

PROJECT REPORT
PERSONAL EXPENSE TRACKER APPLICATION
CLOUD APPLICATION
TEAM ID: PNT2022MID29782

Submitted by

GOWSHIKA N.M	(610519104031)
ABITHA T	(610519104002)
DHIVYABHARATHI R	(610519104020)
KAVITHASRI V	(610519104051)

CHAPTER NO	TITLE
1.	INTRODUCTION
	1.1 Project Overview
	1.2 Purpose
2.	LITERATURE SURVEY
	2.1 Existing problem
	2.2 References
	2.3 Problem Statement Definition
3.	IDEATION & PROPOSED SOLUTION
	3.1 Empathy Map Canvas
	3.2 Ideation & Brainstorming
	3.3 Proposed Solution
	3.4 Problem Solution fit
4.	REQUIREMENT ANALYSIS
	4.1 Functional requirement
	4.2 Non-Functional requirements
5.	PROJECT DESIGN
	5.1 Data Flow Diagrams
	5.2 Solution & Technical Architecture
	5.3 User Stories
6.	PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

6.2 Sprint Delivery Schedule

6.3 Reports from JIRA

7. CODING & SOLUTIONING

**(Explain the features added in the project
along with code)**

7.1 Feature 1

7.2 Feature 2

7.3 Database Schema (if Applicable)

8. TESTING

8.1 Test Cases

8.2 User Acceptance Testing

9. RESULTS

9.1 Performance Metrics

10. ADVANTAGES & DISADVANTAGES

CONCLUSION

FUTURE SCOPE

APPENDIX

Source Code

GitHub & Project Demo Link

1.INTRODUCTION

1.1Project Overview

Personal Expense Tracker (PET) is a daily expense management system which is specially designed for non- salaried and salaried personnel for keeping track of their daily expenditure with easy and effective way through computerized system which tends to eliminate manual paper works. It will also manage records in systematic way and user can access the stored data conveniently.

We have tried to design the project in such way that user may not have any difficulty in using this application without much effort. This software can be really used by end user who have Android running devices with them. The language that we use to develop this system is HTML, CSS ,JS , Python flask and IBM DB2 for database

1.2Purpose:

Daily Expense Tracker System is a system which will keep a track of Income-Expense of a House-Wife on a day to day basics, This System takes Income from House-Wife and divides in daily expense allowed, If u exceed that days expense it will cut if from your income and give new daily expense allowed Amt, and if that days expense is less it will add it in savings. Daily expense tracking System will generate report at the end of month to show Income-Expense Curve. It will let you add the savings amt which you had saved for some particular Festivals or day like Birthday or Anniversary.

2.LITERATURE SURVEY

A mobile application capable of monitoring and controlling personal expenses, as well as cautioning the user against reckless and unbudgeted spending.” is a research paper published by– Carvalho L.A and Basso C.(2014).Telecom Expense Management for Large Organizations, A Practical Guide. iUniverse LLC Bloomington, IN 47403. This study is aimed at developing an android based mobile application capable of monitoring and controlling personal expenses, as well as cautioning the user against reckless and unbudgeted spending. The developed system was designed using system flowchart, use case diagram, sequence diagram, class diagram and system architecture diagram. It was implemented using Java programming language on android studio and My SQL. The developed system was evaluated based on basic functionality tests performed on the individual modules, the integrated testing as well as the overall function testing. The results of testing the functionalities of the developed system showed that all the modules worked properly when tested individually. They rejected invalid inputs and responded promptly to user requests. Database operations such as insert, update, delete and add that were performed yielded expected results, and data consistency / integrity are maintained in the reports generated. Thus, the developed system provides an easy to use, portable and secured means of enhancing financial sustainability and promotes individual and societal economic growth via fiscal discipline.

This application will have a two-tier architecture: first one is the database tier, where all the data and financial data will be stored. Second it will be the user interface which will support the application user communicate with the system and also store Information in the database. The proposed system should operate offline so it can be accessed at any time without internet availability. The proposed system should provide different categories for the user to select from and they can enter the amount and mode of payment. This system should be able to analyze the information, provide analytics on which category did the user spent most of their money. The proposed system should provide a user interface where the user could store and observe their past expenses. To create this system, we will use the android studio and it.It will be written in Java, Xml.

2.1 Existing problem

It may seem like a lot of work to itemize your expenses when you first begin, but understanding why it's important to track expenses and how to do so with minimal effort can help you successfully commit to the activity and become more aware of your spending. Monitoring your expenses throughout the month holds you accountable for your finances in a few key ways.

After you set up a budget, which is a monthly plan for spending that takes into account your income and expenses, tracking expenses daily is essential to keeping you on that budget. If you don't track your money, you won't know when to stop spending in a given category (food or clothing, for example).

At the end of each month, review the expenses you tracked to compare what you spent versus what you planned to spend according to your budget. If you overspent, look for ways to cut spending in a certain category. If you spent too little, you might want to allocate more to the savings and debt repayment. In either case, you'll want to use what you learn from tracking expenses and factor in any life changes (marriage or a new child, for example) to make changes to the budget for next month that put you on better financial footing.

2.2 References

1. <https://moneyview.in/insights/best-personal-finance-management-apps-in-india>
2. <https://www.factmr.com/report/personal-finance-mobile-app-market>
3. <https://www.moneytap.com/blog/best-money-management-apps/>
4. <https://relevant.software/blog/personal-finance-app-like-mint/>
5. <https://www.onmanorama.com/lifestyle/news/2022/01/11/financial-literacy-trend-among-todays-youth-investment.html>
6. <https://www.livemint.com/money/personal-finance/96-indian-parents-feel-their-children-lack-financial-know-how-survey-11661336110855.html>
7. <https://economictimes.indiatimes.com/small-biz/money/importance-of-financial-literacy-amongst-youngsters/articleshow/85655134.cms>
8. <https://www.news18.com/news/education-career/only-27-adults-16-7-of-indian-teenagers-financially-literate-4644893.html>

2.3 Problem Statement Definition:

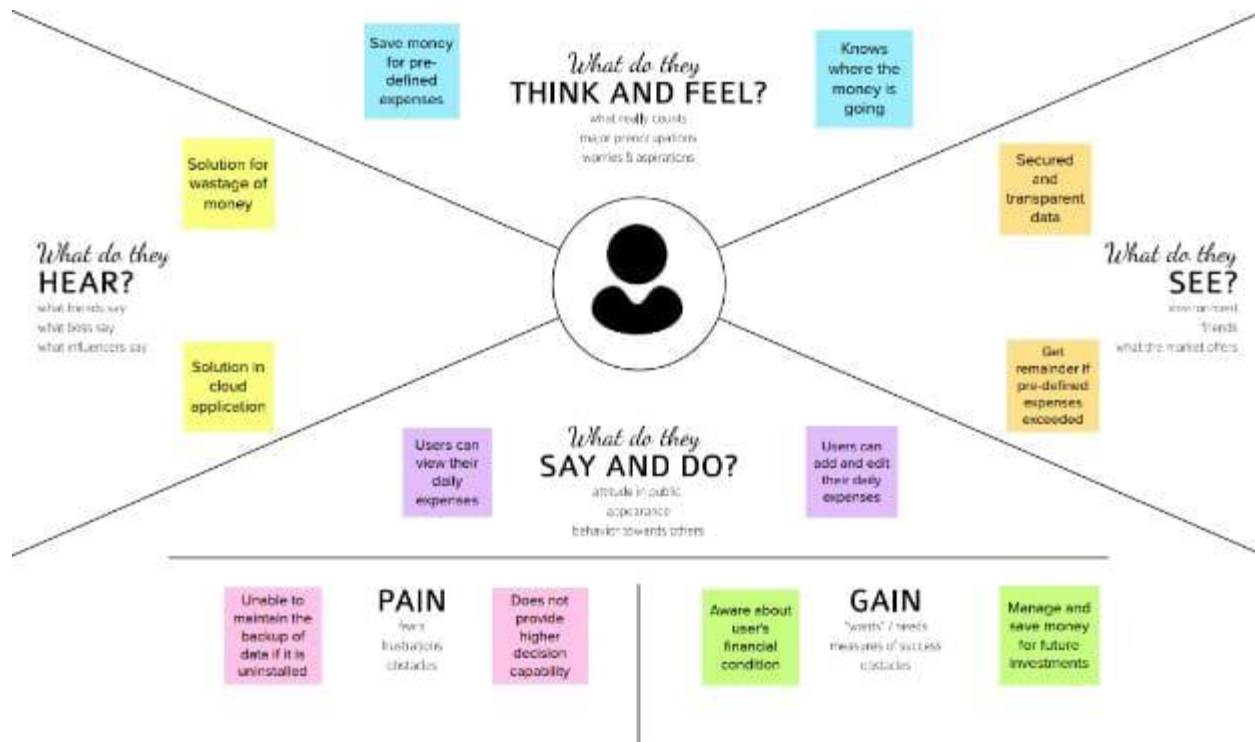
In simple words, personal finance entails all the financial decisions and activities that a Finance app makes your life easier by helping you to manage your finances efficiently. A personal finance app will not only help you with budgeting and accounting but also give you helpful insights about money management. Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert.

3.IDEATION & PROPOSED SOLUTION:

3.1Empathy Map Canvas:

An **empathy map** is a collaborative visualization used to articulate what we know about a particular type of user. It externalizes knowledge about users in order to

- 1) create a shared understanding of user needs, and
- 2) aid in decision making.

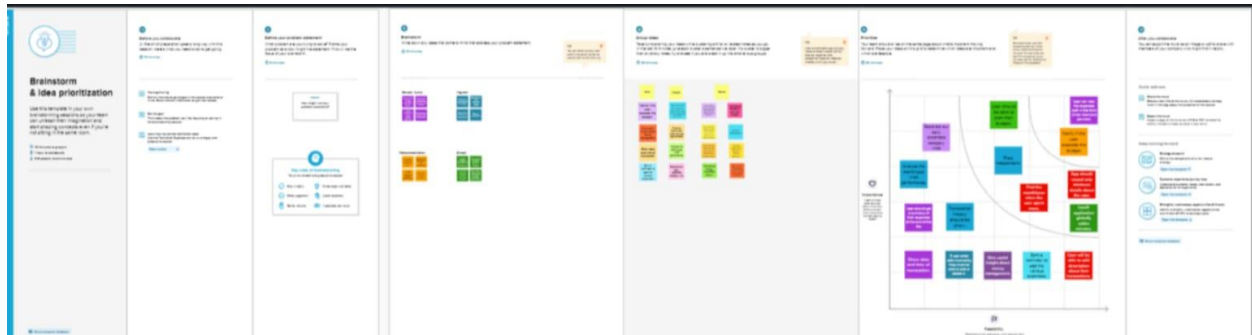


3.2 Ideation & Brainstorming:

Ideation is a creative process where designers generate ideas in sessions (e.g., brainstorming, worst possible idea)

Brainstorming is a method design teams use to generate ideas to solve clearly defined design problems.

Brainstorm & Idea Listing and Grouping:



3.3 Proposed Solution:

1. Problem Statement (Problem to be solved):

Customers live the budget friendly life.

2. Idea / Solution description:

We are building an android application named as “Personal Expense Tracker”. As the name suggests, this project is an android app which is used to track the daily expenses of the user. It is like digital record keeping which keeps the records of expenses done by a user. The application keeps the track of the Income and Expenses both of user on a day-to-day basis. This application takes the income of a user and manage its daily expenses so that the user can save money. If you exceed 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 user’s savings. The application generates report of the expenses of each end of the month. The amount saved can be used for celebrating festivals, Birthdays or Anniversary.

3. Novelty / Uniqueness:

The proposed solution comprises of the Email alert when it reaches the higher limit.

4. Social Impact / Customer Satisfaction:

The users can track their daily expenditure with easy and effective way through computerised system, so the users can save their expenses.

5. Business Model (Revenue Model):

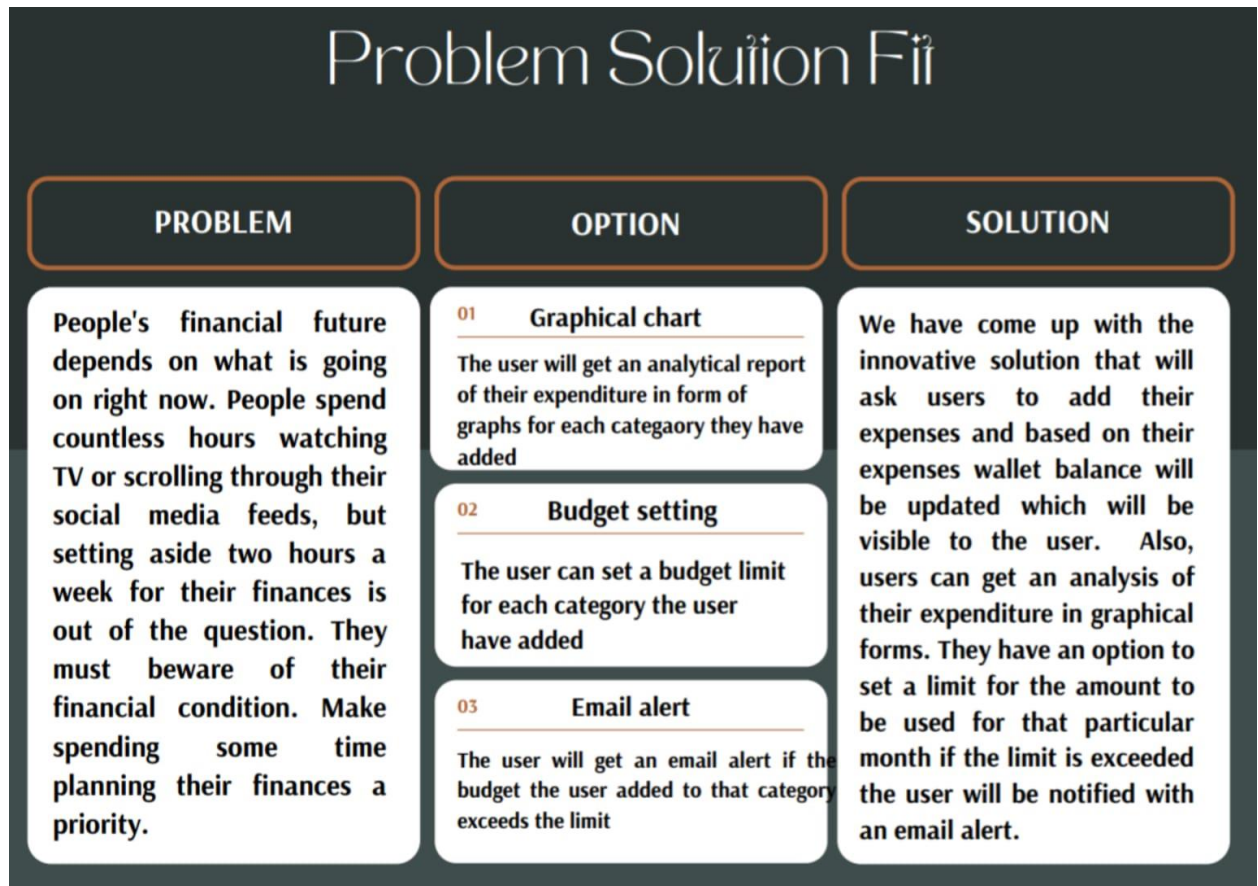
Personal Expense Tracker (PET) is a daily expense management system which is specially designed for non- salaried and salaried personnel for keeping track of their daily expenditure with easy and effective way through computerized system which tends to eliminate manual paper works. It will also manage records in systematic way and user can access the stored data conveniently. We have tried to design the project in such way that user may not have any difficulty in using this application without much effort. This software can be really used by end user who have Android

running devices with them. The language that we use to develop this system is Python , flask and docker.

6.Scalability of the Solution

Whatever may be the expenses done by the user , the user will be aware of this financial conditions.

3.4 Problem Solution fit:



4.REQUIREMENT ANALYSIS:

4.1Functional 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	Orders	Login with EmailID and password
FR-4	Chatbot	Add the financial information for the user
FR-5	Savings	If the budget limit doesn't exceeds , the amount will be added to the savings category

4.2Non-Functional requirements:

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	It helps to keep an accurate record of your money inflow and outflow. It empowers to control spending impulses and eliminate frivolous spending, thereby avoiding debt.
NFR-2	Security	It has encrypted data, accredited data centres and third-party audits to make sure that its security features rise to the standards one would expect from a personal expense tracking app
NFR-3	Reliability	It does the wonderful job of tracking & managing expenses without compromising on security. This app was specifically built to alleviate this risk of security for people who are conscious about it.

		Gives complete user anonymity (no name/phone/email registration) and control over any personal or financial data going out of your phone
NFR-4	Performance	Keep track of all your daily transactions. Keep track of your money lent or borrowed. Suggest you with the best investment options. Offers in popular categories.To read latest authenticated financial news through chatbots
NFR-5	Availability	Equip expense tracking app with a bot that can understand and answer all user queries and address their needs such as account balance, credit score,etc.
NFR-6	Scalability	Boost the productivity of all the processes within the organization. Increase efficiency and customer satisfaction with an app aligned to their needs. Seamlessly integrate with existing infrastructure. Ability to provide valuable insights . Optimize sales processes to generate more revenue through enhanced data collection.

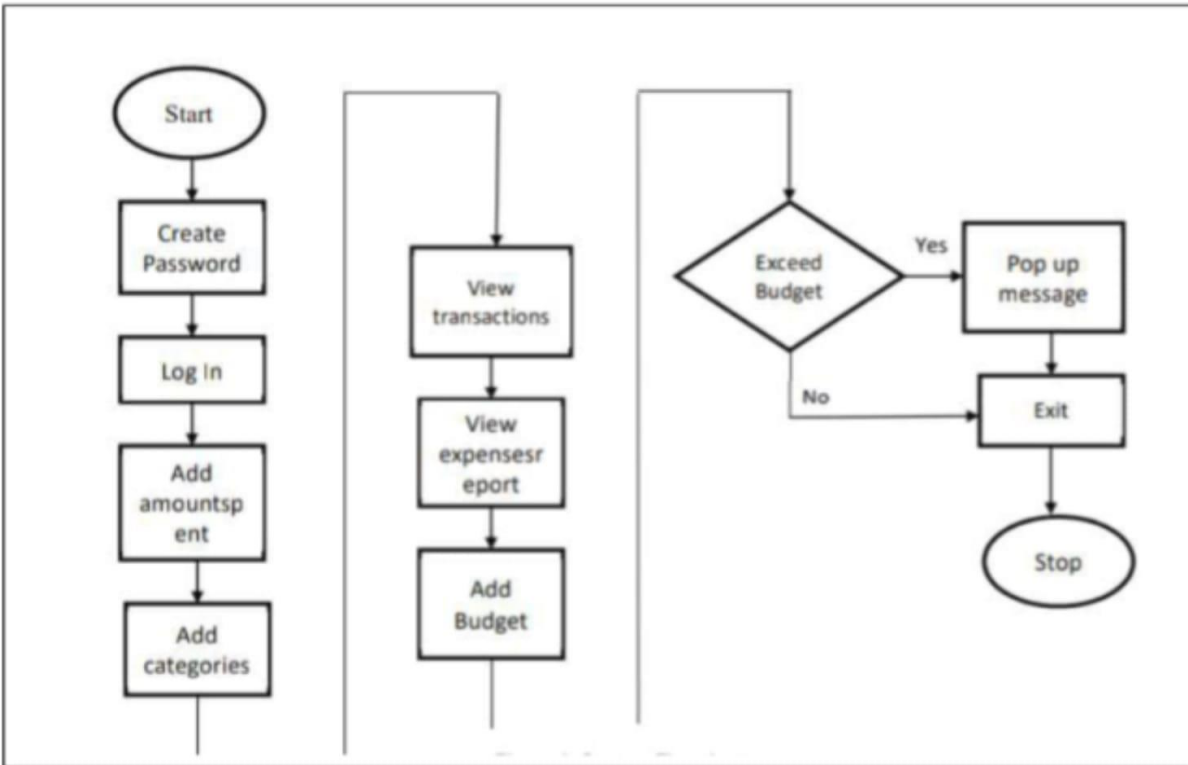
5.PROJECT DESIGN:

5.1 Data Flow Diagrams:

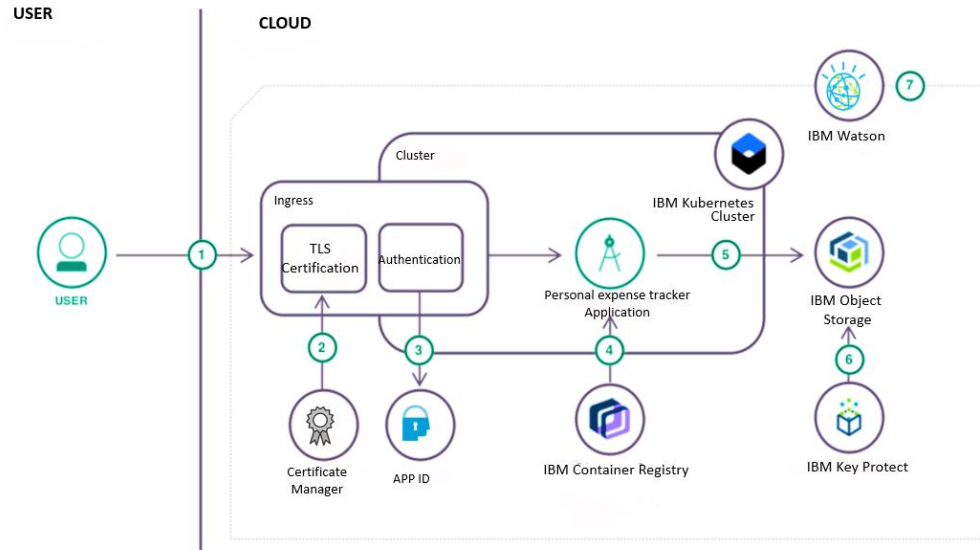
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

Example: (Simplified)





5.2 Solution & Technical Architecture:



5.3 User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance Criteria	Priority	Release
Customer(Mobile user and web user)	Registration	USN-1	As a user , I can register for the application by entering my email , new password and confirming the same 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-2
		USN-3	As a user , I can register for the application through Facebook.	I can register & access the dashboard with a Facebook login.	Low	Sprint-1
		USN-4	As a user, I can register for the application through a Google account.	I can register and access the dashboard with a Google Account login.	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering my email and password.	I can access the application on.	High	Sprint-1

	Dashboard	USN-6	As a user , I can see the expenditure details and the daily expense.	I can view the daily expenses and add the expense details.	high	Sprint-1
Customer care Executive		USN-7	As a customer care executive ,I can solve the problems that customers face.	I can provide support to customers at any time 24*7.	Medium	Sprint-1
Administrator	Application	USN-8	As an administrator ,I can upgrade or update the application.	I can fix any bugs raised by customers and upgrade the application.	Medium	Sprint-1

6.PROJECT PLANNING & SCHEDULING

6.1Sprint Planning & Estimation:

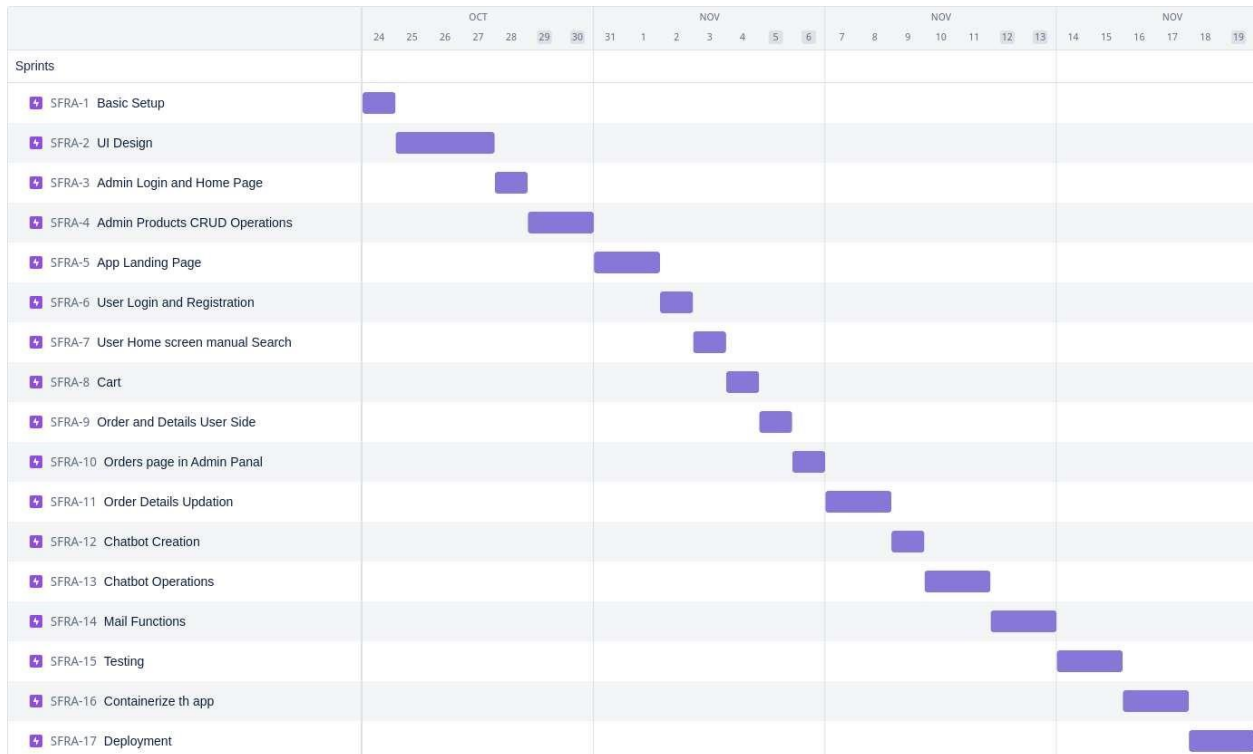
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint - 1	Registration	USN -1	As a user , I can register for the application by entering my email , new password and confirming the same password.	2	High	Gowshika Dhivya Bharathi Abitha
		USN -2	As a user , I will receive confirmation email once I have registered for the application.	1	Low	
	Login	USN -3	As a user , I can log into the application by entering email and password / Google OAuth.	2	High	Abitha Kavithasri
	Dashboard	USN -4	Logging in takes the user to their dashboard.	1	Low	
Sprint - 2		USN -5	As a user ,I will update my salary at the start of each month.	1	Medium	Dhivya Bharathi
		USN -6	As a user , I will set a target/limit to keep track of my expenditure.	1	Medium	Gowshika
	Workspace	USN -7	Workplace for personal expense tracking	1	Medium	kavithasri
	Charts	USN -8	Graphs to show weekly and everyday expenditure	2	High	Dhivya Bharathi
		USN -9	As a user , I can export raw data as csv file.	1	Medium	Abitha

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint - 3	IBM DB2	USN -10	Linking database with dashboard	2	High	Gowshika
		USN -11	Making dashboard interactive with JS	2	High	Abitha
	Watson Assistant	USN -12	Embedding Chatbot to clarify user's queries.	1	Low	Kavithasri
	BCrypt	USN -13	Using BCrypt to store passwords securely.	1	Medium	Dhivya Bharathi
	SendGrid	USN -14	Using SendGrid to send mail to the user. (To alert or remind)	1	Medium	Abitha
Sprint - 4	Integration	USN -15	Integrating frontend and backend.	2	High	Kavithasri
	Docker	USN -16	Creating Docker image of web app.	2	High	Dhivya Bharathi
	Cloud Registry	USN -17	Uploading docker image to IBM cloud registry.	2	High	Abitha
	Kubernetes	USN -18	Creating container using docker and hosting the webapp.	2	High	Gowshika
	Exposing Deployment	USN -19	Exposing IP/Ports for the site.	1	Medium	Kavithasri

6.2Sprint Delivery Schedule:

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

6.3 Reports from JIRA:



7. CODING & SOLUTIONING (Explain the features added in the project along with code

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

7.2 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.

8. TESTING

8.1. Test Cases

With a concerted effort, I conducted research on general well-being to have a rudimentary grasp on health management, as well as the existing job/skill recommender apps in order to get an understanding of what is already existing in the market, the characteristics, specialties, and usability. There are a considerable number of job/skill recommender apps tracking apps existing in the market. They aim to track daily spends intake by logging info given to achieve users' preset goals. To log spend, users can input the expense in the app, or scan the barcode of a package. Most apps allow users to connect with associated activities apps to track spend progress. With a premium upgrade, users can get access to tailor-made saving according to health goals or specified way to spend. In order to build a realistic initial target group, I wanted to conduct some usability tests with 5 users that regularly engage in buying activity and spending tracking, including both first-time and regular users of money planning. I asked these individuals to perform tasks related to general usage

8.2. User Acceptance Testing

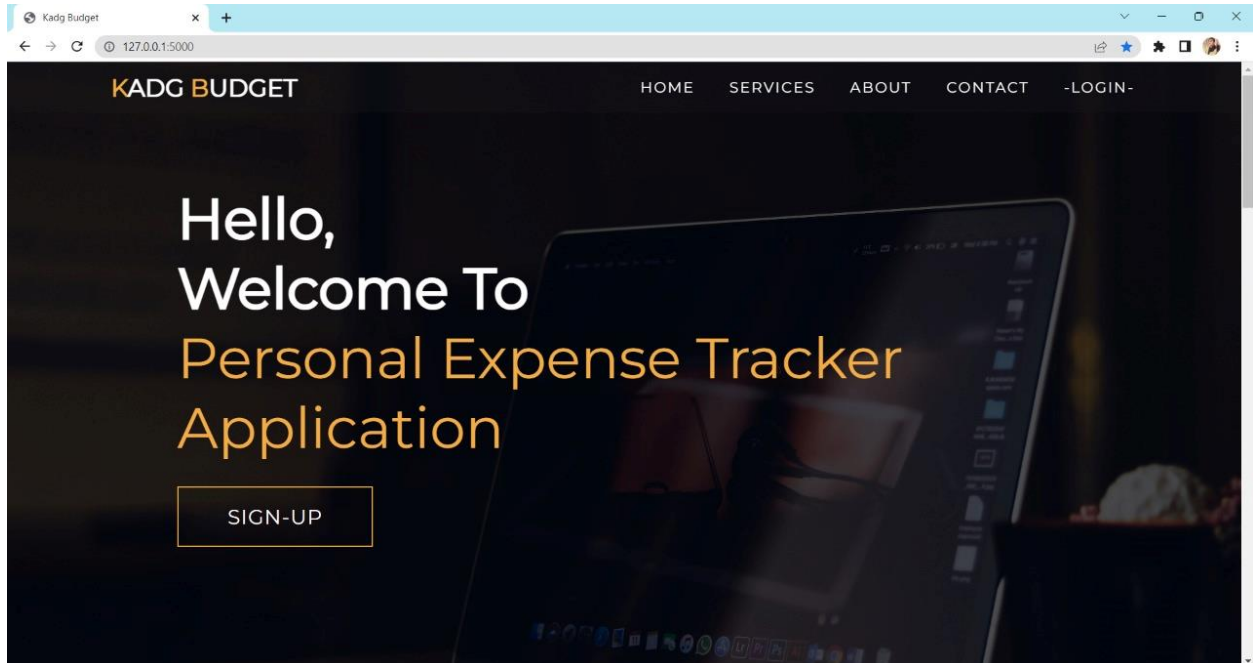
Must-have features of a Personal expense tracker app I wanted to address the user pain points by including (and improving) the core features of the application. Personal profiles After downloading the app, a user needs to register and create an account. At this stage, users should fill in personal information like name, gender, age, height, weight, spend preferences, spend logging and dashboard. Allowing users to analyze their spending habits. They should be able to log expenses and money intake and see their progress on a dashboard that can track overall spends. Push notifications Push notifications are an effective tool for increasing user engagement and retention. To motivate users to keep moving toward their goals, it's pertinent to deliver information on their progress toward the current goal and remind them to log what they spend on. money counter Enabling the application to calculate spent amount of users have gone and done based on the data they've logged. Barcode

scanner Let users count money and see accurate spend information via a built-in barcode

9.RESULTS

9.1. Performance Metrics

Dashboard



Add Expense

Kadg Budget

127.0.0.1:5000/add

KADG Budget Home Add History LIMIT Report User

Add Expense

Date
dd-mm-yyyy


Expense name

Expense Amount

Pay-Mode

Category

Add




CHATBOT ASSISTANCE

Kadg Budget

127.0.0.1:5000/home

KADG Budget Home Add History LIMIT Report User



LET'S START J

Kadg web application helps you to maintain budget and analyse the expense...

Let's Begin

Watson Assistant

expenses

Transfer money

Before we can transfer any funds, please confirm where you'd like to transfer funds from.

Checking account Savings account

Checking account

Okay! How much would you like to transfer?

1000

Alright. When would you like to make the transfer?

Choose a date (mm/dd/yyyy)

mm/dd/yyyy

Type something...

Built with IBM Watson

Edit Expense

Kadg Budget

127.0.0.1:5000/edit/6

KADG BudgetHomeAddHistoryLIMITReport

User

Edit Expense

Date

dd-mm-yyyy

Expense name

Trip

Expense Amount

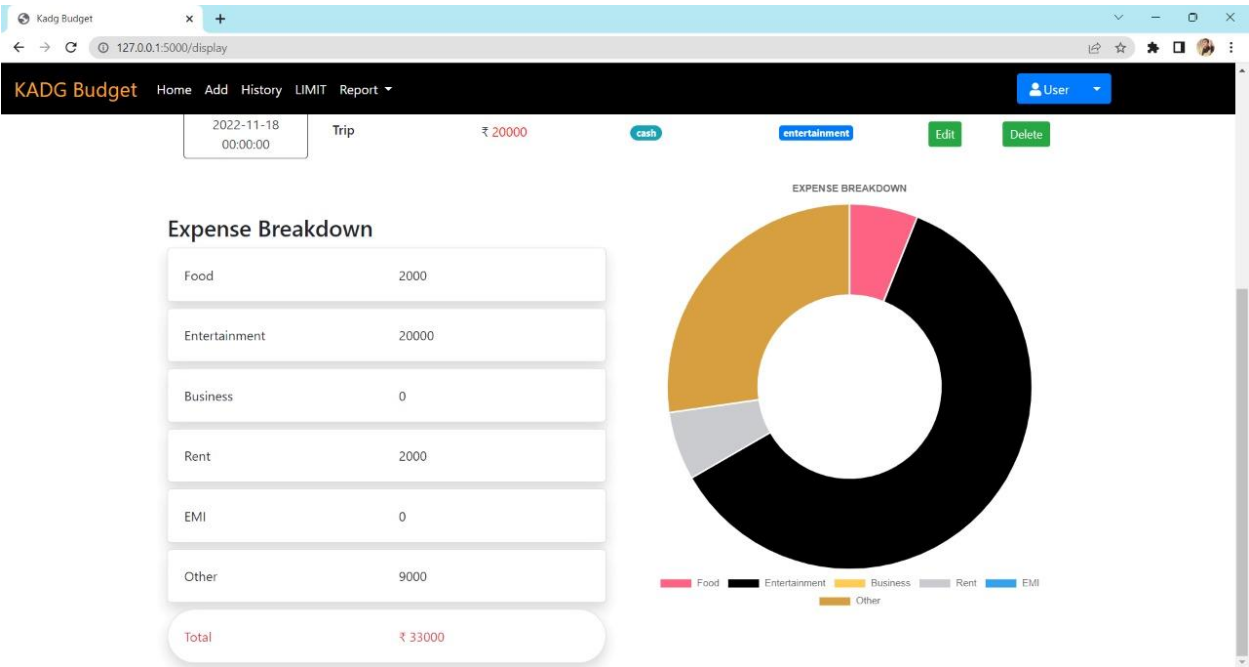
20000

cash

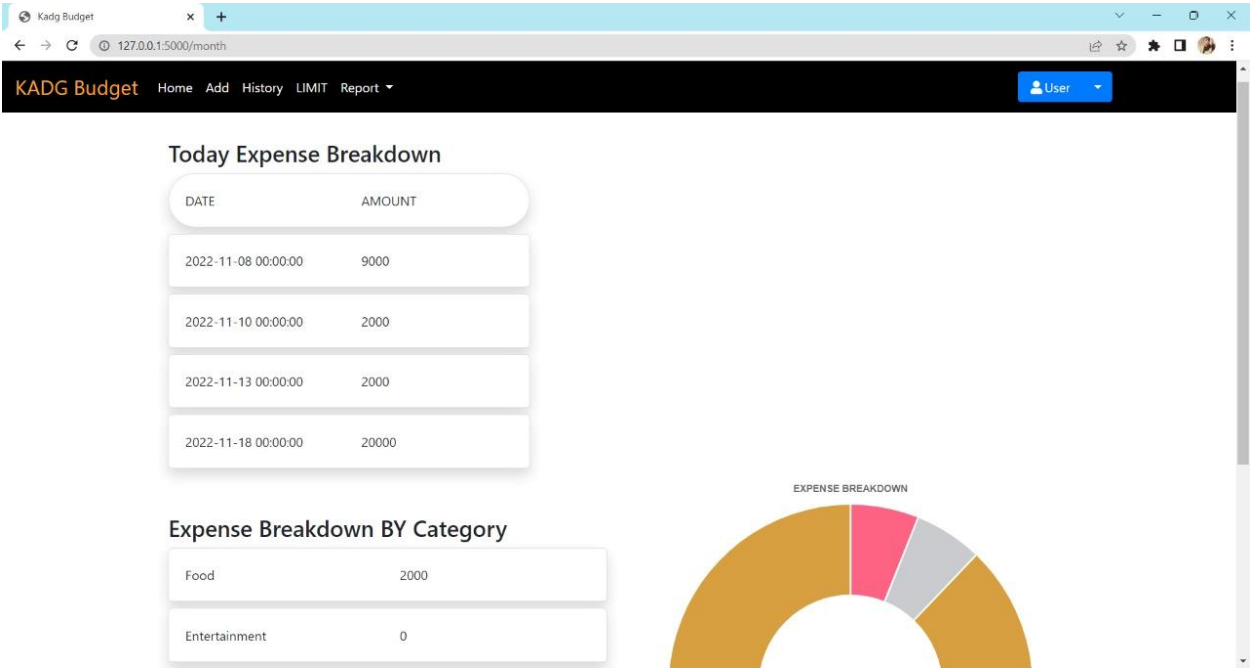
other

Update

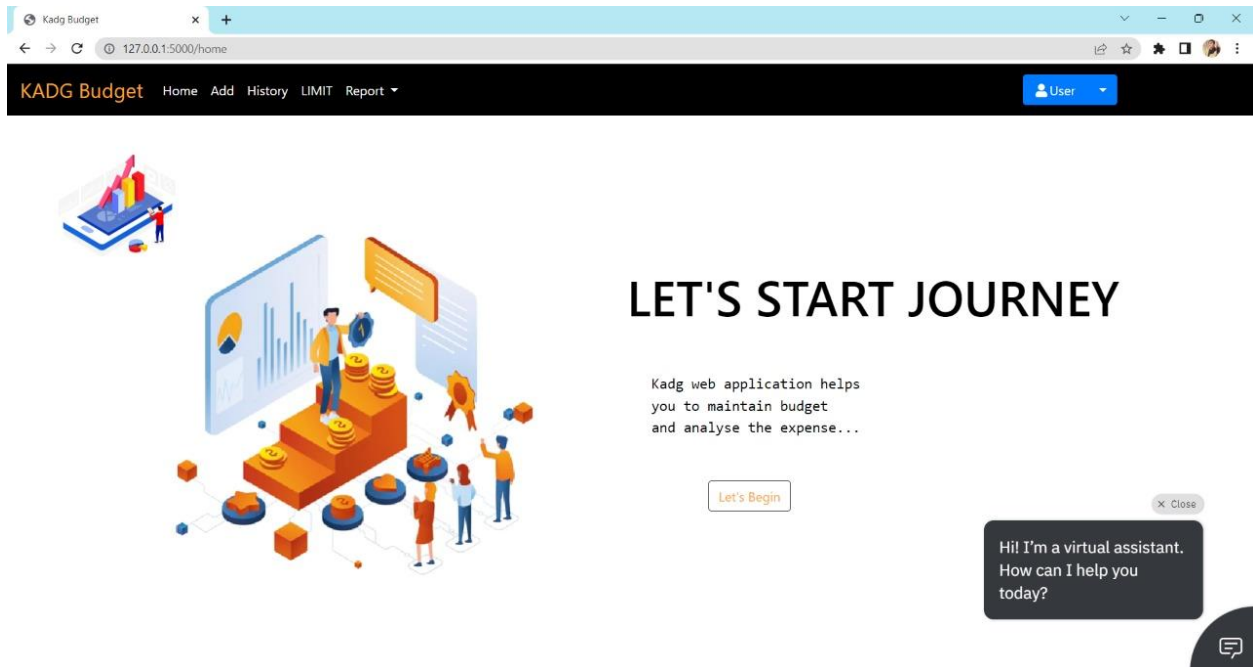
GRAPHICAL REPORT



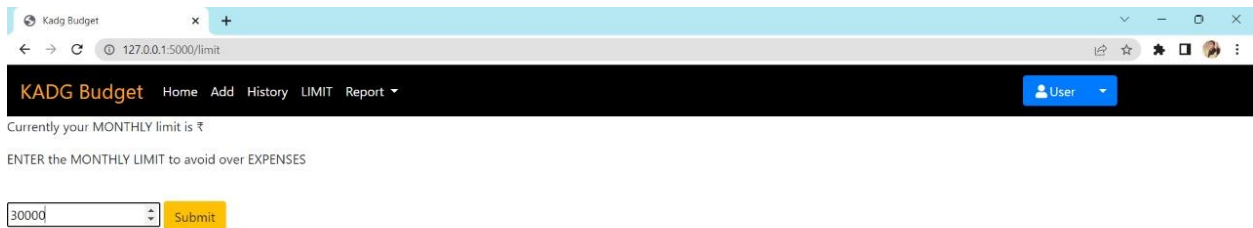
History of expense



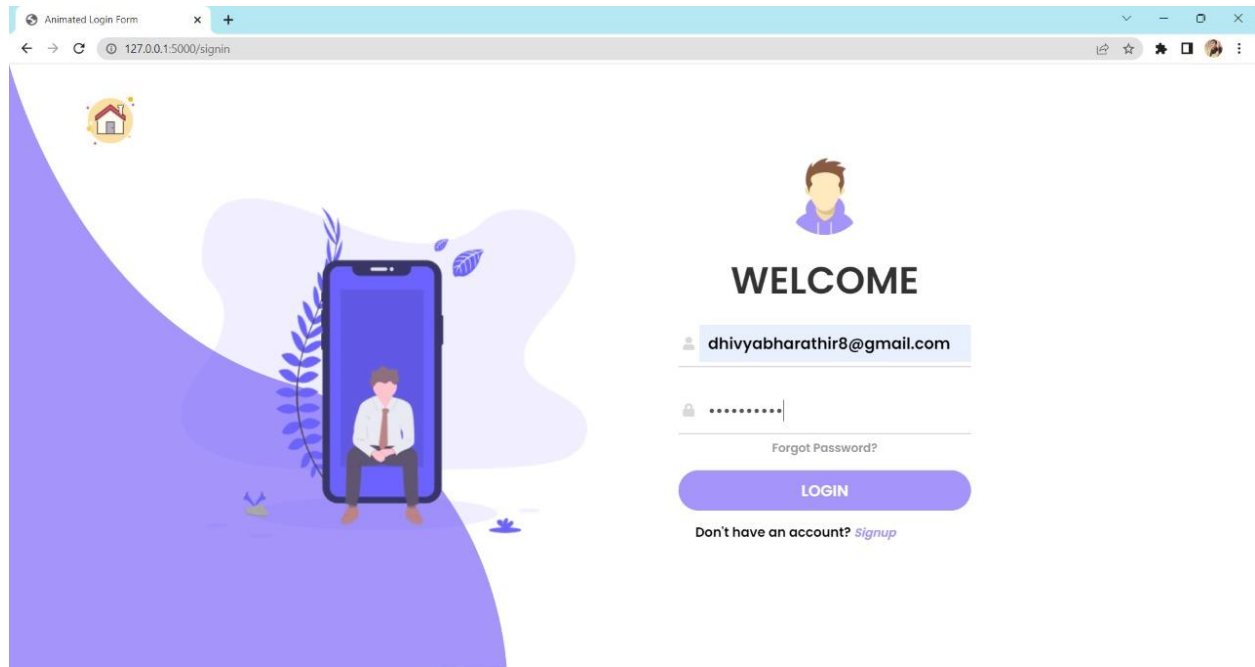
Home Page



SET LIMIT:




Sign in



Sign up

Sign-up

127.0.0.1:5000/signup



Glad to see you

Hello,Friend

Name


Example@gmail.com

Password

☐ I read and agree to [Terms & Conditions](#)

CREATE ACCOUNT

Already have an account? [Sign in](#)



Welcome,Please Fill in the blanks for sign up

10. ADVANTAGES & DISADVANTAGES:

ADVANTAGES:

One of the major pros of tracking spending is always being aware of the state of one's personal finances. Tracking what you spend can help you stick to your budget, not just in a general way, but in each category such as housing, food, transportation and gifts. While a con is that manually tracking all cash that is spent can be irritating as well as time consuming, a pro is that doing this automatically can be quick and simple. Another pro is that many automatic spending tracking software programs are available for free. Having the program on a hand-held device can be a main pro since it can be checked before spending occurs in order to be sure of the available budget.

DISADVANTAGES:

A con with any system used to track spending is that one may start doing it then taper off until it's forgotten about all together. Yet, this is a risk for any new goal such as trying to lose weight or quit smoking. If a person first makes a budget plan, then places money in savings before spending any each new pay period or month, the tracking goal can help. In this way, tracking spending and making sure all receipts are accounted for only needs to be done once or twice a month. Even with constant tracking of one's spending habits, there is no guarantee that financial goals will be met. Although this can be considered to be a con of tracking spending, it could be changed into a pro if one makes up his or her mind to keep trying to properly manage all finances.

11.CONCLUSION:

A comprehensive money management strategy requires clarity and conviction for decision- making. You will need a defined goal and a clear vision for grasping the business and personal finances. That's when an expense tracking app comes into the picture. An expense tracking app is an exclusive suite of services for people who seek to handle their earnings and plan their expenses and savings efficiently. It helps you track all transactions like bills, refunds, payrolls, receipts, taxes, etc., on a daily, weekly, and monthly basis.

12. FUTURE SCOPE:

- Achieve your business goals with a tailored mobile app that perfectly fits your business.
- Scale-up at the pace your business is growing.
- Deliver an outstanding customer experience through additional control over the app.
- Control the security of your business and customer data.
- Open direct marketing channels with no extra costs with methods such as push notifications.
- Boost the productivity of all the processes within the organization.
- Increase efficiency and customer satisfaction with an app aligned to their needs.
- Seamlessly integrate with existing infrastructure.
- Ability to provide valuable insights.
- Optimize sales processes to generate more revenue through enhanced data collection.

- Chats: Equip your expense tracking app with a bot that can understand and answer all user queries and address their needs such as account balance, credit score, etc.

- Prediction: With the help of AI, your mobile app can predict your next purchase, according to your spending behavior. Moreover, it can recommend products and provide unique insights on saving money.

13.APPENDIX:

13.1 SOURCE CODE:

```
from flask import Flask, render_template, request, redirect, session ,url_for

import ibm_db

import re

import sendemail


app = Flask(__name__, template_folder = 'templates')


app.config['SECRET_KEY'] = 'verification'

app.config['MAIL_SERVER'] = 'smtp.sendgrid.net'

app.config['MAIL_PORT'] = 587

app.config['MAIL_USE_TLS'] = True

app.config['MAIL_USERNAME'] = 'apikey'

app.config['MAIL_PASSWORD'] = ''

app.config['MAIL_DEFAULT_SENDER'] = 'abitha.thangadurai@gmail.com'


conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=55fbc997-9266-4331-afd3-888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31929;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UID=spq76013;PWD=lcAFi0FUxoaVlpTn","")


#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():  
    global user_email  
    msg = "  
    if request.method == 'POST':  
        username = request.form['username']
```

```
email = request.form['email']

password = request.form['password']

query = "SELECT * FROM register WHERE email=?;"

stmt = ibm_db.prepare(conn, query)

ibm_db.bind_param(stmt, 1, email)

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:

    query = "INSERT INTO register values(?,?,?);"

    stmt = ibm_db.prepare(conn, query)

    ibm_db.bind_param(stmt, 1, username)

    ibm_db.bind_param(stmt, 2, email)

    ibm_db.bind_param(stmt, 3, password)

    ibm_db.execute(stmt)

    session['loggedin'] = True

    session['id'] = email

    user_email = email
```

```
session['email'] = email  
  
session['username'] = username
```

```
msg = 'You have successfully registered ! Proceed Login Process'  
  
return render_template('login.html', msg = msg)
```

```
else:
```

```
msg = 'PLEASE FILL OUT OF THE FORM'  
  
return render_template('register.html', msg=msg)
```

```
#LOGIN--PAGE
```

```
@app.route("/signin")
```

```
def signin():
```

```
    return render_template('login.html')
```

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

```
def login():
```

```
    global user_email
```

```
    msg = "
```

```
    if request.method == 'POST' :
```

```
        email = request.form['email']
```

```
password = request.form['password']

sql = "SELECT * FROM register WHERE email =? AND password=?"

stmt = ibm_db.prepare(conn, sql)

ibm_db.bind_param(stmt,1,email)

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['EMAIL']

    user_email= account['EMAIL']

    session['email']=account['EMAIL']

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


    return redirect('/home')

else:

    msg = 'Incorrect username / password !'

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

#CHANGE FORGOT PASSWORD

```
@app.route("/forgot")
```

```

def forgot():

    return render_template('forgot.html')


@app.route("/forgotpw", methods=['GET', 'POST'])

def forgotpw():

    msg = "

    if request.method == 'POST' :

        email = request.form['email']

        password = request.form['password']

        query = "SELECT * FROM register WHERE email=?;"

        stmt = ibm_db.prepare(conn, query)

        ibm_db.bind_param(stmt, 1, email)

        ibm_db.execute(stmt)

        account = ibm_db.fetch_assoc(stmt)

        print(account)

        if account:

            query = "UPDATE register SET password = ? WHERE email = ?;"

            stmt = ibm_db.prepare(conn, query)

            ibm_db.bind_param(stmt, 1, password)

            ibm_db.bind_param(stmt, 2, email)

            ibm_db.execute(stmt)

            msg = 'Successfully changed your password ! Proceed Login Process'

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

        else:

```

```
msg = 'PLEASE FILL OUT THE CORRECT DETAILS'

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

```
#ADDING----DATA
```

```
@app.route("/add")
```

```
def adding():
```

```
    return render_template('add.html')
```

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

```
def addexpense():
```

```
    global user_email
```

```
    que = "SELECT * FROM expenses where id = ? ORDER BY 'dates' DESC"
```

```
    stm = ibm_db.prepare(conn, que)
```

```
    ibm_db.bind_param(stm, 1, session['email'])
```

```
    ibm_db.execute(stm)
```

```
    dictionary=ibm_db.fetch_assoc(stm)
```

```
    expense=[]
```

```
    while dictionary != False:
```

```
exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],di
ctionary["PAYMODE"],dictionary["CATEGORY"])
```



```
expense.append(exp)

dictionary = ibm_db.fetch_assoc(stmt)

i=len(expense)+1

idx=str(i)

dates = request.form['date']

expensename = request.form['expensename']

amount = request.form['amount']

paymode = request.form['paymode']

category = request.form['category']

query = "INSERT INTO expenses VALUES (?, ?, ?, ?, ?, ?, ?);"

stmt = ibm_db.prepare(conn, query)

ibm_db.bind_param(stmt, 1, session['email'])

ibm_db.bind_param(stmt, 2, dates)

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.bind_param(stmt, 7, idx)

ibm_db.execute(stmt)

print(dates + " " + expensename + " " + amount + " " + paymode + " " + category)

return redirect("/display")
```

```
#DISPLAY---graph
```

```
@app.route("/display")
```

```
def display():
```

```
    query = "SELECT * FROM expenses where id = ? ;"
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1, session['email'])
```

```
    ibm_db.execute(stmt)
```

```
    dictionary=ibm_db.fetch_assoc(stmt)
```

```
    rexpense=[]
```

```
    while dictionary != False:
```

```
        exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
        rexpense.append(exp)
```

```
        dictionary = ibm_db.fetch_assoc(stmt)
```

```
        que = "SELECT MONTH(dates) as DATES, SUM(amount) as AMOUNT FROM expenses WHERE  
id=? AND YEAR(dates)= YEAR(now()) GROUP BY MONTH(dates);"
```

```
        stm = ibm_db.prepare(conn, que)
```

```
        ibm_db.bind_param(stm, 1,session['email'])
```

```
        ibm_db.execute(stm)
```

```
        dictionary=ibm_db.fetch_assoc(stm)
```

```
        texpense=[]
```

```
        while dictionary != False:
```

```
            exp=(dictionary["DATES"],dictionary["AMOUNT"])
```

```
texpanse.append(exp)
```

```
dictionary = ibm_db.fetch_assoc(stm)
```

```
print(texpanse)
```

```
quer = "SELECT * FROM expenses WHERE id = ? AND YEAR(dates)= YEAR(now());"
```

```
st = ibm_db.prepare(conn, quer)
```

```
ibm_db.bind_param(st, 1,session['email'])
```

```
ibm_db.execute(st)
```

```
dictionary=ibm_db.fetch_assoc(st)
```

```
expense=[]
```

```
while dictionary != False:
```

```
exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
expense.append(exp)
```

```
dictionary = ibm_db.fetch_assoc(st)
```

```
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[3]
```

```
    if x[5] == "food":
```

```
        t_food += x[3]
```

```
    elif x[5] == "entertainment":
```

```
        t_entertainment += x[3]
```

```
    elif x[5] == "business":
```

```
        t_business += x[3]
```

```
    elif x[5] == "rent":
```

```
        t_rent += x[3]
```

```
    elif x[5] == "EMI":
```

```
        t_EMI += x[3]
```

```
    elif x[5] == "other":
```

```
        t_other += x[3]
```

```
print(total)
```

```
print(t_food)
```

```
print(t_entertainment)
```

```
print(t_business)
```

```
print(t_rent)
```

```
print(t_EMI)
```

```
print(t_other)
```

```
qur = "SELECT * FROM expenses WHERE id = ? AND MONTH(dates)= MONTH(now());"
```

```
stt = ibm_db.prepare(conn, qur)
```

```
ibm_db.bind_param(stt, 1, session['email'])
```

```
ibm_db.execute(stt)
```

```
dictionary=ibm_db.fetch_assoc(stt)
```

```
lexpense=[]
```

```
while dictionary != False:
```

```
exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
lexpense.append(exp)
```

```
dictionary = ibm_db.fetch_assoc(stt)
```

```
ttotal=0
```

```
to_food=0
```

```
to_entertainment=0
```

```
to_business=0
```

```
to_rent=0
```

```
to_EMI=0
```

```
to_other=0
```

```
for x in lexpense:

    tttotal += x[3]

    if x[5] == "food":

        to_food += x[3]

    elif x[5] == "entertainment":

        to_entertainment += x[3]

    elif x[5] == "business":

        to_business += x[3]

    elif x[5] == "rent":

        to_rent += x[3]

    elif x[5] == "EMI":

        to_EMI += x[3]

    elif x[5] == "other":

        to_other += x[3]

print(tttotal)
```

```

qy = "SELECT max(IDX) as IDX FROM limits where id=?;"

smt = ibm_db.prepare(conn, qy)

ibm_db.bind_param(smt, 1, session['email'])

ibm_db.execute(smt)

dictionary = ibm_db.fetch_assoc(smt)

uexpense=[]

while dictionary != False:

    exp=(dictionary["IDX"])

    uexpense.append(exp)

    dictionary = ibm_db.fetch_assoc(smt)

k=uexpense[0]

qu = "SELECT NUMBER FROM limits where id=? and idx=?"

sm = ibm_db.prepare(conn, qu)

ibm_db.bind_param(sm, 1, session['email'])

ibm_db.bind_param(sm, 2, k)

ibm_db.execute(sm)

dictionary = ibm_db.fetch_assoc(sm)

fexpense=[]

while dictionary != False:

    exp=(dictionary["NUMBER"])

    fexpense.append(exp)

    dictionary = ibm_db.fetch_assoc(stmt)

if len(fexpense) <= 0:

```

```

        print("Enter the limit First")

    else:

        if ttotal > fexpense[0]:

            m=sendemail.sendgridmail(session["email"])

            print(m)

        else: print("Error")

    return render_template("display.html",rexpense=rexpense, 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 )


#delete---the--data


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

def delete(idx):

    query = "DELETE FROM expenses WHERE id=? and idx=?;"

    stmt = ibm_db.prepare(conn, query)

    ibm_db.bind_param(stmt, 1, session["email"])

    ibm_db.bind_param(stmt, 2, idx)

    ibm_db.execute(stmt)

    print('deleted successfully')

    return render_template("display.html")

```



```
#UPDATE---DATA
```

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

```
def edit(id):
```

```
    query = "SELECT * FROM expenses WHERE id=? and idx=?"
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1, session['email'])
```

```
    ibm_db.bind_param(stmt, 2, id)
```

```
    ibm_db.execute(stmt)
```

```
    dictionary=ibm_db.fetch_assoc(stmt)
```

```
    expense=[]
```

```
    while dictionary != False:
```

```
        exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
        expense.append(exp)
```

```
        dictionary = ibm_db.fetch_assoc(stmt)
```

```
    print(expense)
```

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

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

def update(id):

    if request.method == 'POST' :

        dates = request.form['date']

        expensename = request.form['expensename']

        amount = request.form['amount']

        paymode = request.form['paymode']

        category = request.form['category']

        query = "UPDATE expenses SET dates = ? , expensename = ? , amount = ? , paymode = ? , category
= ? WHERE id = ? and idx=?;"

        stmt = ibm_db.prepare(conn, query)

        ibm_db.bind_param(stmt, 1, dates)

        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, session['email'])

        ibm_db.bind_param(stmt, 7, id)

        ibm_db.execute(stmt)

        print('successfully updated')

        return redirect("/display")
```

```
#limit

@app.route("/limit" )

def limit():

    return render_template('limit.html')


@app.route("/limitnum" , methods = ['POST' ])

def limitnum():

    que = "SELECT * FROM limits where id = ? ;"

    stm = ibm_db.prepare(conn, que)

    ibm_db.bind_param(stm, 1, session['email'])

    ibm_db.execute(stm)

    if request.method == "POST":

        dictionary=ibm_db.fetch_assoc(stm)

        expense=[]

        while dictionary != False:

            exp=(dictionary['ID'],dictionary['NUMBER'],dictionary['IDX'])

            expense.append(exp)

            dictionary = ibm_db.fetch_assoc(stm)

        i=len(expense)+1
```

```
idx=str(i)

number= request.form['number']

query = "INSERT INTO limits VALUES(?,?,?)"

stmt = ibm_db.prepare(conn, query)

ibm_db.bind_param(stmt, 1, session['email'])

ibm_db.bind_param(stmt, 2, number)

ibm_db.bind_param(stmt, 3, idx)

ibm_db.execute(stmt)

return redirect('/limitn')
```

```
@app.route("/limitn")
```

```
def limitn():
```

```
    query = "SELECT max(IDX) as IDX FROM limits where id=?;"
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1, session['email'])
```

```
    ibm_db.execute(stmt)
```

```
    dictionary = ibm_db.fetch_assoc(stmt)
```

```
    expense=[]
```

```
    while dictionary != False:
```

```
        exp=(dictionary["IDX"])
```

```
        expense.append(exp)
```

```
        dictionary = ibm_db.fetch_assoc(stmt)
```

```
    k=expense[0]
```

```
que = "SELECT NUMBER FROM limits where id=? and idx=?"
```

```
stmt = ibm_db.prepare(conn, que)
```

```
ibm_db.bind_param(stmt, 1, session['email'])
```

```
ibm_db.bind_param(stmt, 2, k)
```

```
ibm_db.execute(stmt)
```

```
dictionary = ibm_db.fetch_assoc(stmt)
```

```
texpanse=[]
```

```
while dictionary != False:
```

```
    exp=(dictionary["NUMBER"])
```

```
    texpanse.append(exp)
```

```
    dictionary = ibm_db.fetch_assoc(stmt)
```

```
s=texpense[0]
```

```
return render_template("limit.html" , y= s)
```

#REPORT

```
@app.route("/today")
```

```
def today():
```

```
    query = "SELECT dates, amount FROM expenses  WHERE id = ? AND DATE(dates) =  
DATE(NOW()); "
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1, str(session['email']))
```

```
    ibm_db.execute(stmt)
```

```
    dictionary=ibm_db.fetch_assoc(stmt)
```

```
texpense=[]  
  
while dictionary != False:  
  
    exp=(dictionary["DATES"],dictionary["AMOUNT"])  
  
    texpense.append(exp)  
  
    dictionary = ibm_db.fetch_assoc(stmt)  
  
print(texpense)
```

```
query = "SELECT * FROM expenses WHERE id = ? AND DATE(dates) = DATE(NOW())"  
  
stmt = ibm_db.prepare(conn, query)  
  
ibm_db.bind_param(stmt, 1, session['email'])  
  
ibm_db.execute(stmt)  
  
dictionary=ibm_db.fetch_assoc(stmt)  
  
expense=[]  
  
while dictionary != False:  
  
    exp=(dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"])  
  
    expense.append(exp)  
  
    dictionary = ibm_db.fetch_assoc(stmt)
```

```
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[0]
```

```
    if x[2] == "food":
```

```
        t_food += x[0]
```

```
    elif x[2] == "entertainment":
```

```
        t_entertainment += x[0]
```

```
    elif x[2] == "business":
```

```
        t_business += x[0]
```

```
    elif x[2] == "rent":
```

```
        t_rent += x[0]
```

```
    elif x[2] == "EMI":
```

```
        t_EMI += x[0]
```

```
    elif x[2] == "other":
```

```
        t_other += x[0]
```

```
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():
```

```
    query = "SELECT dates, SUM(amount) as AMOUNT FROM expenses WHERE id= ? AND  
MONTH(dates)= MONTH(now()) GROUP BY dates ORDER BY dates;"
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1, str(session['email']))
```

```
    ibm_db.execute(stmt)
```

```
    dictionary=ibm_db.fetch_assoc(stmt)
```

```
    texpanse=[]
```

```
    while dictionary != False:
```



```
exp=(dictionary["DATES"],dictionary["AMOUNT"])

texpanse.append(exp)

dictionary = ibm_db.fetch_assoc(stmt)

print(texpanse)
```

```
query = "SELECT * FROM expenses WHERE id = ? AND MONTH(dates)= MONTH(now());"

stmt = ibm_db.prepare(conn, query)

ibm_db.bind_param(stmt, 1, session['email'])

ibm_db.execute(stmt)

dictionary=ibm_db.fetch_assoc(stmt)

expense=[]

while dictionary != False:
```

```
exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],di
ctionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
expense.append(exp)

dictionary = ibm_db.fetch_assoc(stmt)
```

```
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[3]
```

```
    if x[5] == "food":
```

```
        t_food += x[3]
```

```
    elif x[5] == "entertainment":
```

```
        t_entertainment += x[3]
```

```
    elif x[5] == "business":
```

```
        t_business += x[3]
```

```
    elif x[5] == "rent":
```

```
        t_rent += x[3]
```

```
    elif x[5] == "EMI":
```

```
        t_EMI += x[3]
```

```
    elif x[5] == "other":
```

```
        t_other += x[3]
```

```
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():
```

```
    query = "SELECT MONTH(dates) as DATES, SUM(amount) as AMOUNT FROM expenses  
WHERE id=? AND YEAR(dates)= YEAR(now()) GROUP BY MONTH(dates);"
```

```
    stmt = ibm_db.prepare(conn, query)
```

```
    ibm_db.bind_param(stmt, 1,session['email'])
```

```
    ibm_db.execute(stmt)
```

```
    dictionary=ibm_db.fetch_assoc(stmt)
```

```
    texpanse=[]
```

```
    while dictionary != False:
```

```
        exp=(dictionary["DATES"],dictionary["AMOUNT"])
```

```
        texpanse.append(exp)
```

```
dictionary = ibm_db.fetch_assoc(stmt)

print(texpanse)
```

```
query = "SELECT * FROM expenses WHERE id = ? AND YEAR(dates)= YEAR(now());"
```

```
stmt = ibm_db.prepare(conn, query)
```

```
ibm_db.bind_param(stmt, 1,session['email'])
```

```
ibm_db.execute(stmt)
```

```
dictionary=ibm_db.fetch_assoc(stmt)
```

```
expense=[]
```

```
while dictionary != False:
```

```
exp=(dictionary["ID"],dictionary["DATES"],dictionary["EXPENSENAME"],dictionary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"],dictionary["IDX"])
```

```
expense.append(exp)
```

```
dictionary = ibm_db.fetch_assoc(stmt)
```

```
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[3]

    if x[5] == "food":

        t_food += x[3]

    elif x[5] == "entertainment":

        t_entertainment += x[3]

    elif x[5] == "business":

        t_business += x[3]

    elif x[5] == "rent":

        t_rent += x[3]

    elif x[5] == "EMI":

        t_EMI += x[3]

    elif x[5] == "other":

        t_other += x[3]

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)

    return render_template('home.html')
```

```
if __name__ == "__main__":
```

```
app.run(host='0.0.0.0',port=5000)
```

13.2 Github link:

<https://github.com/IBM-EPBL/IBM-Project-36519-1660295673>

13.3 Project demo link:

<https://youtu.be/K7qQTLXqZB0>