**Project Report: Personal Finance Tracker (PyTrack)**

**1. Introduction**

This project details the design and implementation of **PyTrack**, a command-line interface (CLI) application developed using Python for tracking personal income, expenses, and savings. The goal is to provide users with a simple, effective tool for monitoring their financial health, categorizing transactions, and generating basic reports. By utilizing Python's robust features for data processing and a lightweight embedded database, PyTrack offers a locally controlled and private solution for managing personal finances.

**2. Project Goals and Objectives**

The primary objectives of the PyTrack project are to:

1. **Track Transactions:** Allow users to efficiently record and store income and expense transactions with date, amount, category, and description.
2. **Categorization:** Implement a flexible system for defining and assigning categories (e.g., 'Groceries', 'Rent', 'Salary', 'Investment').
3. **Data Persistence:** Ensure all financial data is securely stored locally and persists between application sessions.
4. **Reporting:** Provide basic summary reports, such as monthly income/expense breakdowns and spending trends by category.
5. **User Interface:** Develop an intuitive and easy-to-use Command Line Interface (CLI).

**3. Technology Stack and Tools**

| Component | Technology | Rationale |
|---|---|---|
| **Primary Language** | Python 3.x | Excellent for data manipulation, clear syntax, and extensive library support. |
| **Database** | SQLite3 | Python's standard embedded database library. Requires no separate server setup, ensuring lightweight and local data storage. |
| **CLI Framework** | Standard Python input() / print() or Typer | For creating a simple, navigable command-line interface. |

| | | |
|---|---|---|
| **Data Handling** | datetime and csv modules | For date management and optional data export/import. |

**4. System Architecture**

The application follows a simple **Three-Layer Architecture** common in standalone applications:

1. **Presentation Layer (CLI):** Handles user input and displays results. This layer interacts directly with the Logic Layer.
2. **Logic Layer (Finance Manager):** Contains the core business logic (e.g., calculating totals, filtering transactions, generating reports). It interacts with the Data Access Layer.
3. **Data Access Layer (Database Adapter):** Manages all interactions with the SQLite database, abstracting SQL queries from the Logic Layer.

**Key Python Classes/Modules:**

- **Transaction.py**: Defines the Transaction class, handling data validation and structuring transaction records.
- **DatabaseManager.py**: A module responsible for connecting to the SQLite database, creating the necessary tables (transactions and categories), and executing CRUD (Create, Read, Update, Delete) operations.
- **FinanceTracker.py**: The central class/module that orchestrates the application. It calls methods from DatabaseManager and applies business logic to generate reports.
- **main.py**: The entry point for the CLI, presenting menus and capturing user input.

**5. Core Features and Functionality**

| Feature | Description | Technical Implementation Detail |
|---|---|---|
| **Add Transaction** | Records a new entry (Income or Expense). | Stores (date, amount, type, category_id, description) into the SQLite transactions table. |
| **View Transactions** | Lists all or filtered transactions. | SQL SELECT query. Filtered by date range, type, or category. Data is formatted and printed to the CLI. |

| | | |
|---|---|---|
| **Define Categories** | Manages a list of spending/income categories. | Stores (id, name, type) in a separate categories table. Uses foreign key constraint in the transactions table. |
| **Spending Report** | Calculates total spending for a given period (e.g., a month). | SQL aggregation (SUM()) and grouping (GROUP BY category_id) with a WHERE clause for the date range. |
| **Balance Calculation** | Calculates the running net balance (Total Income - Total Expenses). | A simple SUM(amount) grouped by type, then subtracting the expense sum from the income sum. |
| **Data Export** | Allows the user to export all recorded data. | Uses Python's built-in csv module to write database query results into a .csv file. |

**6. Implementation Highlights**

**A. Database Schema Design**

A normalized structure is used to ensure data integrity:

| Table Name | Fields | Data Type | Notes |
|---|---|---|---|
| **categories** | id | INTEGER | PRIMARY KEY |
| | name | TEXT | E.g., 'Groceries', 'Salary' |
| | type | TEXT | 'Income' or 'Expense' |
| **transactions** | id | INTEGER | PRIMARY KEY |
| | date | TEXT | Stored as 'YYYY-MM-DD' |
| | amount | REAL | Floating-point number |
| | type | TEXT | 'Income' or 'Expense' |
| | category_id | INTEGER | FOREIGN KEY reference to categories.id |
| | description | TEXT | Optional details |

**B. Error Handling and Robustness**

Robust input validation is crucial. The application uses Python's try...except blocks to handle:

- ValueError: For non-numeric input when an amount is expected.

- sqlite3.Error: For handling database connection failures or integrity issues (e.g., invalid foreign key constraints).

**7. Future Enhancements**

The following features are planned for future versions of PyTrack:

1. **Budgeting Module:** Implement monthly spending limits per category, alerting the user when a limit is exceeded.
2. **Recurring Transactions:** Allow users to mark transactions as recurring (e.g., monthly rent) to auto-populate future entries.
3. **Goal Tracking:** Feature to set financial goals (e.g., 'Save $5,000') and track progress against the net balance.
4. **GUI Integration:** Migrate the interface from CLI to a simple graphical user interface using a library like **Tkinter** or **PyQt** for a more user-friendly experience.

**8. Conclusion**

PyTrack serves as a practical, foundational project for applying Python development skills to a real-world problem. By successfully integrating database management (SQLite) with core application logic and a user-friendly interface, the project achieves its goal of creating a functional, persistent, and useful personal finance tracking tool. The project is scalable, with a clear path defined for advanced features like budgeting and GUI integration.