**IBM-NALAIYA THIRAN 2022**

**LOYOLA- ICAM COLLEGE OF ENGINEERING AND TECHNOLOGY**

**PERSONAL EXPENSE TRACKER**

**TEAM MEMBERS:**

DELVYN JONES M

RUBERT SHELDON A

SURYANARAYAN R K

SERENA J E

**TEAM ID**: PNT2022TMID27454

**FACULTY MENTOR :** Mary Virgil Nithya S

**INDUSTRY MENTOR  :** Kusboo

**CHAPTER NO** **TITLE** **PAGE NO**

1. **INTRODUCTION**

1.1 Project Overview

1.2 Purpose

1. **LITERATURE SURVEY**

2.1 Existing problem

2.2 References

2.3 Problem Statement Definition

1. **IDEATION & PROPOSED SOLUTION**

3.1 Empathy Map Canvas

* + 1. Ideation & Brainstorming
    2. Proposed Solution
    3. Problem Solution fit
  1. **REQUIREMENT ANALYSIS**
  2. Functional requirement
  3. Non-Functional requirements

1. **PROJECT DESIGN**
   1. Data Flow Diagrams
   2. Solution & Technical Architecture
   3. User Stories
2. **PROJECT PLANNING & SCHEDULING**
   1. Sprint Planning & Estimation
   2. Sprint Delivery Schedule
   3. Reports from JIRA
3. **CODING & SOLUTIONING (Explain the features added in the project along with code)**
   1. Feature 1
   2. Feature 2
   3. Database Schema (if Applicable)
4. **TESTING**
   1. Tests Cases
   2. User Acceptance Testing
5. **RESULTS**
   1. Performance Metrics
6. **ADVANTAGES & DISADVANTAGES**
7. **CONCLUSION**
8. **FUTURE SCOPE**
9. **APPENDIX**

Source Code

GitHub & Project Demo Link

**CHAPTER 1**

**INTRODUCTION**

**1.1 Project Overview:**

f you’ve tried budgeting and failed miserably, using an expense tracker can solve your budget planning problems. Expense tracking isn't just about saving all of your receipts or writing down every cent you spend for the rest of eternity. It’s actually much simpler than that.

Tracking expenses is the difference between creating a budget that works, and one that doesn’t. Whether you’re about to start budgeting for the first time, or have been living on a budget for years, knowing where your money goes is the one thing that is guaranteed to make your budget work.

**1.2 Purpose:**

**1. Tracking Expenses Will Build a Budget That Works**

If your goal is to build a budget spreadsheet that works, you must start by tracking your expenses. You can’t live on a budget where the numbers are pulled out of thin air. You must track where your money is actually going, so that you know where you’re starting from. When tracking your spending, spend as you normally would. Tracking isn’t meant to make you feel guilty or to stop you from spending. It’s meant to highlight what your habits are so that you can make some choices and changes later.

**2. Monitor Your Spending to Make Sure All Monthly Expenses are Covered**

Once you’ve tracked your spending and created a balanced budget, it’s vital to track your spending for the first few months to make sure you’re able to cover all of your monthly and seasonal expenses. You can use tracking as a way to re-assure yourself that your budget plan is actually working.

**3. Even with a Successful Budget, Check-In to Ensure Your Spending Plan is Up to Date** If you’ve been successfully living on a budget, it’s still important to monitor your spending every once in a while to make sure your expenses are still the same as they used to be. When gas prices rise, your fuel expenses will likely go up. Your grocery bill will grow with your growing family. Check-in every now and again to ensure your spending plan is still up to date.

**CHAPTER 2**

**LITERATURE SURVEY**

**2.1 Existing problem:**

There are personal expense tracking sites out in the web. But they charge users after a limit of usage and has more advertisements. Most of them are built on a clean set of features. We have carefully analysed and identified several factors that could be used to improve expense tracking of very individual. These factors fall under the categories of taxes, organizing payments, HTML & JavaScript based features. Using these features, we build an efficient system which can identify all expenditure with high accuracy and efficiency. It is also an open-source website which will be easily accessible to all users.

**2.2 References:**

[1] Kan, C., Lynch, J., & Fernbach, P. (2015). How budgeting helps consumers achieve financial goals. ACR North American Advances.

[2] M N Rajaprabha 2017 IOP Conf. Ser.: Mater. Sci. Eng. 263 042050

 [3] Rajaprabha, M. N. (2017). Family Expense Manager Application in Android. MS&E, 263(4), 042050

[4]. RAJAPRABHA M N,” Family expenses manager” VIT University, Vellore 632014, Tamilnadu, India, 14th icset-2017, IOP publishing

[5]Sabab, S. A., Islam, S. S., Rana, M. J., & Hossain, M. (2018, September). eExpense: A smart approach to track everyday expense.

[6] Sharma, R., 2020. Case Study Of Expense Tracking App: Get Daily Alerts Of Your Expense. [online] Medium

[7] Velmurugan A, Albert mayan J, Niranjana P AND Richard Francis "Expense Manager Application " Journal of physics: Conference series,2020

[8] Atiya Kazi, Praphulla s. Kherade, raj s. Vilankar, Parag m. Sawant, “Expense Tracker”, Iconic Research And Engineering Journals, pp 19-21, may 2021.

**2.3 Problem statement definition:**

At the instant, there is no as such complete solution present easily or we should say free of cost which enables a person to keep a track of its daily expenditure easily. To do so a person has to keep a log in a diary or in a computer, also all the calculations needs to be done by the user which may sometimes results in errors leading to losses.

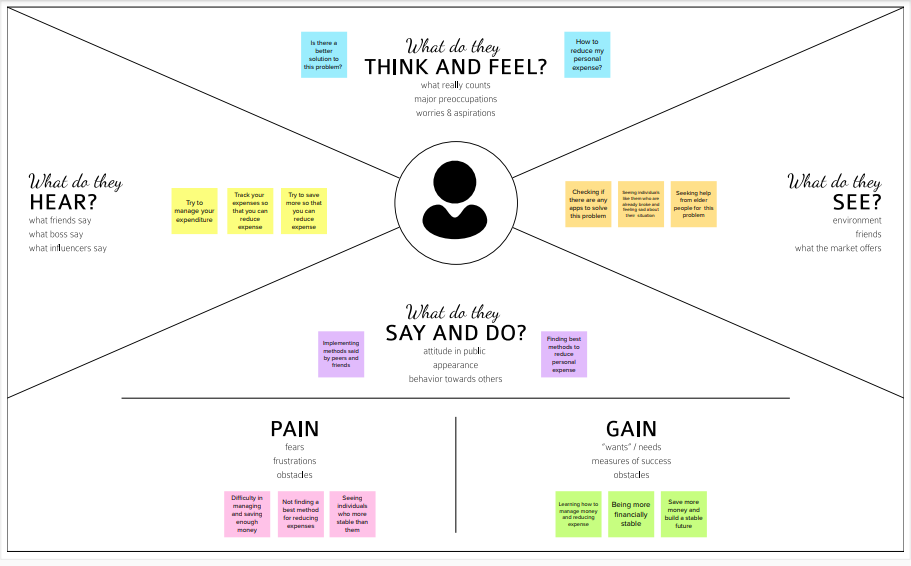
Due to lack of a complete tracking system, there is a constant overload to rely on the daily entry of the expenditure and total estimation till the end of the month. As the name itself suggests, this project is an attempt to manage our daily expenses in a more efficient and manageable way. The system attempts to free the user with as much as possible the burden of manual calculation and to keep the track of the expenditure.

**CHAPTER 3**

**IDEATION & PROPOSED SOLUTION**

**3.1** E**mpathy Map Canvas:**

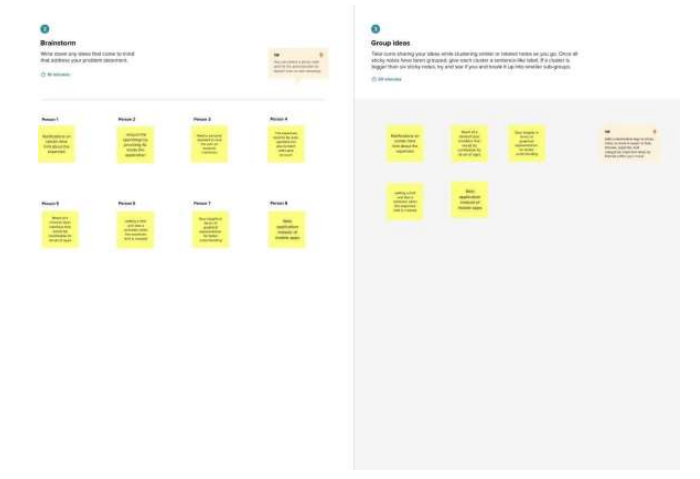
An empathy map is a collaborative tool teams can use to gain a deeper insight into their customers. Much like a user persona, an empathy map can represent a group of users, such as a customer segment. Empathy maps should be used throughout any UX process to establish common ground among team members and to understand and prioritize user needs. In user-centered design, empathy maps are best used from the very beginning of the design process.



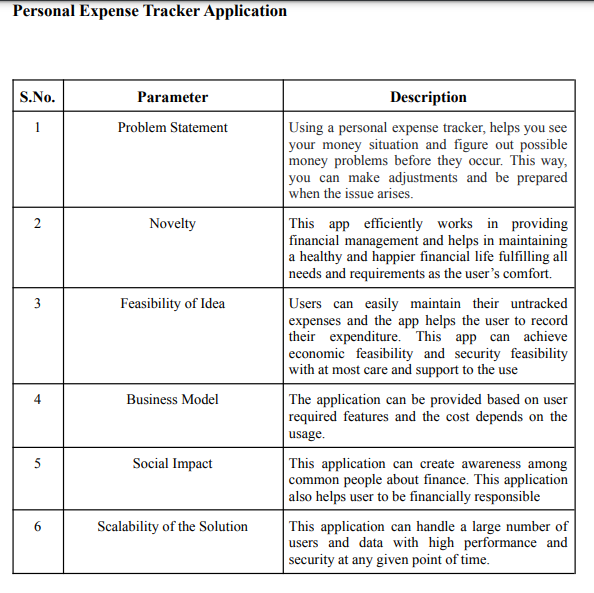
**3.2 Ideation & Brainstorming:**

Ideation essentially refers to the whole creative process of coming up with and communicating new ideas. Ideation is innovative thinking, typically aimed at solving a problem or providing a more efficient means of doing or accomplishing something.

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity.



**3.3 Proposed Solution:**

****

**3.4 Problem Solution fit**

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it solves the customer’s problem. It helps entrepreneurs, marketers and corporate innovators identify behavioural patterns and recognize what would work and why.

Purpose:

❑ Solve complex problems in a way that fits the state of your customers.

❑ Succeed faster and increase your solution adoption by tapping into existing mediums and channels of behaviour.

❑ Sharpen your communication and marketing strategy with the right triggers and messaging.

❑ Increase touchpoints with your company by finding the right problem-behaviour fit and building trust by solving frequent annoyances, or urgent or costly problems.

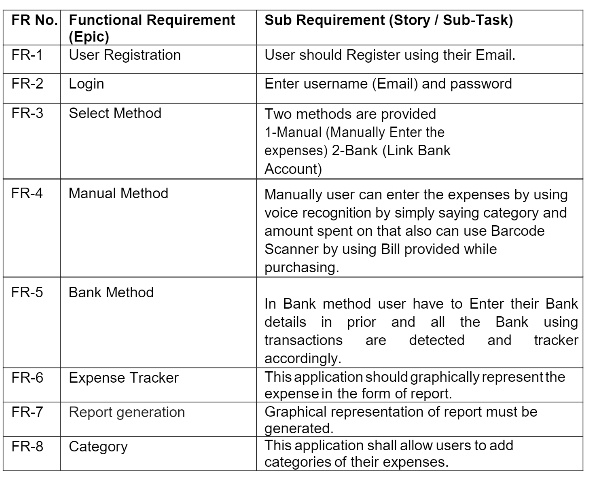
❑ Understand the existing situation in order to improve it for your target group.

****

**CHAPTER 4**

**REQUIREMENT ANALYSIS**

**4.1 Functional requirements:**

**4.2 Non-functional requirements:**

|  |  |  |
| --- | --- | --- |
| **FR No.** | **Non-Functional Requirement** | **Description** |
| NFR-1 | Usability | Helps to keep an accurate record of your income and expenses. |
| NFR-2 | Security | Budget tracking apps are considered very safe from those who commit cyber crimes |
| NFR-3 | Reliability | Each data record is stored on a well built efficient database schema. There is no risk of data loss. |
| NFR-4 | Performance | The types of expense are categories along with an option. Throughput of the system is increased due to light weight database support.. |
| NFR-5 | Availability | The application must have a 100% up-time. |
| NFR-6 | Scalability | The ability to appropriately handle increasing demands. |

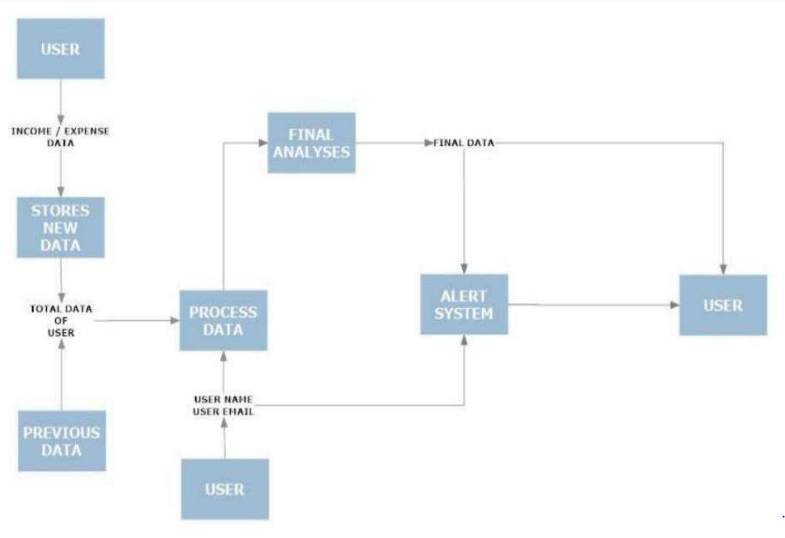
**CHAPTER 5**

**PROJECT DESIGN**

**5.1 Data Flow 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 enters and leaves the system, what changes the information, and where data is stored.

**DFD level 0:**



**5.2 Solution & Technical Architecture:**

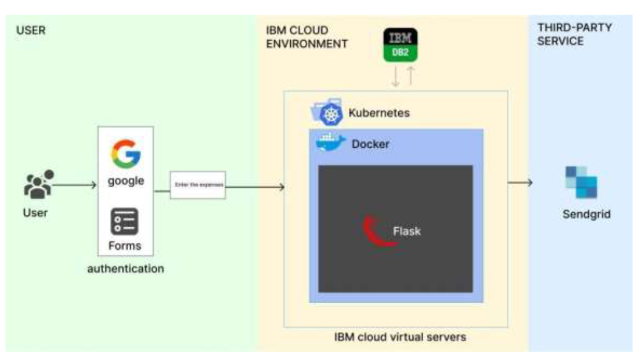
**SOLUTION:**

Our solution is to build an efficient and intelligent system to detect and manage all expenses by applying an user friendly interface which is easy to use by the user. The calculations done in the backend is to calculate all expenses done by the user everyday, he/she can enter his/her daily expenses and can finally have a report of expenses done in that month will display in the website.

By this, the user can keep good track of his/her expenses and if the monthly expense limit set by the user exceeds then a mail will sent to his/her email to alert the user.

**TECHNICAL ARCHITECTURE:**

Technical architecture which is also often referred to as application architecture includes the major components of the system, their relationships, and the contracts that define the interactions between the components. The goal of technical architects is to achieve all the business needs with an application that is optimized for both performance and security.



1. The application developer builds a Python-based app and deploys it.

2. The user enters the date and the expenditure done for the day.

3. The user submits the submits the expenses through the web-based application and it is stored in the history for future updating

4. The user makes a decision whether to re-enter more, view/edit expenses stored in the history and wishes to view the report of the month.

**5.3 User Stories:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **User type** | **Functional requirement**  **(Epic)** | **User Story number** | **User story/task** | **Acceptance criteria** |
| Customer (web user) | Login | USN-1 | As a user, I can navigate into the website. | I can access the page. |
|  | Dashboard | USN-2 | As a user, I will paste the URL that needs to be checked if it’s a phishing website or not. | I can paste the URL in the textbox. |
|  |  | USN-3 | As a user, I can see the output. | I can see if it’s a safe site. |
| Administrator |  | USN-4 | If the new URL is found, I can add into the database.the new state into the database. | I can add the new URL. |

**CHAPTER 6**

**PROJECT PLANNING & SCHEDULING**

**6.1 Sprint Planning & Estimation:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| Sprint-1 | Login | USN-1 | As a user, I can navigate into the website. | 1 | High | [Serena J E](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Serena%20J%20E) |
| Sprint-1 | Dashboard | USN-2 | As a user, I will set the limit for the expense. | 1 | High | [Serena J E](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Serena%20J%20E) |
| Sprint-1 |  | USN-3 | As a user, I will input the expense . | 2 | High | Suryanarayan |
| Sprint-2 | Backend | USN-4 | As an admin, I will connect IBM cloud with the database. | 3 | Medium | [Rubert Sheldon](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Rubert%20Sheldon)  Suryanarayan |
| Sprint-3 | Report | USN-5 | As a admin, The expense will be tracked continuously and compared with the limit that has been set. | 1 | Low | [Rubert Sheldon](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Rubert%20Sheldon)  [Delvyn Jones M](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Delvyn%20Jones%20M) |
| Sprint-4 |  | USN-6 | As a admin,If expense is exceed the limit a mail will be sent to the user | 2 | Low | [Delvyn Jones M](https://github.com/IBM-EPBL/IBM-Project-54920-1663089936/tree/main/Assignment%201/Delvyn%20Jones%20M) |

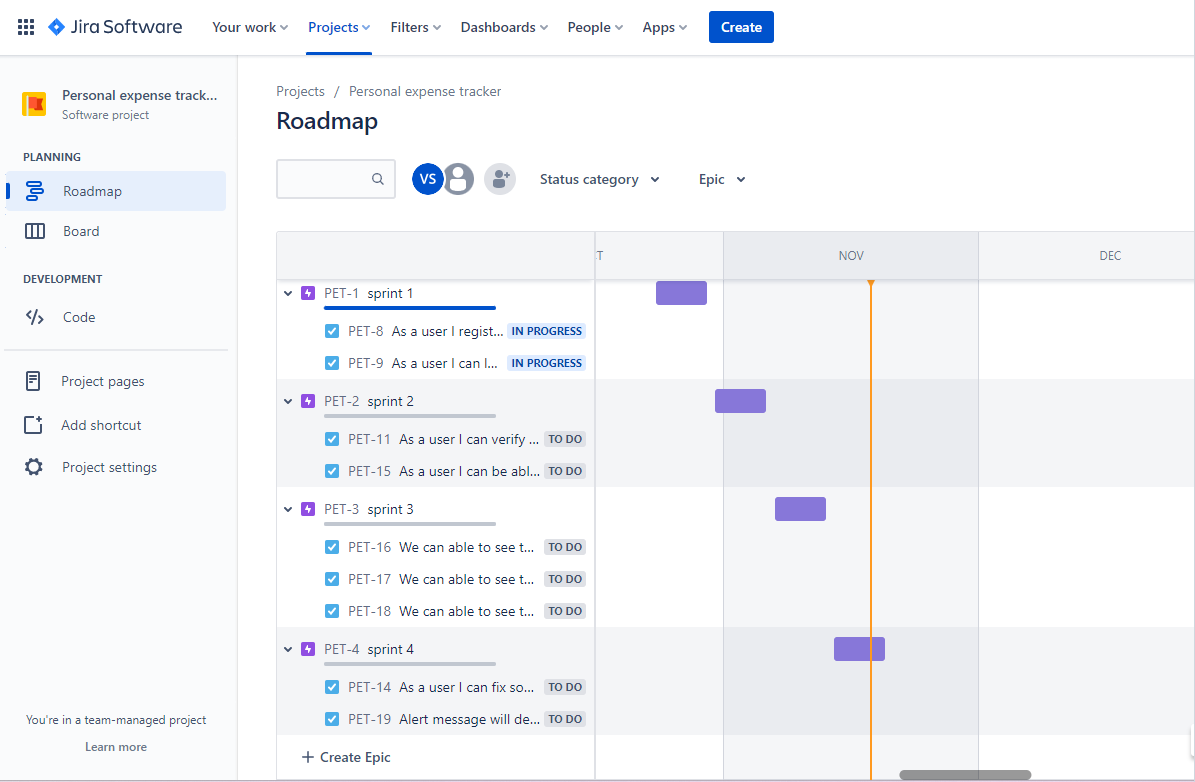
**6.2 Sprint Delivery Schedule:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 | 6 Days | 14 Nov 2022 | 19 Nov 2022 | 20 | 19 Nov 2022 |

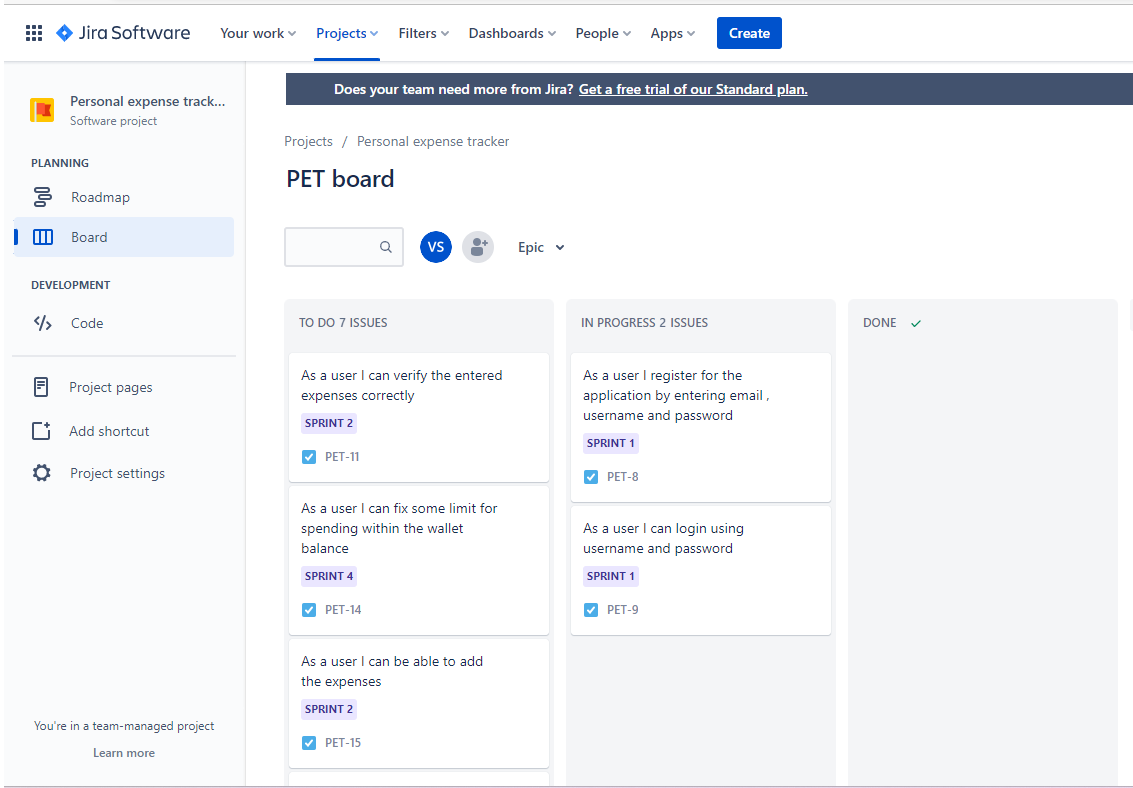
**6.3 Reports from JIRA:**

**Backlog:**

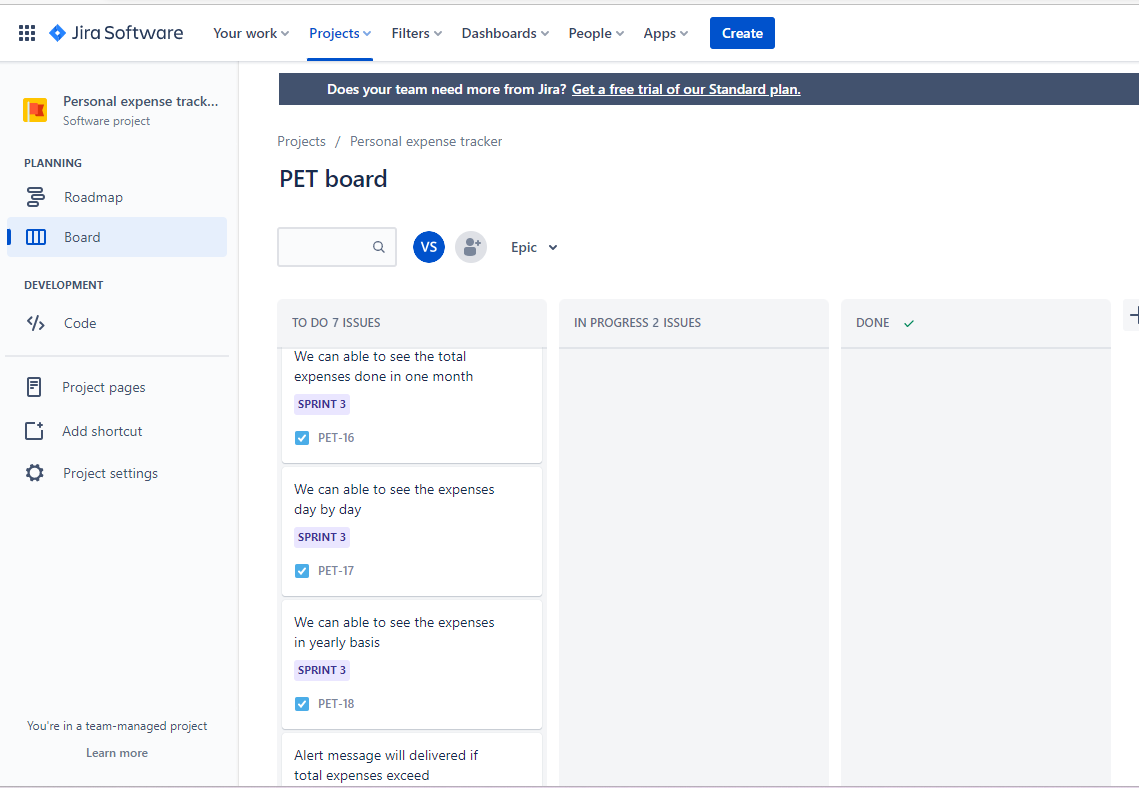
**Sprint 1:**



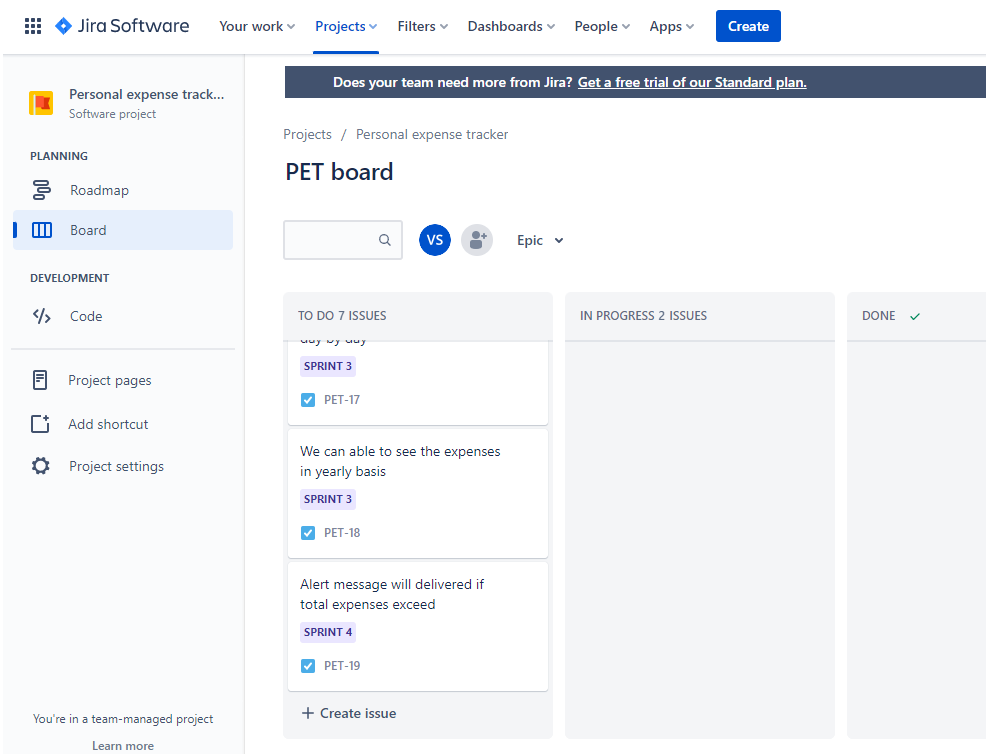
**Insights:**



**Sprint and Sprint 3:**

****

**Sprint 4:**

****

**CHAPTER 7**

**CODING & SOLUTIONING**

**7.1 Feature 1 – Setting Expense Limit:**

Limit option in our website leads the user to the limit setting page the user set the limit for his/her expenses. For instances, if a user wants to the limit for expenses as Rs.10000 for a month he can do it and if that limit is crossed, then the user will be receiving an alert mail that he/she has exceeded the limit spend. This makes sure that the user stays right in the limit of expenditure and not crossing it far more.

**Code:**

# check wether the overall expenses is higher than the limit

sql = "SELECT expenses.amount FROM expenses WHERE userid=? and MONTH(CURRENT DATE) - MONTH(expenses.date) = 0"

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

ibm\_db.bind\_param(stmt, 1, session['id'])

ibm\_db.execute(stmt)

itr = ibm\_db.fetch\_assoc(stmt)

amount = 0

while itr != False:

amount += itr["AMOUNT"]

itr = ibm\_db.fetch\_assoc(stmt)

print("Details")

print(amount)

sql = "SELECT limitss FROM limits ORDER BY limits.id DESC LIMIT 1"

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

ibm\_db.execute(stmt)

s = ibm\_db.fetch\_tuple(stmt)

if s and amount > s[0]:

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

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

ibm\_db.bind\_param(stmt, 1, session['id'])

ibm\_db.execute(stmt)

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

sendmail(account["EMAIL"], "Limits Exceeded", str(s))

return redirect("/display")

**2 Feature 2 – Report:**

Report page of our site allows users to backtrack his/her expenses done for the month and the users can review their previous month expenses done in order. Here the report page is important the user needs to know how much he/she has spent and what should be done to reduce those expenses in the coming months.

**Code**:

# REPORT

@app.route("/month")

def month():

sql = "SELECT expenses.amount, expenses.paymode, expenses.category FROM expenses WHERE userid=? and MONTH(CURRENT DATE) - MONTH(expenses.date) = 0"

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

ibm\_db.bind\_param(stmt, 1, str(session['id']))

ibm\_db.execute(stmt)

tuple\_t = ibm\_db.fetch\_tuple(stmt)

total\_amount = 0

food = 0

entertainment = 0

Business = 0

Rent = 0

EMI = 0

Other = 0

while tuple\_t != False:

tuple\_l = list(tuple\_t)

total\_amount += tuple\_l[0]

if tuple\_l[2] == "entertainment":

entertainment += tuple\_l[0]

elif tuple\_l[2] == "business":

Business += tuple\_l[0]

elif tuple\_l[2] == "rent":

Rent += tuple\_l[0]

elif tuple\_l[2] == "food":

food += tuple\_l[0]

elif tuple\_l[2] == "EMI":

EMI += tuple\_l[0]

else:

Other += tuple\_l[0]

tuple\_t = ibm\_db.fetch\_tuple(stmt)

return render\_template("month.html", total\_amount=total\_amount, food=food, entertainment=entertainment, Business=Business, Rent=Rent,EMI=EMI, Other=Other)

@app.route('/logout')

def logout():

session.pop('loggedin', None)

session.pop('id', None)

session.pop('username', None)

return render\_template('home.html')

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

app.run(port=5000,debug=True)

**CHAPTER 8**

**TESTING**

**8.1 Test Cases:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case ID** | **Feature Type** | **Component** | **Test Scenario** | **Steps To Execute** | **Expected Result** | **Actual Result** | **Status** |
| DashBoard\_TC\_OO1 | Functional | Home Page | Verify user is able to enter the data correctly | 1.Open Kosten website 2.Enter an expense and click add | It will be added correctly and history is recorded | Working as expected | Pass |
| DashBoard\_TC\_OO2 | UI | Home Page | Verify the options in the home page | 1.Enter expense and go to history 2.The history has all the expenses entered which can be edited 3.Edit and update contents. Delete contents | Application should show below UI elements: a. Delete/update  b. add c. report  d. history | Working as expected | Pass |
| DashBoard\_TC\_OO3 | Functional | Home page | Verify user is able to see an alert mail when expense crosses the limit | 1.Enter expenses and add 2.set a limit and submit 3.An alert mail is sent if the expense crosses the limit | Alert of crossed expense limit | Working as expected | Pass |
| DashBoard\_TC\_OO4 | Functional | Home page | Verify user is able to see the result updated | 1.Enter expense and go to history 2.The history has all the expenses entered which can be edited 3.Edit and update contents. Delete contents | Result of updated contents will be displayed | Working as expected | Pass |
| Report\_TC\_OO1 | Functional | Report page | Verify user is able see the expenses of the month in a pie chart like structure | 1.Enter the expenses and click on add 2.check the report if it is updated  3.Log out if needed | Details are stored in the database | Working as expected | Pass |

**8.2 User Acceptance Testing:**

# **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 | 3 | 20 |
| Duplicate | 1 | 0 | 3 | 0 | 4 |
| External | 2 | 3 | 0 | 1 | 6 |
| Fixed | 11 | 2 | 4 | 20 | 37 |
| Not Reproduced | 0 | 0 | 1 | 0 | 1 |
| Skipped | 0 | 0 | 1 | 1 | 2 |
| Won't Fix | 0 | 5 | 2 | 1 | 8 |
| Totals | 24 | 14 | 13 | 26 | 77 |

**Test Case Analysis:**

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Section** | **Total Cases** | **Not Tested** | **Fail** | **Pass** |
| Client Application | 51 | 0 | 0 | 51 |
| Notification | 2 | 0 | 0 | 2 |
| Outsource Shipping | 3 | 0 | 0 | 3 |
| Exception Reporting | 9 | 0 | 0 | 9 |
| Final Report Output | 4 | 0 | 0 | 4 |
| Version Control | 2 | 0 | 0 | 2 |

**CHAPTER 9**

**RESULTS**

**9.1 Performance metrics:**

As a website owner - whether you want to deliver an incredible experience on your website, or increase conversion rates - it's critical to understand how your end users are truly experiencing your site.

Measuring your website's most important metrics is particularly important as you make changes to your website over time and may introduce issues without realising it. Nothing frustrates visitors more than a slow website, a complicated UI, or unfamiliar navigation patterns. That's why it's mission critical to continuously monitor for user experience and performance issues.

Add a tracking code to your website

Code collects data about visits to your website

Analytics software summarizes the data for you

**CHAPTER 10**

**ADVANTAGES & DISADVANTAGES**

**ADVANTAGES:**

* It Helps You Stick to Your Budget.
* Tracking Your Expenses Can Reveal Spending Issues.
* It Helps You Meet Your Financial Objectives.
* Record Expenses with Pen and Paper.
* Make It Easier With an Software.

**DISADVANTAGES:**

* **Negligence during auditing:**

Although software simplifies the work of auditors, there is less scope for auditors to investigate suspicious activities. External verification can become minimal or redundant since all expense reports are already internally verified by the software. Due to this, auditors may choose to trust the software’s discretion and neglect uncertainties.

**CHAPTER 11**

**CONCLUSION**

Monitoring your everyday expenses can set aside you cash, yet it can likewise help you set your monetary objectives for what’s to come. On the off chance that you know precisely where your sum is going much of a stretch see where a few reductions and bargains can be made. Expense Tracker project is for keeping our day-to-day expenditures will helps us to keep record of our money daily. The project what we have created is work more proficient than the other income and expense tracker. The project effectively keeps away from the manual figuring for trying not to ascertain the pay and cost each month. It’s a user-friendly application.

**CHAPTER-12**

**FUTURE SCOPE**

1) It will have various options to keep record (for example Food, Travelling Fuel, Salary etc.).

2) Automatically it will keep on sending notifications for our daily expenditure.

3) In today’s busy and expensive life, we are in a great rush to make moneys, but at the end of the month we broke off. As we are unknowingly spending money on title and unwanted things. So, we have come over with the plan to follow our profit.

4) Here user can define their own categories for expense type like food, clothing, rent and bills where they have to enter the money that has been spend and likewise can add some data in extra data to indicate the expense

**CHAPTER 13**

**APPENDIX**

**13.1 Source code:**

**app.py**

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

"""

Spyder Editor

This is a temporary script file.

"""

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

import re

import ibm\_db

from sendemail import sendmail

# from sqlalchemy import create\_engine

# from sqlalchemy.orm import sessionmaker

# import sqlalchemy as db

# from sqlalchemy.ext.declarative import declarative\_base

# Base = declarative\_base()

#

# engine = create\_engine("db2+ibm\_db://nby71676:x1pmgkWtUJLmXz0n@ea286ace-86c7-4d5b-8580-3fbfa46b1c66.bs2io90l08kqb1od8lcg.databases.appdomain.cloud:31505/bludb")

# connection = engine.connect()

# metadata = db.MetaData()

# census = db.Table('expenses', metadata, autoload=True, autoload\_with=engine)

#

# DEFINE THE ENGINE (CONNECTION OBJECT)

# CREATE THE TABLE MODEL TO USE IT FOR QUERYING

# class Expenses(Base):

#

    # \_\_tablename\_\_ = 'expenses'

#

    # id = db.Column(db.Numeric,

                        #    primary\_key=True)

    # userid = db.Column(db.Numeric)

    # date = db.Column(db.DATE)

    # expensename = db.Column(db.String(100))

    # amount = db.Column(db.Float)

    # paymode = db.Column(db.String(100))

    # category = db.Column(db.String(100))

#

# CREATE A SESSION OBJECT TO INITIATE QUERY

# IN DATABAS

dsn\_hostname = "ea286ace-86c7-4d5b-8580-3fbfa46b1c66.bs2io90l08kqb1od8lcg.databases.appdomain.cloud"

dsn\_uid = "nby71676"

dsn\_pwd = "5Y2wjQ1D0vOzuzMN"

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

dsn\_database = "bludb"

dsn\_port = "31505"

dsn\_protocol = "TCPIP"

dsn = (

    "DRIVER={0};"

    "DATABASE={1};"

    "HOSTNAME={2};"

    "PORT={3};"

    "PROTOCOL={4};"

    "UID={5};"

    "PWD={6};"

    "SECURITY=SSL;"

    "SSLServerCertificate=DigiCertGlobalRootCA.crt;").format(dsn\_driver, dsn\_database, dsn\_hostname, dsn\_port, dsn\_protocol, dsn\_uid, dsn\_pwd)

try:

    conn = ibm\_db.connect(dsn, "", "")

    print("connected to database: ", dsn\_database,

          "as\_user: ", "on\_host: ", dsn\_hostname)

except:

    print("Unable to connect: ", ibm\_db.conn\_errormsg())

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

app.secret\_key = 'a'

# app.config['MYSQL\_HOST'] = 'localhost'

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

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

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

mysql = ""

# 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 = ''

    if request.method == 'POST':

        username = request.form['username']

        email = request.form['email']

        password = request.form['password']

        # cursor = mysql.connection.cursor()

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

        # account = cursor.fetchone()

#

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

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

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

        ibm\_db.execute(stmt)

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

        print(account)

        if account:

            msg = 'Account 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:

            insert\_sql = "INSERT INTO register(username, email, password) VALUES (?, ?, ?)"

            prep\_stmt = ibm\_db.prepare(conn, insert\_sql)

            ibm\_db.bind\_param(prep\_stmt, 1, username)

            ibm\_db.bind\_param(prep\_stmt, 2, email)

            ibm\_db.bind\_param(prep\_stmt, 3, password)

            ibm\_db.execute(prep\_stmt)

            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()

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

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

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

        ibm\_db.bind\_param(stmt, 2, password)

        ibm\_db.execute(stmt)

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

        print(account)

        if account:

            session['loggedin'] = True

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

            userid = account["ID"]

            session['username'] = account['USERNAME']

            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)

    print(type(date))

    # 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()

    insert\_sql = "INSERT INTO expenses (userid, date, expensename, amount, paymode, category) VALUES (?, ?, ?, ?, ?, ?)"

    prep\_stmt = ibm\_db.prepare(conn, insert\_sql)

    ibm\_db.bind\_param(prep\_stmt, 1, session['id'])

    ibm\_db.bind\_param(prep\_stmt, 2, date)

    ibm\_db.bind\_param(prep\_stmt, 3, expensename)

    ibm\_db.bind\_param(prep\_stmt, 4, amount)

    ibm\_db.bind\_param(prep\_stmt, 5, paymode)

    ibm\_db.bind\_param(prep\_stmt, 6, category)

    ibm\_db.execute(prep\_stmt)

    msg = 'You have successfully registered !'

    print(date + " " + expensename + " " +

          amount + " " + paymode + " " + category)

    # check wether the overall expenses is higher than the limit

    sql = "SELECT expenses.amount FROM expenses WHERE userid=? and MONTH(CURRENT DATE) - MONTH(expenses.date) = 0"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.bind\_param(stmt, 1, session['id'])

    ibm\_db.execute(stmt)

    itr = ibm\_db.fetch\_assoc(stmt)

    amount = 0

    while itr != False:

        amount += itr["AMOUNT"]

        itr = ibm\_db.fetch\_assoc(stmt)

    print("Details")

    print(amount)

    sql = "SELECT limitss FROM limits ORDER BY limits.id DESC LIMIT 1"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.execute(stmt)

    s = ibm\_db.fetch\_tuple(stmt)

    if s and amount > s[0]:

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

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

        ibm\_db.bind\_param(stmt, 1, session['id'])

        ibm\_db.execute(stmt)

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

        sendmail(account["EMAIL"], "Limits Exceeded", str(s))

    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()

    # Session = sessionmaker(bind=engine)

    # dbsession = Session()

    # expense = dbsession.query(Expenses).where(userid == session['id']).order\_by(Expenses.date.desc())

    # expense = e.select([expenses]).where(userid == session['id']).order\_by(expenses.date.desc())

    #Equivalent to 'SELECT \* FROM census'

    sql = "SELECT \* FROM expenses WHERE userid=? ORDER By expenses.date DESC"

    stmt = ibm\_db.prepare(conn, sql)

    # print(str(session['id']))

    ibm\_db.bind\_param(stmt, 1, str(session['id']))

    ibm\_db.execute(stmt)

    tuple\_t = ibm\_db.fetch\_tuple(stmt)

    expense = []

    while tuple\_t != False:

        expense.append(list(tuple\_t))

        tuple\_t = ibm\_db.fetch\_tuple(stmt)

    print(expense)

    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()

    sql = "DELETE FROM expenses WHERE id=?"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.bind\_param(stmt, 1, id)

    ibm\_db.execute(stmt)

    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()

    sql = "SELECT \* FROM expenses WHERE id=?"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.bind\_param(stmt, 1, id)

    ibm\_db.execute(stmt)

    tuple\_t = ibm\_db.fetch\_tuple(stmt)

    # print()

    print(list(tuple\_t))

    tuple\_l = list(tuple\_t)

    tuple\_l[2] = tuple\_l[2].strftime("%Y-%m-%d")

    print(tuple\_l)

    return render\_template('edit.html', expenses=tuple\_l)

@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))

        sql = "UPDATE expenses SET date = ?, expensename = ?, amount = ?, paymode = ?, category = ? WHERE expenses.id = ?"

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

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

        ibm\_db.bind\_param(stmt, 2, expensename)

        ibm\_db.bind\_param(stmt, 3, amount)

        ibm\_db.bind\_param(stmt, 4, str(paymode))

        ibm\_db.bind\_param(stmt, 5, str(category))

        ibm\_db.bind\_param(stmt, 6, id)

        ibm\_db.execute(stmt)

        # mysql.connection.commit()

        print('successfully updated')

        # check wether the overall expenses is higher than the limit

        sql = "SELECT expenses.amount FROM expenses WHERE userid=? and MONTH(CURRENT DATE) - MONTH(expenses.date) = 0"

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

        ibm\_db.bind\_param(stmt, 1, session['id'])

        ibm\_db.execute(stmt)

        itr = ibm\_db.fetch\_assoc(stmt)

        amount = 0

        while itr != False:

            amount += itr["AMOUNT"]

            itr = ibm\_db.fetch\_assoc(stmt)

        print("Details")

        print(amount)

        sql = "SELECT limitss FROM limits ORDER BY limits.id DESC LIMIT 1"

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

        ibm\_db.execute(stmt)

        s = ibm\_db.fetch\_tuple(stmt)

        if s and amount > s[0]:

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

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

            ibm\_db.bind\_param(stmt, 1, session['id'])

            ibm\_db.execute(stmt)

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

            sendmail(account["EMAIL"], "Limits Exceeded", str(s))

        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(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()

    sql = "SELECT limitss FROM limits WHERE userid = ? ORDER BY limits.id DESC LIMIT 1"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.bind\_param(stmt, 1, session['id'])

    ibm\_db.execute(stmt)

    s = ibm\_db.fetch\_tuple(stmt)

    if s:

        s = s[0]

    else:

        s = 0

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

# REPORT

@app.route("/month")

def month():

    # cursor = mysql.connection.cursor()

    # cursor.execute(

        # 'SELECT DATE(date), SUM(amount) FROM expenses WHERE userid= %s AND MONTH(DATE(date))= MONTH(now()) GROUP BY DATE(date) ORDER BY DATE(date) ', (str(session['id'])))

    # texpense = cursor.fetchall()

    # print(texpense)

#

    # 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()

    sql = "SELECT expenses.amount, expenses.paymode, expenses.category FROM expenses WHERE userid=? and MONTH(CURRENT DATE) - MONTH(expenses.date) = 0"

    stmt = ibm\_db.prepare(conn, sql)

    ibm\_db.bind\_param(stmt, 1, str(session['id']))

    ibm\_db.execute(stmt)

    tuple\_t = ibm\_db.fetch\_tuple(stmt)

    # texpense = []

    total\_amount = 0

    food = 0

    entertainment = 0

    Business = 0

    Rent = 0

    EMI = 0

    Other = 0

    while tuple\_t != False:

        tuple\_l = list(tuple\_t)

        total\_amount += tuple\_l[0]

        if tuple\_l[2] == "entertainment":

            entertainment += tuple\_l[0]

        elif tuple\_l[2] == "business":

            Business += tuple\_l[0]

        elif tuple\_l[2] == "rent":

            Rent += tuple\_l[0]

        elif tuple\_l[2] == "food":

            food += tuple\_l[0]

        elif tuple\_l[2] == "EMI":

            EMI += tuple\_l[0]

        else:

            Other += tuple\_l[0]

        # texpense.append()

        tuple\_t = ibm\_db.fetch\_tuple(stmt)

    # print(texpense)

    return render\_template("month.html", total\_amount=total\_amount, food=food, entertainment=entertainment, Business=Business, Rent=Rent,EMI=EMI, Other=Other)

# log-out

@app.route('/logout')

def logout():

    session.pop('loggedin', None)

    session.pop('id', None)

    session.pop('username', None)

    return render\_template('home.html')

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

    app.run(port=5000,debug=True)

**13.2 GitHub & project demo link:**

GitHub link: <https://github.com/IBM-EPBL/IBM-Project-54920-1663089936>

Demo link: <https://drive.google.com/file/d/1f9XRzs5MDw4IEpbT14BI2phs4ixFnsVi/view?usp=share_link>