VIT®

BHOPAL

PROJECT REPORT FILE :

PROJECT TITLE : FINANCE TRACKING

NAME : SUMANTH B

REG NO. : 25BAI11402

SUBJECT : INTRODUCTION TO PROBLEM SOLVING AND PROGRAMMING

FACULTY NAME : SIVABALAN

SLOT : B14+B23+D21

INTRODUCTION

this project is of financial tracking which has been developed using python programming.

This basically helps to track our financial aids like how much money we are spending monthly or yearly and In which field we are spending the most

This project only uses python as its core language no other language is used to execute the program

Objectives

To develop a Python-based finance tracking system that helps users record, categorize, and analyze their income and expenses. The program aims to provide insights into spending habits, support budgeting decisions, and improve overall financial management through automated calculations, data visualization, and report generation.

Functional Requirements

1. **USER ACCOUNT MANAGEMENT**

The system shall allow users to create an account and login

2. Transaction Management

The system shall allow users to add new transactions with details

3. Data Storage

The system shall store all transactions in a persistent format (CSV, JSON, database, etc.).

4. Budget Management

The system shall allow users to set monthly or category-based budgets.

5. Reporting and Summaries

The system shall display total income, expenses, and net balance.

6. Security

The system shall restrict access to each user's financial data (if multi-user).

Non Functional Requirements

1. **PERFORMANCE**

The system shall respond to user actions (adding, viewing, or searching transactions) within 2 seconds.

2. **USABILITY**

The system shall provide a simple and intuitive interface that users can learn without prior technical knowledge.

3. **RELIABILITY**

The system shall maintain at least 99% uptime during normal operation.

4. **SECURITY**

The system shall protect stored financial data using secure storage methods to prevent unauthorized access.

5. **SCALABILITY**

The system shall support an increasing number of transactions without significant performance degradation.

6. **MAINTAINABILITY**

The codebase shall be structured and documented to allow future modifications with minimal effort.

7. **PORTABILITY**

The system shall run on multiple operating systems (Windows, macOS, Linux) without requiring major changes.

System Architecture

1. **USER INTERFACE LAYER**

Purpose: Handles all interactions with the user.

Components:

Command-Line Interface (CLI) or Graphical User Interface (GUI)

Input validation

Display of reports, summaries, and notifications

Responsibilities:

Collect user inputs (transactions, budgets, filters)

Present processed data (charts, summaries, history

2. APPLICATION LOGIC LAYER

Purpose: Executes all core functions of the finance tracking system.

Components:

Transaction Manager

Budget Manager

Organize transactions into categories

3. DATA MANAGEMENT LAYER

Purpose: Manages communication between the application and storage.

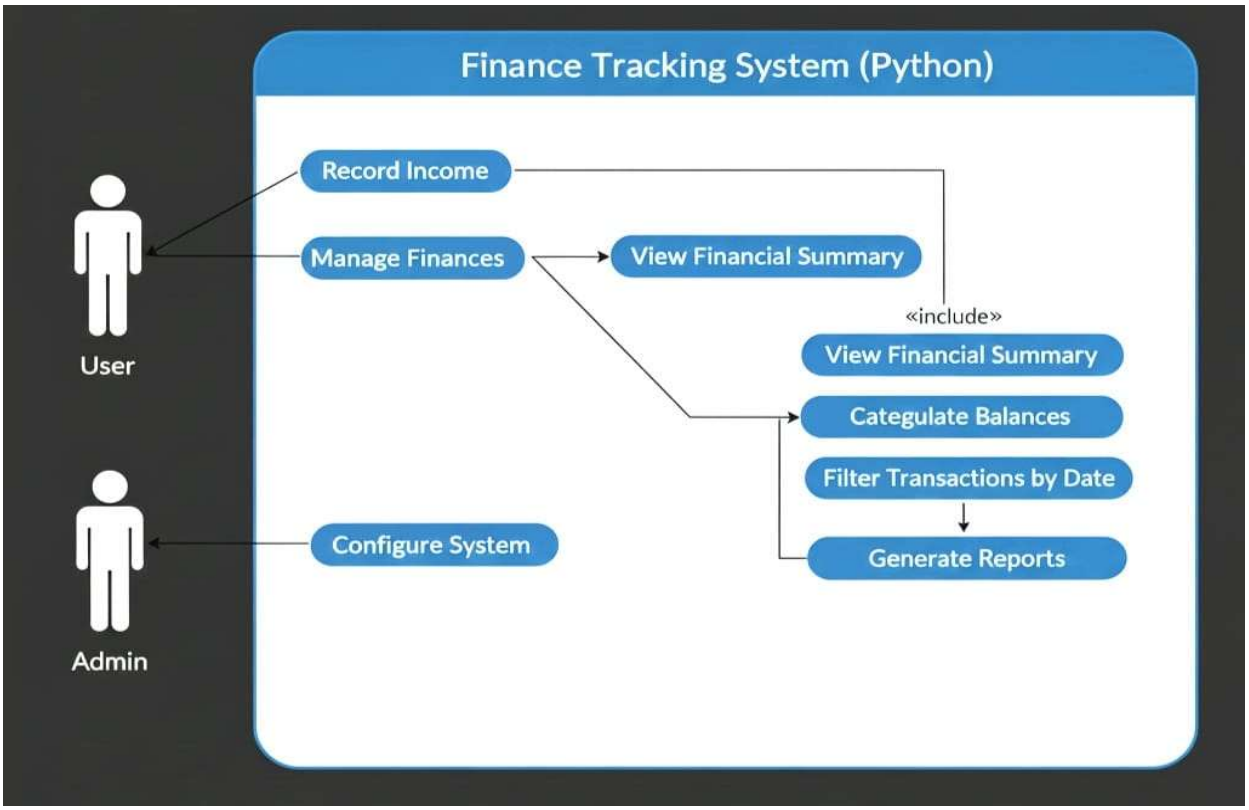
Components:

Data Handler module

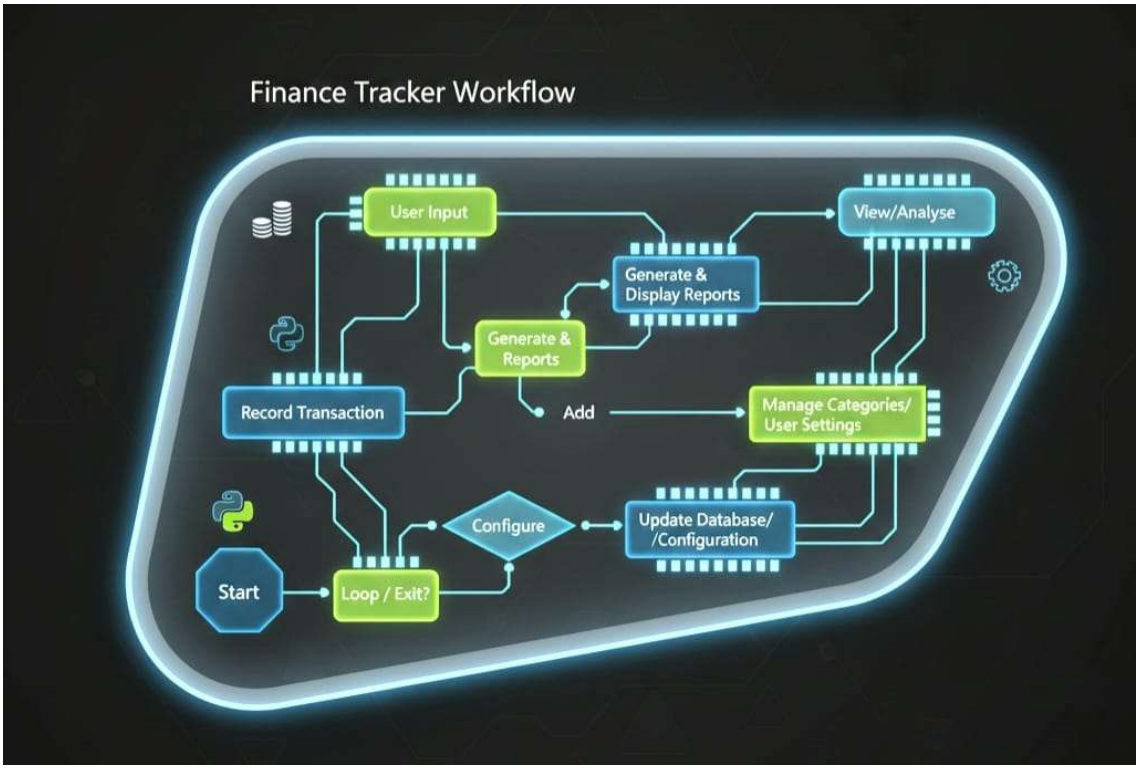
Responsibilities:

Convert data between Python objects and storage format

USE CASE DIAGRAM



WORK FLOW DIAGRAM



Design Decisions and Rationale

1. Programming Language:

Decision: Use Python as the primary language.

Rationale:

Easy to learn and widely used for scripting and automation

Strong libraries for data handling (Pandas, JSON, CSV)

2. Storage Format:

JSON or CSV Decision: Use JSON or CSV files for storing transactions.

Rationale:

Lightweight and easy to read/write

No need for setting up a database for small to medium use

JSON supports structured data (categories, budgets)

CSV is easy to export to Excel and spreadsheets

3. Budget Tracking

reached Decision: Allow users to set budgets per month or category.

Rationale:

Supports financial planning

Makes the tracker practical and useful

Provides meaningful alerts when limits are

4. Security

Decision: Store data locally without sending it to external servers.

Rationale: Protects privacy of financial information

Eliminates cloud security risks

Ideal for offline personal finance tools

Implementation Details

1.Programming Language and Environment

The system is implemented in Python 3.x.

Development environment: VS Code / PyCharm / IDLE (any is acceptable).

No external database is required; data is stored locally.

2. Data Model

Transaction Object Structure

Each transaction is stored as a dictionary:

```
{
  "id": 1,
  "amount": 120.50,
  "type": "expense",
  "category": "Groceries",
  "date": "2025-01-20",
  "description": "Supermarket shopping"
}
```

Budget Model

```
{
  "monthly_budget": 2000,
  "category_budgets": {
    "Food": 400,
    "Transport": 150
  }
}
```

Both are stored inside a JSON file.

3. Core Modules

4.1 Transaction Manager (transactions.py)

Functions include:

add_transaction()

edit_transaction()

delete_transaction()

list_transactions()

search_transactions()

Implementation Approach:

A Python list stores all transactions in memory after loading from JSON.

Each change updates the JSON file immediately.

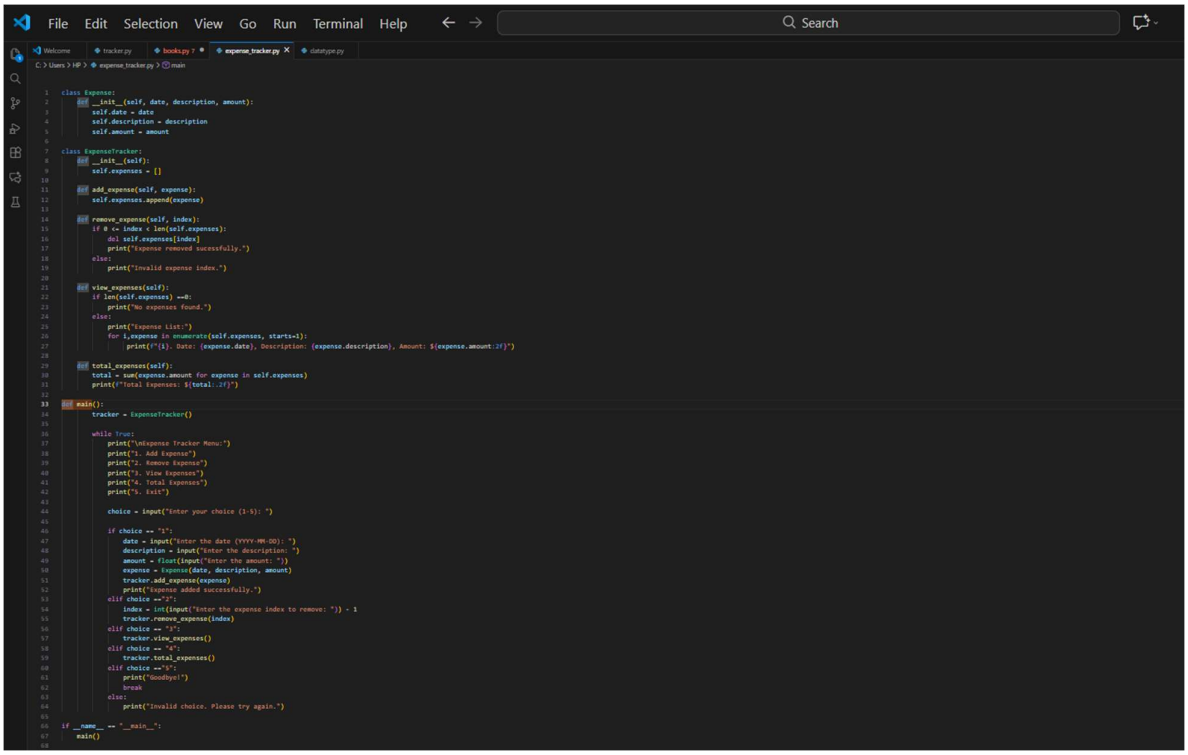
4. User Interface (interface.py)

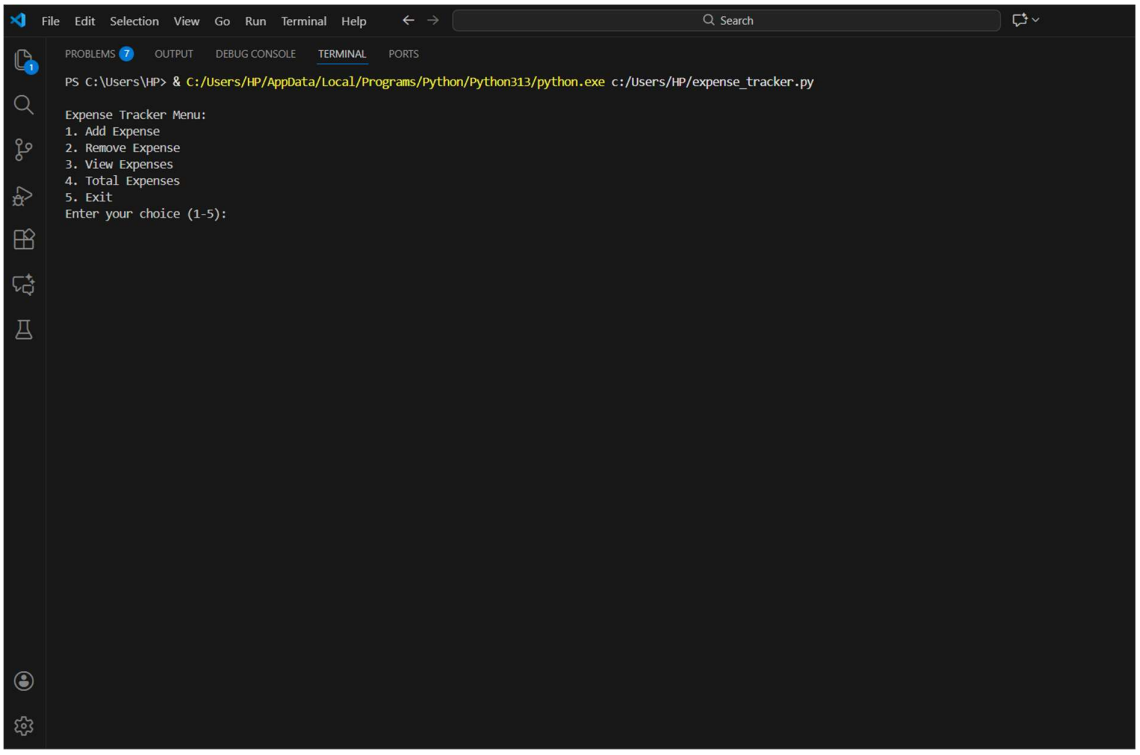
Implementation uses a menu-driven CLI.

Example Menu Display

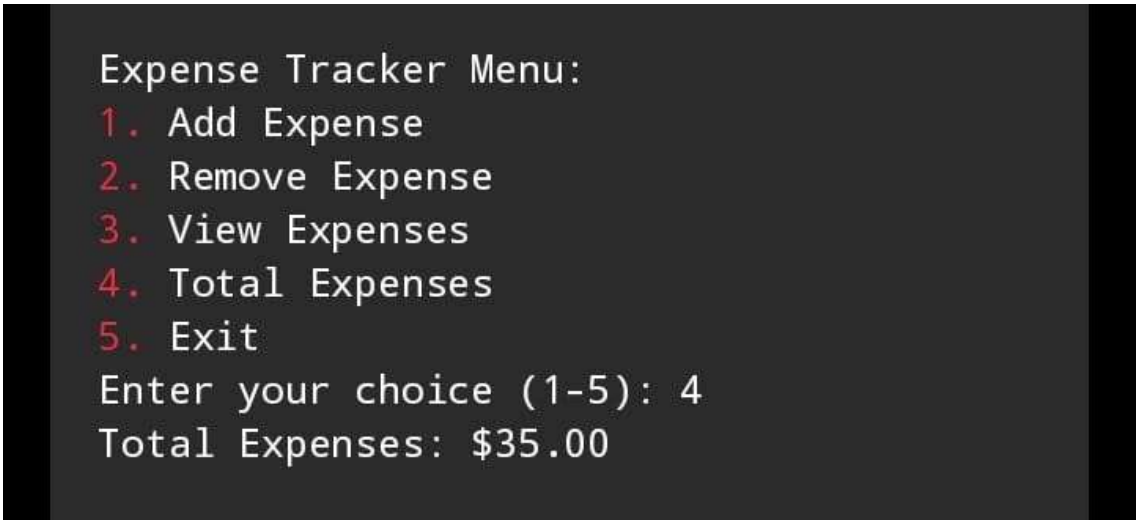
```
print("1. Add Transaction")
print("2. Edit Transaction")
print("3. Delete Transaction")
print("4. View Reports")
print("5. Set Budget")
print("6. Search Transactions")print("0. Exit")
```

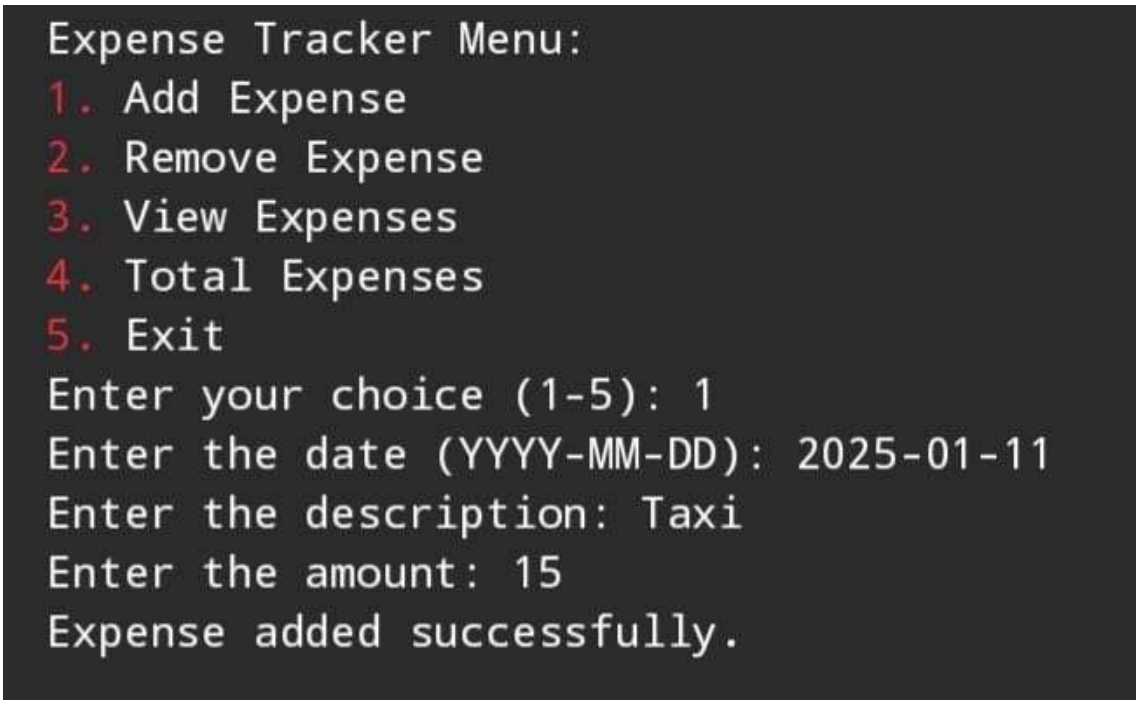
SCREENSHOTS/RESULTS






PROBLEMS 7 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\HP> & C:/Users/HP/AppData/Local/Programs/Python/Python313/python.exe c:/Users/HP/expense_tracker.py
Expense Tracker Menu:
1. Add Expense
2. Remove Expense
3. View Expenses
4. Total Expenses
5. Exit
Enter your choice (1-5): 1
Enter the date (YYYY-MM-DD): 2025-10-25
Enter the description: FOOD
Enter the amount: 1000
Expense added successfully.


Expense Tracker Menu:
1. Add Expense
2. Remove Expense
3. View Expenses
4. Total Expenses
5. Exit
Enter your choice (1-5): 4
Total Expenses: $35.00


Expense Tracker Menu:
1. Add Expense
2. Remove Expense
3. View Expenses
4. Total Expenses
5. Exit
Enter your choice (1-5): 1
Enter the date (YYYY-MM-DD): 2025-01-11
Enter the description: Taxi
Enter the amount: 15
Expense added successfully.

TESTING APPROACH

The testing approach used in this Expense Tracker program is mainly manual and functional testing, since the application is interactive and relies on user inputs through a command-line menu. Manual testing is used to verify that each feature—such as adding, removing, viewing, and totaling expenses—works correctly by entering different inputs and observing the printed output. Functional and black-box testing ensure that valid inputs behave as expected and invalid inputs (such as wrong menu choices or invalid indexes) produce the correct error messages. Boundary testing is applied to check edge cases like removing the first, last, or non-existent expense, as well as handling zero or large amounts. Overall, the code is best validated by directly running it, exploring all menu options, and confirming correct behavior without program crashes.

Challenges faced

Some challenges faced during the development of this Expense Tracker program include handling user input errors, such as non-numeric values for amounts or invalid menu options, which required implementing clear validation and error messages to prevent crashes. Managing the removal of expenses by index also posed challenges, especially ensuring that out-of-range or negative indexes were handled safely. Designing an intuitive console-based user interface while keeping the code simple and readable required careful structuring. Additionally, since the program is interactive, testing and debugging were more time-consuming, as each feature had to be manually executed and verified through simulated user interactions.

LEARNING AND KEY TAKEAWAYS

Through developing this Expense Tracker program, key learning and takeaways include gaining a deeper understanding of Python classes, object-oriented programming, and how to structure a program using multiple methods to handle different tasks. Working with user input reinforced the importance of input validation, error handling, and designing user-friendly menu-driven interfaces.

FUTURE ENHANCMENTS

Future enhancements for this Expense Tracker could include adding data persistence by saving expenses to a JSON or CSV file so that user data is not lost when the program closes, as well as implementing a graphical user interface (GUI) to improve usability and make the application more visually appealing. Additional features like category-based tracking, monthly budget limits, charts for visualizing spending patterns, and search or filter options would make the tool more powerful and user-friendly. Integrating automated backups, exporting reports, and potentially migrating to a database for larger datasets could further enhance functionality, scalability, and reliability.

REFERENCES

Official python documentation (docs.python.org)

Class notes and basic programming guidelines

THANK YOU