

PERSONAL EXPENSES TRACKER

Team ID: PNT2022TMID03120

A PROJECT REPORT

Submitted by

MADHUSUDHAN NS(19EUEC076)

MANISHKUMAR V (19EUEC078)

MANANITHYANANDHAM T M (19EUEC077)

MASSODH M (19EUEC080)

in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING

SRI KRISHNA COLLEGE OF ENGINEERING AND TECHNOLOGY

(An Autonomous Institution, Affiliated to Anna University Chennai - 600 025)

NOVEMBER 2022

ABSTRACT

Tracking regular expense is a key factor to maintain a budget. People often track expense using pen and paper method or take notes in a mobile phone or a computer. These processes of storing expense require further computations and processing for these data to be used as a trackable record. In this work, we are proposing an automated system to store and calculate these data. It is an application that runs on Android smartphones. By using this application, users can save their expense by simply scanning the bills or receipt copies. This application extracts the textual information from the receipts and saves the amount and description for further processing. It also monitors user's income by tracking the received SMS's from the user's saving accounts. By calculating income and expense it produces the user's balance in monthly and yearly basis. Overall, this is a smart automated solution for tracking expense.

1 INTRODUCTION

1.1 PROJECT OVERVIEW:

When it comes to tracking expenses, you can make your system as simple as collecting receipts and organizing them once a month.

You might get a little more information from other expense tracking systems (listing them in a spreadsheet, using money management software or even choosing an online application), but all methods have one thing in common: you have to get in the habit of thinking about your expenses.

It's very easy to misplace a receipt or forget about any cash you spent. You may even think that a cup of coffee or a trip to the vending machine isn't worth tracking — although those little expenses can add up amazingly fast.

There are all sorts of opportunities to throw a kick into your plan to track expenses. You have to get in the habit of doing so, to reduce those lapses, and make sure that the data you're basing financial decisions on is solid.

This project will request the clients to add their expenses and in view of their costs, wallet status will be refreshed which will be noticeable to the client.

- The user interacts with the application.
 - Application 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.
-
- Setting up Application Environment

- Create Flask project
- Work with IBM Cloud CLI, Docker CLI, Sendgrid
- Implementation of Web Application
- Create UI to Interact with the application
- Connect IBM DB2 with Python
- Integration of Sendgrid Service with Python
- Deployment of Cloud Application
- Containerize the application
- Upload Image in IBM container directory
- Deploy on Kubernetes Cluster

1.2 PURPOSE :

- Help the people to track their expenses.
- Alert users when they exceed the limit of their budget.
- A personal finance app will not only help you with budgeting and accounting but also give you helpful insights about financial management

2 LITERATURE SURVEY :

2.1 EXISTING PROBLEM :

- Lack of visual analytics for visual data.
- Lack of support for splitting up group expenses.
- Most of the applications are used only for personal use.
- Most of the applications does not incorporate shared group expenses.
- Efforts has to be made to include each and every transactions into the input field.

2.2 REFERENCES :

SPENDING TRACKER :

A Smart Approach to Track Daily Expenses

Authors:

UP Singh, AK Gupta, Dr. B. Balamurugan

Description:

In this paper, a Java GUI based application was proposed to assure that it will Help its users to manage the cost of their daily expenditure. It will guide them and aware them of their daily expenses. The proposed design contained the basic modules for Adding and viewing expenses, managing expense categories. Supports CRUD Operations on expense data.

Year: 2021

Technologies: Java

EXPENSE TRACKER APPLICATION

Authors: Velmurugan.R, Mrs.P.Usha

Description:

This is an android based application that allows users to maintain a computerized Diary to track expenses on a day-to-day basis to stay on budget and know expenses That are represented via a graphical representation with special features of distributing Expenses in different categories suitable for the user.

Year: 2021

Technologies: Java, XML, and MySQL

STUDENT EXPENSE TRACKING APPLICATION

Authors: Saumya Dubey , Pragya Dubey , Rigved Rishabh Kumar , Aaisha Khatoon

Description:

This is an android application which is used to track the daily expenses of a Student. It is like a digital diary that keeps a record of expenses done by a student. The Application keeps track of money spent and the earnings of both of the students on a Day-to-day basis. It also has the feature that it gives warning messages if we are Exceeding our expenses and hence, we can limit our expenses and avoid Overspending. If you spend less money than the daily expense allowed amount, the Money left after spending is added into the user's savings.

Year: 2022

Technologies: Java

EXPENSE TRACKER USING STATISTICAL ANALYSIS

Authors: Muskaan Sharma, Ayush Bansal, Dr. Raju Ranjan, Shivam Sethi

Description:

In this paper, an approach has been proposed on how to efficiently manage house-old budget. This application will allow users to keep track of their expenses. This novel expense tracker uses statistical analysis which is going to keep a track of your expenses and would even give you results accordingly.

Year: 2021

Technologies: Java

BUDGET ESTIMATOR ANDROID APPLICATION

Authors: Namita Jagtap, Priyanka Joshi, Aditya Kamble

Description:

The system known as Budget Estimator is designed to manage the application user 's daily expenses in a more efficient and manageable way. This project is about mobile application Expenses system with geo-location tracking, based on the location of the user, it using Google Places, to check, the available store in the area, provides a notification for offers purpose, In term of security design, this system may implement a login authentication such as OTP message to your mobile device, this function may bring more security confidence to user. To reduce manual calculations, we propose an application which is developed by android. This application allows users to maintain a digital automated diary.

Year: 2021

Technologies: Java

SAWANT-EXPENSE TRACKER

Authors: Atiya Kazi, Praphulla S. Kherade , Raj S. Vilankar , Parag M

Description:

In this approach, the application keeps track of the Income and Expenses of both users on a day-to-day basis. This application takes the income of a user and manages its daily expenses so that the user can save money. If you exceed the daily expense allowed amount it will give you a warning, so that you don't spend much and that specific day. If you spend less money than the daily expense allowed amount, the money left after spending is added into the user's savings. The application generates report of the expenses of each end of the month.

Year: 2021

Technologies: Java

2.3 PROMBLEM STATEMENT :

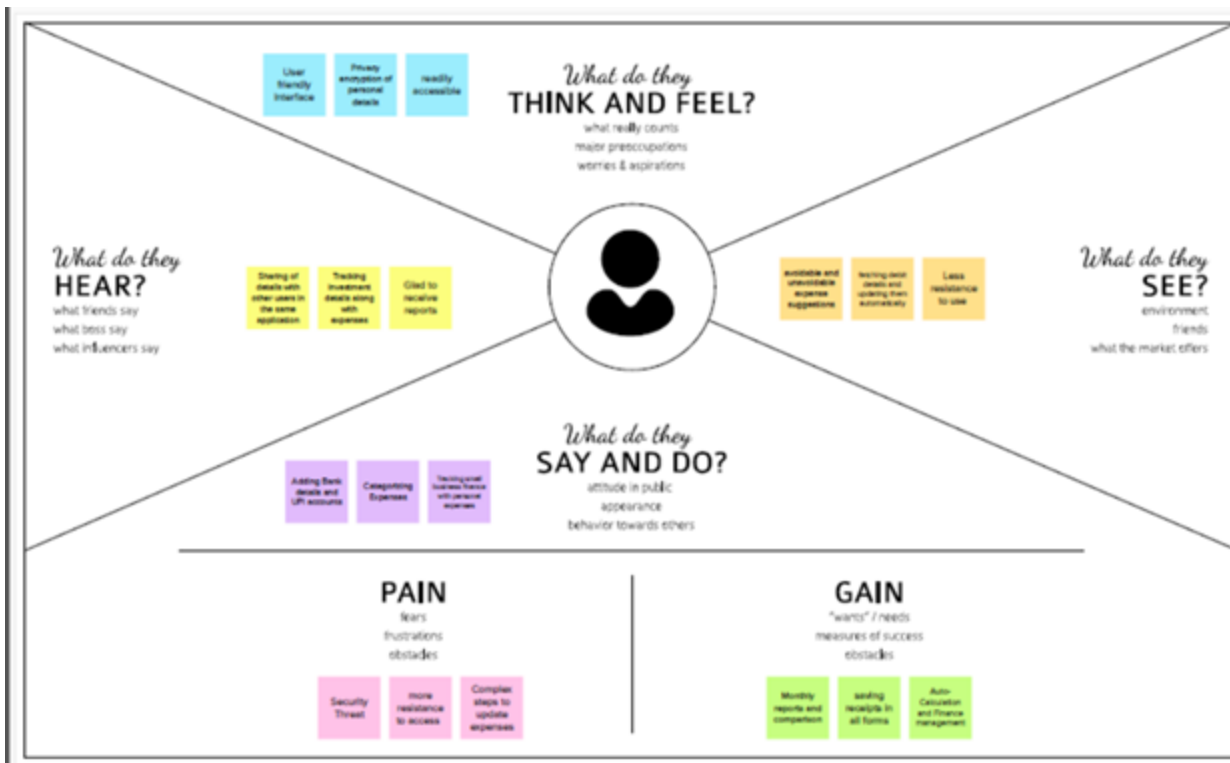
Who does the problem affect?	Investors, savers, big spenders, debtors ,shoppers, budgetconscious consumers.
What are the boundaries of the problem?	Expense tracker for working individuals, students, commonpeople.
What is the issue?	To be vigilant about the expense spent, increases financial stress. Being indecisive about the finances may result in less financial security and exceed the budget.

When does this issue occur?	When using wrong budgeting techniques. When not tracking the expenses doesn't help you to know the amount that is actually spent.
Where is the issue occurring?	Working individuals who find it difficult to track their expenses
Why is it important that we fix the problem?	Fixing this issue, brings accountability and helps to be intentional with the income by assign it to spending, saving and giving. This leads to financial stability.

- Abella, who is a shopaholic, finds it hard to control her desire to shop. To stop her from overindulging in impulsive purchases, she needs to track her expenses and hold herself accountable.
- John, who is interested to invest in stocks, finds it difficult to figure out the expense that he can spend on investing stocks. With the help of expense tracking, he can easily plan out the expenses for investing in an efficient way.
- Akshay, is a high school student, who usually gets a limited allowance from his parents. So tracking his expenses and good budgeting technique allows him to spend on his regular expenses as well as on himself.
- Udhay, who is a novice budgeter, finds it tedious to track and manage the expenses amongst his busy schedule. Prioritizing his expenses will help him to curtail his unnecessary expenditures.

3. IDEATION & PROPOSED SOLUTION :

3.1 EMPATHY MAP CANVAS :



3.2 IDEATION & BRAINSTORMING:

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

PROBLEM

How might we [solve your problem statement]?

Key rules of brainstorming

To run an smooth and productive session

Stay on topic

Defer judgments

Go for volume

Encourage wild ideas

Listen to others

If possible, let others lead

2 Brainstorm

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

10 minutes

Tip

Be open to lots of ideas that are initially silly or out there. It's better to have a lot of ideas and then filter them down to the best ones.

MAINTAINING

1. Maintain focus on the problem statement.

2. Encourage all ideas, no matter how silly or out there.

3. Encourage wild ideas.

4. Encourage others to build on each other's ideas.

5. Encourage others to lead.

6. Encourage others to build on each other's ideas.

MAINTAINING

1. Maintain focus on the problem statement.

2. Encourage all ideas, no matter how silly or out there.

3. Encourage wild ideas.

4. Encourage others to build on each other's ideas.

5. Encourage others to lead.

6. Encourage others to build on each other's ideas.

MAINTAINING

1. Maintain focus on the problem statement.

2. Encourage all ideas, no matter how silly or out there.

3. Encourage wild ideas.

4. Encourage others to build on each other's ideas.

5. Encourage others to lead.

6. Encourage others to build on each other's ideas.

MAINTAINING

1. Maintain focus on the problem statement.

2. Encourage all ideas, no matter how silly or out there.

3. Encourage wild ideas.

4. Encourage others to build on each other's ideas.

5. Encourage others to lead.

6. Encourage others to build on each other's ideas.

Person 1

Person 2

Person 3

Person 4

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 or label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

Multiple scenarios

1. Multiple scenarios

2. Multiple scenarios

3. Multiple scenarios

4. Multiple scenarios

5. Multiple scenarios

6. Multiple scenarios

7. Multiple scenarios

8. Multiple scenarios

9. Multiple scenarios

10. Multiple scenarios

Interface & organization

1. Interface & organization

2. Interface & organization

3. Interface & organization

4. Interface & organization

5. Interface & organization

6. Interface & organization

7. Interface & organization

8. Interface & organization

9. Interface & organization

10. Interface & organization

Tooling requirements

1. Tooling requirements

2. Tooling requirements

3. Tooling requirements

4. Tooling requirements

5. Tooling requirements

6. Tooling requirements

7. Tooling requirements

8. Tooling requirements

9. Tooling requirements

10. Tooling requirements

Goal & priority

1. Goal & priority

2. Goal & priority

3. Goal & priority

4. Goal & priority

5. Goal & priority

6. Goal & priority

7. Goal & priority

8. Goal & priority

9. Goal & priority

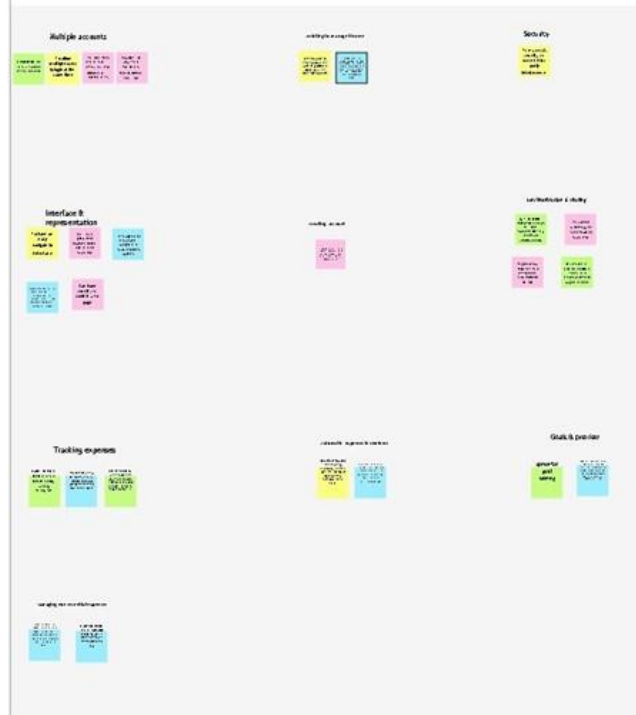
10. Goal & priority

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 can break it up into smaller sub-groups.

20 minutes

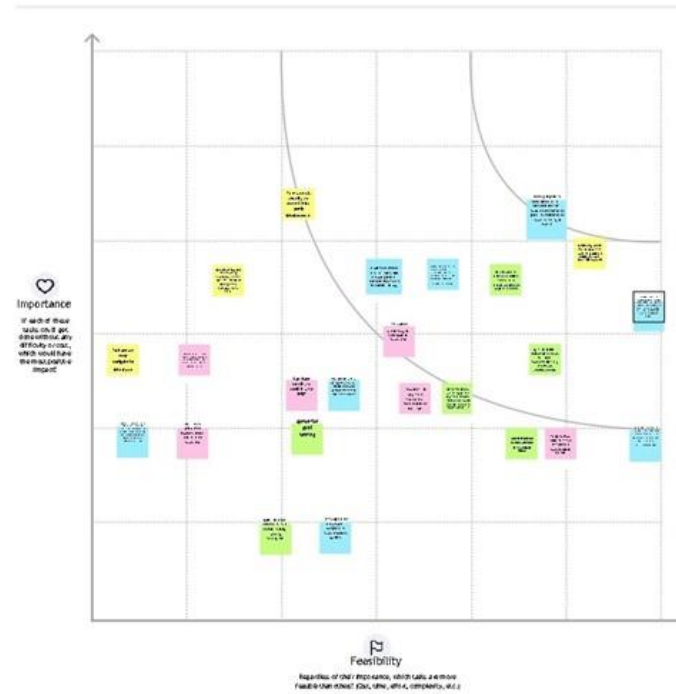


4

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



5

Participants can use this grid to place all ideas on the grid. The facilitator can then use the grid to help the team decide which ideas are important and which are feasible.

3.3 PROPOSED SOLUTION:

2.	Idea / Solution description	Due to the busy and hectic lifestyle people tend to overlook their budget and end up spending an excessive amount of money since they usually didn't plan their budget wisely. user cannot predict future expenses. While they can write down their expenses in a excel spreadsheet, their lack of knowledge in managing finances will be a problem
3.	Novelty / Uniqueness	This application tracks your every expenses anywhere and anytime without using the paper work. Just click and enter your expenditure. to avoid data loss, quick settlements and reduce human error. To provide the pie chart or graph lines in this application.
4.	Social Impact / Customer Satisfaction	Using this application one can track their personal expenses and frame a monthly/annual budget. If your expense exceeded than specified limit, the application will show you an alert message in form of a pie chart.
5.	Business Model (Revenue Model)	Business people can use subscription/premium feature of this application to gain revenue.
6.	Scalability of the Solution	IBM cloud will automatically allocate the storage for the users.

3.4 PROBLEM SOLUTION FIT:

Define CS, fit into CC	1. CUSTOMER SEGMENT(S) Who is your customer? i.e. working parents of 0-5 y.o. kids <ul style="list-style-type: none"> Customers are those who spend money without keeping track of it or struggling to keep track of it Provides a whole lot of different categories of expenditure types to avoid mismatch of expenditure 	6. CUSTOMER CONSTRAINTS What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices <ul style="list-style-type: none"> Most of the solution available in the internet hosts a lot of adds limiting its usability The solution proposed here has a feature to view the expense graphically Also it has an alert via email feature if the expense exceeds the given limit 	5. AVAILABLE SOLUTIONS Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking <ul style="list-style-type: none"> Expense tracker applications which are available in both android and ios. Personal Expense tracker developed in this project 	Explore AS, differentiate
Focus on J&P, lay into BE, understand RC	2. JOBS-TO-BE-DONE / PROBLEMS Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides. <ul style="list-style-type: none"> The objective of this application is to enable customers to keep track of their expenses. The customers are provided with categories for the expenses. They also get an option to view the expenses as a graphical representation given the period of 1 year, 6 months etc. 	9. PROBLEM ROOT CAUSE What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations. <ul style="list-style-type: none"> Improper expenses lead to heavy tax. Makes business forecasting easier Saves a lot of money Existence of lot of payment methods leads to problem in manual expense tracking 	7. BEHAVIOUR What does your customer do to address the problem and get the job done? i.e. directly related, find the right solar panel installer, calculate usage and sending, indirectly associated: customers spend less time on volunteering work (i.e. Greenpeace) <ul style="list-style-type: none"> Start using the expense tracker app Makes sure he categorize the expense done in order to save money Set up a monthly limit on the expense done Have a separate in-hand wallet account and Online accounts 	Focus on J&P, lay into BE, understand RC
Identify strong TR & EM	3. TRIGGERS What triggers customers to act? i.e. seeing their neighbour installing solar panels, reading about a more efficient solution in the news. <ul style="list-style-type: none"> Understanding the fact the customers can save a lot of money by these expense apps 	10. YOUR SOLUTION If you are working on an existing business, write down your current solution first, fit in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fit in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour. <ul style="list-style-type: none"> Design a flask based personal expense tracker application Enable email based expense alerts using sendgrid framework Provide a option for graphical expense view 	8. CHANNELS of BEHAVIOUR 8.1 ONLINE What kind of actions do customers take online? Extract online channels from #7 <ul style="list-style-type: none"> Expense trackers online come with a lot of ads which on clicking steals data like account number if provided 8.2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development. <ul style="list-style-type: none"> Make sure they are aware of the tax rules by reading the available books to make them tax read 	Extract online & offline CH of BE

4. REQUIREMENT ANALYSIS

4.1 REQUIREMENT ANALYSIS :

a. FUNCTIONAL REQUIREMENT :

FR No.	Functional Requirement (Epic)	Sub Requirement (Story/ Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIn
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Multiple login	Many users can log in by using separate mail identities. Also, using the mail identity, the user can log into any device.
FR-4	Alerting the user	A limit must be set on the amount of money to be spent. Whenever the user exceeds the limit, he will be notified through mail or text message.
FR-5	Reporting	An analysis on the expenses should be done. Based on the analysis, a detailed report (in any graphical form) must be generated to help the user in accounting and budgeting.

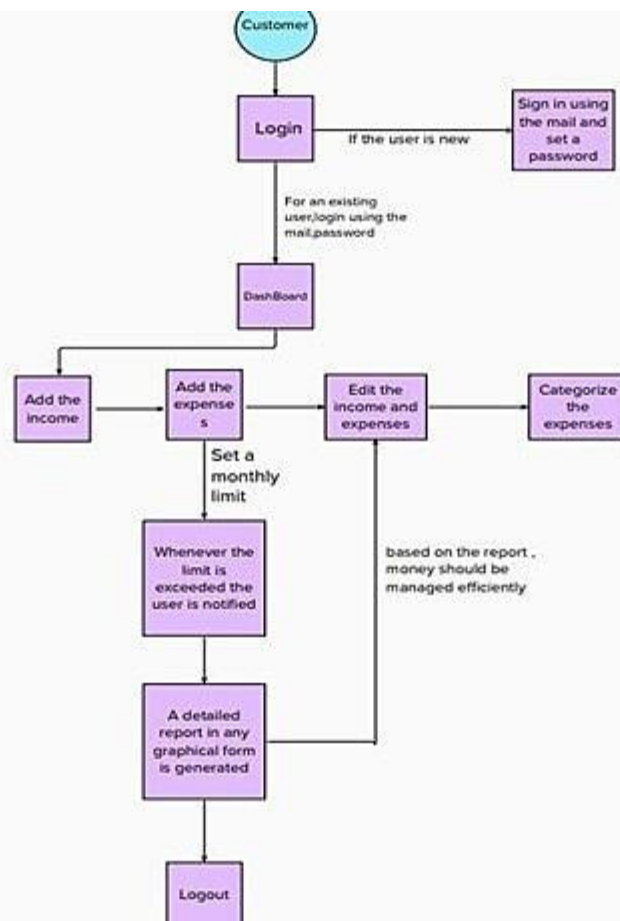
b. NON FUNCTIONAL REQUIREMENT :

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	The interface must be user friendly that makes it easy to use for all types of users. The basic features must be available free of cost to users.
NFR-2	Security	The application should have multi-factor authentication when logging in. Also, banking data must be secured by some encryption technology.
NFR-3	Reliability	The transaction must rollback if there is any technical or network issue. The data must be saved when updation of data fails in between the process. Even if there is a failure, it should be restored within a few minutes.
NFR-4	Performance	The application must not take more than 30 seconds to load. The response time should be quick even when there is heavy traffic.
NFR-5	Availability	When the app is being updated, except for the module that is being updated, the rest can be used.
NFR-6	Scalability	The app must be designed to work efficiently even when there is heavy traffic.

5.PROJECT DESIGN :

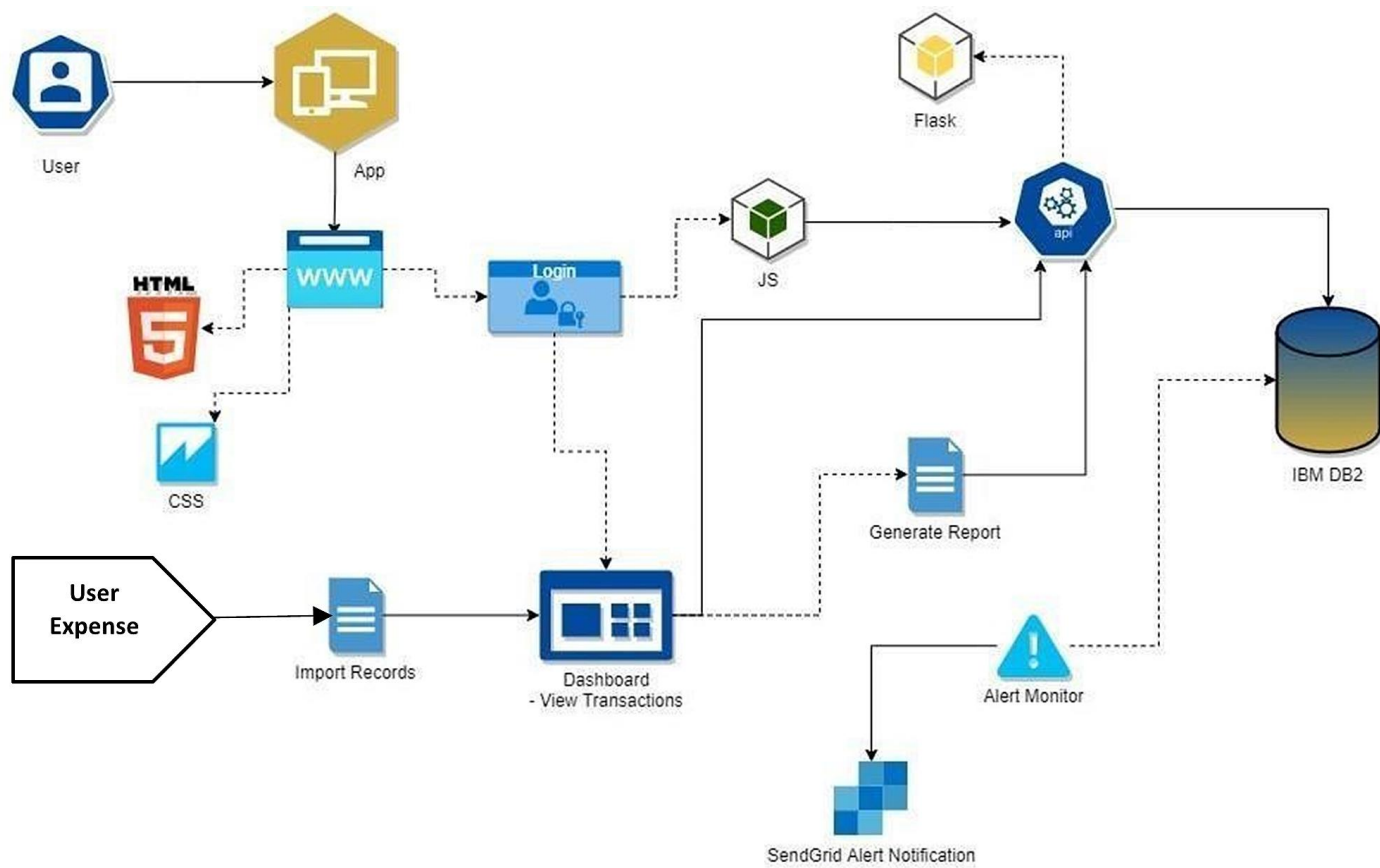
5.1 DATAFLOW DIAGRAM :

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 enter and leaves the system, what changes the information, and where data is stored.

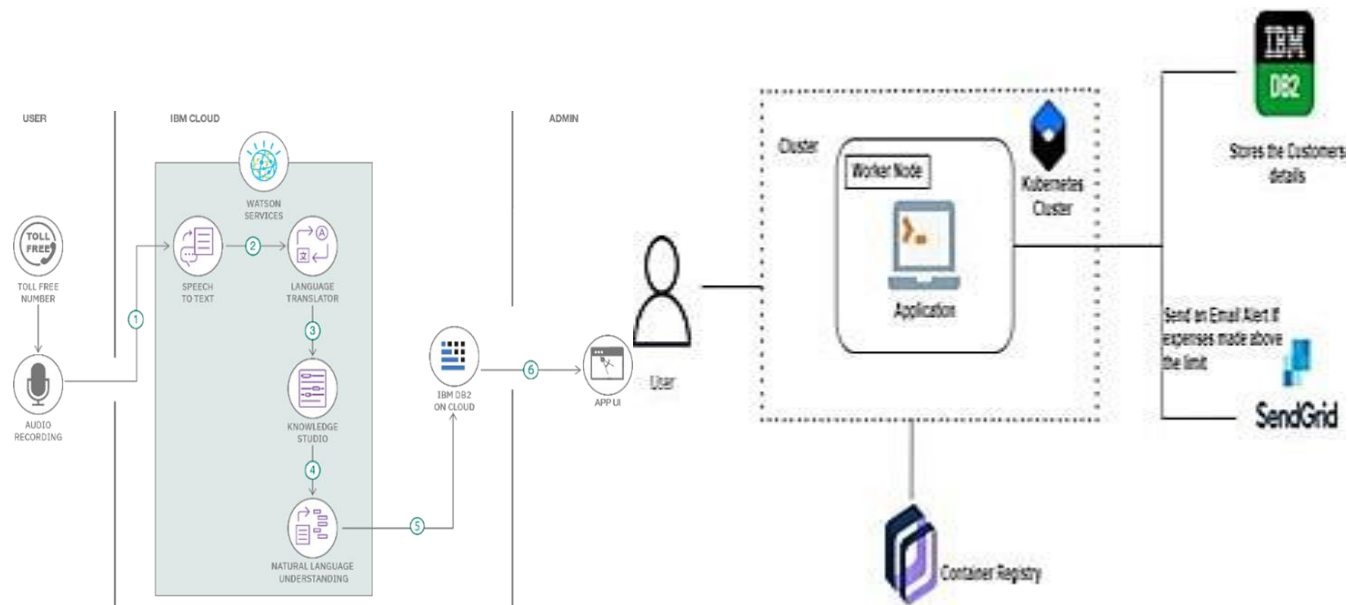


5.2 SOLUTION AND TECHNICAL ARCHITECTURE :

SOLUTION ARCHITECTURE:



TECHNICAL ARCHITECTURE :



5.3 USER STORIES:

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

Table-1 : Components & Technologies:

S. No	Component	Description	Technology
1.	User Interface	The user can Interact with the application with use of IBM Watson Chatbot.	HTML, CSS, JavaScript / Angular-js / React-js etc.
2.	Application Logic-1	The application contains the sign in/sign up where the user will login into the main dashboard.	Java / Python
3.	Application Logic-2	Dashboard contains the fields like Add income, Add Expenses, Save Money, Add budget, Profile etc...	IBM Watson STT service
4.	Application Logic-3	The user will get the expense report in the Statistics form and get alerts if the expense limit exceeds.	IBM Watson Assistant
5.	Database	The Income and Expense data are stored in the IBM Cloud database.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM-Cloudant etc.
7.	File Storage	IBM Cloud Storage used to store the financial data of the user	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.

9.	External API-2	Purpose of External API used in the application	Aadhar API, etc.
10.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
11.	Infrastructure (Server / Cloud)	Application Deployment on Local System/ Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S. No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask Framework in Python is used to implement this application to connect the UI and the Backend.	Flask
2.	Security Implementations	This Application Provides high security to the user financial data. It can be done by using the Container Registry in IBM cloud	SHA-256, Encryptions, IAM Controls, OWASP etc.
3.	Scalable Architecture	Expense Tracker is a lifetime access web application. Its demand will increase when the user's increases.	Container Registry, Kubernetes Cluster.
4.	Availability	This application will be available to the user at any part of time using the Internet.	Container Registry, Kubernetes Cluster

5.	Performance	The performance will be high because there will be no network traffics in the application.	Container Registry, Kubernetes Cluster.
----	-------------	--	--

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Webuser)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Gmail	I can access my account	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can access the dashboard	High	Sprint-1
	Dashboard	USN-5	As a user, I can add income and expenses in the application	I can keep track of the expenses	high	Sprint-1

		USN-6	As a user, I can change the expenses as I spend and can even categorise them	I can keep track, account and budget for the expenses	High	Sprint-
	Alerting	USN-7	As a user, I can set a limit on the amount of money that can be spent.	Whenever the limit is exceeded, the user gets notified through mail or text messages.	High	Sprint-1
	Reporting	USN-8	As a user, the expense that is spent can be categorised and a report (in any graphical form) can be generated.	I can manage money efficiently from the report	High	Sprint-1
Customer (Mobile user)	Accounting	USN-9	As a user, the income and expenses can be added and categorised	From the report generated, money management could be done	Medium	Sprint-2
Administrator	Supervising and updating	USN-10	As an administrator, I supervise and update from the user feedback	Updating the app makes it more user friendly	Medium	Sprint-1

6. PROJECT PLANNING AND SCHEDULING:

6.1 SPRINT PLANNING & ESTIMATION

Use the below template to create product backlog and sprint schedule

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Madhusudhan NS
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Manish Kumar
Sprint-1	Login	USN-3	As a user, I can register for the application through Gmail	1	High	Mananithyanantham
Sprint-1	Dashboard	USN-4	As a user, I can log into the application by entering email & password	2	High	Masoodh
Sprint-2	Workspace	USN-1	Workspace for personal expense tracking	2	High	Madhusudhan NS
Sprint-2	Charts	USN-2	Creating various graphs and statistics of customer's data	1	Medium	Manish Kumar
Sprint-2	Connecting to IBMDB2	USN-3	Linking database with dashboard	2	High	Mananithyanantham
Sprint-2		USN-4	Making dashboard interactive with JS	2	High	Masoodh
Sprint-3		USN-1	Wrapping up the server side works of frontend	1	Medium	Madhusudhan NS

Sprint-3	Watson Assistant	USN-2	Creating Chatbot for expense tracking and for clarifying user's query	1	Medium	Manish Kumar
Sprint-3	SendGrid	USN-3	Using SendGrid to send mail to the user about	1	Low	Mananithyanantham
Sprint-3		USN-4	Integrating both frontend and backend	2	High	Masoodh

Sprint-4	Docker	USN-1	Creating image of website using docker	2	High	Madhusudhan NS
Sprint-4	Cloud Registry	USN-2	Uploading docker image to IBM Cloud registry	2	High	Manish Kumar
Sprint-4	kubernetes	USN-3	Create container using the docker image and hosting the site	2	High	Mananithyanantham
Sprint-4	Exposing	USN-4	Exposing IP/Ports for the site	2	High	Masoodh

6.2 SPRINT DELIVERY SCHEDULE :

Project Tracker, Velocity & Burndown Chart:

Sprint	Total Story Points	Duration	Sprint 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	2 Days	12 Nov 2022	14 Nov 2022	20	1. Nov 2022

6.3 REPORT FROM JIRA:

BACKLOG:

- ☐ PETA-1 As a user, I can register for the application by entering my email, password, and confirming my pass... **REGISTRATION** **2** **IN PROGRESS**
- ☐ PETA-2 As a user, I will receive confirmation email once I have registered for the application **REGISTRATION** **1** **TO DO**
- ☐ PETA-4 As a user, I can log into the application by entering email & password **LOGIN** **1** **TO DO**
- ☐ PETA-5 As a registered user, it takes the user to the dashboard **DASHBOARD** **2** **TO DO**

+ Create issue

▼ **PETA Sprint 2** 31 Oct – 7 Nov (4 issues)

7 **0** **0** **Start sp**

- ☐ PETA-3 Showing the workspace for personal expense tracker **WORKSPACE** **2** **TO DO**
- ☐ PETA-23 Creating various graphs and statistics of customers data **CHARTS** **1** **TO DO**
- ☐ PETA-24 To link the database with dashboard **CONNECTING TO IBM DB2** **2** **TO DO**
- ☐ PETA-28 To make a dashboard with javascript **DASHBOARD** **2** **TO DO**

+ Create issue

▼ PETA Sprint 3 7 Nov – 14 Nov (4 issues)

5 0 0 Start sprint

- ❑ PETA-15 To wrap up the server side works of frontend FRONTEND 1 TODO ▾
- ❑ PETA-29 Creating chatbot WATSON ASSISTANT 1 TODO ▾
- ❑ PETA-31 Integrating SendGrid services SENDGRID 1 TODO ▾
- ❑ PETA-32 Integrating both frontend and backend 2 TODO ▾

+ Create issue



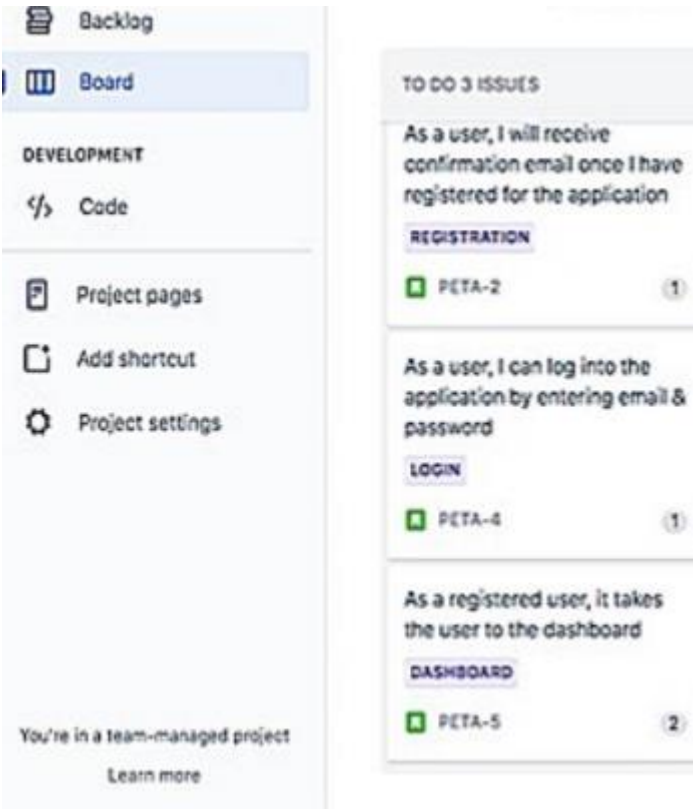
▼ PETA Sprint 4 14 Nov – 21 Nov (4 issues)

8 0 0 Start sprint

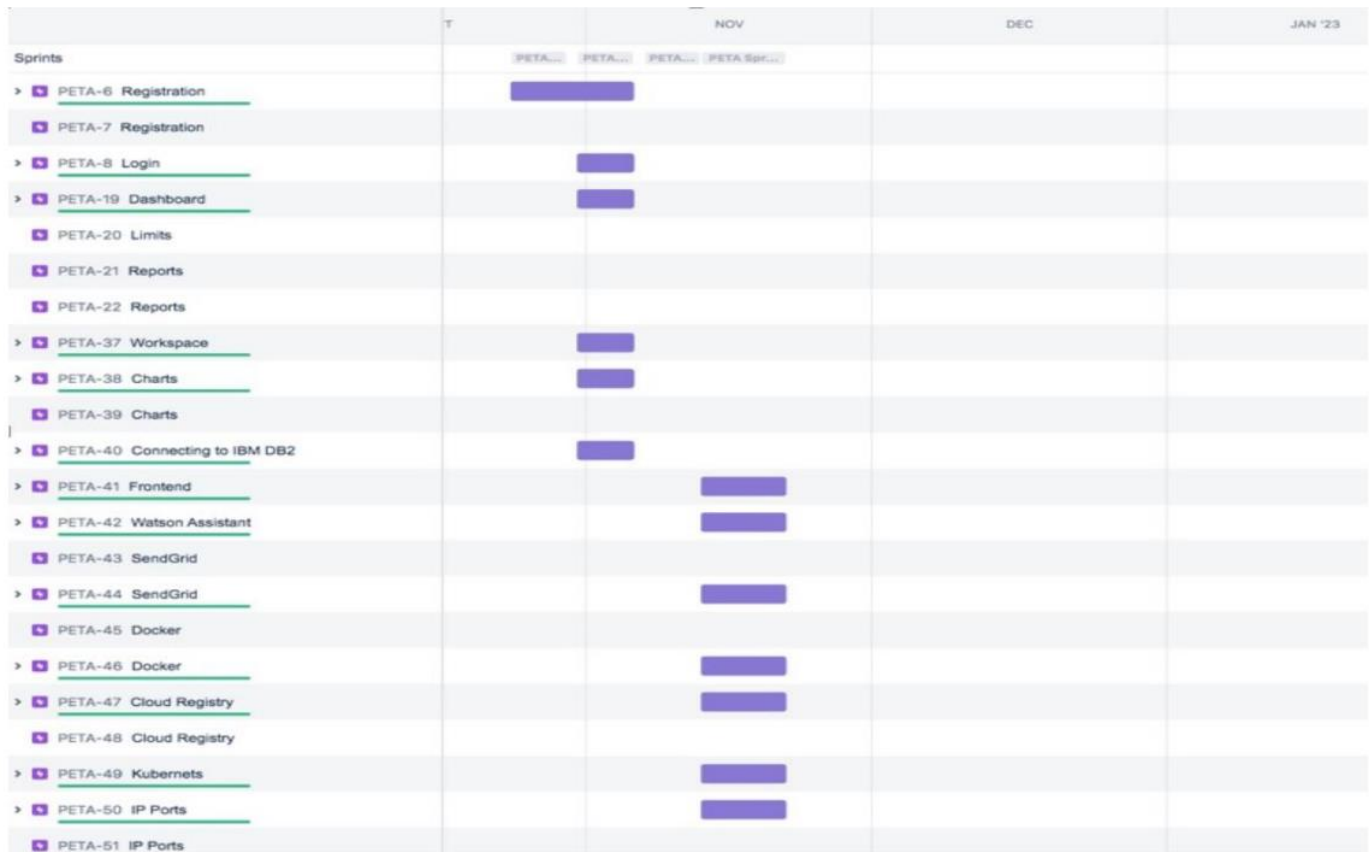
- ❑ PETA-17 To create images of website using docker DOCKER 2 TODO ▾
- ❑ PETA-33 To upload docker image to IBM Cloud Registry CLOUD REGISTRY 2 TODO ▾
- ❑ PETA-34 To create a container using docker image and hosting the site KUBERNETES 2 TODO ▾
- ❑ PETA-35 Exposing IP Ports for the site IP PORTS 2 TODO ▾

+ Create issue

BOARD:



ROAD MAP:



7. CODING & SOLUTIONING

7.1 Features

Feature 1: Add Expense

Feature 2: Update expense

Feature 3: Delete Expense

Feature 4: Set Limit

Feature 5: Send Alert Emails to users

Other Features: Track your expenses anywhere, anytime. Seamlessly manage your money and budget without any financial paperwork. Just click and submit your invoices and expenditures. Access, submit, and approve invoices irrespective of time and location. Avoid data loss by scanning your tickets and bills and saving in the app. Approval of bills and expenditures in real-time and get notified instantly. Quick settlement of claims and reduced human errors with an automated and streamlined billing process.

7.2 Codes:

app.py:

```
# -*- coding: utf-8 -*-  
"""
```

Spyder Editor

```
This is a temporary script file.  
"""
```

```
from flask import Flask, render_template, request, redirect, session  
# from flask_mysqlldb import MySQL # import MySQLdb.cursorsimport re
```

```
from flask_db2 import DB2 import ibm_dbimport 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;\n  
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'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z]+', 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'])))
    # texpanse = cursor.fetchall()
    # print(texpanse)

    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)    texpanse = []

    while dictionary1 != False:
        temp = []    temp.append(dictionary1["TN"])    temp.append(dictionary1["AMOUNT"])
    texpanse.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", texpanse = texpanse, 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'])))
```

```
# texpanse = cursor.fetchall()
```

```
# print(texpanse)
```

```
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) texpanse = []
```

```
while dictionary1 != False:
```

```
temp = []
```

```

        temp.append(dictionary1["DT"])        temp.append(dictionary1["TOT"])        texpanse.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", texpanse = texpanse, 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'])))
# texpanse = cursor.fetchall()
# print(texpanse)

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)
texpanse = []

while dictionary1 != False:
    temp = []      temp.append(dictionary1["MN"])      temp.append(dictionary1["TOT"])
    texpanse.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", texpanse = texpanse, 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-deploymentspec: 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: Servicemetadata:

```

name: flask-app-servicespec: selector:
  app: flask-app ports: - name: http protocol: TCP
  port: 80 targetPort: 5000 type: LoadBalancermanifest.yaml: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 sgimport 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("abcd@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 :

8.1 TEST CASES :

S.N O	Test Cases	Result
1	Verify the user is able to see the Sign up page when the user clicks the signup button in navigation bar	Positive
2	Verify the UI elements in the Sign up page	Positive
3	Verify the user is able to register into the application by providing valid details	Positive
4	Verify the user is able to see the sign in page when the user clicks the sign in button in navigation bar	Positive
5	Verify the UI elements in the Sign in page	Positive
6	Verify the user is able to login into the application by providing valid details	Positive

7	Verify the user is able to see the add expenses page when the user clicks the add expenses navigation bar	Positive
8	Verify the UI elements in the add expenses page	Positive
9	Verify the user is able to add expenses by providing valid details	Positive
10	Verify the user is able to see the home button in the add expenses page	Positive
11	Verify whether the expenses added can be deleted from the cloud	Positive
12	Verify whether the expenses added can be edited	Positive
13	Verify whether the date, month and year are valid while user entering the expenses	Positive
14	Verify whether the data types entered by the user in the add expenses page are float or integers	Positive
15	Verify whether expenses added previously still exist	Positive
16	Verify whether the expenses amount entered fits between in the range of integers or float datatypes	Positive

8.2 USER ACCEPTANCE TESTING :

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Results
LoginPage_TC_OO1	Functional	Home Page	Verify user is able to see the Login/Signup popup when user clicked on My account button	1. Click the my account button. 2. Check whether login/signup pop up appears	-	Login/Signup button should be visible
LoginPage_TC_OO2	UI	Home Page	Verify the UI elements in Login/Signup popup	1. Visit the Signup/Login page. 2. Verify the UI elements.	-	Application should show the following elements: a. email field b. password field c. Login button d. New user? link e. Last updated message
LoginPage_TC_OO4	Functional	Login page	Verify user is able to log into application with Valid credentials	1. Visit the login page. 2. Enter right credentials.	User id: madhusudhans@gmail.com Password: Keerthu	Application should show 'logged in' message

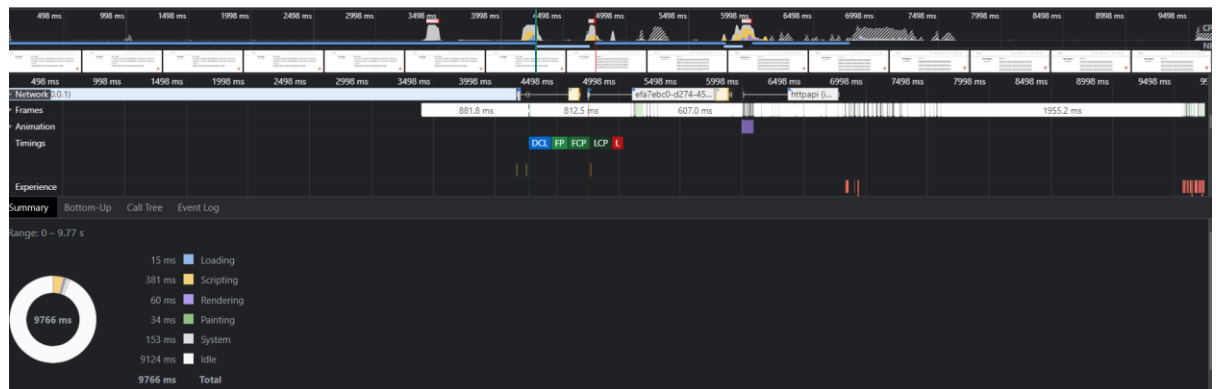
LoginPage_TC_OO5	Functional	Login page	Verify user is able to log into application with InValid credentials	<ol style="list-style-type: none"> 1. Visit the login page. 2. Enter wrong credentials 	User id: madhusudhans@gmail.com Password: madhu	Applic 'Incorr passwo messag
Add_expense_page_TC_001	Functional	Add expense page	Verify the user could add the expenses & delete them	<ol style="list-style-type: none"> 1. Visit the login page. 2. Enter right credentials. 3. Enter input data and delete them. 	Input data: 25	Applic the val deletes delete
Add_expense_page_TC_002	Functional	Add expense page	Verify the user enters only numerals in add expenses input.	<ol style="list-style-type: none"> 1. Visit the login page. 2. Enter right credentials. 3. Enter numerical values in input And not strings. 	Input data 1: 50 Input data 2: m	Applic numeri strings
Add_expense_page_TC_003	Functional	Add expense page	Verify the date, month and year are valid	<ol style="list-style-type: none"> 1. Visit the login page. 2. Enter right credentials. 3. Check whether date, month and year are valid. 	June 5, 1999	Applic display date,m

9. RESULTS :

9.1 PERFORMANCE METRICS

The performance metrics include page speed, time to title, bounce rate, time to start render, time to interact, DNS lookup speed, Requests per second, error rate, time to first byte/last byte and conversion rate. The performance metrics of our application are efficient and are given below for reference.

Performance Metrics



10. ADVANTAGES & DISADVANTAGES :

ADVANTAGES :

- With proper tracking of your finances, you will not be able to determine unnecessary spending. This spending, if saved, can easily add up to quite a bit.
- In this day and age, when expenses are going through the roof, it has become crucial that you learn to make your money work for you so that you can create a nest egg for the future.
- Has various components of updating and viewing users expenditure
- User can track his expenses by choosing a day and using various filtering Options to study expenses
- Visualization using pie chart with percentage view shows graphical Representation.
- This approach effectively keeps away from the manual figuring for trying

- Not to ascertain the pay and cost each month.
- With a daily expense manager, you will be able to allocate money to different priorities and this will also help you cut down on unnecessary spending. As a result, you will be able to save and be able to keep worry at bay.
- A daily money tracker helps you budget your money so that you use it wisely. If you find that every month your expenses are more than what you earn, it is time to put your house in order and get a money manager app that keeps track of your money without any problem.

DISADVANTAGES :

- Lack of visual analytics for expense data
- Lack of support for splitting group expenses
- Suitable for only Personal use.
- Errors are another common problem expense reports drafted with Excel. As is the case with all tasks performed manually and based on paper, it is extremely likely that business travellers who report expenses make involuntary mistakes: numbers entered wrongly, duplicated expenses or expenses relating to a previous settlement period, misapplication of the expense policy, etc.
- Frequent tracking of cash spending can allow one to catch and correct errors so that the budget plan is still able to be adhered to despite the mistake.

11. CONCLUSION:

Tracking regular expense is a key factor to maintain a budget. People often track expense using pen and paper method or take notes in a mobile phone or a computer. These processes of storing expense require further computations and processing for these data to be used as a trackable record. In this work, we are proposing an automated system to store and calculate these data. It is an application that runs on Android smartphones. By using this application, users can save their expense by simply scanning the bills or receipt copies. This application extracts the textual information from the receipts and saves the amount and description for further processing. It also monitors user's income by tracking the received SMS's from the user's saving accounts. By calculating income and expense it produces the user's balance in monthly and yearly basis. Overall, this is a smart automated solution for tracking expense.

12. FUTURE SCOPE:

1. Mobility

Businesses are becoming increasingly global and employees are more mobile than ever.

According to the Certify Expense Management Trends Report 2018, 47 percent of organizations felt that mobile applications and accessibility were a critical capability of expense reporting software.

Expense management software will begin to respond to this change in the manner that people work by facilitating mobility. Employees will be able to submit reports and managers can approve the claims from a smartphone. Also, mobile applications will become more intuitive and responsive, encouraging greater adoption.

2. Travel booking

According to a 2018 study by Tripactions, 90 percent of respondents believe that travel is important for growth. However, 50 percent of employees don't use the corporate travel solution offered by their organization owing to difficulties and the time taken to book a trip. They prefer using consumer channels.

Using multiple external vendors and disjointed channels to book travel can become tedious and lead to a lot of last-minute chaos. Also, the expenses have to be recorded accurately in the expense management system. It's imperative to integrate the same into expense management software.

3. Integrated system

In 2019, expense management software will begin to consolidate currently disparate actions into a seamless system with credit cards, bank accounts, accounting, payroll, CRM, and more in one place. All these systems share a lot of common information, and it makes perfect sense to connect all these to ensure data integrity.

When integrated with travel applications and travel management company (TMC) solutions, transactions can be entered directly into the system without the need for manual input. Corporate cards can also be linked to the software, which makes it easier to reconcile credit card statements with expense reports.

4. Optical character recognition

According to 66 percent of respondents from the Certify Expense Management Trends Report 2018, ease of use was the top-most feature needed. Traveling employees spend a lot of time filling out expense reports. By using expense management software, businesses brought down the time and cost of expense reporting. This will further go down with advancements in OCR technology.

Optical character recognition (OCR) will be one of the significant ways in which the ease-of-use of expense management tool will be enhanced. It eliminates the need to manually input data from receipts into forms. It will also facilitate mobility.

Employees could just snap a picture of a receipt and have the software extract all the necessary information from it to fill a report. OCR technology is accurate up to 95 percent and doesn't need too much human intervention to verify the information that's been uploaded.

5. Artificial intelligence

As machine learning and artificial intelligence (AI) evolve, they will add to the sophistication of expense management software. There will be improved ability to assign expense general ledger codes based on submissions and smarter categorization of expense types. Rudimentary versions of this exist in a few tools that use rule-based categorization.

AI-powered automation is also expected to minimize fraudulent expense reports. It can monitor, track, and approve expenses, and compliance-related issues will immediately be flagged and eliminated.

13. APPENDIX

SOURCE CODE:

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.cursorsimport re

from flask_db2 import DB2 import ibm_dbimport 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=tcPIP;
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)', (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'])))
    # texpanse = cursor.fetchall()
    # print(texpanse)

    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)    texpanse = []

    while dictionary1 != False:
        temp = []    temp.append(dictionary1["TN"])    temp.append(dictionary1["AMOUNT"])
texpanse.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", texpanse = texpanse, 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'])))
    # texpanse = cursor.fetchall()
    # print(texpanse)

    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)    texpanse = []

    while dictionary1 != False:
        temp = []
        temp.append(dictionary1["DT"])        temp.append(dictionary1["TOT"])        texpanse.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", texpanse = texpanse, 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'])))
    # texpanse = cursor.fetchall()
    # print(texpanse)
```

```
    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)
    texpanse = []
```

```
    while dictionary1 != False:
        temp = []    temp.append(dictionary1["MN"])    temp.append(dictionary1["TOT"])
texpanse.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-deploymentspec:  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: Servicemetadata:

```
  name: flask-app-servicespec:  selector:
  app: flask-app  ports: - name: http  protocol: TCP
  port: 80  targetPort: 5000  type: LoadBalancermanifest.yaml: 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 sgimport 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", "lxixbmpnxbkiemh")  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("abcd@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

GITHUB LINK:

<https://github.com/IBM-EPBL/IBM-Project-18040-1659678602>

DEMO VIDEO LINK:

https://drive.google.com/drive/folders/1NP5on-kHKpS15ij9cP3Sn_QaZisHGA2k

