**Expense Tracker Project Documentation**

**Introduction**

The Expense Tracker application is a Python-based tool designed to help users manage their expenses effectively. It enables users to record expenses, categorize them, and track spending over different time periods (daily, weekly, monthly). The project is built with Object-Oriented Programming (OOP) principles and is structured to be modular, enabling easy maintenance and potential future expansion. It uses CSV files for data persistence, making it lightweight and easy to use.

**Objectives**:

* Provide a simple interface for adding, viewing, and deleting expenses.
* Allow users to categorize expenses for better financial insights.
* Generate reports on expenses for specific timeframes (daily, weekly, monthly).
* Ensure data is saved between sessions by using CSV files.

**Architecture**

**Project Structure**

* **main.py**: The main entry point of the application, containing the user interface and menu.
* **expense.py**: Defines the Expense class, representing individual expenses with attributes like amount, date, category, and description.
* **category.py**: Contains the Category class, which manages categories assigned to expenses.
* **expense\_tracker.py**: Manages all core functionalities through the ExpenseTracker class, including adding, deleting, viewing expenses, generating reports, and saving/loading data from a CSV file.
* **data/:** A directory to store the expenses.csv file, which persists expense data across sessions.
* **src/:** Directory containing all source files.

**Class Interactions**

1. **Expense**: Represents an individual expense, holding data such as amount, date, category, and description.
2. **Category**: Manages individual categories. Users can add new categories, which are then used to organize expenses.
3. **ExpenseTracker**: Acts as the central manager, interfacing with Expense and Category objects. It provides methods to:

* Add and delete expenses.
* Filter expenses by category.
* Generate spending reports for different timeframes.
* Save/load expense data to and from a CSV file.

**Features**

1. **Expense Management:**

* Add new expenses with details like amount, date, category, and description.
* Delete expenses by specifying details or an index.
* View all recorded expenses.

1. **Category Management:**

* Define custom categories (e.g., groceries, utilities, entertainment).
* View all available categories.
* Filter expenses by category for insights into specific spending areas.

1. **Report Generation:**

* Generate reports on total expenses for daily, weekly, or monthly timeframes.
* Display expenses for a specific category.

1. **Data Persistence:**

* Save expenses to a CSV file to maintain data between sessions.
* Load expenses from the CSV file when starting the application.

1. **Error Handling:**

* User-friendly error messages for invalid input (e.g., incorrect date format, negative amounts).
* Robust file handling to avoid crashes due to missing or inaccessible files.

**Testing**

1. **Manual Testing:**

* Each feature was tested manually to ensure proper functionality.
* Validations were tested by entering valid and invalid inputs (e.g., negative amounts, incorrect dates).
* Report generation was verified by adding known expenses and checking totals for daily, weekly, and monthly reports.

1. **Edge Cases:**

* Empty file handling (e.g., no CSV file initially).
* Handling of duplicate categories or nonexistent categories.
* Testing with different date formats and unexpected characters for robustness.

1. **Results:**

* The application successfully handled valid inputs, displaying accurate reports and filtering results.
* Error messages appeared as expected for invalid inputs.
* Data persisted correctly across sessions, with CSV files loading and saving accurately.

**Challenges**

1. **Category Filtering:**

* Initially, category filtering caused errors if a category was nonexistent. This was solved by adding a validation check to verify category existence before filtering.

1. **Date Formatting:**

* Handling dates for daily, weekly, and monthly reporting was challenging, especially calculating the start of each period. The datetime module’s timedelta and replace functions resolved this by allowing calculation of specific periods.

1. **Error Handling for File Operations:**

* File handling required careful error management to avoid crashes. Adding checks for file existence and handling exceptions for read/write errors allowed the application to continue running smoothly even with unexpected file issues.

1. **CSV Parsing:**

* Parsing CSV data while maintaining correct data types for amounts and dates was challenging. By using float() for amounts and datetime.strptime() for dates, we ensured data consistency when loading from files.

**Future Enhancements**

1. **Enhanced Reporting:**

* Add more reporting options, such as annual reports or custom date range reports.
* Include visualization options (e.g., charts) to display spending trends.

1. **User Authentication:**

* Add a basic user authentication system so multiple users can track expenses independently.

1. **Database Integration:**

* Move from CSV to a database for larger-scale data handling and improved performance.
* Use SQL to perform more complex queries, improving report generation speed and flexibility.

1. **Mobile/Desktop Application:**

* Expand the project into a mobile or desktop application with a graphical interface, making it more accessible to non-technical users.