**PERSONAL EXPENSE**

**TRACKER APPLICATION**

[**IBM-Project-23819-1659931639**](https://github.com/IBM-EPBL/IBM-Project-23819-1659931639)

**NALAIYA THIRAN PROJECT**

**BASED LEARNING ON**

**PROFESSIONAL**

**READLINESS FOR**

**INNOVATION,**

**EMPLOYNMENT AND**

**ENTERPRENEURSHIP**

**A PROJECT REPORT**

**BY**

**AAKASH R(111719205002)**

**MEGARAJ S(111719205025)**

**ASHIK ESWAR (111719205007)**

**LOKESHWAR B(111719205021)**

**BACHELOR OF TECHNOLOGY IN**

**INFORMATION TECHNOLOGY**

**RMK ENGINEERING COLLEGE,CHENNAI -601206**

# INDEX

## 1. INTRODUCTION

1. Project Overview
2. Purpose

## 2. LITERATURE SURVEY

1. Existing problem
2. References
3. Problem Statement Definition

## 3. IDEATION & PROPOSED SOLUTION

1. Empathy Map Canvas
2. Ideation & Brainstorming
3. Proposed Solution
4. Problem Solution fit

## 4. REQUIREMENT ANALYSIS

1. Functional requirement
2. Non-Functional requirements

## 2. PROJECT DESIGN

1. Data Flow Diagrams
2. Solution & Technical Architecture
3. User Stories

## 2. PROJECT PLANNING & SCHEDULING

1. Sprint Planning & Estimation
2. Sprint Delivery Schedule
3. Reports from JIRA

## 2. CODING & SOLUTIONING (Explain the features added in the project along with code)

1. Feature 1
2. Feature 2
3. Database Schema (if Applicable)

## 8. TESTING

1. Test Cases
2. User Acceptance Testing

2. **RESULTS**

a. Performance Metrics

1. **ADVANTAGES & DISADVANTAGES**

1. **CONCLUSION**

1. **FUTURE SCOPE**

## 13. APPENDIX

Source Code

GitHub & Project Demo Link

# 1.INTRODUCTION

## a.Project Overview

TEAM ID : PNT2022TMID09631

INDUSTRY MENTOR : Kusboo FACULTY MENTOR : K Johny Elma

**Skills Required:**

IBM Cloud, HTML, Javascript, IBM

Cloud Object Storage, Python- Flask,

Kubernetes, Docker, IBM DB2, IBM

Container Registry

1. **INTRODUCTION**

1. Project Overview

This project is based on expense tracking.This project aims to create an easy,faster and smooth cloud application .For better expense tracking we developed our project that will help the users a lot.Most of the people cannot track their expenses and income leading to facing money crisis,so this application can help people to track their expense day to day and make life stress free.Money is the most valuable portion of our daily life and without money we will not last one day on earth.So using the daily expense tracker application is important to lead a happy family.It helps the user to avoid unexpected expenses and bad financial situations.It will save time and provide a responsible lifestyle.

1. Purpose

Personal finance management is an important part of people’slives. Ho wever, everyonedoes 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 managerand money manager,an expense trackeris 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 towardsthe end of the month t hey don’thave sufficient money to meet theirneeds. While this problem can arise due to low sal ary, invariably it is due to poor money management skills.

People tend to overspend without realizing, and this can prove to be disastrous. Using a daily

expense managercan help you keep track of how much you spend eve ry day and on what. At the

endof the month, you will have a clear picturewhere your moneyis goin

g. This is one of the best ways to get your expenses under control and bring some semblance of order to your finances.Today, there are several expense manager applications in the market. Some are paid managers while others are free. Even banks like ICICI offer their customers expense tracker to

helpthem out. Beforeyou decide to go in for a money manager,it is imp ortant to decidethe type you want.

2. **LITERATURE SURVEY**

## a.Existing problem

In a study conducted by Forresterin 2016 surveyingsmall and medium busin esses (SMBs) across the world, 56% companies reported expense management as being the biggest challenge for their finance departments.

In another survey conducted by Levvel Research in 2018 in North

America, respondents

reported the following pain points in expense management before adopting automation:

i. Manual entry and routing of expense reports (62%)

ii. Lack of visibility into spend data (42%)

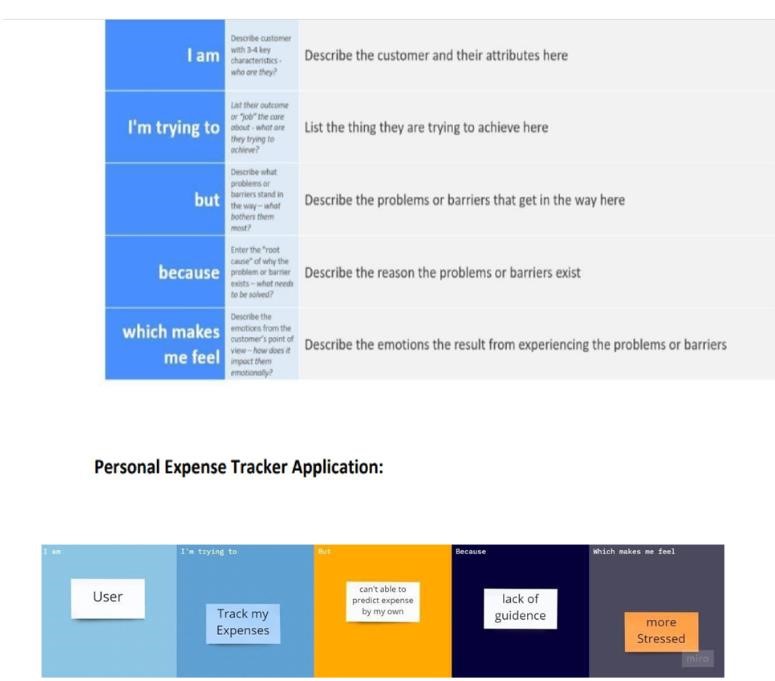
iii. Inability to enforce travel policies (29%)

iv. Lost expense reports (24%)

v. Lengthy expense approval system and reimbursement cycles(23%)

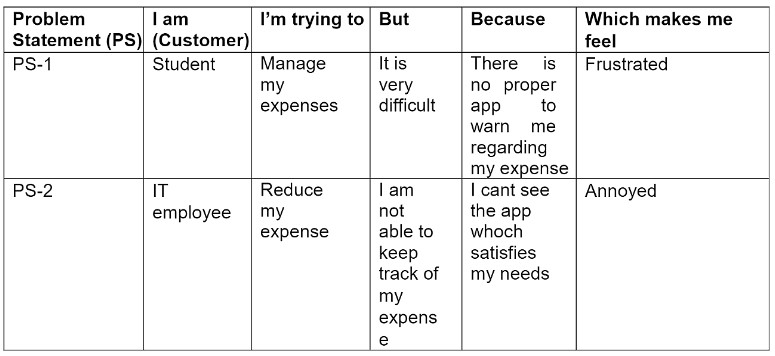
## b. References

3.Problem Statement Definition



**Customer Problem Statement :**

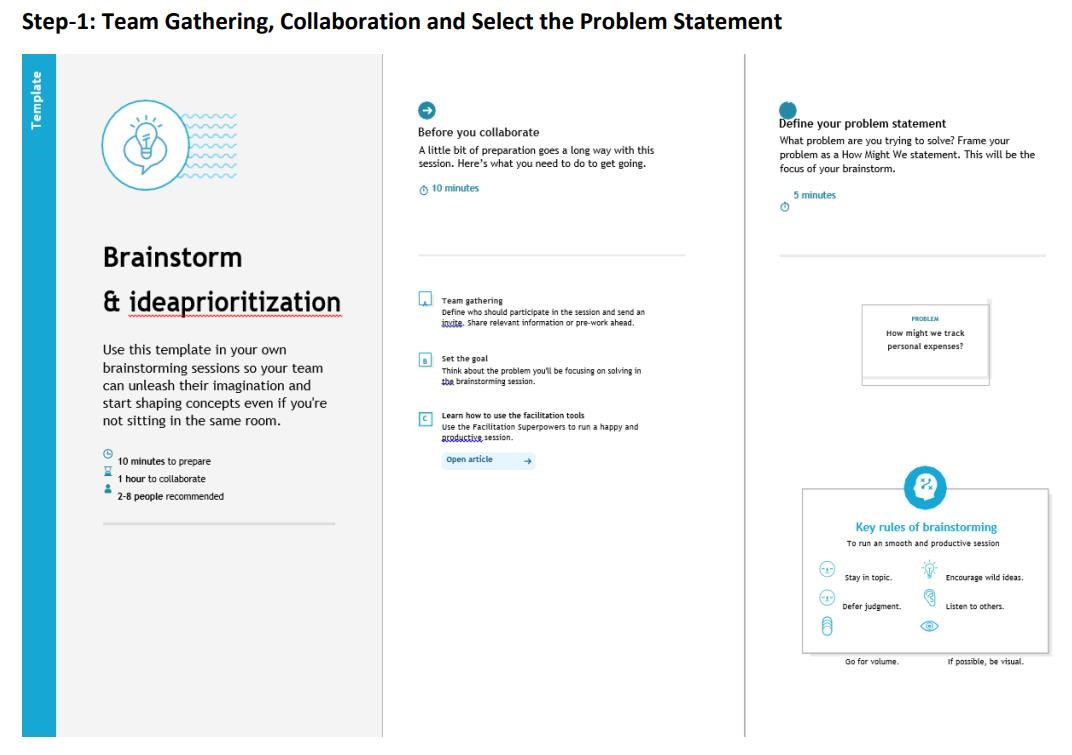
A well-articulated customer problem statement allows us to find the ideal solution for the challenges our customers face. Throughout the process, you’ll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

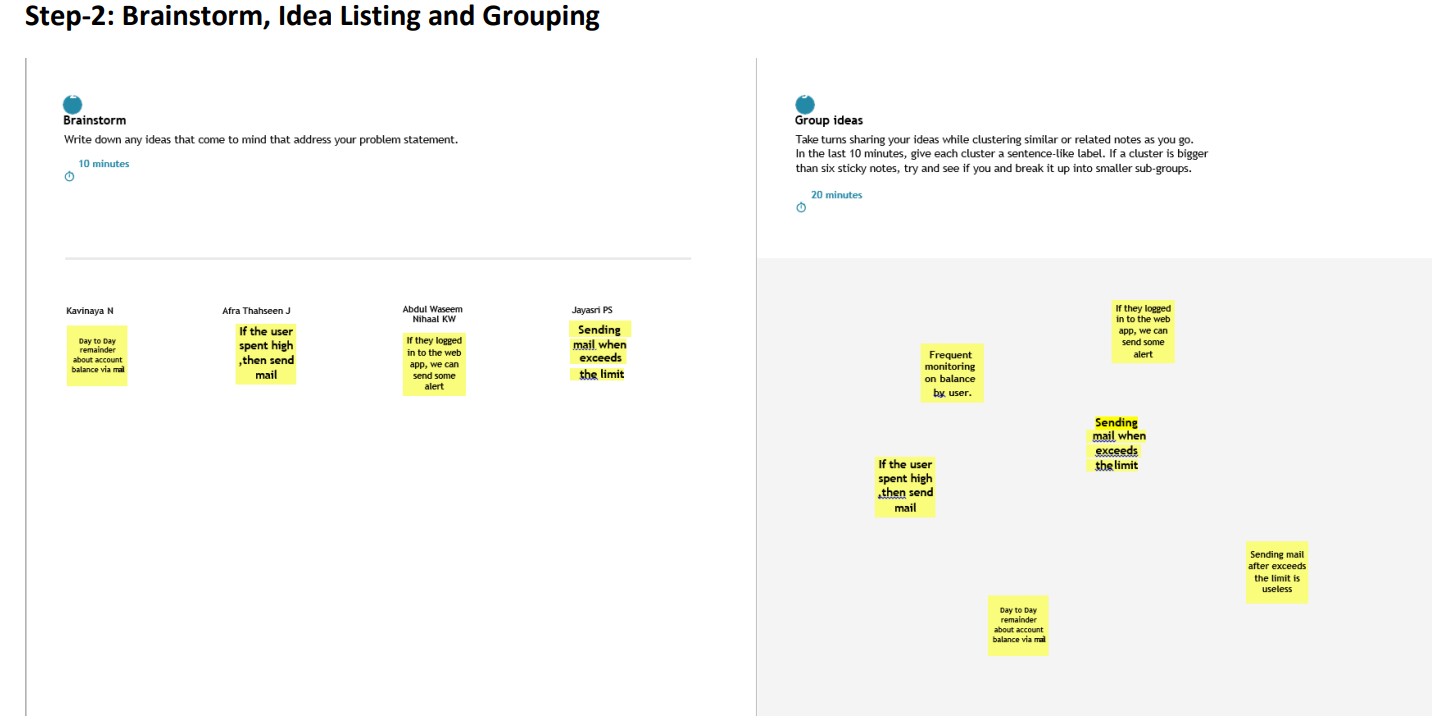


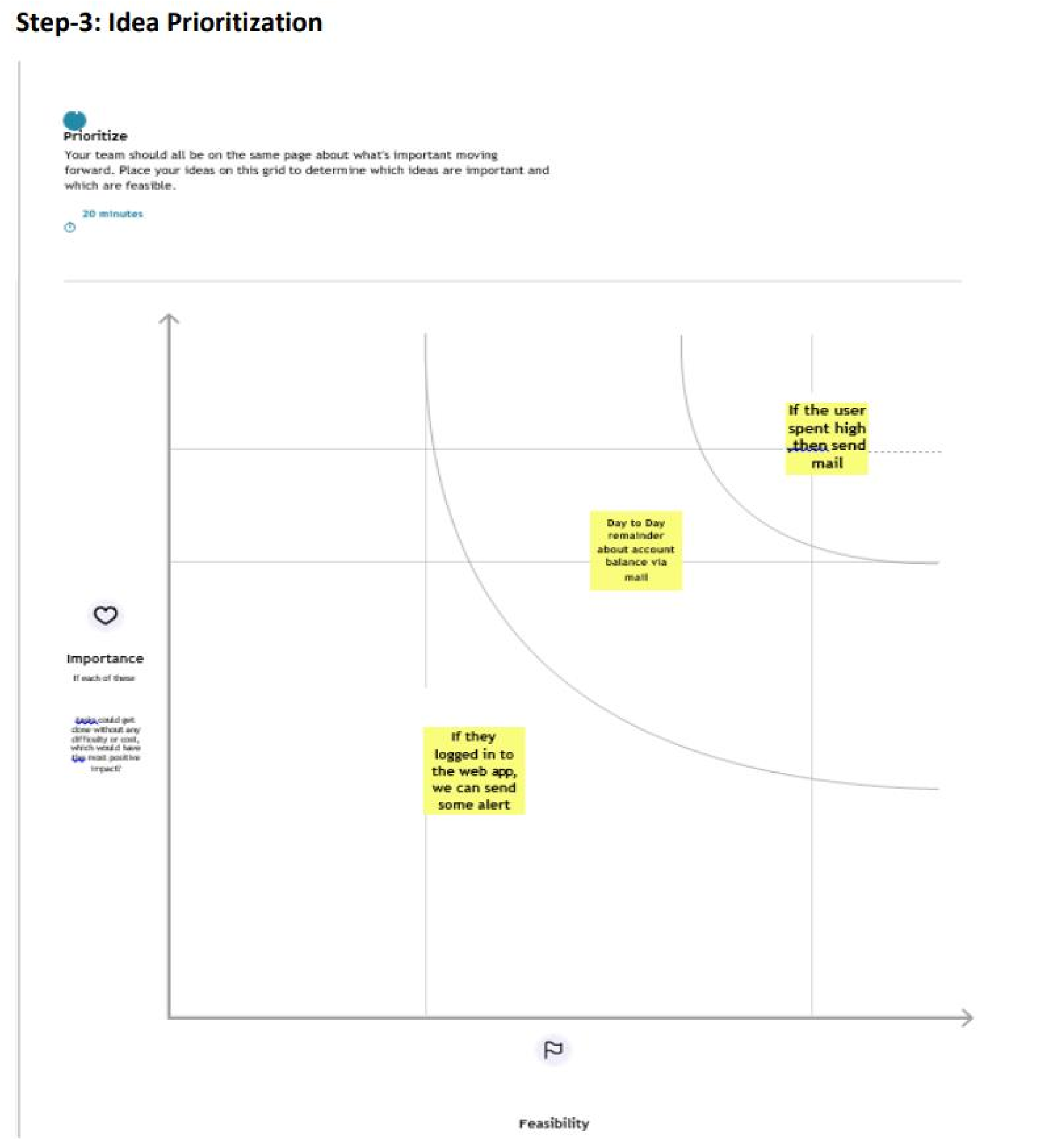
1. **IDEATION & PROPOSED SOLUTION**

a. Empathy Map Canvas



B. Ideation & Brainstorming



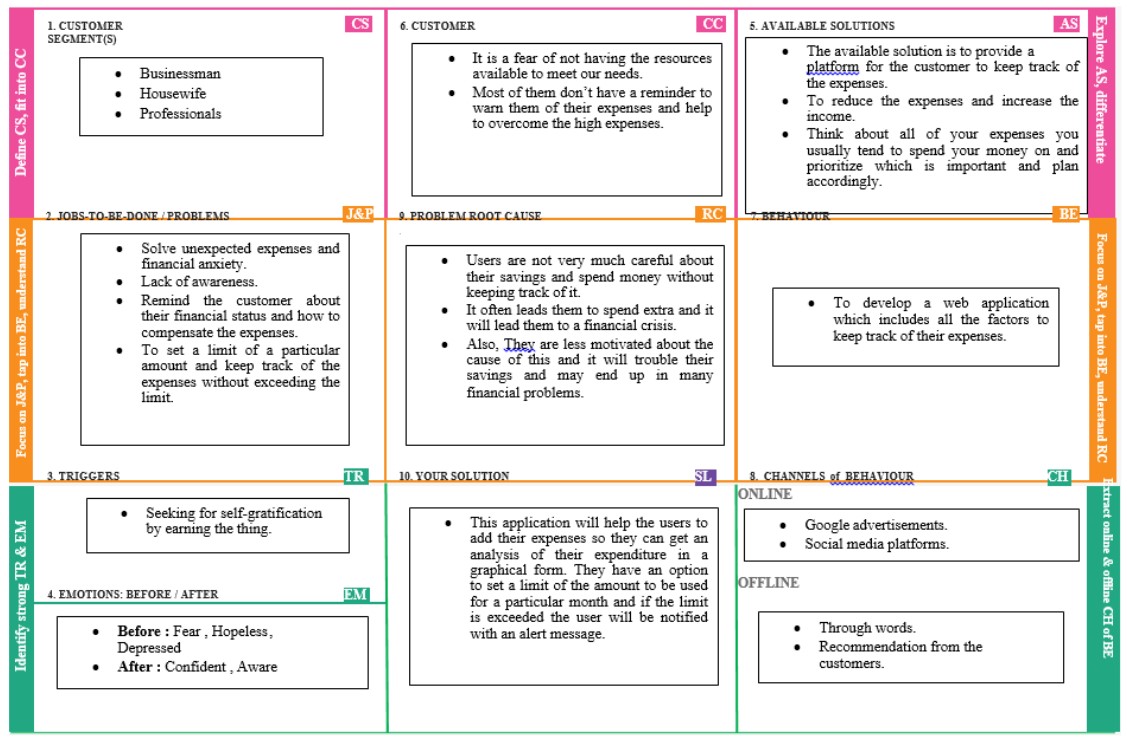


c.Proposed Solution

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Parameter** | **Description** |
| 1. | Problem Statement (Problem to be solved) | Keeping Proper track of our daily expenses is becoming challenging in today’s world. Without the proper money management knowledge people overspend on their wants instead of focusing on their needs.  Especially when using online applications for purchasing their requirements consumers tend to over spend. This problem leads to improper distribution of their daily expenses. Without proper knowledge on managing money poor  are becoming poorer and rich are becoming richer. |
| 2. | Idea / Solution description | An attempt to develop an app to manage our daily expenses and give us insights on managing our money would be a good idea. This app will be able to track expenses on various online platforms and apps. The app can help with proper budgeting and give alerts when the user over spends or crosses the limit previously set by them. This will lead to proper spending habits and make them knowledgeable about money management. IBM cloud can be used to handle the data safely. |
| 3. | Novelty / Uniqueness | The speciality for the app will be the data security with IBM cloud being used for data storage and this app genuinely helps with the money management. The proper and personalized budgeting of the user’s money leads them to trust the app and they wouldn’t have to worry about their expenditure on unnecessary things. |

|  |  |  |
| --- | --- | --- |
| 4. | Social Impact / Customer Satisfaction | People using the app will be becoming better at their spending habits and will be able to save more than their peers who are not using the app. This application aims to improve the users' savings sustainably and steadily which leads them to trust the app without worrying about their money. |
| 5. | Business Model (Revenue Model) | This application leads to a business model, the user can be suggested the right products to buy based on their budget and this can lead to targeted business approaching the right consumers. The model leads to systematic and structured expenses of the user and also leads to predictive analysis of the future expenses of the consumer. This model makes the user more careful with expenses as they are provided with the money management insights. |
| 6. | Scalability of the Solution | This application can be created as a multi user model nationwide. The model can also be modified based on the country’s law on applications and data security which leads to international implementation of this application by maintaining proper gateway rules. This app when developed for multiple nations can be modified to their requirements. The app can also be modified for a particular group of people or organization. |

d.Problem Solution fit



4. REQUIREMENT ANALYSIS

1. **Functional requirements**

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Functional Requirement** | **Description** |
| FR-1 | **Register** | Registration is the process of the user to complete the application's form. Certain details must be submitted such as e-mail address, password, and password confirmation. The user is identified using these details. |
| FR-2 | **Login** | The login screen is used to verify the identity of the user. The account can be accessed using the user's registered email address and password. |
| FR-3 | **Categories** | On the main page, we can see overall revenue and spending, as well as the balance remaining after expenditure, as well as the user's entire categories namely Entertainment, Cloth, Food and Drinks, Health and Fitness and so on. |
| FR-4 | **Update Daily Expensive** | The user can upload the daily  expensive details what they are spending on each day. The details such as cloth, entertainment, food, health etc., |

|  |  |  |
| --- | --- | --- |
| FR-5 | **View Expensive Chart** | This module used to see a pictorial depiction of all details in the form of a pie chart, where each slice of the pie chart represents that the viewer to gain an approximatenotion of which category has the highest expenses. |
| NFR-6 | **Set Alert** | When a user attempts to spend more than the pre-defined amount limit, the app will automatically send an alert if the threshold amount they selected for an alert is exceeded. |

1. **Non-Functional requirements**

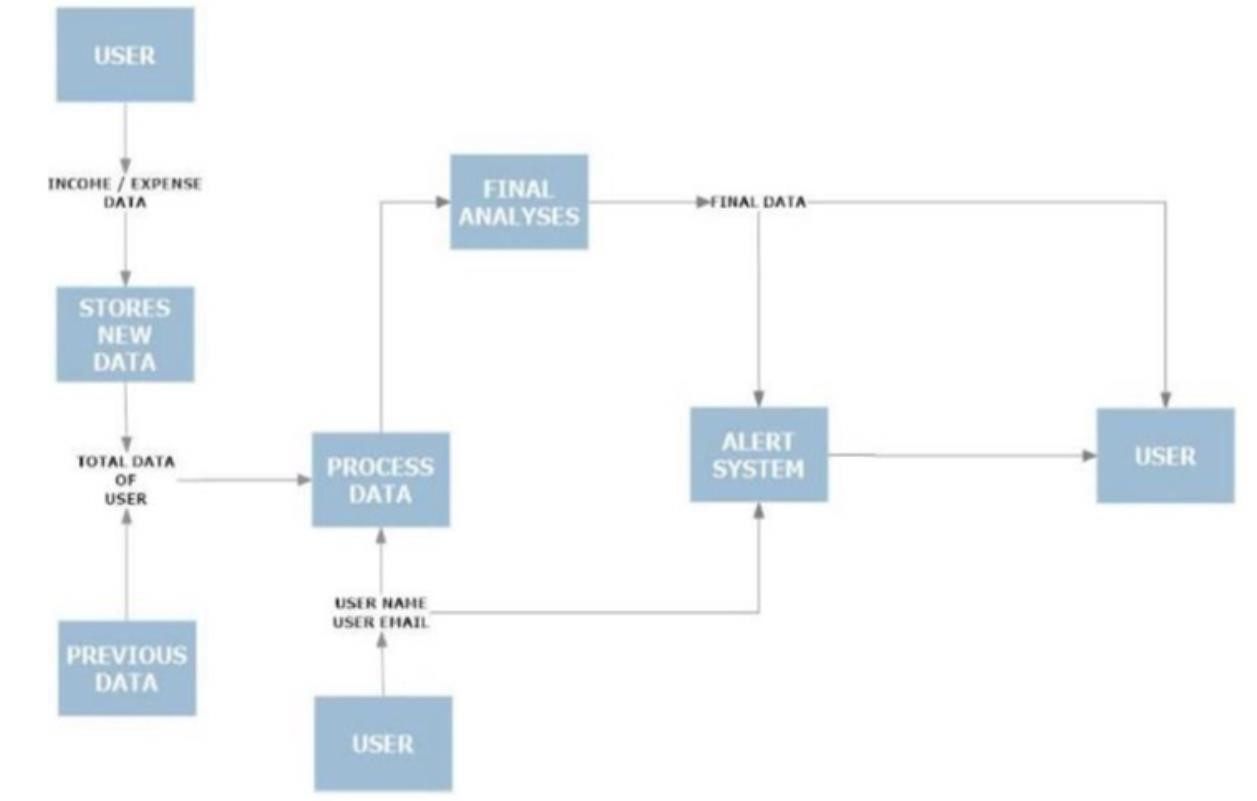
|  |  |  |
| --- | --- | --- |
| **NFR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | **Usability** | The system shall allow the users to access the system with pc using web application. The system uses a web application as an interface. The system is user friendly which makes the system easy. |
| NFR-2 | **Security** | A security requirement is a statement of needed security functionality that ensures one of many different security properties of software is being satisfied. |

|  |  |  |
| --- | --- | --- |
| NFR-3 | **Reliability** | he system has to be 100% reliable due to the importance of data and the damages that can be caused by incorrect or incomplete data.  The system will run 7 days a week.  24 hours a day. |
| NFR-4 | **Performance** | The information is refreshed depending upon whether some updates have occurred or not in the application. The system shall respond to the member in not less than two seconds from the time of the request submittal. The system shall be allowed to take more time when doing large processing jobs. Responses to view information shall take no longer than 5 seconds to appear on the screen. |
| NFR-5 | **Availability** | The system is available 100% for the user and isused 24 hrs a day and 365 days a year. The system shallbe operational 24 hours a day and 7 days a week. |
| NFR-6 | **Scalability** | Scalability is the measure of a  system's ability toincrease or decrease in performance and cost in response to changes in application and system processing demands. |

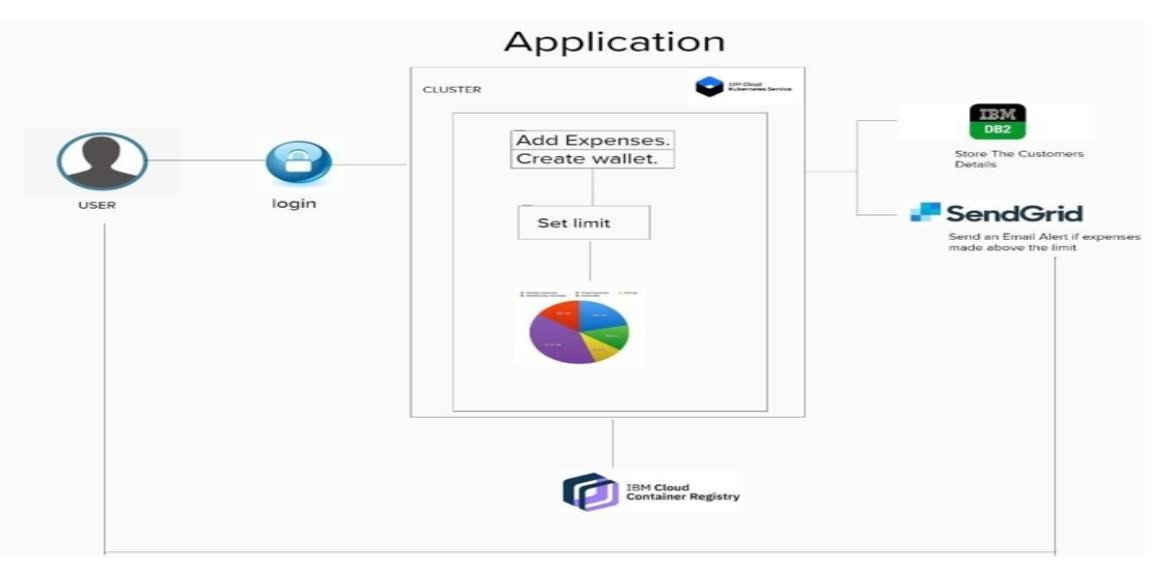
### 5. PROJECT DESIGN

a. **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.



### B.Solution & Technical Architecture



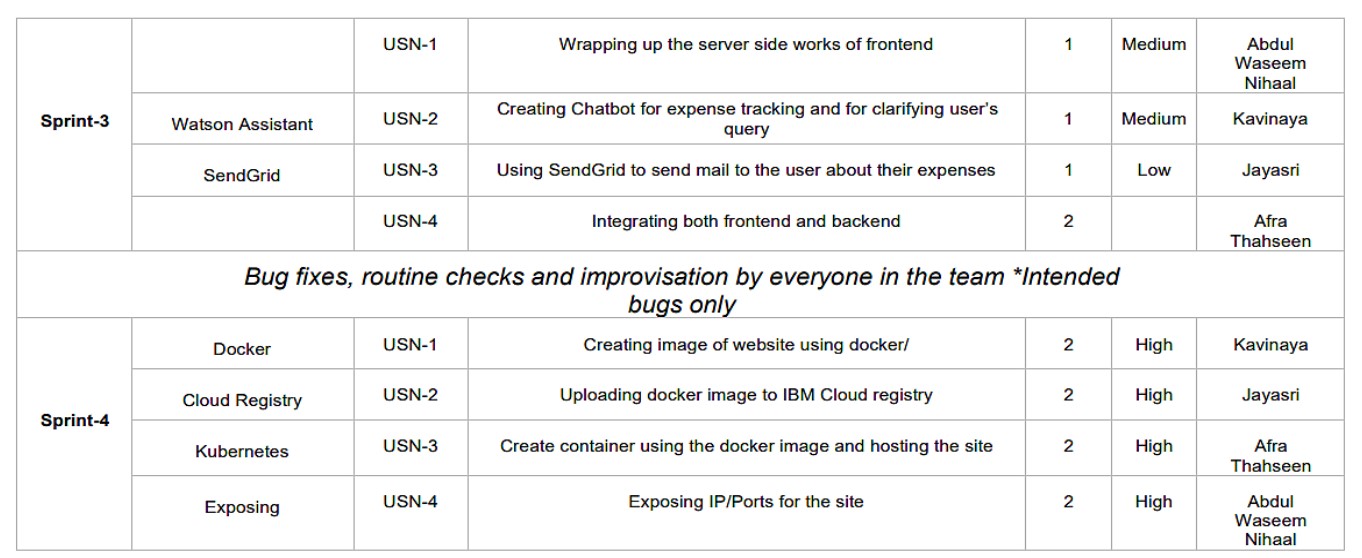
### C.User Stories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **User Type** | **Functional**  **Requireme**  **nt (Epic)** | **User**  **Story**  **Number** | **User Story /**  **Task** | **Acceptance** **criteria** | **Priority** |
|  |  |  |  |  |  |
| Customer  (web user) | Registration | USN-1 | As a user, I can register for the application by entering mail id and password | I can access my account/ dashboard | High |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  | USN-2 | As a user,I will receive a confirmation email once I have registered for the email and click application | I can receive a confirmation email | High |
|  |  | USN-3 | As a user, I can access using mail | I can register through mail | Low |
|  | Login | USN-4 | As a user, I can login application by entering application using email and password | I can access the application | High |
|  | 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 care executive, I can solve the login issue and other issues of the solution at any application | I can provide support | Medium |
| Administrator | Application | USN-7 | As an administrator,I can upgrade or update the application | I can fix the bug | Medium |

### 6.PROJECT PLANNING & SCHEDULING

1. Sprint planning and estimation





|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprin t** | **Total Story Points** | **Duratio n** | **Sprin t**  **Start Date** | **Sprint End Date**  **(Planned)** | **Story Points**  **Completed**  **(as on Planned End**  **Date)** | **Sprint Release Date (Actual)** |
| Sprint -1 | 20 | 6 Days | 24 Oct  2022 | 29 Oct  2022 | 20 | 29 Oct 2022 |
| Sprint -2 | 20 | 6 Days | 31 Oct  2022 | 05 Nov  2022 | 20 | 05 Nov 2022 |
| Sprint -3 | 20 | 6 Days | 07 Nov  2022 | 12 Nov  2022 | 20 | 12 Nov 2022 |
| Sprint -4 | 20 | 6 Days | 14 Nov  2022 | 19 Nov  2022 | 20 | 19 Nov 2022 |

1. Sprint Delivery Schedule

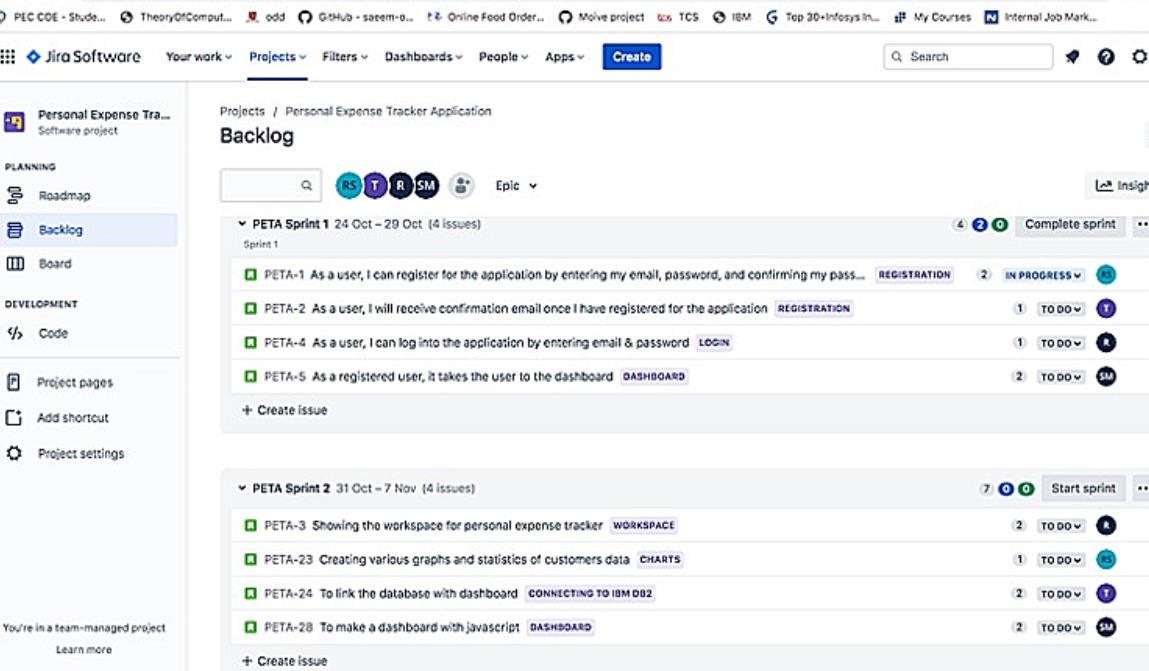
|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **MILESTONES** | **ACTIVITIES** | **DATE** |
| **1.** | **Preparation Phase** | Pre-requisites | 24 Aug 2022 |
| Prior Knowledge | 25 Aug 2022 |
| Project Structure | 23 Aug 2022 |
| Project Flow | 23 Aug 2022 |
| Project Objectives | 22 Aug 2022 |
| Registrations | 26 Aug 2022 |
| Environment Set-up | 27 Aug 2022 |
| **2.** | **Ideation Phase** | Literature Survey | 29 Aug 2022  – 03 Sept 2022 |
| Empathy Map | 5 Sept 2022 -  7 Sept 2022 |
| Problem Statement | 8 Sept 2022 -  10 Sept 2022 |
| Ideation | 12 Sept 2022  – 16 Sept 2022 |

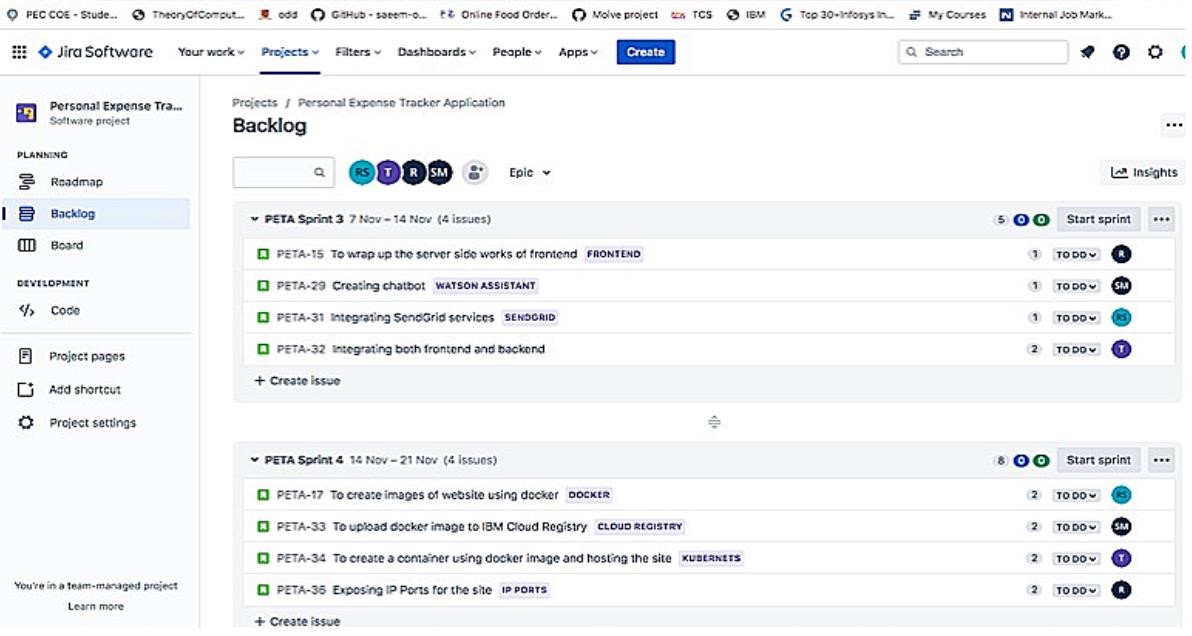
|  |  |  |  |
| --- | --- | --- | --- |
| **3.** | **Project Design Phase - 1** | Proposed Solution | 19 Sept 2022  – 23 Sept  2022 |
| Problem Solution Fit | 24 Sept 2022  – 26 Sept  2022 |
|  |  | Solution Architecture | 27 Sept 2022  – 30 Sept  2022 |

|  |  |  |  |
| --- | --- | --- | --- |
| **4.** | **Project Design Phase**  **- 2** | Customer Journey Map | 03 Oct 2022 –  08 Oct 2022 |
| Requirement Analysis | 09 Oct 2022 – 11 Oct 2022 |
| Data Flow Diagrams | 11 Oct 2022 –  14 Oct 2022 |
| Technology Architecture | 1. Oct 2022 - 2. Oct 2022 |
| **5.** | **Project Planning Phase** | Milestones & Tasks | 1. Oct 2022 – 2. Oct 2022 |
| Sprint Schedules | 19 Oct 2022 –  22 Oct 2022 |
| **6.** | **Project Development Phase** | Sprint - 1 | 24 Oct 2022 –  29 Oct 2022 |
|  |  | Sprint – 2 | 31 Oct 2022  – 05 Nov 2022 |
|  |  | Sprint – 3 | 07 Nov 2022  – 12 Nov 2022 |
|  |  | Sprint – 4 | 14 Nov 2022  – 19 Nov 2022 |

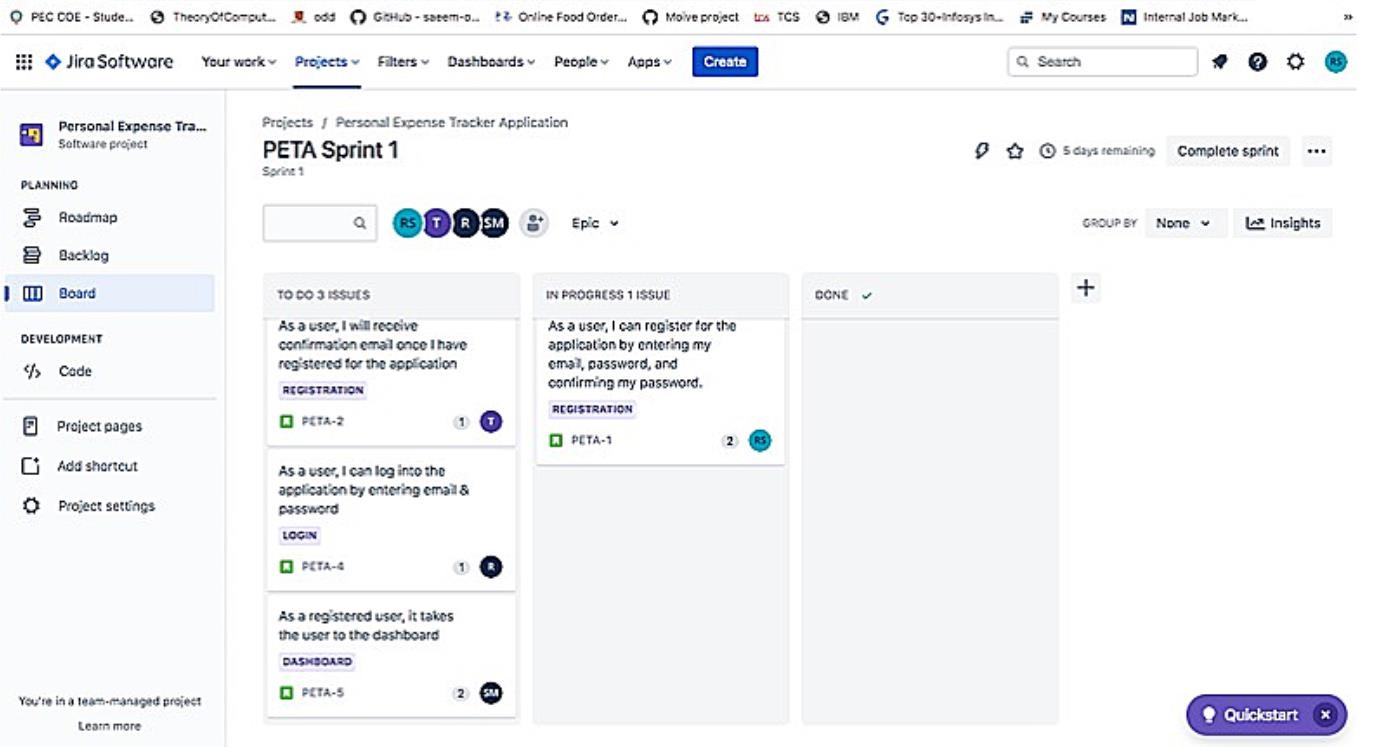
a. **Reports from JIRA**

### i. Backlog

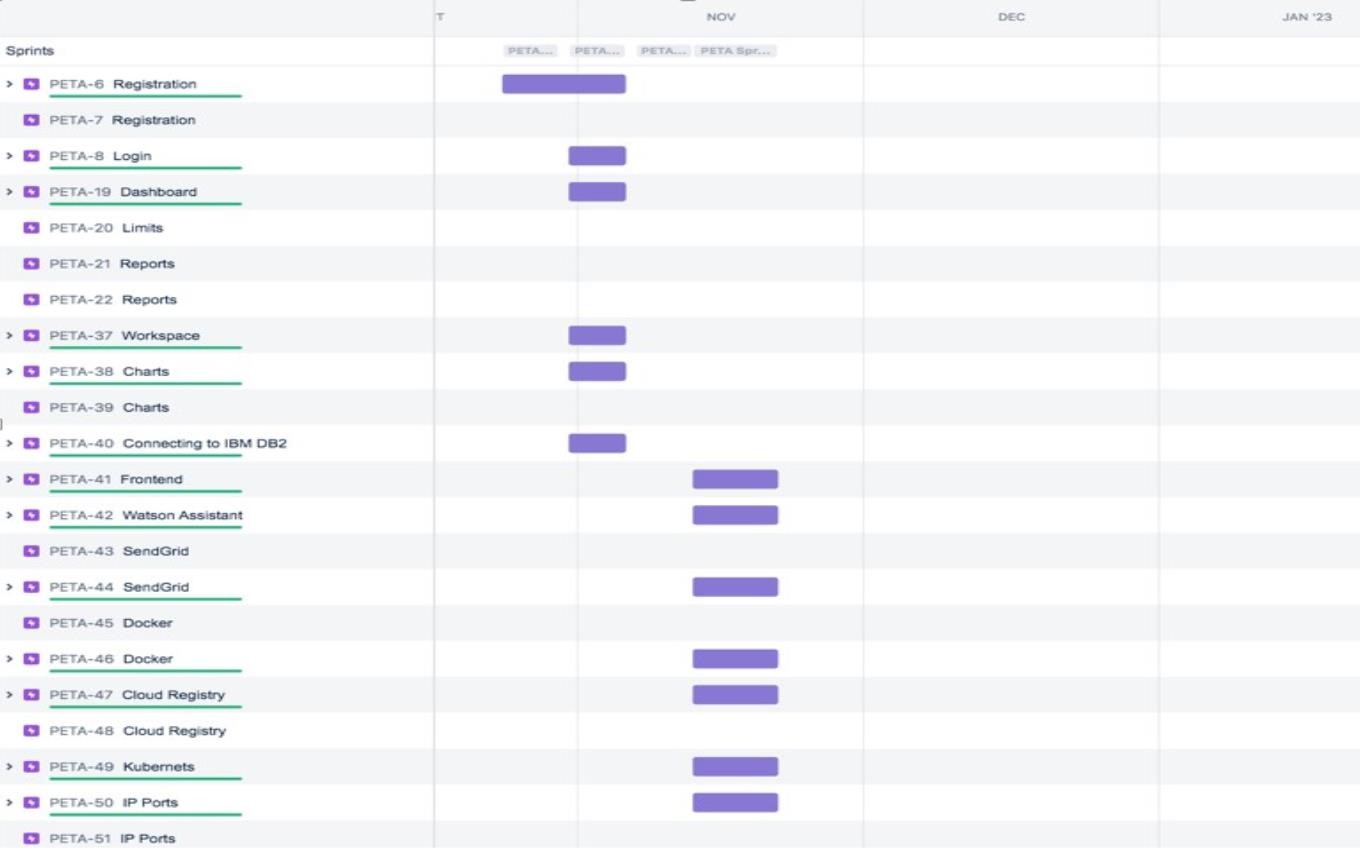




ii.Board



iii.Road Map



### 7. CODING & SOLUTIONING

**app.py:**

# -\*- coding: utf-8 -\*-

"""

Spyder Editor

This is a temporary script file.

"""

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

# from flask\_mysqldb import MySQL # import MySQLdb.cursors import re

from flask\_db2 import DB2 import ibm\_db import ibm\_db\_dbi from sendemail import sendgridmail,sendmail

# from gevent.pywsgi import WSGIServer import os

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

app.secret\_key = 'a'

# app.config['MYSQL\_HOST'] = 'remotemysql.com'

# app.config['MYSQL\_USER'] = 'D2DxDUPBii'

# app.config['MYSQL\_PASSWORD'] = 'r8XBO4GsMz'

# app.config['MYSQL\_DB'] = 'D2DxDUPBii'

"""

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-be8cfa31c41761d2.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=3883e7e4-18f5-4afe-be8c-

fa31c41761d2.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;port=31498;protocol=tcpi p;\

uid=sbb93800;pwd=wobsVLm6ccFxcNLe;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())

# app.config['']

# mysql = MySQL(app)

#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(): 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")

# cursor = mysql.connection.cursor() # with app.app\_context():

# print("Break point3")

# cursor = ibm\_db\_conn.cursor()

# print("Break point4")

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)

# dictionary = ibm\_db.fetch\_assoc(result)

# cursor.execute(stmt)

# account = cursor.fetchone()

# print(account)

# while ibm\_db.fetch\_row(result) != False:

# # account = ibm\_db.result(stmt)

# print(ibm\_db.result(result, "username"))

# print(dictionary["username"]) 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)

# cursor.execute('INSERT INTO register VALUES (NULL, % s, % s, % s)', (username, email,password))

# mysql.connection.commit() 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']

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM register WHERE username = % s AND password = % s', (username, password ),)

# account = cursor.fetchone()

# print (account)

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)

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

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)

# cursor = mysql.connection.cursor()

# cursor.execute('INSERT INTO expenses VALUES (NULL, % s, % s, % s, % s, % s, % s)', (session['id'] ,date, expensename, amount, paymode, category))

# mysql.connection.commit()

# print(date + " " + expensename + " " + amount + " " + paymode + " " + category)

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(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 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.

" + 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'])

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM expenses WHERE userid = % s AND date ORDER

BY `expenses`.`date` DESC',(str(session['id'])))

# expense = cursor.fetchall()

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):

# cursor = mysql.connection.cursor()

# cursor.execute('DELETE FROM expenses WHERE id = {0}'.format(id))

# mysql.connection.commit()

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):

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM expenses WHERE id = %s', (id,))

# row = cursor.fetchall()

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']

# cursor = mysql.connection.cursor()

# cursor.execute("UPDATE `expenses` SET `date` = % s , `expensename` = % s , `amount` = % s, `paymode` = % s, `category` = % s WHERE `expenses`.`id` = % s ",(date, expensename, amount, str(paymode), str(category),id))

# mysql.connection.commit()

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'] # cursor = mysql.connection.cursor()

# cursor.execute('INSERT INTO limits VALUES (NULL, % s, % s) ',(session['id'], number))

# mysql.connection.commit()

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():

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT limitss FROM `limits` ORDER BY `limits`.`id` DESC LIMIT 1')

# x= cursor.fetchone()

# s = x[0]

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 = " /-" while dictionary != False:

temp = [] 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():

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT TIME(date) , amount FROM expenses WHERE userid =

%s AND DATE(date) = DATE(NOW()) ',(str(session['id'])))

# texpense = cursor.fetchall()

# print(texpense)

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)

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM expenses WHERE userid = % s AND DATE(date) =

DATE(NOW()) AND date ORDER BY `expenses`.`date` DESC',(str(session['id']))) # expense = cursor.fetchall()

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():

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT DATE(date), SUM(amount) FROM expenses WHERE

userid= %s AND MONTH(DATE(date))= MONTH(now()) GROUP BY DATE(date) ORDER

BY DATE(date) ',(str(session['id'])))

# texpense = cursor.fetchall()

# print(texpense)

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 = []

temp.append(dictionary1["DT"]) temp.append(dictionary1["TOT"]) texpense.append(temp) print(temp) dictionary1 = ibm\_db.fetch\_assoc(res1) # cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM expenses WHERE userid = % s AND

MONTH(DATE(date))= MONTH(now()) AND date ORDER BY `expenses`.`date`

DESC',(str(session['id'])))

# expense = cursor.fetchall()

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():

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT MONTH(date), SUM(amount) FROM expenses WHERE

userid= %s AND YEAR(DATE(date))= YEAR(now()) GROUP BY MONTH(date) ORDER BY

MONTH(date) ',(str(session['id'])))

# texpense = cursor.fetchall()

# print(texpense)

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)

# cursor = mysql.connection.cursor()

# cursor.execute('SELECT \* FROM expenses WHERE userid = % s AND

YEAR(DATE(date))= YEAR(now()) AND date ORDER BY `expenses`.`date`

DESC',(str(session['id'])))

# expense = cursor.fetchall()

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)

**deployment.yaml:** apiVersion: apps/v1 kind: Deployment metadata:

name: sakthi-flask-node-deployment spec: replicas: 1 selector:

matchLabels: app: flasknode template: metadata: labels:

app: flasknode spec:

containers: - name: flasknode

image: icr.io/sakthi\_expense\_tracker2/flask-template2 imagePullPolicy: Always

ports:

* containerPort: 5000

**flask-service.yaml:** apiVersion: v1 kind: Service metadata:

name: flask-app-service spec: selector:

app: flask-app ports: - name: http protocol: TCP port: 80 targetPort: 5000 type: LoadBalancer **manifest.yml:** applications:

* name: Python Flask App IBCMR 2022-10-19 random-route: true memory: 512M disk\_quota: 1.5G

**sendemail.py:** import smtplib import sendgrid as sg import os

from sendgrid.helpers.mail import Mail, Email, To, Content SUBJECT = "expense tracker" s = smtplib.SMTP('smtp.gmail.com', 587)

def sendmail(TEXT,email):

print("sorry we cant process your candidature")

s = smtplib.SMTP('smtp.gmail.com', 587)

s.starttls()

# s.login("il.tproduct8080@gmail.com", "oms@1Ram")

s.login("tproduct8080@gmail.com", "lxixbmpnexbkiemh") message = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT) # s.sendmail("il.tproduct8080@gmail.com", email, message) s.sendmail("il.tproduct8080@gmail.com", email, message)

s.quit() def sendgridmail(user,TEXT):

# from\_email = Email("shridhartp24@gmail.com") from\_email = Email("tproduct8080@gmail.com") to\_email = To(user) subject = "Sending with SendGrid is Fun" content = Content("text/plain",TEXT) mail = Mail(from\_email, to\_email, subject, content)

# Get a JSON-ready representation of the Mail object mail\_json = mail.get()

# Send an HTTP POST request to /mail/send response = sg.client.mail.send.post(request\_body=mail\_json) print(response.status\_code) print(response.headers)

**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

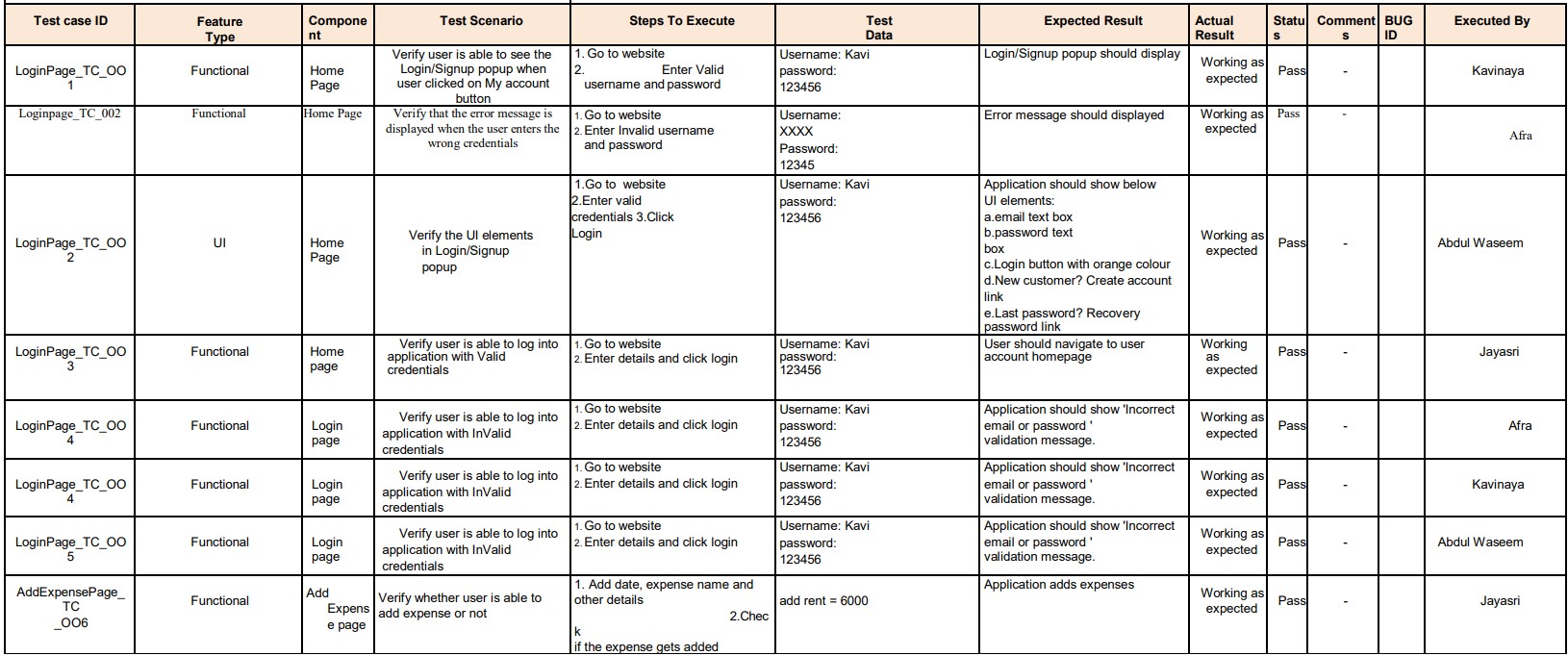
3.LIMIT

id INT NOT NULL GENERATED ALWAYS AS IDENTITY,userid VARCHAR(32) NOT NULL, limit

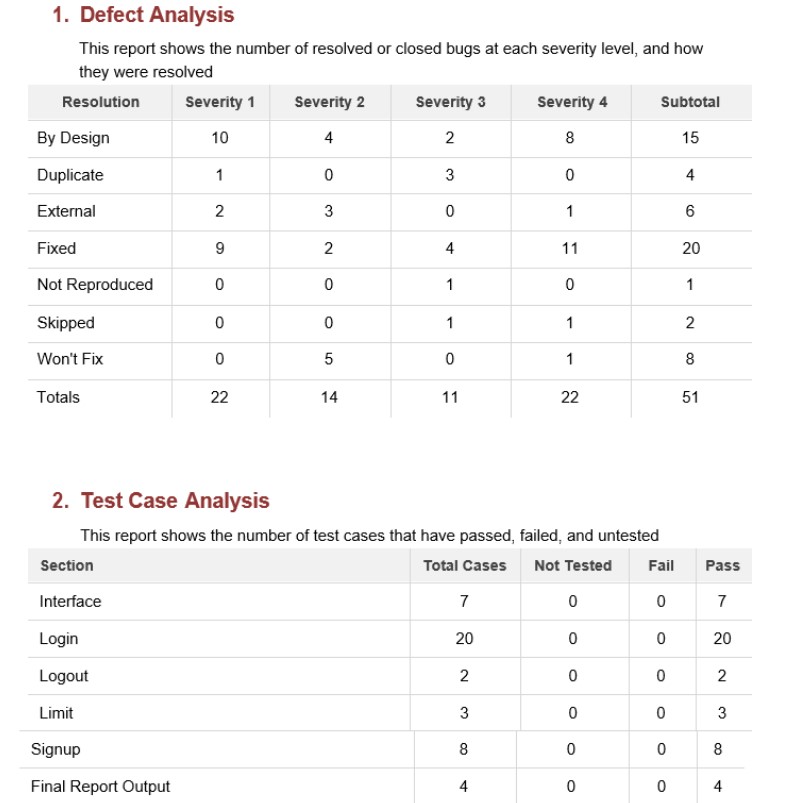
VARCHAR(32) NOT NULL

**8.TESTING:**

**a.TestCases:**



b.User Acceptance Testing



**9.RESULTS**

a. Performance Metrics

1. Tracking income and expenses: Monitoring the income and tracking all expenditures (through bank accounts, mobile wallets, and credit & debit cards).
2. Transaction Receipts: Capture and organize your payment receipts to keep track of your expenditure.
3. Organizing Taxes: Import your documents to the expense tracking app, and it will streamline your income and expenses under the appropriate tax categories.
4. 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 trackingapp sends remindersfor payments an d automatically matches the payments with invoices.
5. Reports: The expense tracking app generates and sends reports to give a detailed insight about profits, losses, budgets, income, balance sheets, etc.,
6. Ecommerce integration: Integrateyour expense trackingapp wit h your eCommerce store and track your sales through payments received via multiple payment methods.
7. Vendors and Contractors: Manage and track all the payments to the vendors and contractors added to the mobile app.
8. Access control: Increase your team productivity by providing access control to particular users through custom permissions.
9. Track Projects: Determine project profitability by tracking labor costs, payroll, expenses, etc., of your ongoing project.

1. 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.
2. In-depth insights and analytics: Provides in-built tools to generate reports with easy-to- understand visuals and graphics to gain insights about the performance of your business.
3. Recurrent Expenses: Rely on your budgeting app to track, streamline, and automate all the recurrent expenses and remind you on a timely basis.

10. ADVANTAGES & DISADVANTAGES

1. **Achieve your business goals** with a tailored mobile app that perfectly fits your business.

1. **Scale-up** at the pace your business is growing.

1. Deliver an **outstanding** customer experience through additional control over the app.

1. Control the **security** of your business and customer data

1. Open **direct marketing channels** with no extra costs with methods such as push notifications.
2. **Boost the productivity** of all the processes within theorganization.

1. Increase **efficiency** and **customer satisfaction** with an app aligned to their needs.

1. **Seamlessly integrate** with existing infrastructure.

1. Ability to provide **valuable insights**.

1. Optimize sales processes to generate **more revenue** through enhanced data collection.

1. **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.

1. **FUTURE**

The project assists well to record the income and expenses in general. However, this project has some limitations:

1. The application is unable to maintain the backup of data once it is uninstalled.

1. This application does not provide higher decision capability.

To further enhance the capability of this application, we recommend the following features to be incorporated into the system:

1. Multiple language interface.

1. Provide backup and recovery of data.

1. Provide better user interface for user.

1. Mobile apps advantage.

### 13. APPENDIX

**Source Code Github Link :** https://github.com/IBM-EPBL/IBM-Project-25691-1659970868 **Project Demo Link:**

https://drive.google.com/file/d/1-UsFhM7BT6615PmwzjColfMoyyMBfd3T/view?usp=sharing