PERSONAL EXPENSE TRACKER APPLICATION

IBM-Project-Id - PNT2022TMID48571

A PROJECT REPORT

BY

Team Member: ISSAC SAMUEL
AKSHAY.R
ATHITHYAN.A
CHANDU.P

BACHELOR OF ENGINEERING IN ELECTRONICS AND COMMUNICATION ENGINEERING

CHRISTIAN COLLEGE OF ENGINEERING AND TECHNOLOGY, ODDANCHATRAM

INDEX

1. INTRODUCTION

1.1 PROJECT OVERVIEW

Category: Cloud App Development

Skills Required:

IBM Cloud, HTML, Javascript, IBM Cloud Object Storage, Python- Flask, Kubernetes, Docker, IBM DB2, IBM Container Registry

Project Description:

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

In this python Flask project, we will create an expense tracker that will take details of our expenses. While filling the signup form a person will also need to fill in the details about the income and the amount he/she wants to save. Some people earn on a daily basis, so their income can also be added on a regular basis. Details of expenses will be shown in

the form of a pie chart on a weekly, monthly, and yearly basis. Installation of Flask is a must to start with the Personal Expense Tracker Application project.

Personal finance management is an important part of people's lives. However, everyone does not have the knowledge or time to manage their finances in a proper manner. And, even if a person has time and knowledge, they do not bother with tracking their expenses as they find it tedious and time-consuming. Now, you don't have to worry about managing your expenses, as you can get access to an expense tracker that will help in the active management of your finances. Also known as expense manager and money manager, an expense tracker is a software or application that helps to keep an accurate record of your money inflow and outflow. Many people in India live on a fixed income, and they find that towards the end of the month they don't have sufficient money to meet their needs. While this problem can arise due to low salary, invariably it is due to poor money management skills.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

The problem of current generation population is that they can't remember where all of the money they earned have gone and ultimately have to live while sustaining the little money they have left for theirs sential needs. In this time there is no such perfect solution which helps a person to track their daily expenditure easily and efficiently and notify them about the money shortage they have. For doing sot hey have to maintain long ledger's or computer logs to maintain such data and the calculation is done manually by the user, which may generate error leading to losses. Not having a complete tracking.

2.2 REFERENCES

- ✓ **https://www.w3schools.com/python/**
- ✓ **https://getbootstrap.com/**
- ✓ **https://www.tutorialspoint.com/flask/index.htm**
- ✓ **https://www.w3schools.com/html/**

2.3 PROBLEM STATEMENT DEFINITION

Create a problem statement to understand your customer's point of view. The Customer Problem Statement template helps you focus on what matters to create experiences people will love.

A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

I am	Describe customer with 3-4 key characteristics - who are they?	Describe the customer and their attributes here
I'm trying to	List their outcome or "job" the care about - what are they trying to achieve?	List the thing they are trying to achieve here
but	Describe what problems or barriers stand in the way – what bothers them most?	Describe the problems or barriers that get in the way here
because	Enter the "root cause" of why the problem or barrier exists – what needs to be solved?	Describe the reason the problems or barriers exist
which makes me feel	Describe the emotions from the customer's point of view – how does it impact them emotionally?	Describe the emotions the result from experiencing the problems or barriers

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.

It is a useful tool to helps teams better understand their users.Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

3.2 IDEATION & BRAINSTROMING

- ✓ Step-1: Team Gathering, Collaboration and Select the Problem Statement
- ✓ Step-2 : Brainstorm Idea Listing and Grouping
- ✓ Step-3: Idea Prioritization
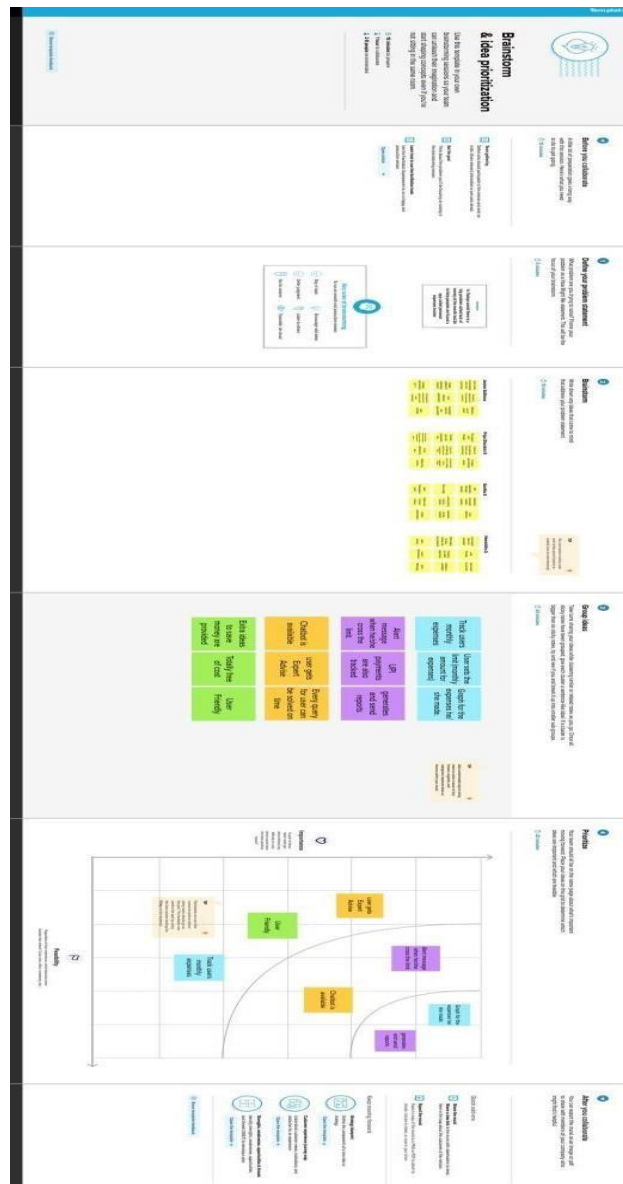
3.3 PROPOSED SOLUTION

All people in the earning sector needs a way to manage their financial resources and track their expenditure, so that they can improve and monitor their spending habits. This makes them understand the importance of financial management and makes them better decisions in the future .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. The solution to this problem is, the people who gets regular payment scan able to track their payments and void unwanted expenses. If the limit is exceeded the user will be notified with an email alert.

S.No.	Parameter	Description
1.	Problem Statement	Many organizations have their own system to record their income and expenses. It is good habit for a person to record daily expenses and tracking the expenses throughout the month is essential. Thus, personal expense tracker application has made tracking and managing expenses a breeze.
2.	NOVELTY	This app efficiently works in providing financial management and helps in maintaining healthy and happier financial life fulfilling all needs and requirements as the user's comfort. This app provides a higher range of accuracy regarding real- time efficiency and security.
3.	Feasibility of Idea	User can easily maintain their untracked expenses and the app helps the user to record their expenditure. This app can achieve economic feasibility and security feasibility with at most care and support to the user
4.	Business Model	The application can be provided based on user required feature and the cost depends on the usage

5.	**Social Impact**	This application can create awareness among common people about finance and stuffs. This application also helps user to be financially responsible
6.	**Scalability of the Solution**	This application can handle large number of users and data with high performance and security atany given point of time.

3.4 PROBLEM SOLUTION FIT

4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

Following are the functional requirements of the proposed solution.

- FR-1 User Registration ,Registration through Form Registration through Gmail
- Registration through linker IN
- FR-2 User Confirmation ,Confirmation via Email Confirmation via OTP
- FR-3 Tracking Expense Helpful insights about money management
- FR-4 Alert Message Give alert mail if the amount exceeds the budget limit
- FR-5 Category This application shall allow users to add categories of their expenses

4.2 NON-FUNCTIONAL REQUIREMENTS

Following are the non-functional requirements of the proposed solution.

- NFR-1 Usability You will able to allocate money to different priorities and also helpyoutocut down on unnecessary spending
- NFR-2 Security More security of the customer data and bank account details.
- NFR-3 Reliability Used to manage his/her expense so that the user is the path of financial stability. It is categorized by week, month, and year and also helps to see more expensesmade. Helps to define their own categories.
- NFR-4 Performance The types of expense are categories along with an option .Throughput of the system is increased due to light weight database support.
- NFR-5 Availability Able to track business expense and monitor important for maintaininghealthy cash flow.
- .NFR-6 Scalability The ability to appropriately handle increasingdemands.

5. PROJECT DESING

5.1 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a traditional visual representation of the information flows with in 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 store.
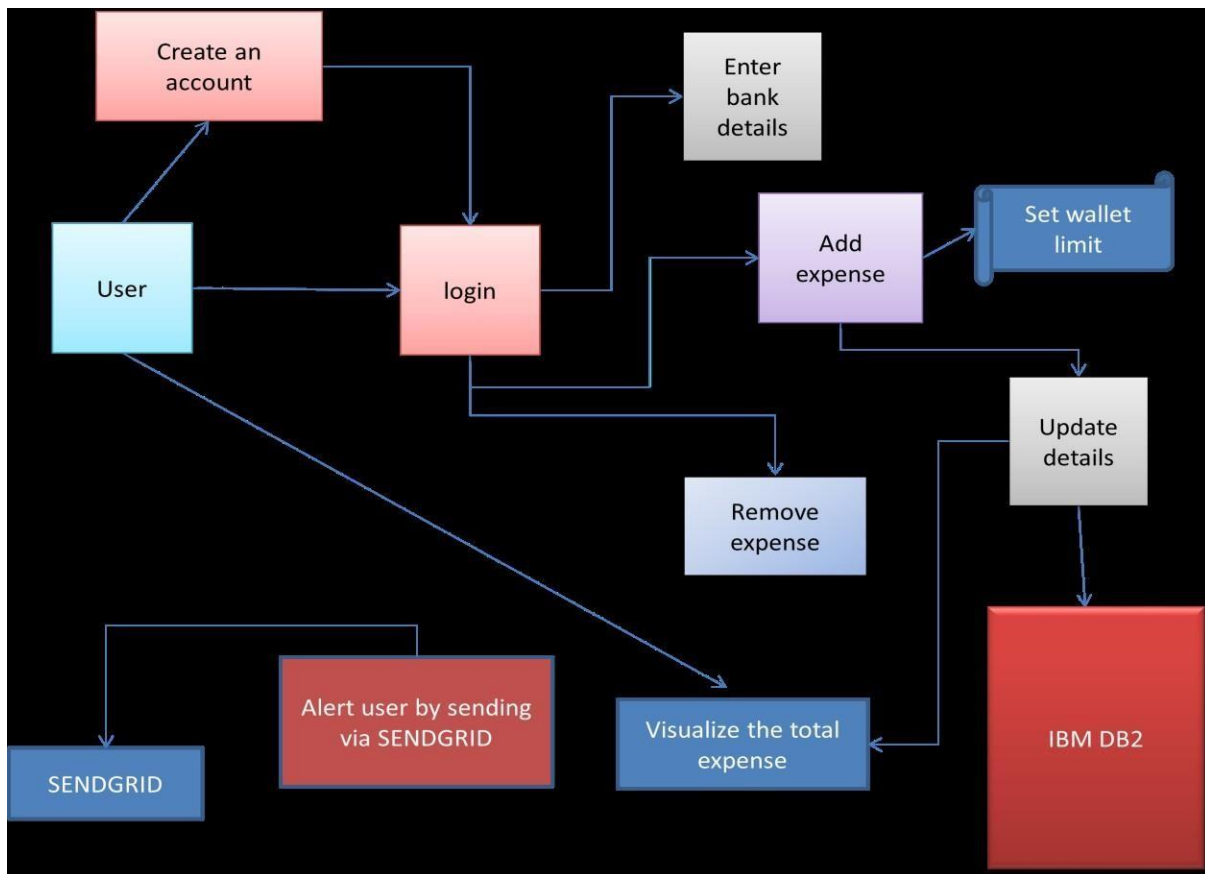
5.2 SOLUTION & TECHNICAL ARCHIECTURE

Solution Architecture:

Technical Architecture:
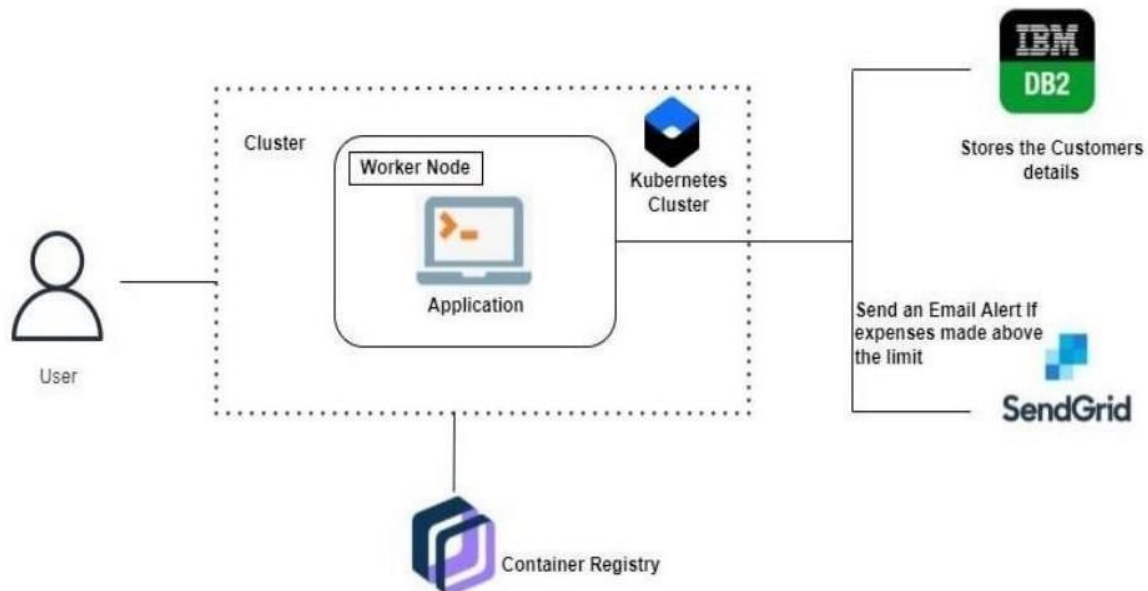
5.3 USER STORIES

Use the below template to list all the user stories for the product.

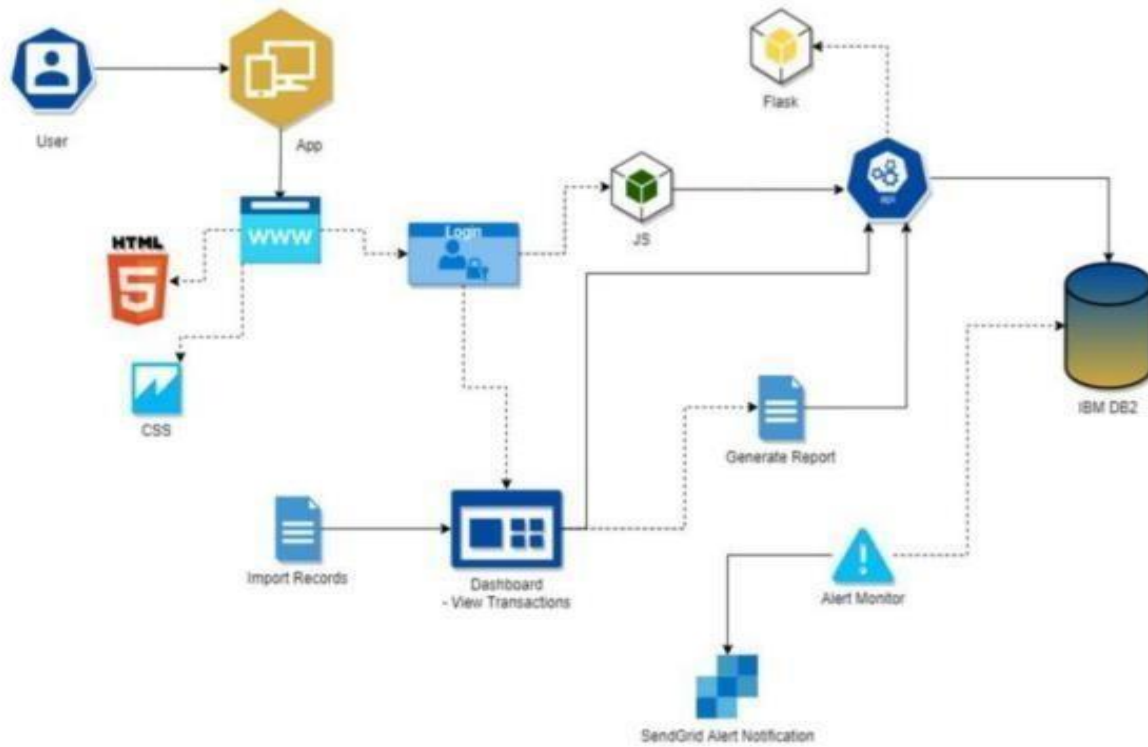| User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can access my account through Gmail login	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password	I can receive login confirmation and login credentials	High	Sprint-1

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
	Dashboard	USN-6	As a user, can access dashboard my to manage my expenses	Overall credit outlook	Low	Sprint-1
Customer (Web user)	Web user	USN-7	As a customer, can access the application using the web based platform also	Can have separate web page form	Medium	Sprint-1
Customer Care Executive	Expense management		As a customer care executive, periodically update and maintains expense application	Can have the login access when Admin permits	High	Sprint-1
Administrator	Creates and makes the application into use		As a administrator, is responsible for every expense count management	I can have the direct access to the application	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
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
		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High
		USN-3	As a user, I can register for the application through the gmail	1	Medium
	Login	USN-4	As a user, I can log into the application by entering email & password	1	High
	Dashboard	USN-5	Logging in takes to the dashboard for the logged user.	`2	High
*Bug fixes, routine checks and improvisation by everyone in the team *Intended bugs only*					
Sprint 2	Workspace	USN-1	Workspace for personal expense tracking	2	High
	Charts	USN-2	Creating various graphs and statistics of customer's data	1	Medium
	Connecting to IBM DB2	USN-3	Linking database with dashboard	2	High
Sprint-3		USN-1	Wrapping up the server side works of frontend	1	Medium
	Watson Assistant	USN-2	Creating Chatbot for expense tracking and for clarifying user's query	1	Medium
	SendGrid	USN-3	Using SendGrid to send mail to the user about their expenses	1	Low
		USN-4	Integrating both frontend and backend	2	High
*Bug fixes, routine checks and improvisation by everyone in the team *Intended bugs only*					
Sprint-4	Docker	USN-1	Creating image of website using docker/	2	High
	Cloud Registry	USN-2	Uploading docker image to IBM Cloud registry	2	High
	Kubernetes	USN-3	Create container using the docker image and hosting the site	2	High
	Exposing	USN-4	Exposing IP/Ports for the site	2	High

6.2 SPRINT DELIVERY SCHEDULE
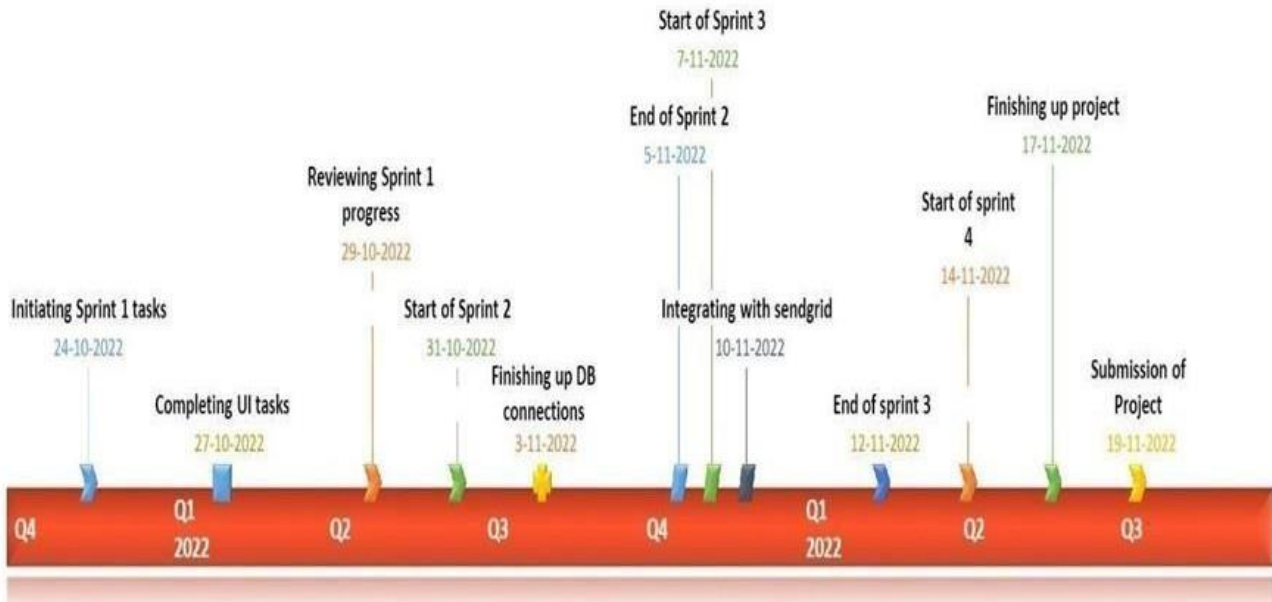
6.3 REPORTS FROM JIRA
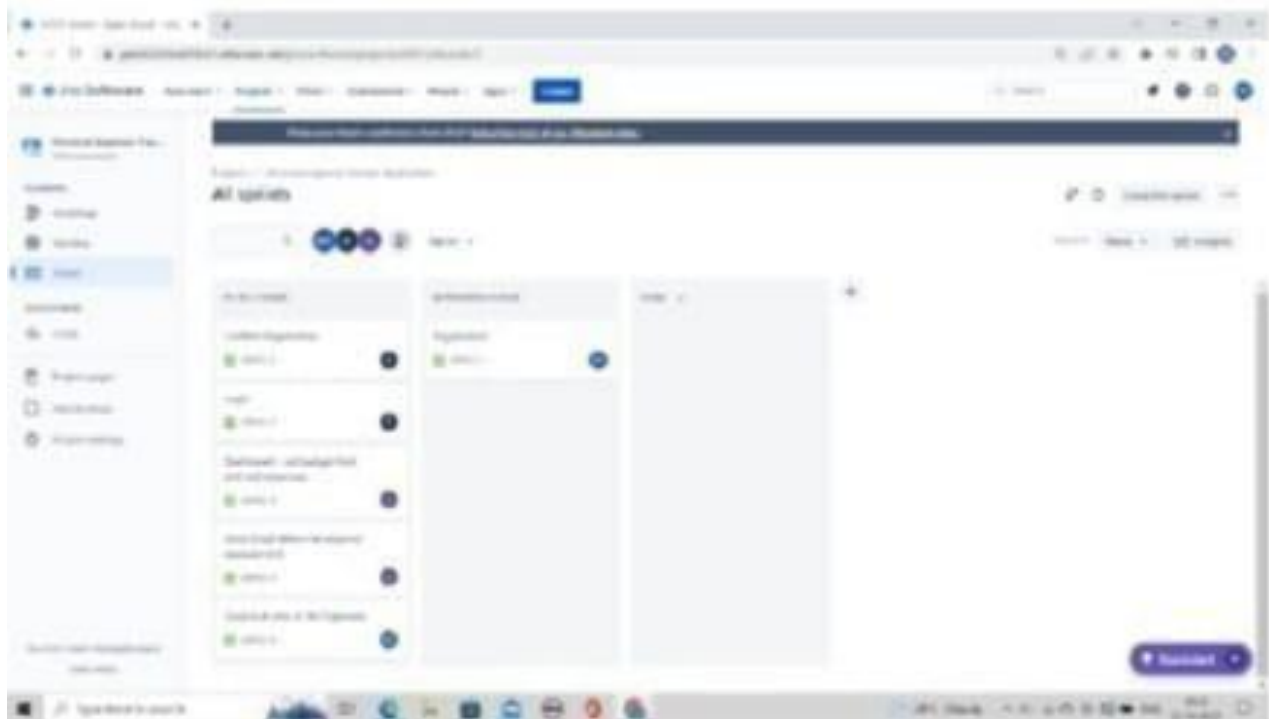
7. CODING & SOLUTIONING

7.1 FEATURE 1

We have added the data visualization on methods for expenditure. The pie chart have been used to represent the monthly expenses. The pie chart is a pictorial representation of data that makes it possible to visualize the relationships between the parts and the whole of a variable. For example, it is possible to understand the industry count or percentage of a variable level from the   division by areas or sectors. The recommended use for pie charts is two- dimensional, as three-dimensional use can be confusing.

CODE

TEMPLATES:

1. **Home.html**
2. **Homepage.html**
3. **Login.html**
4. **Add.html**
5. **Base.html**
6. **Signup.html**

Python code:

app.py

```python
from flask import Flask, render_template, request, redirect, session
import re
import sendgrid
from flask_db2 import DB2
import ibm_db
import ibm_db_dbi
import os
from sendemail import sendmail


app = Flask(__name__)
```

```python
app.secret_key = 'a'

app.config['database'] = 'bludb'
app.config['hostname'] = 'b1bc1829-6f45-4cd4-bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud'
app.config['port'] = '32304'
app.config['protocol'] = 'tcpip'
app.config['uid'] = 'ynk28411'
app.config['pwd'] = '23sBb3Fb0Z7wuUFY'
app.config['security'] = 'SSL'
try:
    mysql = DB2(app)

    conn_str='database=bludb;hostname=b1bc1829-6f45-4cd4-bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;port=32304;protocol=t
cpip;\
            uid=ynk28411;pwd=23sBb3Fb0Z7wuUFY;security=SSL'
    ibm_db_conn = ibm_db.connect(conn_str,'','')

    print("Database connected without any error !!")
except:
    print("IBM DB Connection error   :      " + DB2.conn_errormsg())

#HOME--PAGE
@app.route("/home")
def home():
    return render_template("homepage.html")
@app.route("/")
def add():
    return render_template("home.html")

#SIGN--UP--OR--REGISTER

@app.route('/signup')
def signup():
    return render_template('signup.html')

@app.route('/register', methods =['GET', 'POST'])
def register():
    if request.method == "POST":
        user_name = request.form['username']
        email = request.form['email']
        pass_word = request.form['password']
        query = "INSERT INTO Admin (username,email,password) values (?,?,?)"
```

```python
        insert_stmt = ibm_db.prepare(ibm_db_conn, query)
        ibm_db.bind_param(insert_stmt, 1, user_name)
        ibm_db.bind_param(insert_stmt, 2, email)
        ibm_db.bind_param(insert_stmt, 3, pass_word)
        ibm_db.execute(insert_stmt)
        msg = 'Account Created Successfully'
        return render_template("signup.html", msg=msg)

@app.route("/signin",methods=['post','get'])
def signin():
    if request.method=="post":
        return render_template("login.html")
    return render_template("login.html")

@app.route('/login',methods =['GET', 'POST'])
def login():
    global userid
    msg = ''


    if request.method == 'POST' :
        username = request.form['username']
        password = request.form['password']

        sql = "SELECT * FROM Admin WHERE username = ? and password = ?"
        stmt = ibm_db.prepare(ibm_db_conn,  sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        result = ibm_db.execute(stmt)
        print(result)
        account = ibm_db.fetch_row(stmt)
        print(account)

        param = "SELECT * FROM Admin WHERE username = " + "\'" + username + "\'"
+ " and password = " + "\'" + password + "\'"
        res = ibm_db.exec_immediate(ibm_db_conn, param)
        dictionary = ibm_db.fetch_assoc(res)

        # sendmail("hello sakthi","sivasakthisairam@gmail.com")

        if account:
            session['loggedin'] = True
            session['id'] = dictionary["ID"]
            userid = dictionary["ID"]
            session['username'] = dictionary["USERNAME"]
```

```python
            session['email'] = dictionary["EMAIL"]

            return redirect('/home')
        else:
            msg = 'Incorrect username / password !'

    return render_template('login.html', msg = msg)

@app.route("/add")
def adding():
    return render_template('add.html')

@app.route('/addexpense',methods=['GET', 'POST'])
def addexpense():

    date = request.form['date']
    expensename = request.form['expensename']
    amount = request.form['amount']
    paymode = request.form['paymode']
    category = request.form['category']

    print(date)
    p1 = date[0:10]
    p2 = date[11:13]
    p3 = date[14:]
    p4 = p1 + "-" + p2 + "." + p3 + ".00"
    print(p4)
    sql = "INSERT INTO Expense (userid, date, expensename, amount, paymode,
category) VALUES (?,?, ?, ?, ?, ?)"
    stmt = ibm_db.prepare(ibm_db_conn, sql)
    ibm_db.bind_param(stmt, 1, session['id'])
    ibm_db.bind_param(stmt, 2, p4)
    ibm_db.bind_param(stmt, 3, expensename)
    ibm_db.bind_param(stmt, 4, amount)
    ibm_db.bind_param(stmt, 5, paymode)
    ibm_db.bind_param(stmt, 6, category)
    ibm_db.execute(stmt)

    print("Expenses added")
    return redirect("/display")


    #DISPLAY---graph

@app.route("/display")
```

```python
def display():
    print(session["username"],session['id'])
    param = "SELECT * FROM Expense WHERE userid = " + str(session['id']) + "
ORDER BY date DESC"
    res = ibm_db.exec_immediate(ibm_db_conn, param)
    dictionary = ibm_db.fetch_assoc(res)
    expense = []
    while dictionary != False:
        temp = []
        temp.append(dictionary["ID"])
        temp.append(dictionary["USERID"])
        temp.append(dictionary["DATE"])
        temp.append(dictionary["EXPENSENAME"])
        temp.append(dictionary["AMOUNT"])
        temp.append(dictionary["PAYMODE"])
        temp.append(dictionary["CATEGORY"])
        expense.append(temp)
        print(temp)
        dictionary = ibm_db.fetch_assoc(res)

    return render_template('display.html' ,expense = expense)


#delete---the--data

@app.route('/delete/<string:id>', methods = ['POST', 'GET' ])
def delete(id):
    param = "DELETE FROM Expense WHERE  id = " + id
    res = ibm_db.exec_immediate(ibm_db_conn, param)

    print('deleted successfully')
    return redirect("/display")


#UPDATE---DATA

@app.route('/edit/<id>', methods = ['POST', 'GET' ])
def edit(id):
    param = "SELECT * FROM Expense WHERE  id = " + id
    res = ibm_db.exec_immediate(ibm_db_conn, param)
    dictionary = ibm_db.fetch_assoc(res)
    row = []
    while dictionary != False:
```

```python
        temp = []
        temp.append(dictionary["ID"])
        temp.append(dictionary["USERID"])
        temp.append(dictionary["DATE"])
        temp.append(dictionary["EXPENSENAME"])
        temp.append(dictionary["AMOUNT"])
        temp.append(dictionary["PAYMODE"])
        temp.append(dictionary["CATEGORY"])
        row.append(temp)
        print(temp)
        dictionary = ibm_db.fetch_assoc(res)

    print(row[0])
    return render_template('edit.html', expenses = row[0])




@app.route('/update/<id>', methods = ['POST'])
def update(id):
  if request.method == 'POST' :

      date = request.form['date']
      expensename = request.form['expensename']
      amount = request.form['amount']
      paymode = request.form['paymode']
      category = request.form['category']
      p1 = date[0:10]
      p2 = date[11:13]
      p3 = date[14:]
      p4 = p1 + "-" + p2 + "." + p3 + ".00"

      sql = "UPDATE Expense SET date = ? , expensename = ? , amount = ?, paymode
= ?, category = ? WHERE id = ?"
      stmt = ibm_db.prepare(ibm_db_conn, sql)
      ibm_db.bind_param(stmt, 1, p4)
      ibm_db.bind_param(stmt, 2, expensename)
      ibm_db.bind_param(stmt, 3, amount)
      ibm_db.bind_param(stmt, 4, paymode)
      ibm_db.bind_param(stmt, 5, category)
      ibm_db.bind_param(stmt, 6, id)
      ibm_db.execute(stmt)

      print('successfully updated')
      return redirect("/display")
```

```python
#limit
@app.route("/limit" )
def limit():
        return redirect('/limitn')

@app.route("/limitnum" , methods = ['POST' ])
def limitnum():
    if request.method == "POST":
        number= request.form['number']


        sql = "INSERT INTO limit (userid, limit) VALUES (?, ?)"
        stmt = ibm_db.prepare(ibm_db_conn, sql)
        ibm_db.bind_param(stmt, 1, session['id'])
        ibm_db.bind_param(stmt, 2, number)
        ibm_db.execute(stmt)

        return redirect('/limitn')



@app.route("/limitn")
def limitn():


    param = "SELECT id, limit FROM limit WHERE userid = " + str(session['id']) +
" ORDER BY id DESC LIMIT 1"
    res = ibm_db.exec_immediate(ibm_db_conn, param)
    dictionary = ibm_db.fetch_assoc(res)
    row = []
    s = "/-"
    while dictionary != False:
        temp = []
        temp.append(dictionary["LIMIT"])
        print(temp)
        row.append(temp)
        dictionary = ibm_db.fetch_assoc(res)
        s = temp[len(temp)-1]


    return render_template("limit.html" , y= s)

#REPORT
```

```python
@app.route("/today")
def today():

    param1 = "SELECT TIME(date) as tn, amount FROM Expense WHERE userid = " +
str(session['id']) + " AND DATE(date) = DATE(current timestamp) ORDER BY date
DESC"
    res1 = ibm_db.exec_immediate(ibm_db_conn, param1)
    dictionary1 = ibm_db.fetch_assoc(res1)
    texpense = []

    while dictionary1 != False:
        temp = []
        temp.append(dictionary1["TN"])
        temp.append(dictionary1["AMOUNT"])
        texpense.append(temp)
        print(temp)
        dictionary1 = ibm_db.fetch_assoc(res1)
param = "SELECT * FROM Expense WHERE userid = " + str(session['id']) + " AND
DATE(date) = DATE(current timestamp) ORDER BY date DESC"
    res = ibm_db.exec_immediate(ibm_db_conn, param)
    dictionary = ibm_db.fetch_assoc(res)
    expense = []
    while dictionary != False:
        temp = []
        temp.append(dictionary["ID"])
        temp.append(dictionary["USERID"])
        temp.append(dictionary["DATE"])
        temp.append(dictionary["EXPENSENAME"])
        temp.append(dictionary["AMOUNT"])
        temp.append(dictionary["PAYMODE"])
        temp.append(dictionary["CATEGORY"])
        expense.append(temp)
        print(temp)
        dictionary = ibm_db.fetch_assoc(res)


    total=0
    t_food=0
    t_entertainment=0
    t_business=0
    t_rent=0
    t_EMI=0
    t_other=0
```

```python
    for x in expense:
        total += int(x[4])
        if x[6] == "food":
            t_food += int(x[4])

        elif x[6] == "entertainment":
            t_entertainment += int(x[4])

        elif x[6] == "business":
            t_business += int(x[4])
        elif x[6] == "rent":
            t_rent += int(x[4])

        elif x[6] == "EMI":
            t_EMI += int(x[4])

        elif x[6] == "other":
            t_other += int(x[4])


    print(total)

    print(t_food)
    print(t_entertainment)
    print(t_business)
    print(t_rent)
    print(t_EMI)
    print(t_other)



    return render_template("today.html", texpense = texpense, expense =
expense,  total = total ,
                        t_food = t_food,t_entertainment = t_entertainment,
                        t_business = t_business,  t_rent = t_rent,
                        t_EMI =  t_EMI,  t_other =  t_other )



@app.route("/month")
def month():

    param1 = "SELECT DATE(date) as dt, SUM(amount) as tot FROM Expense WHERE
userid = " + str(session['id']) + " AND MONTH(date) = MONTH(current timestamp)
AND YEAR(date) = YEAR(current timestamp) GROUP BY DATE(date) ORDER BY DATE(date)"
    res1 = ibm_db.exec_immediate(ibm_db_conn, param1)
    dictionary1 = ibm_db.fetch_assoc(res1)
```

```python
        texpense = []

        while dictionary1 != False:
            temp = []
            temp.append(dictionary1["DT"])
            temp.append(dictionary1["TOT"])
            texpense.append(temp)
            print(temp)
            dictionary1 = ibm_db.fetch_assoc(res1)


        param = "SELECT * FROM Expense WHERE userid = " + str(session['id']) + " 
AND MONTH(date) = MONTH(current timestamp) AND YEAR(date) = YEAR(current 
timestamp) ORDER BY date DESC"
        res = ibm_db.exec_immediate(ibm_db_conn, param)
        dictionary = ibm_db.fetch_assoc(res)
        expense = []
        while dictionary != False:
            temp = []
            temp.append(dictionary["ID"])
            temp.append(dictionary["USERID"])
            temp.append(dictionary["DATE"])
            temp.append(dictionary["EXPENSENAME"])
            temp.append(dictionary["AMOUNT"])
            temp.append(dictionary["PAYMODE"])
            temp.append(dictionary["CATEGORY"])
            expense.append(temp)
            print(temp)
            dictionary = ibm_db.fetch_assoc(res)


        total=0
        t_food=0
        t_entertainment=0
        t_business=0
        t_rent=0
        t_EMI=0
        t_other=0


        for x in expense:
            total += int(x[4])
            if x[6] == "food":
                t_food += int(x[4])
```

```python
        elif x[6] == "entertainment":
            t_entertainment  += int(x[4])

        elif x[6] == "business":
            t_business  += int(x[4])
        elif x[6] == "rent":
            t_rent  += int(x[4])

        elif x[6] == "EMI":
            t_EMI  += int(x[4])

        elif x[6] == "other":
            t_other  += int(x[4])

    print(total)

    print(t_food)
    print(t_entertainment)
    print(t_business)
    print(t_rent)
    print(t_EMI)
    print(t_other)



    return render_template("today.html", texpense = texpense, expense =
expense,  total = total ,
                        t_food = t_food,t_entertainment = t_entertainment,
                        t_business = t_business,  t_rent = t_rent,
                        t_EMI =  t_EMI,  t_other =  t_other )

@app.route("/year")
def year():


    param1 = "SELECT MONTH(date) as mn, SUM(amount) as tot FROM Expense WHERE
userid = " + str(session['id']) + " AND YEAR(date) = YEAR(current timestamp)
GROUP BY MONTH(date) ORDER BY MONTH(date)"
    res1 = ibm_db.exec_immediate(ibm_db_conn, param1)
    dictionary1 = ibm_db.fetch_assoc(res1)
    texpense = []

    while dictionary1 != False:
        temp = []
        temp.append(dictionary1["MN"])
```

```python
            temp.append(dictionary1["TOT"])
            texpense.append(temp)
            print(temp)
            dictionary1 = ibm_db.fetch_assoc(res1)


        param = "SELECT * FROM Expense WHERE userid = " + str(session['id']) + "
AND YEAR(date) = YEAR(current timestamp) ORDER BY date DESC"
        res = ibm_db.exec_immediate(ibm_db_conn, param)
        dictionary = ibm_db.fetch_assoc(res)
        expense = []
        while dictionary != False:
            temp = []
            temp.append(dictionary["ID"])
            temp.append(dictionary["USERID"])
            temp.append(dictionary["DATE"])
            temp.append(dictionary["EXPENSENAME"])
            temp.append(dictionary["AMOUNT"])
            temp.append(dictionary["PAYMODE"])
            temp.append(dictionary["CATEGORY"])
            expense.append(temp)
            print(temp)
            dictionary = ibm_db.fetch_assoc(res)


        total=0
        t_food=0
        t_entertainment=0
        t_business=0
        t_rent=0
        t_EMI=0
        t_other=0


        for x in expense:
            total += int(x[4])
            if x[6] == "food":
                t_food += int(x[4])

            elif x[6] == "entertainment":
                t_entertainment  += int(x[4])

            elif x[6] == "business":
                t_business  += int(x[4])
            elif x[6] == "rent":
```

```python
                t_rent  += int(x[4])

            elif x[6] == "EMI":
                t_EMI  += int(x[4])

            elif x[6] == "other":
                t_other  += int(x[4])

    print(total)

    print(t_food)
    print(t_entertainment)
    print(t_business)
    print(t_rent)
    print(t_EMI)
    print(t_other)



    return render_template("today.html", texpense = texpense, expense =
expense,  total = total ,
                        t_food = t_food,t_entertainment = t_entertainment,
                        t_business = t_business,  t_rent = t_rent,
                        t_EMI =  t_EMI,  t_other =  t_other )

#log-out

@app.route('/logout')

def logout():
    session.pop('loggedin', None)
    session.pop('id', None)
    session.pop('username', None)
    session.pop('email', None)
    return render_template('home.html')



app.run(debug=True)
```

7.2 FEATURES 2

Email notifications will be sent to the users once they cross the expenditure limit through send grid mail system. Most notifications are transactional, meaning a recipient's action or account activity triggers them. But some notifications are marketing related, encouraging the recipient to take a specific action. Ecommerce product notifications inform recipients about new products or discounts. Plus, unlike general marketing emails, these are highly personalized and focus on a single product. For example, if a customer views an item on your website and that item goes on sale, you can send the customer a notification to let them know this is the best time to buy. Users can also opt into receiving notifications when an out-of-stock item is back in stock.

Notification emails tend to perform well because the content is highly relevant to the recipient. But the only way for the recipient to know this is if you state the content clearly in the subject line. For example, the subject line "New Sign-in to Your Account" gets straight to the point, letting the user know why you sent this notification.

Sendemail.py

```python
import  smtplib
import sendgrid as sg
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail, Email, To, Content
SUBJECT = "personal expense tracker"
s = smtplib.SMTP('smtp.gmail.com', 587)

def sendmail(TEXT,email):
    from_email = Email("kanishkar@gmail.com")to_email =
    To(email)
    subject = "Sending with SendGrid is Fun"
    content = Content("text/plain",TEXT)
    mail = Mail(from_email, to_email, subject, content)
    try:
        sg=SendGridAPIClient('SG.3wVvuDLTQ-
aoSvEgQ8xy7w.2Mp38QJmhoG_p09E3x7HA2OAGRCx9TD7QTenuEHfp9k')
        response = sg.send(mail)
        print(response.status_code)
```

```
    print(response.body)
    print(response.headers)
except Exception as e:
    print(e)
```

7.3 DATABASE SCHEMA

Tables :

1) ADMIN

        id INT NOT NULL GENERATED ALWAYS AS IDENTITY,

        username VARCHAR(32) NOT NULL,

        email   VARCHAR(32) NOT NULL,

        password VARCHAR(32) NOT NULL

2) EXPENSE

        id INT NOT NULL GENERATED ALWAYS AS IDENTITY,

        userid INT NOT NULL, date TIMESTAMP(12) NOT NULL,

        expensename VARCHAR(32) NOT NULL,

        amount VARCHAR(32) NOT NULL,

        paymode VARCHAR(32) NOT NULL,

        category VARCHAR(32) NOT NULL

3) LIMIT

        id INT NOT NULL GENERATED ALWAYS AS

        IDENTITY,

        userid VARCHAR(32) NOT NULL,

        limit VARCHAR(32) NOT NULL

8. TESTING

8.1 TEST CASES

Test Case ID	Purpose	TestCases	Result
TC1	Authentication	password with length less than 4 characters	password cannot be less than 4 characters
TC2	Authentication	User name with length less than 2 characters	User name cannot be less than 2 characters
TC3	Authentication	Valid user name with minimum 2 characters	User name accepter
TC4	Authentication	User name is blank	User name cannot be less than 2 characters
TC5	Authentication	Password field blank	User password cannot be empty
TC6	Authentication	Minimum 4 characters valid password	Password accepted
TC7	Authentication	Password and Confirm password did not match	Please enter same password
TC8	Authentication	Confirm Password field bland	Please enter same password

8.2 USER ACCEPTANCE TESTING

Test Case ID	Test Case Description
TC_001	Verify if user is able to order single product.
TC_002	Verify if user is able to order multiple products.
TC_003	Verify if user can apply single or multiple filters
TC_004	Verify if user can apply different sort by
TC_005	Verify if user is able to pay by Master Card
TC_006	Verify if user is able to pay by Debit Card
TC_007	Verify if user is able to pay fully by reward points
TC_008	Verify if user is able to pay partially by reward points

9. RESULTS

9.1 PERFORMANCE METRICS

- Tracking income and expenses: Monitoring the income and tracking all expenditures (through bank accounts, mobile wallets, and credit & debit cards).

- Transaction Receipts: Capture and organize your payment receipts to keep track of your expenditure.

- Organizing Taxes: Import your documents to the expense tracking app, and it will streamline your income and expenses under the appropriate tax categories.

- Payments & Invoices: Accept and pay from credit cards, debit cards, net banking, mobile wallets, and bank transfers, and track the status of your invoices and bills in the mobile app itself. Also, the tracking app sends reminders for payments and automatically matches the payments with invoices.

- Reports: The expense tracking app generates and sends reports to give a detailed insight about profits, losses, budgets, income, balance sheets, etc.,

- E-commerce integration: Integrate your expense tracking app with your eCommerce store and track your sales through payments received via multiple payment methods.

- Vendors and Contractors: Manage and track all the payments to the vendors and contractors added to the mobile app.

- Access control: Increase your team productivity by providing access control to particular users through custom permissions.

- Track Projects: Determine project profitability by tracking labor costs, payroll, expenses, etc., of your ongoing project.

- Inventory tracking: An expense tracking app can do it all. Right from tracking products or the cost of goods, sending alert notifications when the product is running out of stock or the product is not selling, to purchase orders.

- In-depth insights and analytics: Provides in-built tools to generate reports with easy-tounderstand visuals and graphics to gain insights about the performance of your business.

- Recurrent Expenses: Rely on your budgeting app to track, streamline, and automate all the recurrent expenses and remind you on a timely basis.

OUTPUTS:

1.SignUp page

2.login page

3.add expense page
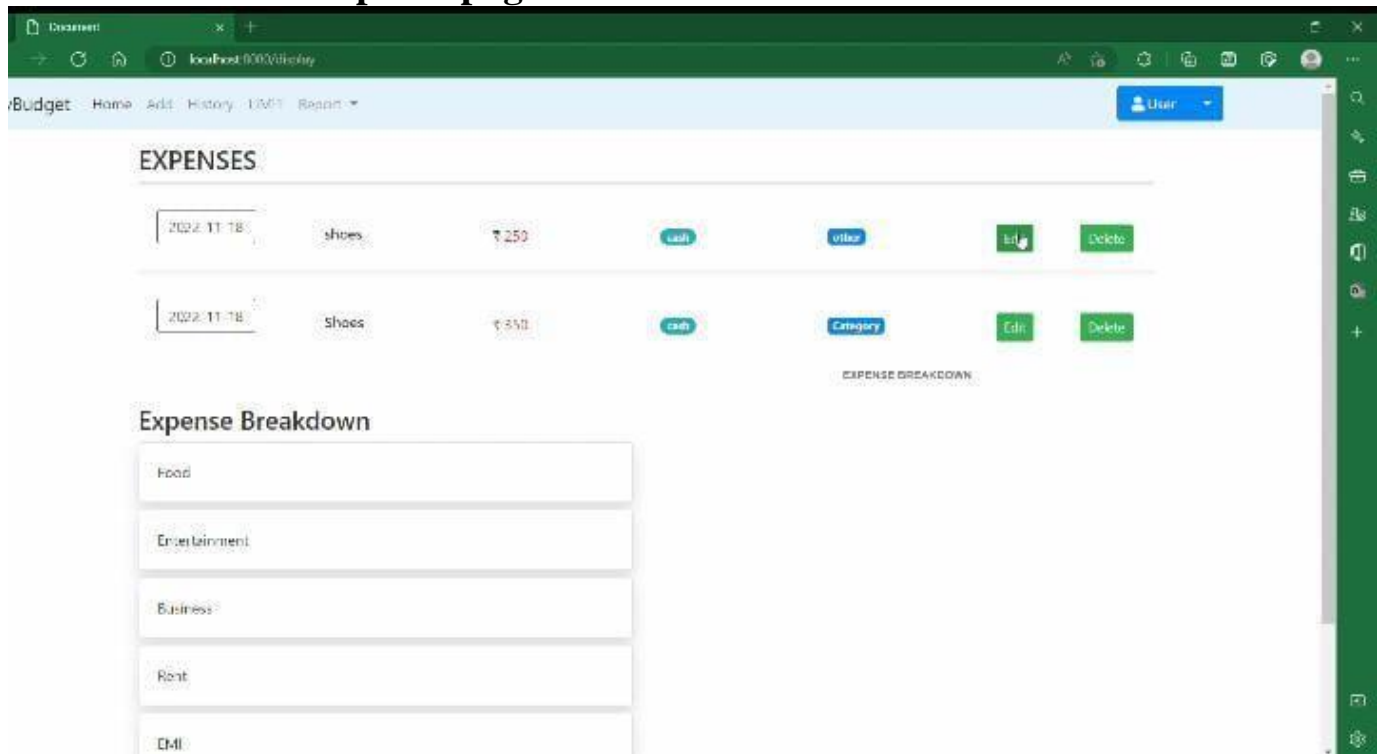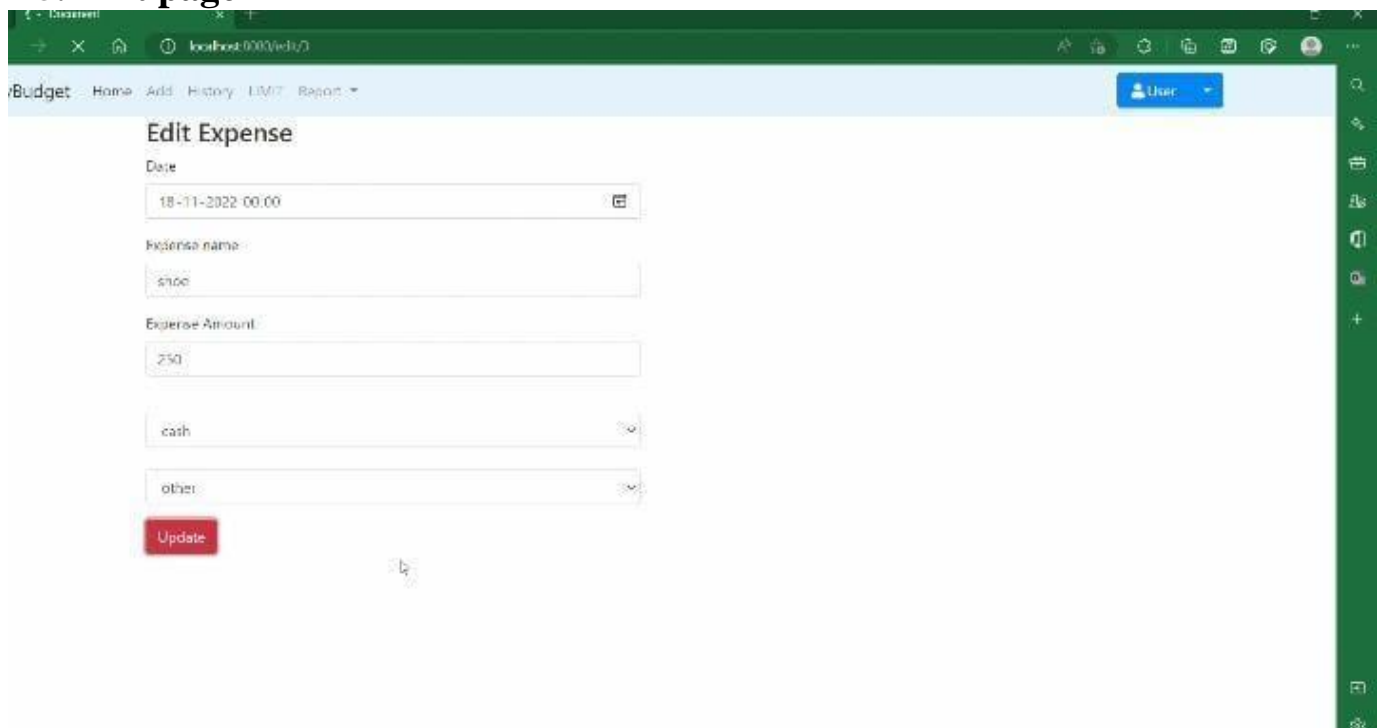
4. Breakdown of expense page

5. limit page

10. ADVANTAGES AND DISADVANTAGES

10.1 ADVANTAGES

- **Daily Expense Tracker**

  An expense manager consolidates all incoming and outgoing transactions, so that you're not found organizing data separately.

- **Keep an Eye on your Finances**

  Request for a customized report of your spending & expenses and keep a close eye on your everyday money.

- **Personal Finance Management**

  An Expense Manager makes your personal money management totally effortless & track your expenses & spend according to the set up monthly budget.

- **Money Saving & Growth**

  It's quite easy to save money using an expense manager. The tracker observes your expenses so that you're aware of what's being spent & take note for future.

- **Expenditure Summary**

  Extract daily, weekly & monthly summary of your spends to line up with your budget goals & achievements.

- **Home Credit Expense Manager** uses new era technology to automatically categorize & provide a 360 degree view of all your income & expenses. It works like a single passbook for all the transactions of money. It's a brilliant step towards futuristic approach in your financial planning & strategy. It's time to march in a forward looking way so that every technology is well-utilized, keeping manual efforts at bay & grow your wealth.

10.2 DISADVANTAGES

- There are many misconceptions about budgeting. Beyond the mindset shift, budgeting isn't always easy for those who are just starting or haven't done it consistently. The pros outweigh the cons, but there may be some disadvantages of budgeting as well.

- Your information is less secure, and probably being used and sold. If the service is free, then the product is you. Mint.com, like other financial apps, is a free service.

- They have to pay their bills somehow, so regardless of what their privacy policy may or may not say, just assume that your spending history and trends are going to be recorded and analyzed, by someone, somewhere.

- Now, you shouldn't have to worry about credit card fraud or identity theft, these companies are large enough and secure enough that you'll never have to worry about something like that. Just recognize that your information, most likely anonymous, will be used and potentially even sold. Personally, I have no problem with that, but if you do, then make sure you avoid these types of services.

11. CONCLUSION

From this project, we are able to manage and keep tracking the daily expenses as well as income. While making this project, we gained a lot of experience of working as a team. We discovered various predicted and unpredicted problems and we enjoyed a lot solving them as a team. We adopted things like video tutorials, text tutorials, internet and learning materials to make our project complete. I personally consider the activity of budgeting and expense tracking as one of those things which has profound benefits in long term. The benefits are so huge that it may lead to financial independence.People who've practiced yoga for an extended period of time, will vouch for its benefits on physical and mental health. "Budgeting and expense tracking" can have similar positive impact on *ones* financial health.

12. FUTURE SCOPE

- Achieve your business goals with a tailored mobile app that perfectly fits your business.
- Scale-up at the pace your business is growing.
- Deliver an outstanding customer experience through additional control over the app.
- Control the security of your business and customer data.
- Open direct marketing channels with no extra costs with methods such as push
- notifications.
- Boost the productivity of all the processes within the organization.
- Increase efficiency and customer satisfaction with an app aligned to their needs.
- Seamlessly integrate with existing infrastructure.
- Ability to provide valuable insights.
- Optimize sales processes to generate more revenue through enhanced data collection.

13. APPENDIX

Source code

The source code has been uploaded in GitHub. To refer the final source code click
' FINAL CODE ' in GitHub

1. Github link: https://github.com/IBM-EPBL/IBM-Project-41059-1660638996
2. Demolink:
https://drive.google.com/file/d/1qa8RFl5ilon_RWMrorAMX1DQed68yPuw/view