PERSONAL EXPENSE TRACKER APPLICATION

IBM-Project-2110-1658456268

Project Details

Team ID	PNT2022TMID42215
Project Name	Personal Expense Tracker Application
Batch	B1-1M3E

Team Members

Team Member	Name	Register No
Team Lead	Nandhini K	710019104023
Team Member 1	Kaviya G	710019104018
Team Member 2	Jai Jansi A	710019104301
Team Member 3	Renishta T R	710019104701
Team Member 4	Devadharsini S	710019104704

1. INTRODUCTION

1.1Project Overview:

Personal Expense Tracker is a window store application to be designed for tracking daily personal and business expense and income related transaction. The main features include managing daily transaction, report generation, accounts management, set limit for expenses, etc. This application helps you to track all the expenses and incomes of a users. It keeps analyzing the incomes and expenses of a user to generate reports in daily basis. It is flexible and adaptive application to save money and also helps to achieve the goal.

1.2 Purpose:

This app focus us to think about the money.It can reveal the spending issues. It will build the disciple and organisation also helps to track the expense in organized way. Personal Expense Tracker Application makes life easier by helping to manage all the expenditure efficiently. This project will request the users to add their expenses and in view of their costs ,wallet status will be refreshed which will be noticeable to the client. 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. In this way the user can save the money from unnecessary expenditure

2.LITERATURE SURVEY

2.1 Existing Problem:

In today's busy and expensive life we are in a great rush to make money. But at the end of the day we broke off. As we are unknowingly spending money on little and unwanted things. this problem can arise due to low salary, invariably it is due to poor money management skills.. So, we have come over with the ideas to track our earnings Using a daily expense tracker can help you keep track of how much you spend every day and on what. At the end of the month, we will have a clear picture of where our money had spent.

2.2 References:

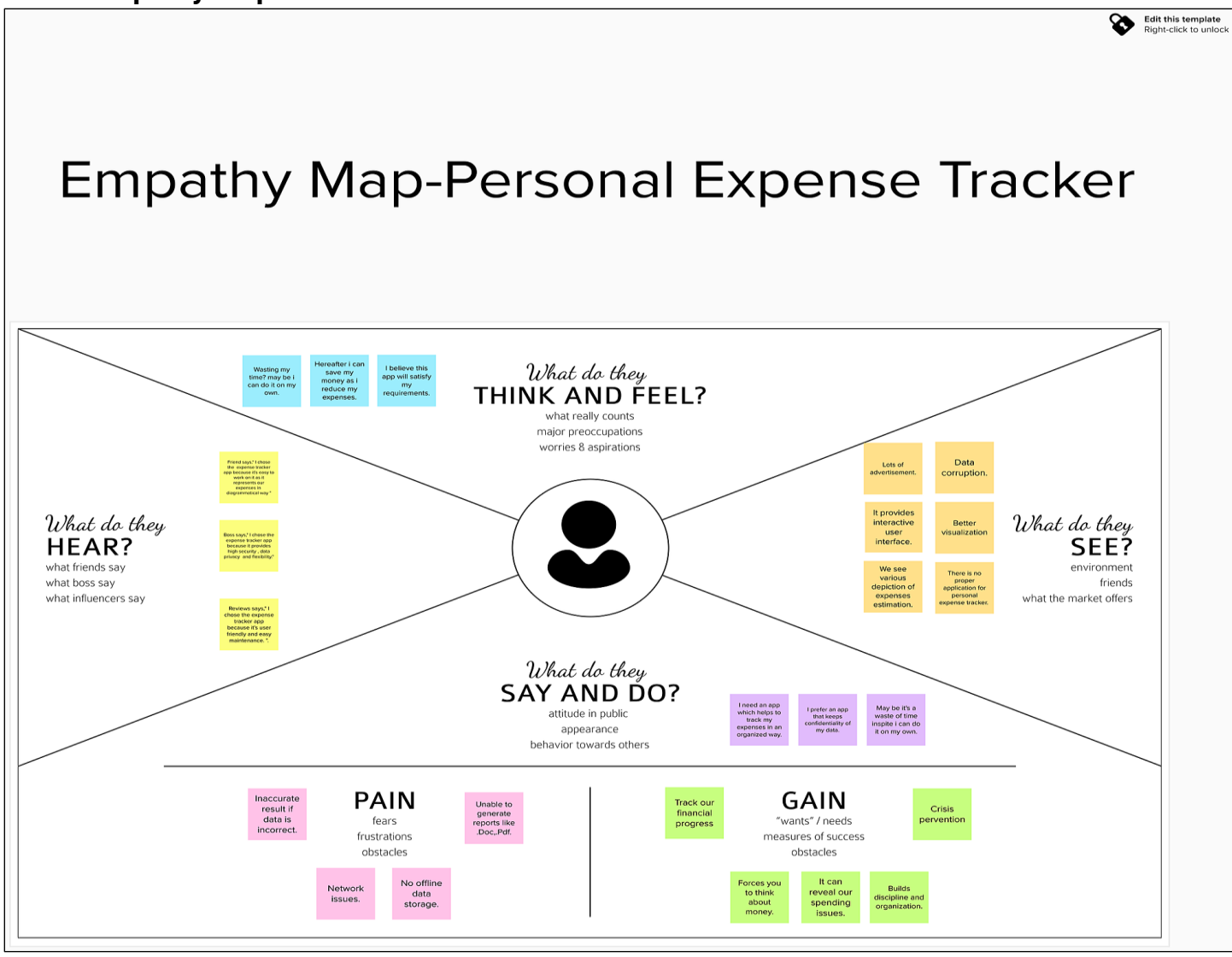
https://www.researchgate.net/publication/347972162_Expense_Manager_Application

2.3 Problem Statement Definition:

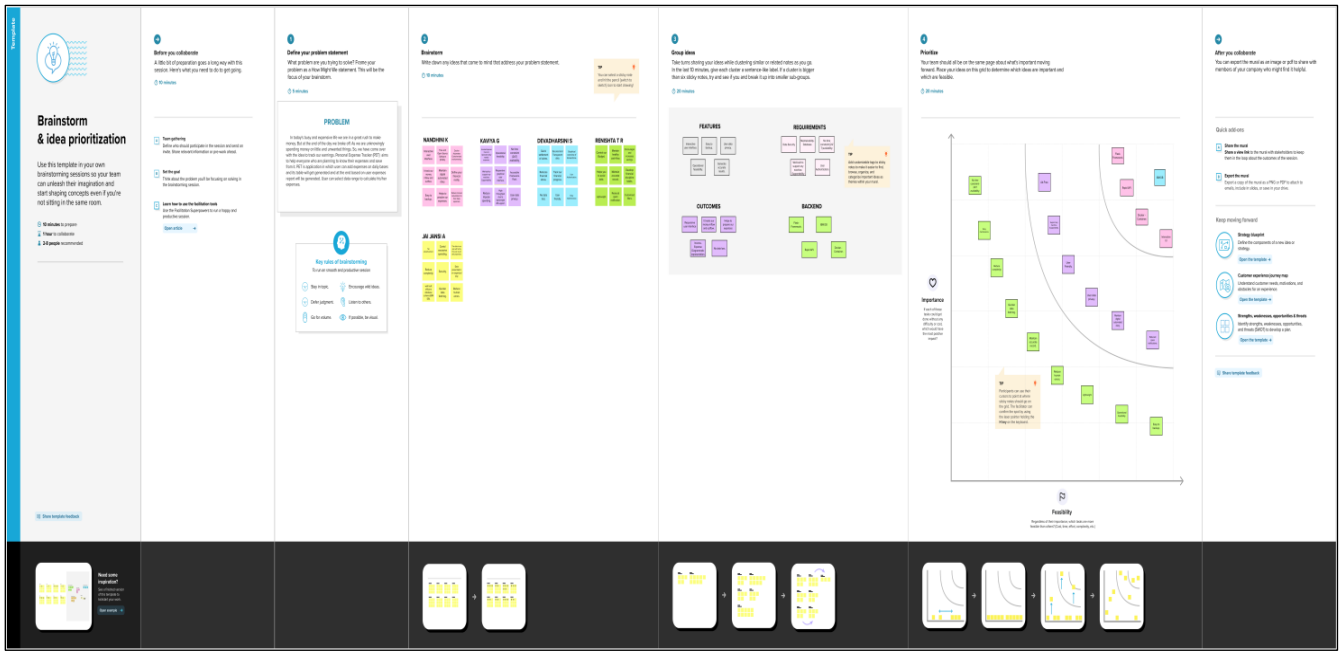
Personal Expense Tracker (PET) aims to help everyone who are planning to know their expenses and save from it. PET is application in which user can add expenses on daily bases and its table will get generated and at the end based on user expenses report wil be generated. User can select data range to calculate his/her expenses. This app entails all the financial decisions and activities,This app makes life easier by helping you to manage expenditure efficiently. A personal expense app will not only help you with budgeting and accounting but also give you helpful insights about money management.

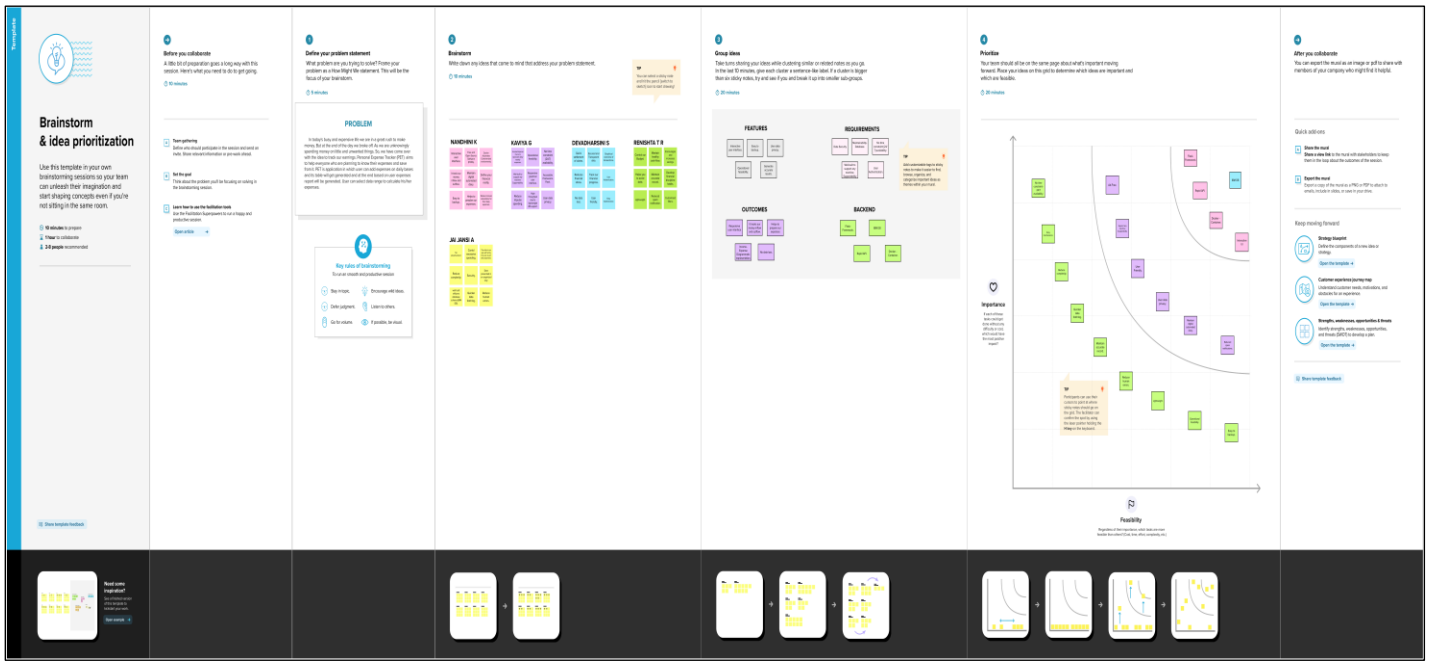
3. IDEATION & PROPOSED SOLUTION:

3.1 Empathy Map Canva



3.2 Ideation & Brainstorming:





3.3 Proposed Solution:

S.NO	PARAMETER	DESCRIPTION
1.	Problem Statement (Problem to be solved)	Every earning people are mostly obsessedat the end of the month as the they cannot remember where all of their money has gone when they have spent and ultimately have to sustain in little money minimizing their essential needs. 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 and notify them if they are going to have money shortage. To do so a person has to keep a log in a diary or in a computer, alsoall thecalculation needs to be doneby theuser 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 tillthe end of the month.
2.	Idea / Solution description	1. Dashboard Panel - The system will authenticate the user and then display panel basedon the particular identifieduser. 2. Add Bill – The system allows the user to add billdetails based on the user'sneed totrackthe type of expenses. 3. Expense Planner – The system graphically represents the current monthfigure based on the user'scurrent month expenses and user's own budget share. 4. Expense Tracker– The systemgraphically represents the yearly expense numbers in the formof report. 5. Add Notes– The system allows userstoadd notesto their expenses 6. Category – The system allows users to add

		categories of their expenses. 7. Calender – The system allows users to add date to their expenses.
3.	Novelty / Uniqueness	1. The systemis well built to supportanymachine – Supportability. 2. Each data record is stored on well-built efficient databaseschema (IBM database).There is no risk of data loss. The internal evaluation of data is well coded.
4.	Social Impact / CustomerSatisfaction	1. The aim of this application is to providea solution for users on how to manage finances in any circumstance by keeping track of their expenseseveryday. Ultimately, this contributes to societalwell-being. 2. Mental budgeting leads people to overconsume some goods and under consume others. Because budgets are setbefore consumption

		opportunities arise, they sometimes over estimate or under estimate the money required for a particular amount.Thus, the people don't know where their money is going. The idea of an expense tracking tool where with just a few inputs you can make yourself organized and make your life a bit easier in the long run.
5.	Business Model	Waterfall model is used for the project because all the requirements are clear asthis projectis not dealing with the clients and hence beforehand planning can be made about how to carry out each phase of development.
6.	Scalability of the Solution	1. This ideal practice guarantees that the expenses tracked are accurately and in a timely manner. 2.It helps in making financial awareness and

		improving moneymanagement, tracking your expenditures ensuresyou to achieve your financial target

3.4 Problem Solution fit:

Project Title:Personal expense Tracker | **Project Design Phase-I - Solution Fit Template** | **Team ID:** PNT2022TMID2215

1. CUSTOMER SEGMENT(S) CS

- Working Individuals
- Students
- Budget Conscious Consumers.

5.AVAILABLE SOLUTIONS

- Expense Diary or Excel Sheet

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

CONS: Inconvenient,takes a lot of time.

6. CUSTOMER CONSTRAINTS AS

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

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

2. JOBS-TO-BE-DONE / PROBLEMS

- To keep track of money lent or borrowed.
- To keep track of money inflow and outflow.
- Alert when a threshold limit is reached.

9. PROBLEM ROOT CAUSE

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

7. BEHAVIOUR

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

3. TRIGGERS

- Excessive spending
- No money in case of emergency.

4. EMOTIONS: BEFORE / AFTER

BEFORE

- Anxious,Confused,Fear

AFTER

- Confident,Composed,Calm

10. YOUR SOLUTION

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

8. CHANNELS OF BEHAVIOUR CH

ONLINE

- Maintain excel sheets and use visualizing tools.

OFFLINE

- Maintain an expense diary.

4. REQUIREMENT ANALYSIS

4.1 Functional requirement:

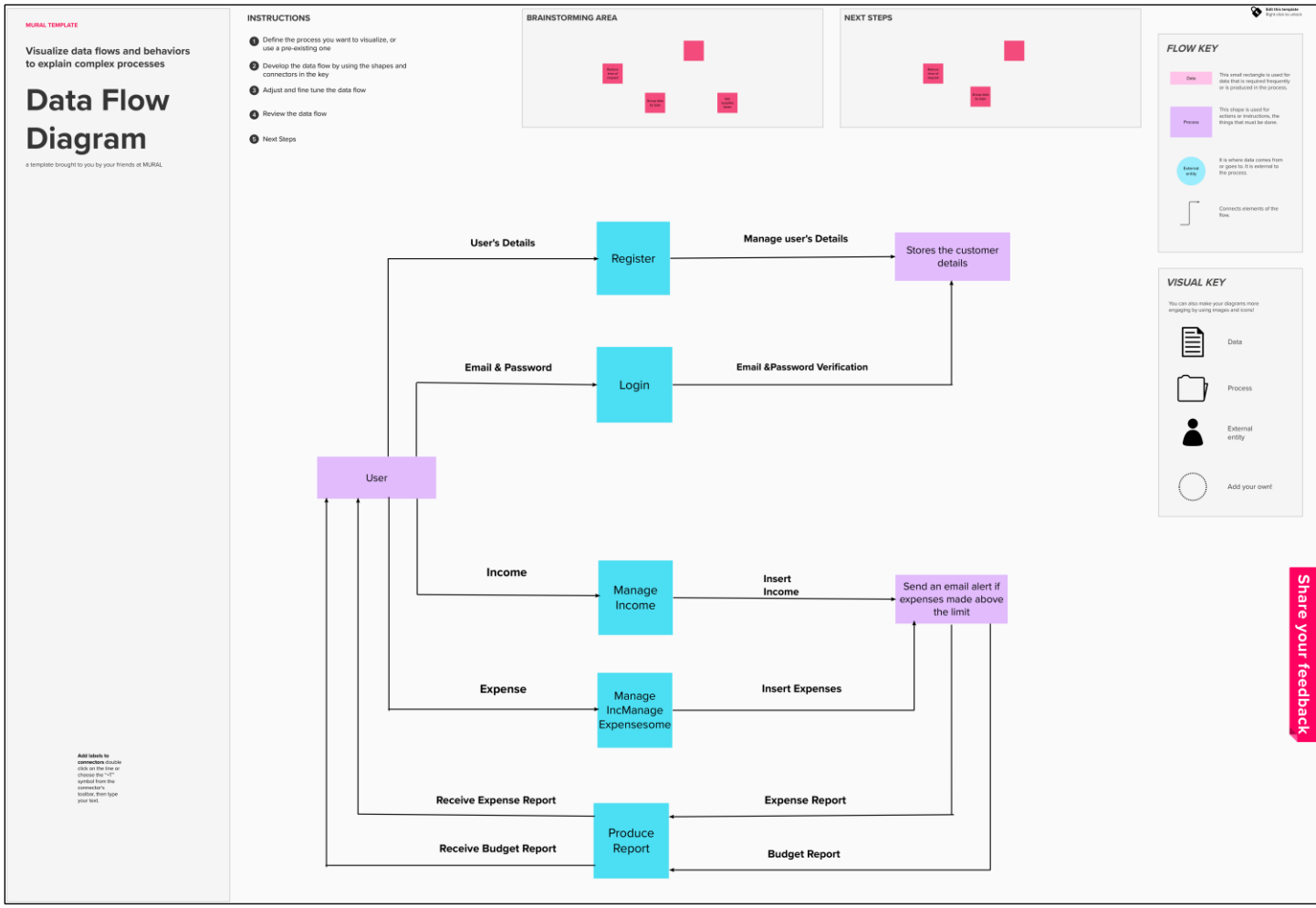
FR No.	Functional Requirements (Epic)	Sub Requirement (Story/ Sub- Task)
FR-1	User Registration	Registration through Application Registration through E-mail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User monthly expense tentative data	Data to be registered in the app
FR-4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

4.2 Non-Functional requirements:

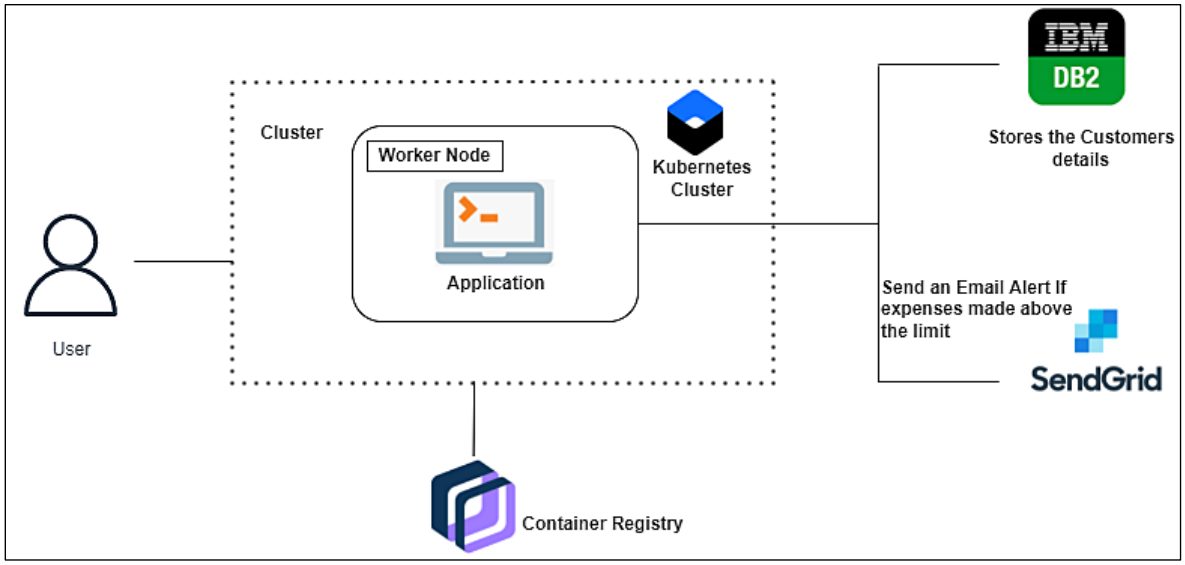
NFR No.	Non-FunctionalRequirements	Description
NFR-1	Usability	Effectiveness, efficiency and overall satisfaction of the user while interacting with our application.
NFR-2	Security	Authentication, authorization, encryption of the application
NFR-3	Reliability	Probability of failure-free operations in 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	Without near 100% availability, application reliability and the user satisfaction will affect the solution.
NFR-6	Scalability	Capacity of the application to handle growth, especially in handling more users.

5. PROJECT DESIGN

5.1 Data Flow Diagrams:



5.2 Solution & Technical Architecture:



5.3 User Stories:

Table-1: Components & Technologies:

S.NO	Component	Description	Technology
1.	User Interface	The user can Interactwith the application with use of Chatbot	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	The application contains the sign in/sign up where the user will login into the main dashboard	Java / Python
3.	Application Logic-2	Dashboard contains the fields like Add income, Add Expenses, Save Money	IBM Watson STT service
4.	Application Logic-3	The user will get the expense report in the graph form and also get alerts if the expense limit exceeds	IBM Watson Assistant,SendGrid
5.	Database	The Income and Expense data are stored in the MySQL database	MySQL, NoSQL, etc.
6.	Cloud Database	With use of Database Service on Cloud, the User data are stored in a well secured Manner	IBM DB2, IBM Cloudant etc.

7.	File Storage	IBM Block Storage used to store the Financial data of the user	IBM Block Storage or Other Storage Service or Local Filesystem

Table-2: Application Characteristics:

S.No.	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask Framework in Python is used to implement this Application	Python-Flask
2.	Security Implementations	This Application Provides high security to the user Financial data. It can be done by using the Container Registry in IBM cloud	Container Registry, Kubernetes Cluster
3.	Scalable Architecture	Expense Tracker is a life time access supplication. It's demand will increase when the user's income are high	Container Registry, Kubernetes Cluster
4.	Availability	This application will be available to the user at any part of time	Container Registry, Kubernetes Cluster
5.	Performance	The performance will be high because there will be no network	Kubernetes Cluster

		traffics in the application	

6.PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation:

· SPRINT PLAN

• ANALYZE THE PROBLEM

• PREPARE an ABSTRACT, PROBLEM STATEMENT

• LIST A REQUIRED OBJECT NEEDED

• CREATE A PROGRAM CODE AND RUN IT

• MAKE A PROTOTYPE TO IMPLEMENT

• TEST WITH THE CREATED CODE AND CHECK THE DESIGNED PROTOTYPE

6.2 Sprint Delivery Schedule

Sprint	Functional Requirements	User Story Number	User Story/ Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a User, I need to register user id and password for accessing the application.	10	High	Kaviya G Nandhini K
			As a user, I need to login with user id and password to	10	High	Devadharsini S Jai Jansi A Renishta TR

			get into the application..			
			As a User I can add the day to day expense that I spend on to the application..	10	High	Devadharsini S Jai Jansi A
			As a User, I can edit and delete expenses as I wish..	10	Medium	Devadharsini S Jai Jansi A
			As a User I can view my expense in a graph of overview of the expense I spend.	10	High	Renishta TR Nandhini K
			As a User I will get mail when I cross my estimated budget.	10	High	Renishta TR Kaviya G
			Deployment of Application.	20	High	Kaviya G Nandhini K

7.CODING AND SOLUTIONING

7.1 Feature 1

Tracking your spending is often the first step in getting your finances in order. By understanding what you spend money on and how much you spend, you can see exactly where your cash is going and areas where you can cut back.

It’s easy to make this part of your everyday routine thanks to expense tracker apps that help you manage your money on the go. These apps certainly overlap with budgeting apps, but while the latter provides a big-picture view of your finances, expense tracker apps put more of an emphasis on your spending. These apps usually categorize your expenses and help you get a good idea of your purchasing behavior.

CODE:

home.html

```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <link rel="stylesheet" href="..\static\css\home.css">

  <title>TRACK FIN</title>

</head>

<body>

  <!-- Header -->

  <section id="header">

    <div class="header container">

      <div class="nav-bar">

        <div class="brand">

          <a href="#hero">

            <h1><span>T</span>RACK <span>F</span>IN</h1>
```

```
        </a>
      </div>
      <div class="nav-list">
        <div class="hamburger">
          <div class="bar"></div>
        </div>
        <ul>
          <li><a href="#hero" data-after="Home">Home</a></li>
          <LI><a href="/signin" data-after="Login">-Login-</a></LI>
        </ul>
      </div<>
    </div>
  </div>
 </section>
 <!-- End Header -->
<!-- Hero Section  -->
 <section id="hero">
  <div class="hero container">
    <div>
      <h1>Welcome To <span></span></h1>
      <h1>Personal Expense Tracker Application <span></span></h1>
      <a href="/signup" type="button" class="cta">Register here</a>
```

```
      </div>
    </div>
  </section>

  <!-- Footer -->
  <section id="footer">
    <div class="footer container">
      <div class="brand">
        <h1><span>T</span>RACK <span>F</span>IN</h1>
      </div>
      <h2>Track Your Expenses To Stick To Your Budget!</h2>
   <!--End Footer -->
  <script src="..\static\js\home.js"></script>
</body>
</html>
```

homepage.html

```
{% extends 'base.html' %}
{% block body %}
<style>
body{
    /* background-image: url('../static/images/hb2.png'); */
    background-repeat: no-repeat;
    background-attachment: fixed;
```

```
    overflow: hidden;
}
img{
    border-radius: 14px;
}
    H1 {
        position: relative;
        right: -650PX;
        top: -350PX;
        color:#000000;
        font-size: 60px;
}
p{
position: relative;
right: -680px;
top: -300px;
font-family:monospace;
font-size: 18px;
color: #000000;
}
span{
    position: relative;
```

```
right: -650px;
top: -310px;
color: rgb(221, 67, 157);
}
 .ccc {
        position: relative;
        top:80px;
        left:-100px;
        bottom: 100px;
    }
</style>
    <div id=aa class="container">
    <div class="ccc">
        <!-- <img height="450px" width="450px" style="position: relative; top: 50px; left: 90PX;" src="../static/images/homepage1.png"> -->
        <video height="450px" width="450px" style="position: relative; top: 50px; left: 90PX;" autoplay loop>
            <source src="/static/images/hbg1.mp4" type="video/mp4">
            <source src="/static/images/hbg1.ogg" type="video/ogg">
          Your browser does not support the video tag.
          </video>
        <h1>LET'S YOU EARN BETTER!</h1>
        <P>Track Fin web application let you <br>collect and categorise your purchases so you can spot areas<br>
```

```
            where you can save money..</P>

            <quotes>

              It's Your Money Own It!

            </quotes>

        <img height="125px" width="13%" style="position: relative; top: -665PX;
left: -50PX;" src="../static/images/stock3.png">

    </div>

    <span  class="btn btn-outline-dark"><a style="color: #A594F9;"
href="/add">Get Started?</a></span>

</div>

{% endblock %}
```

app.py

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

import ibm_db

import reapp = Flask(_name_)

hostname = 'fbd88901-ebdb-4a4f-a32e-
9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;'

uid = 'mfb94822'

pwd = 'PXYtupb5Ky3RBSDk'

driver = "{IBM DB2 ODBC DRIVER}"

db_name = 'Bludb'

port = '32731'

protocol = 'TCPIP'

cert = "certi.crt"
```

```
dsn = (
    "DATABASE ={0};"
    "HOSTNAME ={1};"
    "PORT ={2};"
    "UID ={3};"
    "SECURITY=SSL;"
    "PROTOCOL={4};"
    "PWD ={6};"
).format(db_name, hostname, port, uid, protocol, cert, pwd)
connection = ibm_db.connect(dsn, "", "")
app.secret_key = 'a'
#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(connection, 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(connection, 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(connection, 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(connection, 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(connection, 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(connection, 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"],dictio
nary["AMOUNT"],dictionary["PAYMODE"],dictionary["CATEGORY"])
        expense.append(exp)
        dictionary = ibm_db.fetch_assoc(stm)
    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(connection, 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(connection, 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"],dictio
nary["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(connection, 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"])

        texpense.append(exp)

        dictionary = ibm_db.fetch_assoc(stm)
```

```
    print(texpense)

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

    st = ibm_db.prepare(connection, 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"],dictio
nary["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(connection, 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"],dictio
nary["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 expense:

          ttotal += 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(total)
qy = "SELECT max(IDX) as IDX FROM limits where id=?;"
    smt = ibm_db.prepare(connection, 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(connection, 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(connection, 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(connection, 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"],dictio
nary["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(connection, 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")
@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(connection, 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(connection, 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(connection, 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(connection, 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)
```

```
    texpense=[]
    while dictionary != False:
        exp=(dictionary["NUMBER"])
        texpense.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(connection, 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(connection, 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", 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():

    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(connection, 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 MONTH(dates)=
MONTH(now());"

    stmt = ibm_db.prepare(connection, 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"],dictio
```

```
nary["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", 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():
    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(connection, query)
    ibm_db.bind_param(stmt, 1,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 YEAR(dates)=
YEAR(now());"
    stmt = ibm_db.prepare(connection, 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"],dictio
nary["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", 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)

  return render_template('home.html')

if _name_ == "_main_":

   app.run(debug=True)
```

8.TESTING

8.1 TEST CASES

Test case ID	Feature type	Component	Text scenario	Pre-requisite	Steps to Execute	Test data	Expected result	Actual result	Statu
LoginPage_TC_001	Functional	Home page	verify user is able to see the Login/signup popup	None	1.Go to website 2.Home page appears	Username :test Password :123456	Login/sigup popup should display	Working as Expected	Pass
LoginPage_TC_002	UI	Home page	Verify the UI element in Login/sig	Home	1.Go to website	Username :test Password :123456	Application should show below UI elements:	Working as Expected	Pass

			nup popup		2.Enter details and click login		a.email text box b.password text box		
LoginPage_TC_003	Functional	Username and Password	verify user to login to the application	Username and password	1.Go to website 2.Enter details and click login	Username :test Password :123456	User should navigate to user account homepage	Working as Expected	Pass
LoginPage_TC_004	Functional	Username and Password	verify user to login to the application	Username and password	1.Go to website 2.Enter details and click login	Username :test Password :123456	Application should show incorrect email.	Working as Expected	Pass
LoginPage_TC_004	Functional	Login First	verify user to login to the application	Login First	1.Go to website 2.Enter details and click login	Username :test Password :123456	Application should show incorrect email.	Working as Expected	Pass
LoginPage_TC_005	Functional	Login First	verify user to login to the application	Login First	1.Go to website 2.Enter details and click login	Username :test Password :123456	Application should show incorrect email.	Working as Expected	Pass
AddExpensepage_005_TC	Functional	Have some expense to add	verify whether user is able to add expense or not	Have some expense to add	Add date,expense name and other details	add rent=6000	Application adds expenses	Working as Expected	Pass

8.2 USER ACCEPTANCE TESTING

Purpose of Document:

The purpose of the document to breifly explain the test coverage and open issues of the skills / budget application project at that time release to User Acceptance Testing(UAT).

1.Defeat Analysis:

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	5	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	0	0	0
Skipped	0	0	0	0	0
Won't fix	0	5	2	1	8
Totals	24	14	13	26	75

2.Test Case Analysis:

Sectiom	Total Cases	Not tested	Fail	Pass
Print Engine	7	0	0	7
Client application	29	0	0	29
Security	4	0	0	4
Outsource Shipping	6	0	0	6
Exception Reporting	7	0	0	2
Final Report Output	5	0	0	5
Version Control	1	0	0	1

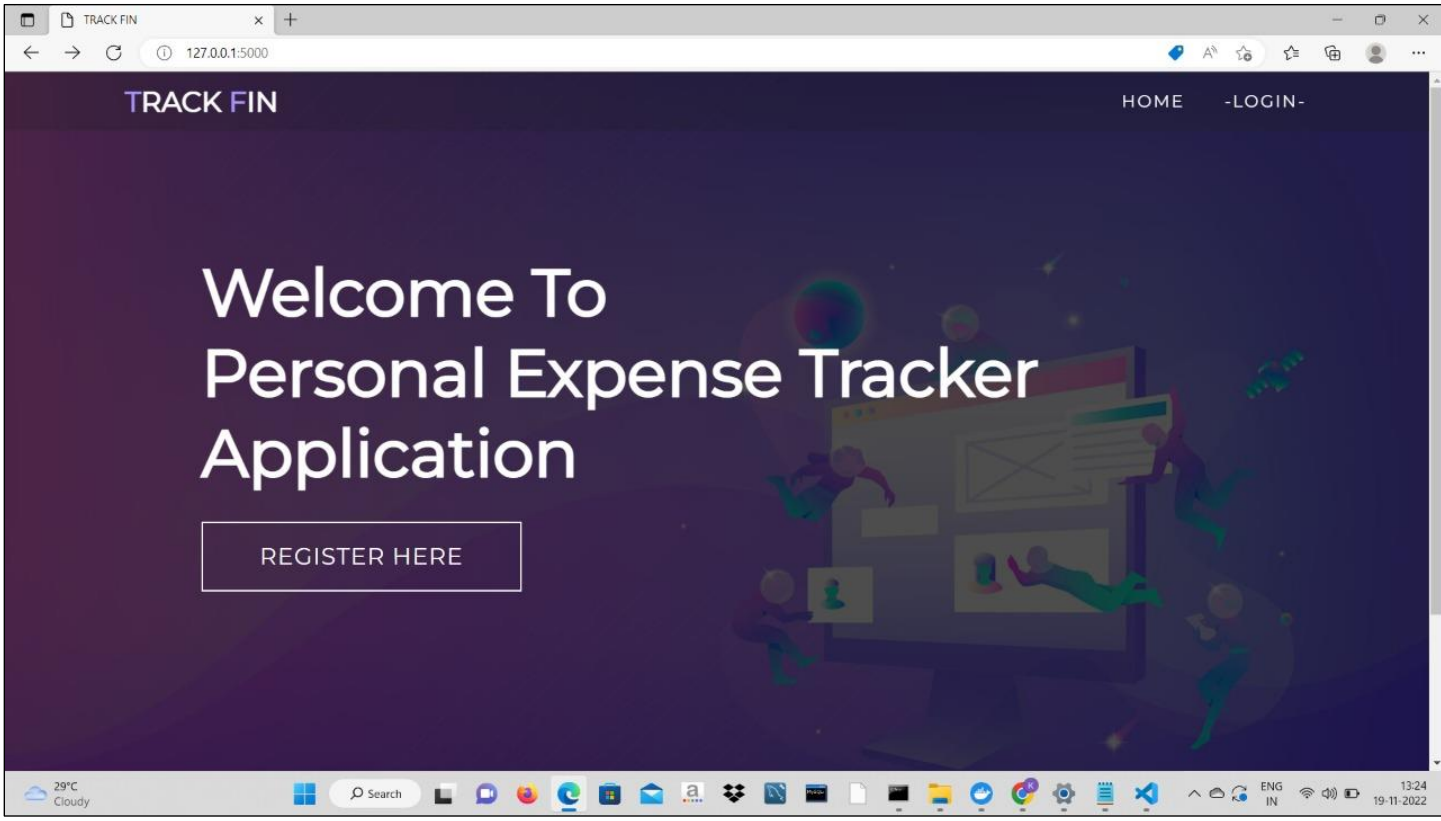
9.RESULTS

Performance Metrics:

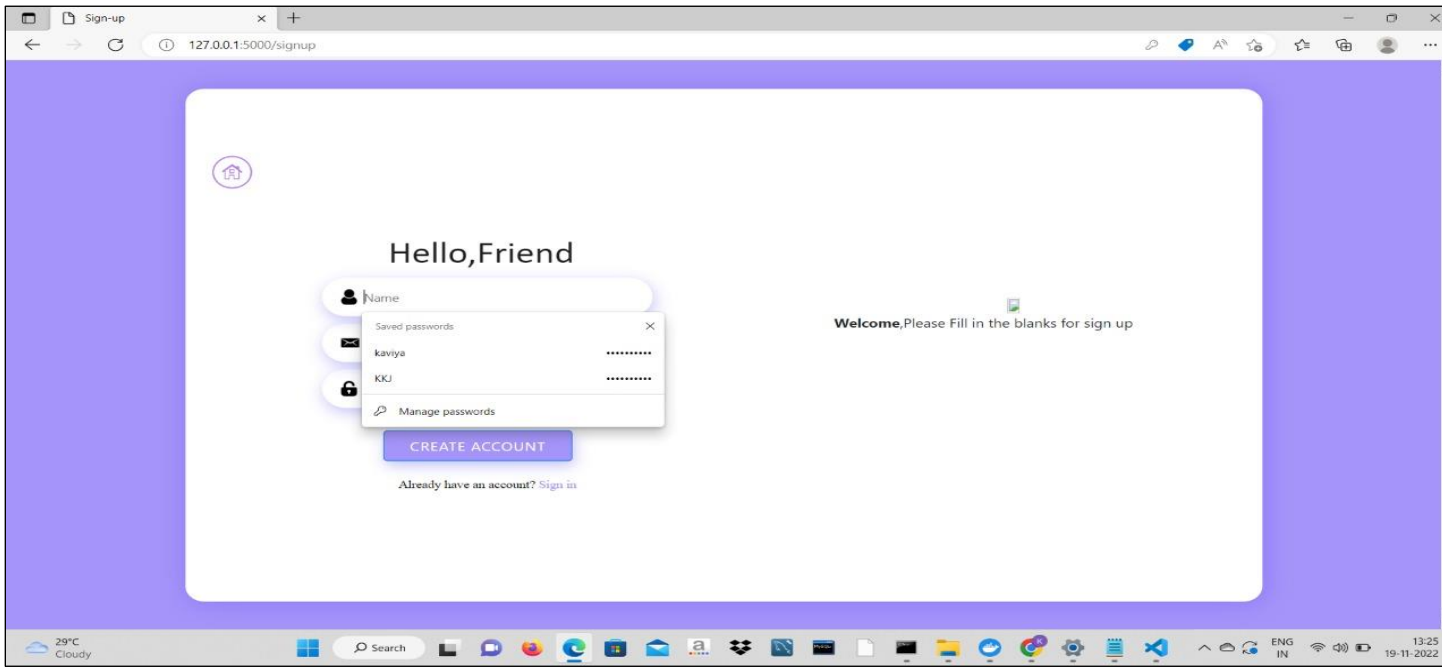
- Transaction Receipts: Capture and organize your payment receipts to keep track of your expenditure.
- Organizing Taxes: Import your documents to the expense tracking app, and it will streamline your income and expenses under theappropriate tax categories.
- Payments & Invoices:Accept and pay from credit cards, debit cards,net banking, mobile wallets, and bank transfers, and track the status of your invoices and bills in the mobile app itself.Also, the tracking app sends reminders for payments and automatically matches the payments with invoices.
- Reports: The expense tracking app generates and sends reports to give a detailed insight about profits, losses, budgets, income, balance sheets, etc.,

OUTPUT:

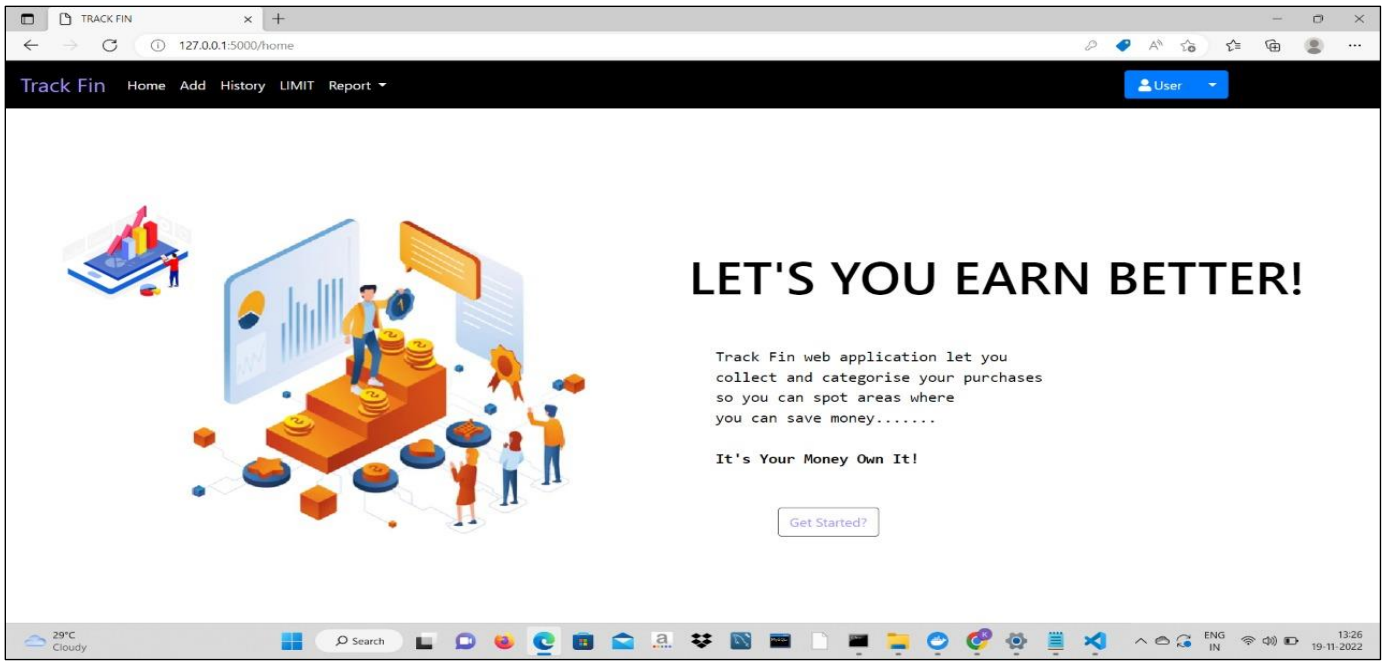
HOME PAGE



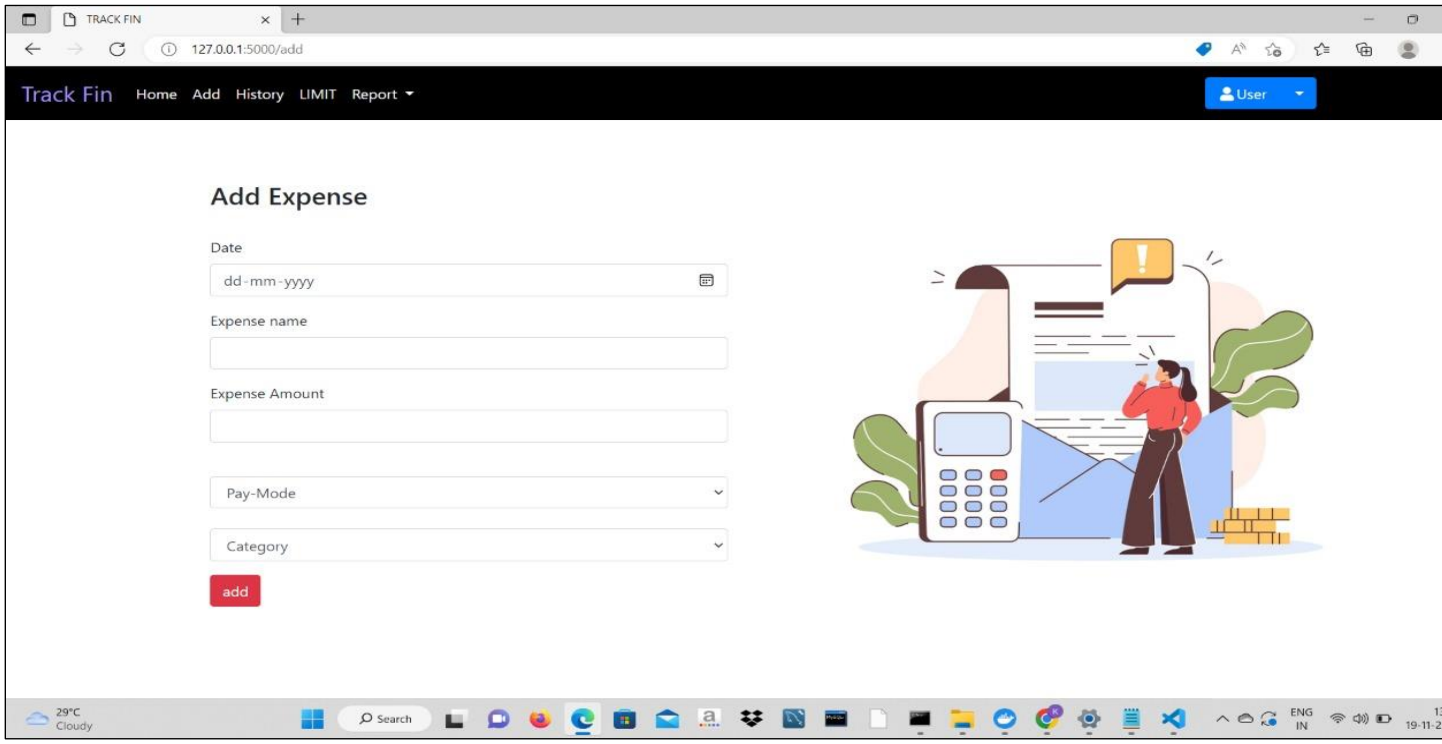
SIGNUP PAGE



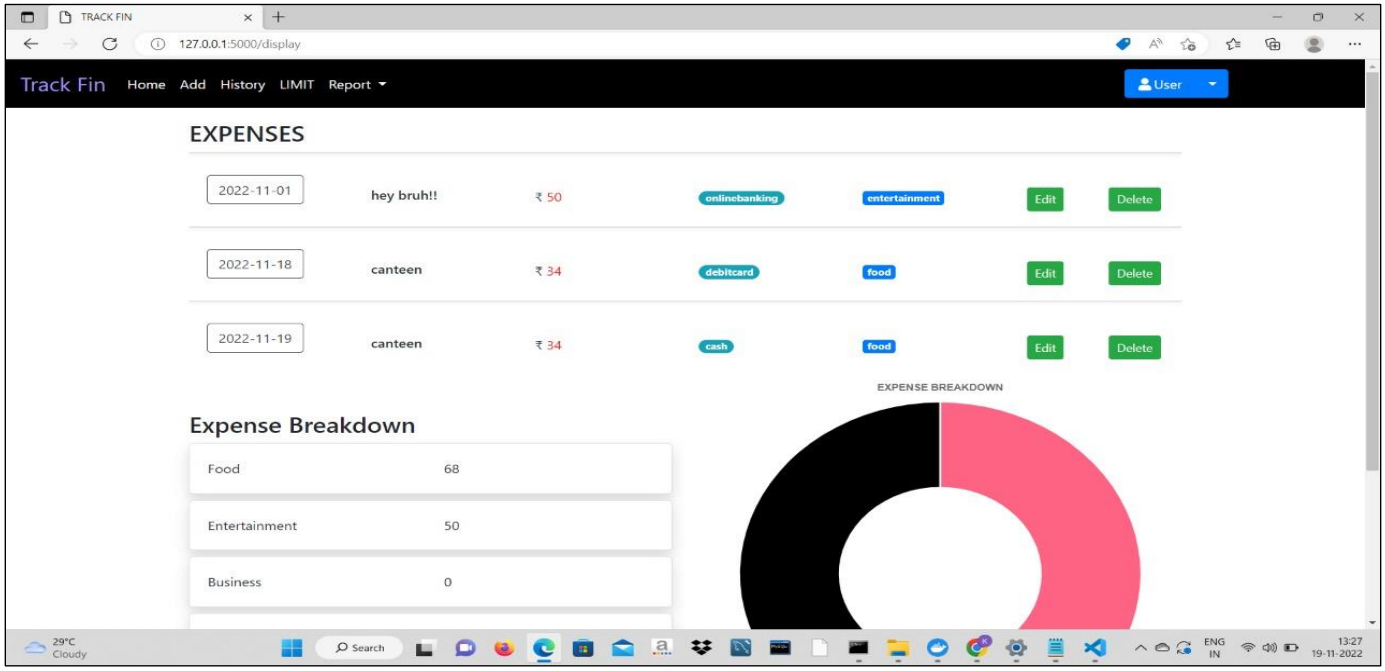
FEATURES



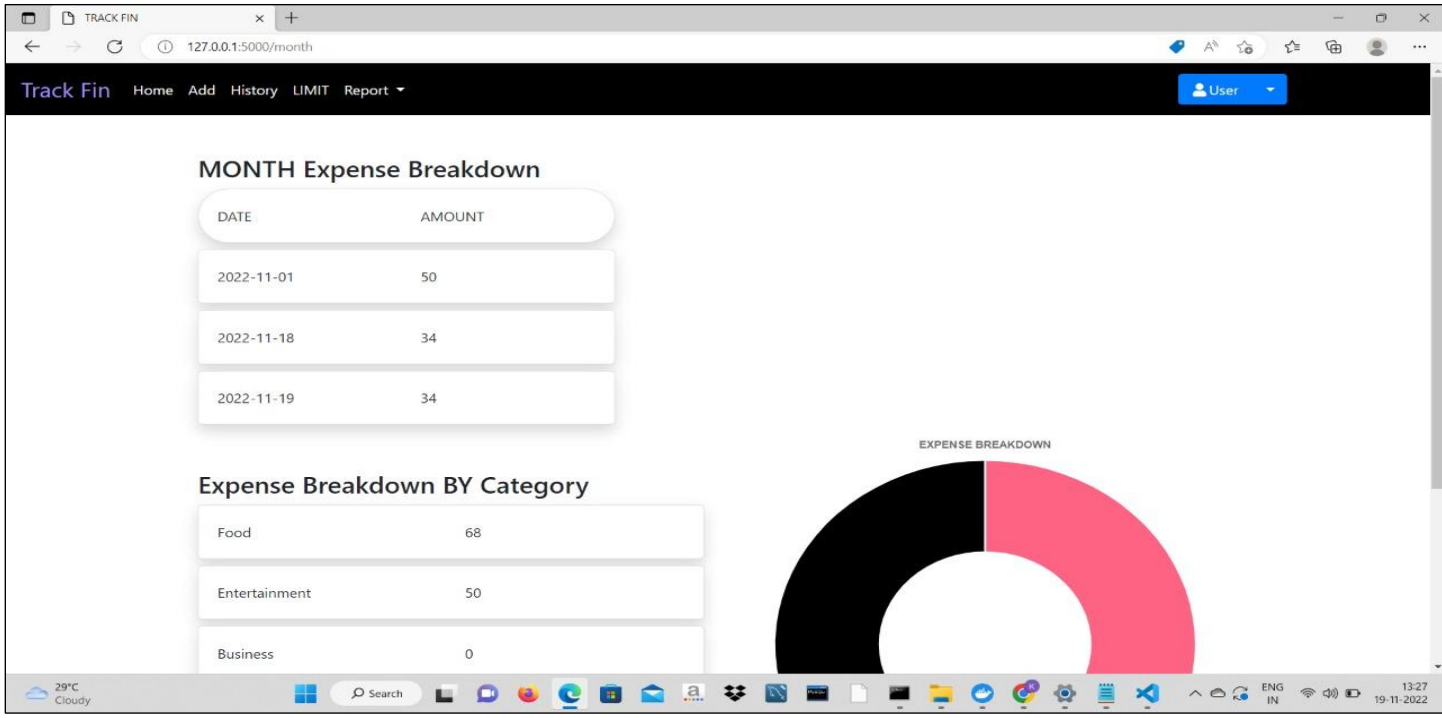
ADD EXPENSE PAGE



HISTORY PAGE



REPORT PAGE



10. ADVANTAGES AND DISADVANTAGES

- 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.

- Another pro is that for those who just wish to keep tracking spending by hand with a paper and pen or by entering data onto a computer spreadsheet, these options are also available.
- Some people like to keep a file folder or box to store receipts and record the cash spent each day.
- A pro of this simple daily tracking system is that it can make one more aware of where the money is going way before the end of a pay period or month.

- Tracking spending can help save money by recognizing unnecessary spending.
- 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.
- Inaccurate or unreasonable assumptions can quickly make a budget unrealistic. Budgets can lead to inflexibility in decision-making.
- Budgets need to be changed as circumstances change.
- Budgeting is a time consuming process – in large businesses, whole departments are sometimes dedicated to budget setting and control.

11.CONCLUSION

Tracking the expense daily can save the amount, but it can also help to set and work for financial goals for the future. It is exactly where the amount is going every month, It can be easily seen where some cutbacks and compromises can be made and are possible. The project what have developed works more efficient than the other available income and expense tracker. The project successfully avoids the manual calculation for avoiding calculating the income and expense per month and save time of user. The modules are developed with efficient, reliable and also in an attractive manner.

12.FUTURE SCOPE

In the future, The Online Income and Expense Tracker application can be further enhanced to include following features:

- The application can be extended to include scanning of barcode on the price tag which decreases the effort of entering the data in the input fields.

- Group: Apart from keeping a personal log, we are planning to extend this system to incorporate a shared expense group.
- The application can be designed in a way as to create a monthly analysis and report of the user's income and expenses to provide better understanding to the user and gain control of his or her expenditure.
- A notification system can be enabled in case when the expenses crosses over the income generated by the user to warn him or her about the situation.

13.APPENDIX

Demo Link:
https://youtu.be/w-0JeWEM5yU

Github:
IBM-EPBL/IBM-Project-2110-1658456268: Personal Expense Tracker Application (github.com)