Contents

Table of figures

1. Introduction

1.1 Overview

This project is a full-stack Finance Tracker with user authentication through a signup/login system built in Tkinter with MySQL as the backend database.

At first run the user can input its details while after that it only require password authentication to login the user.
Once logged in, users can add, view, edit, delete, and analyze their transactions categorized as income or expense.
The tracker also supports budgets, search filters, CSV export, receipt uploads, and monthly analysis graphs.
A dark/light theme toggle, a calendar view, and budget overspending alerts provide a modern, interactive user experience**.**

1.2 Objectives

1.2.1 User Authentication:

Implement a secure signup and login mechanism to protect user data and ensure personalized access.

1.2.2 Financial Transaction Management:

Allow users to add, edit, view, and delete income and expense records with category and receipt support.

1.2.3 Budget Monitoring and Alerts:

Enable users to set monthly budgets per category and receive alerts when spending exceeds the limit.

1.2.4 Data Visualization and Export:

Provide analysis tools, calendar views, and export options like CSV to help users better understand their financial habits.

1.3 Key Features

1.3.1 User Registration and Login:

Secure signup/login system with email validation and password strength checking.

1.3.2 Transaction Management:

Easily add, edit, delete, and search income and expense transactions with category tags.

1.3.3 Receipt Upload and Preview:

Attach receipt images to transactions and view them directly inside the app.

1.3.4 Budget Setting and Alerts:

Set monthly spending limits per category and get notified if you exceed your budget.

1.3.5 Data Visualization:

Analyze financial trends with graphs like pie charts, line charts, and category breakdowns.

1.3.6 Callendar View:

View daily transaction summaries using an interactive, clickable calendar.

1.3.7 Theme Toggle (Dark/Light Mode):

Switch between dark mode and light mode for a comfortable viewing experience.

1.3.8 Export to CSV:

Download filtered transaction reports for backup or external analysis.

2. Targeted Audience

2.1 Targeted

This application is designed for individuals, freelancers, and small business owners who want an easy way to manage their finances.
It is ideal for users looking for a simple, offline, and secure personal finance management tool with rich features.
Anyone who prefers visual insights, budget tracking, and receipt organization without relying on complex or paid software will benefit from this app.

2.2 How to use it?

2.2.1 Signup window

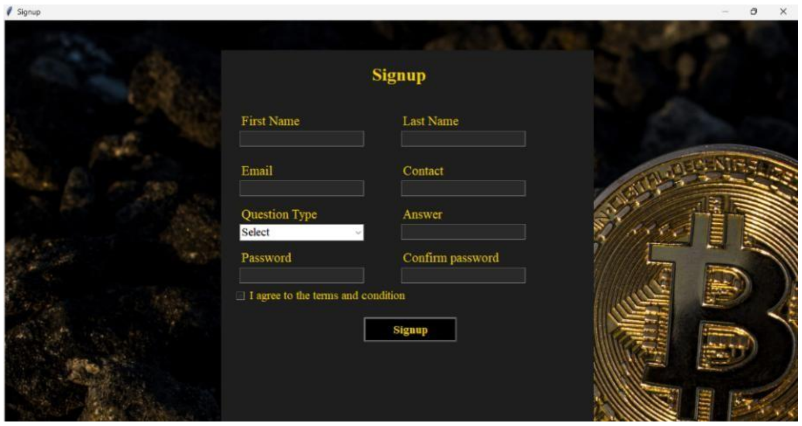


Figure 1: Signup

2.2.2 Login window

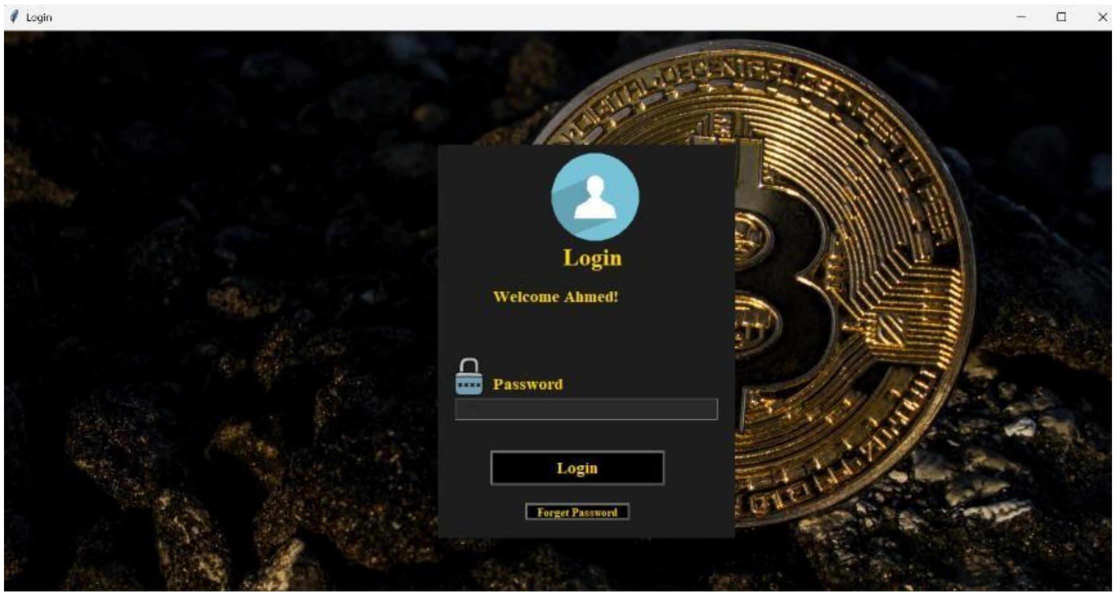
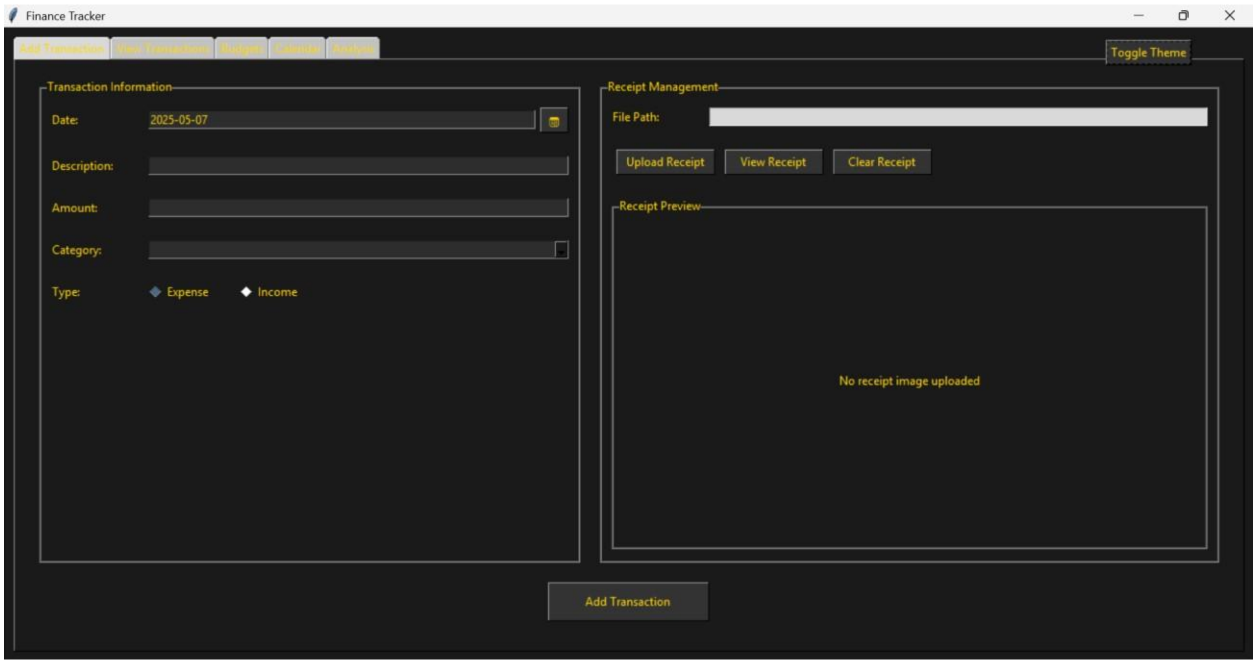


Figure 2: Login window

2.2.3 Transaction tab



Transaction Information

Date: 2025-05-07

Description:

Amount:

Category:

Type: Expense Income

Receipt Management

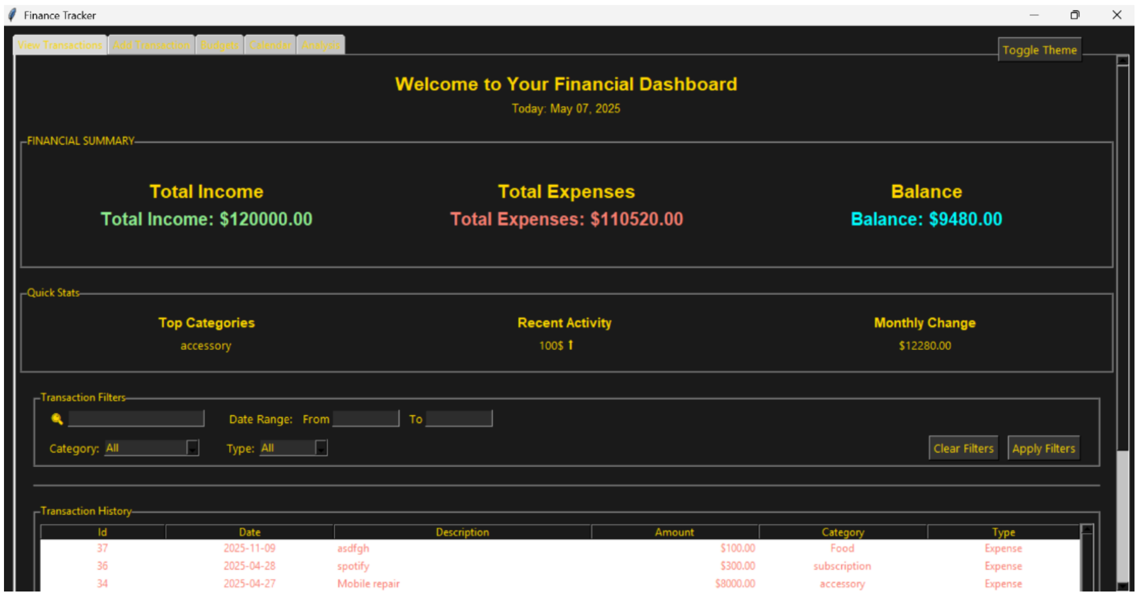
File Path:

Receipt Preview

No receipt image uploaded

Figure 3: Add Transaction

2.2.4 View Transaction tab



Welcome to Your Financial Dashboard

Today: May 07, 2025

FINANCIAL SUMMARY

Total Income	Total Expenses	Balance
Total Income: $120000.00	Total Expenses: $110520.00	Balance: $9480.00

Quick Stats

Top Categories	Recent Activity	Monthly Change
accessory	100$ ⬆	$12280.00

Transaction Filters

Date Range: From To

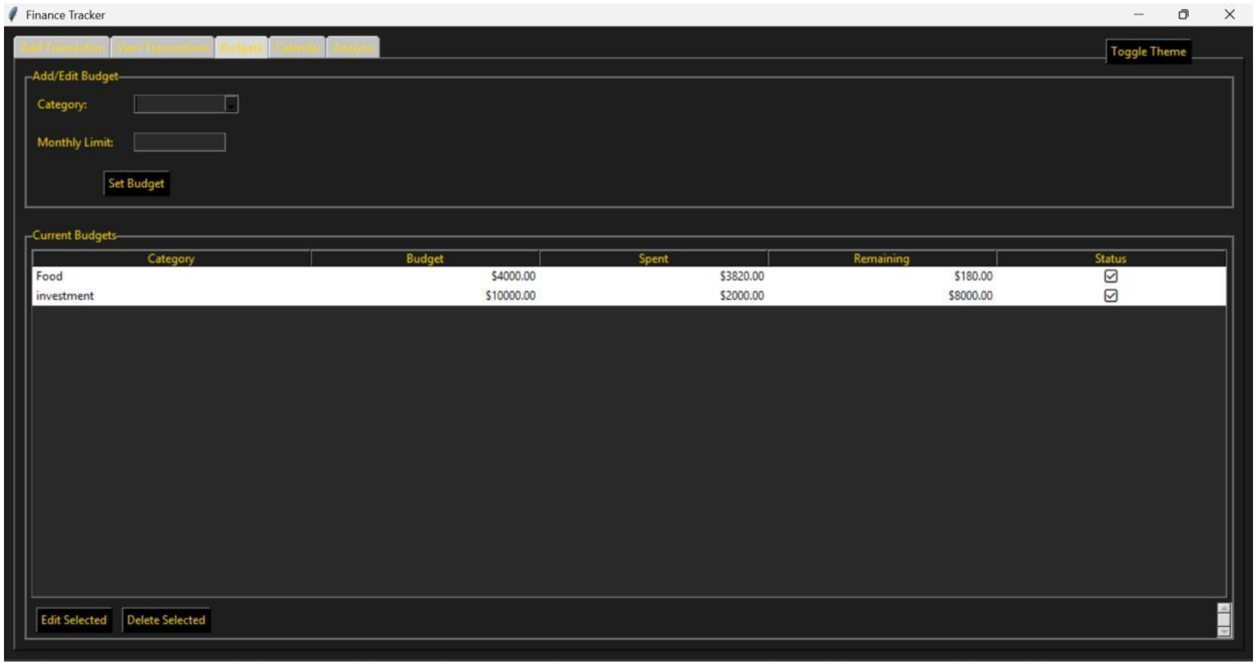
Category: All Type: All

Transaction History

Id	Date	Description	Amount	Category	Type
37	2025-11-09	asdfgh	$100.00	Food	Expense
36	2025-04-28	spotify	$300.00	subscription	Expense
34	2025-04-27	Mobile repair	$8000.00	accessory	Expense

Figure 4: View Transaction

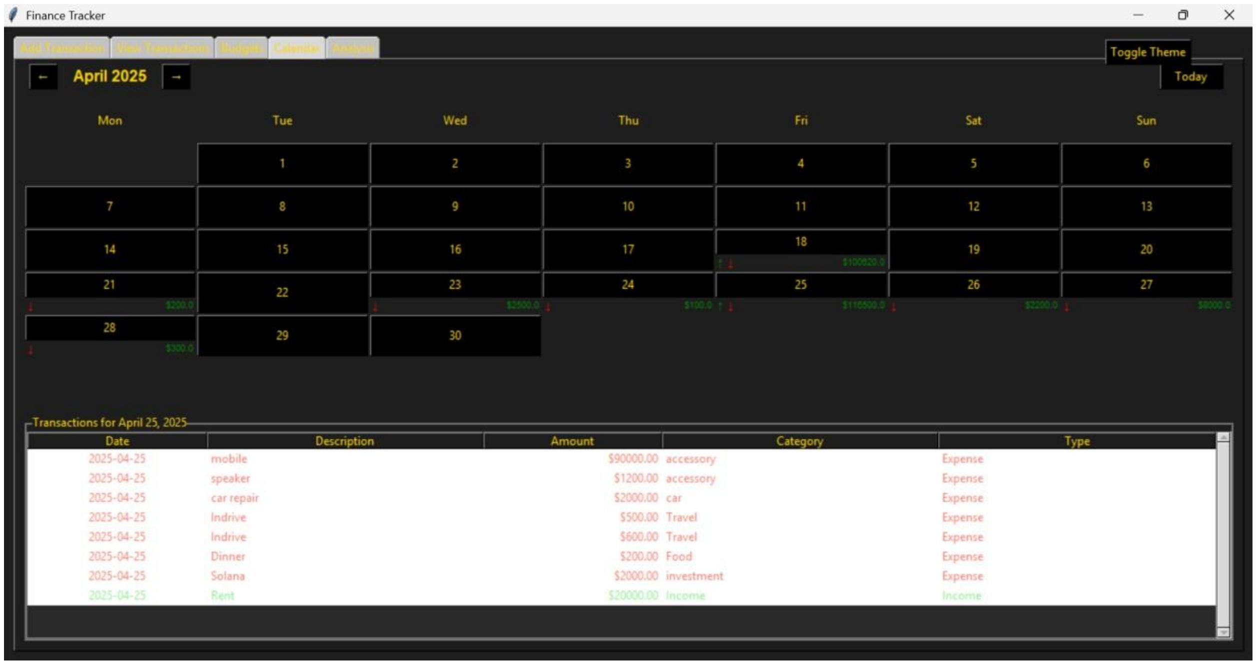
2.2.5 Budget tab



Category	Budget	Spent	Remaining	Status
Food	$4000.00	$3820.00	$180.00	☑
investment	$10000.00	$2000.00	$8000.00	☑

Figure 5: Budget Tab

2.2.6 Calendar view



Transactions for April 25, 2025

Date	Description	Amount	Category	Type
2025-04-25	mobile	$90000.00	accessory	Expense
2025-04-25	speaker	$1200.00	accessory	Expense
2025-04-25	car repair	$2000.00	car	Expense
2025-04-25	Indrive	$500.00	Travel	Expense
2025-04-25	Indrive	$600.00	Travel	Expense
2025-04-25	Dinner	$200.00	Food	Expense
2025-04-25	Solana	$2000.00	investment	Expense
2025-04-25	Rent	$20000.00	Income	Income

Figure 6: Calender Tab

2.2.7 Analysis tab

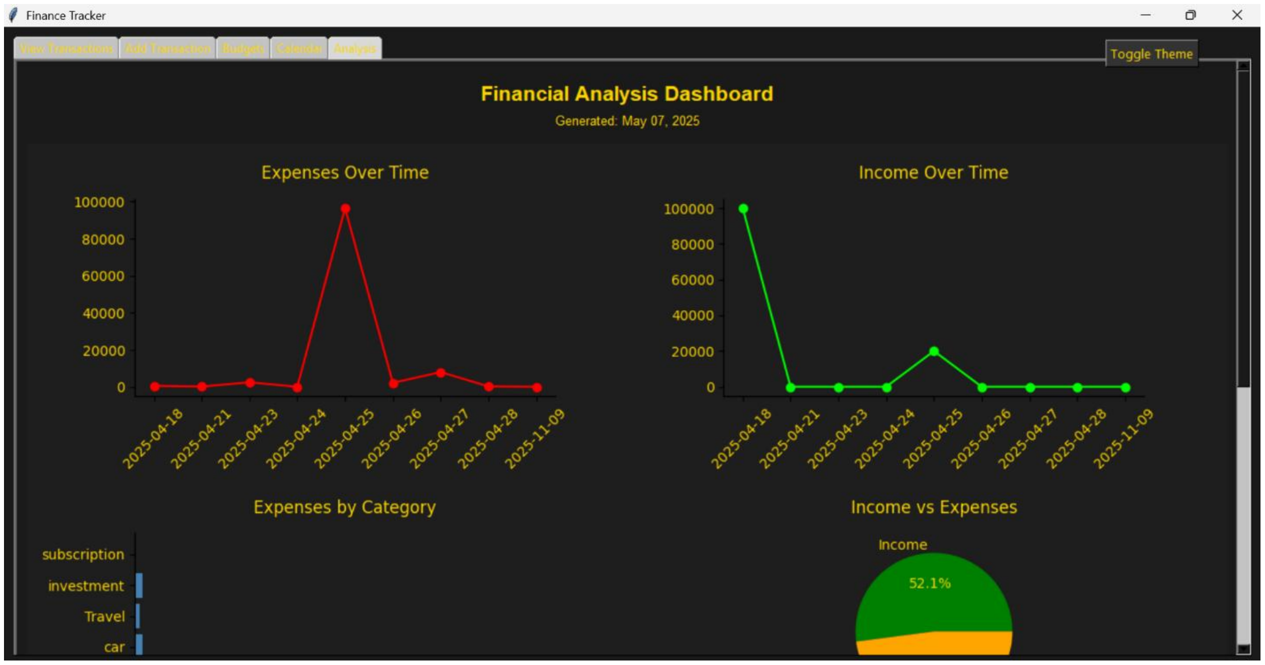


Figure 7: Analysis tab

2.3 Best case

This app is best used as a daily expense tracker to monitor personal or small business finances. It helps users stay within budget, analyze spending habits, and organize receipts securely in one place.
Perfect for managing monthly budgets, preparing financial reports, and maintaining better control over money flow.

3. Technology used

3.1 GUI framework

I have used tkinter in this project as a GUI framework. Tkinter is Python's standard built-in library for creating Graphical User Interface (GUI) applications.
It provides a simple and efficient way to build windows, buttons, forms, and other interactive elements.
Tkinter is lightweight, easy to learn, and perfect for building both small and large desktop apps.

3.2 Database

I have used MySQL as the database in the project. MySQL is a powerful, open-source relational database management system (RDBMS) used to store and manage data.
It is highly reliable, fast, and supports structured data storage with SQL queries for creating, reading, updating, and deleting records.
MySQL is widely used in web and software applications due to its stability, scalability, and ease of integration with Python and other languages.

3.3 Analysis

For analysis purposes I have used Matplotlib. Matplotlib is a popular data visualization library in Python used for creating static, animated, and interactive graphs.
It helps developers plot line charts, bar graphs, pie charts, and many other types of visual data representations.
Matplotlib is highly customizable, making it perfect for creating professional and publication quality plots.

4. DBMS

4.1 Database used

I have used MySQL as the database in the project. MySQL is a powerful, open-source relational database management system (RDBMS) used to store and manage data.
It is highly reliable, fast, and supports structured data storage with SQL queries for creating, reading, updating, and deleting records.
MySQL is widely used in web and software applications due to its stability, scalability, and ease of integration with Python and other languages.

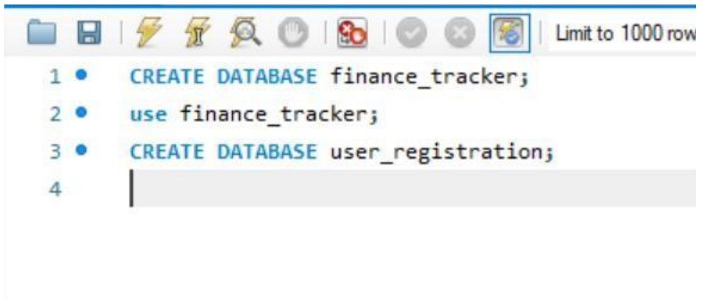
4.2 CRUD Operations

For CRUD operations I have used RAW SQL. It is because the queries written in the raw SQL executes faster than any ORM layer. The queries are simpler and easy to write. In my project I have used MySQL connector to write raw SQL queries. The MySQL connector provides flexibility and easy to write functions for the python program. I have written the CRUD functions in a separate file named DB.py to store all the CRUD operations written in RAW SQL.

5. Database Functioning

5.1 Creating database

The program connects to MySQL and checks if the required database exists, creating it if necessary.
This ensures all user and transaction data is organized under a dedicated database (user_registration or finance_tracker).

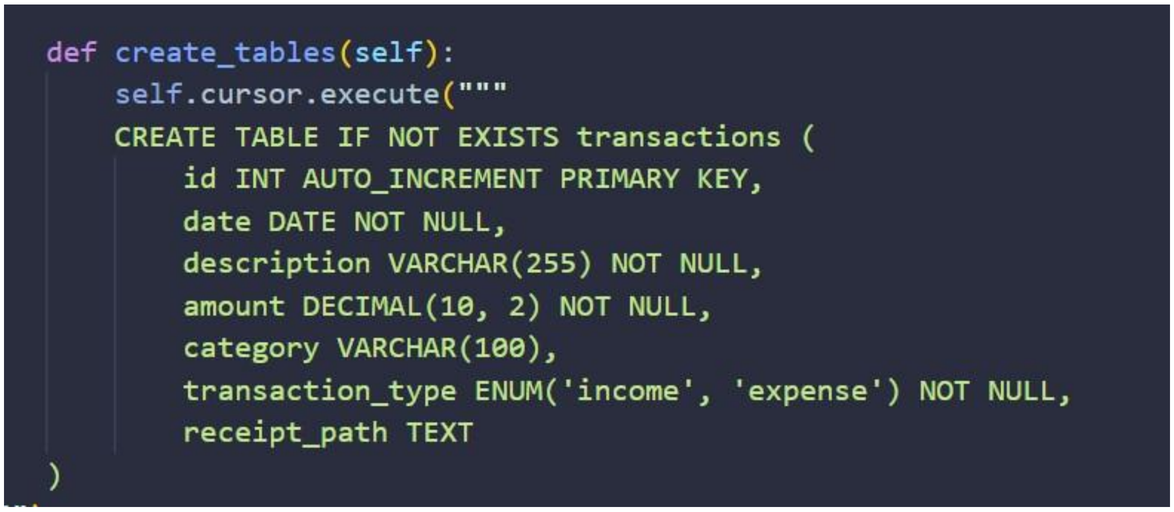


```
CREATE DATABASE finance_tracker;
use finance_tracker;
CREATE DATABASE user_registration;
```

Figure 8: Create database

5.2 Creating tables

SQL commands are executed to create essential tables like users, transactions, and budgets. These tables define the structure for storing different types of data, such as user details and financial records.



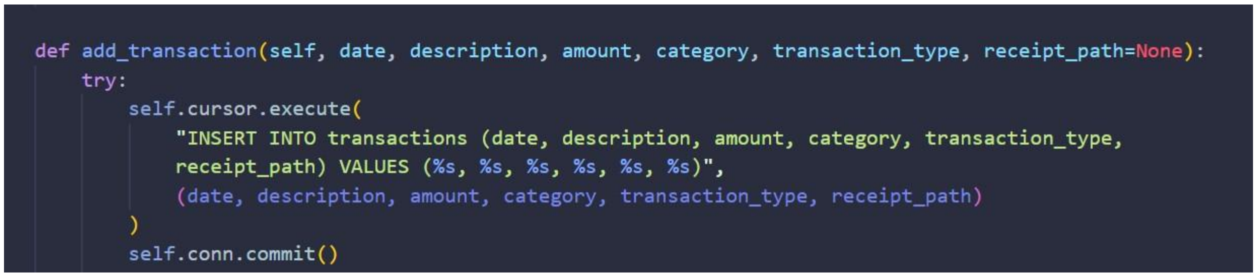
```
def create_tables(self):
    self.cursor.execute("""
    CREATE TABLE IF NOT EXISTS transactions (
        id INT AUTO_INCREMENT PRIMARY KEY,
        date DATE NOT NULL,
        description VARCHAR(255) NOT NULL,
        amount DECIMAL(10, 2) NOT NULL,
        category VARCHAR(100),
        transaction_type ENUM('income', 'expense') NOT NULL,
        receipt_path TEXT
)
```

Figure 9: Create table

5.3 Storing data

User inputs like signup details and financial transactions are inserted into the database using SQL INSERT queries.

Receipts and additional information are also stored securely to ensure a complete record of each entry.

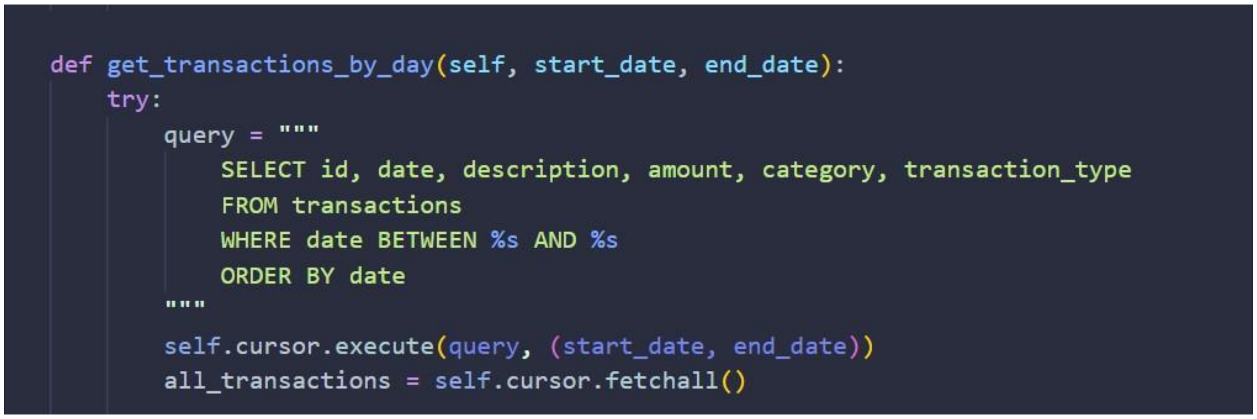


```
def add_transaction(self, date, description, amount, category, transaction_type, receipt_path=None):
    try:
        self.cursor.execute(
            "INSERT INTO transactions (date, description, amount, category, transaction_type,
            receipt_path) VALUES (%s, %s, %s, %s, %s, %s)",
            (date, description, amount, category, transaction_type, receipt_path)
        )
        self.conn.commit()
```

Figure 10: save data

5.4 Retrieving data

The application uses SELECT queries to fetch data for login verification, display transactions, budgets, and analysis.
Filters like date range, category, and type are applied to retrieve only the necessary information for users.



```
def get_transactions_by_day(self, start_date, end_date):
    try:
        query = """
            SELECT id, date, description, amount, category, transaction_type
            FROM transactions
            WHERE date BETWEEN %s AND %s
            ORDER BY date
        """
        self.cursor.execute(query, (start_date, end_date))
        all_transactions = self.cursor.fetchall()
```

Figure 11: Retreive data

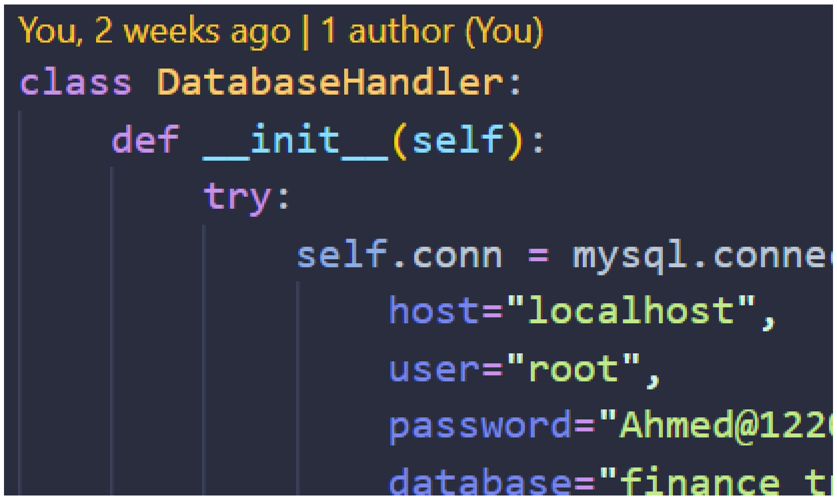
5.5 Data updating for analysis

Users can update transaction details, and budgets dynamically through SQL UPDATE queries. This updated data is immediately reflected in graphs, reports, and alerts for real-time financial analysis.

6. OOP

6.1. Classes

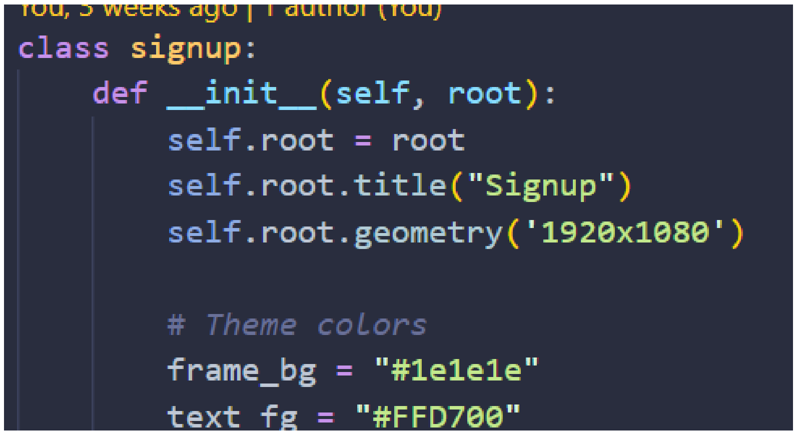
6.1.1. Database Handler



```
You, 2 weeks ago | 1 author (You)
class DatabaseHandler:
    def __init__(self):
        try:
            self.conn = mysql.conne
                host="localhost",
                user="root",
                password="Ahmed@122
                database="finance t
```

Figure 12: DB

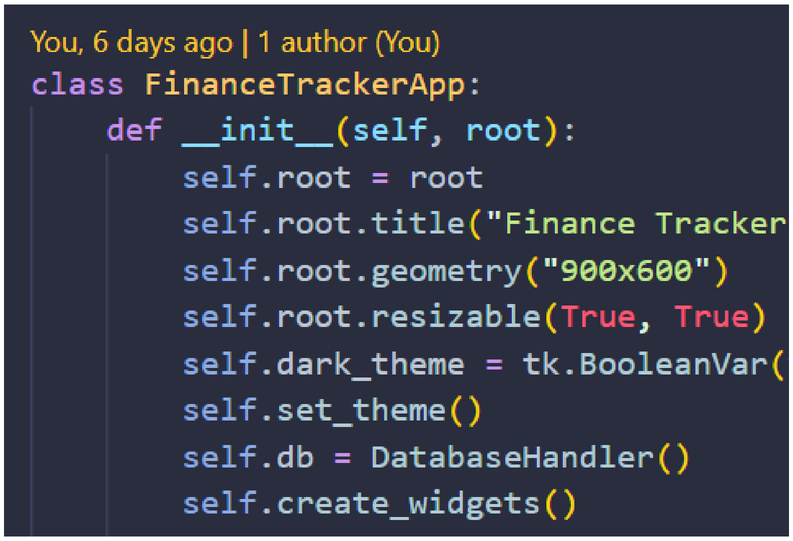
6.1.2. Signup



```
You, 3 weeks ago | 1 author (You)
class signup:
    def __init__(self, root):
        self.root = root
        self.root.title("Signup")
        self.root.geometry('1920x1080')

        # Theme colors
        frame_bg = "#1e1e1e"
        text_fg = "#FFD700"
```

Figure 13: Signup

6.1.3. Main App

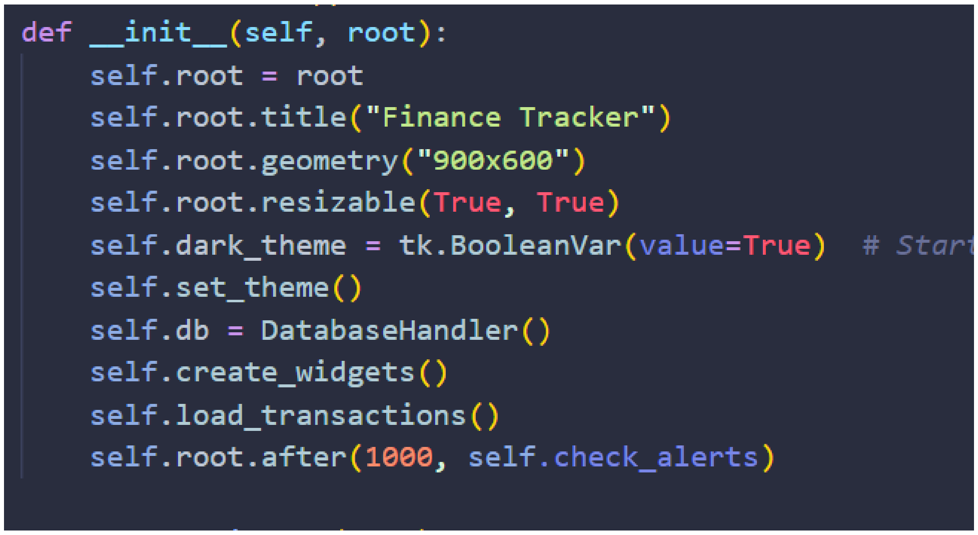


```
You, 6 days ago | 1 author (You)
class FinanceTrackerApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Finance Tracker
        self.root.geometry("900x600")
        self.root.resizable(True, True)
        self.dark_theme = tk.BooleanVar(
        self.set_theme()
        self.db = DatabaseHandler()
        self.create_widgets()
```

Figure 14: Mainapp

6.2. Constructors

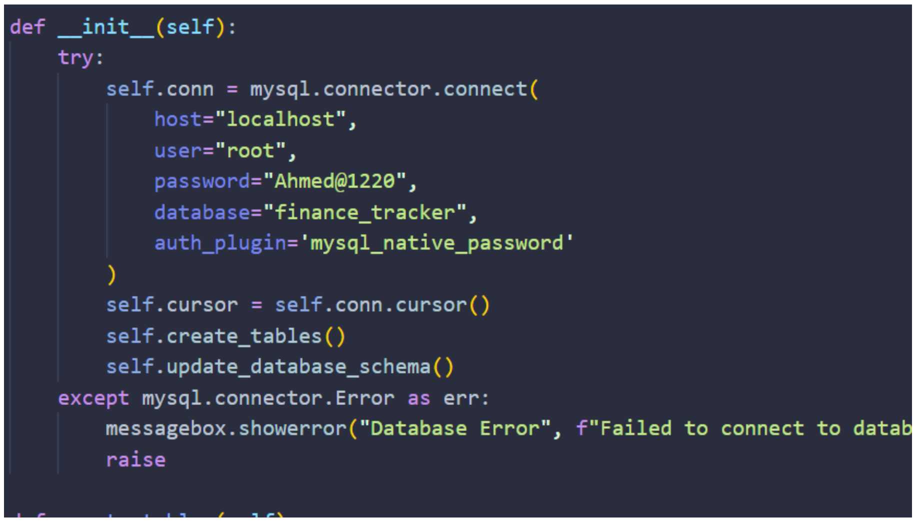
6.2.1. Main App



```
def __init__(self, root):
    self.root = root
    self.root.title("Finance Tracker")
    self.root.geometry("900x600")
    self.root.resizable(True, True)
    self.dark_theme = tk.BooleanVar(value=True)  # Start
    self.set_theme()
    self.db = DatabaseHandler()
    self.create_widgets()
    self.load_transactions()
    self.root.after(1000, self.check_alerts)
```

Figure 15: C1: Mainapp

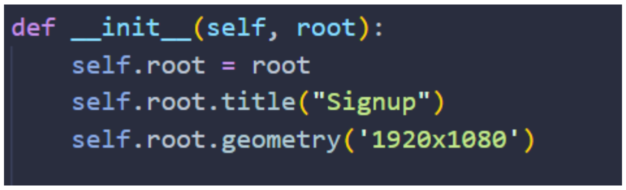
6.2.2. Login/Signup



```
def __init__(self):
    try:
        self.conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="Ahmed@1220",
            database="finance_tracker",
            auth_plugin='mysql_native_password'
        )
        self.cursor = self.conn.cursor()
        self.create_tables()
        self.update_database_schema()
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Failed to connect to datab
        raise
```

Figure 16: Login - Signup

6.2.3. Database

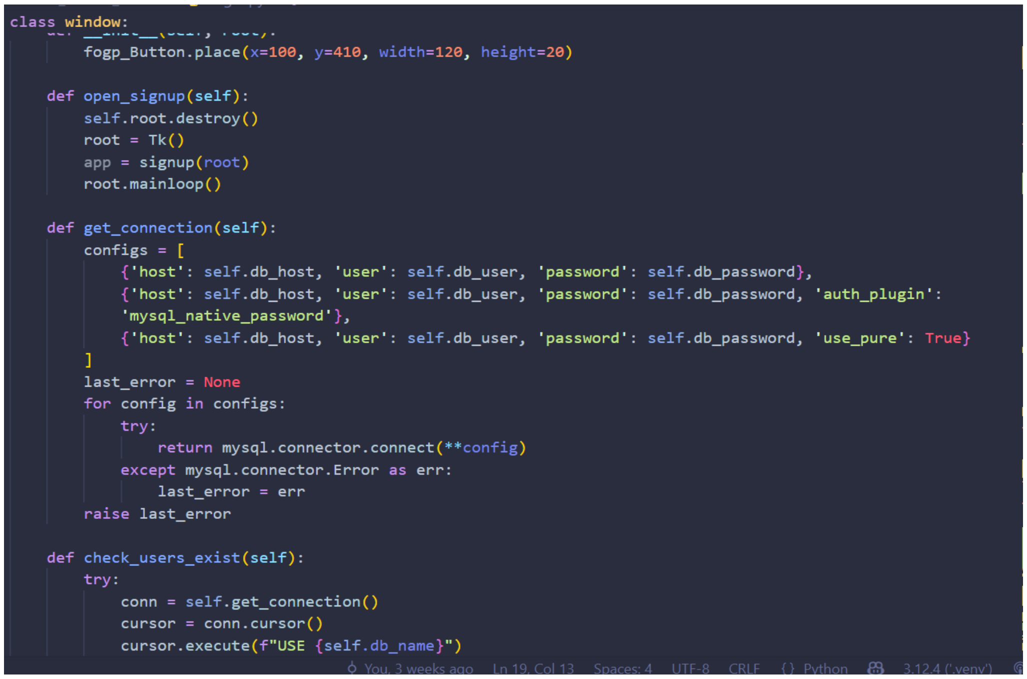


```
def __init__(self, root):
    self.root = root
    self.root.title("Signup")
    self.root.geometry('1920x1080')
```

Figure 17: Database

6.3 Methods

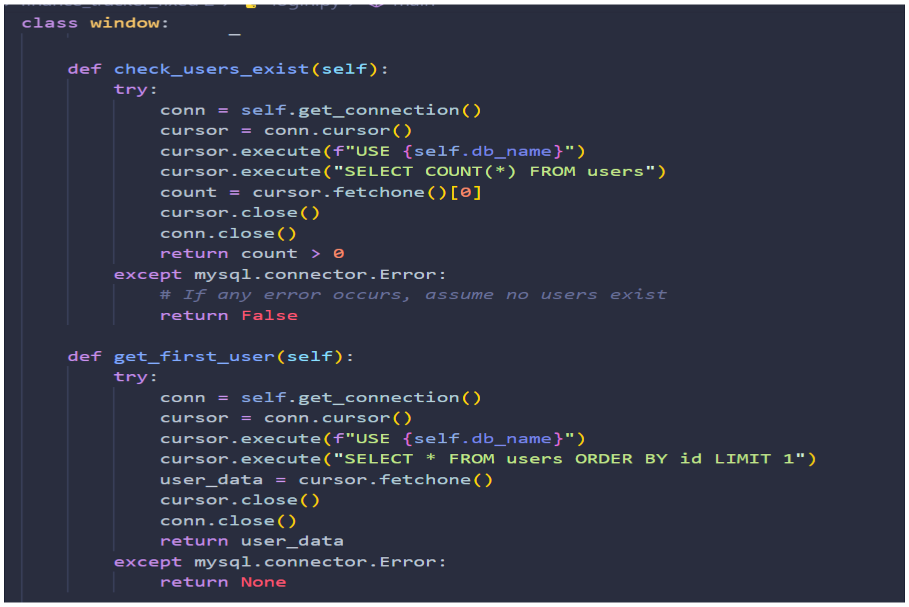
6.3.1. Main Application



```
class window:
        fogp_Button.place(x=100, y=410, width=120, height=20)

    def open_signup(self):
        self.root.destroy()
        root = Tk()
        app = signup(root)
        root.mainloop()

    def get_connection(self):
        configs = [
            {'host': self.db_host, 'user': self.db_user, 'password': self.db_password},
            {'host': self.db_host, 'user': self.db_user, 'password': self.db_password, 'auth_plugin':
            'mysql_native_password'},
            {'host': self.db_host, 'user': self.db_user, 'password': self.db_password, 'use_pure': True}
        ]
        last_error = None
        for config in configs:
            try:
                return mysql.connector.connect(**config)
            except mysql.connector.Error as err:
                last_error = err
        raise last_error

    def check_users_exist(self):
        try:
            conn = self.get_connection()
            cursor = conn.cursor()
            cursor.execute(f"USE {self.db_name}")
```

Figure 18: Main app

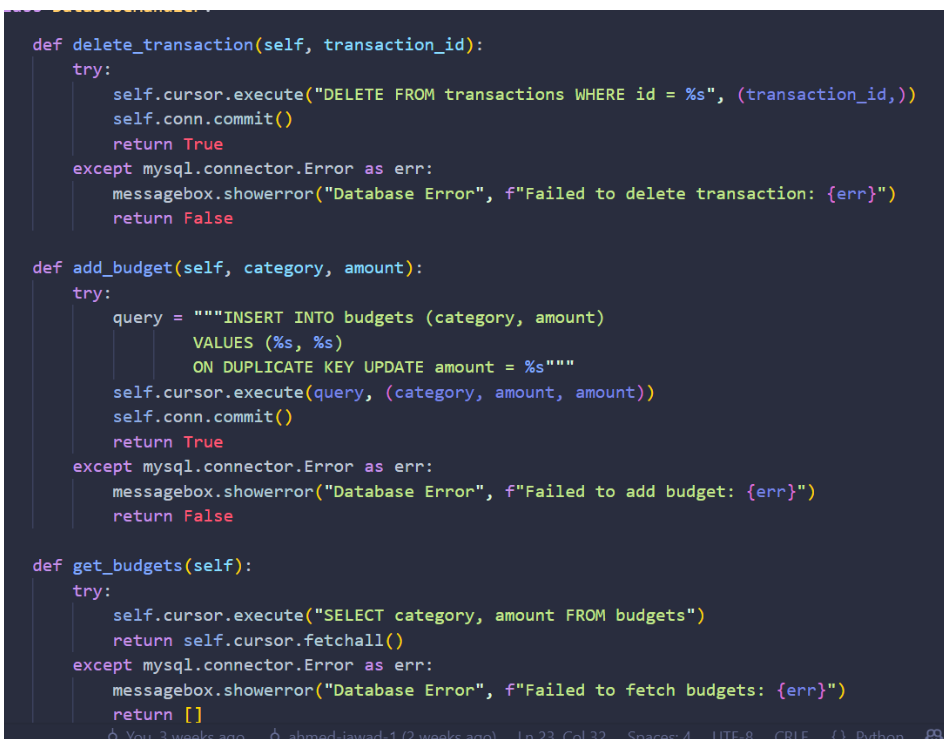
6.3.2. Login/Signup



```
class window:

    def check_users_exist(self):
        try:
            conn = self.get_connection()
            cursor = conn.cursor()
            cursor.execute(f"USE {self.db_name}")
            cursor.execute("SELECT COUNT(*) FROM users")
            count = cursor.fetchone()[0]
            cursor.close()
            conn.close()
            return count > 0
        except mysql.connector.Error:
            # If any error occurs, assume no users exist
            return False

    def get_first_user(self):
        try:
            conn = self.get_connection()
            cursor = conn.cursor()
            cursor.execute(f"USE {self.db_name}")
            cursor.execute("SELECT * FROM users ORDER BY id LIMIT 1")
            user_data = cursor.fetchone()
            cursor.close()
            conn.close()
            return user_data
        except mysql.connector.Error:
            return None
```

Figure 19: Login - signup

6.3.3. Database

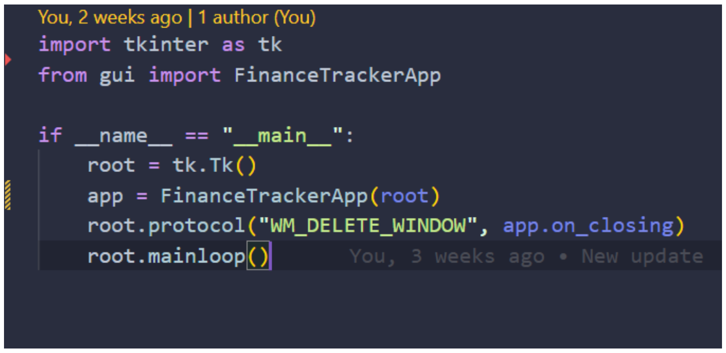


```
def delete_transaction(self, transaction_id):
    try:
        self.cursor.execute("DELETE FROM transactions WHERE id = %s", (transaction_id,))
        self.conn.commit()
        return True
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Failed to delete transaction: {err}")
        return False

def add_budget(self, category, amount):
    try:
        query = """INSERT INTO budgets (category, amount)
                VALUES (%s, %s)
                ON DUPLICATE KEY UPDATE amount = %s"""
        self.cursor.execute(query, (category, amount, amount))
        self.conn.commit()
        return True
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Failed to add budget: {err}")
        return False

def get_budgets(self):
    try:
        self.cursor.execute("SELECT category, amount FROM budgets")
        return self.cursor.fetchall()
    except mysql.connector.Error as err:
        messagebox.showerror("Database Error", f"Failed to fetch budgets: {err}")
        return []
```

Figure 20: Data Base

6.4. Abstraction

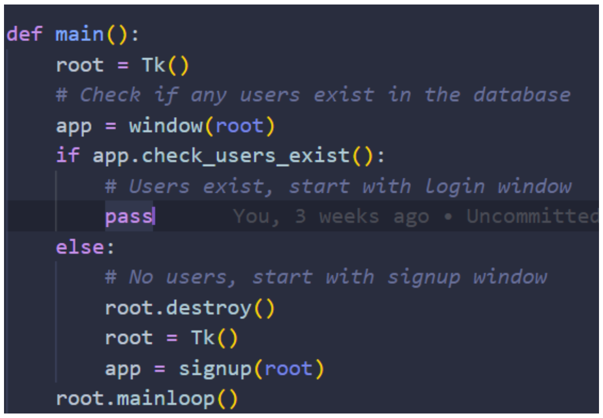
I have created a separate file to run the main function the file serves as the function calling that calls the functions of the main app. The root and the main window is main looped here.



You, 2 weeks ago | 1 author (You)

```
import tkinter as tk
from gui import FinanceTrackerApp

if __name__ == "__main__":
    root = tk.Tk()
    app = FinanceTrackerApp(root)
    root.protocol("WM_DELETE_WINDOW", app.on_closing)
    root.mainloop()      You, 3 weeks ago • New update
```

Figure 21: Abstraction 1



```python
def main():
    root = Tk()
    # Check if any users exist in the database
    app = window(root)
    if app.check_users_exist():
        # Users exist, start with login window
        pass        You, 3 weeks ago • Uncommitted
    else:
        # No users, start with signup window
        root.destroy()
        root = Tk()
        app = signup(root)
    root.mainloop()
```

Figure 22: Abstraction 2

7. Run time analysis

7.1 Library used

For analysis purposes I have used Matplotlib.

7.2 Why Matplotlib

Matplotlib is a popular data visualization library in Python used for creating static, animated, and interactive graphs.
It helps developers plot line charts, bar graphs, pie charts, and many other types of visual data representations.
Matplotlib is highly customizable, making it perfect for creating professional and publication quality plots.

7.3 Graph description

7.3.1 Expenses Over Time:

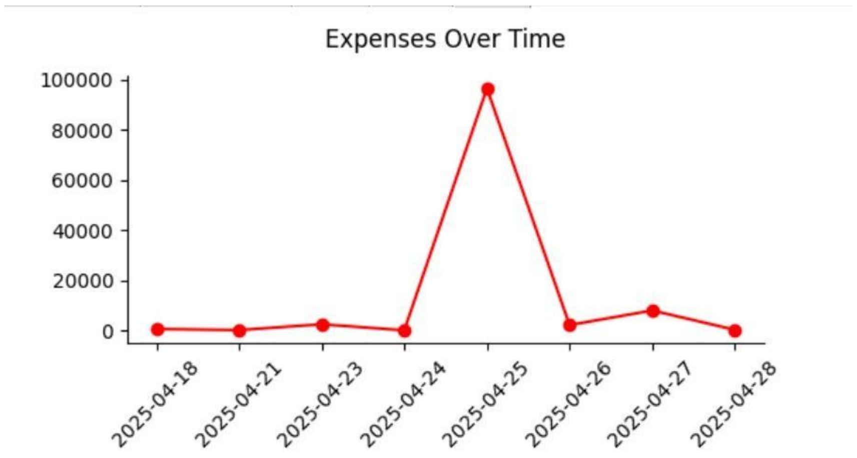
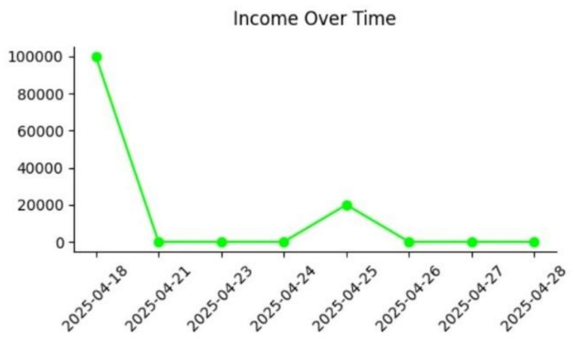


Figure 23: Expenses over time

7.3.2 Income Over Time:

Figure 24: Income over time



7.3.3 Expenses by Category:

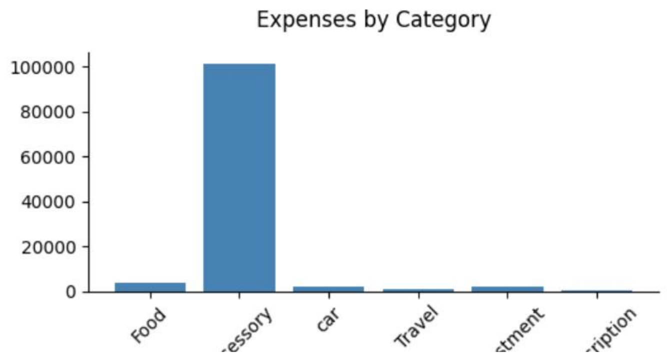


Figure 25: Expenses over category

7.3.4 Income vs Expenses Pie Chart:

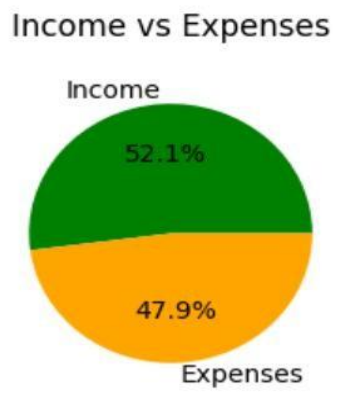


Figure 26: Income vs Expenses

8. Dataflow diagram

8.1 Overview

This diagram shows the flow of a financial management app where users can sign up, log in, and access the main app. From the main app, users can add transactions, view transactions, set budgets, view analysis, and access a calendar view. All transaction, budget, and analysis data are stored and retrieved from a central database.

The app also incorporates an alert system that notifies users when they approach or exceed budget limits based on their transaction history. Users can customize notification preferences and set personalized financial goals with progress tracking features. The system includes machine learning capabilities that provide spending pattern insights and future expense predictions. Additionally, the app offers bill payment reminders and recurring transaction automation to help users maintain financial discipline. For security, the platform implements multi-factor authentication and end-to-end encryption to protect sensitive financial data.

8.2 DFD

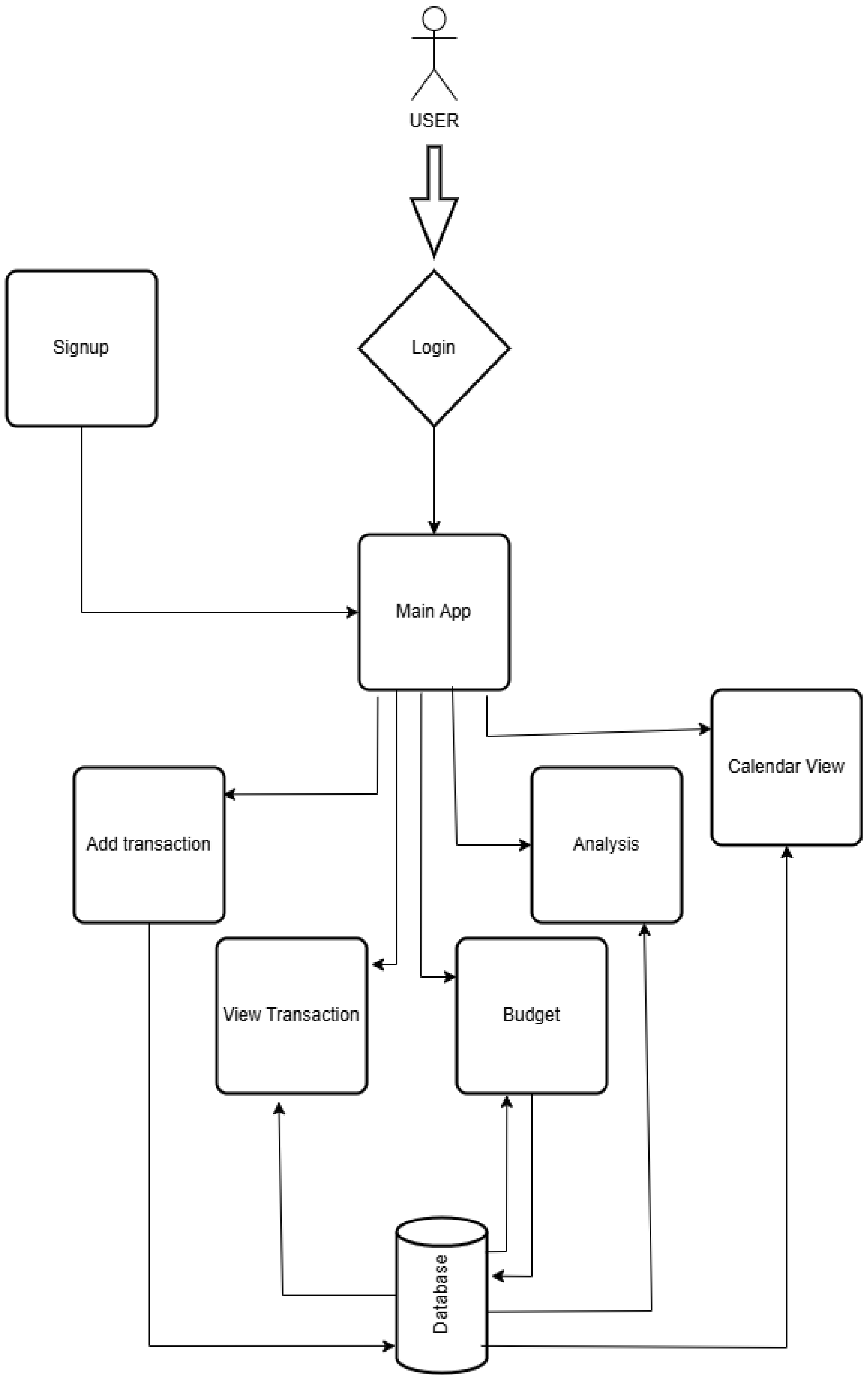


Figure 27: DFD

9. Entity relationship Diagram

9.1 Overview

This ERD represents a system with three entities: User, Transaction, and Budget. Users have personal details and are linked to multiple transactions, while transactions are categorized and can relate to budgets. Budgets define spending limits for specific categories and are connected back to transactions through the category field.

9.2 ERD

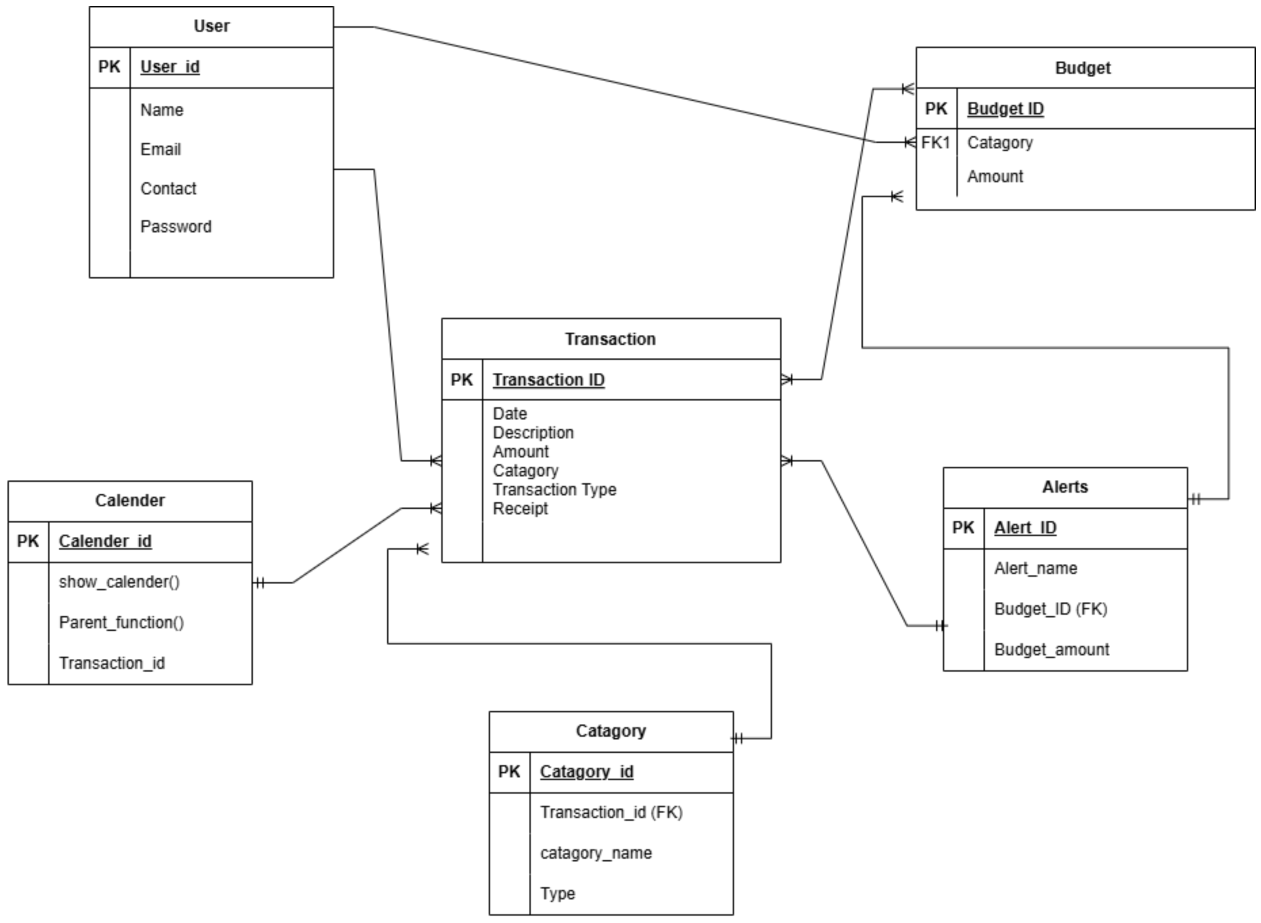


Figure 28: ERD

10 Relationships

10.1 Entities

In my project, the main entities are **User**, **Transaction**, and **Budget**, each represented by a table in the database.

10.2 Relationships

The User entity is related to Transaction — meaning a user (after login) can add/view/edit multiple transactions; similarly, Budgets relate to transaction categories.

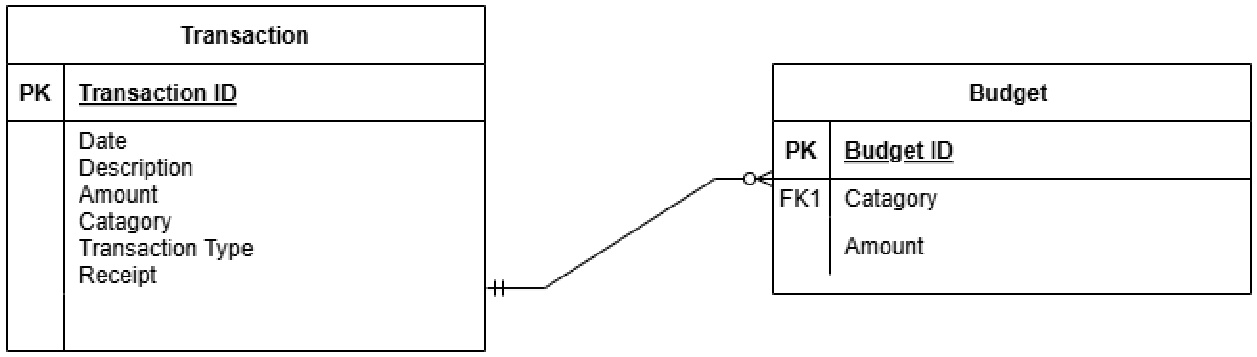


Figure 29: Relation 1

This ERD shows a one-to-many relationship where each transaction belongs to a category, but the model is incorrectly structured — the Category table should not have a foreign key to Transaction. Instead, Transaction should reference Categorified as a foreign key.

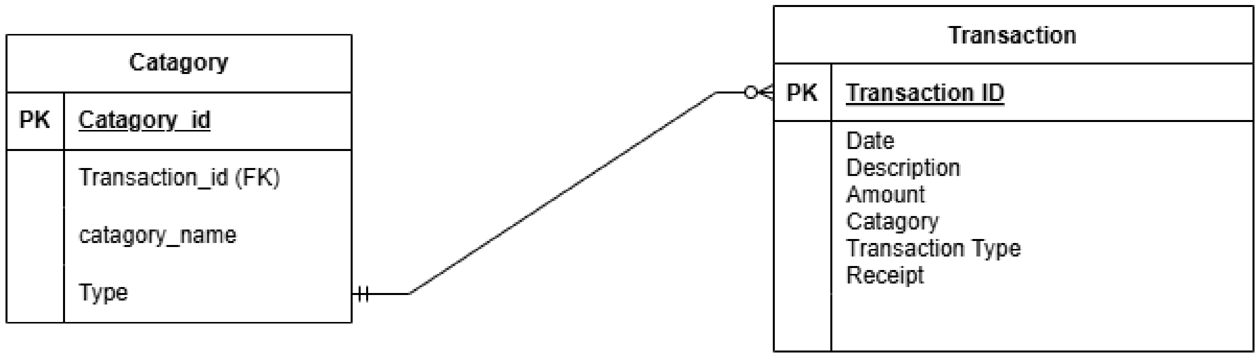


Figure 30: Relation 2

This ERD shows a one-to-many relationship where each calendar entry is linked to a transaction. However, functions like show_calender() and Parent_function() should not be attributes in an ERD, as ERDs only model data, not behavior.

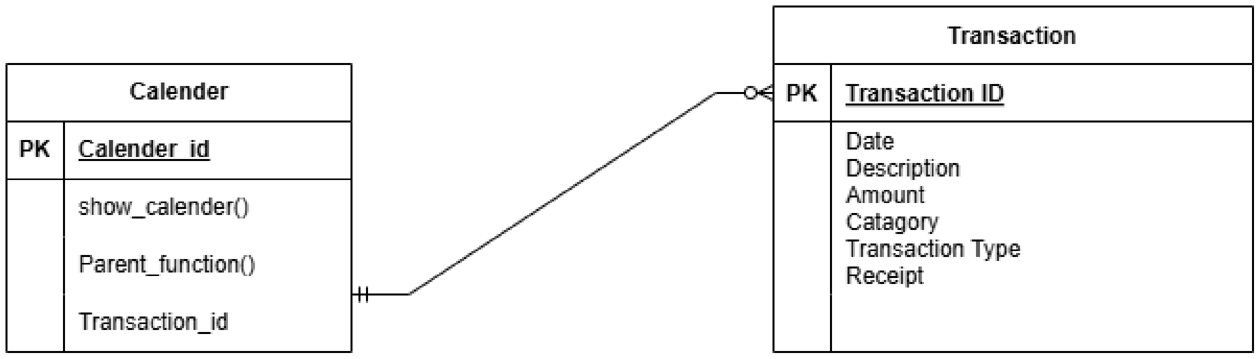


Figure 31: Relation 3

This ERD shows a relationship between "Alerts" and "Transaction" entities. The Alerts table has a primary key of Alert_ID and contains fields for Alert_name, Budget_ID (a foreign key), and Budget_amount. It has a many-to-one relationship with the Transaction table, which has Transaction_ID as its primary key and includes fields for Date, Description, Amount, Category, Transaction Type, and Receipt.

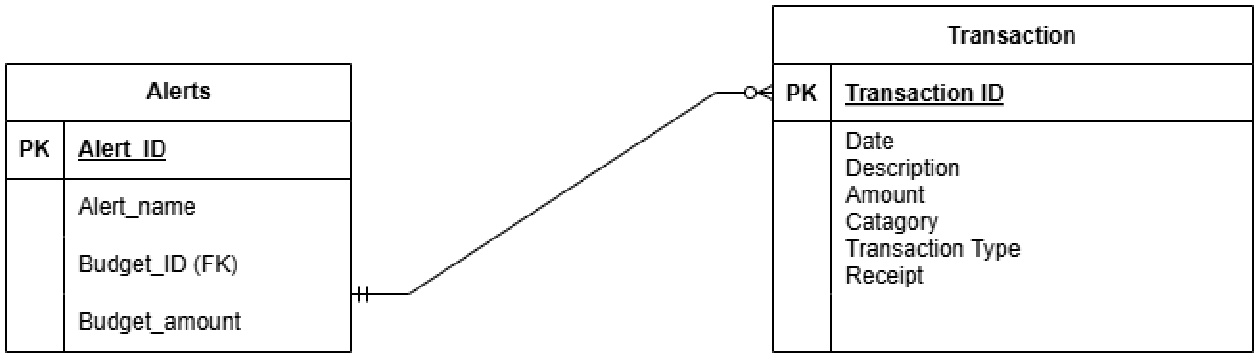


Figure 32: Relation 4

This ERD illustrates a relationship between "Alerts" and "Budget" entities. The Alerts table contains Alert_ID (primary key), Alert_name, Budget_ID (foreign key), and Budget_amount fields. It has a many-to-one relationship with the Budget table, which has Budget_ID as its primary key and includes Category (FK1) and Amount fields. This structure allows multiple alerts to be associated with a single budget.

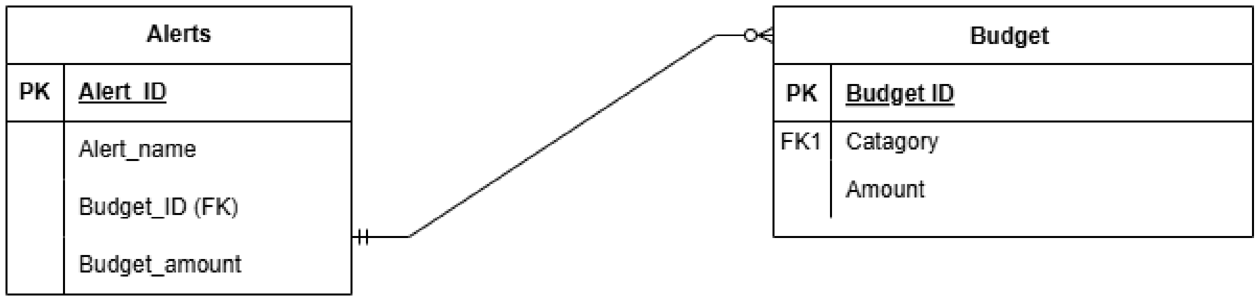


Figure 33: Relation 5

10.3 Strong entities

User, Transaction, and Budget are all strong entities because they each have their own primary keys (id) and can exist independently.

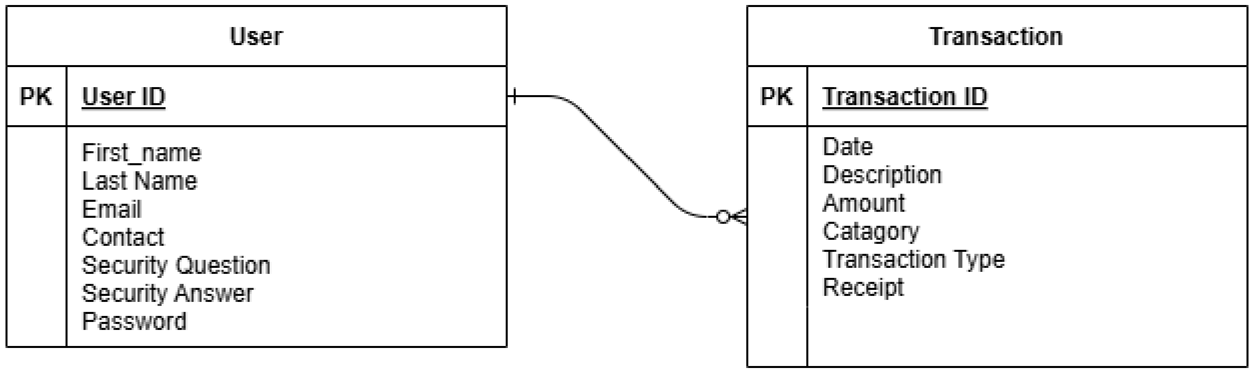


Figure 34: relation 6

10.4 Weak entities

Your project does not use weak entities because all records are uniquely identifiable on their own without needing a composite key or dependency.

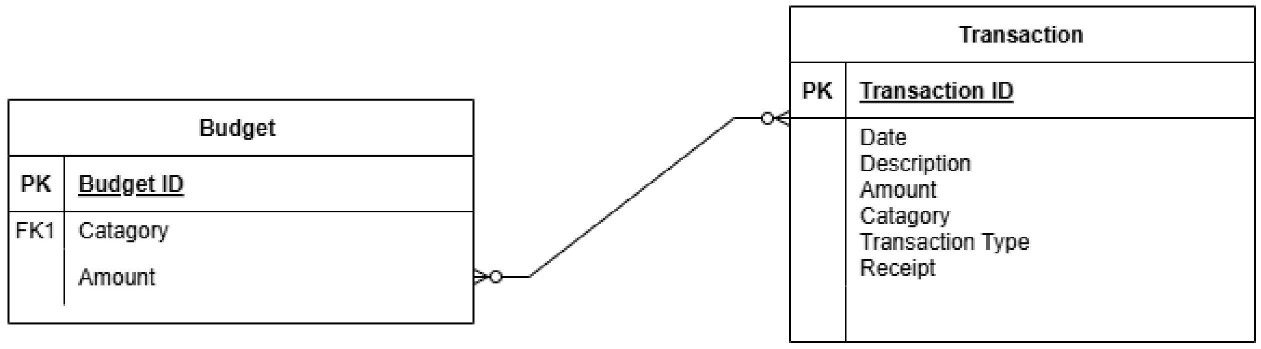


Figure 35: Relation 7

11. Limitations

11.1 Scaling

The project is designed for individual or small business use; handling many users or massive transaction data would require upgrading to a more scalable backend (e.g., using connection pooling, ORM, or cloud databases).

11.2 Deployment

Since it is built with Tkinter (a desktop-only GUI) and local MySQL, it is not web-based and needs manual installation on each machine where it's used.

11.3 Framework out of date

Tkinter, while stable, is limited in modern UI features compared to newer frameworks (like PyQT, Flutter, or web apps), making it harder to deliver a sleek, mobile-friendly or highly interactive experience.

12. Conclusion

In conclusion, this project successfully demonstrates how personal finance management can be made simple and user-friendly. The application provides users with a secure way to register, log in, and manage their financial transactions efficiently. By offering features like budgeting, calendar views, receipt uploads, and data analysis graphs, it covers essential financial needs. The use of Tkinter ensures a clean and responsive interface, while MySQL provides a reliable backend for storing critical data. Real-time alerts and detailed analysis help users make better financial decisions and control overspending. Although the application is designed for smallscale usage, it lays a strong foundation for future scalability. It also shows good coding practices by separating database operations, GUI design, and application logic into different modules. Challenges like database handling and file management were overcome with effective solutions and thorough testing. There is still room for improvements, such as deploying the app to the web or adding more intelligent reporting features. Overall, this project achieves its goal of delivering a complete, offline finance tracking system with modern user experience.

13. GitHub link

You can find the complete source code for this project on GitHub.
Visit the repository here: https://github.com/ahmed-jawad-5/finance_tracker.git