BudgetBuddy: Your Household Budget Tracker

Documentation

Submitted by: ah_codes & Mike22

Date: May 30, 2025

Content

1 About the Application

BudgetBuddy is a lightweight, web-based tool for managing personal finances without the overhead of a full database. Built entirely in Python using Streamlit, it keeps all data in memory (Streamlit's session state) and lets users import historical transactions via CSV or start from scratch. Once running, you can view aggregated totals, add new income or expense entries, edit or remove past records, and instantly see your spending visualized in bar and pie charts. This approach makes BudgetBuddy ideal for anyone who wants a code-centric budgeting app that's simple to run and easy to extend.

2 Language & Resources

BudgetBuddy requires Python 3.8 or higher. To prepare your environment, open your terminal (or Anaconda Prompt on Windows) and install the three main libraries:

pip install streamlit pandas altair

- Streamlit provides the web interface, automatically generating forms, tables, and charts from Python code.
- Pandas handles reading, writing, and filtering CSV data in DataFrame structures.
- Altair produces the interactive bar and pie charts that help you explore your budget.

Installing these packages ensures that all core features (data import/export, UI rendering, and charting) work out of the box.

3 Structure & UI

BudgetBuddy divides its interface into a sidebar and a main area. In the sidebar you select your desired time frame (such as the past week, month, year, or a custom date range) and you can upload an existing CSV or download the current session's data. A reset button clears all entries, and a small info section shows the app version and author.

The main area presents your data in four parts:

1. At the top, two metrics display Total Income and Total Expenses, so you instantly grasp your overall balance.
2. Below that, an "Add New Entry" form lets you type in Date, Name, Description, Amount, Category, Type (Income or Expense), Currency, Payment Method, and Project.
3. Once you submit, the new entry appears immediately in the table underneath. This table shows every transaction with options to select a row for editing or deletion.
4. Finally, at the bottom, interactive grouped bar charts break down transaction amounts by Category, Project, Payment Method, Currency, or Type, and a donut chart shows each category's share of the overall total (income + expenses).

4 Application Logic

While the code does not strictly separate logic into standalone Python functions, its structure is cleanly modular, with each section of the app fulfilling a well-defined purpose. Streamlit's event-driven model and use of st.session_state allow for interactive, stateful programming without the need for a traditional function-based architecture. Nevertheless, the code follows clear logical blocks that behave like functional components, ensuring maintainability, readability, and testability.

4.1 Data Management

The app manages its dataset using Streamlit's session state (st.session_state.data) and offers a full cycle of data operations:

- Initialization: If no data exists, an empty DataFrame with strict column ordering is created: COLS_ORDER=["Select","Date","Name","Description","Amount","Category","Type", "Currency", "Payment Method", "Project"]
- Importing CSV: Uploaded files are read via pandas.read_csv(), validated against the required structure, and sanitized:
    - Dates are parsed and coerced
    - Amounts are cleaned of formatting issues
    - Expense entries are automatically converted to negative values
    - Income entries are enforced as positive

- Sanitizing Inputs: The custom sanitize_input() function ensures user text fields are clean and CSV-safe by removing commas to avoid delimiter errors during export.
- Resetting Data: A dedicated button clears the entire session state, allowing users to start fresh.
- Exporting Data: At any time, the current dataset can be exported as a CSV file via Streamlit's download_button().

A minimal example CSV (displayed as table) might look like:

Date	*Name*	*Description*	*Amount*	*Category*	*Type*	*Currency*	*Payment Method*	*Project*
05/01/2025 00:00	Monthly Salary	April net salary	6000	Income	Income	CHF	Bank Transfer	Personal
05/02/2025 00:00	Groceries	Monthly food shopping	-150	Groceries	Expense	CHF	Debit Card	Personal

4.2 Entry Handling

New entries are collected through a form that includes fields for all required attributes. On submission, the app:

- Validates and formats the input
- Constructs a dictionary
- Appends it to the main DataFrame using pd.concat()
- Re-renders the UI via st.rerun()

4.3 Editing & Deleting

The app allows inline editing of the dataset via st.data_editor():

- Rows can be selected via a "Select" checkbox
- When exactly one row is selected, an edit form appears
- Changes are written directly to the session DataFrame using .at[]
- A delete button allows users to remove the selected entry by index

4.4 Visualization

Data analysis is a core part of BudgetBuddy. The app uses Altair to generate:

- Grouped Bar Charts for visualizing amounts over time, grouped by:
    - Project
    - Payment Method
    - Category
    - Type
    - Currency
- Pie/Donut Chart to show each category's share of the combined total (income + expenses), with slice sizes proportional to the sum of all transaction amounts per category

These charts update dynamically based on the filtered date range and data state.

4.5 Error Handling

Robust error management is included:

- Missing columns during CSV upload are detected and reported
- Non-numeric amounts and invalid dates are coerced or skipped
- Attempting to edit without a selected row results in a visible warning
- Adding or editing entries without loading data prompts the user to upload a file
- If the user selects an invalid time range, they get notified

4.6 Code Design Principles

The app adheres to strong design principles:

- Single-responsibility blocks: Each section (data load, entry form, display, charts) is clearly separated
- Descriptive naming and consistent layout
- Use of helper logic like sanitize_input() to enforce clean, reusable behavior
- PEP 8-compliant structure with clear indentation and comments

This makes the codebase professional, readable, and extensible.

Have fun using the app!