SRM UNIVERSITY AP

DATABASE MANAGEMENT SYSTEM REPORT

Submitted for partial fulfilment for the award of the degree in
Bachelor or Technology
in
Computer Science and Engineering

EXPENSE TRACKER

USING PYTHON AND SQL

Submitted by :

- Chukka Chanakya Devendra
  AP21110011577
- Lolugu Charansai Venkat Nanarayana
  AP21110011621
- Aditya Sesha Sai Samineni
  AP21110011627
- Phaneendra Swamy Pasem
  AP21110011611

Page Intentionally Left Blank
for printing purposes

Problem Statement

The Expense Tracker project aims to address the challenges individuals face in managing and tracking their personal expenses effectively. In today's fast-paced world, keeping a meticulous record of daily expenditures, incomes, and maintaining a budget can be a daunting task. Many individuals struggle to maintain financial discipline due to a lack of efficient tools for expense management. Existing solutions may be complex, lack user-friendliness, or not cater to the specific needs of users.

Common problems include:

1. **Lack of User-Friendly Expense Tracking Tools:**
    - Many available expense tracking tools are overly complex, making it challenging for users to navigate and utilize them effectively.
2. **Difficulty in Budget Management:**
    - Individuals often find it hard to set and stick to a budget, leading to overspending and financial stress.
3. **Inefficient Record Keeping:**
    - Traditional methods of manual record-keeping can be time-consuming and prone to errors, hindering individuals from gaining accurate insights into their financial habits.
4. **Limited Integration of Features:**
    - Some existing tools may lack comprehensive features, such as budget visualization, transaction categorization, and timely alerts, limiting users' ability to make informed financial decisions.

The Expense Tracker project seeks to address these issues by providing a user-friendly and efficient platform for users to track their expenses, manage budgets, and gain valuable insights into their financial habits. Through a streamlined and intuitive interface, users can easily input and monitor their expenditures and revenues, set budgets, and receive timely notifications, fostering better financial management practices.

Objective

The primary objectives of the Expense Tracker project are as follows:

1. **Efficient Expense Tracking:**
    - Develop a user-friendly interface that allows individuals to effortlessly input and track their daily expenses and revenues.
2. **Budget Management:**
    - Implement a budget management system that enables users to set spending thresholds and receive notifications when approaching or exceeding their budget limits.
3. **User Authentication and Security:**
    - Ensure the security of user data by implementing robust user authentication mechanisms, including password hashing, to safeguard personal and financial information.
4. **Comprehensive User Registration:**
    - Facilitate a smooth user registration process, validating user input for emails, mobile numbers, and usernames to ensure uniqueness and adherence to standard formats.
5. **Data Visualization:**
    - Provide users with visual representations of their spending patterns through graphs and charts, allowing for a quick and easy understanding of their financial habits.
6. **Expense and Revenue Categorization:**
    - Allow users to categorize their expenses and revenues, providing a detailed breakdown to enhance insights into their financial activities.
7. **Notification System:**
    - Implement a notification system to alert users when they approach or surpass their budget limits, encouraging better financial discipline.
8. **Future Scalability:**
    - Design the system architecture to be scalable, allowing for future enhancements and additional features to meet evolving user needs.
9. **User-Friendly Interface:**
    - Create an intuitive and visually appealing frontend using Streamlit, making it accessible for users with varying levels of technical expertise.
10. **Database Integration:**
    - Integrate a MySQL database to securely store user information, transactions, and budget-related data.

By achieving these objectives, the Expense Tracker project aims to empower users with a comprehensive and user-friendly tool to manage their personal finances effectively, promote better spending habits, and ultimately contribute to their overall financial well-being.

Abstract

The Expense Tracker project presents a comprehensive solution to address the challenges individuals encounter in managing personal finances. In a world characterized by hectic lifestyles and dynamic financial landscapes, maintaining meticulous control over daily expenditures is crucial for financial well-being. This project introduces an intuitive and efficient Expense Tracker system, designed to simplify expense management, budget tracking, and financial planning.

The system's key features include a user-friendly interface developed using Streamlit, allowing users to effortlessly record and monitor their daily expenses and revenues. Through a secure backend powered by MySQL, user data is managed with a focus on authentication, ensuring the confidentiality and integrity of personal and financial information.

The project's objectives include efficient expense tracking, budget management, user authentication, data visualization, and notification systems. Users can categorize their expenses and revenues, gaining valuable insights through visually appealing charts and graphs. The notification system alerts users when they approach or exceed predefined budget limits, encouraging responsible financial habits.

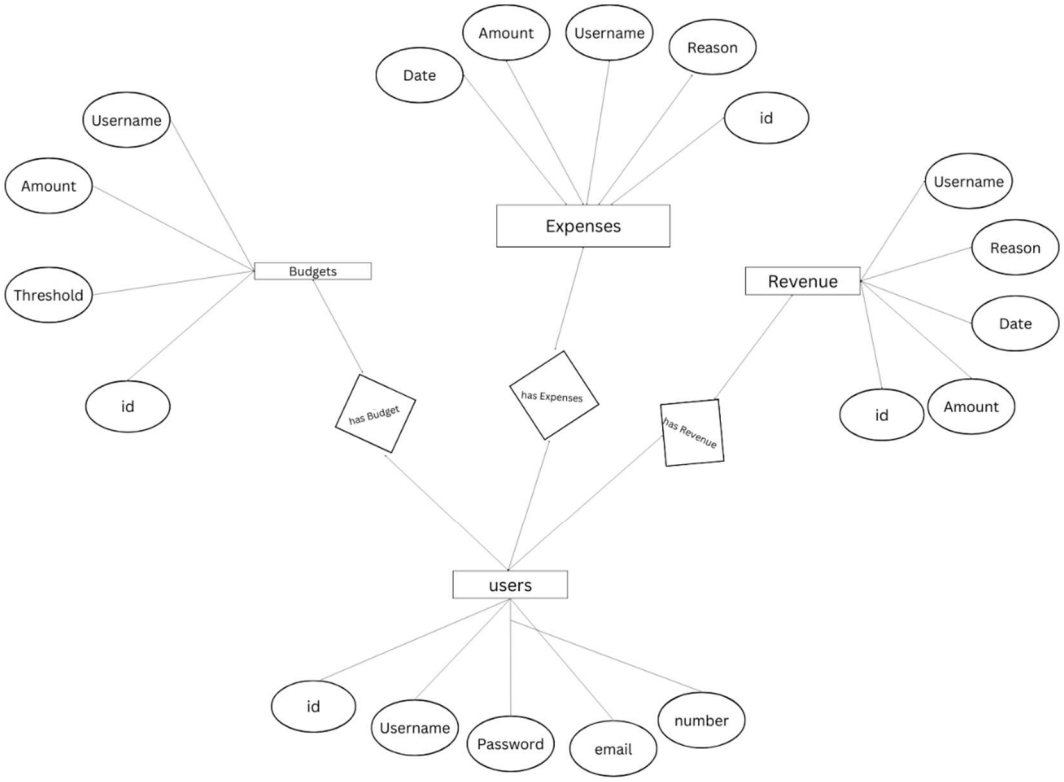
The Expense Tracker system not only simplifies the user experience but also promotes financial literacy by providing clear and actionable insights into spending patterns. With an emphasis on scalability, the project lays the foundation for future enhancements and additional features to adapt to evolving user needs.

In conclusion, the Expense Tracker project offers a user-centric and technologically advanced solution to the challenges of personal financial management. By combining user-friendly design with robust security measures and insightful data visualization, the system empowers individuals to take control of their finances, make informed decisions, and work towards achieving their financial goals.

System Design

ER Diagram

The Entity-Relationship (ER) Diagram illustrates the relationships between entities in the Expense Tracker system.

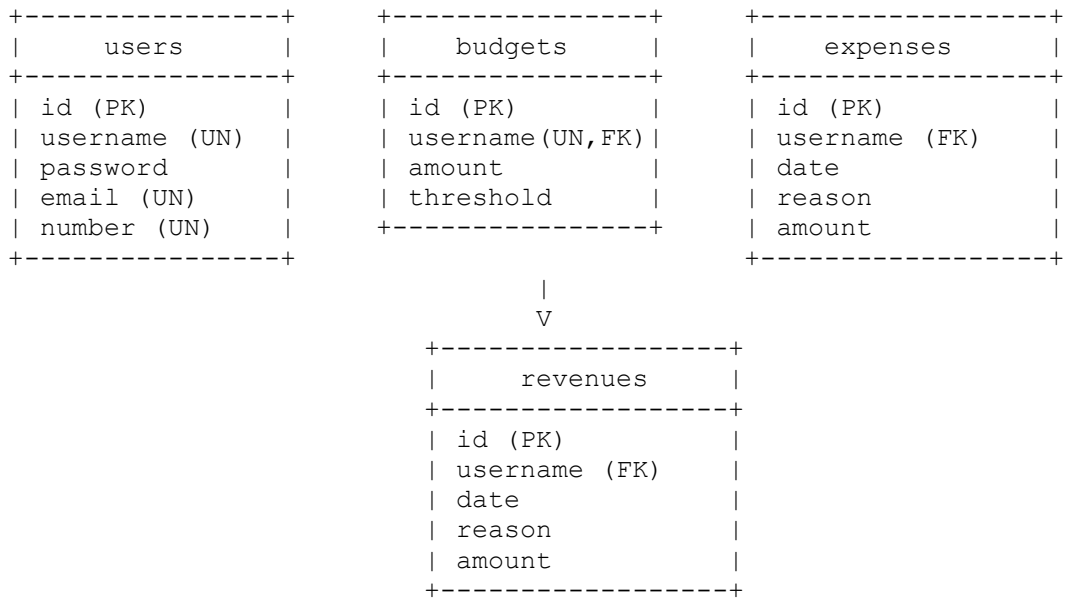


Schema Diagram and Tables

The schema diagram represents the structure of the database tables used in the Expense Tracker system. Each table corresponds to a specific entity, and relationships between entities are established using foreign keys.

1. **users:**
   - Attributes: `id` (Primary Key), `username` (Unique), `password`, `email` (Unique), `number` (Unique)
2. **budgets:**
   - Attributes: `id` (Primary Key), `username` (Foreign Key), `amount`, `threshold`
3. **expenses:**
   - Attributes: `id` (Primary Key), `username` (Foreign Key), `date`, `reason`, `amount`
4. **revenues:**
   - Attributes: `id` (Primary Key), `username` (Foreign Key), `date`, `reason`, `amount`

These tables are interconnected through foreign key relationships, enabling the system to maintain data integrity and provide a comprehensive view of user transactions and budgetary information.



References:

- `(PK)`: Primary Key
- `(UN)`: Unique Constraint
- `(FK)`: Foreign Key

Implementation

frontend.py

```
import streamlit as st
from streamlit_option_menu import option_menu
import backend as db
import validations as val
import time
import send_mail as sm
import hasher as hs
import matplotlib.pyplot as plt
import pandas as pd

#-----------------------------------------------------
# page config settings:

page_title="Expense Tracker"
page_icon=":dollar:"
layout="centered"

st.set_page_config(page_title=page_title,page_icon=page_icon,layout=layout)
st.title(page_title+" "+page_icon)

#-----------------------------------------------------
#hide the header and footer

hide_ele="""
        <style>
        #Mainmenu {visibility:hidden;}
        footer {visibility:hidden;}
        header {visibility:hidden;}
        </style>
        """
st.markdown(hide_ele,unsafe_allow_html=True)
#-----------------------------------------------------
curlogin=""
otp=""

def log_sign():
    selected=option_menu(
        menu_title=None,
        options=["Login","Signup"],
        icons=["bi bi-fingerprint","bi bi-pencil-square"],
        orientation="horizontal"
    )
    global submit
    if(selected=="Login"):
        tab1,tab2=st.tabs(["Login","Forgot Password"])
        with tab1:
            with st.form("Login",clear_on_submit=True):
                st.header("Login")
                username=st.text_input("Username")
                password=st.text_input("Password",type="password")
                submit=st.form_submit_button()
                if(submit):
                    if(username=="" or password==""):
                        st.warning("Enter your login credentials")
                    else:
                        password=hs.hasher(password)
                        if(db.authenticate(username,password)):
                            st.session_state["curlogin"]=username
                            st.session_state["key"]="main"
                            st.experimental_rerun()
                        else:
```

```
                            st.error("Please check your username / password ")
        with tab2:
            with st.form("Forgot Password",clear_on_submit=True):
                st.header("Forgot Password")
                email=st.text_input("Email")
                submit=st.form_submit_button()
                if(submit):
                    if(email==""):
                        st.warning("Enter your email")
                    elif(not db.emailexists(email)):
                        st.warning("User with associated email is not found,kindly
recheck the email!")
                    else:
                        otp=sm.forgot_password(email)
                        db.forgot_pass(email,otp)
                        st.success("Check your email for password reset
instructions!.")

    elif(selected=="Signup"):
         with st.form("Sign Up",clear_on_submit=False):
            st.header("Sign Up")
            email=st.text_input("Enter your email")
            number=st.text_input("Enter your Mobile Number")
            username=st.text_input("Enter your username")
            password=st.text_input("Enter your password",type="password")
            submit=st.form_submit_button()
            if(submit):
                var=True
                emails=db.get_all_emails()
                numbers=db.get_all_numbers()
                usernames=db.get_all_usernames()
                if(db.check_user_existence(username,email,number)):
                   var=False
                if(val.validate_email(email)==False):
                    st.error("Enter email in a valid format like 'yourname@org.com'")
                elif(email in emails):
                    st.error("email already exists!\\nTry with another email !")
                elif(val.validate_mobile(number)==False):
                    st.error("Please Check your mobile Number")
                elif(number in numbers):
                    st.error("Phone number already exists\\nTry with another number")
                elif(val.validate_username(username)==False):
                    st.error("Invalid Username!\\nUsername must be between 4-20
characters and can contain only _ and . , and username cannot begin with special
characters")
                elif(username in usernames):
                    st.error("Username already exists!\\nTry another username !")
                elif(val.validate_password(password)==False):
                    st.error("Password must be between 6-20 characters in length and
must have at least one Uppercase Letter , Lowercase letter , numeric character and A
Special Symbol(#,@,$,%,^,&,+,=)")
                elif(var):
                    password=hs.hasher(password)
                    db.insert_user(username,password,email,number)
                    st.success("Signed Up Successfully....Redirecting!!")
                    time.sleep(2)
                    st.session_state["curlogin"]=username
                    st.session_state["key"]="main"
                    st.experimental_rerun()

def main():
    btn=st.button("Logout")
    if(btn):
        st.session_state["key"] = "log_sign"
        st.experimental_rerun()
    selected=option_menu(
            menu_title=None,
```

```
            options=["Manage Expenses","View Spendings"],
            icons=["bi bi-search","bi bi-box"],
            orientation="horizontal"
        )
    if selected=="Manage Expenses":
        st.header("Budget Remaining :
"+str(db.get_budget(st.session_state["curlogin"])[0]))

if(int(db.get_budget(st.session_state["curlogin"])[0])<int(db.get_budget(st.session_sta
te["curlogin"])[1])):
            st.error("Budget is less than your st threshold")
        else:
            st.success("You are in the safe zone now")
        tab1,tab2=st.tabs(["Expense","Revenue"])
        with tab1:
            with st.form("expense_input",clear_on_submit=True):
                st.header("Expense")
                date=st.date_input("Enter the date of the expense")
                reason=st.text_input("Enter the reason for
spending",placeholder="Canteen , Records etc.")
                amount=st.text_input("Amount in Rupees?",placeholder="1000 etc..")
                submitted=st.form_submit_button("Submit")
                if(submitted):
                    if(db.get_budget(st.session_state["curlogin"])[0]>=int(amount)):

db.insert_expense(st.session_state["curlogin"],date.strftime("%d/%m/%Y"),reason,int(amo
unt))

        with tab2:
            with st.form("revenue_input",clear_on_submit=True):
                st.header("Revenue")
                date=st.date_input("Enter the date of the revenue")
                reason=st.text_input("Enter the source of
revenue",placeholder="Allowance , Pocket Money etc.")
                amount=st.text_input("Amount in Rupees?",placeholder="1000 etc..")
                submitted=st.form_submit_button("Submit")
                if(submitted):

db.insert_revenue(st.session_state["curlogin"],date.strftime("%d/%m/%Y"),reason,int(amo
unt))
    else:
        def plot_bar_charts(expenses, revenues):
            if expenses and len(expenses) > 0:
                df_expenses = pd.DataFrame(expenses, columns=["Date", "Reason",
"Amount"])
                df_expenses = df_expenses.sort_values(by="Date")
                st.subheader("Expense Data:")
                st.dataframe(df_expenses)
                fig, ax1 = plt.subplots()
                ax1.bar(df_expenses["Date"], df_expenses["Amount"], color='red',
alpha=0.7)
                ax1.set_xlabel('Date')
                ax1.set_ylabel('Amount Spent', color='red')
                ax1.tick_params('y', colors='red')
                ax1.set_title("Expense Overview")
                plt.xticks(rotation=45)
                st.pyplot(fig)
                fig, ax3 = plt.subplots()
                expense_reasons = df_expenses.groupby("Reason").sum()["Amount"]
                ax3.pie(expense_reasons, labels=expense_reasons.index,
autopct='%1.1f%%', startangle=90)
                ax3.axis('equal')
                ax3.set_title("Expense Distribution by Reasons")
                st.pyplot(fig)

            if revenues and len(revenues) > 0:
```

```
                df_revenues = pd.DataFrame(revenues, columns=["Date", "Reason",
"Amount"])
                df_revenues = df_revenues.sort_values(by="Date")

                st.subheader("Revenue Data:")
                st.dataframe(df_revenues)

                fig, ax2 = plt.subplots()
                ax2.bar(df_revenues["Date"], df_revenues["Amount"], color='green',
alpha=0.7)
                ax2.set_xlabel('Date')
                ax2.set_ylabel('Amount Received', color='green')
                ax2.tick_params('y', colors='green')
                ax2.set_title("Revenue Overview")
                plt.xticks(rotation=45)
                st.pyplot(fig)
                fig, ax4 = plt.subplots()
                revenue_reasons = df_revenues.groupby("Reason").sum()["Amount"]
                ax4.pie(revenue_reasons, labels=revenue_reasons.index,
autopct='%1.1f%%', startangle=90)
                ax4.axis('equal')
                ax4.set_title("Revenue Distribution by Reasons")
                st.pyplot(fig)
        expenses, revenues = db.get_user_transactions(st.session_state["curlogin"])
        plot_bar_charts(expenses, revenues)
        plot_bar_charts(expenses, revenues)

if "key" not in st.session_state:
    st.session_state["key"] = "log_sign"

if st.session_state["key"] == "log_sign":
    log_sign()

elif st.session_state["key"] == "main":
    main()
```

backend.py

```
import mysql.connector
from mysql.connector import Error
import os
from dotenv import load_dotenv
from datetime import datetime, timedelta

load_dotenv(".env")

def create_connection():
    try:
        cnx = mysql.connector.connect(user='root', password='Dev_Password',
                                      host='localhost',
                                      database='expense_tracker')
        return cnx
    except Error as e:
        print(f"Error: {e}")
        return None

def close_connection(cnx):
    if cnx:
        cnx.close()

def create_tables():
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
```

```
            # Create users table
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    username VARCHAR(255) UNIQUE NOT NULL,
                    password VARCHAR(255) NOT NULL,
                    email VARCHAR(255) UNIQUE NOT NULL,
                    number VARCHAR(20) UNIQUE NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS budgets (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    username VARCHAR(255) UNIQUE NOT NULL,
                    amount DECIMAL(10, 2) NOT NULL,
                    threshold DECIMAL(10, 2) NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS expenses (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    username VARCHAR(255) NOT NULL,
                    date DATE NOT NULL,
                    reason VARCHAR(255) NOT NULL,
                    amount DECIMAL(10, 2) NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS revenues (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    username VARCHAR(255) NOT NULL,
                    date DATE NOT NULL,
                    reason VARCHAR(255) NOT NULL,
                    amount DECIMAL(10, 2) NOT NULL
                )
            """)

            cnx.commit()
            print("Tables created successfully!")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def insert_user(username, password, email, number):
    print("Creating user")
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "INSERT INTO users (username, password, email, number) VALUES (%s,
%s, %s, %s)"
            data = (username, password, email, number)
            cursor.execute(query, data)
            cnx.commit()
            print("User inserted successfully!")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def authenticate(username, password):
```

```
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT * FROM users WHERE username = %s AND password = %s"
            data = (username, password)
            cursor.execute(query, data)
            result = cursor.fetchone()
            return result is not None
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return False

create_tables()

def login(username, password):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT * FROM users WHERE username = %s AND password = %s"
            data = (username, password)
            cursor.execute(query, data)
            result = cursor.fetchone()
            return result
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return None

def signup(email, number, username, password):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "INSERT INTO users (email, number, username, password) VALUES (%s,
%s, %s, %s)"
            data = (email, number, username, password)
            cursor.execute(query, data)
            cnx.commit()
            print("User signed up successfully!")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def check_user_existence(username=None, email=None, number=None):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            conditions = []
            data = []

            if username is not None:
                conditions.append("username = %s")
                data.append(username)

            if email is not None:
                conditions.append("email = %s")
                data.append(email)
```

```
            if number is not None:
                conditions.append("number = %s")
                data.append(number)

            query = f"SELECT * FROM users WHERE {' AND '.join(conditions)}"
            cursor.execute(query, tuple(data))
            result = cursor.fetchone()
            return result is not None
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return False

def get_all_emails():
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT email FROM users"
            cursor.execute(query)
            result = cursor.fetchall()
            return [record[0] for record in result]
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return []

def get_all_usernames():
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT username FROM users"
            cursor.execute(query)
            result = cursor.fetchall()
            return [record[0] for record in result]
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return []

def get_all_numbers():
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT number FROM users"
            cursor.execute(query)
            result = cursor.fetchall()
            return [record[0] for record in result]
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return []

def set_budget(username, amount, threshold):
    cnx = create_connection()
    if cnx:
```

```
        try:
            cursor = cnx.cursor()
            query = "INSERT INTO budgets (username, amount, threshold) VALUES (%s, %s,
%s) ON DUPLICATE KEY UPDATE amount = %s, threshold = %s"
            data = (username, amount, threshold, amount, threshold)
            cursor.execute(query, data)
            cnx.commit()
            print("Budget set successfully!")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def get_budget(username):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            query = "SELECT budget_amount, threshold FROM budgets WHERE username = %s"
            data = (username,)
            cursor.execute(query, data)
            result = cursor.fetchone()
            return result if result else None
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return None

def insert_expense(username, date, reason, amount):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
            formatted_date = datetime.strptime(date, "%d/%m/%Y").strftime("%Y-%m-%d")

            cnx.start_transaction()

            try:
                # Insert expense entry
                query_expense = "INSERT INTO expenses (username, date, reason, amount)
VALUES (%s, %s, %s, %s)"
                data_expense = (username, formatted_date, reason, amount)
                cursor.execute(query_expense, data_expense)
                query_budget = "UPDATE budgets SET budget_amount = budget_amount - %s
WHERE username = %s"

                data_budget = (amount, username)
                cursor.execute(query_budget, data_budget)
                cnx.commit()
                print("Expense entry added successfully!")
            except Error as e:
                cnx.rollback()
                print(f"Transaction failed. Error: {e}")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def insert_revenue(username, date, reason, amount):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()
```

```
            formatted_date = datetime.strptime(date, "%d/%m/%Y").strftime("%Y-%m-%d")

            cnx.start_transaction()

            try:
                query_revenue = "INSERT INTO revenues (username, date, reason, amount)
VALUES (%s, %s, %s, %s)"
                data_revenue = (username, formatted_date, reason, amount)
                cursor.execute(query_revenue, data_revenue)
                query_budget = "UPDATE budgets SET budget_amount = budget_amount + %s
WHERE username = %s"
                data_budget = (amount, username)
                cursor.execute(query_budget, data_budget)

                cnx.commit()
                print("Revenue entry added successfully!")
            except Error as e:
                cnx.rollback()
                print(f"Transaction failed. Error: {e}")
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)

def get_user_transactions(username):
    cnx = create_connection()
    if cnx:
        try:
            cursor = cnx.cursor()

            query_expenses = """
                SELECT date, reason, amount
                FROM expenses
                WHERE username = %s AND date >= CURDATE() - INTERVAL 30 DAY
                ORDER BY date DESC
            """
            data = (username,)
            cursor.execute(query_expenses, data)
            expenses = cursor.fetchall()

            query_revenues = """
                SELECT date, reason, amount
                FROM revenues
                WHERE username = %s AND date >= CURDATE() - INTERVAL 30 DAY
                ORDER BY date DESC
            """
            cursor.execute(query_revenues, data)
            revenues = cursor.fetchall()

            print("Expenses:", expenses)
            print("Revenues:", revenues)

            return expenses, revenues
        except Error as e:
            print(f"Error: {e}")
        finally:
            cursor.close()
            close_connection(cnx)
    return None, None

# insert_expense("mrdev","15/11/2023","Canteen",50)
#insert user working for creating new users
#authrnticate working for login
```

validations.py

```
import re

#regular expression for email validation
regex_e=r'\\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Z|a-z]{2,}\\b'
regex_m=r'(""|91)?[6-9][0-9]{9}'
regex_p='^(?=.*[0-9])'+'(?=.*[1-z])(?=.*[A-
Z])'+'(?=.*[@#$%^&+=])'+'(?=\\\\S+$).{6,20}$'
regex_u='^(?=[a-zA-Z0-9._]{4,20}$)(?!.*[_.]{2})[^_.].*[^_.]$'
def validate_email(email):
    if(re.fullmatch(regex_e,email)):
        return True
    else:
        return False

def validate_mobile(number):
    if((len(number)==13 and number[0:3]=="+91")or(len(number)==12 and
number[0:2]=="91")):
        number=number[3:]
    if(re.fullmatch(regex_m,number)):
        return True
    else:
        return False

def validate_username(username):
    if(re.fullmatch(regex_u,username)):
        return True
    else:
        return False

def validate_password(password):
    if(re.fullmatch(regex_p,password)):
        return True
    else:
        return False
```

hasher.py

```
strs = 'abcdefghijklmnopqrstuvwxyz'
def hasher(password):
    shift=1
    data = []
    for i in password:
        if i.strip() and i in strs:
            data.append(strs[(strs.index(i) + shift) % 26])
        else:
            data.append(i)
    output = ''.join(data)
    return output
```

send_mail.py

```
import smtplib
import ssl
import random as ran
import os
def forgot_password(email_to):
    smtp_port=587
    smtp_server="smtp.gmail.com"

    email_from="mrdevinfinity@gmail.com"
    passw=os.getenv("pass")
```

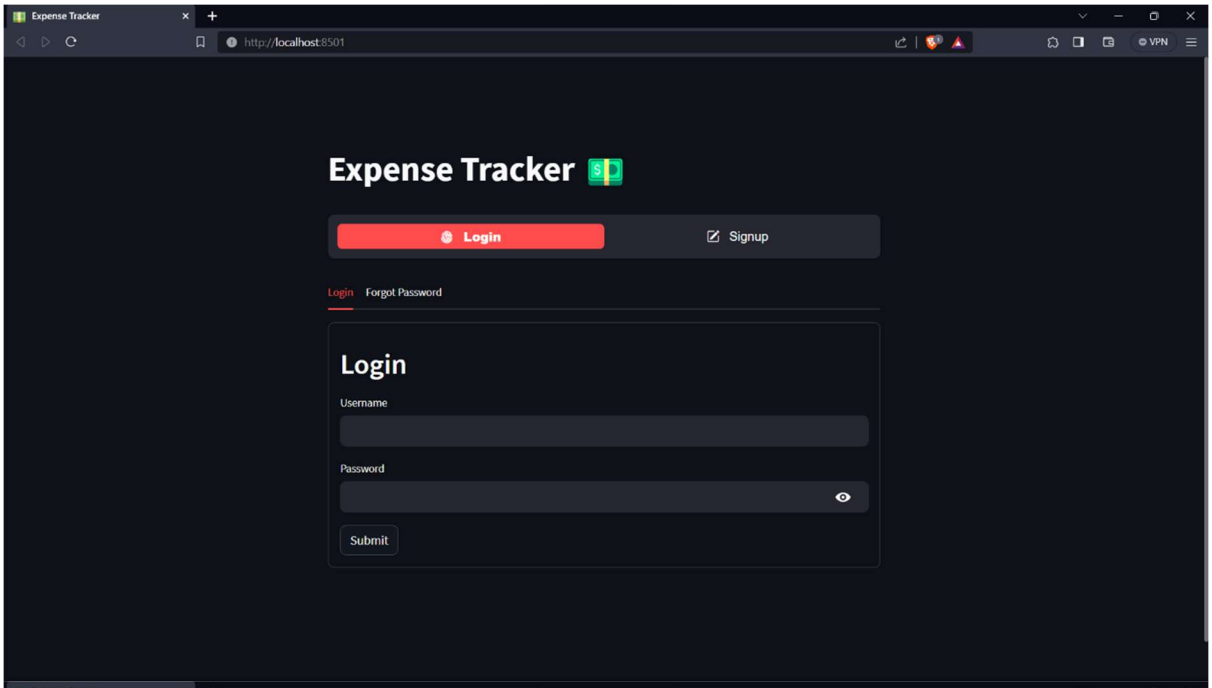
```
    def otpgen():
        otp=""
        for i in range(6):
            otp+=str(ran.randint(1,9))
        return "Your 6 Digit OTP for Expense Tracker is "+otp
    message=otpgen()
    email_context = ssl.create_default_context()

    try:
        TIE_server = smtplib.SMTP(smtp_server,smtp_port)
        TIE_server.starttls(context=email_context)
        TIE_server.login(email_from,passw)
        TIE_server.sendmail(email_from,email_to,message)
        TIE_server.quit()
        return message[46:52]
    except Exception as d:
        print(d)
        return -1
```

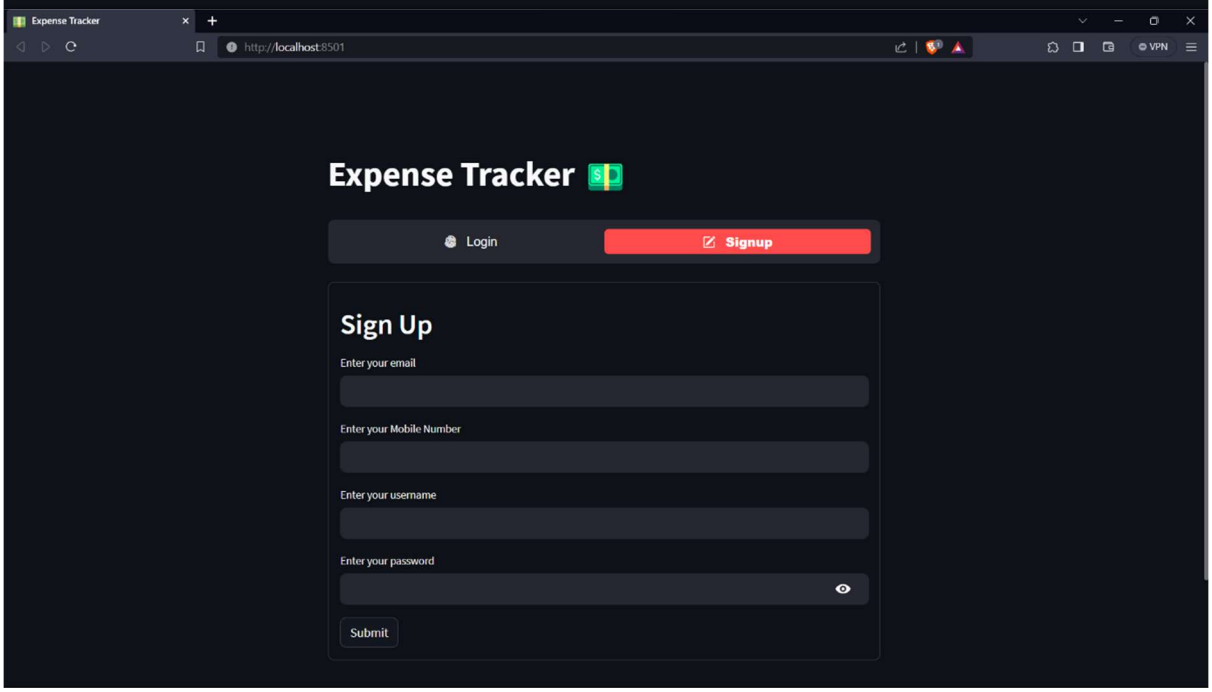
requirement.txt (for github)

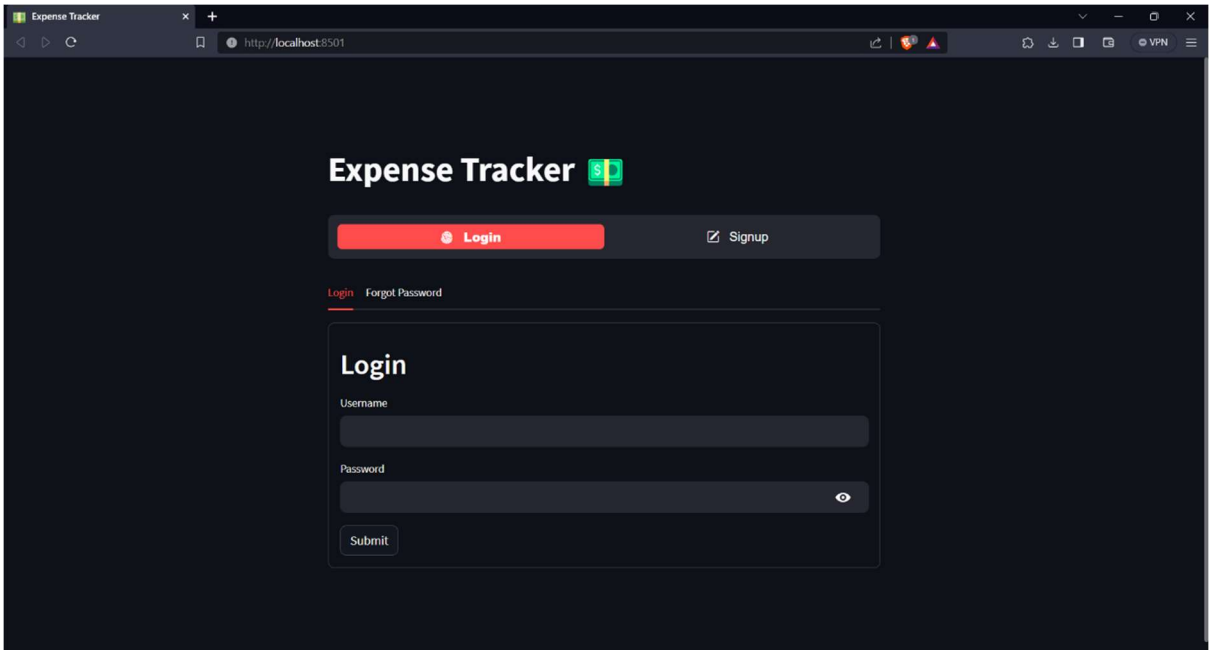
```
streamlit
streamlit-option-menu
deta
python-dotenv
pandas
matplotlib
```

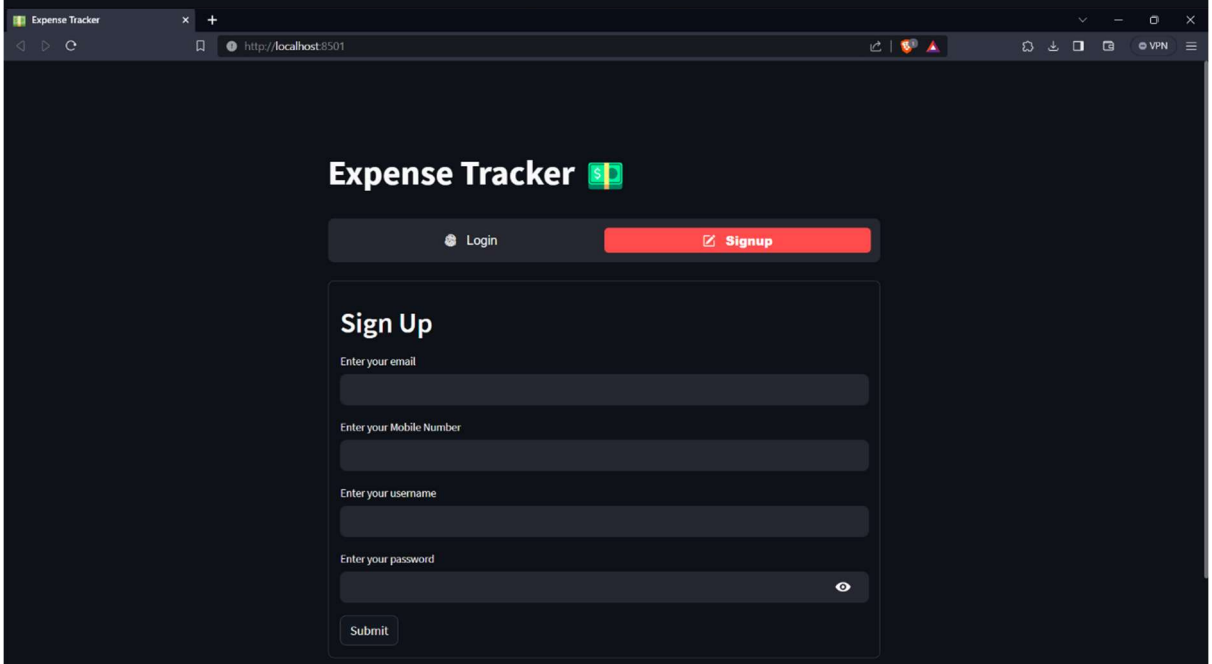
Screenshots

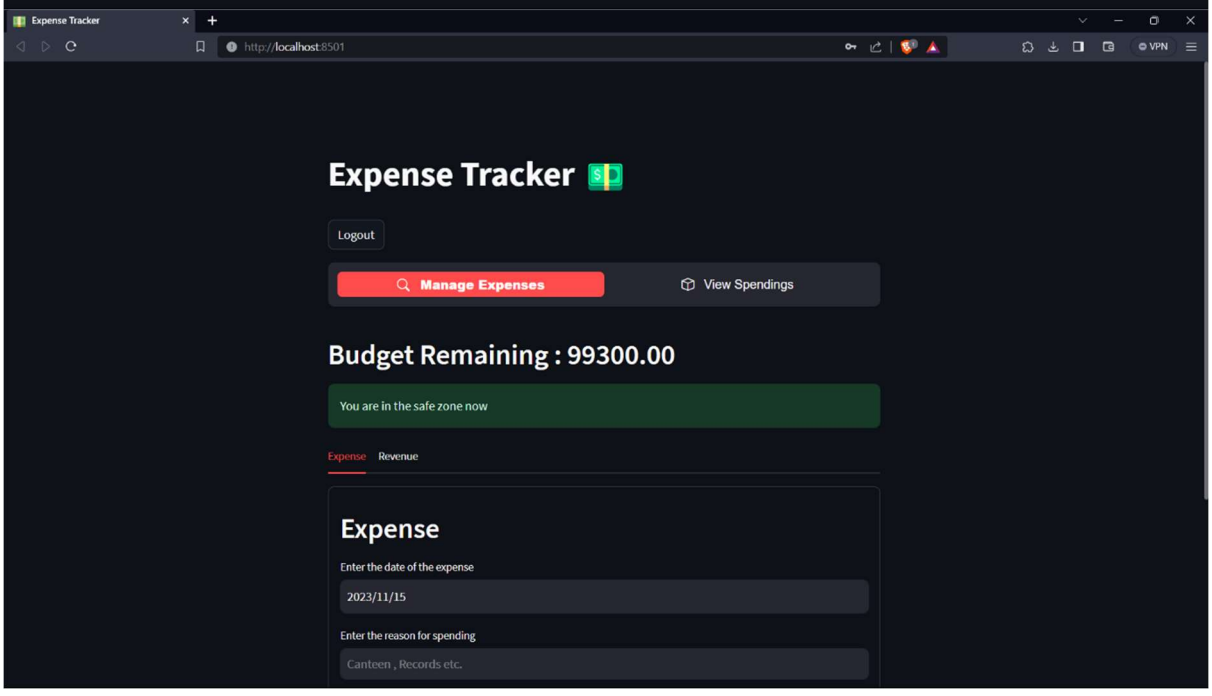
Frontend

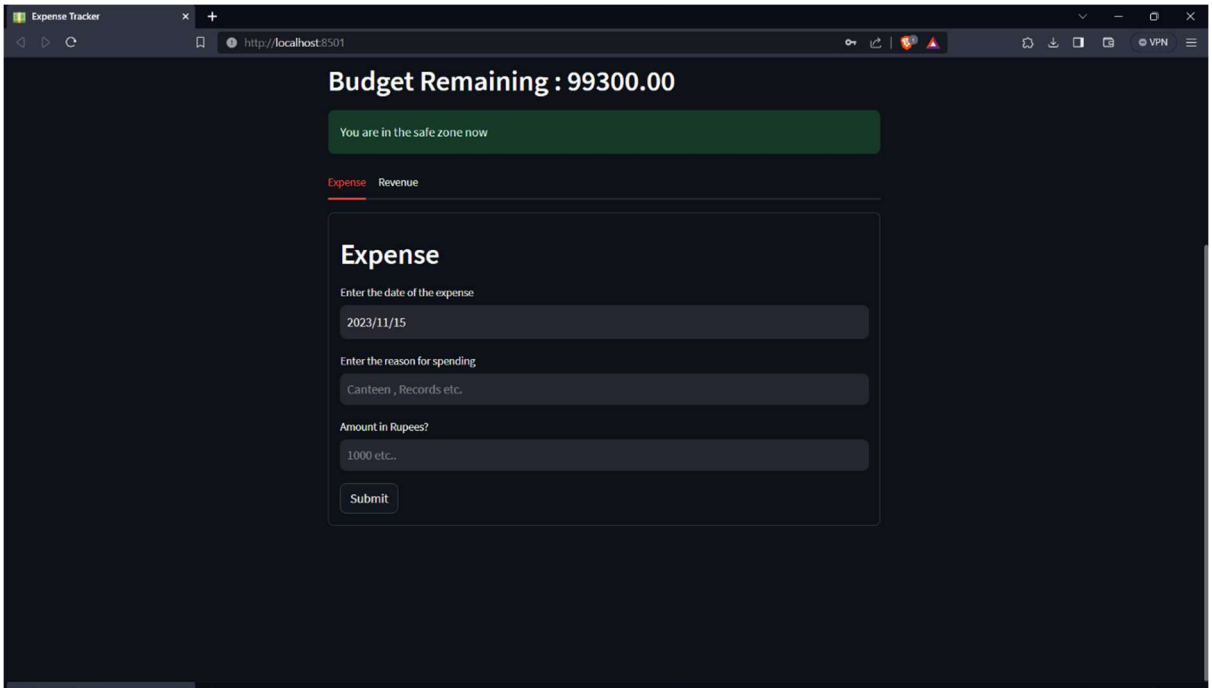


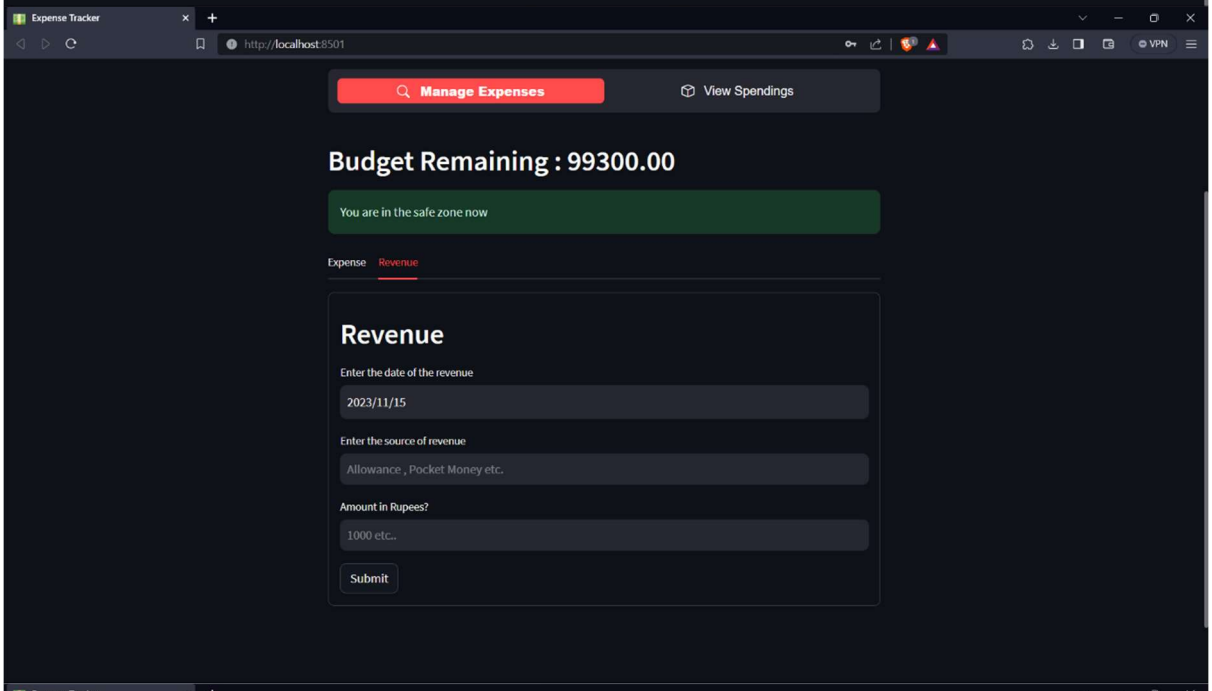


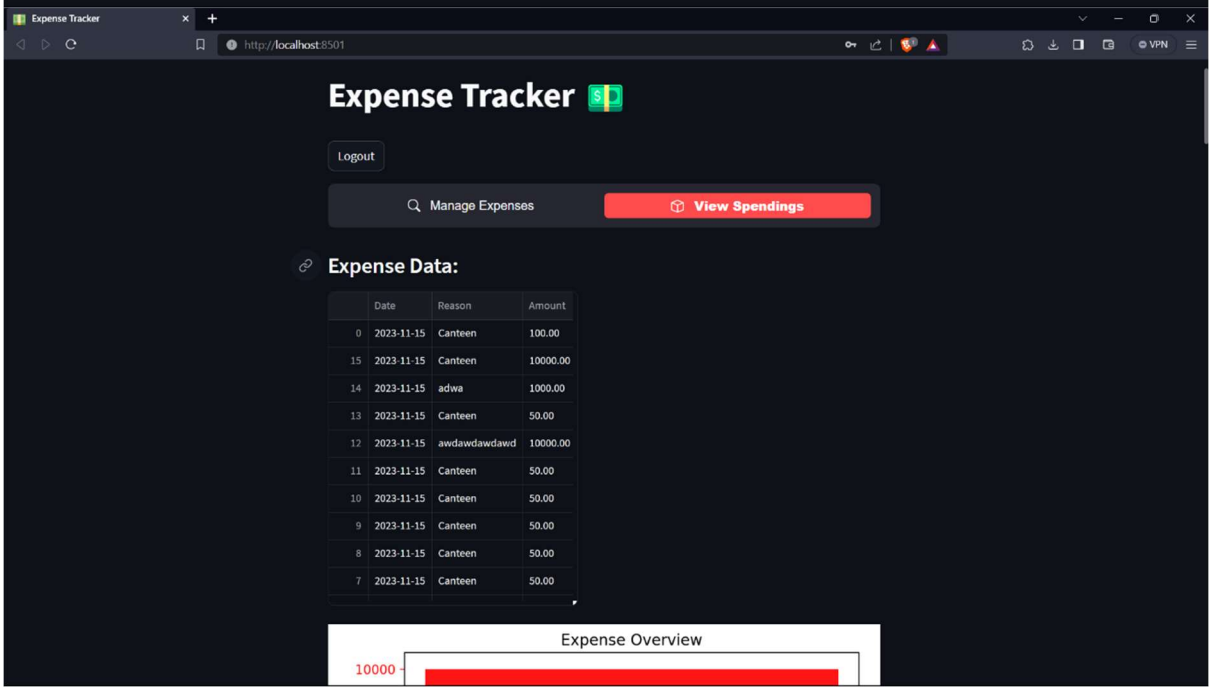

Expense Tracker
http://localhost:8501
VPN
Expense Tracker
Login
Signup
Login
Forgot Password
Login
Username
Password
Submit

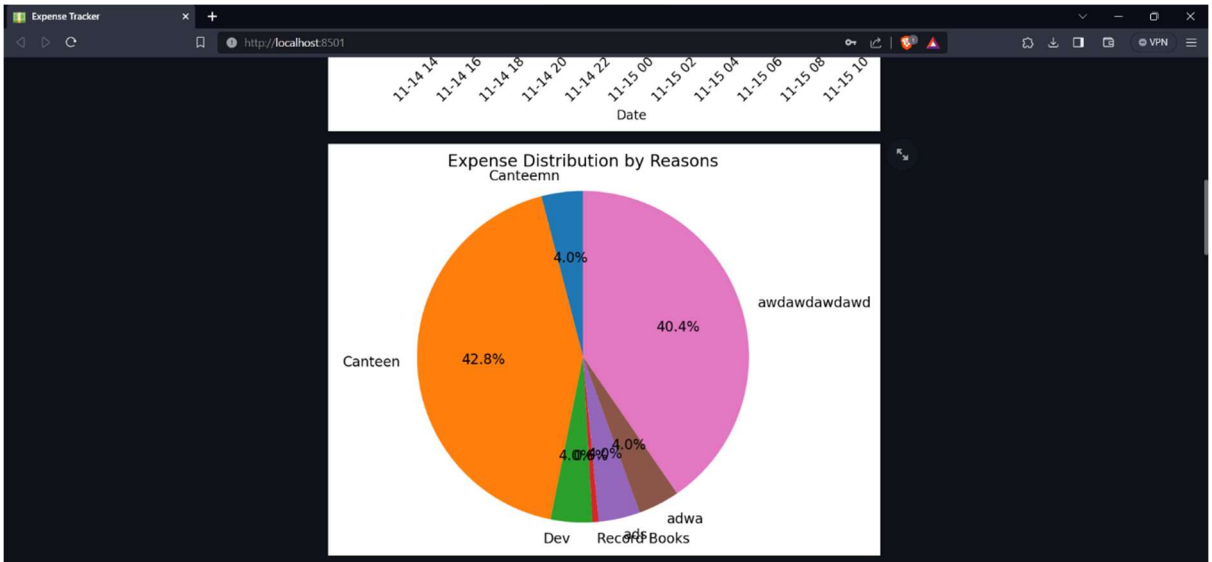

Expense Tracker
http://localhost:8501
VPN
Expense Tracker
Login
Signup
Sign Up
Enter your email
Enter your Mobile Number
Enter your username
Enter your password
Submit


Expense Tracker
http://localhost:8501
VPN
Expense Tracker
Logout
Manage Expenses
View Spendings
Budget Remaining : 99300.00
You are in the safe zone now
Expense
Revenue
Expense
Enter the date of the expense
2023/11/15
Enter the reason for spending
Canteen , Records etc.


Expense Tracker
http://localhost:8501
VPN
Budget Remaining : 99300.00
You are in the safe zone now
Expense
Revenue
Expense
Enter the date of the expense
2023/11/15
Enter the reason for spending
Canteen , Records etc.
Amount in Rupees?
1000 etc..
Submit

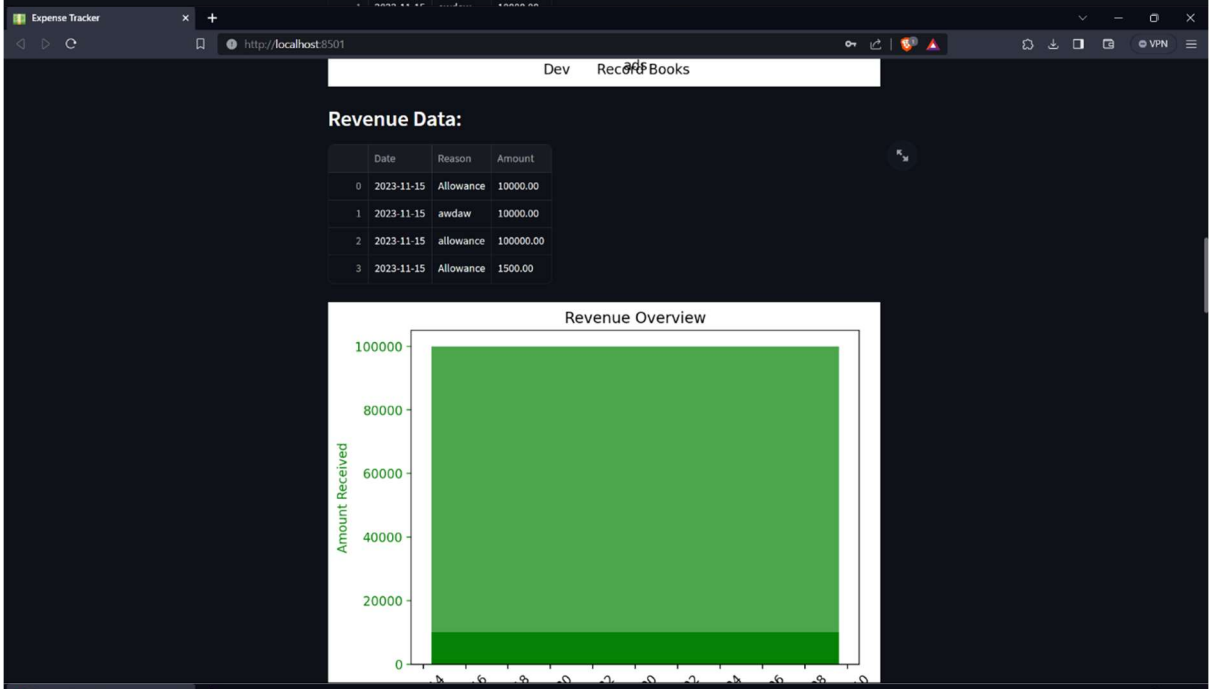

Expense Tracker
http://localhost:8501
VPN
Manage Expenses
View Spendings
Budget Remaining : 99300.00
You are in the safe zone now
Expense
Revenue
Revenue
Enter the date of the revenue
2023/11/15
Enter the source of revenue
Allowance , Pocket Money etc.
Amount in Rupees?
1000 etc..
Submit


Expense Tracker
http://localhost:8501
VPN
Expense Tracker
Logout
Manage Expenses
View Spendings
Expense Data:
Date
Reason
Amount
0 2023-11-15 Canteen 100.00
15 2023-11-15 Canteen 10000.00
14 2023-11-15 adwa 1000.00
13 2023-11-15 Canteen 50.00
12 2023-11-15 awdawdawdawd 10000.00
11 2023-11-15 Canteen 50.00
10 2023-11-15 Canteen 50.00
9 2023-11-15 Canteen 50.00
8 2023-11-15 Canteen 50.00
7 2023-11-15 Canteen 50.00
Expense Overview
10000



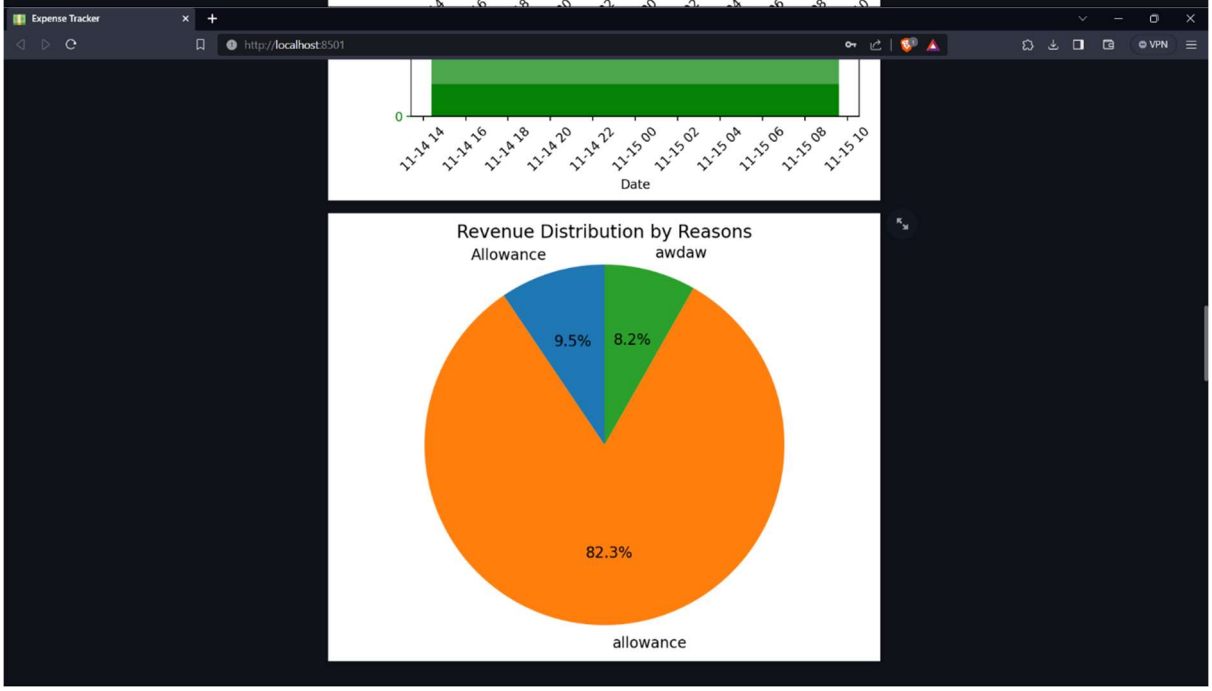
Revenue Data:

	Date	Reason	Amount
0	2023-11-15	Allowance	10000.00



Revenue Data:

	Date	Reason	Amount
0	2023-11-15	Allowance	10000.00
1	2023-11-15	awdaw	10000.00
2	2023-11-15	allowance	100000.00
3	2023-11-15	Allowance	1500.00

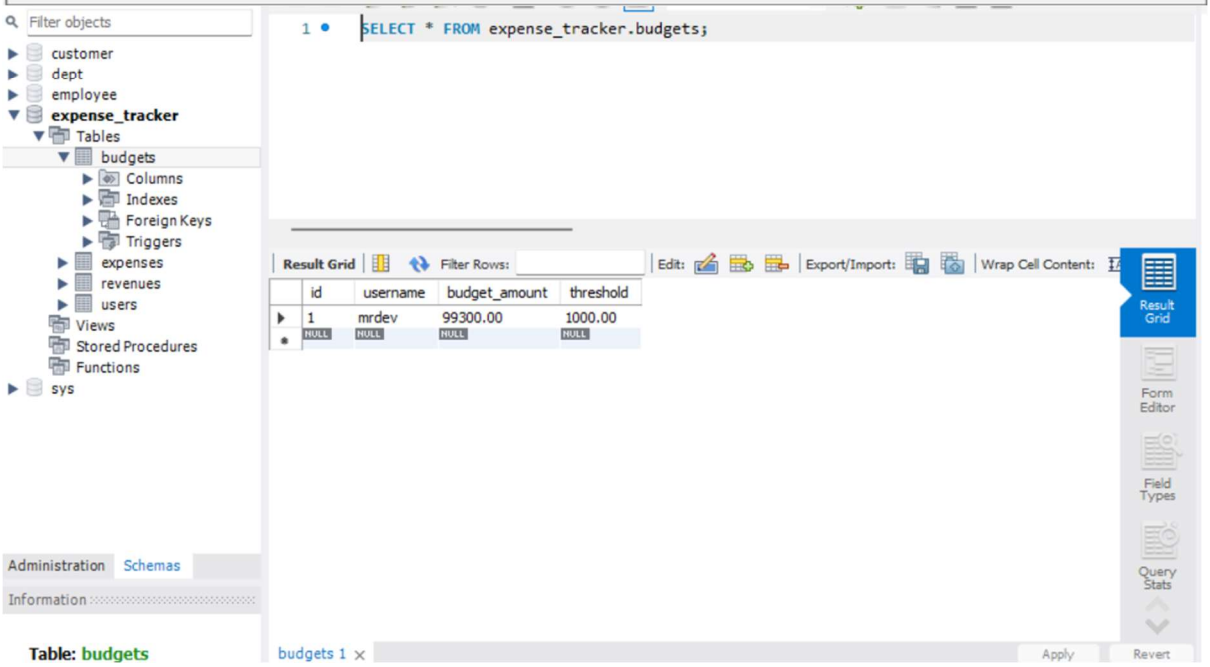


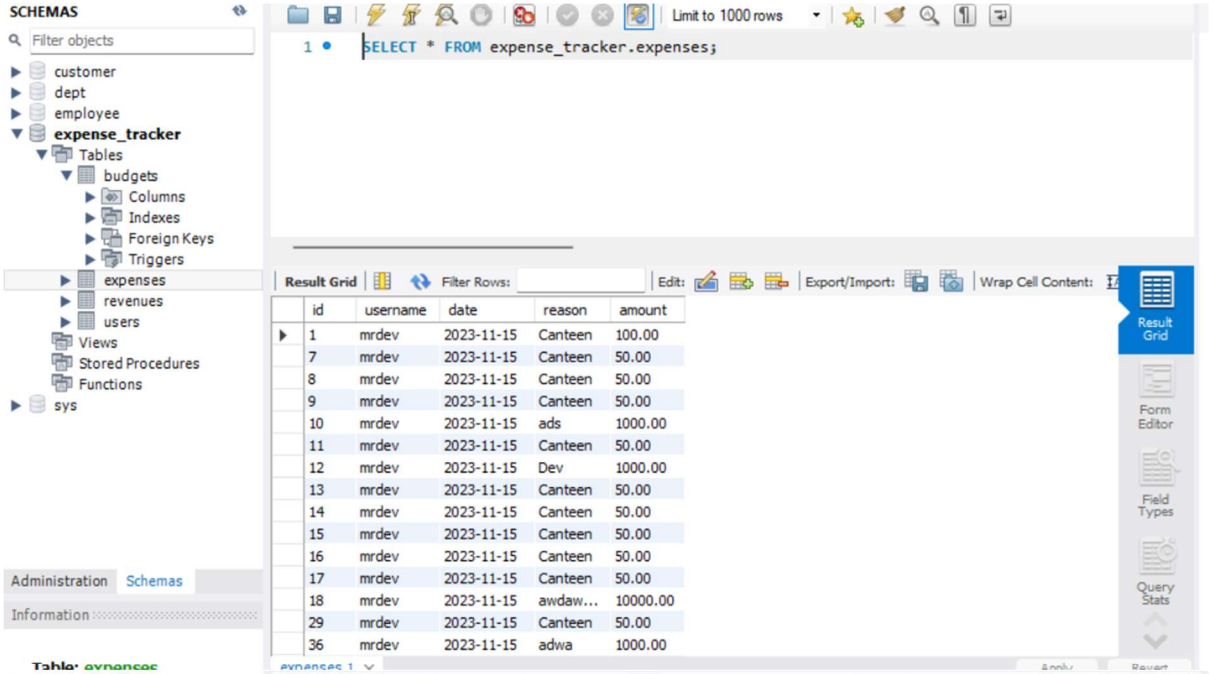
Name	Engine	Version	Row Format	Rows	Avg Row Length	Data Length	M
admins	InnoDB	10	Dynamic	0	0	16.0 KiB	
budgets	InnoDB	10	Dynamic	1	16384	16.0 KiB	
expenses	InnoDB	10	Dynamic	18	910	16.0 KiB	
revenues	InnoDB	10	Dynamic	4	4096	16.0 KiB	
users	InnoDB	10	Dynamic	2	8192	16.0 KiB	

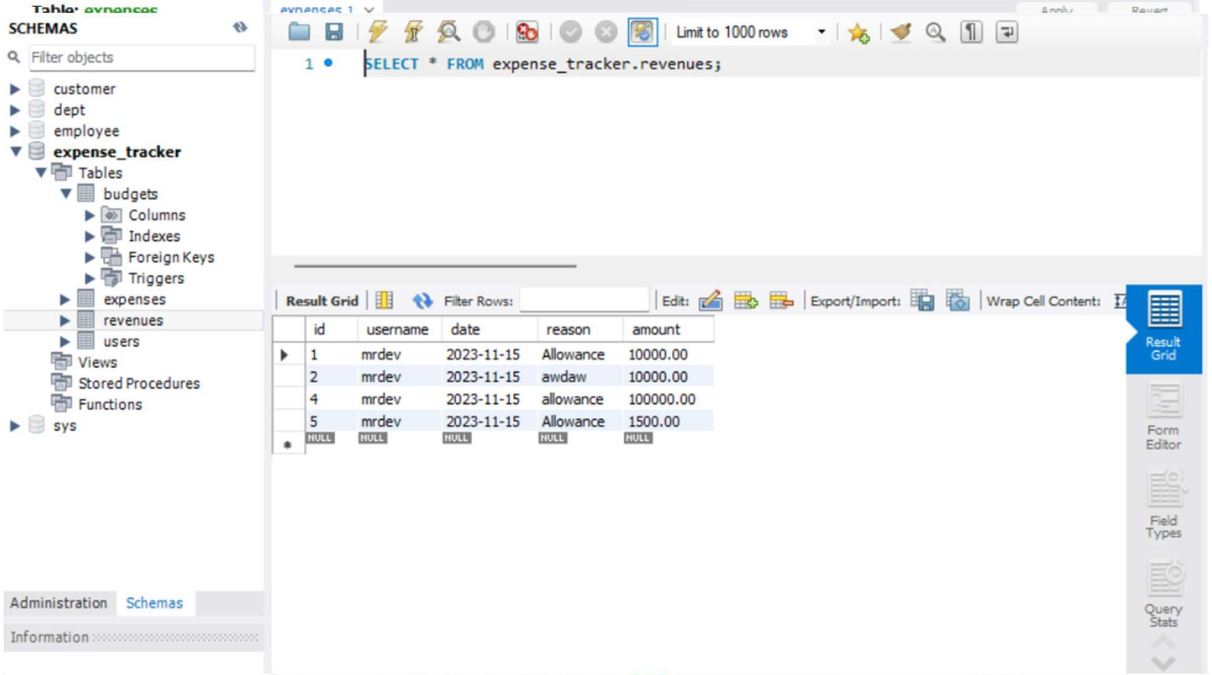
Count: 5 Maintenance > Inspect Table Refresh

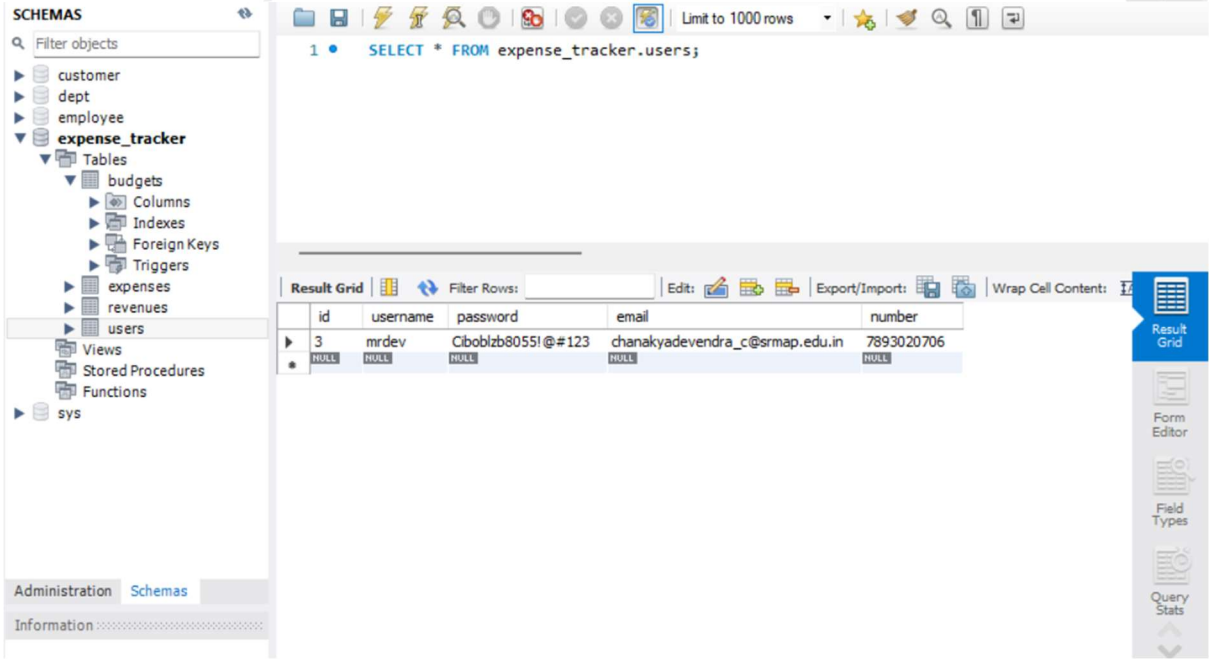
Table	Column	Type	Default Value	Nullable	Character Set	C
admins	email	varchar(255)		NO	utf8mb4	u
admins	id	int		NO		
admins	number	varchar(20)		NO	utf8mb4	u
admins	password	varchar(255)		NO	utf8mb4	u
admins	username	varchar(255)		NO	utf8mb4	u
budgets	budget_amount	decimal(10,2)		NO		
budgets	id	int		NO		
budgets	threshold	decimal(10,2)		NO		
budgets	username	varchar(255)		NO	utf8mb4	u
expenses	amount	decimal(10,2)		NO		
expenses	date	date		NO		
expenses	id	int		NO		
expenses	reason	varchar(255)		NO	utf8mb4	u
expenses	username	varchar(255)		NO	utf8mb4	u
revenues	amount	decimal(10,2)		NO		
revenues	date	date		NO		
revenues	id	int		NO		
revenues	reason	varchar(255)		NO	utf8mb4	u
revenues	username	varchar(255)		NO	utf8mb4	u
users	email	varchar(255)		NO	utf8mb4	u
users	id	int		NO		
users	number	varchar(20)		NO	utf8mb4	u
users	password	varchar(255)		NO	utf8mb4	u
users	username	varchar(255)		NO	utf8mb4	u

Table	Name	Unique	Index...	Index Comment
admins	PRIMARY	Yes	BTREE	
admins	username	Yes	BTREE	
admins	email	Yes	BTREE	
admins	number	Yes	BTREE	
budgets	PRIMARY	Yes	BTREE	
budgets	username	Yes	BTREE	
expenses	PRIMARY	Yes	BTREE	
revenues	PRIMARY	Yes	BTREE	
users	PRIMARY	Yes	BTREE	
users	username	Yes	BTREE	
users	email	Yes	BTREE	
users	number	Yes	BTREE	




SCHEMAS
Filter objects
customer
dept
employee
expense_tracker
Tables
budgets
Columns
Indexes
Foreign Keys
Triggers
expenses
revenues
users
Views
Stored Procedures
Functions
sys
Administration
Schemas
Information
Limit to 1000 rows
1 • SELECT * FROM expense_tracker.expenses;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
id username date reason amount
1 mrdev 2023-11-15 Canteen 100.00
7 mrdev 2023-11-15 Canteen 50.00
8 mrdev 2023-11-15 Canteen 50.00
9 mrdev 2023-11-15 Canteen 50.00
10 mrdev 2023-11-15 ads 1000.00
11 mrdev 2023-11-15 Canteen 50.00
12 mrdev 2023-11-15 Dev 1000.00
13 mrdev 2023-11-15 Canteen 50.00
14 mrdev 2023-11-15 Canteen 50.00
15 mrdev 2023-11-15 Canteen 50.00
16 mrdev 2023-11-15 Canteen 50.00
17 mrdev 2023-11-15 Canteen 50.00
18 mrdev 2023-11-15 awdaw... 10000.00
29 mrdev 2023-11-15 Canteen 50.00
36 mrdev 2023-11-15 adwa 1000.00
Result Grid
Form Editor
Field Types
Query Stats


SCHEMAS
Filter objects
customer
dept
employee
expense_tracker
Tables
budgets
Columns
Indexes
Foreign Keys
Triggers
expenses
revenues
users
Views
Stored Procedures
Functions
sys
Administration
Schemas
Information
Limit to 1000 rows
1 • SELECT * FROM expense_tracker.revenues;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
id username date reason amount
1 mrdev 2023-11-15 Allowance 10000.00
2 mrdev 2023-11-15 awdaw 10000.00
4 mrdev 2023-11-15 allowance 100000.00
5 mrdev 2023-11-15 Allowance 1500.00
NULL NULL NULL NULL NULL
Result Grid
Form Editor
Field Types
Query Stats


SCHEMAS
Filter objects
customer
dept
employee
expense_tracker
Tables
budgets
Columns
Indexes
Foreign Keys
Triggers
expenses
revenues
users
Views
Stored Procedures
Functions
sys
Administration
Schemas
Information
Limit to 1000 rows
1 • SELECT * FROM expense_tracker.users;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
id username password email number
3 mrdev Ciboblzb8055!@#123 chanakyadevendra_c@srmap.edu.in 7893020706
NULL NULL NULL NULL NULL
Result Grid
Form Editor
Field Types
Query Stats

Conclusion

The Expense Tracker project has successfully addressed the challenges associated with personal expense management by providing a robust and user-friendly solution. The system's architecture, characterized by a well-defined database schema and clear relationships between entities, ensures the effective storage and retrieval of user data.

The ER Diagram and Schema Diagram highlight the organization of the system's entities, such as users, budgets, expenses, and revenues. The use of primary and foreign keys establishes relationships, facilitating data integrity and enabling comprehensive insights into users' financial activities.

Throughout the development process, the project has achieved its objectives, including efficient expense tracking, budget management, user authentication, and data visualization. The frontend, developed using Streamlit, offers an intuitive interface for users to input and monitor their financial transactions. The backend, powered by MySQL, ensures secure storage and retrieval of user data, with a focus on password hashing and validation.

The notification system provides timely alerts to users, encouraging responsible financial habits. Additionally, the categorization of expenses and revenues, coupled with insightful data visualization, empowers users to make informed decisions about their spending patterns.

Looking ahead, the project is designed with scalability in mind, allowing for future enhancements and additional features to meet evolving user needs. The system's adaptability positions it as a valuable tool for individuals seeking to achieve and maintain financial discipline.

In conclusion, the Expense Tracker project has successfully delivered a user-centric, secure, and technologically advanced solution for personal finance management. By combining streamlined user interfaces with robust backend functionalities, the system empowers users to take control of their finances, make informed decisions, and work towards achieving their financial goals. The project stands as a testament to the effective integration of technology to address real-world challenges in personal finance.

Future Scope

The Expense Tracker project lays the foundation for future enhancements and features, ensuring its adaptability to evolving user needs. Some potential areas for future development and improvement include:

1. **Enhanced Data Analytics:**
    - Implement advanced data analytics features to provide users with deeper insights into their spending habits, trends, and areas for potential savings.
2. **Machine Learning Integration:**
    - Explore the integration of machine learning algorithms to analyze user spending patterns, offering personalized financial advice and proactive budget recommendations.
3. **Multi-User Support:**
    - Extend the system to support multiple users with individual accounts, allowing families or groups to manage their finances collaboratively while maintaining privacy and data security.
4. **Mobile Application Development:**
    - Develop a dedicated mobile application for Android and iOS platforms, enhancing accessibility and providing users with a seamless experience on their smartphones.
5. **Expense Receipt Upload:**
    - Integrate functionality for users to upload images or receipts of their expenses, allowing for more detailed documentation and analysis of transactions.
6. **Expense Categories Customization:**
    - Enable users to customize and create their own expense and revenue categories, providing a more personalized and adaptable tracking experience.
7. **Integration with Financial Institutions:**
    - Explore integrations with financial institutions or APIs to automate the import of transaction data, reducing manual input and improving accuracy.
8. **Budget Forecasting:**
    - Implement budget forecasting features, helping users anticipate future expenses and plan accordingly to achieve long-term financial goals.
9. **Expense Comparison and Benchmarking:**
    - Introduce features that allow users to compare their spending habits with similar demographic groups or financial benchmarks, providing additional context for financial decision-making.
10. **Security Enhancements:**
    - Continuously assess and enhance security measures to protect user data, including regular security audits, encryption upgrades, and adherence to the latest industry standards.
11. **User Feedback and Collaboration:**
    - Establish a feedback mechanism to gather user input and suggestions for further improvements, fostering a collaborative development environment based on user needs.

By incorporating these future enhancements, the Expense Tracker project can evolve into a comprehensive and dynamic personal finance management solution, staying relevant and valuable to users seeking efficient and intelligent financial tools. The flexibility of the current system architecture ensures that these features can be seamlessly integrated, enhancing the overall user experience and utility of the Expense Tracker.