PROJECT REPORT

EXPENDITURE MANAGER APPLICATION

Submitted by:

Name: Debasish Kumar Sahoo

Registration No: 25BAI10173

Department: Computer Science / AI & Data Science

1. INTRODUCTION

1.1 Abstract

The **Expenditure Manager** is a Python-based console application designed to help users track their daily financial expenses. In an era where financial management is crucial, this tool provides a simple, lightweight, and efficient way to record, view, and analyze spending habits. The system utilizes file handling to ensure data persistence, allowing users to close the program and retrieve their data later.

1.2 Objective

The primary objectives of this project are:

1. To develop a user-friendly interface for managing personal finances.
2. To implement **CRUD** (Create, Read, Update, Delete) operations using Python.
3. To utilize **File Handling** (.txt files) for permanent data storage without needing a complex database.
4. To generate a basic financial summary to help the user understand their spending distribution.

2. SYSTEM ANALYSIS

2.1 Technical Requirements

- **Programming Language:** Python 3.x
- **Modules Used:**
  - os: Used for checking file existence.
  - datetime: Used for automatically fetching the current date.
- **Storage:** Text file (expenses.txt).
- **Interface:** Command Line Interface (CLI).

2.2 Data Structure

The data is stored in expenses.txt in a pipe-separated value format to ensure easy parsing:
Date|Category|Amount|Description
Example Data Entry:
2023-10-25|Food|150.0|Lunch with friends

2.3 Functional Modules

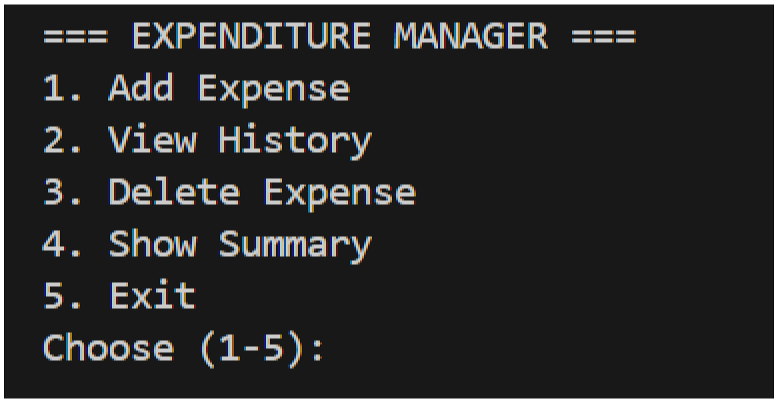
1. **Add Expense:** Accepts amount, category, and description from the user. Validates that the amount is positive and saves the entry with the current date.
2. **View History:** Reads the text file and formats the data into a readable table with aligned columns.
3. **Delete Expense:** Loads all data into a list, removes the item selected by the user via ID, and overwrites the file with the updated list.
4. **Show Summary:** Aggregates the expenses by category and calculates the Grand Total using a dictionary for accumulation.

3. SOURCE CODE

```
import os
from datetime import date

FILE_NAME = "expenses.txt"

def save_expenses_to_file(expenses):
    """Helper function: Overwrites the file with the current list of expenses."""
    try:
        with open(FILE_NAME, "w") as file:
            for item in expenses:
                # Reconstruct the line: Date|Category|Amount|Description
                line = f"{item['date']}|{item['category']}|{item['amount']}|{item['description']}\n"
                file.write(line)
    except IOError as e:
        print(f"Error saving data: {e}")

def get_all_expenses():
    """Reads the file and returns a list of dictionaries."""
    expenses = []
    if not os.path.exists(FILE_NAME):
        return expenses

    with open(FILE_NAME, "r") as file:
        for line in file:
            parts = line.strip().split("|")
            if len(parts) == 4:
                expenses.append({
                    "date": parts[0],
                    "category": parts[1],
                    "amount": float(parts[2]),
                    "description": parts[3]
                })
    return expenses

def add_expense():
    """Appends a new expense to the file."""
    print("\n--- Add New Expense ---")
    try:
        amount = float(input("Enter Amount: "))
        if amount <= 0:
            print("Amount must be positive!")
            return

        category = input("Enter Category: ").strip().capitalize()
        description = input("Description: ").strip().replace("|", "-")
        today = date.today().strftime("%Y-%m-%d")

        # Append directly to file
        entry = f"{today}|{category}|{amount}|{description}\n"
        with open(FILE_NAME, "a") as file:
            file.write(entry)

        print("✅ Expense saved!")

    except ValueError:
        print("❌ Error: Please enter a valid number.")

def view_expenses():
    """Displays expenses in a table."""
    expenses = get_all_expenses()
    print("\n--- Your Expenditure History ---")
    if not expenses:
        print("No records found.")
        return

    print(f"{'ID':<4} | {'Date':<12} | {'Category':<15} | {'Amount':<10} | {'Description'}")
    print("-" * 70)

    for i, item in enumerate(expenses):
        print(f"{i+1:<4} | {item['date']:<12} | {item['category']:<15} | ${item['amount']:<9.2f} |
{item['description']}")
    print("-" * 70)

def delete_expense():
    """Lists expenses and asks user which ID to delete."""
    print("\n--- Delete an Expense ---")
    expenses = get_all_expenses()

    if not expenses:
        print("No expenses to delete.")
        return

    # Show the list first so user knows the ID
    view_expenses()

    try:
        choice = int(input("\nEnter the ID of the expense to delete: "))

        if 1 <= choice <= len(expenses):
            removed = expenses.pop(choice - 1) # Remove item from list
            save_expenses_to_file(expenses)    # Rewrite the file
            print(f"✅ Deleted: {removed['description']} (${removed['amount']})")
        else:
            print("❌ Invalid ID. Please check the list and try again.")

    except ValueError:
        print("❌ Please enter a valid integer ID.")

def show_summary():
    """Calculates and displays totals."""
    expenses = get_all_expenses()
    if not expenses:
        print("\nNo data to summarize.")
        return

    total_spent = sum(item['amount'] for item in expenses)

    category_totals = {}
    for item in expenses:
        cat = item['category']
        category_totals[cat] = category_totals.get(cat, 0) + item['amount']

    print("\n--- Financial Summary ---")
    print(f"💰 Grand Total: ${total_spent:.2f}")
    print("\n--- Breakdown by Category ---")
    for cat, amount in category_totals.items():
        print(f"{cat:<15}: ${amount:.2f}")

def main():
    while True:
        print("\n=== EXPENDITURE MANAGER ===")
        print("1. Add Expense")
        print("2. View History")
        print("3. Delete Expense")
        print("4. Show Summary")
        print("5. Exit")

        choice = input("Choose (1-5): ")

        if choice == '1':
            add_expense()
        elif choice == '2':
            view_expenses()
        elif choice == '3':
            delete_expense()
        elif choice == '4':
            show_summary()
        elif choice == '5':
            print("Goodbye!")
            break
        else:
            print("Invalid option.")

if __name__ == "__main__":
    main()
```

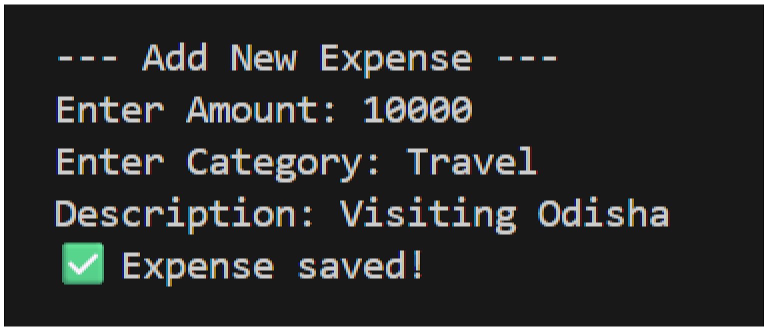
4. OUTPUT SCREENSHOTS (Simulated)

4.1 Main Menu



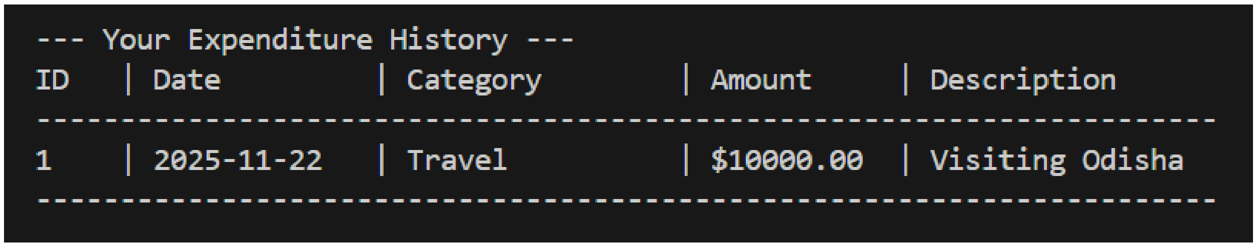
```
=== EXPENDITURE MANAGER ===
1. Add Expense
2. View History
3. Delete Expense
4. Show Summary
5. Exit
Choose (1-5):
```

4.2 Adding an Expense



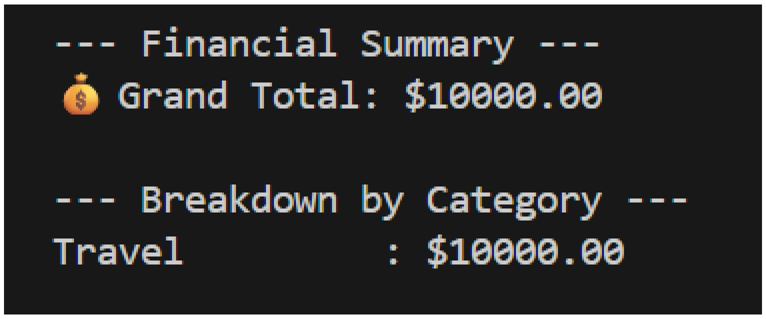
```
 --- Add New Expense ---
 Enter Amount: 10000
 Enter Category: Travel
 Description: Visiting Odisha
 ✅ Expense saved!
```

4.3 Viewing History



```
--- Your Expenditure History ---
ID   | Date         | Category        | Amount     | Description
----------------------------------------------------------------------
1    | 2025-11-22   | Travel          | $10000.00  | Visiting Odisha
----------------------------------------------------------------------
```

4.4 Financial Summary



```
--- Financial Summary ---
💰 Grand Total: $10000.00

--- Breakdown by Category ---
Travel          : $10000.00
```

5. CONCLUSION

The "Expenditure Manager" project successfully demonstrates the application of Python programming concepts to solve real-world problems. By implementing file handling, the system ensures that user data is preserved across sessions. The application is robust, handling basic errors such as invalid inputs or missing files gracefully.

5.1 Future Scope

Future enhancements to this project could include:

1. **Graphical User Interface (GUI):** Using Tkinter or PyQt for a more modern look.
2. **Database Integration:** Replacing text files with SQLite or MySQL for better scalability.
3. **Visual Analytics:** Using Matplotlib to generate pie charts of spending by category.
4. **Budget Limits:** Adding a feature to set monthly limits and alert the user when exceeded.

End of Report