**PERSONAL EXPENSE TRACKER APPLICATION**

# A PROJECT REPORT

***Submitted***

***by***

**UDITH RAGAV**

**SRI SAI M**

**SARANKARTHIK V**

**VINOTHKUMAR R**

**TEAM ID :** PNT2022TMID02834

**INDUSTRY MENTOR :** Khusboo

**FACULTY MENTOR :** M.VENGATESHWARAN

**TABLE OF CONTENT**

**CHAPTER NO** **TITLE**

# 1. INTRODUCTION

1.1 Project Overview

1.2 Purpose

# 2. LITERATURE SURVEY

2.1 Existing problem

2.2 References

2.3 Problem Statement Definition

# 3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

3.2 Ideation & Brainstorming

3.3 Proposed Solution

3.4 Problem Solution fit

# 4. REQUIREMENT ANALYSIS

4.1 Functional requirement

4.2 Non-Functional requirements

# 5. PROJECT DESIGN

5.1 Data Flow Diagrams

5.2 Solution & Technical Architecture

5.3 User Stories

# 6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

6.2 Sprint Delivery Schedule

6.3 Reports from JIRA

1. **CODING & SOLUTIONING**

7.1 Feature

# 8. TESTING

8.1 Test Cases

8.2 User Acceptance Testing

**9.** **RESULTS**

# 10 ADVANTAGES & DISADVANTAGES

10.1 Advantages

10.2 Disadvantages

1. **CONCLUSION**
2. **FUTURE SCOPE**

# 13. APPENDIX

13.1 Github Link

13.2 Demo Link

13.3 Sample Code

# 1. INTRODUCTION

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.

## 1.1 Project Overview

The personal expense tracker application is designed to track the users expense on a daily basis. This system splits your income based on your daily expense. If the daily expenses are exceeded, the appliacation sends an alert email to the users mail.The personal expense tracker application produces a report at the end of the month and displays the chart for the expenses.

## 1.1 Purpose

The main purpose of this personal expense tracker application is to reduce the difficulties in managing money in our day to day life. In our daily life cash is the most important component. Most of the people cannot track their expense manually so this motivates the users to use an application that tracks their expenses and set limits for their expenses so that they are well aware of their expenses.

# 2. LITERATURE SURVEY

## 2.1 Existing problem

In the existing system, the data are stored in the local storage of the device and data handling is a tedious process. There is no proper assistance in the current system and virtualization does not provide full interoperability to the user. In this existing system traditional statistical approach is used. Email alert is not sent to the user when he exceeds the limit for the expense. In this existing system, month end statement is in .csv file format.

## 2.2 References

1. [http://expense-manager.com/how-expense software/](http://expense-manager.com/how-expense%20software/)
2. <https://www.splitwise.com/terms>
3. <http://code.google.com/p/socialauthandroid/wiki/Facebook>
4. <http://code.google.com/p/socialauth-android>
5. <https://ijirt.org/master/publishedpaper/IJIRT150860_PAPER.pdf>
6. http://www.appbrain.com/app/expensemanager/ com.expensemanager
7. <http://dspace.daffodilvarsity.edu.bd:8080/handle/123456789/4026>
8. [http://expense-manager.com/how-expense software/](http://expense-manager.com/how-expense%20software/)
9. Donn Felker, “Android Application Development for Dummies”, published by For Dummies, 2010.

[10][https://www.irjet.net/archives/V6/i3/IRJET-V6I31110.pdf.](https://www.irjet.net/archives/V6/i3/IRJET-V6I31110.pdf) [11]https://www.proquest.com/openview/a372033c3cafa03eaa5e18e68e99c86b/1.pdf?p q-origsite=gscholar&cbl=2045096

## 2.3 Problem Statement Definition

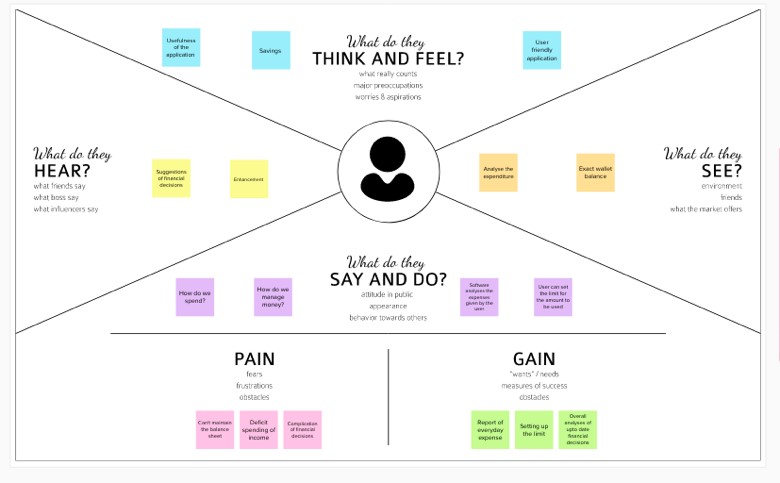
Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert.



# 3. IDEATION & PROPOSED SOLUTION

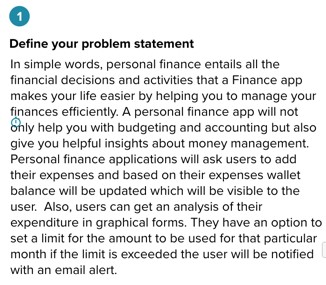
## 3.1 Empathy Map Canvas

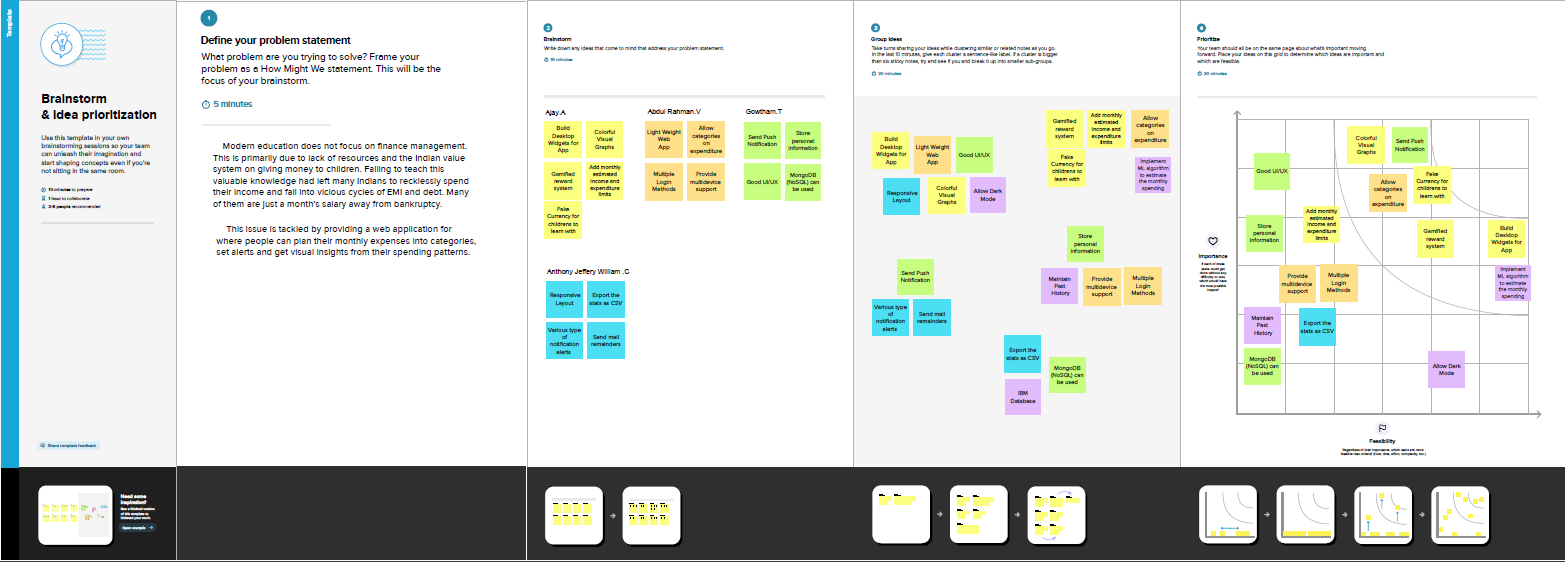
An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behavior’s and attitudes. It is a useful tool to helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user’s perspective along with his or her goals and challenges.



## 3.2 Ideation and Brainstorming

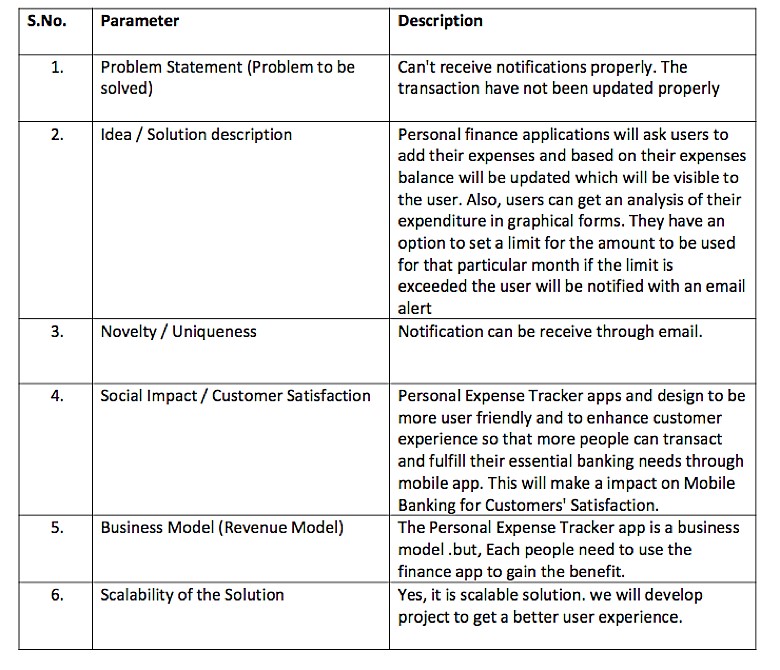
Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich amount of creative solutions.





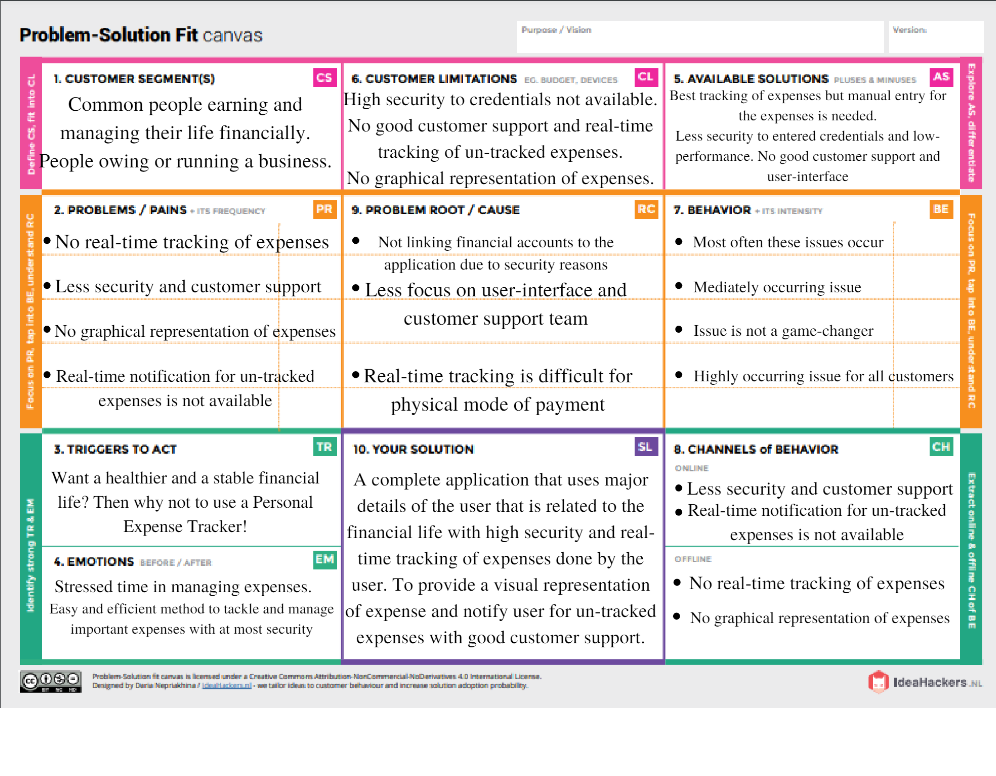
## 3.3 Proposed Solution

In this personal expense tracker project IBM DB2 cloud is used to store the data instead of storing in local storage.Here containerization is a concept that took over virtualization, which allows the user to run the application uniformly, and consistently on any infrastructure using the Docker application. IBM Watson Assistant Chat bot is used to guide the user and explain about the application. In this system project backup details is recorded in IBM Cloud Foundry so incase of any failure, the information will be automatically roll backed to the latest checkpoint. Here our project is built using python flask that allows better scalability to this project. If the user exceeds the limit then he will be sent an alert email stating that he has exceeded his expense limit using Send Grid.



## 3.4 Problem Solution Fit

The Problem Solution Fit is used to find a problem with your customer and that the solution you have realized for it actually solves the customer's problem.



# 4. REQUIREMENT ANALYSIS

## 4.1 Functional requirements

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Functional Requirement**  **(Epic)** | **Sub Requirement (Story / Sub-Task)** |
| FR-1 | User Registration | Registration through Form  Registration through Gmail  Registration through Email Account |
| FR-2 | User Confirmation | Confirmation via Email |
| FR-3 | Calendar | Personal expense tracker application must allow user to add the data to their expenses. |
| FR-4 | Graphical Representation | This application should graphically represent the expense in the form of report. |
| FR-5 | Report Generation | Graphical representation of report must be generated. |
| FR-6 | Category | This application shall allow users to add categories of their expenses. |

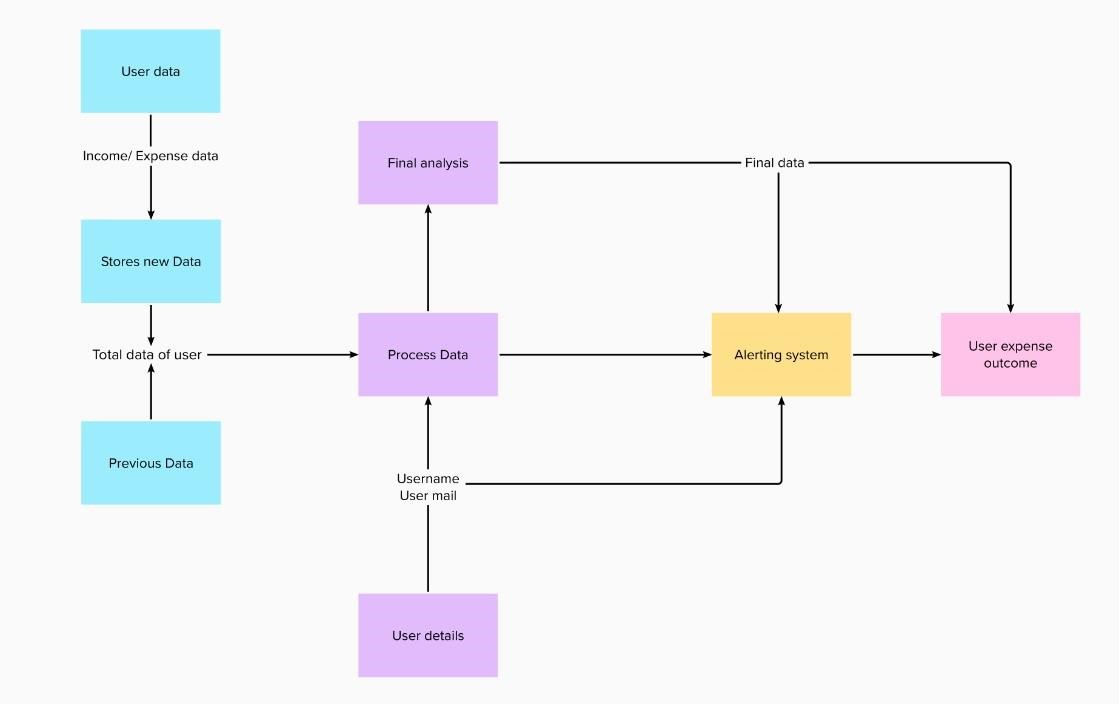
## 4.2 Non-Functional requirements

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | Helps to keep an accurate record and track of their income and expenses easily. |
| NFR-2 | **Security** | We save the password in the encrypted form so it will add more secure to the application user. |
| NFR-3 | **Reliability** | Each data record is stored on a well-built efficient database schema. There is no risk of data loss. |
| NFR-4 | **Performance** | Expense kinds include categories and an option. The system's throughput is boosted because to the lightweight database support. |
| NFR-5 | **Availability** | User can able to access the application with the help of the internet throw the web browser. |
| NFR-6 | **Scalability** | The ability to appropriately handle increasing demands. |

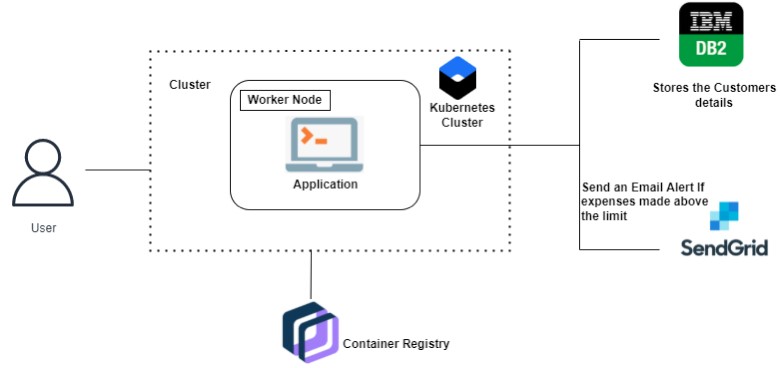
# 5. PROJECT DESIGN

## 5.1 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.



## 5.2 Solution & Technical Architecture



## 5.3 User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional Requireme**  **nt (Epic)** | **User**  **Story**  **Numb**  **er** | **User Story /**  **Task** | **Acceptance**  **criteria** | **Priority** |
| Customer  (web user) | Registration | USN-1 | As a user, I can register for the application by entering my email and confirming my password. | I can access my account/dashboa  rd | High |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | USN-2 | As a user, I will receive a I can receive a confirmation email once I confirmation have registered for the email & click application | I can receive a confirmation email | High |
|  |  | USN-3 | As a user, I can register access the Facebook | I can register through access the  Facebook | Low |
|  | Login | USN-4 | As a user, I can log in application by entering application my email and password. | I can access the application | Medi um |
|  | Dashboard | USN-5 | As a user, I can view my income and expenditure details | I can view my daily expenses. | High |
| Customer  Care  Executive |  | USN-6 | As a customer executive, I can solve the login issue and other issues of the solution at any application | I can provide support | Medi um |
| Administrat or | Application | USN-7 | As an administrato r, I can  upgrade or update the application | I can fix the bug | Medi um |

# 6. PROJECT PLANNING AND SCHEDULING

## 6.1 Sprit Planning and Estimation

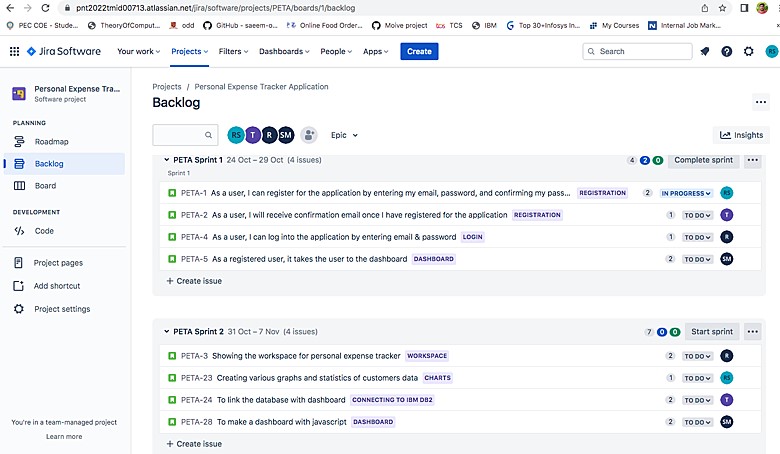
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional**  **Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team**  **Members** |
| Sprint 1 | Registration | USN-1 | As a user, I can register for the application by entering my email, password, and confirming my password. | 2 | High | Sri sai |
| Sprint 2 | Login | USN-2 | As a user, I can log into the application by entering email & password | 2 | High | Udith ragav |
| Sprint 2 | Dashboard | USN-3 | As a user, after logging in, I will be able to enter my budget. | 2 | High | Vinoth kumar |
| Sprint 3 | Dashboard | USN-4 | As a user, I will be able to enter incomes and Expenses | 2 | High | Vinoth kumar |
| Sprint 4 | Dashboard | USN-5 | As a user, I will download my monthly expense reports | 2 | Medium | Udith ragav |
| Sprint 1 | Support | USN-6 | As a user, I will contact customer support for login issues / assistance. | 1 | Low | Sri sai |
| Sprint 1 | Support | USN-7 | As a user, I will contact administrator for critical issues | 1 | Low | Sarankarthik |

## 6.2 Sprit Planning and Estimation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sprint-1 | 20 | 6 Days | 1 Oct 2022 | 6 Oct 2022 | 10 | 6 Oct 2022 |
| Sprint-2 | 20 | 6 Days | 6 Oct 2022 | 11 Oct 2022 | 20 | 11 Oct 2022 |
| Sprint-3 | 20 | 18 Days | 12 Oct 2022 | 30 Oct 2022 | 20 | 30 Oct 2022 |
| Sprint-4 | 20 | 14 Days | 1 Nov 2022 | 14 Nov 2022 | 20 | 14 Nov 2022 |

## 6.3 Reports from JIRA

### 6.3.1 Backlog



# 7. CODING & SOLUTIONING

## 7.1 Features

### 7.1.1 Python Flask

We have used python flask to develop our project. Python flask is a web framework and a python module that lets you develop web applications easily.

### 7.1.2 IBM Cloud DB2

When the new user registers into the application and the details of the user gets stored in IBM Cloud DB2 . We have connected the DB2 with our project using the below code

|  |
| --- |
| app.config['database'] = 'bludb'app.config['hostname'] = '3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud'app.config['port'] = '31498'app.config['protocol'] = 'tcpip'app.config['uid'] = 'sbb93800'app.config['pwd'] = 'wobsVLm6ccFxcNLe'app.config['security'] = 'SSL'try:  mysql = DB2(app)  conn\_str='database=bludb;hostname=2d46b6b4-cbf6-40eb-bbce-  6251e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;port=32328;protocol=tcpip;\  uid=lsc91268;pwd=dlWyz6qJK3v27xP6;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()) |

### 7.1.3 Send Grid

When the new user registers, the confirmation mail is sent to the user’s mail using the SendGrid and when the user exceeds the limit the alert email is send to the user using this SendGrid service.

|  |
| --- |
| app = Flask(\_\_name\_\_)app.config['SECRET\_KEY'] = 'top-secret!'app.config['MAIL\_SERVER'] = 'smtp.sendgrid.net'app.config['MAIL\_PORT'] = 587app.config['MAIL\_USE\_TLS'] = Trueapp.config['MAIL\_USERNAME'] = 'apikey'  app.config['MAIL\_PASSWORD'] = 'SG.PU\_eO2bJTI-HAnjene8ngw.P8zB2XEy14FM4Efn0wTV-  5JG98963QWXKZZza\_bugb8' |

app.config['MAIL\_DEFAULT\_SENDER'] = 'tarunvinodh@gmail.com'mail = Mail(app)

|  |
| --- |
| total=0  forxinexpense:  total += x[4]  param = "SELECT id, limitss FROM limits 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 = 0  whiledictionary != False:  temp = []  temp.append(dictionary["LIMITSS"])  row.append(temp)  dictionary = ibm\_db.fetch\_assoc(res)  s = temp[0]  iftotal>int(s):  msg = "Hello " + session['username'] + " , " + "you have crossed the monthly limit of Rs. " + str(s) + "/- !!!" + "\n" + "Thank you, " + "\n" + "Team Personal Expense Tracker."  sendmail(msg,session['email']) returnredirect("/display") |

|  |
| --- |
| importsmtplibimport sendgrid assgimportos  from sendgrid.helpers.mail import Mail, Email, To, ContentSUBJECT = "expense tracker"s = smtplib.SMTP('smtp.gmail.com', 587)  defsendmail(TEXT,email):  print("sorry we cant process your candidature")  s = smtplib.SMTP('smtp.gmail.com', 587)  s.starttls()  s.login("demo123demo987@gmail.com", "taryluhlooidfwvj")  message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT)  s.sendmail("demo123demo987@gmail.com", email, message)  s.quit() |

### 7.1.4 IBM Watson Assistant Chatbot

We have integrated IBM Watson Assistant Chatbot. Here the users can know about the personal expense tracker application using the information given by the bot.

### 7.1.5 Deploying flask app in Docker

We have deployed our flask app in Docker where they package all the code, libraries, and dependencies together to make it possible for multiple containers to run in the same host and we can run our flask app using this Docker Desktop.

FROM python:3.6

WORKDIR /app

ADD . /app

COPY requirements.txt /app

RUN python3 -m pip install -r requirements.txt

RUN python3 -m pip install ibm\_db

EXPOSE 5000

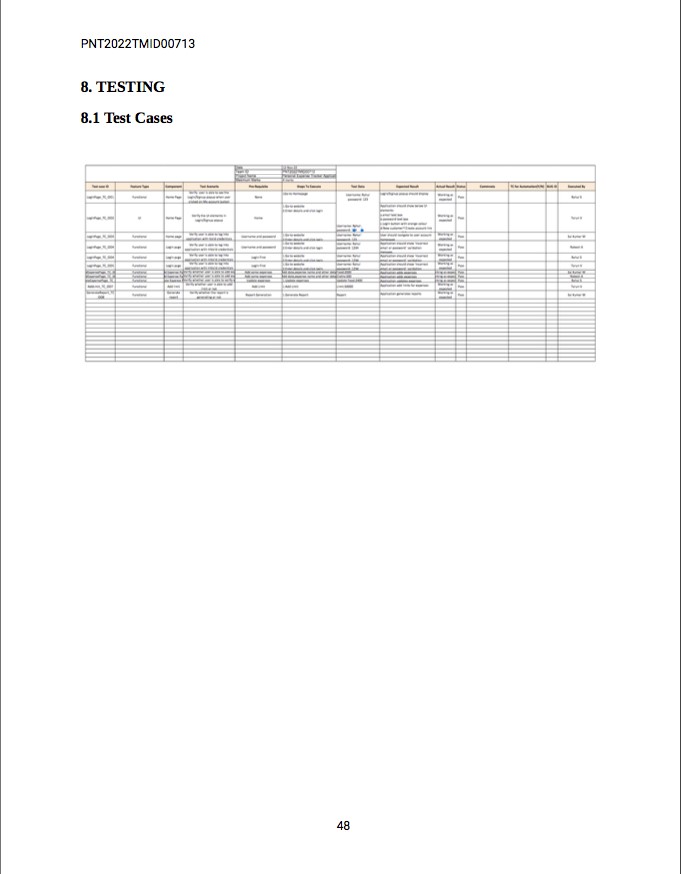
CMD ["python","app.py"]

### 7.1.6 IBM Cloud Container Registry

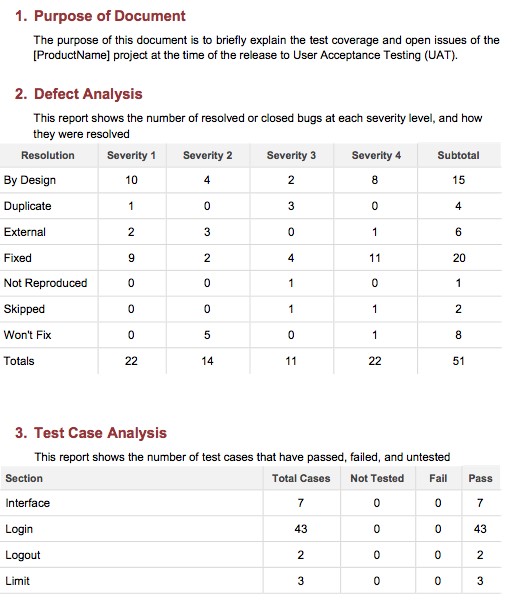
We have deployed our app as Docker image at IBM Cloud Registry.

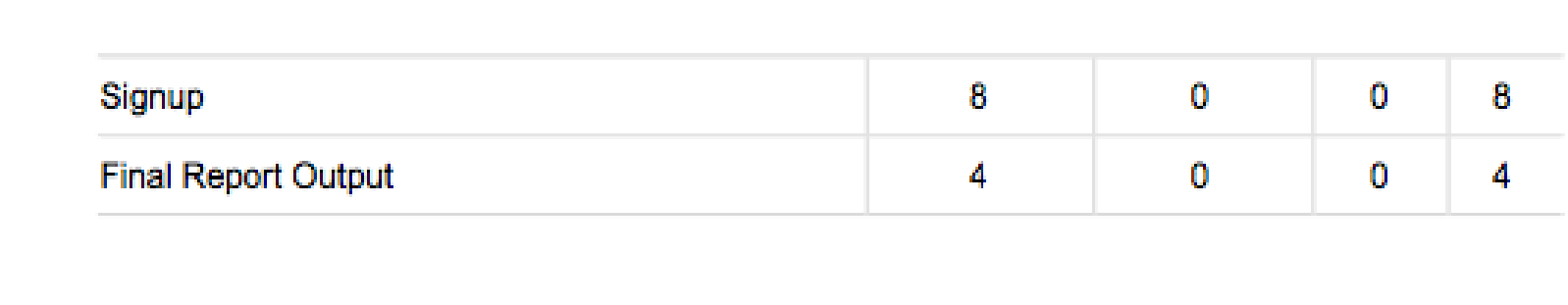
### 7.1.7 Kubernetes

These containers are managed by Kubernetes which automates the operational tasks of the container.



## 8.2 User Acceptance Testing





# 9. RESULTS

The new system has overcome most of the limitations of the existing system and works according to the design specification given. The project what we have developed is work more efficient than the other income and expense tracker. The project successfully avoids the manual calculation for avoiding calculating the income and expense per month. The modules are developed with efficient and also in an attractive manner. The developed systems dispense the problem and meet the needs of by providing reliable and comprehensive information. All the requirements projected by the user have been met by the system. The newly developed system consumes less processing time and all the details are updated and processed immediately.

# 10. ADVANTAGES & DISADVANTAGES

## 10.1 Advantages

* User can have a control over their money and expenses.
* Users are alerted with an email when they exceed their limit.
* Reports are generated based on the users expenses.

## 10.2 Disadvantages

* Less Secured
* Limited Accessbility

# 11. CONCLUSION

Personal Expense Tracker Application is an web based application. We created this application so that a user can accurately calculate his daily cost. Using this application, the user will see the amount of his income and how much a user is spending, and a notification will be sent to the user's if he exceeds the limit and also report is generated.

# 12. FUTURE SCOPE

Now in our application we covered almost all features but in future we will add some more futures. The features are below

* Multiple account support.
* Include currency converter.

1. **APPENDIX13.1 Github Link**https://github.com/IBM-EPBL/IBM-Project-4220-1658724565

## 13.3 Sample Code

**app.py**

from flask import Flask, render\_template, request, redirect, session

from flask\_db2 import DB2

import ibm\_db

import ibm\_db\_dbi

from sendemail import sendgridmail,sendmail

from flask\_mail import Mail, Message

import os

app = Flask(\_\_name\_\_)

app.secret\_key = 'a'

"""

dsn\_hostname = "3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud"

dsn\_uid = "sbb93800"

dsn\_pwd = "wobsVLm6ccFxcNLe"

dsn\_driver = "{IBM DB2 ODBC DRIVER}"

dsn\_database = "bludb"

dsn\_port = "31498"

dsn\_protocol = "tcpip"

dsn = (

"DRIVER={0};"

"DATABASE={1};"

"HOSTNAME={2};"

"PORT={3};"

"PROTOCOL={4};"

"UID={5};"

"PWD={6};"

).format(dsn\_driver, dsn\_database, dsn\_hostname, dsn\_port, dsn\_protocol, dsn\_uid, dsn\_pwd)

"""

# app.config['DB2\_DRIVER'] = '{IBM DB2 ODBC DRIVER}'

app.config['database'] = 'bludb'

app.config['hostname'] = '3883e7e4-18f5-4afe-be8c-fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud'

app.config['port'] = '31498'

app.config['protocol'] = 'tcpip'

app.config['uid'] = 'sbb93800'

app.config['pwd'] = 'wobsVLm6ccFxcNLe'

app.config['security'] = 'SSL'

try:

mysql = DB2(app)

conn\_str='database=bludb;hostname=2d46b6b4-cbf6-40eb-bbce-6251e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;port=32328;protocol=tcpip;\

uid=lsc91268;pwd=dlWyz6qJK3v27xP6;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("/sign")

def signup():

return render\_template("signup.html")

@app.route('/register', methods =['GET', 'POST'])

def register():

msg = ''

print("Break point1")

if request.method == 'POST' :

username = request.form['username']

email = request.form['email']

password = request.form['password']

print("Break point2" + "name: " + username + "------" + email + "------" + password)

try:

print("Break point3")

connectionID = ibm\_db\_dbi.connect(conn\_str, '', '')

cursor = connectionID.cursor()

print("Break point4")

except:

print("No connection Established")

print("Break point5")

sql = "SELECT \* FROM register WHERE username = ?"

stmt = ibm\_db.prepare(ibm\_db\_conn, sql)

ibm\_db.bind\_param(stmt, 1, username)

ibm\_db.execute(stmt)

result = ibm\_db.execute(stmt)

print(result)

account = ibm\_db.fetch\_row(stmt)

print(account)

param = "SELECT \* FROM register WHERE username = " + "\'" + username + "\'"

res = ibm\_db.exec\_immediate(ibm\_db\_conn, param)

print("---- ")

dictionary = ibm\_db.fetch\_assoc(res)

while dictionary != False:

print("The ID is : ", dictionary["USERNAME"])

dictionary = ibm\_db.fetch\_assoc(res)

print("break point 6")

if account:

msg = 'Username already exists !'

elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):

msg = 'Invalid email address !'

elif not re.match(r'[A-Za-z0-9]+', username):

msg = 'name must contain only characters and numbers !'

else:

sql2 = "INSERT INTO register (username, email,password) VALUES (?, ?, ?)"

stmt2 = ibm\_db.prepare(ibm\_db\_conn, sql2)

ibm\_db.bind\_param(stmt2, 1, username)

ibm\_db.bind\_param(stmt2, 2, email)

ibm\_db.bind\_param(stmt2, 3, password)

ibm\_db.execute(stmt2)

msg = 'You have successfully registered !'

return render\_template('signup.html', msg = msg)

#LOGIN--PAGE

@app.route("/signin")

def signin():

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 register 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 register WHERE username = " + "\'" + username + "\'" + " and password = " + "\'" + password + "\'"

res = ibm\_db.exec\_immediate(ibm\_db\_conn, param)

dictionary = ibm\_db.fetch\_assoc(res)

if account:

session['loggedin'] = True

session['id'] = dictionary["ID"]

userid = dictionary["ID"]

session['username'] = dictionary["USERNAME"]

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

return redirect('/home')

else:

msg = 'Incorrect username / password !'

return render\_template('login.html', msg = msg)

#ADDING----DATA

@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 expenses (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")

# email part

param = "SELECT \* FROM expenses 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(resi)

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

for x in expense:

total += x[4]

param = "SELECT id, limitss FROM limits 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 = 0

while dictionary != False:

temp = []

temp.append(dictionary["LIMITSS"])

row.append(temp)

dictionary = ibm\_db.fetch\_assoc(res)

s = temp[0]

if total > int(s):

msg = "Hello " + session['username'] + " , " + "you have crossed the monthly limit of Rs. " + str(s) + "/- !!!" + "\n" + "Thank you, " + "\n" + "Team Personal Expense Tracker."

sendmail(msg,session['email'])

return redirect("/display")

#DISPLAY---graph

@app.route("/display")

def display():

print(session["username"],session['id'])

param = "SELECT \* FROM expenses 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 expenses 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 expenses WHERE id = " + id

res = ibm\_db.exec\_immediate(ibm\_db\_conn, param)

dictionary = ibm\_db.fetch\_assoc(res)

row = []

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"])

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 expenses 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")

#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 limits (userid, limitss) 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, limitss FROM limits 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)

while dictionary != False:

temp.append(dictionary["LIMITSS"])

row.append(temp)

dictionary = ibm\_db.fetch\_assoc(res)

s = temp[0]

return render\_template("limit.html" , y= s)

#REPORT

@app.route("/today")

def today():

param1 = "SELECT TIME(date) as tn, amount FROM expenses 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 expenses 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

for x in expense:

total += x[4]

if x[6] == "food":

t\_food += x[4]

elif x[6] == "entertainment":

t\_entertainment += x[4]

elif x[6] == "business":

t\_business += x[4]

elif x[6] == "rent":

t\_rent += x[4]

elif x[6] == "EMI":

t\_EMI += x[4]

elif x[6] == "other":

t\_other += 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 expenses 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)

texpense = []

while dictionary1 != False:

temp.append(dictionary1["DT"])

temp.append(dictionary1["TOT"])

texpense.append(temp)

print(temp)

dictionary1 = ibm\_db.fetch\_assoc(res1)

param = "SELECT \* FROM expenses 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 += x[4]

if x[6] == "food":

t\_food += x[4]

elif x[6] == "entertainment":

t\_entertainment += x[4]

elif x[6] == "business":

t\_business += x[4]

elif x[6] == "rent":

t\_rent += x[4]

elif x[6] == "EMI":

t\_EMI += x[4]

elif x[6] == "other":

t\_other += 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 expenses 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"])

temp.append(dictionary1["TOT"])

texpense.append(temp)

print(temp)

dictionary1 = ibm\_db.fetch\_assoc(res1)

param = "SELECT \* FROM expenses 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 += x[4]

if x[6] == "food":

t\_food += x[4]

elif x[6] == "entertainment":

t\_entertainment += x[4]

elif x[6] == "business":

t\_business += x[4]

elif x[6] == "rent":

t\_rent += x[4]

elif x[6] == "EMI":

t\_EMI += x[4]

elif x[6] == "other":

t\_other += 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')

port = os.getenv('VCAP\_APP\_PORT', '8080')

if \_\_name\_\_ == "\_\_main\_\_":

app.secret\_key = os.urandom(12)

app.run(debug=True, host='0.0.0.0', port=port)

# Home.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"/>

<meta name="viewport"content="width=device-width, initial-scale=1.0" />

<link rel="stylesheet"href="..\static\css\home.css" />

<link rel="icon"type="image/x-icon" href="logo.png" />

<title>Personal Expense Tracker</title>

</head>

<body>

<script>

window.watsonAssistantChatOptions = {

integrationID: "28378cac-2276-4a28-8b4a-b60ad3b6cf4c", // The ID of this integration.

region: "au-syd", // The region your integration is hosted in.

serviceInstanceID: "1970e6fb-5cd5-41ae-9ff3-b10f36e2cf34", // The ID of your service instance.

onLoad: function (instance) {

instance.render();

},

};

setTimeout(function () {

const t = document.createElement("script");

t.src =

"https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +

(window.watsonAssistantChatOptions.clientVersion || "latest") +

"/WatsonAssistantChatEntry.js";

document.head.appendChild(t);

});

</script>

<!-- Header -->

<section id="header">

<div class="header container">

<div class="nav-bar">

<div class="brand">

<a href="#hero">

<h1>Personal Expense Tracker</h1>

</a>

</div>

<div class="nav-list">

<div class="hamburger">

<div class="bar"></div>

</div>

<ul>

<li><a href="#hero"data-after="Home">Home</a></li>

<li><a href="#services"data-after="Service">Services</a></li>

<li><a href="/signin"data-after="Login">Login</a></li>

</ul>

</div>

</div>

</div>

</section>

<!-- End Header -->

<!-- Hero Section -->

<section id="hero">

<div class="hero container">

<div>

<h1>Welcome to</h1>

<h1>Personal Expense Tracker</h1>

<a href="/signup"type="button" class="but">Sign-up</a>

</div>

</div>

</section>

<!-- End Hero Section -->

<!-- Service Section -->

<section id="services">

<div class="services container">

<div class="service-top">

<h1 class="section-title">Our Servces</h1>

</div>

<div class="service-bottom">

<div class="service-item">

<h2>Reciept Management</h2>

<p>

Tired of losing your business expense receipts? This helps you

automatically track them through features like advanced autoscan.

Save time, and spare yourself the hassle of manually sorting and

keeping track of paper receipts.

</p>

</div>

<div class="service-item">

<h2>Expense Management</h2>

<p>

It offers you robust features to upload any business charge you

encounter, saving you time, money, and stress. Never allow another

expense to go unaccounted for.

</p>

</div>

<div class="service-item">

<h2>Expense Reports</h2>

<p>

Make employees look forward to adding expenses to a report and

submitting it for approval.Here you can make your expense report

management process a breeze for your entire organization.

</p>

</div>

</div>

</div>

</section>

<!-- End Service Section -->

<!-- Footer -->

<section id="footer">

<div class="footer container">

<div class="brand">

<h1>Personal Expense Tracker</h1>

</div>

<h2>Your Finance in our Hands</h2>

</div>

</section>

<!-- End Footer -->

<script src="..\static\js\home.js"></script>

</body>

</html>

# Login.html

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

<link rel="stylesheet"type="text/css" href="..\static\css\login.css" />

<link

href="https://fonts.googleapis.com/css?family=Poppins:600&display=swap"

rel="stylesheet"

/>

<script src="https://kit.fontawesome.com/a81368914c.js"></script>

<meta name="viewport"content="width=device-width, initial-scale=1" />

</head>

<style></style>

<body>

<script>

window.watsonAssistantChatOptions = {

integrationID: "28378cac-2276-4a28-8b4a-b60ad3b6cf4c", // The ID of this integration.

region: "au-syd", // The region your integration is hosted in.

serviceInstanceID: "1970e6fb-5cd5-41ae-9ff3-b10f36e2cf34", // The ID of your service instance.

onLoad: function (instance) {

instance.render();

},

};

setTimeout(function () {

const t = document.createElement("script");

t.src =

"https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +

(window.watsonAssistantChatOptions.clientVersion || "latest") +

"/WatsonAssistantChatEntry.js";

document.head.appendChild(t);

});

</script>

<div class="container">

<div class="img">

<div id="png"><a href="/"title="HOME"></a></div>

</div>

<div class="login-content">

<form action="/login"method="POST">

<h2 class="title">Welcome</h2>

<br />

<div class="input-div one">

<div class="i">

<i class="fas fa-user"></i>

</div>

<div class="div">

<h5>Username</h5>

<input type="text"name="username" class="input" required />

</div>

</div>

<div class="input-div pass">

<div class="i">

<i class="fas fa-lock"></i>

</div>

<div class="div">

<h5>Password</h5>

<input type="password"name="password" class="input" required />

</div>

</div>

<a href="#">Forgot Password?</a>

<input type="submit"class="btn" value="Login" />

<br /><br /><br />

<div class="app">

<b>Don't have an account?</b>

<a class="app1"href="\signup">Register</a>

</div>

</form>

</div>

</div>

<script type="text/javascript" src="..\static\js\login.js"></script>

</body>

</html>

# Signup.html

<html>

<head>

<meta charset="utf-8">

<title>Register</title>

<link href="..\static\css\signup.css" rel="stylesheet">

<script src="https://kit.fontawesome.com/a81368914c.js"></script>

<link rel="stylesheet"href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity="sha384-

Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">

</head>

<body>

<script>

window.watsonAssistantChatOptions = {

integrationID: "28378cac-2276-4a28-8b4a-b60ad3b6cf4c", // The ID of this integration.

region: "au-syd", // The region your integration is hosted in.

serviceInstanceID: "1970e6fb-5cd5-41ae-9ff3-b10f36e2cf34", // The ID of your service instance.

onLoad: function (instance) {

instance.render();

},

};

setTimeout(function () {

const t = document.createElement("script");

t.src =

"https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +

(window.watsonAssistantChatOptions.clientVersion || "latest") +

"/WatsonAssistantChatEntry.js";

document.head.appendChild(t);

});

</script>

<!--container---------------------->

<div class="container">

<!--sign-up-box-container--->

<!--heading-->

<form action="/register"method="post">

<h1 class="heading">Register</h1>

<!--name-box-->

<div class="text">

<img height="20px"src="..\static\images\user.png" />

<input placeholder="Name"type="text" name="username"/>

</div>

<!--Email-box-->

<div class="text">

<img height="12px"src="..\static\images/email.png" />

<input placeholder=" Example@gmail.com" type="email" name="email"" />

</div>

<!--Password-box-->

<div class="text">

<img height="20px"src="..\static\images\password.png" />

<input placeholder=" Password"type="password" name="password"/>

</div>

<!--trems-->

<!--button-->

<div class="toop">

<button type="submit"class="btn btn-primary" >CREATE ACCOUNT</button> </div>

</form>

<!--sign-in-->

<div class="t"><p class="conditions"id="p3">Already have an account <a href="/login">Sign in</a></p> </div></div>

</div>

<!--text-container-->

<div class="text-container">

</div>

</div>

</body>

</html>

# Docker file

FROM python:3.6

WORKDIR /app

ADD . /app

COPY requirements.txt /app

RUN python3 -m pip install -r requirements.txt

RUN python3 -m pip install ibm\_db

EXPOSE 5000

CMD ["python","app.py"]

# Deployment.yaml

|  |
| --- |
| apiVersion: apps/v1kind: Deploymentmetadata:  name: flask-node-deploymentspec:  replicas: 1  selector:  matchLabels:  app: flasknode  template:  metadata:  labels:  app: flasknode  spec: |

containers:

* name: flasknode

image: icr.io/peta-muni/docker\_personalexpensetracker\_muni

imagePullPolicy: Always

ports:

* containerPort: 5000

# Sendmail.py

|  |
| --- |
| importsmtplibimport sendgrid assgimportos  from sendgrid.helpers.mail import Mail, Email, To, ContentSUBJECT = "expense tracker"s = smtplib.SMTP('smtp.gmail.com', 587)  defsendmail(TEXT,email):  print("sorry we cant process your candidature")  s = smtplib.SMTP('smtp.gmail.com', 587)  s.starttls()  s.login("demo123demo987@gmail.com", "taryluhlooidfwvj")  message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT)  s.sendmail("demo123demo987@gmail.com", email, message)  s.quit() |

Github:

<https://github.com/IBM-EPBL/IBM-Project-24457-1659943142>

Demo :

https://drive.google.com/file/d/15sjS0NPj8EfNO8Kxk515dqBLivY8OZqg/view?usp=drivesdk