Personal Expense Tracker CLI

A PROJECT REPORT

Submitted by

Devendra Singh Thakur

(24MIM10245)

in partial fulfillment for the award of the degree

of

INTEGRATED MASTER OF TECHNOLOGY

In

ARTIFICIAL INTELLIGENCE

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING AND

ARTIFICIAL INTELLIGENCE

VIT BHOPAL UNIVERSITY

KOTHRI KALAN, SEHORE

MADHYA PRADESH - 466114

SEPTEMBER 2025

2. Introduction

This report details the design, implementation, and evaluation of the Personal Expense Tracker CLI, a Java-based command-line application developed as part of the flipped course evaluation. The project aims to provide a reliable and accessible tool for individuals to manage and analyze their personal expenditures. By employing modular Java programming and file-based data structures, the system offers three key functions: data management, expense analysis, and budget tracking.

3. Problem Statement

Many individuals lack clear, aggregated visibility into their daily, weekly, and monthly spending, making it difficult to adhere to financial goals. Manual expense tracking is often inefficient and prone to human error. The need exists for a straightforward, automated tool that can efficiently process raw transaction data and provide immediate, actionable analytical insights.

4. Functional Requirements (FRs)

The project includes three major functional modules:

ID	Requirement	Description
FR1.1	Add New Expense	Allow the user to input Date, Category, and Amount, appending the new record to the expenses.csv file.
FR2.1	Calculate Total Spend	Sum all amounts in the CSV file and output the total spend.
FR2.2	Calculate Category Percentages	Calculate and display the percentage breakdown of total spend for each category.
FR3.1	Set Category Budget	Allow the user to input a maximum spending limit for a specific category and store it in budgets.txt.
FR3.2	Budget Status Report	Compare the actual spending in each category against the set budget and report the variance (Under/OVER).

5. Non-functional Requirements (NFRs)

The system enforces at least four non-functional criteria:

ID	Requirement	Category	Rationale
NFR1	Clear User Prompts	Usability	The application provides clear, text-based menus and prompts for all user interactions.
NFR2	Fast Analysis Time	Performance	The analysis module must complete all calculations for typical datasets (up to 5,000 records) within 1 second.
NFR3	File Error Handling	Reliability	The system includes robust try-catch blocks to handle critical errors such as FileNotFoundException and NumberFormatException without crashing.
NFR4	Modular Logic	Maintainability	The core data loading logic is isolated in a reusable loadExpenseData() method, simplifying debugging and future enhancements.

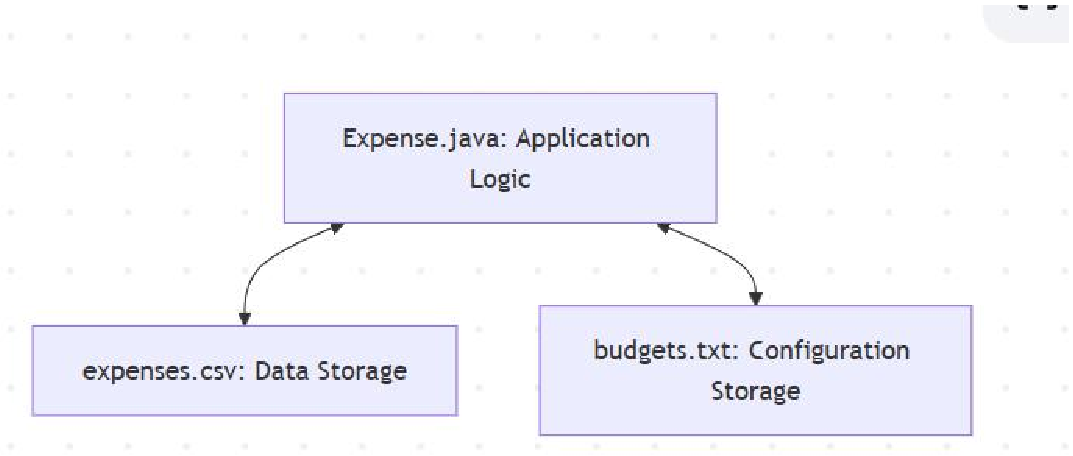
6. System Architecture

The Expense Tracker employs a simple Client-Server/File-Based Architecture. The Java application acts as the client processing data from local flat files.

Application Logic (Expense.java): Contains the user interface (CLI menu) and all functional logic.

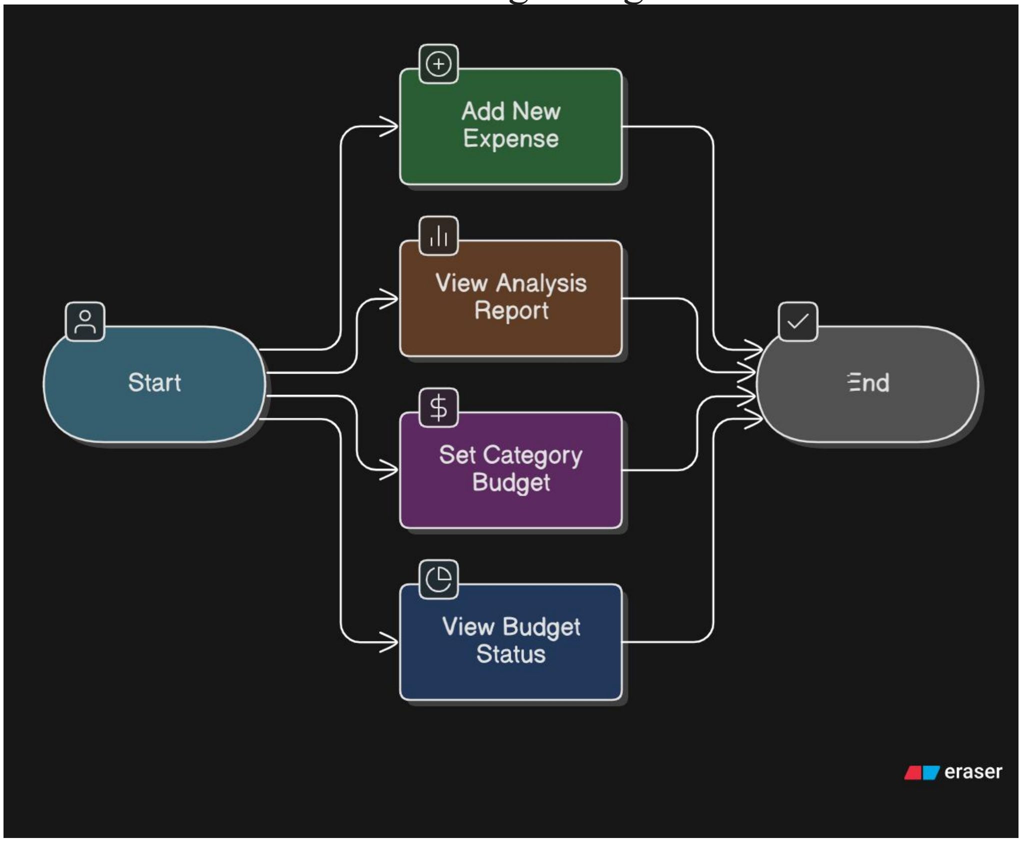
Data Storage (expenses.csv): The primary persistence layer for all transaction records.

Configuration Storage (budgets.txt): Stores budget settings, decoupled from the core transaction data.


Expense.java: Application Logic
expenses.csv: Data Storage
budgets.txt: Configuration Storage

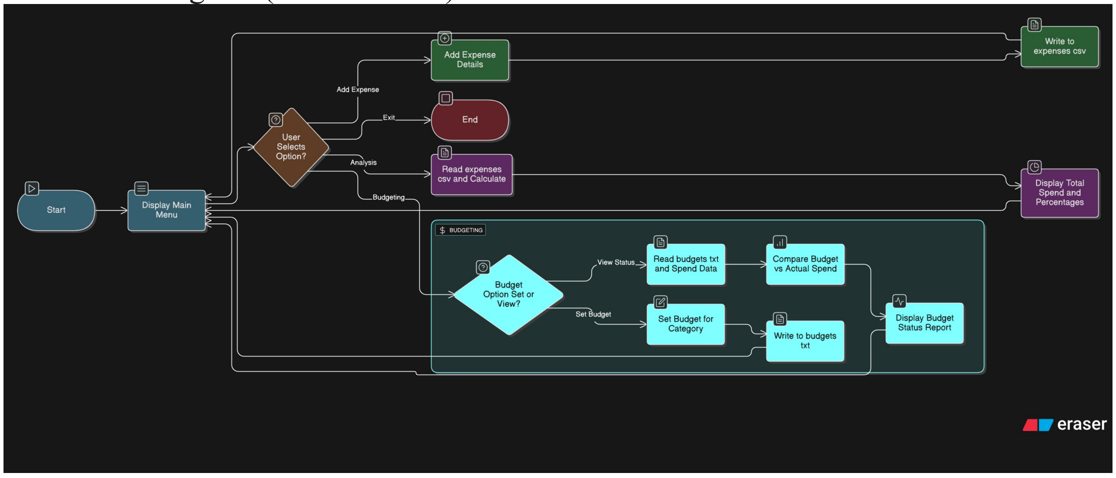
The application is structured around a central Java execution layer that communicates directly with two distinct flat files for persistence: one for transaction data and one for configuration (budgets).

7. Design Diagrams



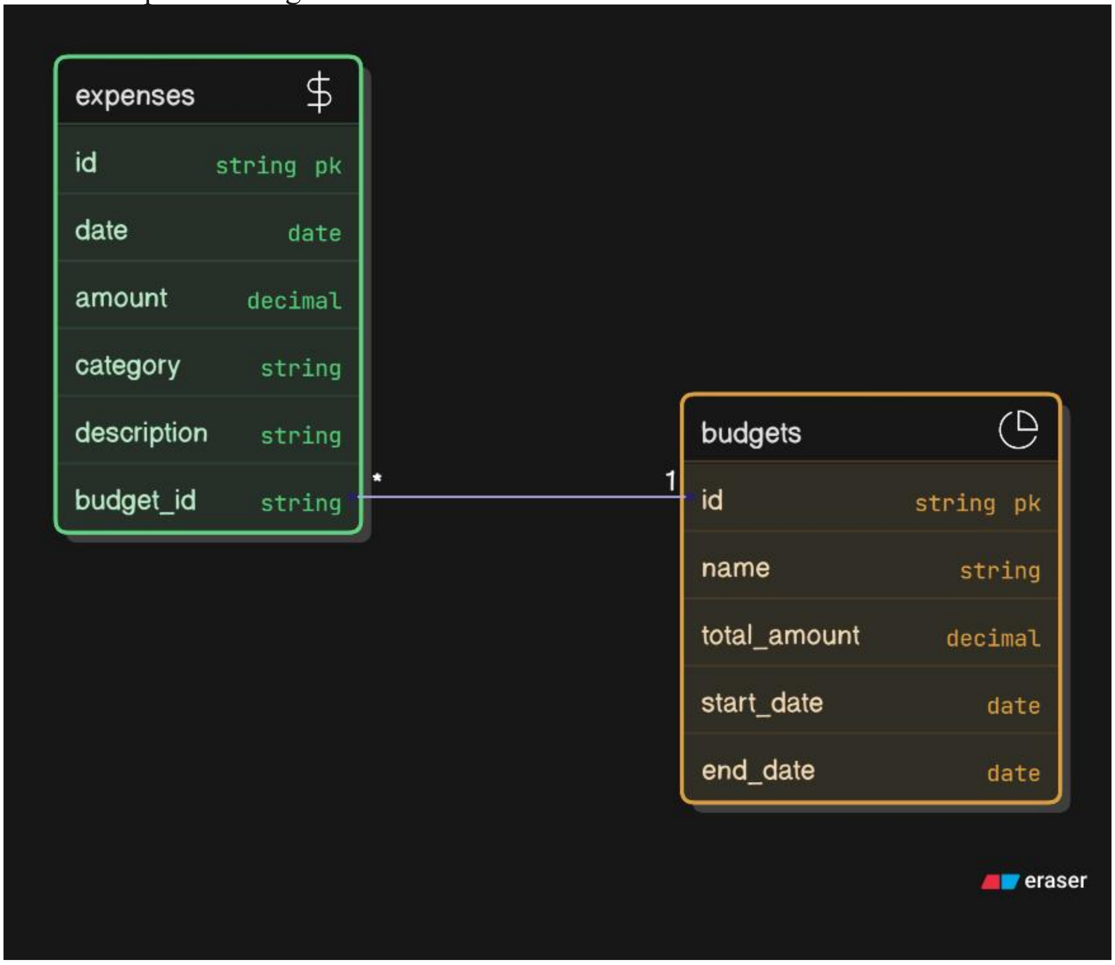
The diagram below illustrates the interactions between the primary User and the key functional modules of the Expense Tracker.

Workflow Diagram (Process Flow)



This diagram maps the step-by-step user experience through the application's menu.

Class/Component Diagram



The system relies on a single main class but utilizes standard Java structures for processing.

8. Design Decisions & Rationale

Decision	Rationale
Data Structure (HashMap)	Used to store and aggregate category-based spending amounts. This allows for efficient O(1) average-time lookups and updates, crucial for fast expense analysis (FR2.2).
File I/O Method (BufferedReader)	Chosen for reading data line-by-line. This approach is memory-efficient and contributes to the Performance (NFR2) requirement, especially if the CSV file were to grow large.
Modularity (loadExpenseData())	Separating the file reading into its own method ensures the core data acquisition logic is reusable by both the Analysis and Budgeting modules, supporting Maintainability (NFR4).
CLI Interface	A simple command-line interface was chosen to focus the development effort solely on the core data processing algorithms, aligning with the constrained project scope.

9. Implementation Details

The entire application is written in Java. Key implementation aspects include:

Version Control: Git was used for version control, committing changes modularly and ensuring a complete development history.

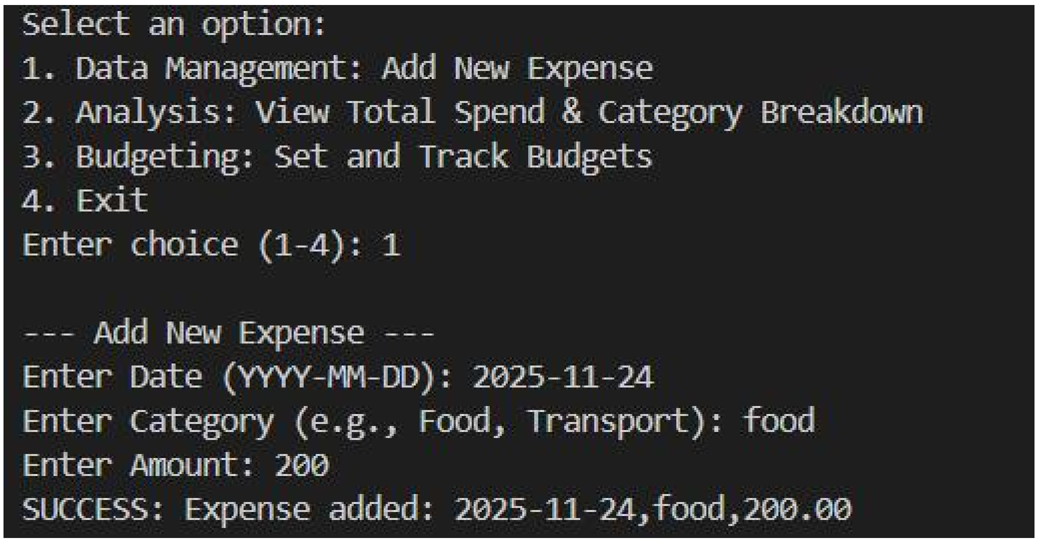
Data Parsing: The String.split(",") method is used to parse CSV lines into category and amount components.

Persistence: File writers are used in "append mode" (new FileWriter(..., true)) for adding new expenses, ensuring data integrity.

Calculations: The analysis module uses stream API for efficient summation and basic arithmetic for percentage calculations

10. Screenshots / Results

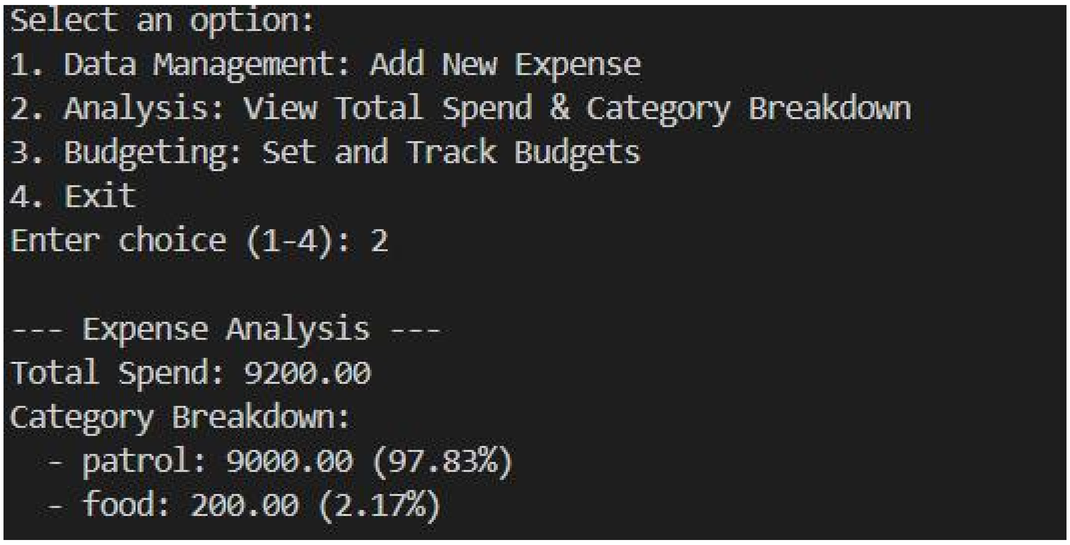
Screenshot 1: Main Menu and Data Entry



```
Select an option:
1. Data Management: Add New Expense
2. Analysis: View Total Spend & Category Breakdown
3. Budgeting: Set and Track Budgets
4. Exit
Enter choice (1-4): 1

--- Add New Expense ---
Enter Date (YYYY-MM-DD): 2025-11-24
Enter Category (e.g., Food, Transport): food
Enter Amount: 200
SUCCESS: Expense added: 2025-11-24,food,200.00
```

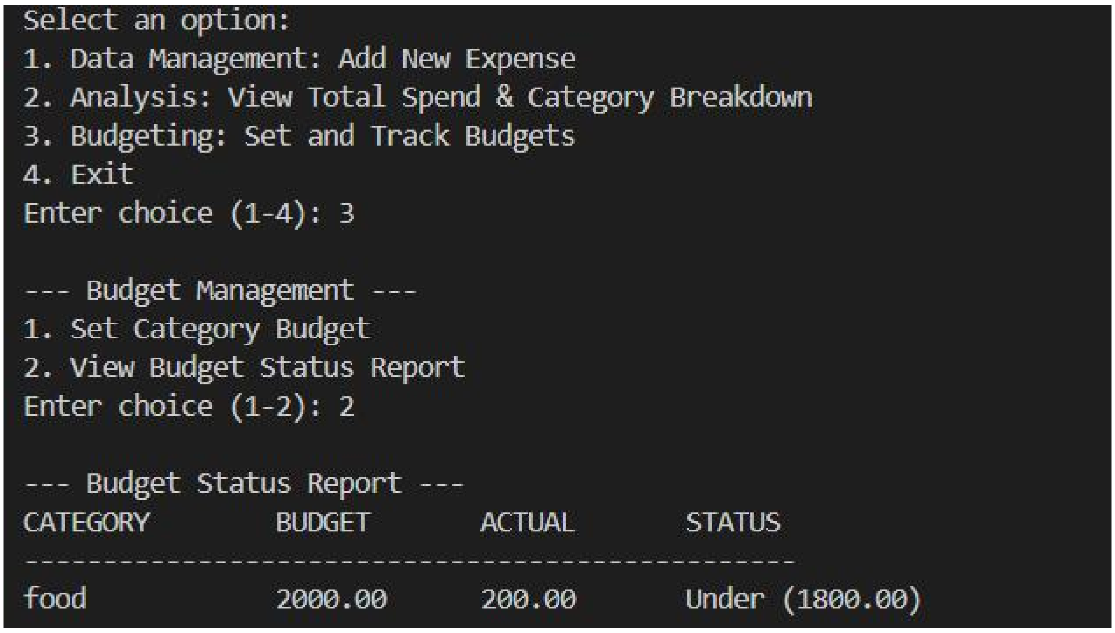
Screenshot 2: Expense Analysis Results



```
Select an option:
1. Data Management: Add New Expense
2. Analysis: View Total Spend & Category Breakdown
3. Budgeting: Set and Track Budgets
4. Exit
Enter choice (1-4): 2

--- Expense Analysis ---
Total Spend: 9200.00
Category Breakdown:
  - patrol: 9000.00 (97.83%)
  - food: 200.00 (2.17%)
```

Screenshot 3: Budget Status Report



```
Select an option:
1. Data Management: Add New Expense
2. Analysis: View Total Spend & Category Breakdown
3. Budgeting: Set and Track Budgets
4. Exit
Enter choice (1-4): 3

--- Budget Management ---
1. Set Category Budget
2. View Budget Status Report
Enter choice (1-2): 2

--- Budget Status Report ---
CATEGORY        BUDGET      ACTUAL      STATUS
--------------------------------------------------
food            2000.00     200.00      Under (1800.00)
```

11. Testing Approach

Testing was conducted using the following methodology:

Unit Testing (Implied): Each functional method (addNewExpense, performAnalysis, viewBudgetReport) was tested independently to confirm correct output before integration.

Edge Case Testing (Reliability): Manual testing was performed by inputting non-numeric values for the amount field and by entering an incorrect file path to confirm the try-catch blocks correctly enforced NFR3 (Reliability).

Integration Testing: A full run-through was executed, adding multiple categories and then immediately running the analysis and budget reports to ensure data flow between the modules was accurate.

12. Challenges Faced

The primary challenge was managing data persistence for the separate Budgeting Module (FR3.1). Storing budgets within the CSV file was ruled out as it would complicate the core expense analysis logic. The solution was to create a second, simpler file (budgets.txt) and implement custom saving/loading logic to ensure separation of concerns and maintain clarity in data structures.

13. Learnings & Key Takeaways

The project reinforced the importance of modular design (NFR4). By isolating data loading into loadExpenseData(), the code was significantly cleaner, reducing the risk of errors across different modules. Furthermore, the practical use of Java Collections (HashMap) demonstrated their efficiency in aggregating large datasets, a critical element in data analysis.

14. Future Enhancements

Potential future developments for the Expense Tracker include:

Database Integration: Replace CSV/TXT files with a proper embedded database (e.g., SQLite) for faster queries and relational data integrity.

Advanced Analysis: Add functionality to track spending trends over time (monthly comparisons) and identify anomalies.

Configuration: Allow users to define custom date formats and CSV delimiters.

15. References

- **Project Requirements & Constraints:**
    - VITyarthi Course Document: *Build Your Own Project - General Project Instructions & Submission Guidelines*. (This source provided all functional and structural requirements for the submission).

- **Programming Language & Core Tools:**
    - Oracle Java Documentation: Official API Specifications for Java Development Kit (JDK). (Specifically, documentation for java.io.BufferedReader, java.io.FileWriter, java.util.HashMap, and java.util.Scanner was consulted for implementation).
    - Git Documentation: Used for version control and repository management.