowchart for Sign-In Process

1. The flowchart for the Sign-In Process is a visual representation that outlines the sequence of steps and decision points a user goes through when attempting to log in to a system. Here's an explanation of each part of the flowchart:
2. Start:
3. The process begins when the user decides to sign in to the system, which prompts them to open the sign-in form on the application interface.
4. User enters username and password:
5. The user is prompted to enter their username and password into the provided fields. These are the credentials that the system will validate to authenticate the user.
6. Validate username and password:
7. This step represents the system's attempt to validate the user's credentials. The system checks whether the provided username exists in its database and whether

the password matches the one associated with that username.

8. Username exists? (Decision Point):
9. Yes: If the system finds that the entered username exists in the database, the process moves forward to check if the password is correct.
10. No: If the username is not found, the system will display an error message stating "Username not found" and end the process.
11. Password is correct? (Decision Point):
12. Yes: If the password entered by the user is correct, the user is successfully authenticated. The system then proceeds to show a success message, such as "Logged In", and redirects the user to the main application or dashboard.
13. No: If the password entered by the user does not match the stored password, the system displays an error message such as "Incorrect password" and the process ends.
14. End:
15. The flow ends after displaying the appropriate error or success message based on the validation results. If the username or password is incorrect, the process stops after showing an error message, prompting the user to retry or recover their credentials.

Explanation of the Flowchart for Sign-Up Process:

The flowchart for the Sign-Up Process represents the steps a user takes to create a new account in a system. Here's an explanation of each part of the flowchart:

1. Start:
    - The process begins when the user chooses to sign up for a new account by accessing the sign-up page on the application interface.
2. User enters required information:
    - The user is prompted to fill in details like username, password, email, and any other required information (e.g., phone number, date of birth). These details are required for account creation.
3. Validate information:
    - This step checks whether the entered data meets the system's requirements. Common validation rules include:
        - Username: Must be unique and not already taken.

        - Password: Must meet security standards (e.g., minimum length, complexity).
        - Email: Must follow a valid email format.
        - Other fields may have additional rules, such as ensuring the phone number is valid.
4. Is username unique? (Decision Point):
    - Yes: If the username entered by the user is unique and not already taken, the process continues to the next step.
    - No: If the username already exists in the system, the user is informed with an error message such as "Username already taken" and asked to choose a different username.
5. Is password valid? (Decision Point):
    - Yes: If the password meets the system's criteria (e.g., length, complexity), the process continues to the next step.
    - No: If the password does not meet the required standards, the user is prompted with an error message like "Password does not meet criteria" and asked to enter a valid password.
6. Is email valid? (Decision Point):
    - Yes: If the email is in the correct format (e.g., "user@example.com"), the process continues.
    - No: If the email is invalid, the system displays an error message such as "Invalid email format" and asks the user to provide a valid email address.
7. Save user information:
    - If all the provided information passes validation, the system stores the new account details in its database (e.g., username, password, email).
8. Display success message:
    - The user is shown a success message such as "Account created successfully" indicating that their account has been created. The system may also provide a link to log in.
9. End:
    - The process ends after the user is informed of the successful account creation. If the process encounters any errors (such as invalid username, password, or email), the user is asked to correct the issue and attempt sign-up again.

Explanation of the Flowchart for adding values:

1. Start: The flowchart begins when the user decides to add a new expense.
2. Display "Add Expense" Form: The system shows the form for entering expense details.
3. User Inputs Details: The user provides the necessary details like the amount, category, description, and date for the expense.
4. Validate Inputs: The system checks if all fields are filled correctly and if the data is valid (e.g., numeric for amount, valid date format).
5. Validation Check:
    - If valid, the system saves the expense information to the database and shows a success message.
    - If invalid, an error message is shown, and the process stops.
6. Return to Main Menu: After a successful addition, the system returns to the main menu or a similar screen.