PROJECT REPORT

Personal Expense Tracking System

1. INTRODUCTIO

1.1PROJECT OVERVIEW

In simple words, personal finance entails all the financial decisions and activities that a Finance app makes your life easier by helping you to manage your finances efficiently. A personal finance app will not only help you with budgeting and accounting but also give you helpful insights about money management. Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert.

1.2 PURPOSE

Expense tracking is an important part of creating a budget for your money management. Keeping a daily record of your expenses by tracking receipts, invoices and other outgoing expenses improves the financial health of your budget. Tracking expenses can help you stay on top of your cash flow and prepare you for tax season.

2. LITERATURE SURVEY

2.1.EXISTING PROBLEM

[1] THANAPAL: He proposed an expense tracker to prevent having to calculate income and expenses, as well as to remind someone to keep their expenses in track and also to add some details on how much money comes from other people and what expenses or payments the user have to make on a given date or month, User have categories in the expenditure tracker such as add expense, monthly expenses, add new expense, see categories of spending, export expenses in a date range, remove export files, and view expenses by category.

{2] CHANDINI: She proposed an expense tracker that will maintain all the expenses record of users and manage them efficiently. The user can choose an expense category and provide additional information such as a photo, a location, and the amount of the expense, among other things. This will save the information to the local database. The user can examine and sort expenses on a weekly, monthly, or annual basis. By utilising this, they reduced the quantity of manual calculations for their expenses and maintain track of their spending. The user can enter his income to compute histotal daily expenses, and the data will be saved for each individual user. This tracker could be useful for people who frequently go on trips or to the theatre with their buddies. This tracker will make it easier for them to disburse the bill. This will show the graph in the chosen view.

{3] KARIM: He proposed an expense tracker to create a system for recording expenses and income that is simple, quick, and easy to use. This project also includes features that will assist the user in maintaining all financial operations, such as a digital automatic diary. So, in order to create a better expense tracking system, they created a project that will greatly benefit the users. Most people are unable to track their expenses and income, resulting in financial difficulties. In this scenario, a daily cost tracker can assist people in tracking their income and expenses on a daily basis, allowing them to live a stress-free life.

2.2 REFERENCES.

[1] Bekaroo, G., & Sunhaloo, S. Intelligent Online Budget Tracker. [2] Underwood, D. (2011). A Case Study of Tracking Expenses by Commodity at Widget Farmers' Cooperative. [3] Chandini, S., Poojitha, T., Ranjith, D., Akram, V. M., Vani, M. S., & Rajyalakshmi, V. (2019). Online Income and Expense Tracker. [4] Satpute, M. K., Kale, A., Mandal, A., & Krishnan, R. SURVEY ON CLASSIFICATION ENGINE FOR MONETARY TRANSACTIONS [5] Sabab, S. A., Islam, S. S., Rana, M. J., & Hossain, M. (2018, September). eExpense: A smart approach to track everyday expense. In 2018

4th International Conference on Electrical Engineering and Information & Communication Technology (iCEEiCT) (pp. 136-141). IEEE. [6] Rajaprabha, M. N. (2017). Family Expense Manager Application in Android. MS&E, 263(4), 042050 [7] Kan, C., Lynch, J., & Fernbach, P. (2015). How budgeting helps consumers achieve financial goals. ACR North American Advances. [8] Sharma, R., 2020. Case Study Of Expense Tracking App: Get Daily Alerts Of Your Expense. [online] Medium. [9] Thanapal, M. P., Patel, Y., Lokesh, R. T. P., & Satheesh, K. J. (2015). Income and expense tracker. Indian Journal of Science and Technology, 8(S2), 118-122. [10] Manchanda, A. (2012). Expense Tracker Mobile Application (Doctoral dissertation, San Diego State University)

3. PROBLEM STATEMENT DEFINITION

At the instant, there is no as such complete solution present easily or we should say free of cost which enables a person to keep a track of its daily expenditure easily. To do so a person has to keep a log in a diary or in a computer, also all the calculations needs to be done by the user which may sometimes results in errors leading to losses. Due to lack of a complete tracking system, there is a constant overload to rely on the daily entry of the expenditure and total estimation till the end of the month. As the name itself suggests, this project is an attempt to manage our daily expenses in a more efficient and manageable way. The system attempts to free the user with as much as possible the burden of manual calculation and to keep the track of the expenditure. Instead of keeping a dairy or a log of the expenses on the smartphones or laptops, this system enables the user to not just help user with budgeting and accounting but also give them helpful insights about money management. In simple words, personal finance entails all the financial decisions and activities that a Finance app makes your life easier by helping you to manage your finances efficiently. Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert.

3. IDEATION & PROPOSED SOLUTION

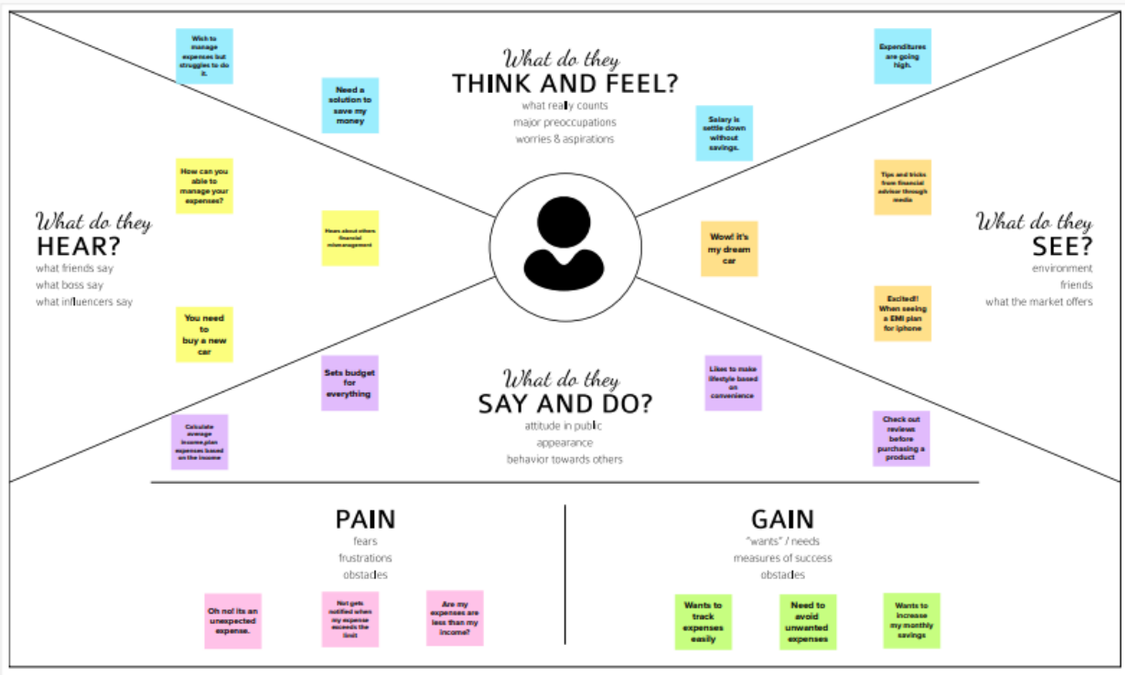
3.1.EMPATHY MAP CANVAS

Empathy Map Canvas

Gain insight and understanding on solving customer problems.

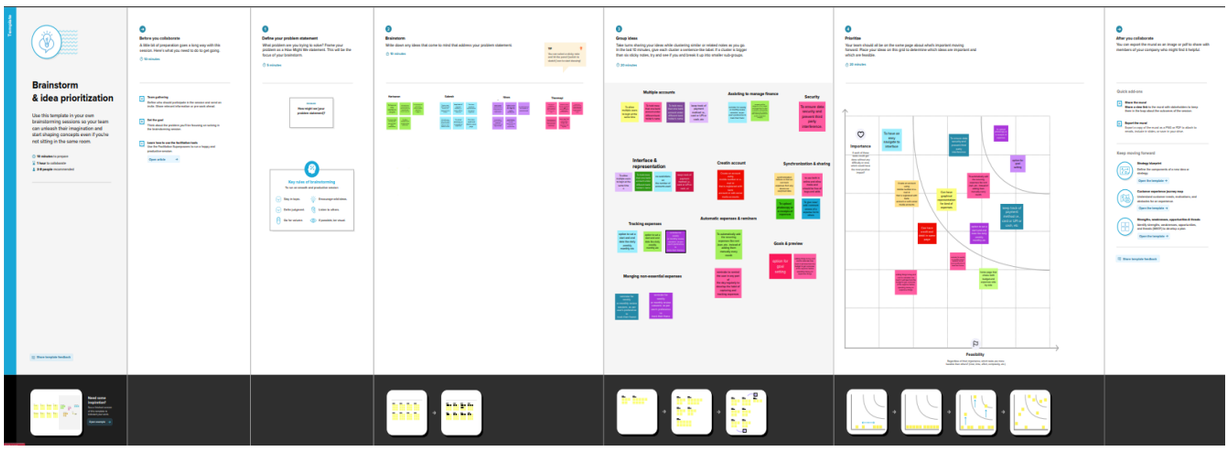
1

Build empathy and keep your focus on the user by putting yourself in their shoes.



Share your feedback

3.2.IDEATION AND BRAINSTORMING



3.3.PROBLEM SOLUTION FIT

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?

Predominantly Engineers who are just starting to earn and manage their personal finance. Typically from middle and lower class family, who badly need financial discipline .

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions?

The impulse buying and lacking to awareness to look into bigger picture

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem

Totally shunning to spend even on necessities under the impression that the spending could result in bad financial position.

The existing solutions are otherwise over complicated and designed to extract data from user.

Manual physical logging in time consuming

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

- Logging expenses into categories
- Show historical stats
- Generate insightful charts
- Alert user to imbibe good discipline

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?

Lack of proper education in financial literacy in school education. More children are not given pocket money to learn by spending/wasting less / saving.

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?

Get frustrated and fall into debt traps by taking unpayable loans for unnecessary items leading to increase in mental stress

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act?

Frequent sales in e-commerce platforms and seamless shopping experience online.

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?

Dejected and paranoid about the future as they would need relatively more money to provide for a family and to handle unexpected financial needs.

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.

If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behavior.

Graphical Application with simple UI and to the point clutter free objective. Avoids provision to pay through the app, to minimize the spending and ensure that only necessary spendings are made. The aim is to make the spending process harder throughout the application and keep it clean.

8.CHANNELS of BEHAVIOUR CH

8.1 ONLINE

What kind of actions do customers take online? Extract online channels from #7

1. Shop from e-commerce
2. Subscribe to OTT platforms
3. Order food frequently

8.2 OFFLINE

What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

1. Shop in malls during sales
2. Keep the money somewhere around and forget about /lose it

Identify strong TR & EM

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Email/SignUp Registration through Gmail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Add expenses	Enter the everyday expenses Split it into categories(example : food, petroI,movies)
FR-4	Reminder mail	Sending reminder mail on target (for ex : if user wants a reminder when his/her balance reaches some amount(S000)) Sending reminder mail to the user if he/she has not filled that day's expenses.
FR-S	Creating Graphs	Graphs showing everyday and weekly expenses. Categorical graphs on expenditure.
FR-6	Add salary	Users must enter the salary at the start of the month.
FR-7	Export CSV	User can export the raw data of their expenditure as CSV

4. REQUIREMENT ANALYSIS

4.1Functional Requirements And Non Functional Requirements

FR No.	Non-Functional **Requirement**	**Description**
NFR-1	Usability	A simple web application which is accessible across devices
NFR-2	**Security**	The OAuth Google sign in and email login are secure with hashed and salted secure storage of credentials.
NFR-3	Reliability	Containerized service ensures that new instance can kick up when there is a failure
NFR-4	Performance	The load is managed through the load balancer used with docker. Thus ensuring good performance
NFR-5	Availability	With load balancing and multiple container instances, the service is always available.
NFR-6	Scalability	Docker and Kubernetes are designed to accommodate scaling based on need

5.PROJECT DESIGN

1. DATA FLOW DIAGRAMS

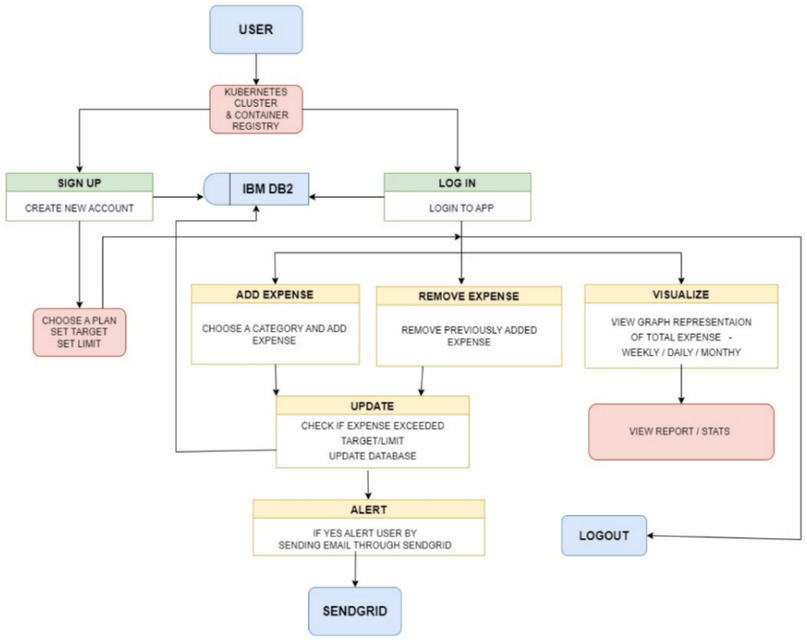
Data Flow Diagrams:

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

The general flow goes like this:

- The user can create a signup or Log in from Kubernetes Cluster and container registry.
- After logging in the user can perform any action like adding or removing expense.
- The user can view their expense in in the form of graphs as the data entered will be visualized.
- After this operation the expense will be updated and then the app will check if expense exceeded target or limit .
- Then the user can logout.

Data Flow Diagram of Personal Expense Tracker: DFD Level 2



2. SOLUTION AND TECHNICAL ARCHITECTURE

Solution Architecture:

Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript in Python Flask
2.	User Login	The user can login either through their gmail account or an account in the app server	Google Oauth for Google Signin. Hashed password in DB
3.	Graph Visualisation	Rendering plots and graphs based on the user spending data	Seaborn, Mathplotlib
4.	Database	Data Type, Configurations etc.	NoSQL database can be used as it promotes flexible structuring of data
5.	Cloud Database	Database Service on Cloud	IBM DB2 is used to store the user details and the data entries
6.	SendGrid	a cloud-based SMTP provider that allows you to send email without having to maintain email servers	SendGrid is used to trigger mail to user emails when a particular condition is met
7.	Google OAuth	OAuth 2.0 allows users to share specific data with an application while keeping their usernames, passwords, and other information private.	Enables login through gmail account, thus making the application accessible
8.	Cloud Deployment	Application Deployment onCloud Server	Docker and Kubernetes is used for deployment as it promises scalability and high availability

Table-2: Application Characteristics:

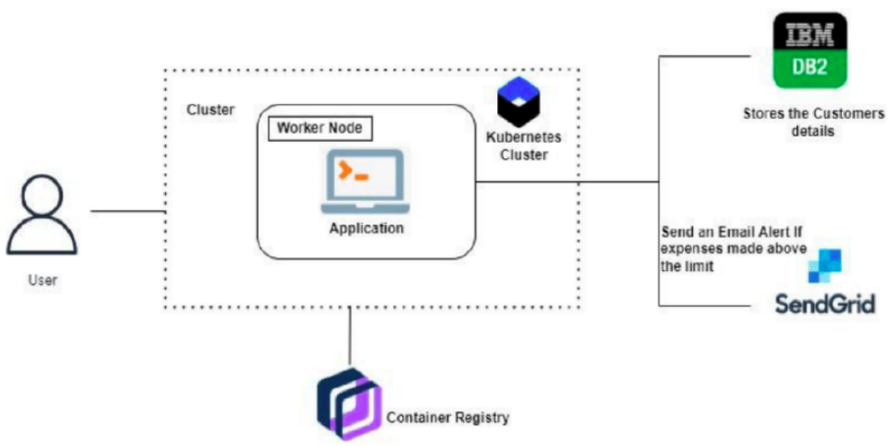
S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.	Python Flask Framework
2.	Security Implementations	Passwords cant be stored as plaintext so it is hashed and salted	BCrypt
3.	Scalable Architecture	Containerized application is deployed to rapidly increase scale on demand	Docker

S.No	Characteristics	Description	Technology
4.	Availability, Performance	Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management. Availability and Performance enhances user experience	Kubernetes

TECHNICAL ARCHITECTURE

Technical Architecture:

The Deliverable shall include the architectural diagram as below and the information as per the table1 & table 2



3. User Stories

User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.
	Login	USN-2	As a user, I can log into the application by entering email & password
	Add	USN -3	As a user , I can add in new expenses.
	Remove	USN – 4	As a user , I can remove previously added expenses.
	View	USN - 5	As a user , I can view my expenses in the form of graphs and get insights.
	Get alert message	USN - 6	As a user , I will get alert messages if I exceed my target amount.
Administrator	Add / remove user	USN – 7	As admin , I can add or remove user details on db2 manually.
		USN - 8	As admin , I can add or remove user details on sendgrid.

6. PROJECT PLANNING & SCHEDULING

1. Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Harisaran Sabesh
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Harisaran Sabesh
Sprint-1	Login	USN-3	As a user, I can register for the application through Facebook	2	Low	Thanmayi Nivas
Sprint-1	Dashboard	USN-4	Logging in takes the user to their dashboard.	1	Medium	Thanmayi Nivas
Sprint-2		USN-5	As a user ,I will update my salary at the start of each month..	1	High	Thanmayi Nivas
Sprint-2		USN-6	As a user , I will set a target/limit to keep track of my expenditure.	1	Medium	Nivas
Sprint-2	Workspace	USN-7	Workplace for personal expense tracking.	1	Medium	Thanmayi
Sprint-2	Charts	USN-8	Analytics to show weekly and everyday expenditure	2	Medium	Harisaran
Sprint-2		USN-9	As a user , I can export raw data as csv file	1	High	Harisaran
Sprint-3	IBM DB2	USN-10	Linking database with dashboard	2	Medium	Thanmayi
Sprint-3		USN-11	Making dashboard interactive with JS	2	Medium	Thanmayi

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-3	Watson Assistant	USN-12	Embedding Chatbot to clarify user's queries.	2	Medium	Nivas
Sprint-3	BCrypt	USN-13	Using BCrypt to store passwords securely.	2	High	Nivas
Sprint-3	SendGrid	USN-14	Using SendGrid to send mail to the user. (To alert or remind)	1	High	Sabesh
Sprint-4		USN-15	Integrating frontend and backend.	3	High	Sabesh
Sprint-4	Docker	USN-16	Creating Docker image of web app.	1	Medium	Harisaran
Sprint-4	Cloud Registry	USN-17	Uploading docker image to IBM cloud registry.	2	Medium	Harisaran
Sprint-4	Kubernetes	USN-18	Creating a container using docker and hosting the webapp.	2	Medium	Thanmayi Nivas
Sprint-4	Deployment	USN-19	Exposing IP/Ports for the site.	2	High	Sabesh

2. Sprint Delivery Schedule

Phase 1

- Register for the application by entering email , new password and confirming the same password
- Receive confirmation email once I have registered for the application.
- Log into the application by entering email and password

Phase 2

- Set a target/limit to keep track of expenditure
- Track the debits and credits made by the user
- Analytics to show weekly and everyday expenditure

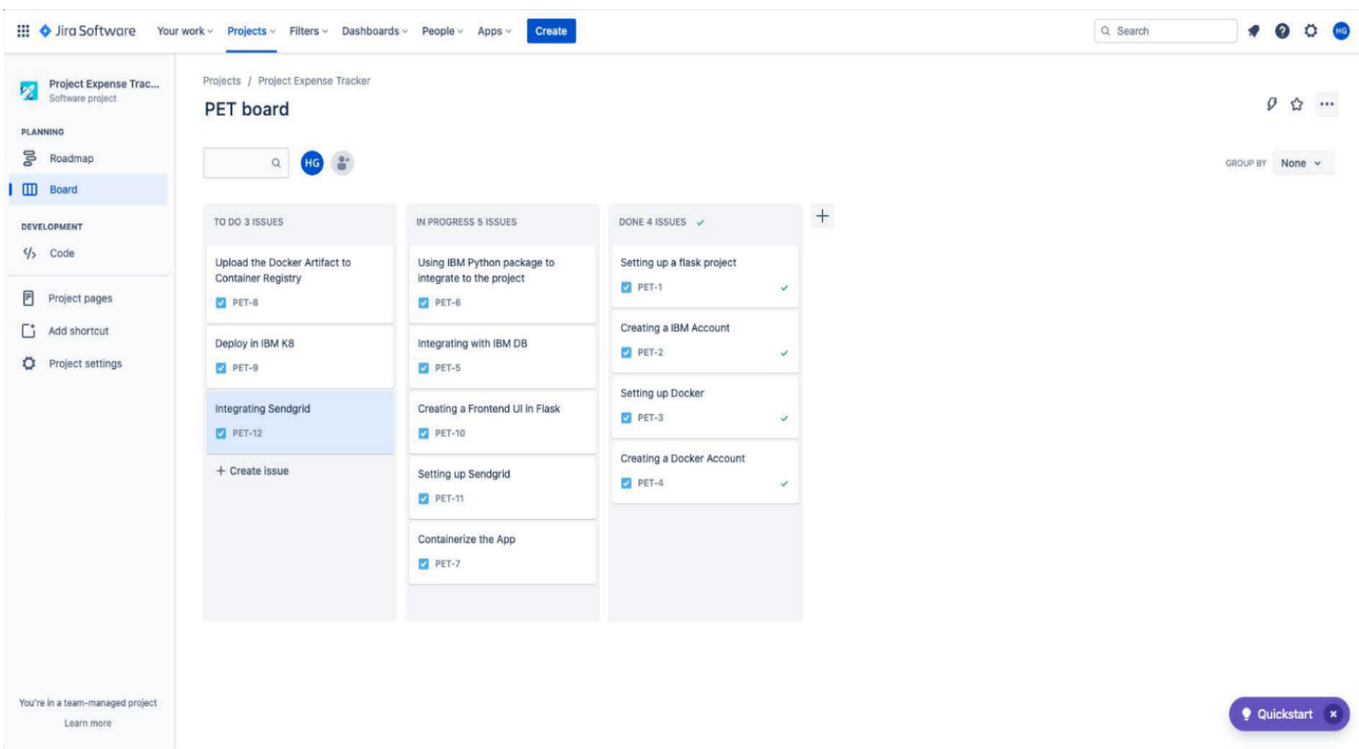
Phase 3

- Linking database with dashboard
- Chatbot linking to clear user queries
- Secure the passwords
- Reminder alerts to user via email

Phase 4

- Intergrate the IBM DB2, Sendgrid and other services
- Containerize the app and use IBM cloud to host the web app

6.3Reports from JIRA



7. CODIND AND SOLUTIONING

Feature 1

```
from flask import Flask, render_template, request, redirect, url_for
from flask_mail import Mail, Message
from datetime import datetime
from flask_cors import CORS, cross_origin
import ibm_db
import json
import plotly
import plotly.graph_objs as go
import pandas as pd
from flask import send_file
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np
import base64
from PIL import Image
import time
import atexit
from datetime import datetime
from apscheduler.schedulers.background import BackgroundScheduler

app = Flask(__name__, template_folder='templates')
app.config['SECRET_KEY'] = 'top-secret!'
app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_PASSWORD'] = 'SG.rRPqo3ZyRhWUD6RhljE1CA.894zN6QMM9UjOpgPlO-4KT-_mjT9-
KwXZ9ArygkEnis'
app.config['MAIL_DEFAULT_SENDER'] = 'nunnaaarthi@gmail.com'
mail = Mail(app)
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

# GLobal variables
EMAIL = ''
USERID = ''

conn =
ibm_db.connect("DATABASE=bludb;HOSTNAME=54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases
.appdomain.cloud;PORT=32733;Security=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=nlg66799;PWD=CXtQ
LAGZ06fD0fhC;", "", "")

# FUNCTIONS INTERACTING WITH DB #


def fetch_walletamount():
    sql = 'SELECT WALLET FROM PETA_USER WHERE EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, EMAIL)
    ibm_db.execute(stmt)
    user = ibm_db.fetch_assoc(stmt)
    # print(user['WALLET'])
    return user['WALLET']  # returns int


def fetch_categories():

    sql = 'SELECT * FROM PETA_CATEGORY WHERE USERID = ?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, USERID)
    ibm_db.execute(stmt)

    categories = []
    while ibm_db.fetch_row(stmt) != False:
        categories.append([ibm_db.result(stmt, "CATEGORYID"),
                           ibm_db.result(stmt, "CATEGORY_NAME")])

    sql = 'SELECT * FROM PETA_CATEGORY WHERE USERID IS NULL'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)

    while ibm_db.fetch_row(stmt) != False:
        categories.append([ibm_db.result(stmt, "CATEGORYID"),
                           ibm_db.result(stmt, "CATEGORY_NAME")])

    # print(categories)
    return categories  # returns list
```

```
def fetch_userID():
    sql = 'SELECT USERID FROM PETA_USER WHERE EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, EMAIL)
    ibm_db.execute(stmt)
    user = ibm_db.fetch_assoc(stmt)
    # print(user['USERID'])
    return user['USERID']  # returns int


def fetch_groups():
    sql = 'SELECT * FROM PETA_GROUPS'
    stmt = ibm_db.exec_immediate(conn, sql)
    groups = []
    while ibm_db.fetch_row(stmt) != False:
        groups.append([ibm_db.result(stmt, "GROUPID"),
                       ibm_db.result(stmt, "GROUPNAME")])
    # print(groups)
    return groups  # returns list


def fetch_expenses():
    sql = 'SELECT * FROM PETA_EXPENSE where USERID = ' + str(USERID)
    # print(sql)
    stmt = ibm_db.exec_immediate(conn, sql)
    expenses = []
    while ibm_db.fetch_row(stmt) != False:
        category_id = ibm_db.result(stmt, "CATEGORYID")
        category_id = str(category_id)
        sql2 = "SELECT * FROM PETA_CATEGORY WHERE CATEGORYID = " + category_id
        stmt2 = ibm_db.exec_immediate(conn, sql2)
        category_name = ""
        while ibm_db.fetch_row(stmt2) != False:
            category_name = ibm_db.result(stmt2, "CATEGORY_NAME")
        expenses.append([ibm_db.result(stmt, "EXPENSE_AMOUNT"), ibm_db.result(
            stmt, "DATE"), ibm_db.result(stmt, "DESCRIPTION"), category_name])
    # print(expenses)
    return expenses


def fetch_rec_expenses_cron():
    sql = 'SELECT * FROM PETA_REC_EXPENSES;'
    stmt = ibm_db.exec_immediate(conn, sql)
    rec_expenses = []
    while ibm_db.fetch_row(stmt) != False:
        amt = ibm_db.result(stmt, "AMOUNT")
        amt = str(amt)
        description = ibm_db.result(stmt, "DESCRIPTION")
        userid = ibm_db.result(stmt, "USERID")
        date = ibm_db.result(stmt, "RECDATE")
        rec_expenses.append([amt, description, date, userid])
    # print(rec_expenses)
    return rec_expenses


def fetch_rec_expenses():
    sql = 'SELECT * FROM PETA_REC_EXPENSES WHERE USERID = ' + str(USERID)
    stmt = ibm_db.exec_immediate(conn, sql)

    rec_expenses = []
    while ibm_db.fetch_row(stmt) != False:
        amt = ibm_db.result(stmt, "AMOUNT")
        amt = str(amt)
        description = ibm_db.result(stmt, "DESCRIPTION")
        userid = ibm_db.result(stmt, "USERID")
        date = ibm_db.result(stmt, "RECDATE")
        rec_expenses.append([amt, description, date, userid])
    # print(rec_expenses)
    return rec_expenses


def fetch_limits():
    now = datetime.now()
    year = now.year

    limits = [0 for i in range(12)]

    sql = 'SELECT LIMITAMOUNT, LIMITMONTH FROM PETA_LIMIT WHERE USERID = ? AND LIMITYEAR = ?'
    statement = execute_sql(sql, USERID, year)

    while ibm_db.fetch_row(statement):
        limit_amount = int(ibm_db.result(statement, 'LIMITAMOUNT'))
        limit_month = int(ibm_db.result(statement, 'LIMITMONTH'))
```

```
        limits[limit_month] = limit_amount

    return limits


# HELPER FUNCTIONS #
def fetch_latest_expenses(expenses):
    # must return expenses of last month
    latest_month = datetime.today().month
    latest_expenses = []
    for exp in expenses:
        if exp[1].month == latest_month:
            latest_expenses.append(exp)

    return latest_expenses


def fetch_monthly_expenses(expenses):
    latest_year = datetime.today().year
    monthly_expenses = {}

    for month in range(1, 13):
        monthly_expenses[month] = 0

    for exp in expenses:
        if exp[1].year == latest_year:
            monthly_expenses[exp[1].month] += exp[0]

    return monthly_expenses.values()


def draw_graph1(expenses):
    # TOTAL EXPENSE / DAY OF MONTH
    # x-axis: day , y-axis: expense/day

    latest_expenses = fetch_latest_expenses(expenses)
    mp = {}
    for day in range(1, 31):
        mp[day] = 0

    for exp in latest_expenses:
        mp[exp[1].day] += exp[0]

    x = mp.keys()
    y = mp.values()

    # print(mp)

    plt.figure()
    plt.title('Expense recorded over the past month')
    plt.plot(x, y)
    plt.xlabel('Day of the month')
    plt.ylabel('Recorded expense')
    plt.xlim(1, 32)

    buffer = BytesIO()
    plt.savefig(buffer, format='png')

    encoded_img_data = base64.b64encode(buffer.getvalue())

    return encoded_img_data


def draw_graph2(expenses, limits):
    # limit/month vs expense/month -> 2 line graphs

    monthly_expenses = fetch_monthly_expenses(expenses)
    x = range(1, 13)
    y1 = limits
    y2 = monthly_expenses

    plt.figure()
    plt.title('Month-wise comparison of limit and expense')
    plt.plot(x, y1, label="Limit/month")
    plt.plot(x, y2, label="Expenses/month")
    plt.xlabel('Month')
    plt.legend()

    buffer = BytesIO()
    plt.savefig(buffer, format='png')

    encoded_img_data = base64.b64encode(buffer.getvalue())

    return encoded_img_data
```

```
# finds the category id that matches that of the recurring expense category


def fetch_recurring_category_id():
    categories = fetch_categories()
    for category in categories:
        p = ''
        for i in category[1]:
            if (i == ' '):
                break
            p += i
        category[1] = p
    print(categories)
    categoryid = -1
    for category in categories:
        if category[1] == 'recurring':
            categoryid = category[0]
    print(categoryid)
    return categoryid


# cron to autodeduct the expenses each day
def auto_renew():
    global USERID
    # print(time.strftime("%A, %d. %B %Y %I:%M:%S %p"))
    rec_expenses = fetch_rec_expenses_cron()
    print(rec_expenses)
    current_day = time.strftime("%d")
    print(current_day)
    for expense in rec_expenses:
        here = str(expense[2])
        here = here.split('-')
        here = here[2]
        print(here)
        if (here == current_day):
            sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, DESCRIPTION, DATE)
VALUES(?,?,?,?,?);"
            USERID = str(expense[3])
            categoryid = fetch_recurring_category_id()
            print(categoryid)
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, expense[3])
            ibm_db.bind_param(stmt, 2, expense[0])
            ibm_db.bind_param(stmt, 3, categoryid)
            ibm_db.bind_param(stmt, 4, expense[1])
            d3 = time.strftime("%Y-%m-%d")
            ibm_db.bind_param(stmt, 5, d3)
            print(d3, categoryid, expense[0],
                  expense[1], expense[2], expense[3])
            ibm_db.execute(stmt)
            # print(here, d3, expense[0], expense[1], expense[2])
            sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?"
            statement = ibm_db.prepare(conn, sql)
            print(USERID)
            ibm_db.bind_param(statement, 1, expense[0])
            ibm_db.bind_param(statement, 2, expense[3])
            print("deducted")
            ibm_db.execute(statement)


# caller code for the cron
scheduler = BackgroundScheduler()
scheduler.add_job(func=auto_renew, trigger="interval", seconds=3600 * 24)
scheduler.start()

atexit.register(lambda: scheduler.shutdown())


# END POINTS #
@app.route('/', methods=['GET', 'POST'])
@cross_origin()
def registration():
    global EMAIL
    print("hello")
    if request.method == 'GET':
        return render_template('registration.html')
    if request.method == 'POST':
        email = request.form['email']
        EMAIL = email
        password = request.form['password']
        wallet = request.form['wallet']
        sql = "INSERT INTO PETA_USER(EMAIL,PASSWORD,WALLET) VALUES(?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
```

```
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.bind_param(stmt, 3, wallet)
        print(stmt)
        ibm_db.execute(stmt)
        # msg = Message('Registration Verfication',recipients=[email])
        # msg.body = ('Congratulations! Welcome user!')
        # msg.html = ('<h1>Registration Verfication</h1>'
        #             '<p>Congratulations! Welcome user!'
        #             '<b>PETA</b>!</p>')
        # mail.send(msg)
        EMAIL = email
    return redirect(url_for('dashboard'))


@app.route('/login', methods=['GET', 'POST'])
def login():
    global EMAIL
    print("login")
    if request.method == 'POST':
        email = request.form['email']
        EMAIL = email
        print(EMAIL)
        password = request.form['password']
        sql = "SELECT * FROM PETA_USER WHERE email=? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            return redirect(url_for('dashboard'))
        else:
            return redirect(url_for('login'))
    elif request.method == 'GET':
        return render_template('login.html')


@app.route('/dashboard', methods=['GET'])
def dashboard():
    global USERID
    global EMAIL
    print("dashboard")
    if USERID == '' and EMAIL == '':
        print("null email")
        return render_template('login.html')
    elif USERID == '':
        USERID = fetch_userID()
        print(USERID)
    expenses = fetch_expenses()
    wallet = fetch_walletamount()
    return render_template('dashboard.html', expenses=expenses, wallet=wallet, email=EMAIL)


@app.route('/updatebalance', methods=['GET', 'POST'])
def update_balance():
    if request.method == 'GET':
        wallet = fetch_walletamount()
        return render_template('updatebalance.html', wallet=wallet)
    elif request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()

        new_balance = request.form['balanceupdated']
        sql = 'UPDATE PETA_USER SET WALLET = ? WHERE USERID = ?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, new_balance)
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/addcategory', methods=['GET', 'POST'])
def add_category():
    if request.method == 'GET':
        # categories = fetch_categories()
        return render_template('addcategory.html')
```

```
    elif request.method == 'POST':
        categoryname = request.form['category']
        sql = 'INSERT INTO PETA_CATEGORY(CATEGORY_NAME, USERID) VALUES(?,?)'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, categoryname)
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/addgroup', methods=['POST'])
def add_group():
    if request.method == 'POST':
        if USERID == '':
            return render_template('login.html', msg='Login before proceeding')
        sql = "INSERT INTO PETA_GROUPS(GROUPNAME, USERID) VALUES(?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, request.form['groupname'])
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        group_info = {}

        sql = "SELECT * FROM PETA_GROUPS WHERE GROUPNAME=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, request.form['groupname'])
        ibm_db.execute(stmt)
        group_info = ibm_db.fetch_assoc(stmt)
        return {"groupID": group_info['GROUPID'], 'groupname': group_info['GROUPNAME']}


@app.route('/addexpense', methods=['GET', 'POST'])
def add_expense():
    if request.method == 'GET':
        groups = fetch_groups()
        categories = fetch_categories()
        if len(categories) == 0:
            return redirect(url_for('add_category'))
        return render_template('addexpense.html', categories=categories, groups=groups)

    elif request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()

        amount_spent = request.form['amountspent']
        category_id = request.form.get('category')
        description = request.form['description']
        date = request.form['date']
        groupid = request.form.get('group')
        # print(amount_spent, category_id, description, date, groupid, USERID)

        sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, GROUPID, DESCRIPTION,
DATE) VALUES(?,?,?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, amount_spent)
        ibm_db.bind_param(stmt, 3, category_id)
        ibm_db.bind_param(stmt, 4, groupid)
        ibm_db.bind_param(stmt, 5, description)
        ibm_db.bind_param(stmt, 6, date)
        ibm_db.execute(stmt)
        print(date, amount_spent, category_id)
        sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?"
        statement = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(statement, 1, amount_spent)
        ibm_db.bind_param(statement, 2, USERID)
        ibm_db.execute(statement)

        return redirect(url_for('dashboard'))


@app.route('/viewrecurring', methods=['GET'])
def viewrecurring():
    global USERID
    global EMAIL
    print("viewrecurring")
    if USERID == '' and EMAIL == '':
        print("null email")
```

```
        return render_template('login.html')
    elif USERID == '':
        USERID = fetch_userID()
        # print(USERID)
    print(USERID)
    expenses = fetch_rec_expenses()
    wallet = fetch_walletamount()
    return render_template('viewrecurring.html', expenses=expenses, wallet=wallet, email=EMAIL)


@app.route('/recurringexpense', methods=['GET', 'POST'])
def recurring_expense():
    global USERID, EMAIL
    if request.method == 'GET':
        groups = fetch_groups()
        categories = fetch_categories()
        if len(categories) == 0:
            return redirect(url_for('add_category'))
        USERID = fetch_userID()
        # check if user has added a category for recurring category, if not redirect and ask her to
        recur_id = fetch_recurring_category_id()
        if (recur_id == -1):
            return (redirect(url_for('add_category')))
        return render_template('recurringexpense.html', categories=categories, groups=groups)

    elif request.method == 'POST':
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()
            # check if user has added a category for recurring category, if not redirect and ask her to
            recur_id = fetch_recurring_category_id()
            if (recur_id == -1):
                return (redirect(url_for('add_category')))
        amount_spent = request.form['amountspent']
        category_id = request.form.get('category')
        description = request.form['description']
        date = request.form['date']
        # months = request.form['autorenewals']
        # groupid = request.form.get('group')
        print("recurring : ")
        print(amount_spent, description, date, USERID)

        sql = "INSERT INTO PETA_REC_EXPENSES(AMOUNT, RECDATE, USERID, DESCRIPTION) VALUES (?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, amount_spent)
        ibm_db.bind_param(stmt, 2, date)
        ibm_db.bind_param(stmt, 3, USERID)
        ibm_db.bind_param(stmt, 4, description)
        ibm_db.execute(stmt)

        sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, DESCRIPTION, DATE)
VALUES(?,?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, amount_spent)
        ibm_db.bind_param(stmt, 3, category_id)
        ibm_db.bind_param(stmt, 4, description)
        ibm_db.bind_param(stmt, 5, date)
        ibm_db.execute(stmt)

        sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?;"
        statement = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(statement, 1, amount_spent)
        ibm_db.bind_param(statement, 2, USERID)
        ibm_db.execute(statement)

        return redirect(url_for('dashboard'))


@app.route('/removerecurring', methods=['POST'])
def remove_recurring():
    print("remove recurring")
    if request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()
        description = request.form['description']
        print(description, USERID)
```

```
        sql = 'DELETE FROM PETA_REC_EXPENSES WHERE USERID = ? AND DESCRIPTION = ?;'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, description)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/analysis', methods=['GET', 'POST'])
def analyse():
    if request.method == 'GET':
        expenses = fetch_expenses()
        limits = fetch_limits()

        graph1 = draw_graph1(expenses=expenses)
        graph2 = draw_graph2(expenses=expenses, limits=limits)

        return render_template("analysis.html", img_data1=graph1.decode('utf-8'), img_data2=graph2.decode('utf-8'))

    elif request.method == 'POST':
        return render_template('analysis.html')


def execute_sql(sql, *args):
    stmt = ibm_db.prepare(conn, sql)
    for i, arg in enumerate(args):
        ibm_db.bind_param(stmt, i + 1, arg)
    ibm_db.execute(stmt)
    return stmt


def check_monthly_limit(month, year):
    sql = 'SELECT SUM(EXPENSE_AMOUNT) FROM PETA_EXPENSE WHERE USERID = ? AND MONTH(DATE) = ?
AND YEAR(DATE) = ?'
    statement = execute_sql(sql, USERID, month, year)
    amt_spent = ibm_db.fetch_tuple(statement)

    sql = 'SELECT LIMITAMOUNT FROM PETA_LIMIT WHERE USERID = ? AND LIMITMONTH = ? AND LIMITYEAR
= ?'
    statement = execute_sql(sql, USERID, month, year)
    monthly_limit = ibm_db.fetch_tuple(statement)

    if amt_spent and monthly_limit and int(amt_spent[0]) > int(monthly_limit[0]):
        diff = int(amt_spent[0]) - int(monthly_limit[0])
        print(diff)
        msg = Message('Monthly limit exceeded', recipients=[email])
        msg.body = (
            f'Monthly limit exceeded by {diff} for the month of {month}, {year}')
        mail.send(msg)


def update_monthly_limit(monthly_limit, month, year):
    sql = 'SELECT LIMITAMOUNT FROM PETA_LIMIT WHERE USERID = ? AND LIMITMONTH = ? AND LIMITYEAR
= ?'
    statement = execute_sql(sql, USERID, month, year)

    if ibm_db.fetch_row(statement):
        sql = 'UPDATE PETA_LIMIT SET LIMITAMOUNT = ? WHERE USERID = ? AND LIMITMONTH = ? AND
LIMITYEAR = ?'
        execute_sql(sql, monthly_limit, USERID, month, year)
    else:
        sql = 'INSERT INTO PETA_LIMIT VALUES(?, ?, ?, ?)'
        execute_sql(sql, USERID, monthly_limit, month, year)

    check_monthly_limit(month, year)


@app.route('/setmonthlylimit', methods=['GET', 'POST'])
def set_monthly_limit():
    if request.method == 'GET':
        return render_template('setmonthlylimit.html')
    elif request.method == 'POST':
        new_monthly_limit = request.form['monthlylimit']
        now = datetime.now()
        update_monthly_limit(new_monthly_limit, now.month, now.year)
        return redirect(url_for('dashboard'))


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

FEATURE 2

```
<!DOCTYPE html>
<html lang="en">
<head>
   <!-- Required meta tags -->
   <meta charset="utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1">

   <!-- Bootstrap CSS -->
   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">

   <!-- bootstrap for the cards -->
   <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@4.0.0/dist/css/bootstrap.min.css" integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

   <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM" crossorigin="anonymous"></
script>

   {% block title %}
      <title>Base Template</title>
   {% endblock title %}
</head>
<body>
   <div class="container-fluid">
      <div class="row flex-nowrap">
         <div class="col-auto col-md-3 col-xl-2 px-sm-2 px-0" style="background-color: #B2D3C2">
            <div class="d-flex flex-column align-items-center align-items-sm-start px-3 pt-2 min-vh-100" style="color:black">
               <p class="d-flex align-items-center pb-3 mb-md-0 me-md-auto text-white text-decoration-none">
                  <span class="fs-5 d-none d-sm-inline" style="color:black; font-weight: bold;">Personal Expense Tracker</span>
                  <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/piggybank.png"
style="width:50px;height: 50px;">
               </p>
               <ul class="nav nav-pills flex-column mb-sm-auto mb-0 align-items-center align-items-sm-start" id="menu">
                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'dashboard'}}; height: 50px;
width: 150px; border-radius: 5px;">
                     <a href="dashboard" class="nav-link align-middle px-0" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/house-
outline.svg" style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Home</span>
                     </a>
                  </li>

                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'addexpense'}};">
                     <a href="addexpense" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/pay-15.png"
style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Add Expense</span>
                     </a>
                  </li>

                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'recurringexpense'}};">
                     <a href="recurringexpense" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/pay-15.png"
style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Initiate a recurring expense</span>
                     </a>
                  </li>

                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'modifyexpense'}};">
                     <a href="modifyexpense" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/
edit_icon.svg" style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Modify Expense</span>
                     </a>
                  </li>

                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'viewrecurring'}};">
                     <a href="viewrecurring" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/graph.png"
style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">View recurring expenses</span>
                     </a>
                  </li>

                  <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'analysis'}};">
                     <a href="analysis" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/graph.png"
style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">View Analysis</span>
                     </a>
```

```
                </li>

                <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'rewards'}};">
                    <a href="rewards" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/reward.png"
style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Rewards & Goals</span>
                    </a>
                </li>

                <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'addcategory'}};">
                    <a href="addcategory" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/
category_add.webp" style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Create category</span>
                    </a>
                </li>

                <li class="nav-item mt-2" style="background-color: {{'#00AD83' if highlight == 'setmonthlylimit'}};">
                    <a href="setmonthlylimit" class="nav-link px-0 align-middle" style="color:black;">
                        <img src="https://s3.jp-tok.cloud-object-storage.appdomain.cloud/personalexpensetrackercapd/
category_add.webp" style="width:20px;height:20px;margin-left: 5px;">
                        <span class="ms-1 d-none d-sm-inline">Set Monthly Limit</span>
                    </a>
                </li>
            </ul>
        </div>
    </div>
    {% block content %}
        <h1>This needs to be overriden</h1>
    {% endblock content %}
  </div>
</div>

{% block script %}
<script></script>
{% endblock script %}
</body>
</html>
```

7. TESTING

Test Cases

S.no	S.no
	Authentication
1	Verify user is able to see login page
2	Verify user is able to loginto application or not?
3	Verify user is able to navigate to create your account page?
4	Veriify login page elements
	Expense
1	User after successful authentication can view the the time line?
2	Can user view the consolidated expenses in the form of pie chart?
3	Can user filter by date and view that specific expense by date?

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Expected Result	Actual Result	Status	Commnets	TC for Automation(Y/N)	BUG ID
LoginPage_TC_OO1	Functional	Home Page	Verify user is able to see the Login/ Signup	1.Open the app 2.Incase of registered users,enter username and password. 3.If the user doesnt have an account,go over to register now	1.Newly registed users are redirected to login page. 2.Exisiting users when entered the correct creditionals get redirected to go timeline	Working as expected	Pass			
LoginPage_TC_OO2	UI	Home Page	Verify the UI elements in Login/ Signup.	1.Open the application 2.Verify login/ Singup with below UI elements: a.email text box b.password text box c.Dont have an account? register now d.New customer? Create account link	Application should show below UI elements: a.email text box b.password text box c.Submit button with blue button d.Dont have an account? Register now	Working as expected	Fail	Steps are not clear to follow		BUG-1234
LoginPage_TC_OO3	Functional	Home page	Verify user is able to log into application with Valid credentials	1.Open the application 2.Enter Valid email in Email text box 4.Enter valid password in password text box 5.Click on submit button	User should be navigated to the Homepage	Works as expected	Pass			

LoginPage_TC_OO4	Functional	Login page	Verify user is able to log into application with InValid credentials	1.Open the application 2.Enter IValid username/email in Email text box 4.Enter Invalid password in password text box 5.Click on submit button	Application should show 'Invalid Credentials ' validation message.	Worked as expected				

b. User Acceptance Testing

1. **Purpose of Document**

The purpose of this document is to briefly explain the test coverage and open issues of the Personal expense tracker project at the time of the release to User Acceptance Testing (UAT).

2. **Defect Analysis**

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

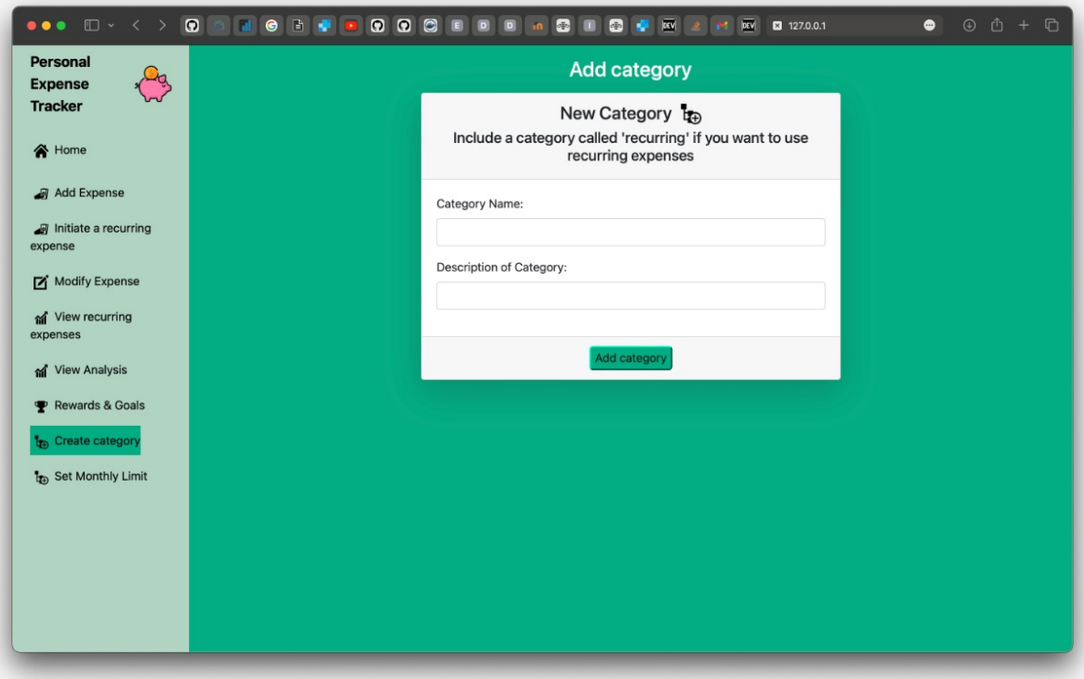
Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	2	0	0	0	2
External	0	2	0	1	3
Fixed	2	1	0	0	3
Skipped	0	1	0	0	1
Won't Fix	0	0	0	1	1
Totals	4	4	0	2	10

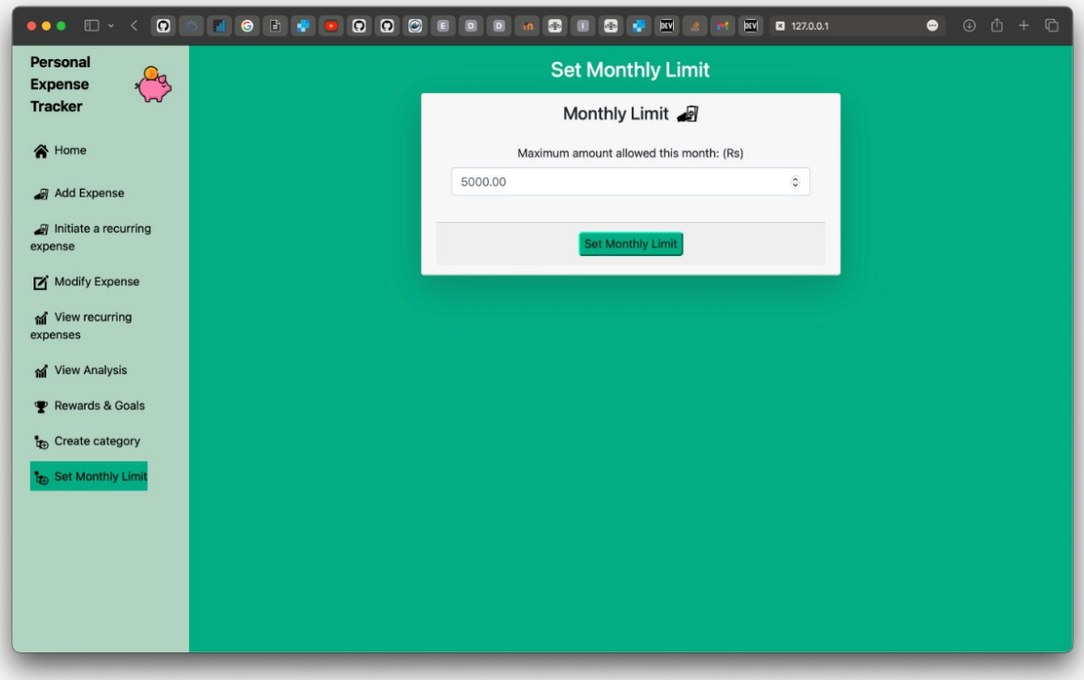
3. **Test Case Analysis**

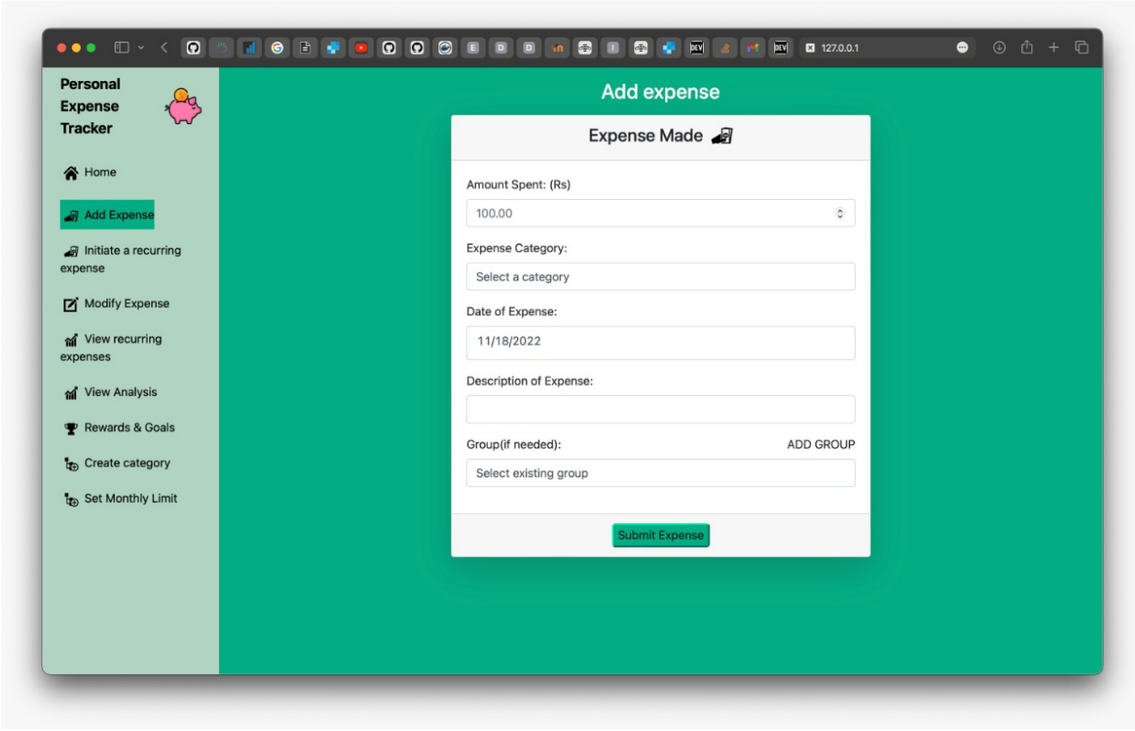
This report shows the number of test cases that have passed, failed, and untested

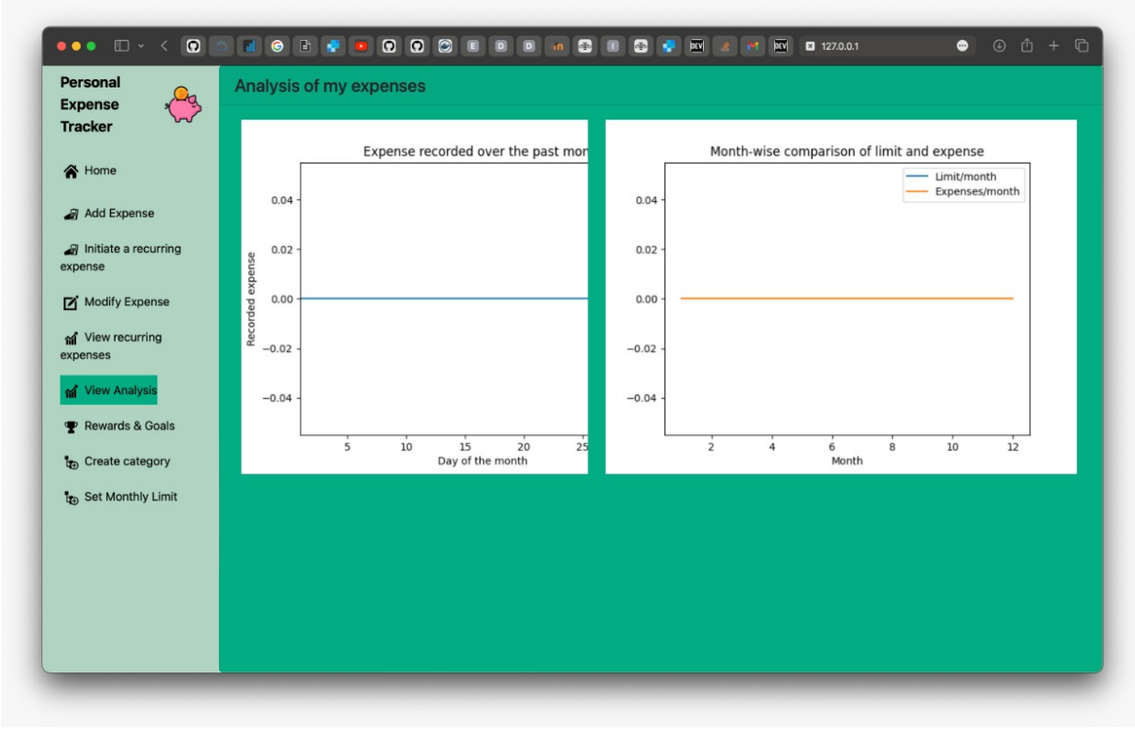
Section	Total Cases	Not Tested	Fail	Pass
UI	3	0	0	3
Database	2	0	0	2
Design	4	0	0	4
Deployment	1	0	1	0

8.RESULTS


127.0.0.1
Personal Expense Tracker
Home
Add Expense
Initiate a recurring expense
Modify Expense
View recurring expenses
View Analysis
Rewards & Goals
Create category
Set Monthly Limit
Add category
New Category
Include a category called 'recurring' if you want to use recurring expenses
Category Name:
Description of Category:
Add category


127.0.0.1
Personal Expense Tracker
Home
Add Expense
Initiate a recurring expense
Modify Expense
View recurring expenses
View Analysis
Rewards & Goals
Create category
Set Monthly Limit
Set Monthly Limit
Monthly Limit
Maximum amount allowed this month: (Rs)
5000.00
Set Monthly Limit





10.ADVANTAGES AND DISADVANTAGES

- Ease of access
- Prioritize Your Spending
- Become Aware of Poor Spending Habits
- Identify Fraud
- Minimal manual intervention
- Reduced turnaround time and faster reimbursements

Disadvantages

- Negligence during approval
- Negligence during auditing
- Although software simplifies the work of auditors, there is less scope for auditors to investigate suspicious activities. External verification can become minimal or redundant since all expense reports are already internally verified by the software. Due to this, auditors may choose to trust the software's discretion and neglect uncertainties. For avoiding such a situation, audit trails come in very handy. It helps keep every report audit-ready and traceable.
- Poor customer support

11. CONCLUSION

Monitoring your everyday expenses can set aside you cash, yet it can likewise help you set your monetary objectives for what's to come. On the off chance that you know precisely where your sum is going much of a stretch see where a few reductions and bargains can be made. Expense Tracker project is for keeping our day-to-day expenditures will helps us to keep record of our money daily. The project what we have created is work more proficient than the other income and expense tracker. The project effectively keeps away from the manual figuring for trying not to ascertain the pay and cost each month. It's a user-friendly application.

12. FUTURE SCOPE

1) It will have various options to keep record (for example Food, Travelling Fuel, Salary etc.).

2) Automatically it will keep on sending notifications for our daily expenditure.

3) In today's busy and expensive life, we are in a great rush to make moneys, but at the end of the month we broke off. As we are unknowingly spending money on title and unwanted things. So, we have come over with the plan to follow our profit.

4) Here user can define their own categories for expense type like food, clothing, rent and bills where they have to enter the money that has been spend and likewise can add some data in extra data to indicate the expense.

13.APPENDIX

Source code

```
from flask import Flask, render_template, request, redirect, url_for
from flask_mail import Mail, Message
from datetime import datetime
from flask_cors import CORS, cross_origin
import ibm_db
import json
import plotly
import plotly.graph_objs as go
import pandas as pd
from flask import send_file
from io import BytesIO
import matplotlib.pyplot as plt
import numpy as np
import base64
from PIL import Image
import time
import atexit
from datetime import datetime
from apscheduler.schedulers.background import BackgroundScheduler

app = Flask(__name__, template_folder='templates')
app.config['SECRET_KEY'] = 'top-secret!'
app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_PASSWORD'] = 'SG.rRPqo3ZyRhWUD6RhljE1CA.894zN6QMM9UjOpgPlO-4KT-_mjT9-
KwXZ9ArygkEnis'
app.config['MAIL_DEFAULT_SENDER'] = 'nunnaaarthi@gmail.com'
mail = Mail(app)
cors = CORS(app)
app.config['CORS_HEADERS'] = 'Content-Type'

# GLobal variables
EMAIL = ''
USERID = ''

conn =
ibm_db.connect("DATABASE=bludb;HOSTNAME=54a2f15b-5c0f-46df-8954-7e38e612c2bd.c1ogj3sd0tgtu0lqde00.databases
.appdomain.cloud;PORT=32733;Security=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=nlg66799;PWD=CXtQ
LAGZ06fD0fhC;", "", "")

# FUNCTIONS INTERACTING WITH DB #


def fetch_walletamount():
```

```
    sql = 'SELECT WALLET FROM PETA_USER WHERE EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, EMAIL)
    ibm_db.execute(stmt)
    user = ibm_db.fetch_assoc(stmt)
    # print(user['WALLET'])
    return user['WALLET']  # returns int


def fetch_categories():

    sql = 'SELECT * FROM PETA_CATEGORY WHERE USERID = ?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, USERID)
    ibm_db.execute(stmt)

    categories = []
    while ibm_db.fetch_row(stmt) != False:
        categories.append([ibm_db.result(stmt, "CATEGORYID"),
                           ibm_db.result(stmt, "CATEGORY_NAME")])

    sql = 'SELECT * FROM PETA_CATEGORY WHERE USERID IS NULL'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)

    while ibm_db.fetch_row(stmt) != False:
        categories.append([ibm_db.result(stmt, "CATEGORYID"),
                           ibm_db.result(stmt, "CATEGORY_NAME")])

    # print(categories)
    return categories  # returns list


def fetch_userID():
    sql = 'SELECT USERID FROM PETA_USER WHERE EMAIL=?'
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, EMAIL)
    ibm_db.execute(stmt)
    user = ibm_db.fetch_assoc(stmt)
    # print(user['USERID'])
    return user['USERID']  # returns int


def fetch_groups():
    sql = 'SELECT * FROM PETA_GROUPS'
    stmt = ibm_db.exec_immediate(conn, sql)
    groups = []
    while ibm_db.fetch_row(stmt) != False:
        groups.append([ibm_db.result(stmt, "GROUPID"),
                       ibm_db.result(stmt, "GROUPNAME")])
    # print(groups)
    return groups  # returns list


def fetch_expenses():
    sql = 'SELECT * FROM PETA_EXPENSE where USERID = ' + str(USERID)
    # print(sql)
    stmt = ibm_db.exec_immediate(conn, sql)
    expenses = []
    while ibm_db.fetch_row(stmt) != False:
        category_id = ibm_db.result(stmt, "CATEGORYID")
        category_id = str(category_id)
        sql2 = "SELECT * FROM PETA_CATEGORY WHERE CATEGORYID = " + category_id
        stmt2 = ibm_db.exec_immediate(conn, sql2)
        category_name = ""
        while ibm_db.fetch_row(stmt2) != False:
            category_name = ibm_db.result(stmt2, "CATEGORY_NAME")
        expenses.append([ibm_db.result(stmt, "EXPENSE_AMOUNT"), ibm_db.result(
            stmt, "DATE"), ibm_db.result(stmt, "DESCRIPTION"), category_name])
    # print(expenses)
    return expenses


def fetch_rec_expenses_cron():
    sql = 'SELECT * FROM PETA_REC_EXPENSES;'
    stmt = ibm_db.exec_immediate(conn, sql)
    rec_expenses = []
    while ibm_db.fetch_row(stmt) != False:
        amt = ibm_db.result(stmt, "AMOUNT")
        amt = str(amt)
        description = ibm_db.result(stmt, "DESCRIPTION")
        userid = ibm_db.result(stmt, "USERID")
        date = ibm_db.result(stmt, "RECDATE")
        rec_expenses.append([amt, description, date, userid])
```

```
    # print(rec_expenses)
    return rec_expenses


def fetch_rec_expenses():
    sql = 'SELECT * FROM PETA_REC_EXPENSES WHERE USERID = ' + str(USERID)
    stmt = ibm_db.exec_immediate(conn, sql)

    rec_expenses = []
    while ibm_db.fetch_row(stmt) != False:
        amt = ibm_db.result(stmt, "AMOUNT")
        amt = str(amt)
        description = ibm_db.result(stmt, "DESCRIPTION")
        userid = ibm_db.result(stmt, "USERID")
        date = ibm_db.result(stmt, "RECDATE")
        rec_expenses.append([amt, description, date, userid])
    # print(rec_expenses)
    return rec_expenses


def fetch_limits():
    now = datetime.now()
    year = now.year

    limits = [0 for i in range(12)]

    sql = 'SELECT LIMITAMOUNT, LIMITMONTH FROM PETA_LIMIT WHERE USERID = ? AND LIMITYEAR = ?'
    statement = execute_sql(sql, USERID, year)

    while ibm_db.fetch_row(statement):
        limit_amount = int(ibm_db.result(statement, 'LIMITAMOUNT'))
        limit_month = int(ibm_db.result(statement, 'LIMITMONTH'))
        limits[limit_month] = limit_amount

    return limits


# HELPER FUNCTIONS #
def fetch_latest_expenses(expenses):
    # must return expenses of last month
    latest_month = datetime.today().month
    latest_expenses = []
    for exp in expenses:
        if exp[1].month == latest_month:
            latest_expenses.append(exp)

    return latest_expenses


def fetch_monthly_expenses(expenses):
    latest_year = datetime.today().year
    monthly_expenses = {}

    for month in range(1, 13):
        monthly_expenses[month] = 0

    for exp in expenses:
        if exp[1].year == latest_year:
            monthly_expenses[exp[1].month] += exp[0]

    return monthly_expenses.values()


def draw_graph1(expenses):
    # TOTAL EXPENSE / DAY OF MONTH
    # x-axis: day , y-axis: expense/day

    latest_expenses = fetch_latest_expenses(expenses)
    mp = {}
    for day in range(1, 31):
        mp[day] = 0

    for exp in latest_expenses:
        mp[exp[1].day] += exp[0]

    x = mp.keys()
    y = mp.values()

    # print(mp)

    plt.figure()
    plt.title('Expense recorded over the past month')
    plt.plot(x, y)
    plt.xlabel('Day of the month')
```

```
    plt.ylabel('Recorded expense')
    plt.xlim(1, 32)

    buffer = BytesIO()
    plt.savefig(buffer, format='png')

    encoded_img_data = base64.b64encode(buffer.getvalue())

    return encoded_img_data


def draw_graph2(expenses, limits):
    # limit/month vs expense/month -> 2 line graphs

    monthly_expenses = fetch_monthly_expenses(expenses)
    x = range(1, 13)
    y1 = limits
    y2 = monthly_expenses

    plt.figure()
    plt.title('Month-wise comparison of limit and expense')
    plt.plot(x, y1, label="Limit/month")
    plt.plot(x, y2, label="Expenses/month")
    plt.xlabel('Month')
    plt.legend()

    buffer = BytesIO()
    plt.savefig(buffer, format='png')

    encoded_img_data = base64.b64encode(buffer.getvalue())

    return encoded_img_data

# finds the category id that matches that of the recurring expense category


def fetch_recurring_category_id():
    categories = fetch_categories()
    for category in categories:
        p = ''
        for i in category[1]:
            if (i == ' '):
                break
            p += i
        category[1] = p
    print(categories)
    categoryid = -1
    for category in categories:
        if category[1] == 'recurring':
            categoryid = category[0]
    print(categoryid)
    return categoryid


# cron to autodeduct the expenses each day
def auto_renew():
    global USERID
    # print(time.strftime("%A, %d. %B %Y %I:%M:%S %p"))
    rec_expenses = fetch_rec_expenses_cron()
    print(rec_expenses)
    current_day = time.strftime("%d")
    print(current_day)
    for expense in rec_expenses:
        here = str(expense[2])
        here = here.split('-')
        here = here[2]
        print(here)
        if (here == current_day):
            sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, DESCRIPTION, DATE)
VALUES(?,?,?,?,?);"
            USERID = str(expense[3])
            categoryid = fetch_recurring_category_id()
            print(categoryid)
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, expense[3])
            ibm_db.bind_param(stmt, 2, expense[0])
            ibm_db.bind_param(stmt, 3, categoryid)
            ibm_db.bind_param(stmt, 4, expense[1])
            d3 = time.strftime("%Y-%m-%d")
            ibm_db.bind_param(stmt, 5, d3)
            print(d3, categoryid, expense[0],
                  expense[1], expense[2], expense[3])
            ibm_db.execute(stmt)
            # print(here, d3, expense[0], expense[1], expense[2])
```

```
        sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?"
        statement = ibm_db.prepare(conn, sql)
        print(USERID)
        ibm_db.bind_param(statement, 1, expense[0])
        ibm_db.bind_param(statement, 2, expense[3])
        print("deducted")
        ibm_db.execute(statement)


# caller code for the cron
scheduler = BackgroundScheduler()
scheduler.add_job(func=auto_renew, trigger="interval", seconds=3600 * 24)
scheduler.start()

atexit.register(lambda: scheduler.shutdown())


# END POINTS #
@app.route('/', methods=['GET', 'POST'])
@cross_origin()
def registration():
    global EMAIL
    print("hello")
    if request.method == 'GET':
        return render_template('registration.html')
    if request.method == 'POST':
        email = request.form['email']
        EMAIL = email
        password = request.form['password']
        wallet = request.form['wallet']
        sql = "INSERT INTO PETA_USER(EMAIL,PASSWORD,WALLET) VALUES(?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.bind_param(stmt, 3, wallet)
        print(stmt)
        ibm_db.execute(stmt)
        # msg = Message('Registration Verfication',recipients=[email])
        # msg.body = ('Congratulations! Welcome user!')
        # msg.html = ('<h1>Registration Verfication</h1>'
        #             '<p>Congratulations! Welcome user!'
        #             '<b>PETA</b>!</p>')
        # mail.send(msg)
        EMAIL = email
    return redirect(url_for('dashboard'))


@app.route('/login', methods=['GET', 'POST'])
def login():
    global EMAIL
    print("login")
    if request.method == 'POST':
        email = request.form['email']
        EMAIL = email
        print(EMAIL)
        password = request.form['password']
        sql = "SELECT * FROM PETA_USER WHERE email=? AND password=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        if account:
            return redirect(url_for('dashboard'))
        else:
            return redirect(url_for('login'))
    elif request.method == 'GET':
        return render_template('login.html')


@app.route('/dashboard', methods=['GET'])
def dashboard():
    global USERID
    global EMAIL
    print("dashboard")
    if USERID == '' and EMAIL == '':
        print("null email")
        return render_template('login.html')
    elif USERID == '':
        USERID = fetch_userID()
        print(USERID)
    expenses = fetch_expenses()
    wallet = fetch_walletamount()
    return render_template('dashboard.html', expenses=expenses, wallet=wallet, email=EMAIL)
```

```
@app.route('/updatebalance', methods=['GET', 'POST'])
def update_balance():
    if request.method == 'GET':
        wallet = fetch_walletamount()
        return render_template('updatebalance.html', wallet=wallet)
    elif request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()

        new_balance = request.form['balanceupdated']
        sql = 'UPDATE PETA_USER SET WALLET = ? WHERE USERID = ?'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, new_balance)
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/addcategory', methods=['GET', 'POST'])
def add_category():
    if request.method == 'GET':
        # categories = fetch_categories()
        return render_template('addcategory.html')

    elif request.method == 'POST':
        categoryname = request.form['category']
        sql = 'INSERT INTO PETA_CATEGORY(CATEGORY_NAME, USERID) VALUES(?,?)'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, categoryname)
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/addgroup', methods=['POST'])
def add_group():
    if request.method == 'POST':
        if USERID == '':
            return render_template('login.html', msg='Login before proceeding')
        sql = "INSERT INTO PETA_GROUPS(GROUPNAME, USERID) VALUES(?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, request.form['groupname'])
        ibm_db.bind_param(stmt, 2, USERID)
        ibm_db.execute(stmt)

        group_info = {}

        sql = "SELECT * FROM PETA_GROUPS WHERE GROUPNAME=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, request.form['groupname'])
        ibm_db.execute(stmt)
        group_info = ibm_db.fetch_assoc(stmt)
        return {"groupID": group_info['GROUPID'], 'groupname': group_info['GROUPNAME']}


@app.route('/addexpense', methods=['GET', 'POST'])
def add_expense():
    if request.method == 'GET':
        groups = fetch_groups()
        categories = fetch_categories()
        if len(categories) == 0:
            return redirect(url_for('add_category'))
        return render_template('addexpense.html', categories=categories, groups=groups)

    elif request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()

        amount_spent = request.form['amountspent']
        category_id = request.form.get('category')
```

```
        description = request.form['description']
        date = request.form['date']
        groupid = request.form.get('group')
        # print(amount_spent, category_id, description, date, groupid, USERID)

        sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, GROUPID, DESCRIPTION,
DATE) VALUES(?,?,?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, amount_spent)
        ibm_db.bind_param(stmt, 3, category_id)
        ibm_db.bind_param(stmt, 4, groupid)
        ibm_db.bind_param(stmt, 5, description)
        ibm_db.bind_param(stmt, 6, date)
        ibm_db.execute(stmt)
        print(date, amount_spent, category_id)
        sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?"
        statement = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(statement, 1, amount_spent)
        ibm_db.bind_param(statement, 2, USERID)
        ibm_db.execute(statement)

        return redirect(url_for('dashboard'))


@app.route('/viewrecurring', methods=['GET'])
def viewrecurring():
    global USERID
    global EMAIL
    print("viewrecurring")
    if USERID == '' and EMAIL == '':
        print("null email")
        return render_template('login.html')
    elif USERID == '':
        USERID = fetch_userID()
        # print(USERID)
    print(USERID)
    expenses = fetch_rec_expenses()
    wallet = fetch_walletamount()
    return render_template('viewrecurring.html', expenses=expenses, wallet=wallet, email=EMAIL)


@app.route('/recurringexpense', methods=['GET', 'POST'])
def recurring_expense():
    global USERID, EMAIL
    if request.method == 'GET':
        groups = fetch_groups()
        categories = fetch_categories()
        if len(categories) == 0:
            return redirect(url_for('add_category'))
        USERID = fetch_userID()
        # check if user has added a category for recurring category, if not redirect and ask her to
        recur_id = fetch_recurring_category_id()
        if (recur_id == -1):
            return (redirect(url_for('add_category')))
        return render_template('recurringexpense.html', categories=categories, groups=groups)

    elif request.method == 'POST':
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()
            # check if user has added a category for recurring category, if not redirect and ask her to
            recur_id = fetch_recurring_category_id()
            if (recur_id == -1):
                return (redirect(url_for('add_category')))
        amount_spent = request.form['amountspent']
        category_id = request.form.get('category')
        description = request.form['description']
        date = request.form['date']
        # months = request.form['autorenewals']
        # groupid = request.form.get('group')
        print("recurring : ")
        print(amount_spent, description, date, USERID)

        sql = "INSERT INTO PETA_REC_EXPENSES(AMOUNT, RECDATE, USERID, DESCRIPTION) VALUES (?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, amount_spent)
        ibm_db.bind_param(stmt, 2, date)
        ibm_db.bind_param(stmt, 3, USERID)
        ibm_db.bind_param(stmt, 4, description)
        ibm_db.execute(stmt)
```

```
        sql = "INSERT INTO PETA_EXPENSE(USERID, EXPENSE_AMOUNT, CATEGORYID, DESCRIPTION, DATE)
VALUES(?,?,?,?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, amount_spent)
        ibm_db.bind_param(stmt, 3, category_id)
        ibm_db.bind_param(stmt, 4, description)
        ibm_db.bind_param(stmt, 5, date)
        ibm_db.execute(stmt)

        sql = "UPDATE PETA_USER SET WALLET = WALLET - ? WHERE USERID = ?;"
        statement = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(statement, 1, amount_spent)
        ibm_db.bind_param(statement, 2, USERID)
        ibm_db.execute(statement)

        return redirect(url_for('dashboard'))


@app.route('/removerecurring', methods=['POST'])
def remove_recurring():
    print("remove recurring")
    if request.method == 'POST':
        global EMAIL
        global USERID
        if EMAIL == '':
            return render_template('login.html', msg='Login before proceeding')
        if (USERID == ''):
            # get user using email
            USERID = fetch_userID()
        description = request.form['description']
        print(description, USERID)
        sql = 'DELETE FROM PETA_REC_EXPENSES WHERE USERID = ? AND DESCRIPTION = ?;'
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, USERID)
        ibm_db.bind_param(stmt, 2, description)
        ibm_db.execute(stmt)

        return redirect(url_for('dashboard'))


@app.route('/analysis', methods=['GET', 'POST'])
def analyse():
    if request.method == 'GET':
        expenses = fetch_expenses()
        limits = fetch_limits()

        graph1 = draw_graph1(expenses=expenses)
        graph2 = draw_graph2(expenses=expenses, limits=limits)

        return render_template("analysis.html", img_data1=graph1.decode('utf-8'), img_data2=graph2.decode('utf-8'))

    elif request.method == 'POST':
        return render_template('analysis.html')


def execute_sql(sql, *args):
    stmt = ibm_db.prepare(conn, sql)
    for i, arg in enumerate(args):
        ibm_db.bind_param(stmt, i + 1, arg)
    ibm_db.execute(stmt)
    return stmt


def check_monthly_limit(month, year):
    sql = 'SELECT SUM(EXPENSE_AMOUNT) FROM PETA_EXPENSE WHERE USERID = ? AND MONTH(DATE) = ?
AND YEAR(DATE) = ?'
    statement = execute_sql(sql, USERID, month, year)
    amt_spent = ibm_db.fetch_tuple(statement)

    sql = 'SELECT LIMITAMOUNT FROM PETA_LIMIT WHERE USERID = ? AND LIMITMONTH = ? AND LIMITYEAR
= ?'
    statement = execute_sql(sql, USERID, month, year)
    monthly_limit = ibm_db.fetch_tuple(statement)

    if amt_spent and monthly_limit and int(amt_spent[0]) > int(monthly_limit[0]):
        diff = int(amt_spent[0]) - int(monthly_limit[0])
        print(diff)
        msg = Message('Monthly limit exceeded', recipients=[email])
        msg.body = (
            f'Monthly limit exceeded by {diff} for the month of {month}, {year}')
        mail.send(msg)
```

```
def update_monthly_limit(monthly_limit, month, year):
    sql = 'SELECT LIMITAMOUNT FROM PETA_LIMIT WHERE USERID = ? AND LIMITMONTH = ? AND LIMITYEAR
= ?'
    statement = execute_sql(sql, USERID, month, year)

    if ibm_db.fetch_row(statement):
        sql = 'UPDATE PETA_LIMIT SET LIMITAMOUNT = ? WHERE USERID = ? AND LIMITMONTH = ? AND
LIMITYEAR = ?'
        execute_sql(sql, monthly_limit, USERID, month, year)
    else:
        sql = 'INSERT INTO PETA_LIMIT VALUES(?, ?, ?, ?)'
        execute_sql(sql, USERID, monthly_limit, month, year)

    check_monthly_limit(month, year)


@app.route('/setmonthlylimit', methods=['GET', 'POST'])
def set_monthly_limit():
    if request.method == 'GET':
        return render_template('setmonthlylimit.html')
    elif request.method == 'POST':
        new_monthly_limit = request.form['monthlylimit']
        now = datetime.now()
        update_monthly_limit(new_monthly_limit, now.month, now.year)
        return redirect(url_for('dashboard'))


if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

GITHUB AND PROJECT DEMO LINK

https://github.com/IBM-EPBL/IBM-Project-17473-1659672094.git