Student Budget Tracker

A Python Application with Tkinter & SQUITe

Submitted By: Rathis Raj.s
Register Number: 25BCY10030

Introduction

Effective financial management is an essential skill for students, particularly those operating within fixed or limited monthly allowances. The Student Budget Tracker application has been developed to address this need by providing a structured and user-friendly platform for monitoring personal expenses. The system enables users to define a monthly budget, record daily expenditures across various categories, and receive real-time feedback on their spending behavior.

The application leverages Python's Tkinter library to deliver an accessible graphical user interface, ensuring ease of interaction for non-technical users. For persistent data handling, the system integrates an SQLite database, which enables secure storage, rapid retrieval, and reliable management of financial records. By consolidating these features, the Student Budget Tracker supports informed decision-making and encourages responsible financial practices among students. This project also highlights the practical application of Python programming, database management, and GUI development in solving real-world problems.

Problem Statemen

Managing personal finances is a recurring problem that students with an inherently constrained monthly budget face. Traditional methods of expense tracking involve a great deal of inefficiency, time consumption, and inaccuracies in manual logs, notes, or spreadsheets. In turn, this greatly limits students from getting timely information regarding their pattern of spending and, hence, their unwanted expenditure, which may further reduce the quality of life.

This illustrates the very root of the problem: the inability to have in place a simple, easy, and automated system for the tracking and control of the limited funds by students. Lacking this tool, students cannot contextualize their spending within the fixed monthly allowance, thereby making it cumbersome to stay on course with financial goals.

Therefore, there is a felt need for an efficient, automated system that allows students to log expenditures, track a monthly budget, and monitor their current financial status. The Student Budget Tracker solves this problem by providing a structured digital framework that encompasses easy data entry, persistent storage, and analytical feedback-such as remaining budget and visualization-based warnings-on spending habits. With this facility, students can enforce better financial discipline by making prudent and timely decisions supported by accurate and current information on expenditure, thereby reducing the possibility of overspending and financial hardship.

Functional Requirements

1: Budget Configuration

The system shall allow the user to specify an initial monthly budget limit and update it thereafter. The value of this budget shall be persisted for subsequent sessions.

2: Expense Entry Management

The system should allow users to record new expenses by specifying necessary details like category, amount, date, and description. The system shall validate inputs so as not to allow incorrect or incomplete data.

3: Expense History Visualization

The system shall present a detailed list of all logged expenses in reverse chronological order. The interface shall be implemented utilizing a structured table component to support readability and scrolling.

4: Automatic Budget Calculation The system shall automatically compute the total expenditure, determining the remaining budget. The calculation shall update dynamically on additions of new expenses or changes in the budget.

5: Smart Budget Alerts The system shall provide visual indicators reflecting the user's financial status. An alert will be activated when the remaining budget falls below a certain threshold value predefined as 20% of the limit, or becomes negative.

6: Dependable Data Persistency A SQLite database shall persistently store all financial records, including expenses and budget configurations. In addition, the system shall guarantee the consistency, accuracy, and durability of data across sessions.

Non-functional Requirements

1: Usability

The system shall be easy to use, providing a consistent user interface that ensures ease of interaction by students with minimal technical background. All GUI components shall be arranged in a clearly structured layout to support easy reading and minimize user errors.

2: Performance Efficiency

The application should provide almost instantaneous response times for typical operations such as entry of expenses, retrieval of records, and computation of budgets. During normal usage, the system should not show any perceptible UI freezes or delays.

3: Reliability and Data Integrity

The system shall ensure the accuracy, consistency, and durability of financial data storage through the SQLite database. It needs to handle data integrity across multiple sessions and recover gracefully in case of unexpected closures.

4: Security and Input Validation

The system shall validate all user inputs to prevent malformed or harmful data. Database operations shall use parameterized queries to protect against SQL injection attacks and ensure secure handling of user records.

5: Maintainability and Modularity

The application shall be developed in a modular fashion such that the GUI and database logic are kept clearly separate. This design approach will support ease of debugging, testing, and future enhancements.

6: Portability and Platform Independence

The system shall operate consistently across major desktop environments - Windows, macOS, Linux - with only standard Python libraries, Tkinter and SQLite, thereby ensuring wide accessibility without additional installation overhead

System Architecture

The Student Budget Tracker is designed using a two-tier architectural model, comprising a presentation layer and an application/data layer. This structure ensures a clear separation of concerns, enhances maintainability, and supports efficient data management.

1. Presentation Layer (Tkinter GUI)

- The presentation layer is implemented using Python’s Tkinter library, which provides the graphical interface through which the user interacts with the system. It is responsible for:
- Capturing user inputs such as budget value and expense details
- Displaying updated expense lists and budget status (remaining budget, alerts, etc.)
- Triggering backend operations through event-driven methods
- This layer acts as the client component of the architecture.

2. Application Logic & Data Access Layer (Backend.py)

The backend layer handles the processing and persistence of financial data.

It performs the following responsibilities:

- Initializing and managing the SQLite database
- Executing CRUD operations (Create, Read, Update) for expenses and budget
- Validating and sanitizing user inputs
- Returning structured data to the GUI for display
- Ensuring secure execution of SQL queries using parameter substitution
- This layer acts as the server component, encapsulating the system’s business logic.

3. Database Layer (SQLite)

The system uses an embedded SQLite database stored locally in a file (tracker.db).

It provides:

- Persistent storage of expenses and budget information
- Lightweight, serverless operation suitable for a standalone desktop application

- Two relational tables:
- budget(id, total_limit)
- expenses(id, category, amount, date, description)

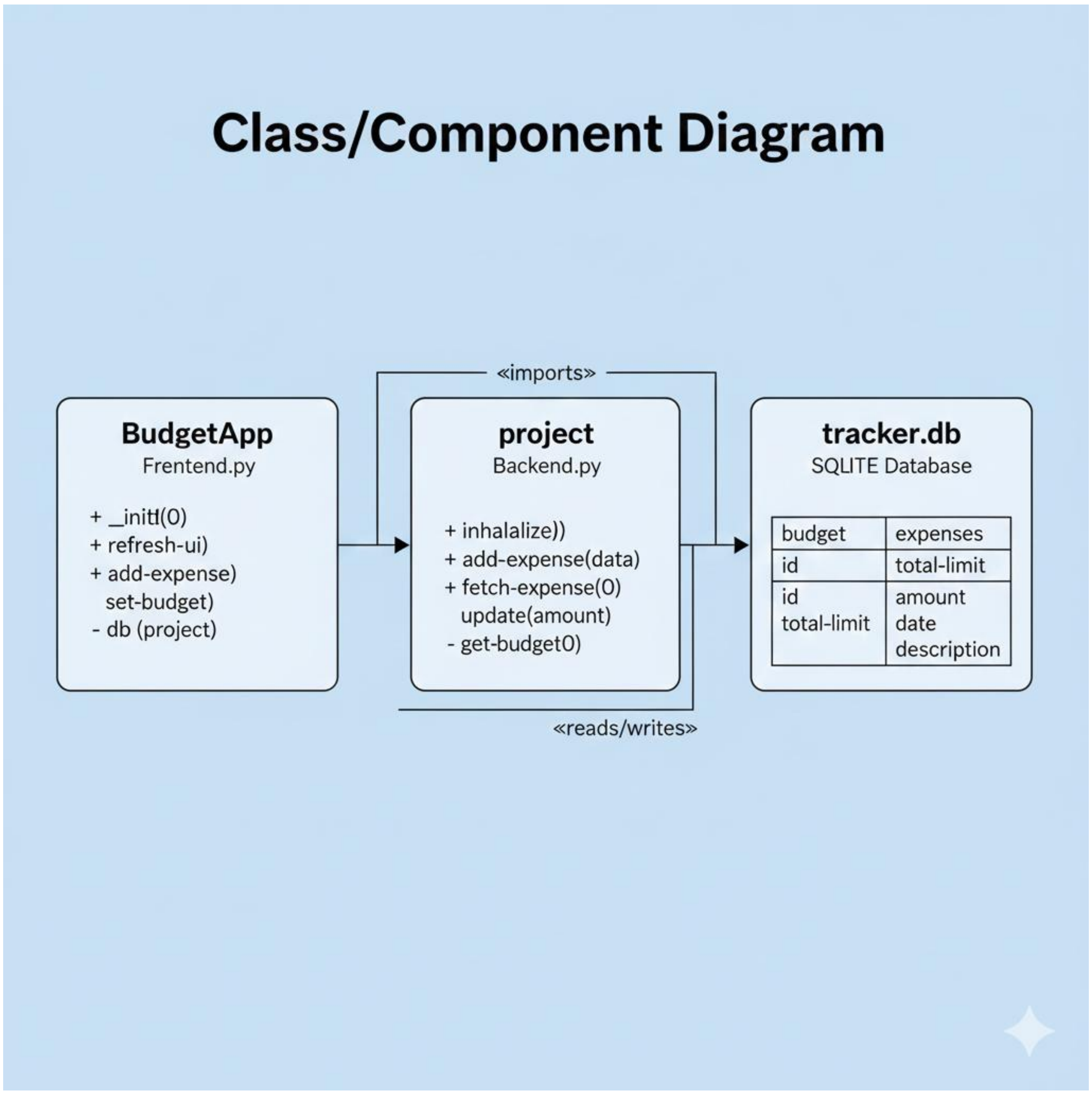
Data Flow

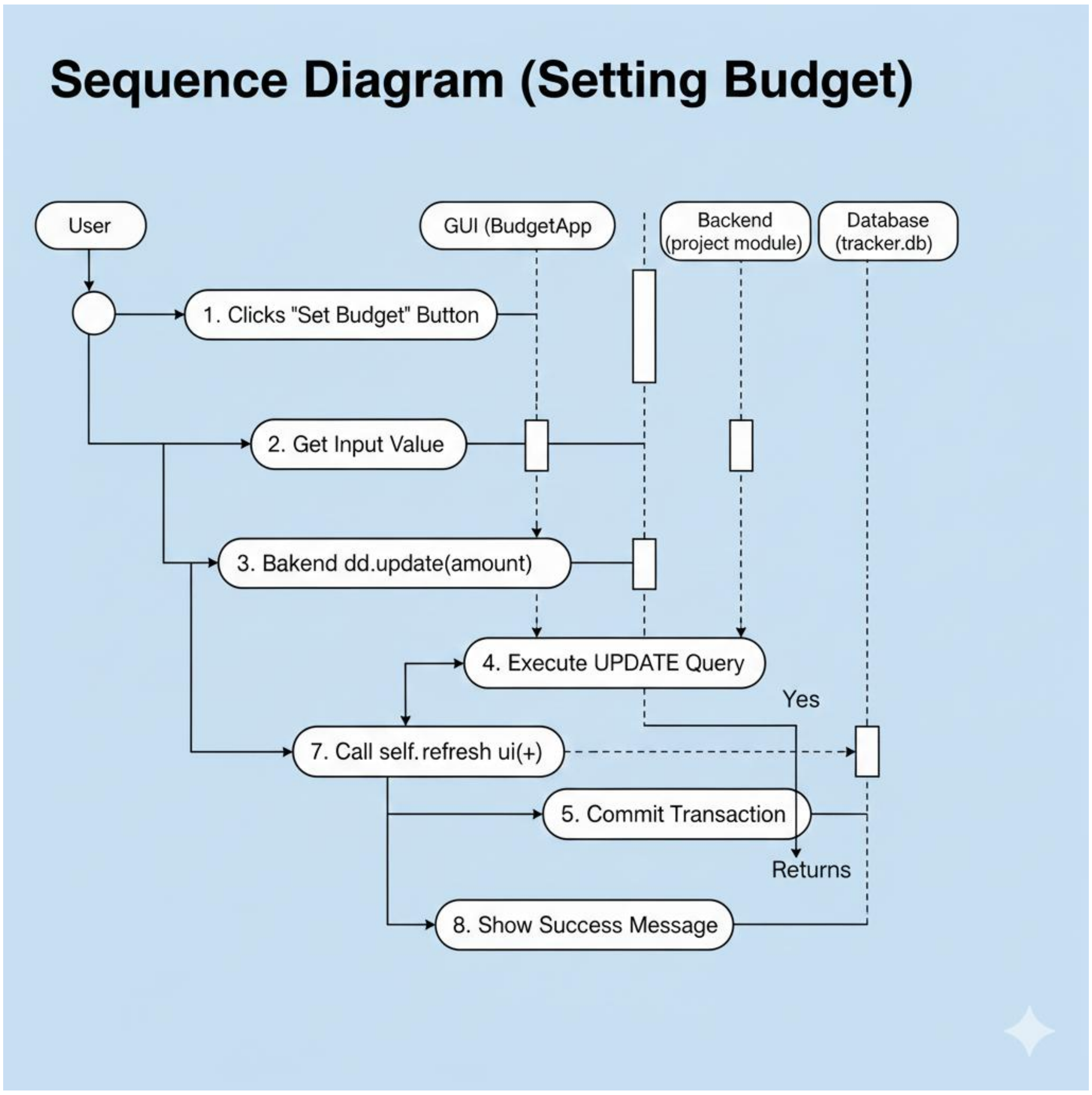
- User performs an action (e.g., add expense).
- GUI forwards the request to the backend.
- Backend validates input and updates the SQLite database.
- GUI retrieves updated values and refreshes the display.

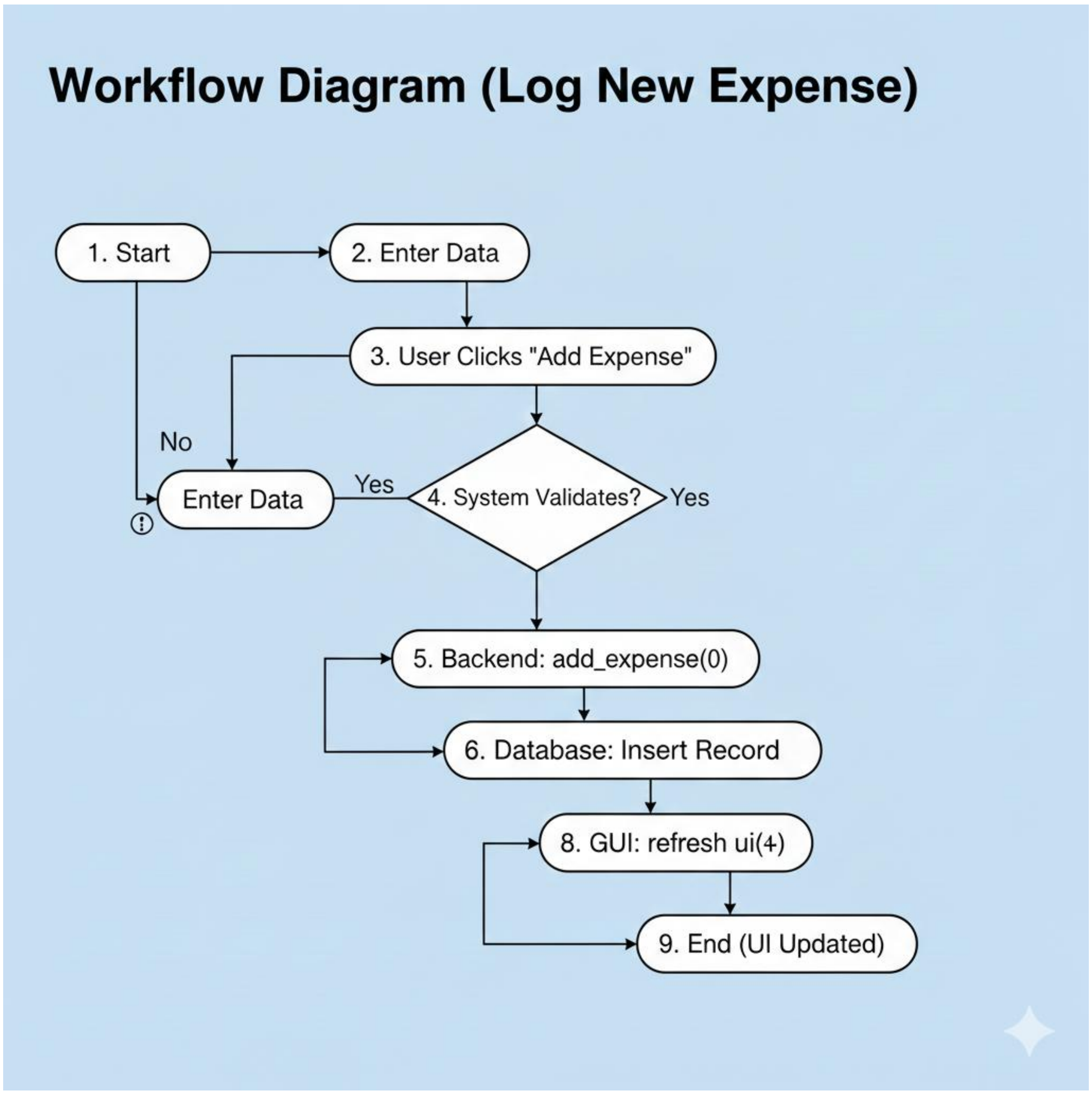
This architecture delivers a streamlined, modular, and efficient solution suitable for personal finance tracking.

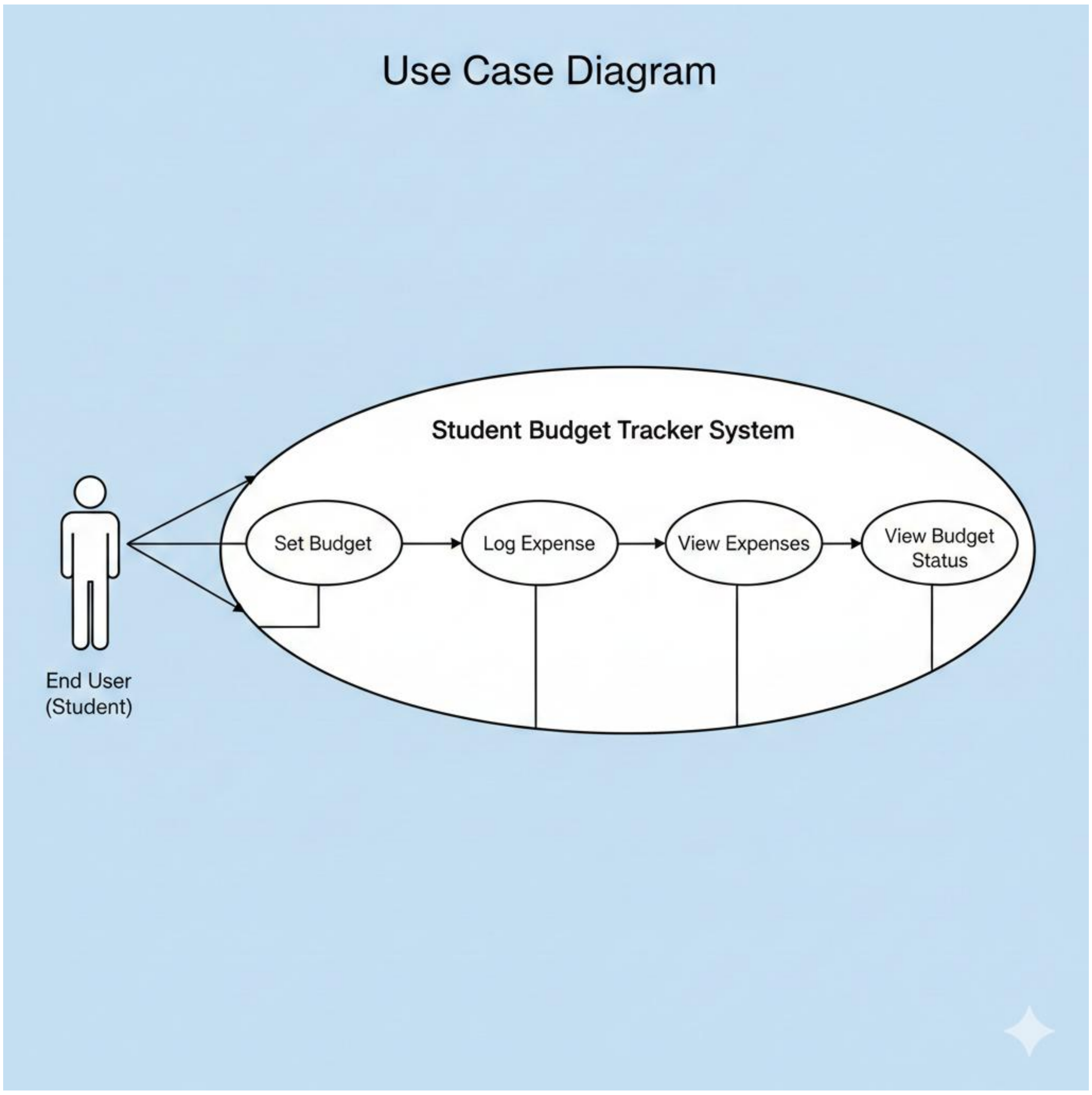
Design Diagrams

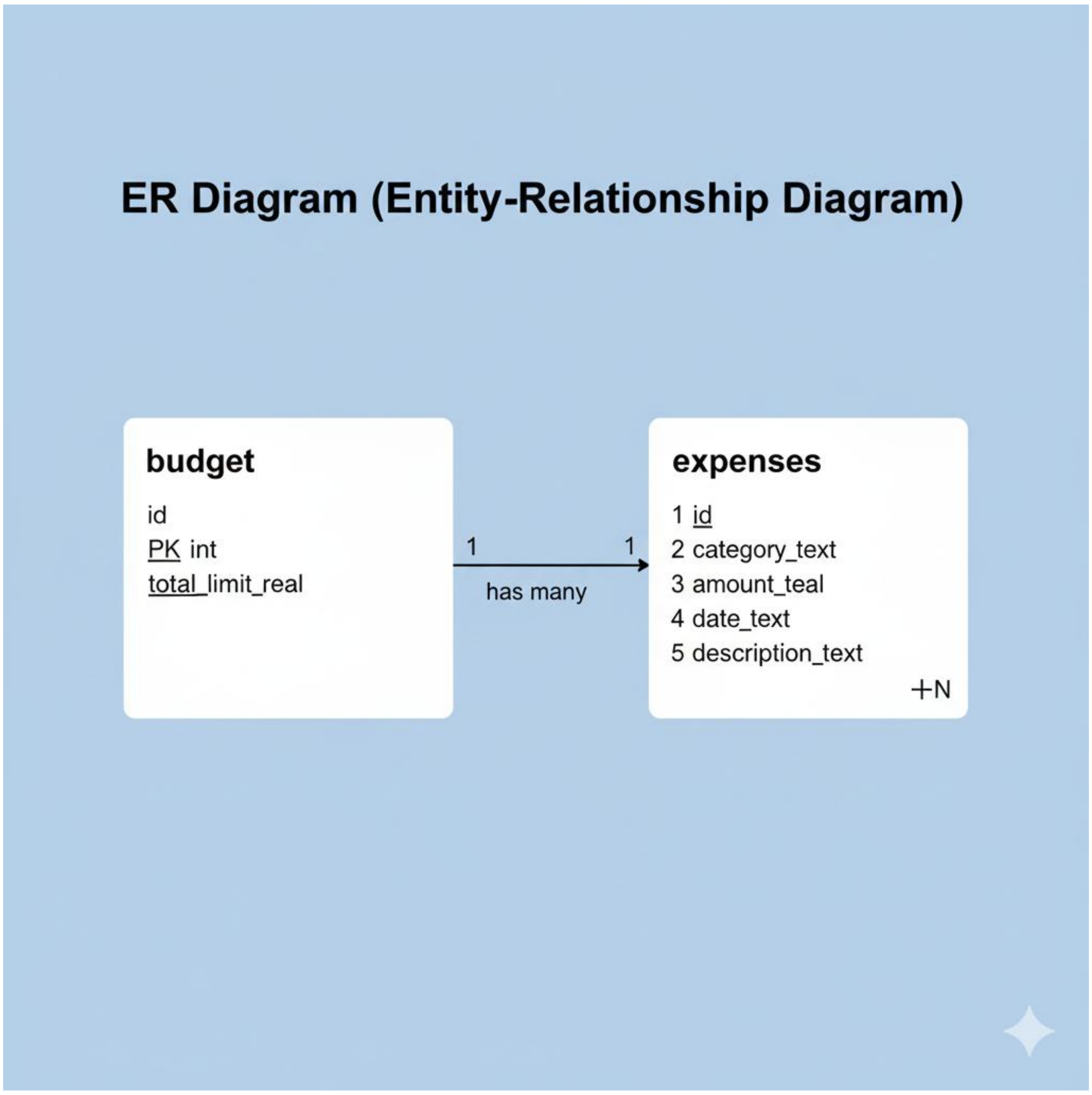
The following diagrams illustrate the structure, interactions, and behavior of the Student Budget Tracker system.


Class/Component Diagram
«imports»
BudgetApp
Frentend.py
+ _initl(0)
+ refresh-ui)
+ add-expense)
set-budget)
- db (project)
project
Backend.py
+ inhalalize))
+ add-expense(data)
+ fetch-expense(0)
update(amount)
- get-budget0)
tracker.db
SQLITE Database
budget
id
id
total-limit
expenses
total-limit
amount
date
description
«reads/writes»


Sequence Diagram (Setting Budget)
User
GUI (BudgetApp
Backend (project module)
Database (tracker.db)
1. Clicks "Set Budget" Button
2. Get Input Value
3. Bakend dd.update(amount)
4. Execute UPDATE Query
Yes
7. Call self.refresh ui(+)
5. Commit Transaction
Returns
8. Show Success Message


Workflow Diagram (Log New Expense)
1. Start
2. Enter Data
3. User Clicks "Add Expense"
No
Enter Data
Yes
4. System Validates?
Yes
5. Backend: add_expense(0)
6. Database: Insert Record
8. GUI: refresh ui(4)
9. End (UI Updated)


Use Case Diagram
Student Budget Tracker System
Set Budget
Log Expense
View Expenses
View Budget Status
End User (Student)


ER Diagram (Entity-Relationship Diagram)
budget
id
PK int
total_limit_real
1
1
has many
expenses
1 id
2 category_text
3 amount_teal
4 date_text
5 description_text
+N

Design Decisions & Rationale

The design decisions for the Student Budget Tracker were guided by the goals of simplicity, usability, and modularity, ensuring the application effectively supports student financial management.

1. Tkinter for Graphical User Interface

Tkinter was selected as the GUI framework due to its native availability within Python, cross-platform compatibility, and minimal setup requirements. Its widget set is sufficient for creating structured interfaces involving forms, tables, and status indicators. This choice aligns with the objective of developing a lightweight desktop application without third-party dependencies.

2. SQLite as the Database Engine

SQLite was chosen as the data storage solution because it offers a serverless architecture, making it ideal for standalone applications. It provides transactional support, fast queries, and persistent storage in a single file, ensuring reliability and ease of deployment. This decision eliminates the need for additional database installations while maintaining robust data management capabilities.

3. Modular Separation of Frontend and Backend

To improve maintainability and scalability, the system architecture separates the GUI layer (Frontend.py) from the data management and business logic layer (Backend.py). This modular design supports independent testing, easier debugging, and cleaner code organization. It also enables future extensions—such as adding analytics—without modifying core logic.

4. Simplified Budget Model

The budget table is intentionally designed to store a single record representing the current monthly limit. This reduces complexity in database queries and ensures predictable behavior when retrieving or updating the budget.

5. Visual Alert Mechanism

A color-coded alert system (green, orange, red) was adopted to provide intuitive feedback on financial status. These alerts help users quickly identify budget thresholds without needing manual calculations, improving usability and encouraging responsible financial decisions.

Implementation Details

The Student Budget Tracker application was implemented using Python, following a modular and extensible architecture. The system integrates a graphical user interface with a lightweight database to provide students an efficient and user-friendly platform for managing monthly expenses.

1. Technologies Used

- Programming Language: Python
- GUI Framework: Tkinter
- Database: SQLite3
- Additional Modules: Datetime (for handling dates), Tkinter TreeView (for tabular display)

2. Application Architecture

The implementation is divided into two main modules to ensure modularity and maintainability:

- **Frontend Module (Frontend.py)**

The frontend handles all user interactions and visual components. Key responsibilities include:

- Designing and rendering the GUI using Tkinter widgets
- Providing input fields for date, category, and amount
- Managing buttons for Add, Update, Delete, and Set Budget
- Displaying expenses using a TreeView table
- Showing remaining budget and status alerts (Green, Orange, Red)
- Sending user input to backend functions

- **Backend Module (Backend.py)**

The backend manages all application logic and data operations. Its functionalities include:

- ➢ Initializing the SQLite database
- ➢ Creating and maintaining the budget and expenses tables
- ➢ Performing CRUD (Create, Read, Update, Delete) operations
- ➢ Calculating total expenses and updating remaining budget
- ➢ Ensuring that only a single budget record exists (id=1)
- ➢ The database schema consists of:
- ➢ budget (id, amount) – stores the monthly budget
- ➢ expenses (id, date, category, amount) – stores expense entries

3. Core Functionality Flow

Setting the Budget

- ➢ User enters the desired monthly budget.
- ➢ Backend inserts or updates the record in the budget table.
- ➢ The UI reflects the updated budget and recalculates remaining balance.

Adding an Expense

- ➢ User fills in date, category, and amount.
- ➢ Backend validates and stores the record.
- ➢ TreeView table refreshes to display updated expenses.
- ➢ Remaining budget is recalculated instantly.

Updating an Expense

- ➢ User selects a record in the table and edits the fields.
- ➢ Backend updates the database entry.
- ➢ UI refreshes to reflect the changes.

Deleting an Expense

- Selected record is removed from the database.
- Remaining budget and alerts are recalculated automatically.

4. Budget Alert Mechanism

A color-coded alert system helps users understand their financial status at a glance:

- Green: Within budget
- Orange: Approaching budget limit
- Red: Budget exceeded
- The color updates dynamically after each expense operation

5. Error Handling & Input Validation

The system includes:

- Validation to prevent empty or invalid inputs
- Error controls for non-numeric values
- Safe database operations using try-except blocks
- Logic to ensure consistent and accurate expense calculations

6. Data Persistence

SQLite ensures all data is stored locally and remains available even after closing the application.

On startup, the application:

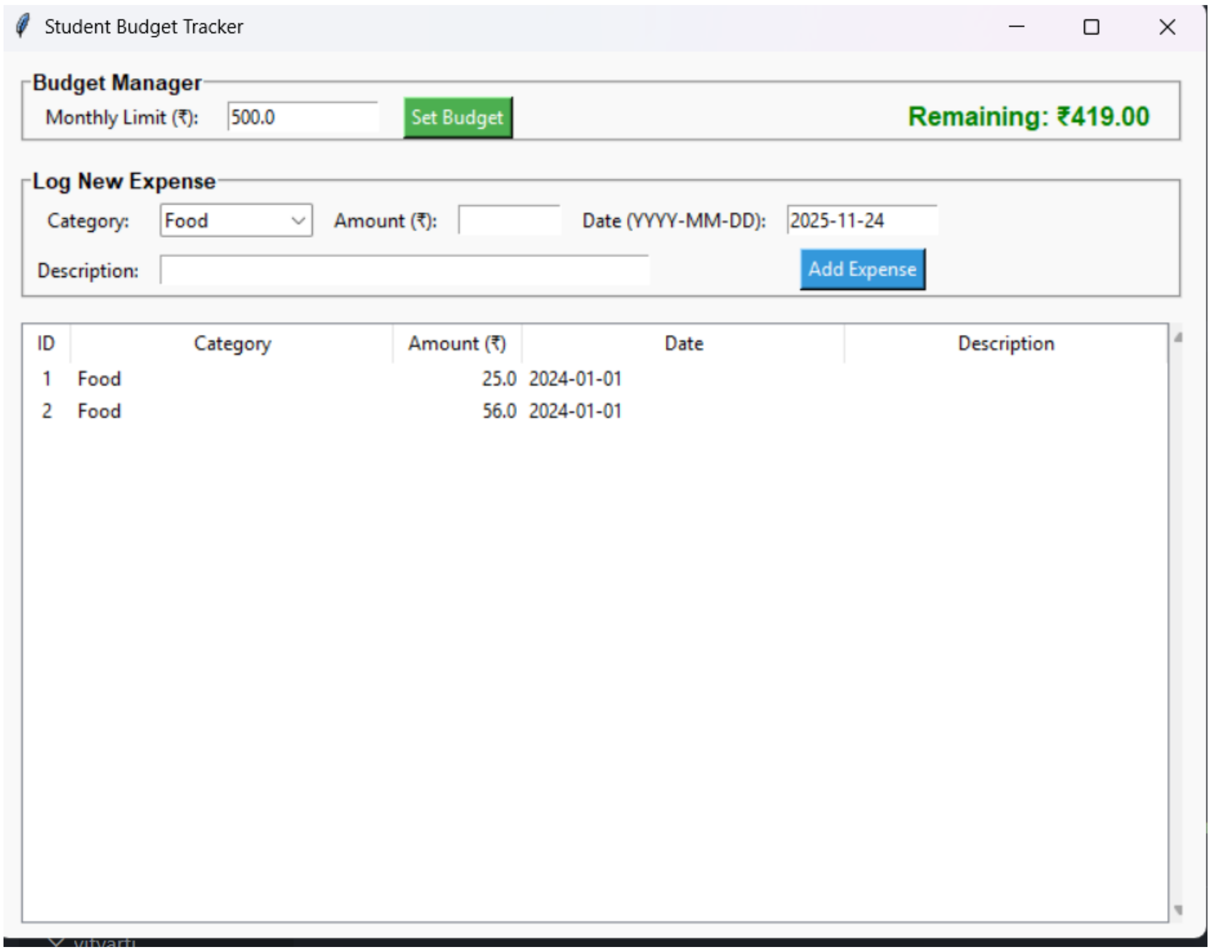
- Creates tables if they do not exist
- Loads existing budget and expenses into the interface

7. Extensibility

The implementation is structured to support future enhancements such as:

- Filtering and sorting expenses
- Graphical charts for monthly analysis
- Exporting data to CSV or PDF
- User authentication
- Monthly reset and history tracking

Screenshots / Results


Student Budget Tracker

Budget Manager

Monthly Limit (₹): 500.0 | Set Budget | **Remaining: ₹419.00**

Log New Expense

Category: Food | Amount (₹): | Date (YYYY-MM-DD): 2025-11-24

Description: | Add Expense

ID	Category	Amount (₹)	Date	Description
1	Food	25.0	2024-01-01	
2	Food	56.0	2024-01-01	

Testing Approach

Testing Approach

The Student Budget Tracker application was tested extensively to ensure reliability, correctness, and a smooth user experience. The testing process primarily involved manual functional testing, supported by validation checks and scenario-based testing. The goal was to verify that each module performed as expected and that the system behaved correctly under different input conditions.

1. Functional Testing

Functional testing was conducted to verify that all features met the specified requirements.

a. Budget Setting Test

Entered valid numeric values for the monthly budget and confirmed that:

- The budget table updated correctly.
- The remaining budget was recalculated.
- The updated value reflected immediately in the UI.
- Tested invalid inputs (e.g., empty values, negative numbers, text input) to ensure appropriate error messages or input rejections

b. Expense Logging Test

Added expenses with different values and categories.

Confirmed that:

- Each entry was stored in the SQLite database.
- The TreeView table reflected the new data instantly.
- The total expenses and remaining budget updated correctly.

c. Expense Updating Test

Modified existing expense records.

Checked whether updated records:

- Refreshed correctly in the UI.
- Overwrote the previous database values.

d. Expense Deletion Test

Deleted selected expenses.

Confirmed that they were removed from:

- The database.
- The TreeView table.
- The total calculation logic.

2. Validation Testing

- ➢ The application includes several validation checks to prevent improper usage.
- ➢ Attempted to submit empty fields to ensure they were detected.
- ➢ Entered invalid numeric values (e.g., "abc", negative values) to verify error handling.
- ➢ Ensured the date field accepted only proper forma

3. Calculation Verification

- Tests were conducted to ensure the correctness of budget calculations.
- Added multiple expenses and manually verified total spending.
- Observed changes in the Remaining Budget label after each operation.
- Confirmed that the alert color (Green/Orange/Red) changed appropriately based on:

  >Remaining < 20% of budget → Orange
  >Remaining < 0 → Red
  >Otherwise → Green

4. Database Consistency Testing

- ➢ The database was checked for errors or inconsistencies:
- ➢ Verified that duplicate budget entries were not created (only one record allowed).
- ➢ Ensured expense entries had unique IDs.
- ➢ Updated and deleted records to confirm data was consistent after each operation.

5. User Interface Testing

- The GUI was evaluated for responsiveness and usability:
- Ensured all buttons triggered the correct backend actions.
- Verified the TreeView scrollbar worked smoothly.
- Checked layout alignment across different window sizes.

6. Edge Case Testing

Unusual or unexpected scenarios were tested:

- Adding a very large number as expense amount.
- Setting budget to zero and then adding expenses.
- Adding expenses without any budget set.
- Closing and reopening the application to verify persistence.

7. Summary of Testing Outcome

All core features performed as expected, with no critical bugs remaining after corrections.

Testing confirmed:

- Correct calculations
- Stable database operations
- Reliable UI interactions
- Proper validation and error handling

The application is functional, stable, and aligned with the project requirements.

Challenges Face

During the development of the Student Budget Tracker, several technical and design challenges were encountered. These challenges helped strengthen understanding of database security, GUI structuring, and application logic

Learnings & Key Takeaways

The development of the Student Budget Tracker provided extensive hands-on experience in GUI programming, database handling, and modular software design. Throughout the project, several valuable lessons were learned that contributed to both technical and conceptual growth.

Future Enhancements

Future enhancements for the Student Budget Tracker could include adding features such as editing and deleting expense records directly from the interface, generating graphical analytics like category-wise charts, and introducing monthly summary reports for better financial insights. The application could also be extended to support multiple user profiles, recurring expenses, data export options (CSV/PDF), and cloud-based synchronization for accessing budgets across devices. These improvements would make the system more versatile, user-friendly, and suitable for long-term personal finance management