*This project proposal framework was generated by Gemini AI using the following prompt: “Write a project proposal for creating a personal finance tracker for a Phyton class”
Note: Of course we added our own details/ideas/project specifications during our team discussions

Team 8 Project Proposal: Personal Finance Tracker

---

1. Project Title

HHB Money Tracker - A Personal Finance Tracker

---

2. Introduction

The objective of this project is to create a Personal Finance Tracker, a Python-based application designed to help users effectively manage their income, expenses, and net savings. Navigating personal finances can be daunting without the appropriate tools, particularly for those new to financial planning. This application aims to provide a straightforward yet powerful interface for tracking transactions, categorizing expenditures, and visualizing overall financial health, ultimately promoting improved financial habits and informed decision-making. Our goal is to encourage better financial habits and enhance decision-making. By incorporating visual elements such as charts and graphs, we aim to make financial data more accessible, transforming numerical figures into valuable insights regarding spending patterns. The application will emphasize user-friendliness and clear presentation of financial information. We will work collaboratively each week and have established action items to ensure we are making progress towards our project goals.

---

3. Project Goals

The primary goals of this project are:

- To create an intuitive system for recording financial transactions (income and expenses).
- To enable categorization of expenses for better analysis.
- To provide summaries and visualizations of financial data.
- To assist users in setting and tracking budgets.
- To develop a robust and maintainable Python application suitable for a class project.
- Challenge Level: Beginner/intermediate

4. Key Features

The personal finance tracker will include the following core functionalities:

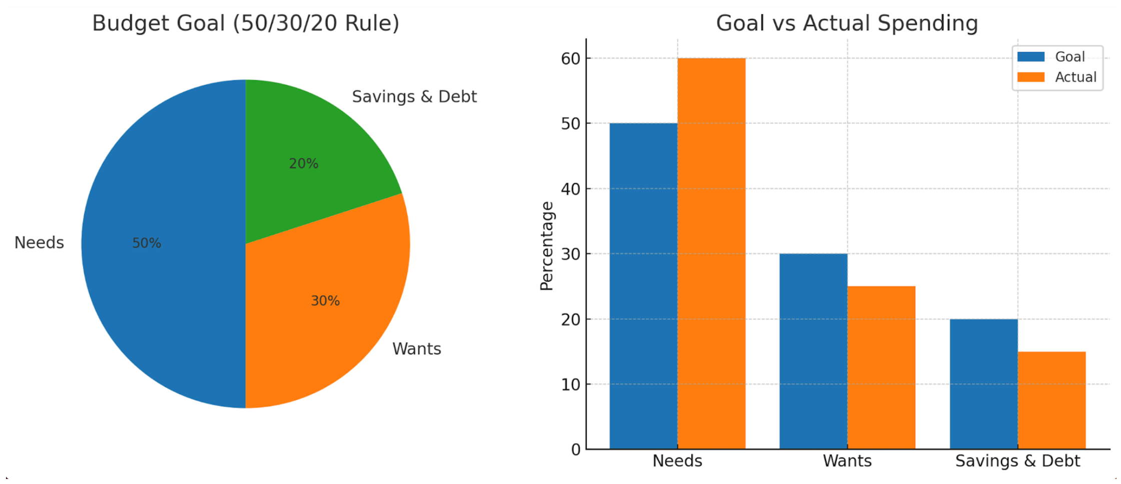
- **Transaction Entry:** Users can input details for each transaction, including date, amount, description, and type (income/expense).
- **Expense Categorization:** Assign categories to expenses (e.g., food, housing, transportation, entertainment) for detailed analysis.
- **Income Tracking:** Record various sources of income.
- **Budget Setting:** Allow users to set monthly budgets for different expense categories.
- **Financial Summaries:** Generate reports showing total income, total expenses, and remaining balance for a specified period.
- **Data Visualization:** Present financial data through simple textual or graphical summaries (e.g., bar charts for spending categories).
- **Data Persistence:** Save and load transaction data to/from a file (e.g., CSV or JSON) to ensure data is not lost between sessions.

Design/Features:

- Authentication of user (username/password, MFA)
    - User enters in what their user and password combination is
    - Request gets sent to server. The server checks the database to see if the user/password combination is in the database.
        - Database should have user/password values hashed for security purpose (don't want to store user/password exactly how it is in the database)
        - 2 scenarios:
            - If user/password combination is found (hash of the login credentials entered match the hash of whatever is in the database):
                - You can log in
                - Step 1: User enters username/password
                - Step 2: Backend hashes password & compares with stored hash
                - Step 3: If valid, server creates a session (or token)
                - Step 4: Session ID or token is sent back (via secure cookie)
                - Step 5: Each future request includes this token/cookie for authentication
            - If user/password combination is not found:
                - deny the login request and tell user they are not able to log in

        - Maybe implement a limit on how many login attempts per unit of time to mitigate the risk of brute force attacks
- Encryption
    - To maintain the security of requests being sent from user to server to database and back
        - HTTPS
    - Encrypt sensitive information like credit card numbers and bank account numbers since users need this info multiple times to do whatever they need for their financial use
- Track income and expenses
    - Users can input details for each transaction, including date, amount, description, and type (income/expense).
        - Importing transactions from linked bank/credit card accounts (Money in)
            - implementing a CSV/excel spreadsheet import of their statements
    - Creating a budget
        - Allows user to see how much money is allocated for each category they specify (being able to set a budget)
        - Users can assign categories to expenses (basic categories such as food, rent, gas, etc. and users can also specify their own categories for personalization)
        - If a user is new to creating a budget, we can have a visualization that briefly introduces them to general budgeting concepts (50/30/20 rule to make is user/beginner friendly)
    - Generate financial reports/graphs
        - Summary Generation/Graphic Visualizations
            - How much money in per week/month
            - How much spent per week/month
            - Show categories from highest to lowest spend
            - Show if you went over budget
            - Etc.
        - Users can search through previous expenses

“Make a basic chart explaining general budgeting concepts for a personal finance tracker”- chatgpt



---

5. Technology Stack

- **Programming Language:** Python 3.x
- **User Interface:** Command-Line Interface (CLI) for initial development. If time and scope allow, a basic GUI using Tkinter or Streamlit could be explored.
- **Libraries:**

SQLite for local data persistence

- Secure, offline storage: An SQLite database will be implemented to provide a lightweight, file-based storage system for all financial data. This approach ensures user privacy, as all data will be stored locally on the user's machine rather than on a remote server.
- Efficient transaction management: A simple, yet effective, database schema will be created with tables for transactions, income, categories, and accounts. This design minimizes data redundancy and optimizes the querying process for quick data retrieval and analysis.
- Schema:
  - id: INTEGER PRIMARY KEY (Unique identifier for each transaction)
  - date: TEXT (Date of the transaction)
  - description: TEXT (Description of the transaction, e.g., "Coffee", "Salary")
  - category: TEXT (User-defined category, e.g., "Food", "Income")
  - amount: REAL (The monetary value of the transaction)

Pandas for data manipulation and analysis

- Data import and processing: The pandas library will be used to read and process transaction data from various sources, such as CSV files exported from bank statements. It will convert this data into DataFrames, which are ideal for structured data analysis.

- Categorization and cleaning: Pandas will facilitate data cleaning operations, such as handling missing values and ensuring data type consistency. It can also automate the categorization of transactions based on keywords in the description field.
- Financial calculations: The library will be used to perform time-series analysis for forecasting expenses, calculating rolling averages for spending trends, and generating periodic reports.

NumPy for numerical operations

- Enhanced performance: While pandas will handle the core data structures, NumPy will provide the underlying numerical power for any complex calculations. Its array-based operations are significantly faster than native Python lists, allowing for more efficient analysis.
- Statistical foundation: NumPy's capabilities will be crucial for performing fundamental statistical analysis on financial data, such as computing means and standard deviations to assess spending volatility.

Matplotlib and Seaborn for data visualization

- Customizable visualizations (Matplotlib): Matplotlib will provide the foundational tools for creating static visualizations, allowing for granular control over every aspect of a chart. This will be used to generate clear, informative plots that visualize financial trends and patterns over time.
- Enhanced aesthetics and statistical graphics (Seaborn): Seaborn, built on Matplotlib, will be used to generate more aesthetically pleasing and complex statistical graphics with minimal code. This includes:
    - Line plots: To track spending over time.
    - Bar charts: To compare spending across different categories.
    - Heatmaps: To visualize correlations between spending habits.
- Clear, actionable insights: The visualizations will transform raw data into easy-to-understand charts, helping users quickly identify trends, manage budgets, and make better financial decisions.

---

6. Project Timeline

Week 1: Research and foundational setup

- **Research:** Analyze similar finance applications to understand standard features and best practices. Research pandas for data handling, sqlite3 for database integration, and matplotlib/seaborn for visualization concepts.

- **Proposal drafting:** Begin writing the project proposal. Define user stories, including recording transactions, generating reports, and viewing visualizations.
- **Environment setup:** Install Python and all necessary libraries (pandas, matplotlib, numpy, seaborn) in a virtual environment.
- **Database schema design:** Sketch the sqlite database schema, including a transactions table with columns for id, date, category, amount, and note.
- Convene virtual meeting on (Thursday 8/21 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Week 2: Finalize proposal and initial database

- **Finalize proposal:** Complete the project proposal, including a list of features, the overall design, a detailed timeline, and the technology stack.
- **Initial database implementation:** Write the Python code to initialize and connect to the sqlite database. Create the transactions table based on the designed schema.
- **Basic data loading (prototype):** Develop a simple script to add a few dummy transactions directly to the database to test the connection and schema.
- **Communications:** Convene virtual meeting on (Thursday 8/28 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Phase 2: Transaction Recording (Weeks 3–4)

Goal: Implement the core functionality for logging and categorizing transactions.

Week 3: Core transaction logic

- **Function development:** Write functions to perform basic CRUD (Create, Read, Update, Delete) operations on the transactions table:
  - add_transaction(date, category, amount, note)
  - view_transactions()
- **Data validation:** Implement basic input validation for transaction data to prevent incorrect entries, such as ensuring amounts are numbers and categories are valid.
  - **Front-end (user interface):** Basic input validation will be performed directly on the user-facing interface to provide immediate feedback and prevent obvious errors, such as:
    - **Data type verification:** Ensuring that numerical fields receive only numbers and date fields receive valid dates.
    - **Range checks:** Preventing negative values for transactions where only positive numbers are valid (e.g., income).
  - **Back-end (core application logic):** Before data is written to the SQLite database, a secondary, more comprehensive layer of validation will occur using Python functions. This ensures data integrity even if the front-end validation is bypassed.

    - **Cross-field validation:** Checking for logical inconsistencies, such as ensuring an "expense" transaction has a negative amount.
    - **Database constraints:** The SQLite database schema will enforce data integrity checks like primary keys, foreign keys (e.g., ensuring a category ID exists), and non-null constraints for essential fields.
- **Field Validation rules**
    - **Date:** ISO (YYYY-MM-DD
    - **Amount:** numeric, >0 (store positive)
    - **Type:** one of {income, expense}
    - **Category:** Must exist in categories list
    - **Description:** printable text, <_ 120 chars
    - **Duplicate transaction:** fingerprint on (date, description, lower(), amount, type) must be unique
- **Storage Layer Checks**
    - **CSV (baseline):** verify required headers; validate types on read; isolate bad rows (line number + reason) during imports; continue processing good rows.
    - **SQLite (stretch):** add NOT NULL, UNIQUE (category names), and constraints to enforce referential integrity
- **Error handling and reporting**
    - A standardized approach to error handling will be implemented to prevent application crashes and provide informative, actionable feedback to the user. The strategy includes:
    - **Specific exceptions:** The application will use specific Python exceptions (e.g., ValueError, TypeError) rather than broad, generic except blocks. This allows for precise handling of different error types.
    - **Meaningful error messages:** Error messages will be user-friendly, specific about what went wrong, and provide clear instructions on how to resolve the issue. For example, instead of a generic "An error occurred," the user might see, "The date entered is invalid. Please use the YYYY-MM-DD format".
    - **Robust import handling:** When importing CSV files, the system will use pandas to attempt data type conversions, but will include error handling to gracefully report rows with invalid data rather than failing the entire import. For example, a try-except block can be used with pd.to_datetime to safely convert dates.
    - **Logging:** For internal debugging and tracking purposes, exceptions will be logged to a file. This log will include timestamps and detailed tracebacks to help with troubleshooting without exposing sensitive technical details to the user.
    - **Graceful failure:** In the event of a critical error, the application will not crash. Instead, it will inform the user of the problem, log the incident, and attempt to return the application to a stable state. For example, if a

        database file is corrupted, the application can alert the user and offer steps for restoration.
  - **Summarized:**
    - Multi-layered validation (UI + core logic + database)
    - Example rules: type checks, range checks, cross-field checks, DB constraints
    - Error Handling: specific exceptions, meaningful error messages, robust imports, and logging, graceful
  - **Categorization:** Create a mechanism for defining and managing expense categories. Consider storing this in a separate database table or a configuration file.
  - Convene virtual meeting on (Thursday 9/4 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Week 4: Data integration with Pandas

- **Database-to-DataFrame:** Create a function to fetch transactions from the sqlite database and load them into a pandas DataFrame for easy manipulation and analysis.
- **DataFrame-to-Database:** Implement the reverse function to save new or updated transactions from a DataFrame back into the database.
- **Data preprocessing:** Use pandas and numpy to clean and preprocess the transaction data, such as converting data types and handling missing values, which will be useful for later reporting.
- **Communications:** Convene virtual meeting on (Thursday 9/11 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Phase 3: Reporting and Exception Handling (Weeks 5–6)

Goal: Add data analysis features, robust error handling, and a command-line interface (CLI).

Week 5: Reporting with Pandas and NumPy

- **Descriptive reports:** Create functions that use pandas to generate financial reports, such as:
  - Monthly or weekly summaries of income and expenses.
  - Total spending by category.
  - Average transaction amount.
- **Advanced analysis:** Implement more sophisticated analysis using pandas and numpy to identify spending trends and patterns over time.
- **CLI framework:** Begin developing the command-line interface for user interaction using a library like argparse or Click. Create menu options for adding transactions and viewing reports.

- **Communications:** Convene virtual meeting on (Thursday 9/18 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Week 6: Exception handling and robust CLI

“Describe some exceptions to handle in creating a basic personal finance tracker in python” - google gemini

Most exceptions will fall into three main categories: invalid user input, file-handling issues, and errors related to calculations and data processing

Invalid user input:

- ValueError: This is a very common exception for a finance tracker. It occurs when a function receives an argument of the correct type but an inappropriate value.
- The except ValueError block can print a message asking the user to enter a valid number and then prompt for input again using a while loop.

Exceptions for file handling:

- FileNotFoundError: This exception occurs when the program attempts to open a file that does not exist.
- PermissionError: This is raised when the user does not have sufficient access rights to read or write a file.
- Catching this error and alerting the user that the program lacks the necessary permissions can prevent a crash

Exceptions for calculations and data:

- ZeroDivisionError: Raised when a number is divided by zero.
- Catching ZeroDivisionError and returning a friendly message (e.g., "No transactions to calculate an average.") is the correct approach

- **Comprehensive exception handling:** Implement try-except blocks throughout the application to gracefully handle common errors, such as database connection failures, file-not-found errors, and invalid user input.
- **User feedback:** Enhance the CLI to provide clear, informative feedback to the user, including success messages and detailed error reports.
- **Refine CLI:** Flesh out the CLI with all necessary commands for managing the personal finance data. Ensure the interface is intuitive for users.
- **Communications:** Convene virtual meeting on (Thursday 9/25 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Phase 4: User Interface and Final Integration (Weeks 7–8)

Goal: Develop the graphical user interface (GUI) with visualizations and integrate all components into a cohesive application.

Week 7: Visualization and GUI development

- **Matplotlib and Seaborn plots:** Create visualization functions using matplotlib and seaborn. Generate plots like:
    - Bar charts of spending by category.
    - Line plots to show spending trends over time.
    - Pie charts for category breakdowns.
- **Turtle Graphics for visualization:** Implement the requested turtle graphics visualizations. This could be a unique feature, such as a simple, animated representation of a user's financial progress.
- **GUI framework selection and setup:** Choose a GUI library (tkinter) and set up the basic window and layout.
- **Communications:** Convene virtual meeting on (Thursday 10/2 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

Week 8: Integration and project finalization

- **Full GUI integration:** Embed the CLI functions and visualization plots into the GUI. Add buttons and text fields to allow users to interact with the application, including adding transactions and displaying reports.
- **User testing and feedback:** Conduct basic user testing to identify and fix usability issues.
- **Final review and documentation:** Review all code for clarity and efficiency. Write comprehensive documentation, including a README.md file explaining how to install and use the application.
- **Final polish:** Add final touches to the user interface and code to ensure a seamless and robust user experience.
- **Communications:** Convene virtual meeting on (Thursday 10/9 @ 7PM) to discuss project status and assign weekly project assignments. In addition to these virtual meetings, the team will communicate regularly via chat.

---

7. Expected Outcomes

Upon completion, the project will deliver a functional Personal Finance Tracker capable of:

- Allowing users to manage their daily financial activities.

- Providing insights into spending patterns.
- Assisting with budget adherence.
- Demonstrating proficiency in Python programming, file handling, data structures, and basic application design.

Team 8 Member Signatures:

Jeremy Henderson

Aamina Khuram

Rohan Singh

Ria Patel

Priscilla Gomez

Travis Villanueva