EXPENSE TRACKER

A MINI PROJECT REPORT

Submitted by

NALIN KARTHIK K 220701178

NAVEEN KUMAR 220701183

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023 – 24

BONAFIDE CERTIFICATE

Certified that this project report "EXPENSE TRACKER" is the

bonafide work of "NALIN KARTHIK K 220701178

NAVEEN KUMAR 220701183 "

who carried out the project work under my supervision.

Submitted for the Practical Examination held on_____

SIGNATURE   SIGNATURE

Dr.R.SABITHA   KALPANA D

Professor and II Year Academic Head   Assistant Professor (SG),

Computer Science and Engineering,   Computer Science and Engineering,

Rajalakshmi Engineering   Rajalakshmi Engineering
College,(Autonomous),   College, (Autonomous),

Thandalam, Chennai - 602 105   Thandalam, Chennai - 602 105

INTERNAL EXAMINER   EXTERNAL EXAMINER

ABSTRACT

This project presents the design and implementation of a comprehensive expense tracker application using Python, SQLite, and Streamlit for the user interface. The primary objective of this application is to help users efficiently manage their personal finances by tracking expenses, setting budgets, and monitoring account balances.

The application leverages SQLite for robust and efficient data storage, ensuring data persistence across sessions. Streamlit is utilized to create an interactive and user-friendly interface, allowing users to easily navigate and utilize the application's features.

This expense tracker provides a comprehensive solution for personal financial management, enabling users to gain better control over their finances, make informed spending decisions, and achieve their financial goals.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 INTRODUCTION

Managing personal finances effectively is a critical skill in today's fast-paced and economically dynamic world. The need for a streamlined and user-friendly tool to track expenses, set budgets, and monitor account balances has become increasingly important. This project introduces a robust expense tracker application designed to address these needs by leveraging the power of Python, SQLite, and Streamlit for a seamless user experience. The application allows users to securely register and log in to their accounts, manage their expenses across various categories, set and monitor budgets, and keep track of their available balance in real-time. By providing comprehensive financial insights and analytics through visual reports, this expense tracker empowers users to make informed financial decisions, ensuring better control over their personal finances and promoting financial stability.

1.2 OBJECTIVE

The primary objective of this expense tracker application is to provide users with a comprehensive tool for managing their personal finances. Specifically, the application aims to:

- Facilitate Secure User Management: Allow users to securely register and log in to their accounts, ensuring data privacy and personalized financial tracking.
- Enable Efficient Expense Tracking: Provide an intuitive interface for users to record and manage their daily expenses, categorized for better organization.
- Support Budget Management: Allow users to set budgets for different categories and monitor their spending against these budgets, with alerts for overspending.
- Track Account Balances: Maintain a real-time account balance that deducts expenses to help users manage their available funds effectively.
- Generate Financial Insights: Offer visual reports to help users analyse their spending patterns and make informed financial decisions.

1.3 MODULES

- User Authentication Module
- Expense Management Module
- Category Management Module
- Budget Management Module
- Account Balance Management Module
- Expense Analytics Module

2. SURVEY OF TECHNOLOGY

2.1 SOFTWARE DESCRIPTION

The expense tracker application is a comprehensive tool designed to help users manage their personal finances effectively. Developed using Python, SQLite, and Streamlit, the application provides a user-friendly interface for tracking expenses, setting budgets, monitoring account balances, and generating financial reports. Below is a detailed description of the software's components and functionalities.

2.2 LANGUAGES

2.2.1. PYTHON

Python is a high-level, interpreted programming language known for its simplicity and readability. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is widely used in web development, data analysis, artificial intelligence, and scientific computing due to its extensive libraries and frameworks.

2.2.2. SQL (Structured Query Language)

SQL is a domain-specific language used in programming and designed for managing and manipulating relational databases. It is used to perform various operations on the data stored in databases, such as querying, updating, and managing schemas.

3. REQUIREMENT AND ANALYSIS

3.1 REQUIREMENTS SPECIFICATION

USER REQUIREMENTS

1. User Registration: Users should be able to create an account with a unique username and password.

2. User Login: Users should be able to log in securely to access their financial data.

3. Add Expense: Users should be able to log their expenses by specifying the date, category, description, and amount.

4. View Expenses: Users should be able to view a list of their recorded expenses.

5. Delete Expense: Users should be able to delete any incorrect or outdated expense entries.

6. Set Budgets: Users should be able to set budget limits for different categories.

7. Monitor Budgets: Users should receive warnings when their expenses exceed the set budgets.

8. Expense Reports: Users should be able to generate visual reports, such as pie charts, to analyse their spending patterns.

SYSTEM REQUIREMENTS

- Operating System: The application should be compatible with Windows, macOS, and Linux.

- Python Environment: Python 3.7 or higher should be installed on the system.

- Libraries**: The following Python libraries should be installed:
1. - `streamlit`
2. - `sqlite3`
3. - `matplotlib`
4. - `pandas`

- Database**: SQLite should be used for data storage, requiring no separate database server.

- Web Browser**: A modern web browser (e.g., Chrome, Firefox, Edge) for accessing the Streamlit web interface.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

Software Requirements

To develop and run the expense tracker application, the following software components are required:

1. Python 3.7+

2. Python Libraries

   - Streamlit   - Installation: `pip install streamlit`

   - SQLite3   - Included in the Python standard library.

   - Matplotlib   - Installation: `pip install matplotlib`

   - Pandas   - Installation: `pip install pandas`

3. SQLite Database

4. Web Browser

5. Operating System Compatibility

   - The application should be compatible with major operating systems, including Windows, macOS, and Linux.

6. Integrated Development Environment (IDE)

Hardware Requirements

To develop and run the expense tracker application effectively, the following hardware components are recommended:

1. Processor

   - A modern multi-core processor (e.g., Intel Core i3 or equivalent) to handle the execution of Python code and web interface smoothly.

2.Memory (RAM)

- At least 4 GB of RAM to ensure that the application runs efficiently, especially when handling multiple expenses and generating visual reports.

3. Storage

   - At least 100 MB of free disk space for installing Python, necessary libraries, and storing the SQLite database file and application code.

4. Display

   - A screen resolution of 1024x768 or higher to properly view and interact with the Streamlit web interface.

5. Peripherals

   - Standard input devices such as a keyboard and mouse for interacting with the application.

3.3 DATA FLOW DIAGRAM

DFD is an important tool used by system analysis. A data flow diagram model, a system using external entities from which data flows through a process which transforms the data and creates output data transforms which go to other processes external entities such as files. The main merit of DFD is that it can provide an overview of what data a system would process.

• A data-flow diagram is a way of representing a flow of data through a process or a system.

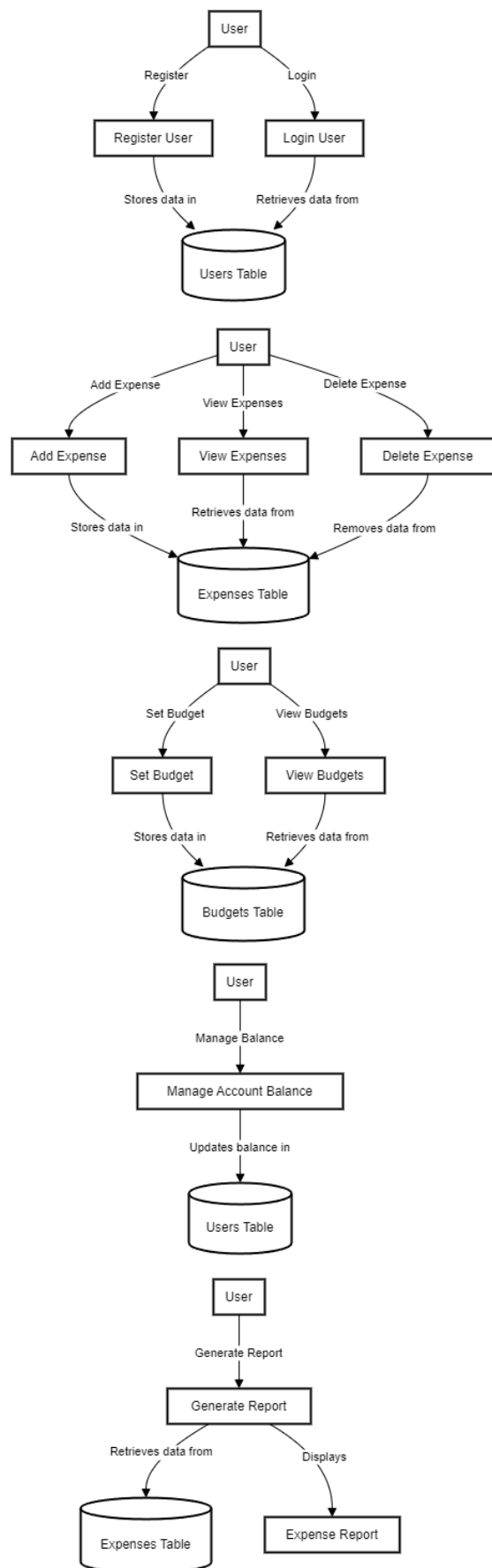• The DFD also provides information about the outputs and inputs of each entity and the process itself

LEVEL 0 DFD DIAGRAM
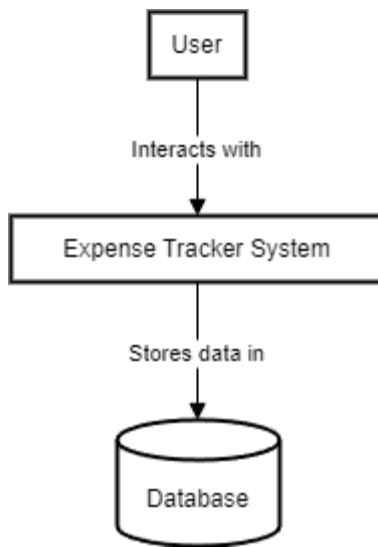
LEVEL 1 DFD DIAGRAM
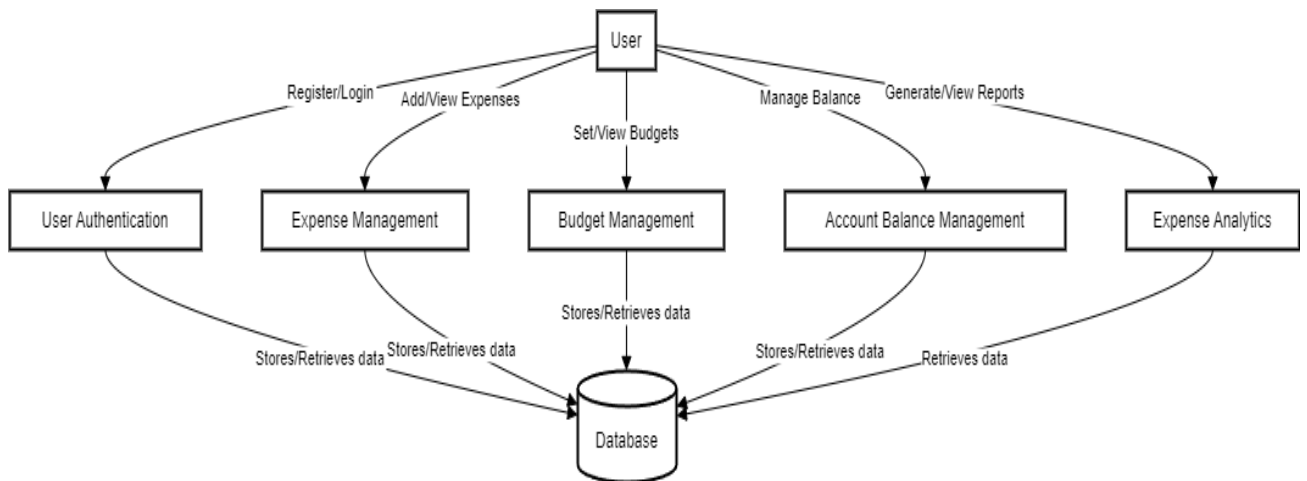
LEVEL 2 DFD DIAGRAM

3.4 DATA DICTIONARY

USERS TABLE

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	0	None	1
1	username	TEXT	1	None	0
2	password	TEXT	1	None	0

SQLITE SEQUENCE TABLE

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	name		0	None	0
1	seq		0	None	0

EXPENSE TABLE

Column ID	Name	Type	Not Null	Default Value	Primary Key
0	id	INTEGER	0	None	1
1	user_id	INTEGER	1	None	0
2	date	TEXT	1	None	0
3	category	TEXT	1	None	0
4	description	TEXT	0	None	0
5	amount	REAL	1	None	0

3.5 E-R DIAGRAM
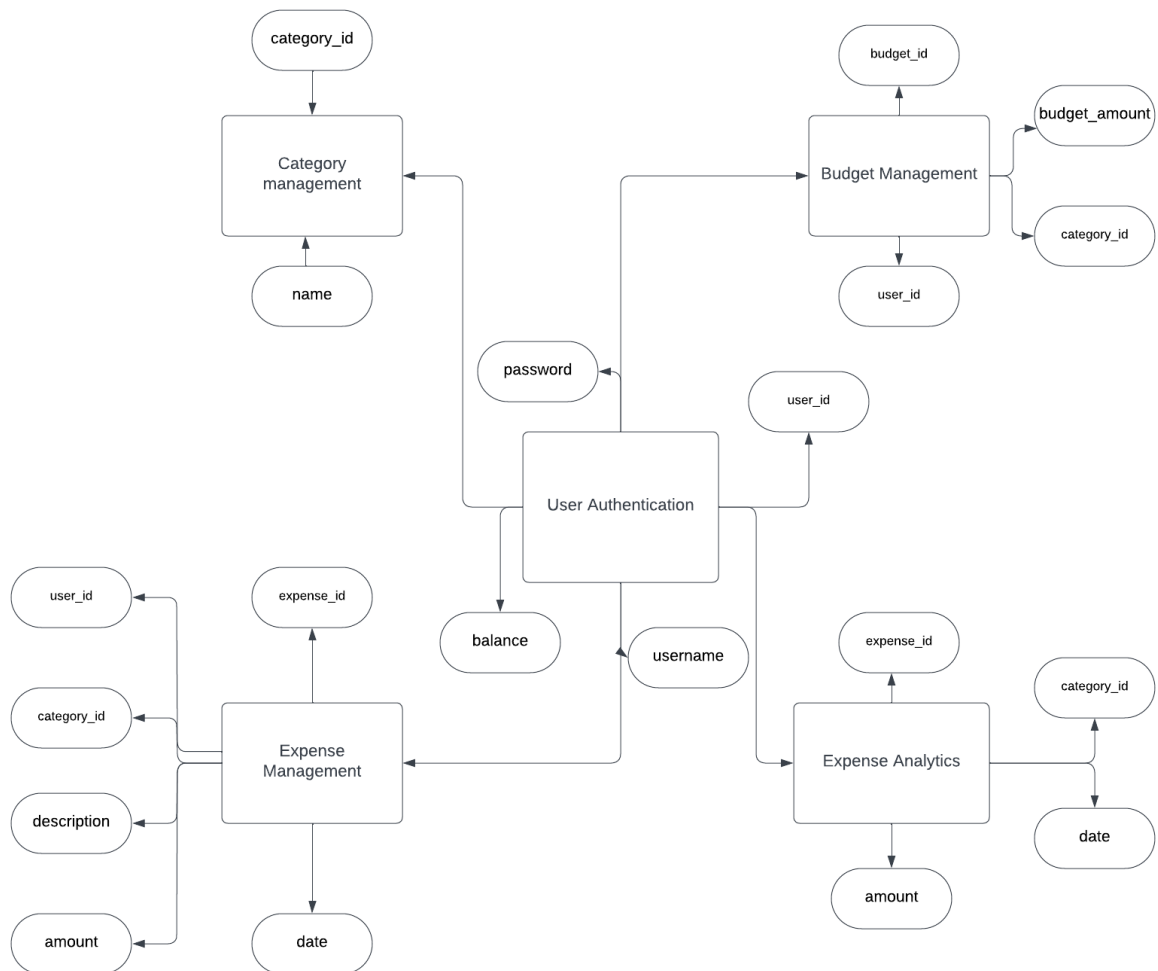
3.6 NORMALIZATION

Normalization is a database design technique that reduces data redundancy and ensures data integrity. It involves organizing tables and their relationships according to a set of rules to achieve a higher normal form. We'll normalize the tables in your expense tracker application up to the Fifth Normal Form (5NF).

First Normal Form (1NF)

Each table should have atomic (indivisible) values and each record should be unique.

- Already satisfied in the initial design.

Second Normal Form (2NF)

Remove partial dependencies; non-key attributes should depend on the whole primary key.

- Each table in the initial design is already in 2NF as each non-key attribute depends on the entire primary key.

Third Normal Form (3NF)

Remove transitive dependencies; non-key attributes should not depend on other non-key attributes.

- Each table in the initial design is already in 3NF as there are no transitive dependencies.

Boyce-Codd Normal Form (BCNF)

A stricter version of 3NF; for every functional dependency (X -> Y), X should be a super key.

- Each table in the initial design satisfies BCNF as all functional dependencies have super keys on the left side.

Fourth Normal Form (4NF)

Remove multi-valued dependencies; a record should not contain two or more independent multi-valued facts about an entity.

- There are no multi-valued dependencies in the initial design, so the tables are already in 4NF.

Fifth Normal Form (5NF)

Remove join dependencies; a table should be broken into smaller tables without loss of information.

- Let's check if there are any join dependencies that need addressing.

Since the initial tables are already in BCNF and 4NF, and there are no complex join dependencies that require further decomposition, the tables are already in 5NF. However, we can ensure this by carefully analyzing each table:

1. **Users**:

   - `id` (Primary Key)

   - `username`

   - `password`

   - `balance`

   This table is already in 5NF.

2. **Categories**:

   - `id` (Primary Key)

   - `name`

   This table is already in 5NF.

3. **Expenses**:

   - `id` (Primary Key)

   - `user_id` (Foreign Key to Users)

   - `category_id` (Foreign Key to Categories)

   - `description`

   - `amount`

   - `date`

   This table is already in 5NF.

4. **Budgets**:

   - `id` (Primary Key)

   - `user_id` (Foreign Key to Users)

   - `category_id` (Foreign Key to Categories)

   - `budget_amount`

   This table is already in 5NF.

5. **AccountBalances**:

   - `user_id` (Primary Key, Foreign Key to Users)

   - `balance`

   This table is already in 5NF

4. PROGRAM CODE

4.1. CODE DETAILS AND CODE EFFICIENCY

```python
import streamlit as st

import sqlite3

from datetime import datetime

import matplotlib.pyplot as plt

import pandas as pd


# Initialize the database

def initialize_db():

    conn = sqlite3.connect('expenses.db')

    c = conn.cursor()

    c.execute('''

    CREATE TABLE IF NOT EXISTS users (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        username TEXT NOT NULL UNIQUE,

        password TEXT NOT NULL

    )

    ''')

    c.execute('''

    CREATE TABLE IF NOT EXISTS categories (

        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```python
        name TEXT NOT NULL UNIQUE
    )
    ''')
    c.execute('''
    CREATE TABLE IF NOT EXISTS budgets (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        user_id INTEGER NOT NULL,
        category TEXT NOT NULL,
        amount REAL NOT NULL,
        FOREIGN KEY (user_id) REFERENCES users (id)
    )
    ''')
    conn.commit()
    conn.close()


# List of predefined categories
categories = ["Food", "Transportation", "Entertainment", "Utilities", "Rent",
"Miscellaneous"]


# Initialize database
initialize_db()


# Streamlit UI
st.title("Expense Tracker")
```

```python
# Session state for logged-in user

if "logged_in_user" not in st.session_state:

    st.session_state.logged_in_user = None


def register_user(username, password):

    if username and password:

        conn = sqlite3.connect('expenses.db')

        c = conn.cursor()

        try:

            c.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username,
password))

            conn.commit()

            st.success("User registered successfully")

        except sqlite3.IntegrityError:

            st.error("Username already exists")

        conn.close()

    else:

        st.error("Please fill all the fields")


def login_user(username, password):

    if username and password:

        conn = sqlite3.connect('expenses.db')

        c = conn.cursor()

        c.execute("SELECT * FROM users WHERE username=? AND password=?",
(username, password))
```

```python
        user = c.fetchone()

        conn.close()


        if user:

            st.session_state.logged_in_user = user[0]

            st.success("Logged in successfully")

            # Create a new table for the user if it does not exist

            create_user_expenses_table(user[0])

        else:

            st.error("Invalid username or password")

    else:

        st.error("Please fill all the fields")


def create_user_expenses_table(user_id):

    conn = sqlite3.connect('expenses.db')

    c = conn.cursor()

    table_name = f"expenses_{user_id}"

    c.execute(f'''

    CREATE TABLE IF NOT EXISTS {table_name} (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        date TEXT NOT NULL,

        category TEXT NOT NULL,

        description TEXT,

        amount REAL NOT NULL
```

```python
    )
    ''')
    conn.commit()
    conn.close()


def add_expense(user_id, category, description, amount):
    date = datetime.now().strftime('%Y-%m-%d')
    table_name = f"expenses_{user_id}"
    if category and amount:
        conn = sqlite3.connect('expenses.db')
        c = conn.cursor()
        c.execute(f"INSERT INTO {table_name} (date, category, description, amount) VALUES (?, ?, ?, ?)",
                  (date, category, description, amount))
        conn.commit()

        # Check if the budget for the category is exceeded
        c.execute(f"SELECT SUM(amount) FROM {table_name} WHERE category=?", (category,))
        total_spent = c.fetchone()[0]
        c.execute("SELECT amount FROM budgets WHERE user_id=? AND category=?", (user_id, category))
        budget = c.fetchone()
        if budget and total_spent > budget[0]:
            st.warning(f"Warning: You have exceeded your budget for {category}!")
```

```python
        conn.close()

        st.success("Expense added successfully")

    else:

        st.error("Please fill all the required fields")


def view_expenses(user_id):

    table_name = f"expenses_{user_id}"

    conn = sqlite3.connect('expenses.db')

    c = conn.cursor()

    c.execute(f"SELECT * FROM {table_name}")

    rows = c.fetchall()

    conn.close()

    return rows


def delete_expense(user_id, expense_id):

    table_name = f"expenses_{user_id}"

    conn = sqlite3.connect('expenses.db')

    c = conn.cursor()

    c.execute(f"DELETE FROM {table_name} WHERE id=?", (expense_id,))

    conn.commit()

    conn.close()

    st.success("Expense deleted successfully")
```

```python
def refresh_categories():

    conn = sqlite3.connect('expenses.db')

    c = conn.cursor()

    c.execute("SELECT * FROM categories")

    categories = c.fetchall()

    conn.close()

    return categories


def set_budget(user_id, category, amount):

    if category and amount:

        conn = sqlite3.connect('expenses.db')

        c = conn.cursor()

        c.execute("INSERT INTO budgets (user_id, category, amount) VALUES (?, ?, ?)",

                (user_id, category, amount))

        conn.commit()

        conn.close()

        st.success("Budget set successfully")

    else:

        st.error("Please fill all the required fields")


def generate_expense_report(user_id):

    table_name = f"expenses_{user_id}"

    conn = sqlite3.connect('expenses.db')
```

```python
    c = conn.cursor()

    c.execute(f"SELECT category, SUM(amount) FROM {table_name} GROUP BY
category")

    data = c.fetchall()

    conn.close()


    if data:

        df = pd.DataFrame(data, columns=['Category', 'Amount'])

        fig, ax = plt.subplots()

        ax.pie(df['Amount'], labels=df['Category'], autopct='%1.1f%%', startangle=140)

        ax.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

        plt.title('Expense Report')

        st.pyplot(fig)

    else:

        st.info("No expenses to report")


# User Authentication

if st.session_state.logged_in_user is None:

    st.subheader("Login")

    username = st.text_input("Username")

    password = st.text_input("Password", type="password")

    if st.button("Login"):

        login_user(username, password)


    st.subheader("Register")
```

```python
    new_username = st.text_input("New Username")

    new_password = st.text_input("New Password", type="password")

    if st.button("Register"):

        register_user(new_username, new_password)


else:

    # Main Application

    st.subheader("Add Expense")

    selected_category = st.selectbox("Category", categories)

    description = st.text_input("Description")

    amount = st.number_input("Amount", min_value=0.01)

    if st.button("Add Expense"):

        add_expense(st.session_state.logged_in_user, selected_category, description, amount)


    st.subheader("View Expenses")

    expenses = view_expenses(st.session_state.logged_in_user)

    if expenses:

        df = pd.DataFrame(expenses, columns=['ID', 'Date', 'Category', 'Description', 'Amount'])

        st.table(df)

        selected_expense_id = st.selectbox("Select Expense to Delete", df['ID'])

        if st.button("Delete Expense"):

            delete_expense(st.session_state.logged_in_user, selected_expense_id)

    else:

        st.info("No expenses found")
```

```python
st.subheader("Set Budget")

budget_category = st.selectbox("Budget Category", categories)

budget_amount = st.number_input("Budget Amount", min_value=0.01)

if st.button("Set Budget"):

    set_budget(st.session_state.logged_in_user, budget_category, budget_amount)


st.subheader("Generate Expense Report")

if st.button("Generate Report"):

    generate_expense_report(st.session_state.logged_in_user)
```
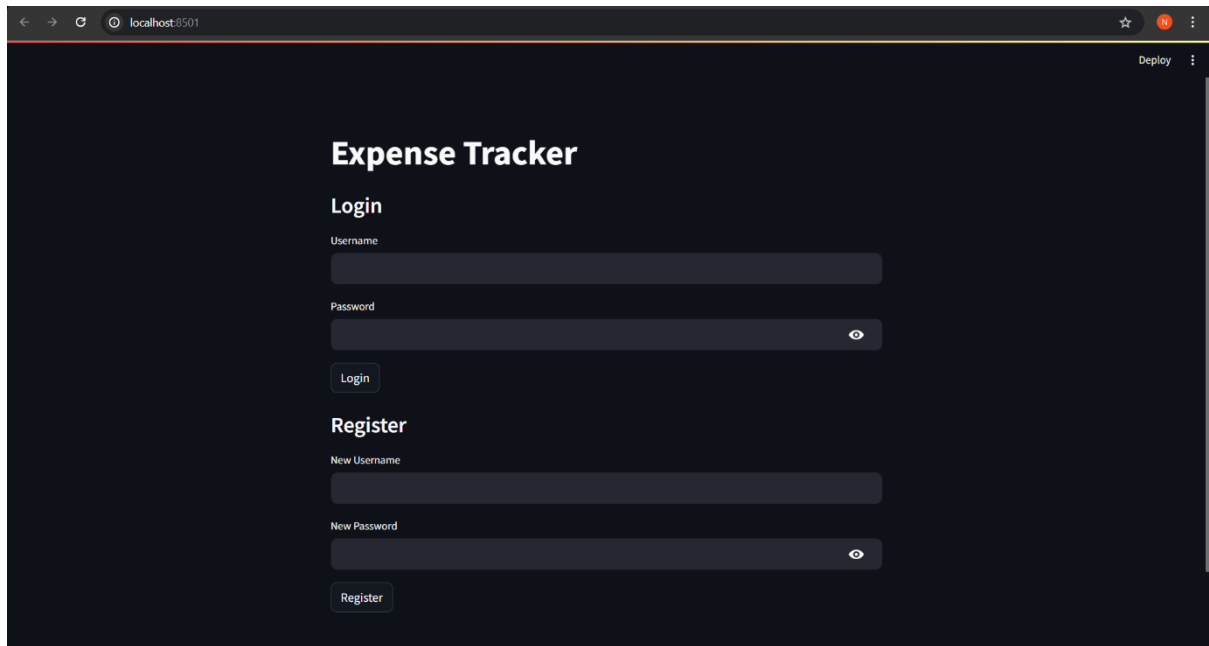
5. RESULT AND DISCUSSION

5.1 USER DOCUMENTATION

LOGIN PAGE

REGISTER

LOGIN

EXPENSE TRACKER
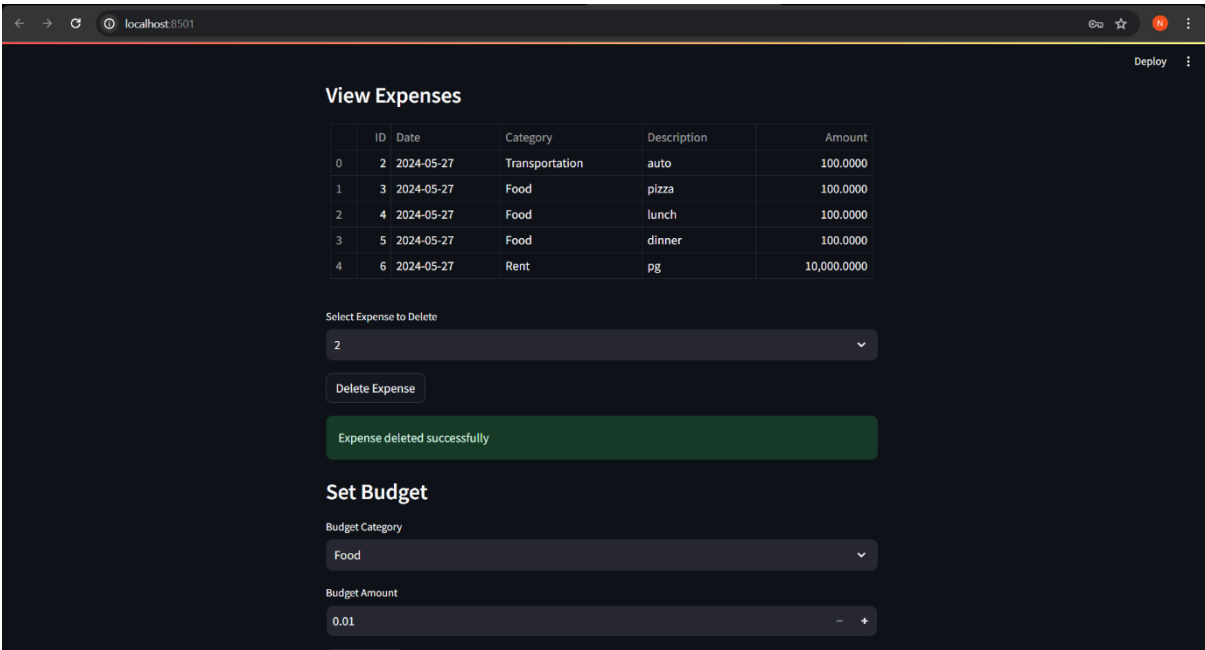
ADDING EXPENSE

SELECTING CATEGORY
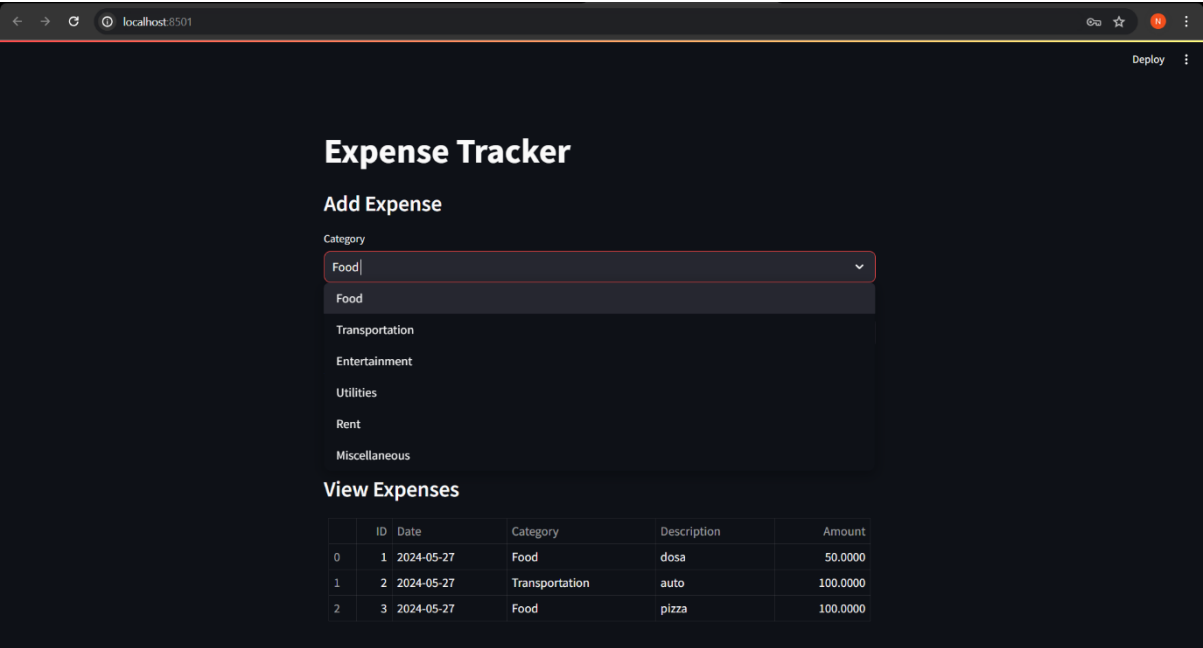
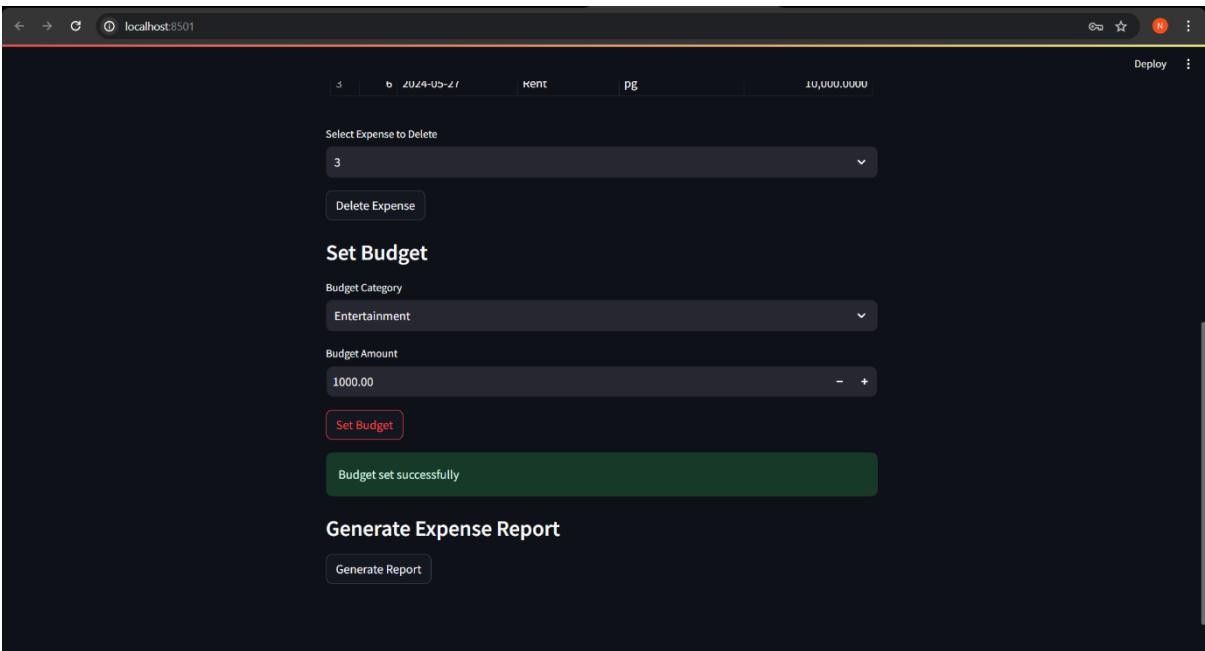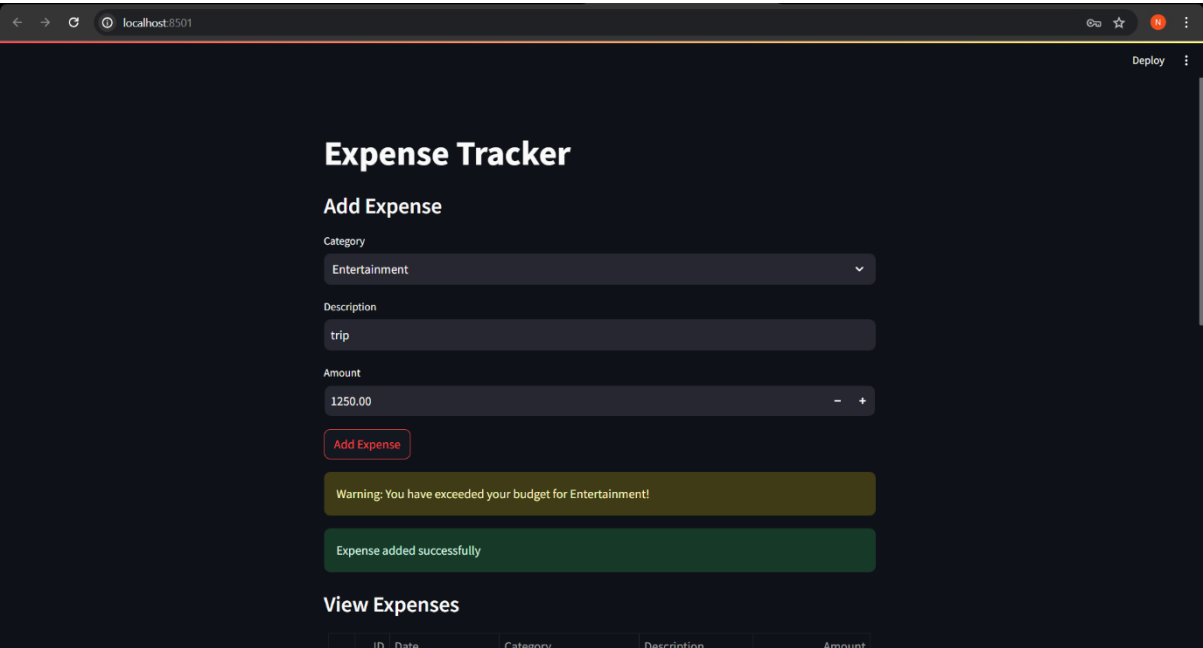**ENTERING DESCRIPTION AND AMOUNT**
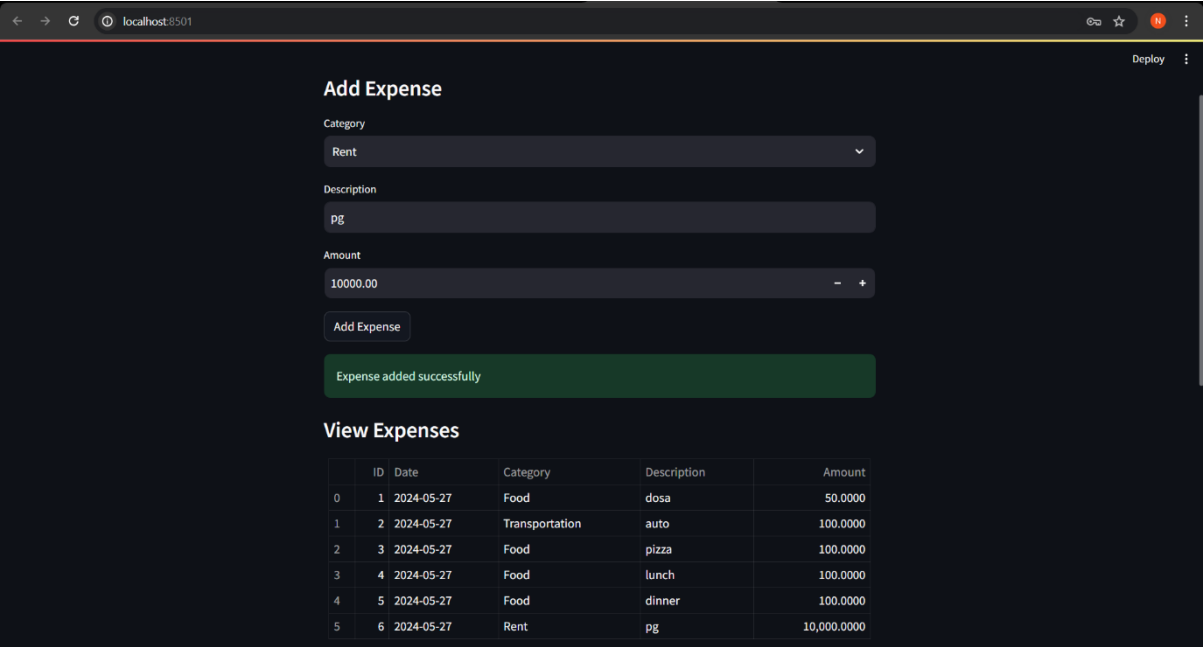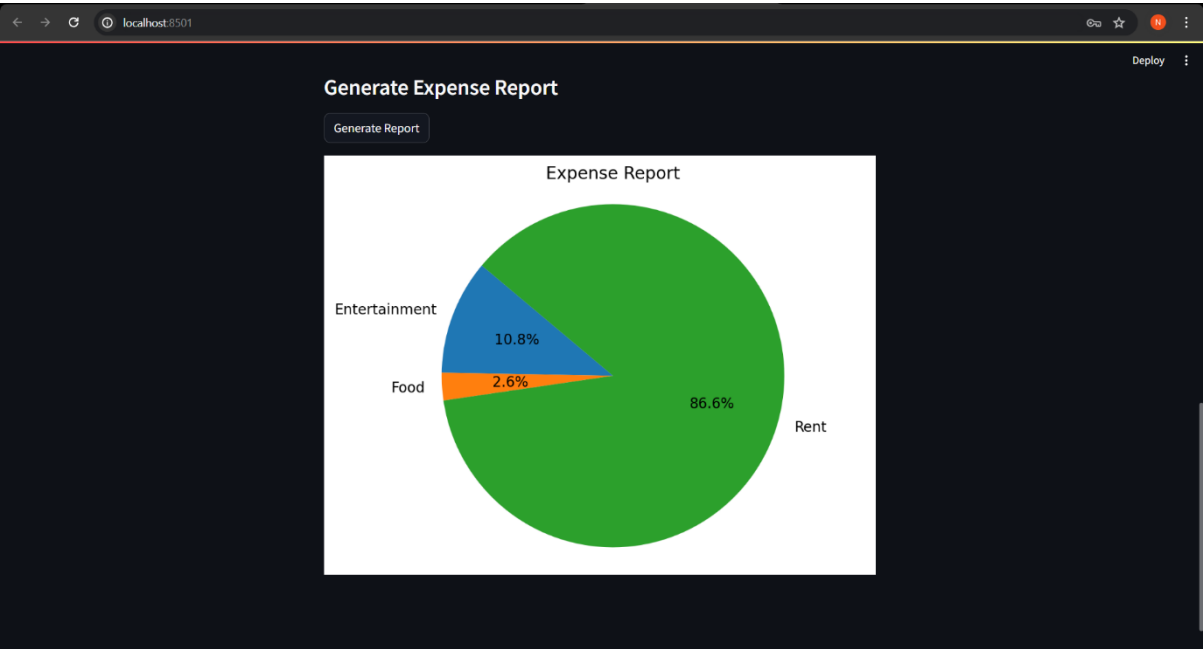
DELETING EXPENSE

SETTING BUDGET

WARNING MESSAGE WHEN EXCEEDING BUDGET

EXPENSE ANALYTICS

6. TESTING

6.1 UNIT TESTING

Unit testing is a testing technique in which modules are tested individually. Small individual units of source code are tested to determine whether it is fit to use or not. Different modules of games are put to test while the modules are being developed. Here modules refer to individual levels, players, scenes

6.2 INTEGRATION TESTING

Integration testing is the technique in which individual components or modules are grouped together and tested. It occurs after testing. The inputs for the integrated testing are the modules that have already been unit tested.

6.3 SYSTEM TESTING

System testing is conducted on the entire system to check whether the system meets its requirements or not. software was installed on different systems and any errors or bugs that occurred were fixed.

6.4 ACCEPTANCE TESTING

User Acceptance is defined as a type of testing performed by the Client to certify the system with respect to the requirements that was agreed upon. This testing happens in the final phase of testing before moving the software application to the Market or Production environment.

7. CONCLUSION

CONCLUSION

The expense tracker application serves as a powerful tool for individuals seeking to manage their personal finances effectively. By leveraging Python for robust backend logic, SQLite for efficient data storage, and Streamlit for an interactive user interface, the application offers a comprehensive solution for tracking expenses, setting and monitoring budgets, and managing account balances. With features such as secure user authentication, detailed expense logging, budget alerts, and insightful visual reports, users are empowered to make informed financial decisions and maintain better control over their spending. The application's simplicity, combined with its robust functionality, makes it an invaluable resource for achieving financial stability and transparency.

REFERENCES

1. Python:

   - Official Documentation: [Python Documentation] (https://docs.python.org/3/)

   - Python Programming Language: Python Software Foundation

2. SQLite:

   - Official Documentation: [SQLite Documentation] (https://sqlite.org/docs.html)

   - SQLite Database: D. Richard Hipp

3. Pandas:

   - Official Documentation: [Pandas Documentation] (https://pandas.pydata.org/pandas-docs/stable/)

   - Pandas Library: Wes McKinney

4. Matplotlib:

   - Official Documentation: [Matplotlib Documentation] (https://matplotlib.org/stable/contents.html)

   - Matplotlib Library: John D. Hunter and the Matplotlib development team

5. Streamlit:

   - Official Documentation: [Streamlit Documentation] (https://docs.streamlit.io/)

   - Streamlit Library: Streamlit Inc.

6. DB Browser for SQLite:

   - Official Website: [DB Browser for SQLite] (https://sqlitebrowser.org/)

7. Mermaid.js:

   - Official Documentation: [Mermaid.js Documentation] (https://mermaid-js.github.io/mermaid/#/)

   - Mermaid.js Library: Knut Sveidqvist

8. SQLiteStudio:

- Official Website: [SQLiteStudio](https://sqlitestudio.pl/)

9. SQLite Manager (Firefox Add-on):

   - Official Page: [SQLite Manager Add-on] (https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/)

These references provide comprehensive information and documentation on the tools, libraries, and technologies used in the expense tracker project, including the data flow diagrams and the code for building the application.