You Choose, We Do it

St. JOSEPH'S COLLEGE OF ENGINEERING

(An Autonomous Institution)

St. Joseph's Group of Institutions

Jeppiaar Educational Trust

OMR, Chennai-119

Team ID	PNT2022TMID00043
Project Name	Project - Personal Expense Tracker

	NAME	REGISTER NO
TEAM LEADER	AVINASH.S	312319104020
TEAM MEMBER 1	GIRIDHARAN M S	312319104034
TEAM MEMBER 2	ADITHYA ANIL	312319104005
TEAM MEMBER 3	KARTHICK S	312319104063

INDEX

1. INTRODUCTION

1. Project Overview

Personal Expense Tracker is a web application that allows you to track the daily expense of the user and help them to keep track of their expenses daily, monthly, weekly and yearly basis. It will alsocreate a digitalrecords for the user's income and various expenses spent by the user is calculated. It also gets the input from the user as how much income earned and the date of the income earned and further creates a transaction entry and sums up all the total income. Further we can give voice commands and it is responsive. It will be very helpful for the users to manage their needs and they can spend in a better way by keeping track of the application daily basis.

2. Purpose

The main purpose of personal expense tracker application is used to keep track of expenses based on the user income and how much they spent and they can keep track of their expenses daily, monthly, weekly and yearly basis.

2. LITERATURE SURVEY

1. Existing problem

The problem of current generation population is that they can't remember where all of the money they earned have gone and ultimately have to live while sustaining the little money they have left for their essential needs. In this time there is no such perfect solution which helps a person to track their daily expenditure easily and efficiently and notify them about the money shortage they have. For doing so have to maintain long ledgers or computer logs to maintain such data and the calculation is done manually by the user, which may generate error leading to losses. Not having a complete tracking.

2. References

• https://nevonprojects.com/daily-expense-tracker-system/

• https://data-flair.training/blogs/expense-tracker-python/

• https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/

• https://ijarsct.co.in/Paper391.pdf

3. Problem Statement Definition

This Expense Tracker is a web application that facilitates the users to keep track and manage their personal as well as business expenses. This application helps the users to keep a digital diary. It will keep track of a user's income and expenses on a daily basis. The user will be able to add his/her expenditures instantly and can review them anywhere and anytime with the help of the internet. He/she can easily import transactions from his/her mobile wallets without risking his/her information and efficiently protecting his/her privacy. This expense tracker provides a complete digital solution to this problem. Excel sheets do very little to help in tracking Furthermore, they don't have the advanced functionality of preparing graphical visuals automatically. Not only it will save the time of the people but also it will assure error free calculations. The user just has to enter the income and expenditures and everything else will be performed by the system. Keywords: Expense Tracker, budget, planning, savings, graphical visualization of expenditure.

3. IDEATION & PROPOSED SOLUTION

1. Empathy Map Canvas

Edit this template
Right-click to unlock

Empathy Map Canvas

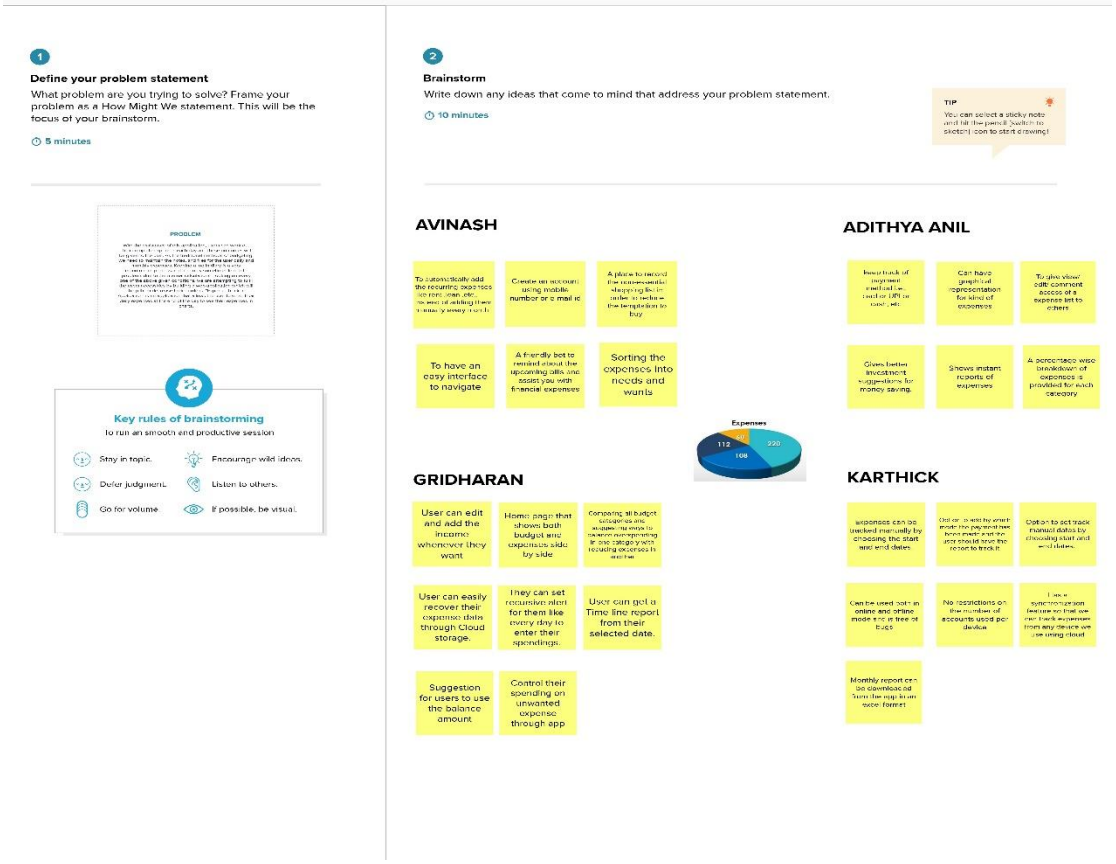
Gain insight and understanding on solving customer problems.

1

Build empathy and keep your focus on the user by putting yourself in their shoes.

Share your feedback

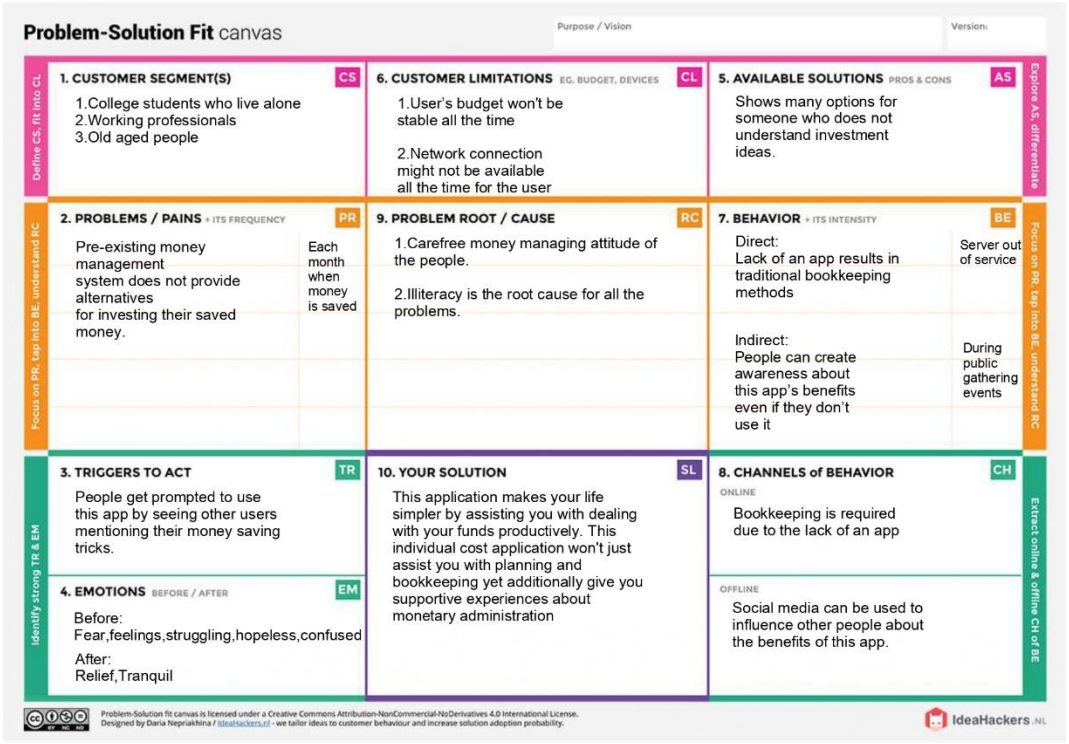
2. Ideation & Brainstorming



3. Proposed Solution

S.No.	Parameter	Description
1.	PROBLEMSTATEMENT	Youth nowadays handle cash without any knowledge of how to deal with it, and when it comes to spending their own cash, it becomes difficult to meet their financial obligations.
2.	IDEA/SOLUTIONDESCRIPTION	This application makes your life simpler by assisting you with dealing with your funds productively. This individual cost application won't just assist you with planning and bookkeeping yet additionally give you supportive experiences about monetary administration.
3.	NOVELTY	•Providing alerts on over expenditure on certain categories. • How to use the saved money. • Instead of manually adding recurring expenses, they are automatically added. • More than 1 profiles can be handled in a single account. • Multiple mode of expenses made can be given as input.
4.	SOCIALIMPACT	• It is possible for teens to learn how to spend their money wisely if they use this properly. • Mostly used by middle class people. • Controlling unnecessary expenses for non-essential items is possible.
5.	BUSINESSMODEL	• We can provide pop-up ads, overlay ads, and other advertising services from third party advertisers. • An account can hold multiple profiles at different subscription levels. • Alerts can be directed to ad promoted suggestions.
6.	FEASIBILITY OF IDEA	• It is compatible with all browsers. • It is economically safe as it doesn't handle with transactions. • We can use this app without fear of scams.
7.	SCALABILITY OF SOLUTION	• Enhancements to the app through expert customer service. • Bill amounts can be scanned for expenditures. • Direct retrievals of expenses made through cards and UPI's.

4. Problem Solution fit



4. REQUIREMENT ANALYSIS

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement	Sub Requirement
FR-1	User Registration	Registration via Application Registration via E-mail
FR-2	User Confirmation	Confirmation via Email
FR-3	User monthly expense tentative data	Data to be registered in the app
FR-4	User monthly income data	Data to be registered in the app
FR-5	Alert / Notification	Alert via E-mail
FR-6	User Budget Plan	Planning and Tracking of user expenses vs. budget limit

Non-functional Requirements:

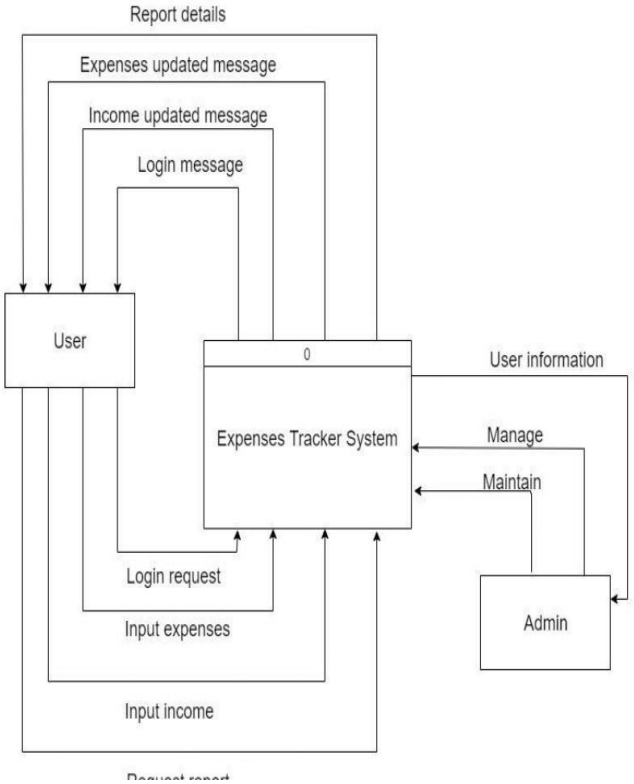
Following are the non-functional requirements of the proposed solution.

NFR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Effectiveness, efficiency, and overall satisfaction of the user while interacting with the application
NFR-2	**Security**	Authentication and encryption of the application
NFR-3	**Reliability**	Probability of failure-free operations in a specified environment for a specified time
NFR-4	**Performance**	How the application is functioning and how responsive the application is to the end-users
NFR-5	**Availability**	In spite of the lack of an active internet connection all features of the application are accessible. Synchronization of data cannot be done
NFR-6	**Scalability**	The capacity of the application to handle growth, especially in handling more users.

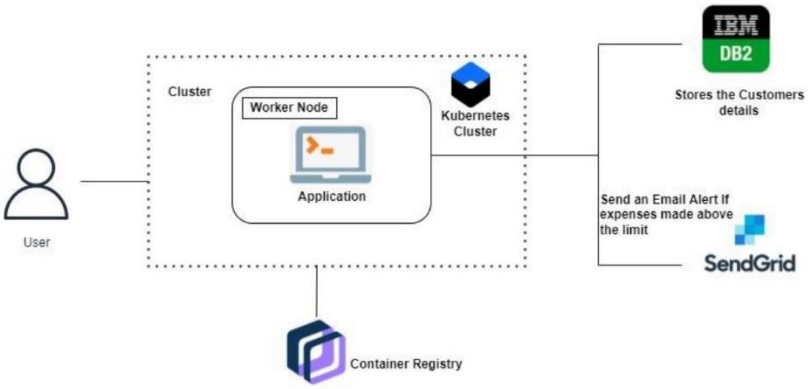
5. PROJECT DESIGN

1. Data Flow Diagrams

LEVEL 0: DIAGRAM OF THE SYSTEM



2. Solution & Technical Architecture



3. User Stories

User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	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 the application	Medium	Sprint-1
	Login	USN-4	As a user, I can log into the application by entering email & password	I can enter into the application by using the registered email and password	High	Sprint-1
	Dashboard	USN-5	As a user, I can administrator I can enter into the dashboardto add my income and expenditures	I can view my daily, monthly and yearly expenses	High	Sprint-1
Customer Care Executive	Alerts and Messages	USN-6	As a Customer Care Executive, I can send Alerts and messages to the user	I can send alerts when the user exceeds the expense limit	High	Sprint-1
Administrator	Application	USN-7	As an upgrade orupdate the application.	The bug can be fixed for the application's customers and users	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

1. Sprint Planning & Estimation

PRODUCT BACKLOG, SPRINT SCHEDULE, AND ESTIMATION (4 MARKS)

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration (Mobile User)	USN-1	In order to register for the application, I must first enter my email address and then confirm my password.	4	High	Avinash S
Sprint-1	Registration	USN-2	The application will send me a confirmation email once User have registered as a user	3	High	Giridharan M S
Sprint-1	Registration	USN-3	Through my Gmail account, I can register for the application	2	Medium	AdithyaAnil
Sprint-1	Login	USN-4	The application allows me to log in by entering my email address and password	3	High	Karthick S
Sprint-2	Registration (WebUser)	USN-5	It is possible for me to register for the application by entering my email address, password, and confirming my password	4	High	Avinash S Giridharan M S
Sprint-2	Registration	USN-6	Once I have registered for the application, User will receive a confirmation email	3	High	Giridharan M S
Sprint-2	Registration	USN-7	User can able to register for the application using my Gmail account	2	Medium	AdithyaAnil Karthick S
Sprint-2	Login	USN-8	The application allows User to log in by entering my email address and password	3	High	Avinash S
Sprint-3	Expense Update	USN-9	As a user, I can add my wallet balance and add or delete my expenses.	6	High	Avinash S

2. Sprint Delivery Schedule

7. CODING & SOLUTIONING

2. Sprint Delivery Schedule

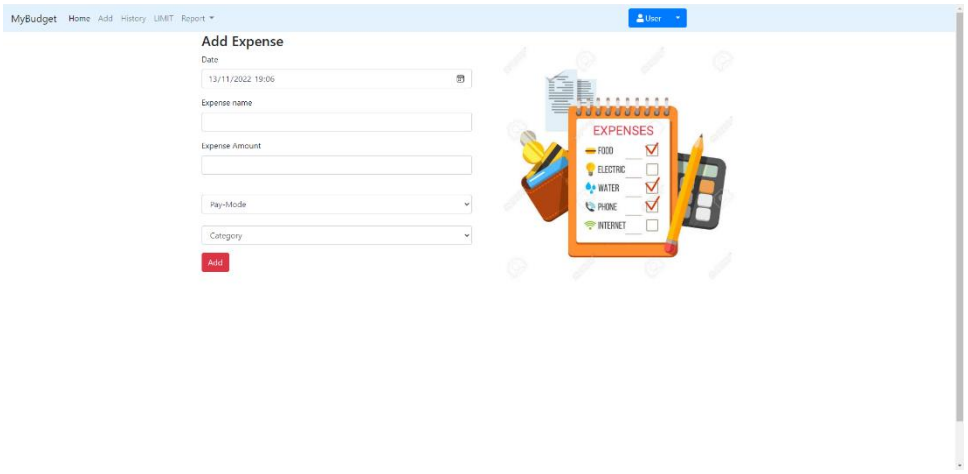
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-3	Expense Update	USN-10	As a user, I have the ability to update my expenses on a regular basis.	6	Medium	Avinash S Giridharan M S
Sprint-4	Email alert	USN-11	As a user, I can set an expense limit out of the Wallet balance.	6	High	AdithyaAnil
Sprint-4	Email alert	USN-12	As a user, I will be notified by e-mail if I have reached the set limit.	6	High	Karthick S

PROJECT TRACKER, VELOCITY & BURNDOWN CHART: (4 MARKS)

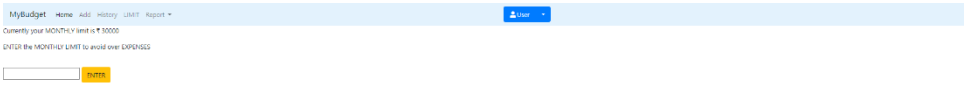
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date(Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date(Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022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

7. CODING & SOLUTIONING

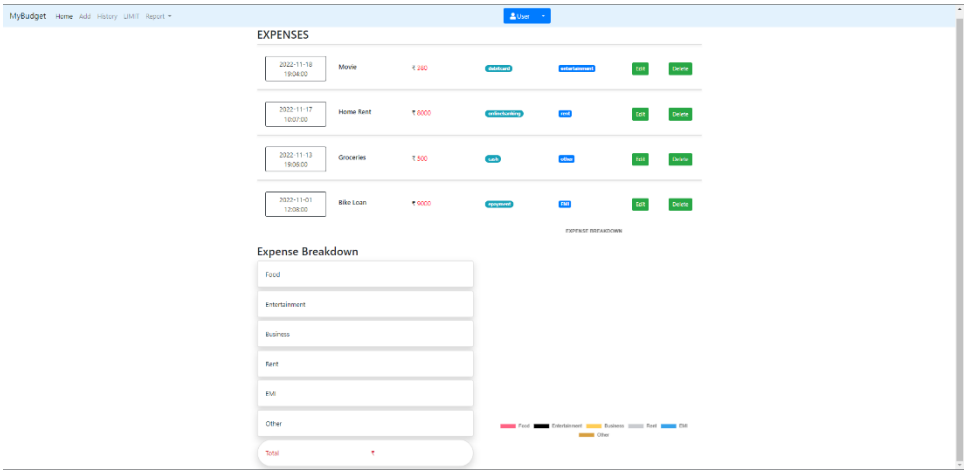
1. Add Expense



2. Add Income



3. Change Budget



8. TESTING

1. Test Cases

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Sub Total
By Design	5	2	1	2	10
Duplicate	1	2	0	0	3
External	3	1	0	0	4
Fixed	5	0	1	0	6
Not Reproduced	0	3	0	1	4
Skipped	1	2	1	1	5
Won't Fix	0	1	0	2	3
Totals	15	11	3	6	35

2. User Acceptance Testing

Section	Test Cases	Not Tested	Fail	Pass
Print Engine	6	0	0	6
Client Application	36	0	0	36
Security	7	0	0	7
Outsource Shipping	10	0	0	10
Exception Reporting	2	0	0	2
Final Report Output	16	0	0	16
Version Control	5	0	0	5

9. RESULTS

1. Performance Metrics

- Tracking income and expenses: Observing the pay and following all consumptions (through ledgers, versatile wallets, and credit and check cards)
- Exchange Receipts: Catch and sort out your instalment receipts to monitor your consumption.
- Sorting out Duties: Import your archives to the cost following application, and it will smooth out your pay and costs under the suitable assessment classes.
- Instalments and Solicitations: Acknowledge and pay from Visas, check cards, net banking, versatile wallets, and bank moves, and track the situation with your solicitations and bills in the portable application itself. Additionally, the following application sends reminders for instalments and naturally coordinates the instalments with solicitations.
- Reports: The cost following application creates and sends reports to give an itemized understanding about benefits, misfortunes, spending plans, pay, monetary records, and so on.,
- Online business combination: Coordinate your cost following application with your Internet business store and track your deals through instalments got by means of various instalment strategies.
- Merchants and Workers for hire: Oversee and follow every one of the instalments to the sellers and project workers added to the versatile application.
- Access control: Increment your group efficiency by giving access control to specific clients through custom authorizations.
- Track Activities: Decide project benefit by following work costs, finance, costs, and so forth, of your continuous venture.
- Stock following: A cost following application can do everything. Right from following items or the cost of merchandise, sending ready notices when the item is running unavailable or the item isn't selling, to buy orders.
- Top to bottom experiences and examination: Gives in-constructed devices to produce reports with easy to-comprehend visuals and illustrations to acquire bits of knowledge about the presentation of your business.
- Intermittent Costs: Depend on your planning application to follow, smooth out, and robotize every one of the repetitive costs and remind you on a convenient premise.

10. ADVANTAGES & DISADVANTAGES:

ADVANTAGES:

One of the significant aces of following spending is continuously monitoring the condition of one's individual budgets. Following what you spend can assist you with adhering to your financial plan, in an overall way, however in every class like lodging, food, transportation and gifts. While a con is that physically following all money that is spent can be disturbing as well as tedious, a genius is that doing this naturally can be fast and basic. Another genius is that numerous programmed spending following programming programs are accessible free of charge. Having the program on a hand-held gadget can be a principal genius since it tends to be checked prior to spending happens to make certain of the accessible financial plan.

DISADVANTAGES:

A con with any framework used to follow spending is that one might begin doing it then tighten until it's overlooked all together. However, this is a gamble for any new objective, for example, attempting to get in shape or stopped smoking. In the event that an individual first makes a financial plan arrangement, places cash in reserve funds prior to spending any each new payroll interval or month, the following objective can help. Along these lines, following spending and ensuring all receipts are represented just should be done more than once per month. Indeed, even with consistent following of one's ways of managing money, there is no assurance that monetary objectives will be met. Albeit this can be viewed as a con of following spending, it very well may be changed into an ace if one makes up their psyche to continue to attempt to deal with all funds appropriately.

11. CONCLUSION

From this venture, we can oversee and continue to follow the everyday costs as well as pay. While making this task, we acquired a great deal of involvement of functioning collectively. We found different anticipated and unpredicted issues and we partook in a great deal tackling them collectively. We embraced things like video instructional exercises, text instructional exercises, web and learning materials to make our task total.

12. FUTURE SCOPE

The venture helps well to keep the pay and costs overall. In any case, this task has a few constraints:

- The application can't keep up with the reinforcement of information once it is uninstalled.
- This application doesn't give higher choice capacity.

To additional upgrade the ability of this application, we prescribe the accompanying elements to be integrated into the framework:

- Different language interface.
- Give reinforcement and recuperation of information.
- Give better UI to client.
- Portable applications advantage.

13. APPENDIX:

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

This is a temporary script file.
"""

from flask import Flask,
render_template, request, redirect,
session
# from flask_mysqldb import MySQL
# import MySQLdb.cursors
import re

from flask_db2 import DB2
import ibm_db
import ibm_db_dbi

# from 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.dat
abases.appdomain.cloud"
dsn_uid = "sbb93800"
dsn_pwd = "wobsVLm6ccFxcNLe"
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "bludb"
dsn_port = "31498"
dsn_protocol = "tcpip"

dsn = (
    "DRIVER={0};"
    "DATABASE={1};"
    "HOSTNAME={2};"
    "PORT={3};"
    "PROTOCOL={4};"
    "UID={5};"
    "PWD={6};"
```

```
).format(dsn_driver, dsn_database,
dsn_hostname, dsn_port, dsn_protocol,
dsn_uid, dsn_pwd)
"""

# app.config['DB2_DRIVER'] = '{IBM
DB2 ODBC DRIVER}'
app.config['database'] = 'bludb'
app.config['hostname'] = '3883e7e4-
18f5-4afe-be8c-
fa31c41761d2.bs2io90l08kqb1od8lcg.dat
abases.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=388
3e7e4-18f5-4afe-be8c-
fa31c41761d2.bs2io90l08kqb1od8lcg.dat
abases.appdomain.cloud;port=31498;pro
tocol=tcpip;\

uid=sbb93800;pwd=wobsVLm6ccFxcNLe;sec
urity=SSL'
    ibm_db_conn =
ibm_db.connect(conn_str,'','')

    print("Database connected without
any error !!")
except:
    print("IBM DB Connection error
:     " + DB2.conn_errormsg())


# app.config['']


# mysql = MySQL(app)



#HOME--PAGE
@app.route("/home")
def home():
    return
render_template("homepage.html")


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



#SIGN--UP--OR--REGISTER


@app.route("/signup")
def signup():
```

```
    return
render_template("signup.html")



@app.route('/register', methods
=['GET', 'POST'])
def register():
    msg = ''
    print("Break point1")
    if request.method == 'POST' :
        username =
request.form['username']
        email = request.form['email']
        password =
request.form['password']



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



        try:
            print("Break point3")
            connectionID =
ibm_db_dbi.connect(conn_str, '', '')
            cursor =
connectionID.cursor()
            print("Break point4")
        except:
            print("No connection
Established")


        # cursor =
mysql.connection.cursor()
        # with app.app_context():
        #     print("Break point3")
        #     cursor =
ibm_db_conn.cursor()
        #     print("Break point4")


        print("Break point5")
        sql = "SELECT * FROM register
WHERE username = ?"
        stmt =
ibm_db.prepare(ibm_db_conn, sql)
        ibm_db.bind_param(stmt, 1,
username)
        ibm_db.execute(stmt)
        result = ibm_db.execute(stmt)
        print(result)
        account =
ibm_db.fetch_row(stmt)
        print(account)



        param = "SELECT * FROM
register WHERE username = " + "\'" +
username + "\'"
        res =
ibm_db.exec_immediate(ibm_db_conn,
param)
        print("---- ")
        dictionary =
ibm_db.fetch_assoc(res)
```

```
        while dictionary != False:
            print("The ID is : ", dictionary["USERNAME"])
            dictionary = ibm_db.fetch_assoc(res)



        # dictionary = ibm_db.fetch_assoc(result)
        # cursor.execute(stmt)



        # account = cursor.fetchone()
        # print(account)



        # while ibm_db.fetch_row(result) != False:
        #     # account = ibm_db.result(stmt)
        #     print(ibm_db.result(result, "username"))



        # print(dictionary["username"])
        print("break point 6")
        if account:
            msg = 'Username already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email address !'
        elif not re.match(r'[A-Za-z0-9]+', username):
            msg = 'name must contain only characters and numbers !'
        else:
            sql2 = "INSERT INTO register (username, email,password) VALUES (?, ?, ?)"
            stmt2 = ibm_db.prepare(ibm_db_conn, sql2)
            ibm_db.bind_param(stmt2, 1, username)
            ibm_db.bind_param(stmt2, 2, email)
            ibm_db.bind_param(stmt2, 3, password)
            ibm_db.execute(stmt2)
            # cursor.execute('INSERT INTO register VALUES (NULL, % s, % s, % s)', (username, email,password))
            # mysql.connection.commit()
            msg = 'You have successfully registered !'
        return render_template('signup.html', msg = msg)



 #LOGIN--PAGE
```

```
@app.route("/signin")
def signin():
    return render_template("login.html")


@app.route('/login',methods =['GET', 'POST'])
def login():
    global userid
    msg = ''



    if request.method == 'POST' :
        username = request.form['username']
        password = request.form['password']
        # cursor = mysql.connection.cursor()
        # cursor.execute('SELECT * FROM register WHERE username = % s AND password = % s', (username, password ),)
        # account = cursor.fetchone()
        # print (account)


        sql = "SELECT * FROM register WHERE username = ? and password = ?"
        stmt = ibm_db.prepare(ibm_db_conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        result = ibm_db.execute(stmt)
        print(result)
        account = ibm_db.fetch_row(stmt)
        print(account)


        param = "SELECT * FROM register WHERE username = " + "\'" + username + "\'" + " and password = " + "\'" + password + "\'"
        res = ibm_db.exec_immediate(ibm_db_conn, param)
        dictionary = ibm_db.fetch_assoc(res)



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



        if account:
            session['loggedin'] = True
            session['id'] = dictionary["ID"]
            userid = dictionary["ID"]
            session['username'] = dictionary["USERNAME"]
            session['email'] = dictionary["EMAIL"]
```

```
            return redirect('/home')
        else:
            msg = 'Incorrect username / password !'

    return render_template('login.html', msg = msg)
```

```
#ADDING----DATA


@app.route("/add")
def adding():
    return render_template('add.html')
```

```
@app.route('/addexpense',methods=['GET', 'POST'])
def addexpense():

    date = request.form['date']
    expensename = request.form['expensename']
    amount = request.form['amount']
    paymode = request.form['paymode']
    category = request.form['category']


    print(date)
    p1 = date[0:10]
    p2 = date[11:13]
    p3 = date[14:]
    p4 = p1 + "-" + p2 + "." + p3 + ".00"
    print(p4)
    # cursor = mysql.connection.cursor()
    # cursor.execute('INSERT INTO expenses VALUES (NULL,  % s, % s, % s, % s, % s, % s)', (session['id'] ,date, expensename, amount, paymode, category))
    # mysql.connection.commit()
    # print(date + " " + expensename + " " + amount + " " + paymode + " " + category)


    sql = "INSERT INTO expenses (userid, date, expensename, amount, paymode, category) VALUES (?, ?, ?, ?, ?, ?)"
    stmt = ibm_db.prepare(ibm_db_conn, sql)
```

```
    ibm_db.bind_param(stmt, 1,
session['id'])
    ibm_db.bind_param(stmt, 2, p4)
    ibm_db.bind_param(stmt, 3,
expensename)
    ibm_db.bind_param(stmt, 4,
amount)
    ibm_db.bind_param(stmt, 5,
paymode)
    ibm_db.bind_param(stmt, 6,
category)
    ibm_db.execute(stmt)



    print("Expenses added")



    # email part



    param = "SELECT * FROM expenses
WHERE userid = " + str(session['id'])
+ " AND MONTH(date) = MONTH(current
timestamp) AND YEAR(date) =
YEAR(current timestamp) ORDER BY date
DESC"
    res =
ibm_db.exec_immediate(ibm_db_conn,
param)
    dictionary =
ibm_db.fetch_assoc(res)
    expense = []
    while dictionary != False:
        temp = []
        temp.append(dictionary["ID"])
temp.append(dictionary["USERID"])

temp.append(dictionary["DATE"])

temp.append(dictionary["EXPENSENAME"]
)

temp.append(dictionary["AMOUNT"])

temp.append(dictionary["PAYMODE"])

temp.append(dictionary["CATEGORY"])
        expense.append(temp)
        print(temp)
        dictionary =
ibm_db.fetch_assoc(res)



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



    param = "SELECT id, limitss FROM
limits WHERE userid = " +
str(session['id']) + " ORDER BY id
DESC LIMIT 1"
    res =
ibm_db.exec_immediate(ibm_db_conn,
param)
    dictionary =
ibm_db.fetch_assoc(res)
    row = []
    s = 0
```

```
    while dictionary != False:
        temp = []
temp.append(dictionary["LIMITSS"])
        row.append(temp)
        dictionary =
ibm_db.fetch_assoc(res)
        s = temp[0]


    if total > int(s):
        msg = "Hello " +
session['username'] + " , " + "you
have crossed the monthly limit Team
Personal Expense Tracker."

#sendmail(msg,session['email'])

    return redirect("/display")



#DISPLAY---graph


@app.route("/display")
def display():

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

    # cursor =
mysql.connection.cursor()
    # cursor.execute('SELECT * FROM
expenses WHERE userid = % s AND date
ORDER BY `expenses`.`date`
DESC',(str(session['id'])))
    # expense = cursor.fetchall()


    param = "SELECT * FROM expenses
WHERE userid = " + str(session['id'])
+ " ORDER BY date DESC"
    res =
ibm_db.exec_immediate(ibm_db_conn,
param)
    dictionary =
ibm_db.fetch_assoc(res)
    expense = []
    while dictionary != False:
        temp = []
        temp.append(dictionary["ID"])
temp.append(dictionary["USERID"])

temp.append(dictionary["DATE"])

temp.append(dictionary["EXPENSENAME"]
)

temp.append(dictionary["AMOUNT"])

temp.append(dictionary["PAYMODE"])

temp.append(dictionary["CATEGORY"])
        expense.append(temp)
        print(temp)
```

```
        dictionary =
ibm_db.fetch_assoc(res)



    return
render_template('display.html'
,expense = expense)

#delete---the--data

@app.route('/delete/<string:id>',
methods = ['POST', 'GET' ])

def delete(id):

    #  cursor =
mysql.connection.cursor()

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

    #  mysql.connection.commit()



    param = "DELETE FROM expenses
WHERE  id = " + id

    res =
ibm_db.exec_immediate(ibm_db_conn,
param)



    print('deleted successfully')

    return redirect("/display")



#UPDATE---DATA



@app.route('/edit/<id>', methods =
['POST', 'GET' ])
```

```
def edit(id):

    # cursor =
mysql.connection.cursor()

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

    # row = cursor.fetchall()



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

    res =
ibm_db.exec_immediate(ibm_db_conn,
param)

    dictionary =
ibm_db.fetch_assoc(res)

    row = []

    while dictionary != False:

        temp = []

        temp.append(dictionary["ID"])


temp.append(dictionary["USERID"])


temp.append(dictionary["DATE"])


temp.append(dictionary["EXPENSENAME"]
)


temp.append(dictionary["AMOUNT"])


temp.append(dictionary["PAYMODE"])


temp.append(dictionary["CATEGORY"])

        row.append(temp)

        print(temp)
```

```
        dictionary =
ibm_db.fetch_assoc(res)



    print(row[0])

    return
render_template('edit.html', expenses
= row[0])




@app.route('/update/<id>', methods =
['POST'])

def update(id):

  if request.method == 'POST' :



      date = request.form['date']

      expensename =
request.form['expensename']

      amount = request.form['amount']

      paymode =
request.form['paymode']

      category =
request.form['category']



    #   cursor =
mysql.connection.cursor()

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

    #   mysql.connection.commit()



      p1 = date[0:10]

      p2 = date[11:13]

      p3 = date[14:]

      p4 = p1 + "-" + p2 + "." + p3 +
".00"



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

      stmt =
ibm_db.prepare(ibm_db_conn, sql)

      ibm_db.bind_param(stmt, 1, p4)

      ibm_db.bind_param(stmt, 2,
expensename)

      ibm_db.bind_param(stmt, 3,
amount)

      ibm_db.bind_param(stmt, 4,
paymode)

      ibm_db.bind_param(stmt, 5,
category)

      ibm_db.bind_param(stmt, 6, id)

      ibm_db.execute(stmt)

      print('successfully updated')

      return redirect("/display")

 #limit

@app.route("/limit" )

def limit():

       return redirect('/limitn')
```

```
@app.route("/limitnum" , methods = ['POST' ])
def limitnum():
     if request.method == "POST":
         number= request.form['number']
        #  cursor = mysql.connection.cursor()
        #  cursor.execute('INSERT INTO limits VALUES (NULL, % s, % s) ',(session['id'], number))
        #  mysql.connection.commit()
         sql = "INSERT INTO limits (userid, limitss) VALUES (?, ?)"
         stmt = ibm_db.prepare(ibm_db_conn, sql)
         ibm_db.bind_param(stmt, 1, session['id'])
         ibm_db.bind_param(stmt, 2, number)
         ibm_db.execute(stmt)

         return redirect('/limitn')
@app.route("/limitn")
def limitn():
    # cursor = mysql.connection.cursor()
    # cursor.execute('SELECT limitss FROM `limits` ORDER BY `limits`.`id` DESC LIMIT 1')
    # x= cursor.fetchone()
    # s = x[0]
    param = "SELECT id, limitss FROM limits WHERE userid = " + str(session['id']) + " ORDER BY id DESC LIMIT 1"
    res = ibm_db.exec_immediate(ibm_db_conn, param)
    dictionary = ibm_db.fetch_assoc(res)
    row = []
    s = " /-"
    while dictionary != False:
        temp = []

temp.append(dictionary["LIMITSS"])
        row.append(temp)
        dictionary = ibm_db.fetch_assoc(res)
        s = temp[0]


    return render_template("limit.html" , y= s)
#REPORT
@app.route("/today")
def today():
    #   cursor = mysql.connection.cursor()
    #   cursor.execute('SELECT TIME(date)   , amount FROM expenses WHERE userid = %s AND DATE(date) = DATE(NOW()) ',(str(session['id'])))
    #   texpense = cursor.fetchall()
    #   print(texpense)
```

```
      param1 = "SELECT TIME(date) as
tn, amount FROM expenses WHERE userid
= " + str(session['id']) + " AND
DATE(date) = DATE(current timestamp)
ORDER BY date DESC"

      res1 =
ibm_db.exec_immediate(ibm_db_conn,
param1)

      dictionary1 =
ibm_db.fetch_assoc(res1)

      texpense = []

      while dictionary1 != False:

          temp = []


temp.append(dictionary1["TN"])


temp.append(dictionary1["AMOUNT"])

          texpense.append(temp)

          print(temp)

          dictionary1 =
ibm_db.fetch_assoc(res1)

    #   cursor =
mysql.connection.cursor()

    #   cursor.execute('SELECT * FROM
expenses WHERE userid = % s AND
DATE(date) = DATE(NOW()) AND date
ORDER BY `expenses`.`date`
DESC',(str(session['id'])))

    #   expense = cursor.fetchall()

      param = "SELECT * FROM expenses
WHERE userid = " + str(session['id'])
+ " AND DATE(date) = DATE(current
timestamp) ORDER BY date DESC"

      res =
ibm_db.exec_immediate(ibm_db_conn,
param)

      dictionary =
ibm_db.fetch_assoc(res)

      expense = []

      while dictionary != False:

          temp = []


temp.append(dictionary["ID"])


temp.append(dictionary["USERID"])


temp.append(dictionary["DATE"])


temp.append(dictionary["EXPENSENAME"]
)


temp.append(dictionary["AMOUNT"])


temp.append(dictionary["PAYMODE"])


temp.append(dictionary["CATEGORY"])

          expense.append(temp)

          print(temp)

          dictionary =
ibm_db.fetch_assoc(res)

      total=0

      t_food=0

      t_entertainment=0

      t_business=0

      t_rent=0

      t_EMI=0

      t_other=0

      for x in expense:

          total += x[4]
```

```
            if x[6] == "food":
                t_food += x[4]

            elif x[6] == "entertainment":
                t_entertainment  += x[4]

            elif x[6] == "business":
                t_business  += x[4]
            elif x[6] == "rent":
                t_rent  += x[4]

            elif x[6] == "EMI":
                t_EMI  += x[4]

            elif x[6] == "other":
                t_other  += x[4]

        print(total)

        print(t_food)
        print(t_entertainment)
        print(t_business)
        print(t_rent)
        print(t_EMI)
        print(t_other)
        return render_template("today.html", texpense = texpense, expense = expense,  total = total , t_food = t_food,t_entertainment = t_entertainment,
                               t_business = t_business,  t_rent =  t_rent,
                               t_EMI = t_EMI,  t_other =  t_other )

@app.route("/month")
def month():
    #   cursor = mysql.connection.cursor()
    #   cursor.execute('SELECT DATE(date), SUM(amount) FROM expenses WHERE userid= %s AND MONTH(DATE(date))= MONTH(now()) GROUP BY DATE(date) ORDER BY DATE(date) ',(str(session['id'])))
    #   texpense = cursor.fetchall()
    #   print(texpense)


      param1 = "SELECT DATE(date) as dt, SUM(amount) as tot FROM expenses WHERE userid = " + str(session['id']) + " AND MONTH(date) = MONTH(current timestamp) AND YEAR(date) = YEAR(current timestamp) GROUP BY DATE(date) ORDER BY DATE(date)"
      res1 = ibm_db.exec_immediate(ibm_db_conn, param1)
      dictionary1 = ibm_db.fetch_assoc(res1)
      texpense = []
      while dictionary1 != False:
          temp = []
```

```
temp.append(dictionary1["DT"])

temp.append(dictionary1["TOT"])
          texpense.append(temp)
          print(temp)
          dictionary1 =
ibm_db.fetch_assoc(res1)
    #   cursor =
mysql.connection.cursor()
    #   cursor.execute('SELECT * FROM
expenses WHERE userid = % s AND
MONTH(DATE(date))= MONTH(now()) AND
date ORDER BY `expenses`.`date`
DESC',(str(session['id'])))
    #   expense = cursor.fetchall()
      param = "SELECT * FROM expenses
WHERE userid = " + str(session['id'])
+ " AND MONTH(date) = MONTH(current
timestamp) AND YEAR(date) =
YEAR(current timestamp) ORDER BY date
DESC"
      res =
ibm_db.exec_immediate(ibm_db_conn,
param)
      dictionary =
ibm_db.fetch_assoc(res)
      expense = []
      while dictionary != False:
          temp = []

temp.append(dictionary["ID"])

temp.append(dictionary["USERID"])

temp.append(dictionary["DATE"])

temp.append(dictionary["EXPENSENAME"]
)

temp.append(dictionary["AMOUNT"])

temp.append(dictionary["PAYMODE"])

temp.append(dictionary["CATEGORY"])
          expense.append(temp)
          print(temp)
          dictionary =
ibm_db.fetch_assoc(res)
      total=0
      t_food=0
      t_entertainment=0
      t_business=0
      t_rent=0
      t_EMI=0
      t_other=0
      for x in expense:
          total += x[4]
          if x[6] == "food":
              t_food += x[4]

          elif x[6] ==
"entertainment":
              t_entertainment  +=
x[4]

          elif x[6] == "business":
              t_business  += x[4]
```

```
            elif x[6] == "rent":
                t_rent  += x[4]

            elif x[6] == "EMI":
                t_EMI  += x[4]

            elif x[6] == "other":
                t_other  += x[4]

        print(total)
        print(t_food)
        print(t_entertainment)
        print(t_business)
        print(t_rent)
        print(t_EMI)
        print(t_other)
        return
render_template("today.html",
texpense = texpense, expense =
expense,  total = total ,
                            t_food =
t_food,t_entertainment =
t_entertainment,
                            t_business
= t_business,  t_rent =  t_rent,
                            t_EMI =
t_EMI,  t_other =  t_other )


@app.route("/year")
def year():
    #   cursor =
mysql.connection.cursor()
    #   cursor.execute('SELECT
MONTH(date), SUM(amount) FROM
expenses WHERE userid= %s AND
YEAR(DATE(date))= YEAR(now()) GROUP
BY MONTH(date) ORDER BY MONTH(date)
',(str(session['id'])))
    #   texpense = cursor.fetchall()
    #   print(texpense)
      param1 = "SELECT MONTH(date) as
mn, SUM(amount) as tot FROM expenses
WHERE userid = " + str(session['id'])
+ " AND YEAR(date) = YEAR(current
timestamp) GROUP BY MONTH(date) ORDER
BY MONTH(date)"
      res1 =
ibm_db.exec_immediate(ibm_db_conn,
param1)
      dictionary1 =
ibm_db.fetch_assoc(res1)
      texpense = []


      while dictionary1 != False:
          temp = []

temp.append(dictionary1["MN"])

temp.append(dictionary1["TOT"])
          texpense.append(temp)
          print(temp)
          dictionary1 =
ibm_db.fetch_assoc(res1)
    #   cursor =
mysql.connection.cursor()
    #   cursor.execute('SELECT * FROM
expenses WHERE userid = % s AND
```

```
YEAR(DATE(date))= YEAR(now()) AND
date ORDER BY `expenses`.`date`
DESC',(str(session['id'])))

    #   expense = cursor.fetchall()

      param = "SELECT * FROM expenses
WHERE userid = " + str(session['id'])
+ " AND YEAR(date) = YEAR(current
timestamp) ORDER BY date DESC"

      res =
ibm_db.exec_immediate(ibm_db_conn,
param)

      dictionary =
ibm_db.fetch_assoc(res)

      expense = []

      while dictionary != False:

          temp = []


temp.append(dictionary["ID"])


temp.append(dictionary["USERID"])


temp.append(dictionary["DATE"])


temp.append(dictionary["EXPENSENAME"]
)


temp.append(dictionary["AMOUNT"])


temp.append(dictionary["PAYMODE"])


temp.append(dictionary["CATEGORY"])

          expense.append(temp)

          print(temp)

          dictionary =
ibm_db.fetch_assoc(res)
```

```
      total=0

      t_food=0

      t_entertainment=0

      t_business=0

      t_rent=0

      t_EMI=0

      t_other=0

      for x in expense:

          total += x[4]

          if x[6] == "food":

              t_food += x[4]


          elif x[6] ==
"entertainment":

              t_entertainment  +=
x[4]


          elif x[6] == "business":

              t_business  += x[4]

          elif x[6] == "rent":

              t_rent  += x[4]


          elif x[6] == "EMI":

              t_EMI  += x[4]


          elif x[6] == "other":

              t_other  += x[4]

      print(total)

      print(t_food)
```

```
        print(t_entertainment)
        print(t_business)
        print(t_rent)
        print(t_EMI)
        print(t_other)
        return
render_template("today.html",
texpense = texpense, expense =
expense,  total = total ,
                              t_food =
t_food,t_entertainment =
t_entertainment,
                              t_business
= t_business,  t_rent =  t_rent,
                              t_EMI =
t_EMI,  t_other =  t_other )
```

```
#log-out
@app.route('/logout')
def logout():
   session.pop('loggedin', None)
   session.pop('id', None)
   session.pop('username', None)
   session.pop('email', None)
   return
render_template('home.html')
port = os.getenv('VCAP_APP_PORT',
'8080')
if __name__ == "__main__":
    app.secret_key = os.urandom(12)
    app.run(debug=True,
host='0.0.0.0', port=port)
```

Project Demonstration Video Link :

https://drive.google.com/file/d/1_D_AmIvDlSe6-U2uwk6gUWr--fUyxd17/view?usp=share_link