PERSONAL EXPENSE TRACKER APPLICATION

A PROJECT REPORT

Submitted

by

LAVANYA BH

LAKSHMI PRIYA G

RAGASRUTHI J

NITHISHA V

TEAM ID	**:**	PNT2022TMID23145
INDUSTRY MENTOR	**:**	Khusboo
FACULTY MENTOR	**:**	M.Sakthivel

TABLE OF CONTENT

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/

[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/

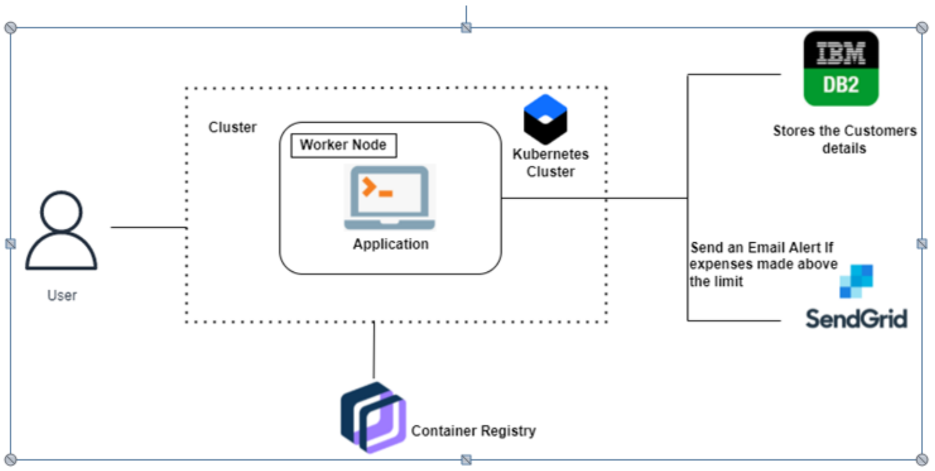
[9] Donn Felker, “Android Application Development for Dummies”, published by For Dummies, 2010.

[10] https://www.irjet.net/archives/V6/i3/IRJET-V6I31110.pdf.

[11]https://www.proquest.com/openview/a372033c3cafa03eaa5e18e68e99c86b/1.pdf?pq-origsite=gscholar&cbl=2045096

2.3 Problem Statement Definition

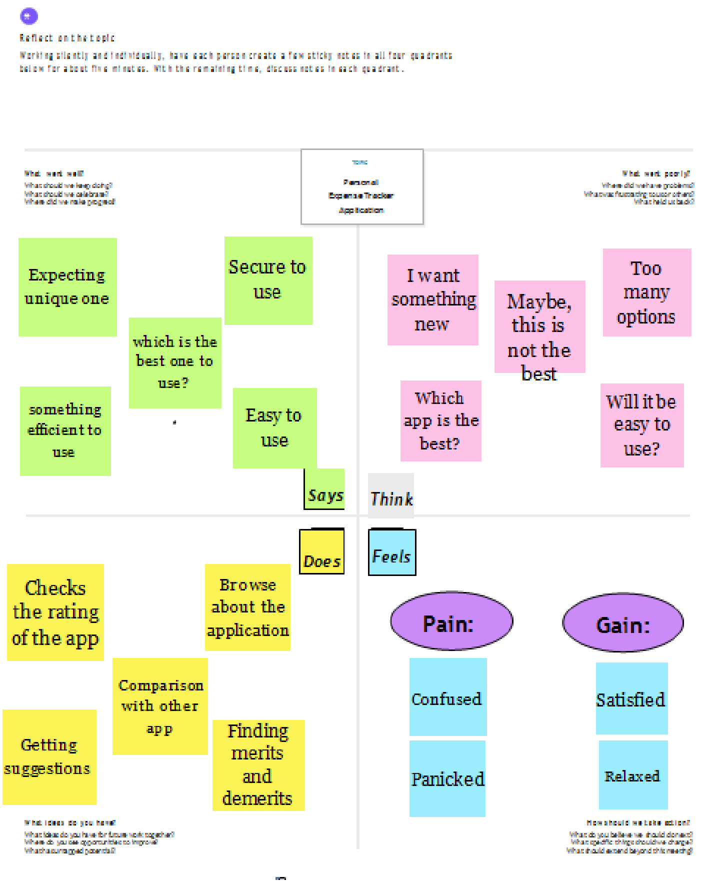
Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that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



3. IDEATION & PROPOSED SOLUTION

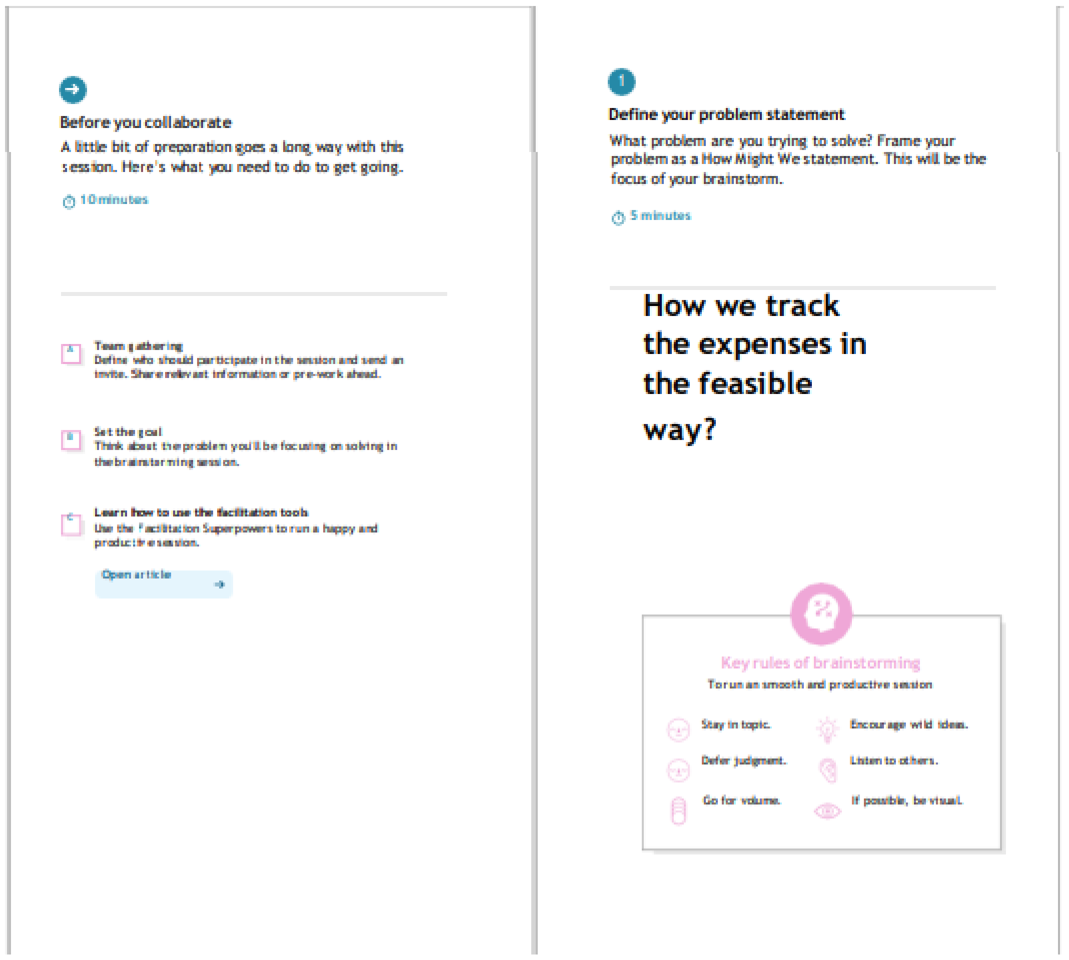
3.1 Empathy Map Canvas

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.


Reflect on the topic
Working silently and individually, have each person create a few sticky notes in all four quadrants below for about five minutes. With the remaining time, discuss notes in each quadrant.
What went well?
What should we keep doing?
What should we celebrate?
Where did we make progress?
TOPIC
Personal Expense Tracker Application
What went poorly?
Where did we have problems?
What was frustrating to us or others?
What held us back?
Expecting unique one
Secure to use
which is the best one to use?
something efficient to use
Easy to use
Says
I want something new
Maybe, this is not the best
Too many options
Which app is the best?
Will it be easy to use?
Think
Does
Feels
Checks the rating of the app
Browse about the application
Comparison with other app
Getting suggestions
Finding merits and demerits
Pain:
Gain:
Confused
Panicked
Satisfied
Relaxed
What ideas do you have?
What ideas do you have for future work together?
Where do you see opportunities to improve?
What's our untapped potential?
How should we take action?
What do you believe we should do next?
What specific things should we change?
What should extend beyond this meeting?

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.



Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

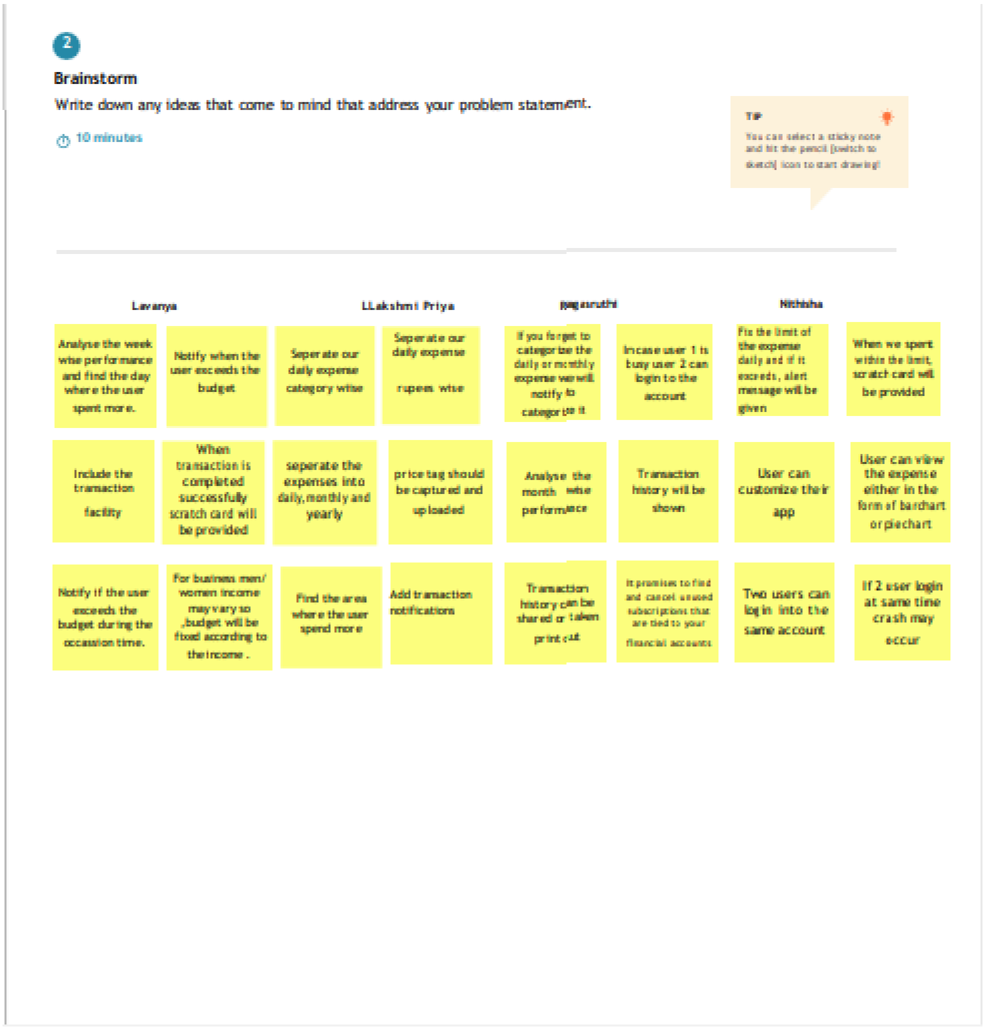
5 minutes

How we track the expenses in the feasible way?

Key rules of brainstorming

To run an smooth and productive session

Stay in topic.	Encourage wild ideas.
Defer judgment.	Listen to others.
Go for volume.	If possible, be visual.



2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Lavanya

Analyse the week wise performance and find the day where the user spent more.

Notify when the user exceeds the budget

Include the transaction facility

When transaction is completed successfully scratch card will be provided

Notify if the user exceeds the budget during the occassion time.

For business men/ women income may vary so ,budget will be fixed according to the income .

LLakshmi Priya

Seperate our daily expense category wise

Seperate our daily expense rupees wise

seperate the expenses into daily,monthly and yearly

price tag should be captured and uploaded

Find the area where the user spend more

Add transaction notifications

Jagasruthi

If you forget to categorize the daily or monthly expense we will notify to categorize it

Incase user 1 is busy user 2 can login to the account

Analyse the month wise performance

Transaction history will be shown

Transaction history can be shared or taken print out

It promises to find and cancel unused subscriptions that are tied to your financial accounts

Nithisha

Fix the limit of the expense daily and if it exceeds, alert message will be given

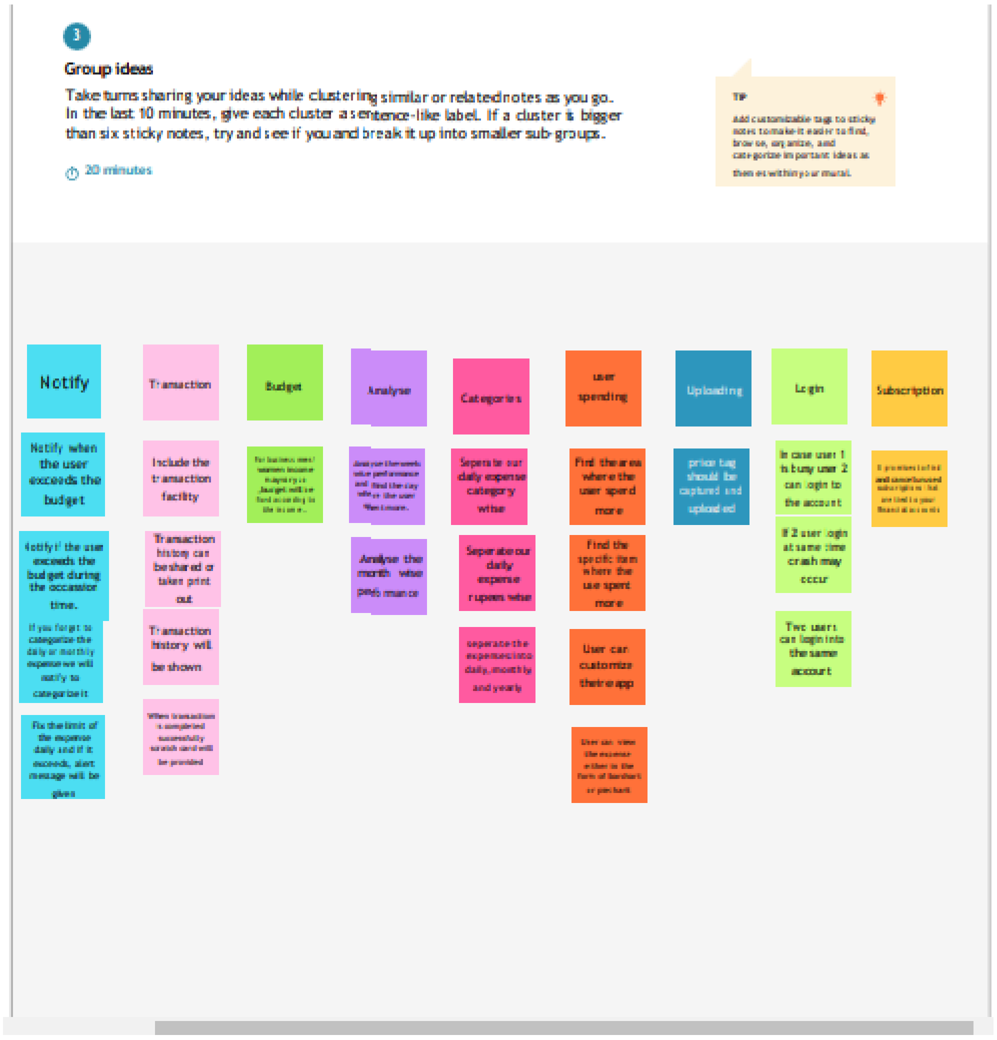
When we spent within the limit, scratch card will be provided

User can customize their app

User can view the expense either in the form of barchart or piechart

Two users can log in into the same account

If 2 user login at same time crash may occur



3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Notify

- Notify when the user exceeds the budget
- Notify if the user exceeds the budget during the occassior time.
- If you forget to categorize the daily or monthly expense we will notify to categorize it
- Fix the limit of the expense daily and if it exceeds, alert message will be given

Transaction

- Include the transaction facility
- Transaction history can be shared or taken print out
- Transaction history will be shown
- When transaction is completed successfully scratch card will be provided

Budget

- [illegible]

Analyse

- Analyse the week wise performance and find the day where the user [illegible]
- Analyse the month wise performance

Categories

- Separate our daily expense category wise
- Separate our daily expense rupees wise
- seperate the expenses into daily, monthly and yearly

user spending

- Find the area where the user spend more
- Find the specific item where the user spent more
- User can customize their app
- User can view the expense either in the form of barchart or piechart

Uploading

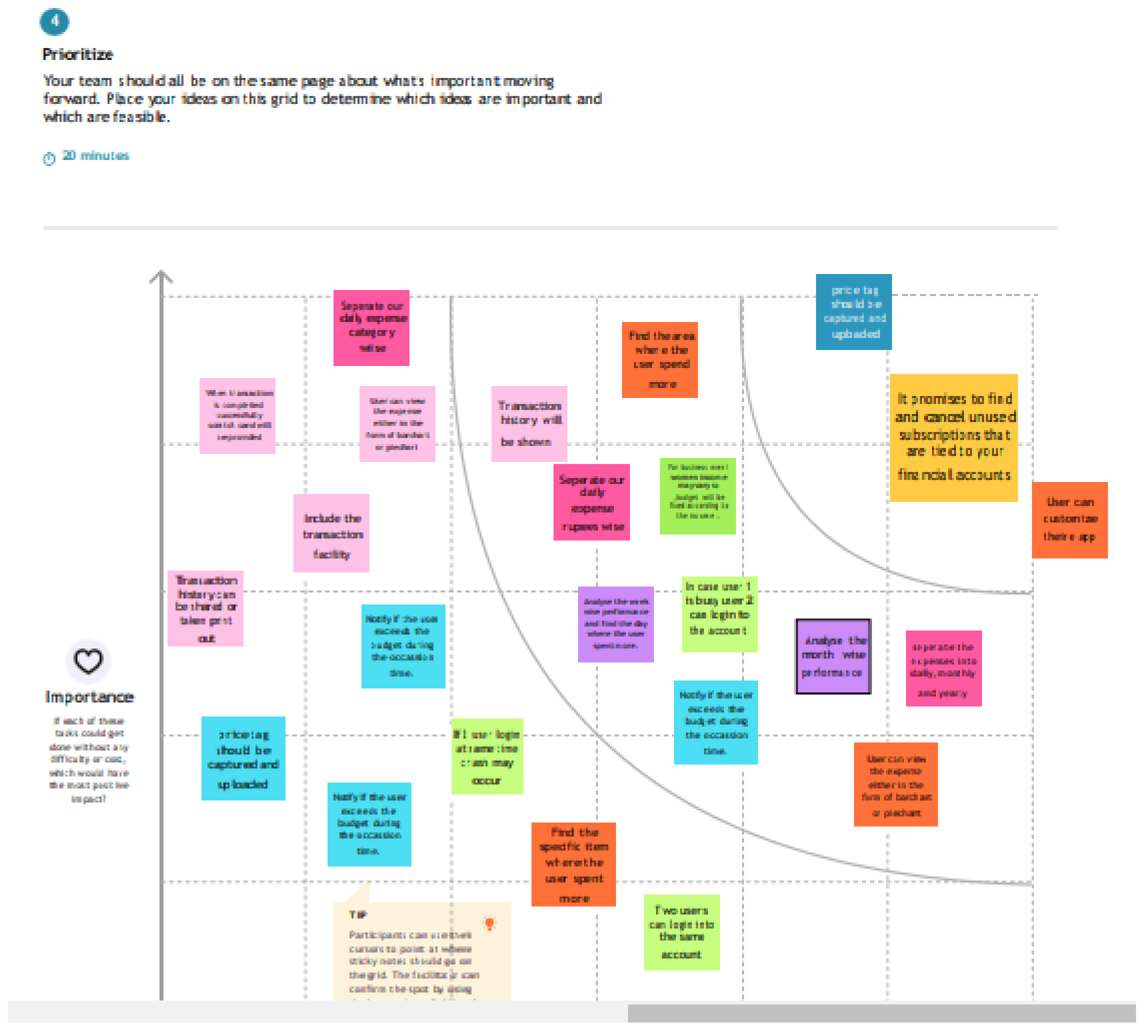
- price tag should be captured and uploaded

Login

- In case user 1 is busy user 2 can login to the account
- If 2 user login at same time crash may occur
- Two users can login into the same account

Subscription

- [illegible]



4

Prioritize

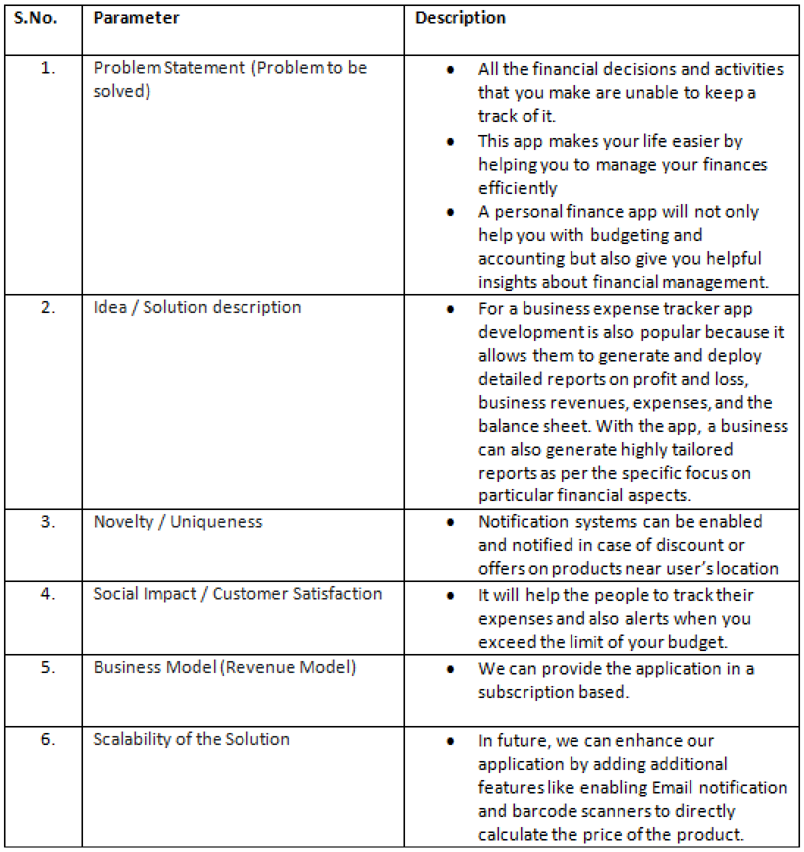
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
Separate our daily expense category wise
price tag should be captured and uploaded
Find the area where the user spend more
When transaction is completed successfully
User can view the expense either in the form of bar chart or piechart
Transaction history will be shown
It promises to find and cancel unused subscriptions that are tied to your financial accounts
User can customize their app
Separate our daily expense rupees wise
Include the transaction facility
Transaction history can be shared or taken print out
Notify if the user exceeds the budget during the occasion time.
Analyse the week wise performance and find the day where the user spent more.
In case user 1 is busy user 2 can login to the account
Analyse the month wise performance
Separate the expenses into daily, monthly and yearly
Notify if the user exceeds the budget during the occasion time.
price tag should be captured and uploaded
If 2 user login at same time crash may occur
Notify if the user exceeds the budget during the occasion time.
User can view the expense either in the form of bar chart or piechart
Find the specific item where the user spent more
Two users can login into the same account
TIP
Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using

3.3 Proposed Solution

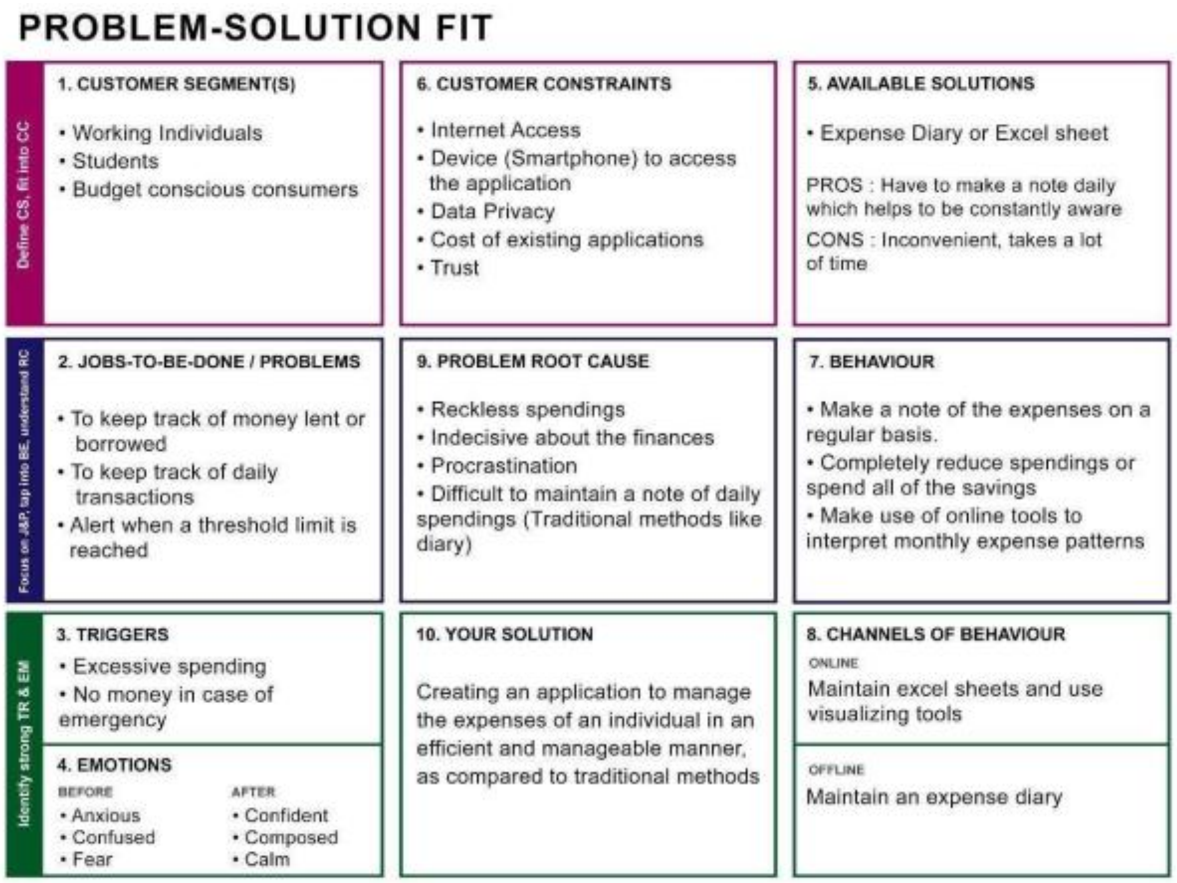
In this personal expense tracker project IBM DB2 cloud is used to store the data instead of storing in local storage. Here containerization is a concept that took over virtualization, which allows the user to run the application uniformly, and consistently on any infrastructure using the Docker application. 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.



S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• All the financial decisions and activities that you make are unable to keep a track of it. • This 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 financial management.
2.	Idea / Solution description	• For a business expense tracker app development is also popular because it allows them to generate and deploy detailed reports on profit and loss, business revenues, expenses, and the balance sheet. With the app, a business can also generate highly tailored reports as per the specific focus on particular financial aspects.
3.	Novelty / Uniqueness	• Notification systems can be enabled and notified in case of discount or offers on products near user's location
4.	Social Impact / Customer Satisfaction	• It will help the people to track their expenses and also alerts when you exceed the limit of your budget.
5.	Business Model (Revenue Model)	• We can provide the application in a subscription based.
6.	Scalability of the Solution	• In future, we can enhance our application by adding additional features like enabling Email notification and barcode scanners to directly calculate the price of the product.

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.



PROBLEM-SOLUTION FIT

Define CS, fit into CC

1. CUSTOMER SEGMENT(S)

- Working Individuals
- Students
- Budget conscious consumers

6. CUSTOMER CONSTRAINTS

- Internet Access
- Device (Smartphone) to access the application
- Data Privacy
- Cost of existing applications
- Trust

5. AVAILABLE SOLUTIONS

- Expense Diary or Excel sheet

PROS : Have to make a note daily which helps to be constantly aware

CONS : Inconvenient, takes a lot of time

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

2. JOBS-TO-BE-DONE / PROBLEMS

- To keep track of money lent or borrowed
- To keep track of daily transactions
- Alert when a threshold limit is reached

9. PROBLEM ROOT CAUSE

- Reckless spendings
- Indecisive about the finances
- Procrastination
- Difficult to maintain a note of daily spendings (Traditional methods like diary)

7. BEHAVIOUR

- Make a note of the expenses on a regular basis.
- Completely reduce spendings or spend all of the savings
- Make use of online tools to interpret monthly expense patterns

Identify strong TR & EM

3. TRIGGERS

- Excessive spending
- No money in case of emergency

4. EMOTIONS

BEFORE	AFTER
• Anxious	• Confident
• Confused	• Composed
• Fear	• Calm

10. YOUR SOLUTION

Creating an application to manage the expenses of an individual in an efficient and manageable manner, as compared to traditional methods

8. CHANNELS OF BEHAVIOUR

ONLINE

Maintain excel sheets and use visualizing tools

OFFLINE

Maintain an expense diary

4. REQUIREMENT ANALYSIS

4.1 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.

4.2 Non-Functional requirements

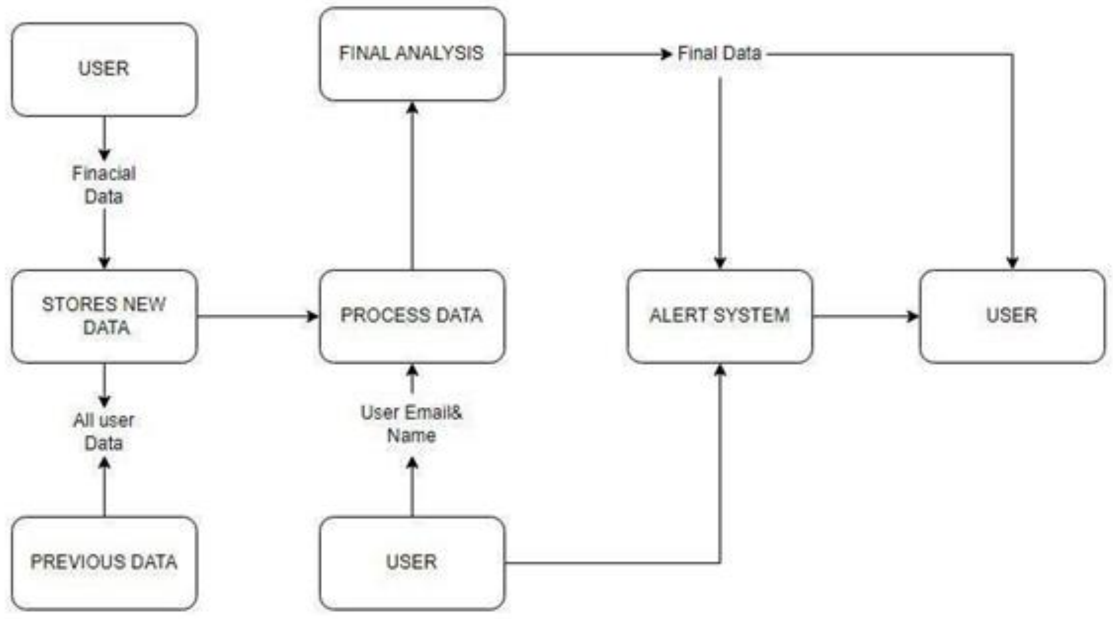
NFR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	The system shall allow the users to access the system with pc using web application. The system uses a web application as an interface. The system is user friendly which makes the system easy.
NFR-2	**Security**	A security requirement 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.

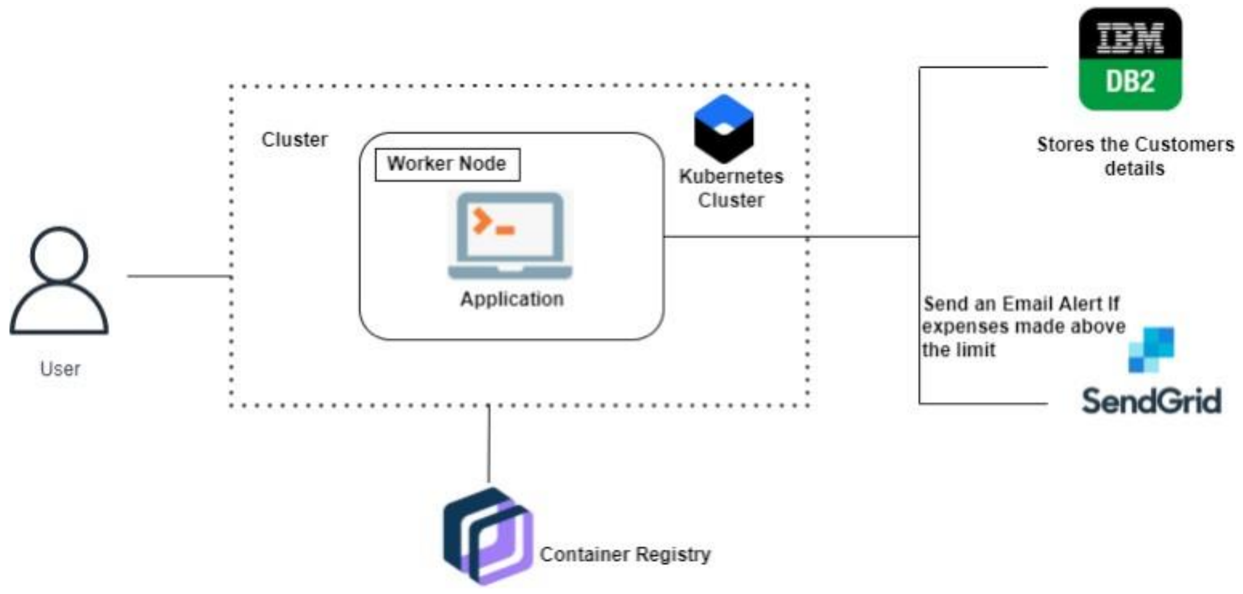
5. PROJECT DESIGN

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

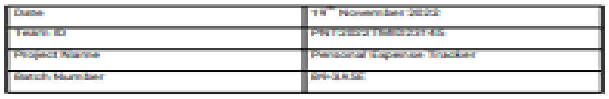


5.3 User Stories

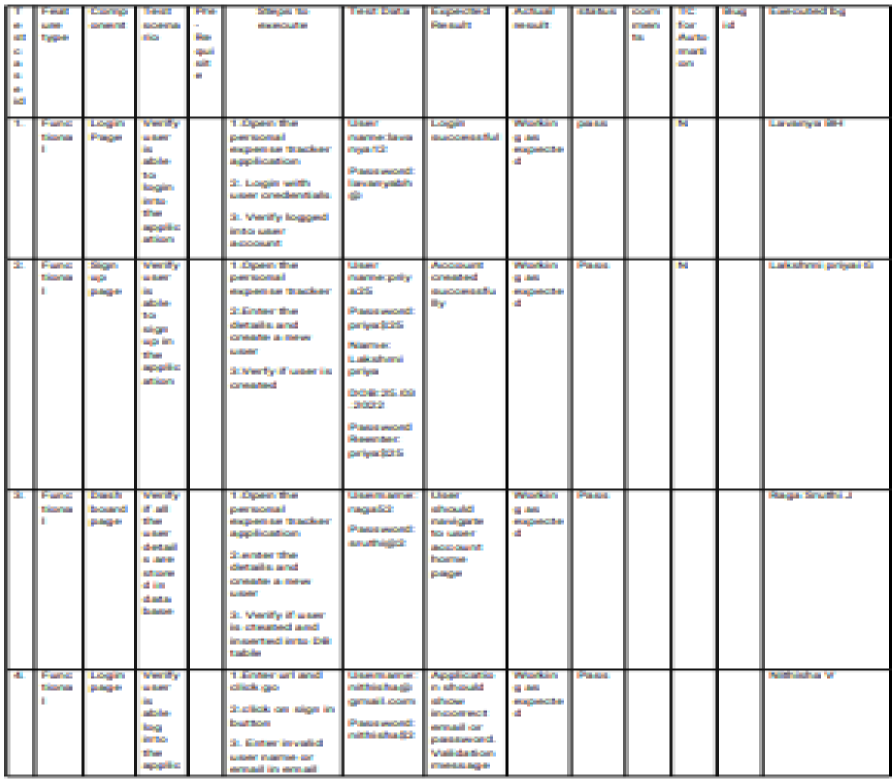
User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user & web user)	Registration	USN-1	As a user, I can register for the application by entering my email, and password, and confirming my password.	I can access my account/dashboard	High	Sprint-1
		USN-2	As a user, I will receive a confirmation email once I have registered for the application	I can receive a confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook3	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through a Google account.	I can register & access the dashboard with a Google Account login.	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering my email & password	I can access the application.	High	Sprint-1
	Dashboard	USN-6	As a user, I can see the daily expenses and expenditure details.	I can view the daily expenses and add the expense details.	High	Sprint-1
Customer Care Executive		USN-7	As a customer care executive ,it is easy to solve the problem that faced by the customers.	I can provide support to customers at any time 24*7.	Medium	Sprint-1
Administrator	Application	USN-8	As an administrator, I can update the application and provide necessary upgrades.	I can fix any bugs raised by customers and upgrade the application.	Medium	Sprint-1

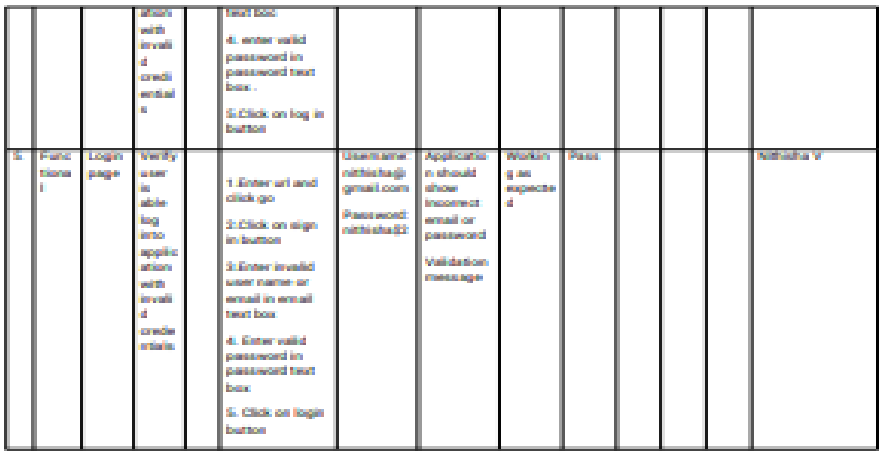
6. PROJECT PLANNING AND SCHEDULING

6.1 Sprit Planning and Estimation



Date	19th November 2022
Team ID	PNT2022TMID23145
Project Name	Personal Expense Tracker
Batch Number	B9-3A5E





Test case id	Feature type	Component	Test scenario	Pre-Requisite	Steps to execute	Test Data	Expected Result	Actual result	status	comments	TC for Automation	Bug id	Executed by
1.	Functional	Login Page	Verify user is able to login into the application		1.Open the personal expense tracker application 2. Login with user credentials 3. Verify logged into user account	User name:lavanya12 Password: lavanyabh@	Login successful	Working as expected	pass		N		Lavanya BH
2.	Functional	Sign up page	Verify user is able to sign up in the application		1.Open the personal expense tracker 2.Enter the details and create a new user 3.Verify if user is created	User name:priya35 Password: priya@35 Name: Lakshmi priya DOB:25.03.2022 Password Reenter: priya@35	Account created successfully	Working as expected	Pass		N		Lakshmi priya G
3.	Functional	Dash board page	Verify if all the user details are stored in data base		1.Open the personal expense tracker application 2.enter the details and create a new user 3. Verify if user is created and inserted into DB table	Username: raga02 Password: sruthi@2	User should navigate to user account home page	Working as expected	Pass				Raga Sruthi J
4.	Functional	Login page	Verify user is able log into the application with invalid credentials		1.Enter url and click go 2.click on sign in button 3. Enter invalid user name or email in email text box 4. enter valid password in password text box . 5.Click on log in button	Username: nithisha@gmail.com Password: nithisha02	Application should show incorrect email or password. Validation message	Working as expected	Pass				Nithisha V
5.	Functional	Login page	Verify user is able log into application with invalid credentials		1.Enter url and click go 2.Click on sign in button 3.Enter invalid user name or email in email text box 4. Enter valid password in password text box 5. Click on login button	Username: nithisha@gmail.com Password: nithisha02	Application should show incorrect email or password Validation message	Working as expected	Pass				Nithisha V

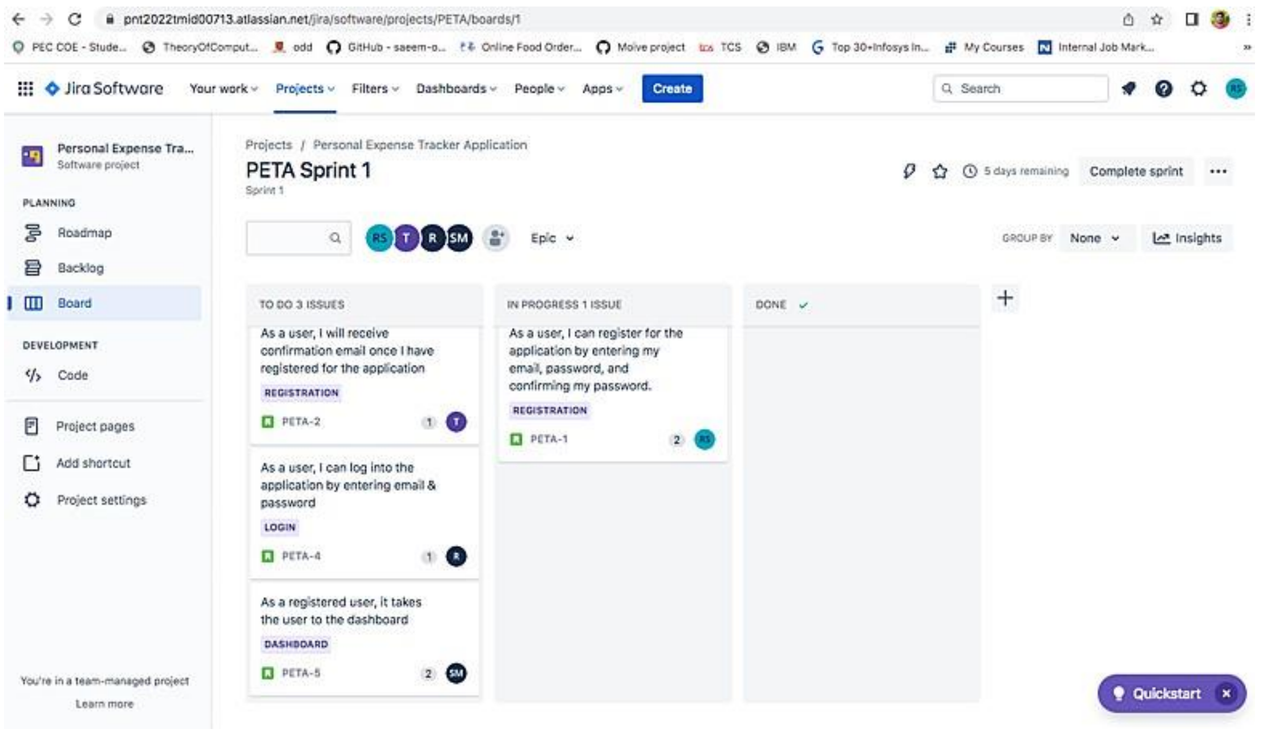
6.2 Sprit Planning and Estimation

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date	Story Poin ts	Sprint Release Date

Sprint-1	20	6 Days	23Oct 2022	28 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	30 Oct 2022	04 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	06Nov 2022	11 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	13 Nov 2022	18 Nov 2022	20	19 Nov 2022

6.3 Reports from JIRA

Board



TO DO 3 ISSUES

As a user, I will receive confirmation email once I have registered for the application
REGISTRATION
PETA-2 1 T

As a user, I can log into the application by entering email & password
LOGIN
PETA-4 1 R

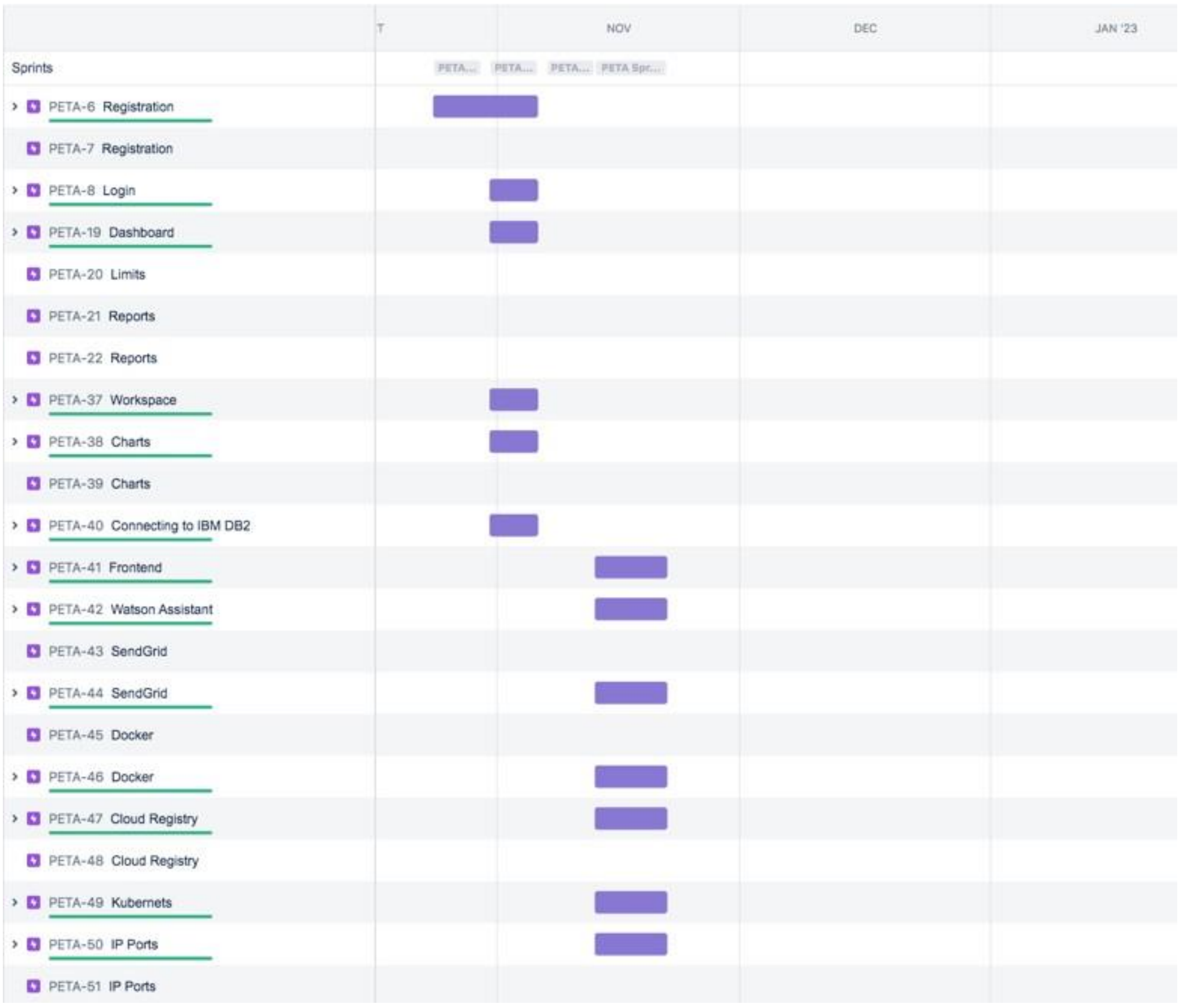
As a registered user, it takes the user to the dashboard
DASHBOARD
PETA-5 2 SM

IN PROGRESS 1 ISSUE

As a user, I can register for the application by entering my email, password, and confirming my password.
REGISTRATION
PETA-1 2 RS

DONE ✓

6.3.1 Road Map



Sprints
PETA-6 Registration
PETA-7 Registration
PETA-8 Login
PETA-19 Dashboard
PETA-20 Limits
PETA-21 Reports
PETA-22 Reports
PETA-37 Workspace
PETA-38 Charts
PETA-39 Charts
PETA-40 Connecting to IBM DB2
PETA-41 Frontend
PETA-42 Watson Assistant
PETA-43 SendGrid
PETA-44 SendGrid
PETA-45 Docker
PETA-46 Docker
PETA-47 Cloud Registry
PETA-48 Cloud Registry
PETA-49 Kubernets
PETA-50 IP Ports
PETA-51 IP Ports

```
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

import os


app = Flask(__name__)
app.config['SECRET_KEY'] = 'top-secret!'
app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'
app.config['MAIL_PORT'] = 587
app.config['MAIL_USE_TLS'] = True
app.config['MAIL_USERNAME'] = 'apikey'
app.config['MAIL_PASSWORD'] = 'SG.PU_eO2bJTI-HAnjene8ngw.P8zB2XEy14FM4Efn0wTV-
5JG98963QWXKZZza_bugb8'
app.config['MAIL_DEFAULT_SENDER'] = 'tarunvinodh@gmail.com'
mail = Mail(app)
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['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())


@app.route("/home")
def home():
   return render_template("homepage.html")

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


@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")


      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)




@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)




@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")



    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")


@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)





@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")



@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")








@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)
    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)

@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 = []
        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 )

@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)
```

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'] = 587
app.config['MAIL_USE_TLS'] = True
app.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)
```

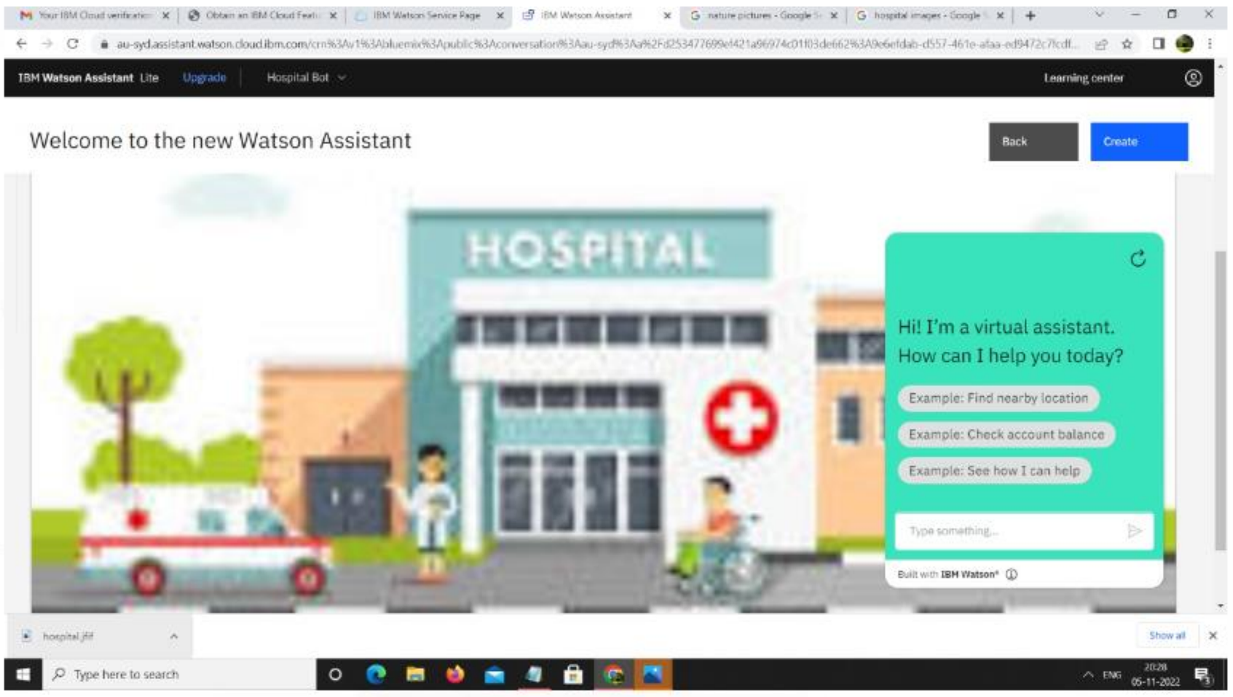
```
total=0
    for x in expense:
          total += x[4]

    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")
```

```
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("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.

```
<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>
```



Welcome to the new Watson Assistant

Hi! I'm a virtual assistant. How can I help you today?

Example: Find nearby location

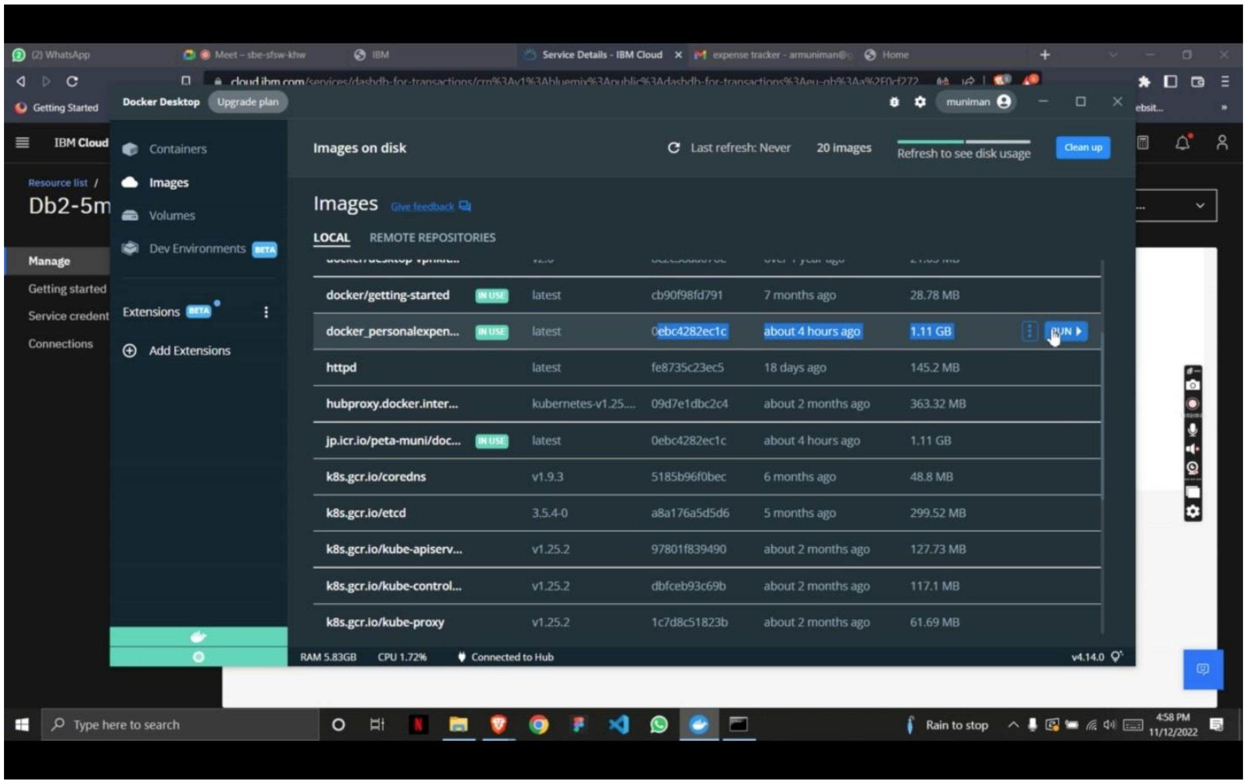
Example: Check account balance

Example: See how I can help

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.

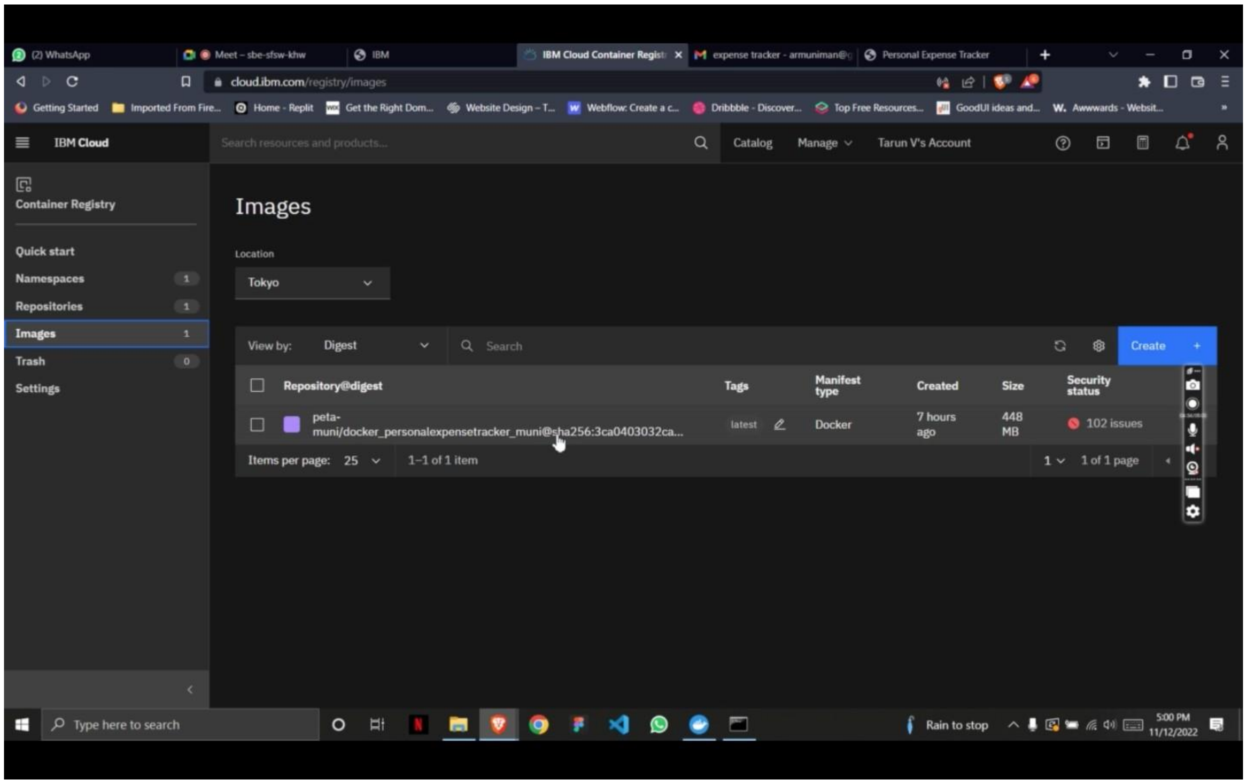
```
FROM python:3.6
WORKDIR /app
ADD . /app
COPY requirements.txt /app
RUN python3 -m pip install -r requirements.txt
RUN python3 -m pip install ibm_db
EXPOSE 5000
CMD ["python","app.py"]
```



docker/getting-started	IN USE	latest	cb90f98fd791	7 months ago	28.78 MB
docker_personalexpen...	IN USE	latest	0ebc4282ec1c	about 4 hours ago	1.11 GB
httpd		latest	fe8735c23ec5	18 days ago	145.2 MB
hubproxy.docker.inter...		kubernetes-v1.25....	09d7e1dbc2c4	about 2 months ago	363.32 MB
jp.icr.io/peta-muni/doc...	IN USE	latest	0ebc4282ec1c	about 4 hours ago	1.11 GB
k8s.gcr.io/coredns		v1.9.3	5185b96f0bec	6 months ago	48.8 MB
k8s.gcr.io/etcd		3.5.4-0	a8a176a5d5d6	5 months ago	299.52 MB
k8s.gcr.io/kube-apiserv...		v1.25.2	97801f839490	about 2 months ago	127.73 MB
k8s.gcr.io/kube-control...		v1.25.2	dbfceb93c69b	about 2 months ago	117.1 MB
k8s.gcr.io/kube-proxy		v1.25.2	1c7d8c51823b	about 2 months ago	61.69 MB

7.1.6 IBM Cloud Container Registry

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



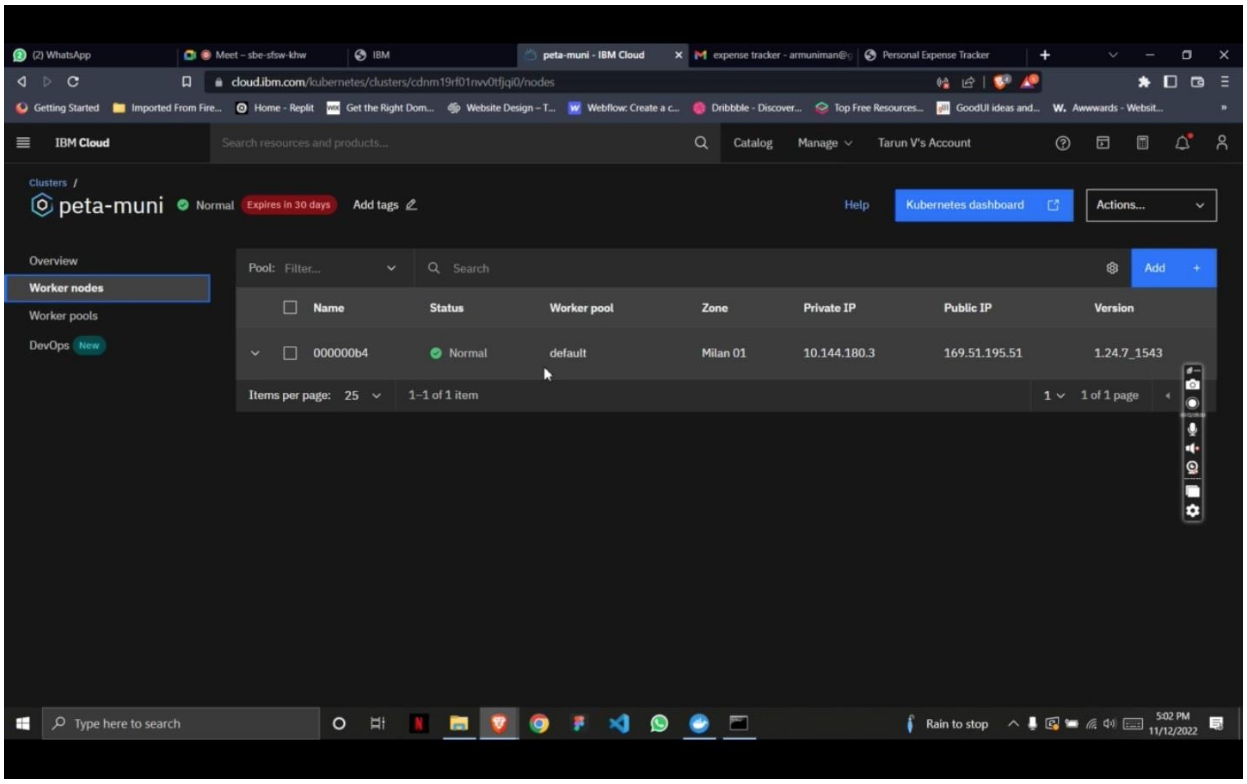
Repository@digest	Tags	Manifest type	Created	Size	Security status
peta-muni/docker_personalexpensetracker_muni@sha256:3ca0403032ca...	latest	Docker	7 hours ago	448 MB	102 issues

Items per page: 25 1–1 of 1 item 1 of 1 page

7.1.7 Kubernetes

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

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: flask-node-deployment
spec:
 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
```



Name	Status	Worker pool	Zone	Private IP	Public IP	Version
000000b4	Normal	default	Milan 01	10.144.180.3	169.51.195.51	1.24.7_1543

Items per page: 25 1–1 of 1 item 1 of 1 page

TESTING :

Date	16th November 2022
Team ID	PNT2022TMID23145
Project Name	Personal Expense Tracker
Batch Number	B9-3A5E

Testcase id	Feature type	Component	Test scenario	Pre-Requisite	Steps to execute	Test Data	Expected Result	Actual result	status	comments	TC for Automation	Bug id	Executed by
1.	Functional	Login Page	Verify user is able to login into the application		1.Open the personal expense tracker application 2. Login with user credentials 3. Verify logged into user account	User name:lavanya12 Password: lavanyabh@	Login successful	Working as expected	pass		N		Lavanya BH
2.	Functional	Sign up page	Verify user is able to sign up in the application		1.Open the personal expense tracker 2.Enter the details and create a new user 3.Verify if user is created	User name:priya25 Password: priya$25 Name: Lakshmi priya DOB:25.03.2022 Password Reenter: priya$25	Account created successfully	Working as expected	Pass		N		Lakshmi priyal G
3.	Functional	Dashboard page	Verify if all the user details are stored in data base		1.Open the personal expense tracker application 2.enter the details and create a new user 3. Verify if user is created and inserted into DB table	Username: raga52 Password: sruthi@2	User should navigate to user account home page	Working as expected	Pass				Raga Sruthi J
4.	Functional	Login page	Verify user is able log into the applic		1.Enter url and click go 2.click on sign in button 3. Enter invalid user name or email in email	Username: nithisha@gmail.com Password: nithisha@2	Application should show incorrect email or password. Validation message	Working as expected	Pass				Nithisha V

			ation with invalid credentials		text box 4. enter valid password in password text box . 5.Click on log in button								
5.	Functional	Login page	Verify user is able log into application with invalid credentials		1.Enter url and click go 2.Click on sign in button 3.Enter invalid user name or email in email text box 4. Enter valid password in password text box 5. Click on login button	Username: nithisha@gmail.com Password: nithisha@2	Application should show Incorrect email or password Validation message	Working as expected	Pass				Nithisha V

8.2 User Acceptance Testing

1. Purpose of Document

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

2. Defect Analysis

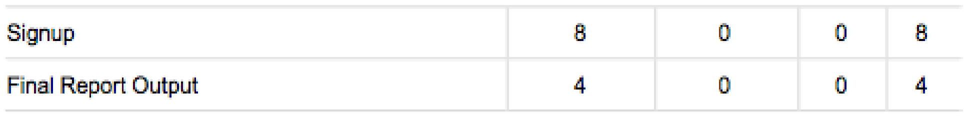
This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	8	15
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	9	2	4	11	20
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	0	1	8
Totals	22	14	11	22	51

3. Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Interface	7	0	0	7
Login	43	0	0	43
Logout	2	0	0	2
Limit	3	0	0	3



Signup	8	0	0	8
Final Report Output	4	0	0	4

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.

13. APPENDIX

13.1 Github Link

https://github.com/IBM-EPBL/IBM-Project-4220-1658724565

13.2 Project Demo Link

https://drive.google.com/file/d/1I3q4e2DFjphp8KEnDV_M_PWd-nCORr9r/view?usp=sharing

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_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("/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")

        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(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. " + 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)
    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():


      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 = []
        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">
```

```html
      <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/v1
kind: Deployment
metadata:
  name: flask-node-deployment
spec:
  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

```
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("demo123demo987@gmail.com", "taryluhlooidfwvj")
    message  = 'Subject: {}\n\n{}'.format(SUBJECT, TEXT)

    s.sendmail("demo123demo987@gmail.com", email, message)
    s.quit()
```