PERSONAL EXPENSE TRACKER APPLICATION

A PROJECT REPORT

Submitted by

J. DAVID - 210519106014

V. DHANASEKAR - 210519106016

M. GAYATHRI - 210519106022

J. JENIN - 210519106032

B.K VINOTHKUMAR - 210519106072

for the course

HX8001 – Professional Readiness for innovation, Employability and Entrepreneurship

in

ELECTRONICS AND COMMUNICATION ENGINEERING

DMI COLLEGE OF ENGINEERING

ANNA UNIVERSITY: CHENNAI 600 025

NOVEMBER- 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"PERSONAL EXPENSE TRACKER APPLICATION"** is the bonafide work of "**J. DAVID (210519106014), V. DHANASEKAR (210519106016), M.GAYATHRI (210519106022), J. JENIIN (210519106032), B. KVINOTHKUMAR (210519106072)"** who carried out the project work under my supervision

SIGNATURE
Dr.M.Latha
HEAD OF THE DEPARTMENT

Department of ECE,
DMI College of Engineering,
Palanchur, Chennai – 600 123.

SIGNATURE
Mrs.J.Lurdhumary M.Tech
MENTOR

Department of ECE,
DMI College of Engineering ,
Palanchur, Chennai – 600 123.

ABSTRACT

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 expense's 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.

CONTENTS

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

8. TESTING

8.1 Test Cases

8.2 User Acceptance Testing

9. RESULTS

9.1 Performance Metrics

10. ADVANTAGES & DISADVANTAGES

11. CONCLUSION

12. FUTURE SCOPE

13. APPENDIX

Source Github Code

Project Demo Link

`

1.INTRODUCTION

1.1 Project overview

Mobile applications are top in user convenience and have over passed the web applications in terms of popularity and usability. There are various mobile applications that provide solutions to manage personal and group expense but not many of them provide a comprehensive view of both cases. In this paper, we develop a mobile application developed for the android platform that keeps record of user personal expenses, his/her contribution in group expenditures, top investment options, view of the current stock market, read authenticated financial news and grab the best ongoing offers in the market in popular categories. The proposed application would eliminate messy sticky notes, spreadsheets confusion and data handling inconsistency problems while offering the best overview of your expenses. With our application can manage their expenses and decide on their budget more effectively.

1.2 Purpose

It also known as expense manager and money manager, an expense tracker is a software or application that helps to keep an accurate record of your money inflow and outflow. Many people in India live on a fixed income, and they find that towards the end of the month they don't have sufficient money to meet their needs.

2.LITERATURE SURVEY

2.1 Existing problem

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

2.2 Reference

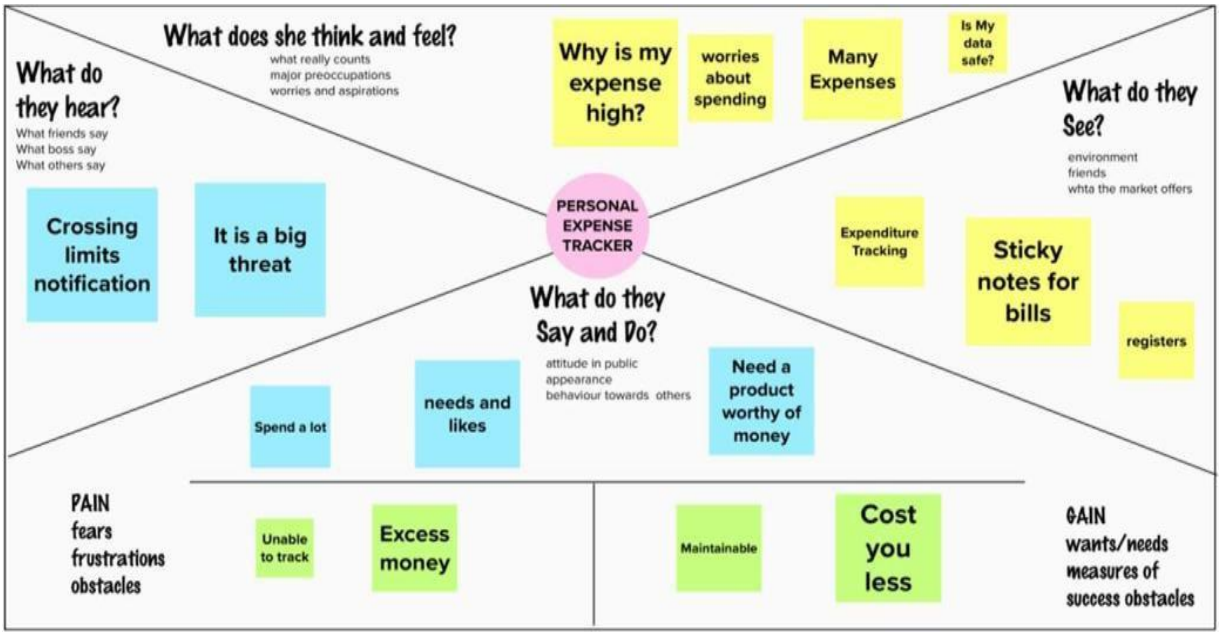
- https://nevonprojects.com/daily-expense-tracker-system/·
- https://data-flair.training/blogs/expense-tracker-python/·
- https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/· ·https://ijarsct.co.in/Paper391.pdf·
- https://kandi.openweaver.com/?landingpage=python_all_projects&utm_source=google&utm_medium=cpc&utm_campaign=promo_kandi_ie&utm_content=kandi_ie_search&utm_term=python_devs&gclid=Cj0KCQiAgribBhDkARIsAASA5bukrZgbI9UZxzpoyf0P-ofB1mZNxzc-okUP-3TchpYMclHTYFYiqP8aAmmwEALw_wcB·

2.3 Problem Statement Definition

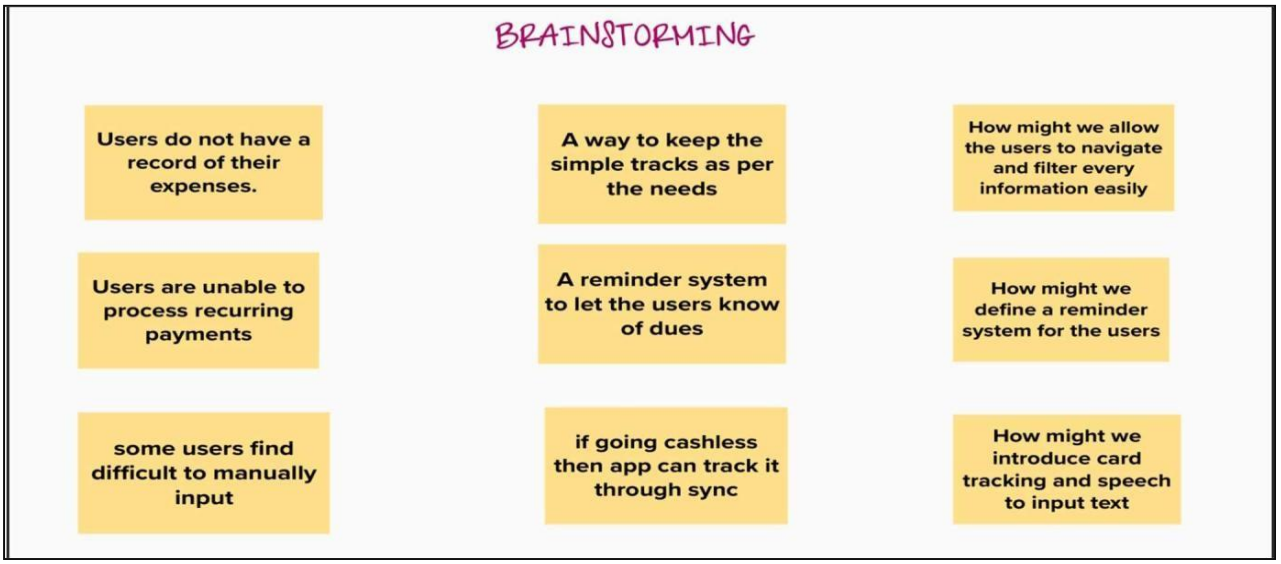
This Expense Tracker is a web application that facilitates the users to keep track and manage their personal as well as business expenses. This application helps the users to keep a digital diary. It will keep track of a user's income and expenses on a daily basis. The user will be able to add his/her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

3.IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



3.3 Proposed Solution

All people in the earning sector needs a way to manage their financial resources and track their expenditure, so that they can improve and monitor their spending habits. This makes them understand the importance of financial management and makes them better decisions in the future. 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. The solution to this problem is, the people who gets regular payments can able to track their payments and void unwanted expenses. If the limit is exceeded the user will be notified with an email alert.

3.4 Proposed Solution Fit

The solution to this problem is, the people who gets regular payments can able to track their payments and avoid unwanted expenses. If the limit is exceeded the user will be notified with an email alert.

- Novelty / Uniqueness Notification can be receive through email.
- Social Impact / Customer Satisfaction Using this application one can track their personal expenses and frame a monthly/annual budget. If your expense exceeded than specified limit, the application will show you an alert message. This will make a impact on Mobile Banking for Customers' Satisfaction.
- Business Model (Revenue Model) Business people can use subscription/premium feature of this application to gain revenue.
- Scalability of the Solution The scalability of the application depends on security, the working of the application even during when the network gets down etc...

4.REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

- 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 Tracking Expense Helpful insights about money management·
- FR-4 Alert Message Give alert mail if the amount exceeds the budget limit·
- FR-5 Category This application shall allow users to add categories of their expenses·

4.2 Non-Functional requirement

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

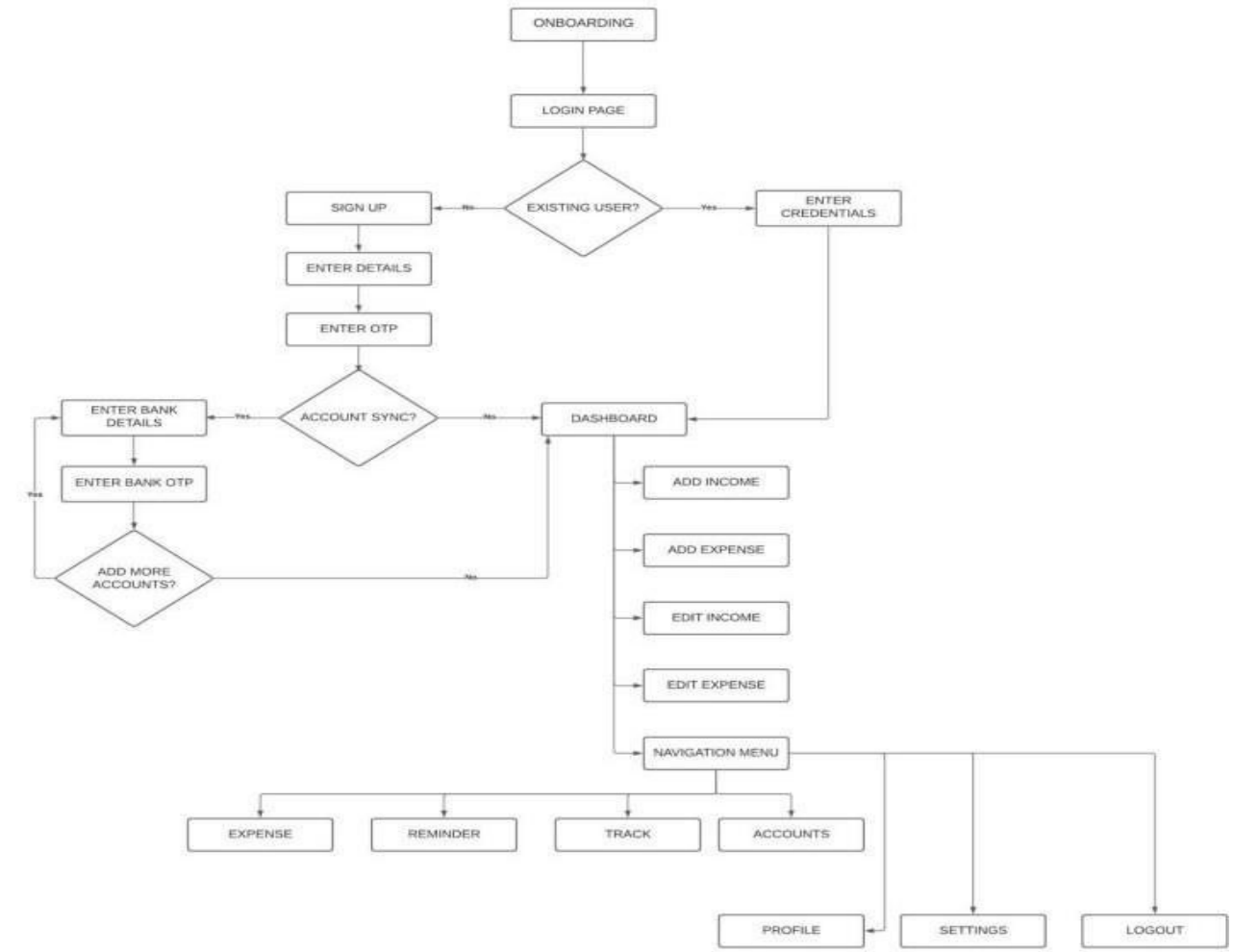
- NFR-1 Usability You will able to allocate money to different priorities and also help you to cut down on unnecessary spending·
- NFR-2 Security More security of the customer data and bank account details.·
- NFR-3 Reliability Used to manage his/her expense so that the user is the path of financial stability. It is categorized by week, month, and year and also helps to see more expenses made. Helps to define their own categories.·
- NFR-4 Performance The types of expense are categories along with an option. Through put of the system is increased due to light weight database support.·
- NFR-5 Availability Able to track business expense and monitor important for maintaining healthy cash flow. NFR-6 Scalability The ability to appropriately handle increasing demands.·

5.PROJECT DESIGNER

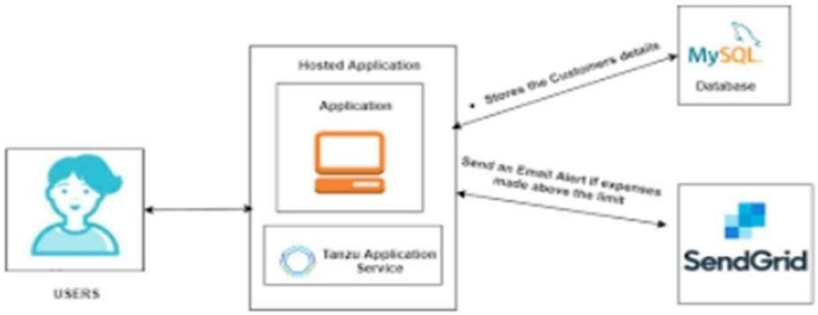
5.1 Data Flow Diagrams

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 stores.

Solution architecture:



5.2 Solution &Technical Architecture



5.3 USER STORIES

Use the below **TABLE.5.3** to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story/ Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user can register for the application by entering my email, password, and confirm my password	I can access my account /dashboard	High	Sprint 1
		USN-2	As a user, I will receive confirmation email once I have	I can receive confirmation email &click confirm	High	Sprint-1

			registered for the application			
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail	I can register by entering the details	Medium	Sprint 1
	Login	USN-5	As a user, I can log into the application by entering	I can access my dashboard	High	Sprint 1
	Dashboard	USN-6	As a user ,I can log into the dashboard	I can access my account /	High	Sprint 1

			and manage income	dashboard		
Customer (Web user)		USN-7	As a user, I can register for the application by Bank account.	I can access my account / dashboard	High	Sprint 1
Customer Care Executive		USN-8	As a user, I can get a report is based on the details	I can manage my money by viewing this report	Medium	Sprint 1
		USN-9	As a user, I can get an email if the money level is above the limit	They can receive alert email	High	Sprint 1
Administrative	Responsibility	USN-10	As a system administrator, track the user expenses anytime	I can track expense	High	Sprint 1

6.PROJECT PLANNING &SCHEDULING

6.1 Sprint Planning & Estimation

TABLE6.1

Sprint	Functional Requirement (Epic)	User Story Number	User Story/Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As user ,I can register for the application by entering my email, password ,and confirming my password.	2	High	David J
Sprint- 1		USN-2	As a user ,I will receive confirmatio nemail once Ihey have registered for the application	1	Medium	Dhanasekar V

Sprint-2	Login	USN-3	As a user ,I can register for the application through Facebook	2	Medium	Jenin J
Sprint- 1	Dashboard	USN-4	As a user, I can register for the application	2	High	Gayathri M

6.2 Sprint Delivery Schedule

TABLE 6.2

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6Days	24Oct2022	29Oct2022	20	29Oct2022
Sprint-2	20	6Days	31Oct2022	05Nov2022	18	06Nov2022
Sprint-3	20	6Days	07Nov2022	12Nov2022	15	14Nov2022
Sprint-4	20	6Days	14Nov2022	19Nov2022	19	21Nov2022

6.3 Reports From JIRA

Burndown Chart:

A burn down chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.

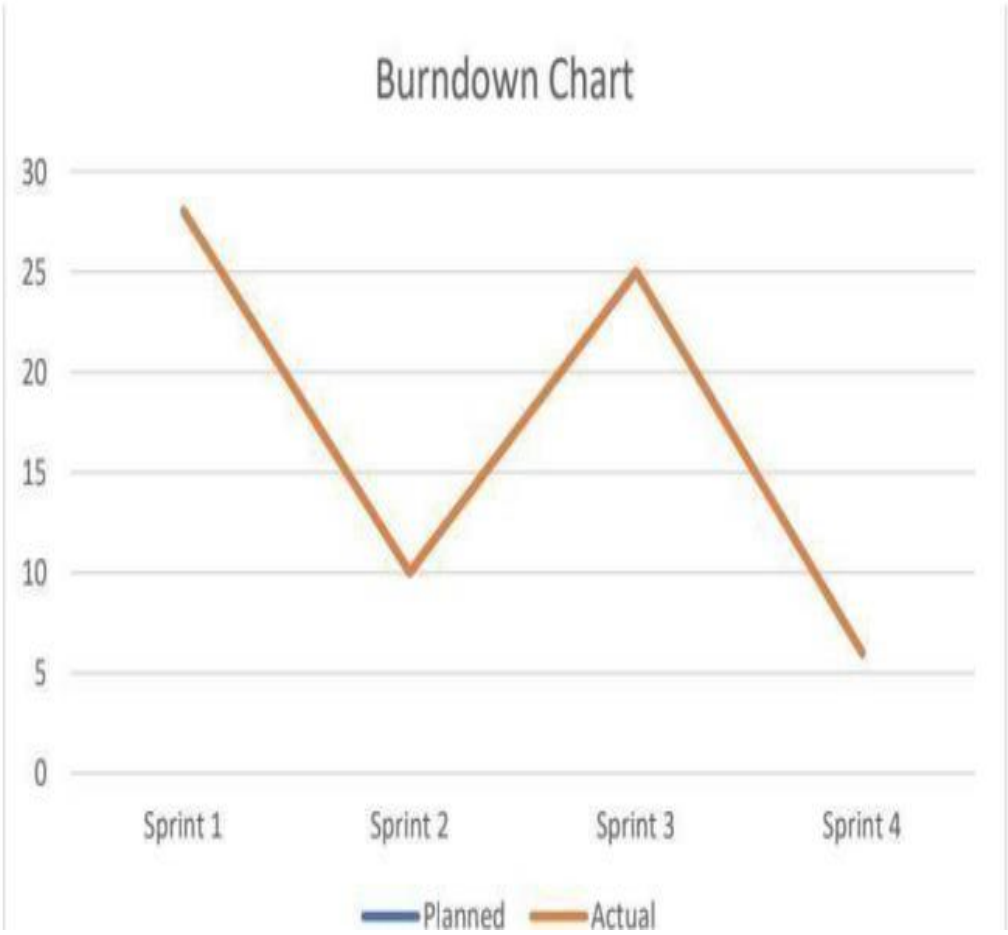


Figure 6.3.1

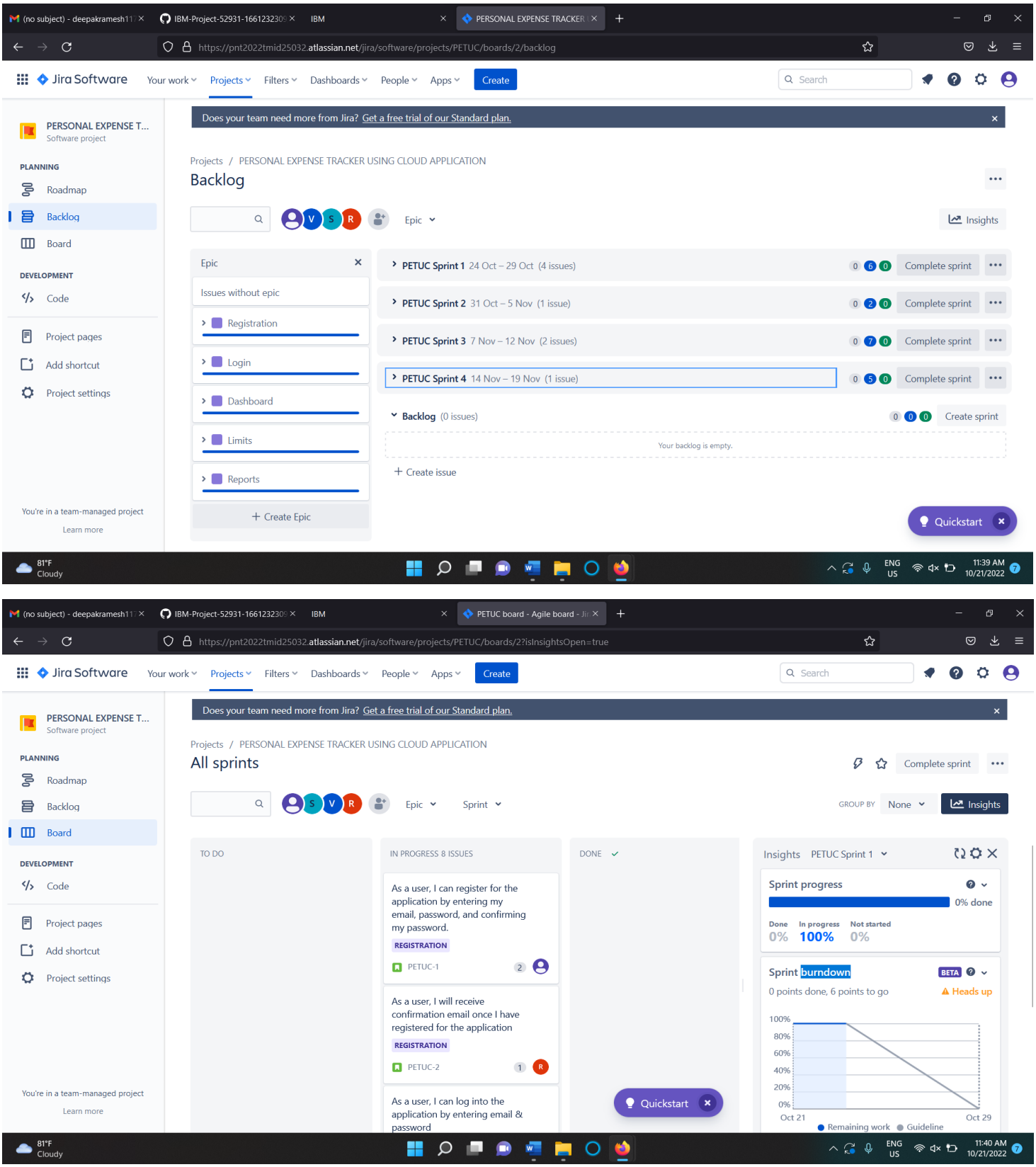

Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
PERSONAL EXPENSE T...
Software project
PLANNING
Roadmap
Backlog
Board
DEVELOPMENT
Code
Project pages
Add shortcut
Project settings
You're in a team-managed project
Learn more
Does your team need more from Jira? Get a free trial of our Standard plan.
Projects / PERSONAL EXPENSE TRACKER USING CLOUD APPLICATION
Backlog
Epic
Insights
Issues without epic
Registration
Login
Dashboard
Limits
Reports
Create Epic
PETUC Sprint 1 24 Oct – 29 Oct (4 issues)
PETUC Sprint 2 31 Oct – 5 Nov (1 issue)
PETUC Sprint 3 7 Nov – 12 Nov (2 issues)
PETUC Sprint 4 14 Nov – 19 Nov (1 issue)
Complete sprint
Backlog (0 issues)
Create sprint
Your backlog is empty.
Create issue
Quickstart
81°F
Cloudy
ENG
US
11:39 AM
10/21/2022
All sprints
Sprint
GROUP BY
None
TO DO
IN PROGRESS 8 ISSUES
DONE
As a user, I can register for the application by entering my email, password, and confirming my password.
REGISTRATION
PETUC-1
As a user, I will receive confirmation email once I have registered for the application
REGISTRATION
PETUC-2
As a user, I can log into the application by entering email & password
Insights
PETUC Sprint 1
Sprint progress
0% done
Done
In progress
Not started
0%
100%
0%
Sprint burndown
BETA
0 points done, 6 points to go
Heads up
100%
80%
60%
40%
20%
0%
Oct 21
Oct 29
Remaining work
Guideline
11:40 AM

Figure 6.3.2

7.CODING AND SOLUTIONING

7.1 Feature 1:

Add Expense, update expenses, delete expense, set limit, send alert E-mails to users by using send grid and app.py code collaborate and run without any error.

7.2 Feature 2:

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.

Codes:

App. py:

```
from flask import Flask,
render_template, request, redirect, session
import re
from flask_db2 import DB2
import ibm_db
import ibm_db_dbi
from sendemail import
sendgridmail,sendmail

# from gevent.pywsgi import
WSGIServer
import os
app = Flask(__name__)
app.secret_key = 'a'
"""

dsn_hostname = "9938aec0-8105-
433e-8bf9-
0fbb7e483086.c1ogj3sd0tgtu0lqde00.datab
ases.appdomain.cloud:32459"
dsn_uid = "dgn29807"
dsn_pwd = "t5KGl9K11IA7kJsM"
dsn_driver = "{IBM DB2 ODBC
DRIVER}"
```

```
        dsn_database = "bludb"
        dsn_port = "32328"
        dsn_protocol = "tcpip"
        dsn = (
            "DRIVER={0};"
            "DATABASE={1};"
            "HOSTNAME={2};"
            "PORT={3};"
            "PROTOCOL={4};"
            "UID={5};"
            "PWD={6};"
        ).format(dsn_driver, dsn_database,
    dsn_hostname, dsn_port, dsn_protocol,
    dsn_uid, dsn_pwd)
        """
        # app.config['DB2_DRIVER'] = '{IBM
    DB2 ODBC DRIVER}'
        app.config['database'] = 'bludb'
        app.config['hostname'] = '9938aec0-
    8105-433e-8bf9-
    0fbb7e483086.c1ogj3sd0tgtu0lqde00.datab
    ases.appdomain.cloud:32459'
        app.config['port'] = '32328'
        app.config['protocol'] = 'tcpip'

        app.config['uid'] = 'qbs37391'
        app.config['pwd'] =
    'Qe0BUshQSIcc66gG'
        app.config['security'] = 'SSL'
        try:
            mysql = DB2(app)


    conn_str='database=bludb;hostname=2d46
    b6b4-cbf6-40eb-bbce-
    6251e6ba0300.bs2io90l08kqb1od8lcg.data
    bases.appdomain.cloud;port=32328;protoco
    l=tcpip;\
uid=qbs37391;pwd=Qe0BUshQSIcc66gG;securit
y=SSL'
            ibm_db_conn =
    ibm_db.connect(conn_str,",")
            print("Database connected without
    any error !!")
        except:
            print("IBM DB Connection error   :
    " + DB2.conn_errormsg())

        #HOME--PAGE
        @app.route("/home")
        def home():
```

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


    #SIGN--UP--OR--REGISTER
    @app.route("/signup")
    def signup():
        return
render_template("signup.html")
    @app.route('/register', methods
=['GET', 'POST'])
    def register():
        msg = ''
        print("Break point1")
        if request.method == 'POST' :
            username =
request.form['username']
            email = request.form['email']
            password =
request.form['password']
            print("Break point2" + "name: " +
username + "------" + email + "------" +
password)
            try:
                print("Break point3")
                connectionID =
ibm_db_dbi.connect(conn_str, '', '')
                cursor = connectionID.cursor()
                print("Break point4")
            except:
                print("No connection
Established")
            print("Break point5")
            sql = "SELECT * FROM register
WHERE username = ?"


       stmt = ibm_db.prepare(ibm_db_conn,
sql)
            ibm_db.bind_param(stmt, 1,
username)
            ibm_db.execute(stmt)
            result = ibm_db.execute(stmt)
            print(result)
            account = ibm_db.fetch_row(stmt)
            print(account)
            param = "SELECT * FROM
register WHERE username = '" + "\'" +
username + "\'"
```

```
            res =
ibm_db.exec_immediate(ibm_db_conn,
param)
            print("---- ")
            dictionary =
ibm_db.fetch_assoc(res)
            while dictionary != False:
               print("The ID is : ",
dictionary["USERNAME"])
               dictionary =
ibm_db.fetch_assoc(res)
            print("break point 6")
            if account:
               msg = 'Username already exists
!'
            elif not
re.match(r'[^@]+@[^@]+\.[^@]+', email):
               msg = 'Invalid email address !'
            elif not re.match(r'[A-Za-z0-9]+',
username):

      msg = 'name must contain only
characters and numbers !'
            else:
               sql2 = "INSERT INTO register
(username, email,password) VALUES (?,
?, ?)"
               stmt2 =
ibm_db.prepare(ibm_db_conn, sql2)
               ibm_db.bind_param(stmt2, 1,
username)
               ibm_db.bind_param(stmt2, 2,
email)
               ibm_db.bind_param(stmt2, 3,
password)
               ibm_db.execute(stmt2)
               msg = 'You have successfully
registered !'
            return
render_template('signup.html', msg = msg)

       #LOGIN--PAGE
      @app.route("/signin")
      def signin():
         return render_template("login.html")
      @app.route('/login',methods =['GET',
'POST'])
      def login():
         global userid
         msg = ''
         if request.method == 'POST' :
```

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

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

            param = "SELECT * FROM
register WHERE username = '" + "\'" +
username + "\'" + " and password = '" + "\'"
+ password + "\'"
            res =
ibm_db.exec_immediate(ibm_db_conn,
param)
            dictionary =
ibm_db.fetch_assoc(res)
            if account:
                session['loggedin'] = True
                session['id'] = dictionary["ID"]
                userid = dictionary["ID"]
                session['username'] =
dictionary["USERNAME"]
                session['email'] =
dictionary["EMAIL"]
                return redirect('/home')

        else:
                msg = 'Incorrect username /
password !'
         return render_template('login.html',
msg = msg)

    #ADDING----DATA
    @app.route("/add")
    def adding():
        return render_template('add.html')
    @app.route('/addexpense',methods=['G
ET', 'POST'])
    def addexpense():
        date = request.form['date']
        expensename =
request.form['expensename']
```

```
        amount = request.form['amount']
        paymode = request.form['paymode']
        category = request.form['category']
        print(date)
        p1 = date[0:10]
        p2 = date[11:13]
        p3 = date[14:]
        p4 = p1 + "-" + p2 + "." + p3 + ".00"
        print(p4)
        sql = "INSERT INTO expenses
(userid, date, expensename, amount,
paymode, category) VALUES (?, ?, ?, ?, ?,
?)"
        stmt =
ibm_db.prepare(ibm_db_conn, sql)


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

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

temp.append(dictionary["USERID"])


     temp.append(dictionary["DATE"])
```

```
temp.append(dictionary["EXPENSENAME"])

        temp.append(dictionary["AMOUNT"])

        temp.append(dictionary["PAYMODE"])

        temp.append(dictionary["CATEGORY"])
                expense.append(temp)
                print(temp)
                dictionary =
        ibm_db.fetch_assoc(res)
            total=0
            for x in expense:
                total += x[4]
            param = "SELECT id, limitss FROM
        limits WHERE userid = " +
        str(session['id']) + " ORDER BY id DESC
        LIMIT 1"
            res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)
            dictionary = ibm_db.fetch_assoc(res)
            row = []
            s = 0
            while dictionary != False:
                temp = []

        temp.append(dictionary["LIMITSS"])
                row.append(temp)


              dictionary = ibm_db.fetch_assoc(res)
                s = temp[0]
             if total > int(s):
                msg = "Hello " +
        session['username'] + " , " + "you have
        crossed the monthly limit of Rs. " + s + "/-
        !!!" + "\n" + "Thank you, " + "\n" + "Team
        Personal Expense Tracker."
                sendmail(msg,session['email'])
             return redirect("/display")

          #DISPLAY---graph
          @app.route("/display")
          def display():

        print(session["username"],session['id'])
            param = "SELECT * FROM
        expenses WHERE userid = " +
        str(session['id']) + " ORDER BY date
        DESC"
```

```
            res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)
            dictionary = ibm_db.fetch_assoc(res)
            expense = []
            while dictionary != False:
               temp = []
               temp.append(dictionary["ID"])

        temp.append(dictionary["USERID"])

        temp.append(dictionary["DATE"])

    temp.append(dictionary["EXPENSENAME"])

        temp.append(dictionary["AMOUNT"])

        temp.append(dictionary["PAYMODE"])

        temp.append(dictionary["CATEGORY"])
               expense.append(temp)
               print(temp)
               dictionary =
        ibm_db.fetch_assoc(res)
            return render_template('display.html'
        ,expense = expense)

            #delete---the--data
            @app.route('/delete/<string:id>',
        methods = ['POST', 'GET' ])
            def delete(id):
               param = "DELETE FROM expenses
        WHERE  id = " + id
            res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)
            print('deleted successfully')
            return redirect("/display")

            #UPDATE---DATA
            @app.route('/edit/<id>', methods =
        ['POST', 'GET' ])
            def edit(id):
               param = "SELECT * FROM
        expenses WHERE  id = " + id

             res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)
            dictionary = ibm_db.fetch_assoc(res)
            row = []
            while dictionary != False:
```

```
                temp = []
                temp.append(dictionary["ID"])

        temp.append(dictionary["USERID"])

        temp.append(dictionary["DATE"])

temp.append(dictionary["EXPENSENAME"])

        temp.append(dictionary["AMOUNT"])

        temp.append(dictionary["PAYMODE"])

        temp.append(dictionary["CATEGORY"])
                row.append(temp)
                print(temp)
                dictionary =
        ibm_db.fetch_assoc(res)
              print(row[0])
              return render_template('edit.html',
        expenses = row[0])
            @app.route('/update/<id>', methods =
        ['POST'])
            def update(id):
             if request.method == 'POST' :
                date = request.form['date']

             expensename =
        request.form['expensename']
                amount = request.form['amount']
                paymode =
        request.form['paymode']
                category = request.form['category']
                p1 = date[0:10]
                p2 = date[11:13]
                p3 = date[14:]
                p4 = p1 + "-" + p2 + "." + p3 +
        ".00"
                sql = "UPDATE expenses SET date
        = ? , expensename = ? , amount = ?,
        paymode = ?, category = ? WHERE id = ?"
                stmt =
        ibm_db.prepare(ibm_db_conn, sql)
                ibm_db.bind_param(stmt, 1, p4)
                ibm_db.bind_param(stmt, 2,
        expensename)
                ibm_db.bind_param(stmt, 3,
        amount)
                ibm_db.bind_param(stmt, 4,
        paymode)
                ibm_db.bind_param(stmt, 5,
        category)
```

```
            ibm_db.bind_param(stmt, 6, id)
            ibm_db.execute(stmt)
            print('successfully updated')
            return redirect("/display")
        #limit
        @app.route("/limit" )
        def limit():

            return redirect('/limitn')
        @app.route("/limitnum" , methods =
['POST' ])
        def limitnum():
            if request.method == "POST":
                number= request.form['number']
                sql = "INSERT INTO limits
(userid, limitss) VALUES (?, ?)"
                stmt =
ibm_db.prepare(ibm_db_conn, sql)
                ibm_db.bind_param(stmt, 1,
session['id'])
                ibm_db.bind_param(stmt, 2,
number)
                ibm_db.execute(stmt)
                return redirect('/limitn')
        @app.route("/limitn")
        def limitn():
            param = "SELECT id, limitss FROM
limits WHERE userid = " +
str(session['id']) + " ORDER BY id DESC
LIMIT 1"
            res =
ibm_db.exec_immediate(ibm_db_conn,
param)
            dictionary = ibm_db.fetch_assoc(res)
            row = []
            s = " /-"
            while dictionary != False:
                temp = []

temp.append(dictionary["LIMITSS"])

            row.append(temp)
                dictionary =
ibm_db.fetch_assoc(res)
                s = temp[0]
            return render_template("limit.html" ,
y= s)
        #REPORT
        @app.route("/today")
        def today():
            param1 = "SELECT TIME(date) as
tn, amount FROM expenses WHERE
```

```
        userid = '' + str(session['id']) + '' AND
        DATE(date) = DATE(current timestamp)
        ORDER BY date DESC''
                res1 =
        ibm_db.exec_immediate(ibm_db_conn,
        param1)
                dictionary1 =
        ibm_db.fetch_assoc(res1)
                texpense = []
                while dictionary1 != False:
                    temp = []
                    temp.append(dictionary1["TN"])

        temp.append(dictionary1["AMOUNT"])
                    texpense.append(temp)
                    print(temp)
                    dictionary1 =
        ibm_db.fetch_assoc(res1)
                param = "SELECT * FROM
        expenses WHERE userid = " +
        str(session['id']) + " AND DATE(date) =

            DATE(current timestamp) ORDER BY
        date DESC"
                res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)
                dictionary =
        ibm_db.fetch_assoc(res)
                expense = []
                while dictionary != False:
                    temp = []
                    temp.append(dictionary["ID"])

        temp.append(dictionary["USERID"])

        temp.append(dictionary["DATE"])

    temp.append(dictionary["EXPENSENAME"])
            temp.append(dictionary["AMOUNT"])
            temp.append(dictionary["PAYMODE"]
        )
            temp.append(dictionary["CATEGORY
        "])
                    expense.append(temp)
                    print(temp)
                    dictionary =
        ibm_db.fetch_assoc(res)
                total=0
                t_food=0
                t_entertainment=0
                t_business=0
```

```
            t_rent=0
            t_EMI=0

        t_other=0
            for x in expense:
                total += x[4]
                if x[6] == "food":
                    t_food += x[4]
                elif x[6] == "entertainment":
                    t_entertainment  += x[4]
                elif x[6] == "business":
                    t_business  += x[4]
                elif x[6] == "rent":
                    t_rent  += x[4]
                elif x[6] == "EMI":
                    t_EMI  += x[4]
                elif x[6] == "other":
                    t_other  += x[4]
            print(total)
            print(t_food)
            print(t_entertainment)
            print(t_business)
            print(t_rent)
            print(t_EMI)
            print(t_other)
            return
render_template("today.html", texpense =
texpense, expense = expense,  total = total ,
                            t_food =
t_food,t_entertainment =  t_entertainment,
                            t_business =
t_business,  t_rent =  t_rent,
                            t_EMI =  t_EMI,
t_other =  t_other )
        @app.route("/month")

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

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

        temp.append(dictionary1["TOT"])
                texpense.append(temp)
                print(temp)
                dictionary1 =
        ibm_db.fetch_assoc(res1)
            param = "SELECT * FROM
        expenses WHERE userid = " +
        str(session['id']) + " AND MONTH(date) =
        MONTH(current timestamp) AND
        YEAR(date) = YEAR(current timestamp)
        ORDER BY date DESC"
            res =
        ibm_db.exec_immediate(ibm_db_conn,
        param)


          dictionary = ibm_db.fetch_assoc(res)
            expense = []
            while dictionary != False:
                temp = []
                temp.append(dictionary["ID"])

        temp.append(dictionary["USERID"])

        temp.append(dictionary["DATE"])

    temp.append(dictionary["EXPENSENAME"])

        temp.append(dictionary["AMOUNT"])

        temp.append(dictionary["PAYMODE"])

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

              t_food += x[4]
                elif x[6] == "entertainment":
```

```
                t_entertainment  += x[4]
            elif x[6] == "business":
                t_business  += x[4]
            elif x[6] == "rent":
                t_rent  += x[4]
            elif x[6] == "EMI":
                t_EMI  += x[4]
            elif x[6] == "other":
                t_other  += x[4]
        print(total)
        print(t_food)
        print(t_entertainment)
        print(t_business)
        print(t_rent)
        print(t_EMI)
        print(t_other)
        return
render_template("today.html", texpense =
texpense, expense = expense,  total = total ,
                        t_food =
t_food,t_entertainment =  t_entertainment,
                        t_business =
t_business,  t_rent =  t_rent,
                        t_EMI =  t_EMI,
t_other =  t_other )
    @app.route("/year")
    def year():
        param1 = "SELECT MONTH(date)
as mn, SUM(amount) as tot FROM
expenses WHERE userid = " +

    str(session['id']) + " AND YEAR(date)
= YEAR(current timestamp) GROUP BY
MONTH(date) ORDER BY
MONTH(date)"
        res1 =
ibm_db.exec_immediate(ibm_db_conn,
param1)
        dictionary1 =
ibm_db.fetch_assoc(res1)
        texpense = []
        while dictionary1 != False:
            temp = []
            temp.append(dictionary1["MN"])

temp.append(dictionary1["TOT"])
            texpense.append(temp)
            print(temp)
            dictionary1 =
ibm_db.fetch_assoc(res1)
        param = "SELECT * FROM
expenses WHERE userid = " +
```

```
str(session['id']) + " AND YEAR(date) =
YEAR(current timestamp) ORDER BY
date DESC"
            res =
ibm_db.exec_immediate(ibm_db_conn,
param)
            dictionary =
ibm_db.fetch_assoc(res)
            expense = []
            while dictionary != False:

         temp = []
                temp.append(dictionary["ID"])

temp.append(dictionary["USERID"])

temp.append(dictionary["DATE"])
      temp.append(dictionary["EXPENSEN
AME"])
 temp.append(dictionary["AMOUNT"])
      temp.append(dictionary["PAYMODE"]
)
      temp.append(dictionary["CATEGORY
"])
                expense.append(temp)
                print(temp)
                dictionary =
ibm_db.fetch_assoc(res)
            total=0
            t_food=0
            t_entertainment=0
            t_business=0
            t_rent=0
            t_EMI=0
            t_other=0
            for x in expense:
                total += x[4]
                if x[6] == "food":
                    t_food += x[4]
                elif x[6] == "entertainment":
                    t_entertainment  += x[4]
                elif x[6] == "business":
                    t_business  += x[4]
           elif x[6] == "rent":
                    t_rent  += x[4]
                elif x[6] == "EMI":
                    t_EMI  += x[4]
                elif x[6] == "other":
                    t_other  += x[4]
            print(total)
            print(t_food)
            print(t_entertainment)
```

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

The other code features are submitted in github : refer the link ‘ GIT HUB ‘

8.TESTING

8.1 TEST CASES:

- Login Page (Functional)
- Login Page (UI)
- Add Expense Page (Functional)

8.2 User Acceptance Testing:

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

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

TABLE 8.2.1

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	8	15
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	9	2	4	11	20
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	0	1	8
Totals	22	14	11	22	51

3. Test Case Analysis

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

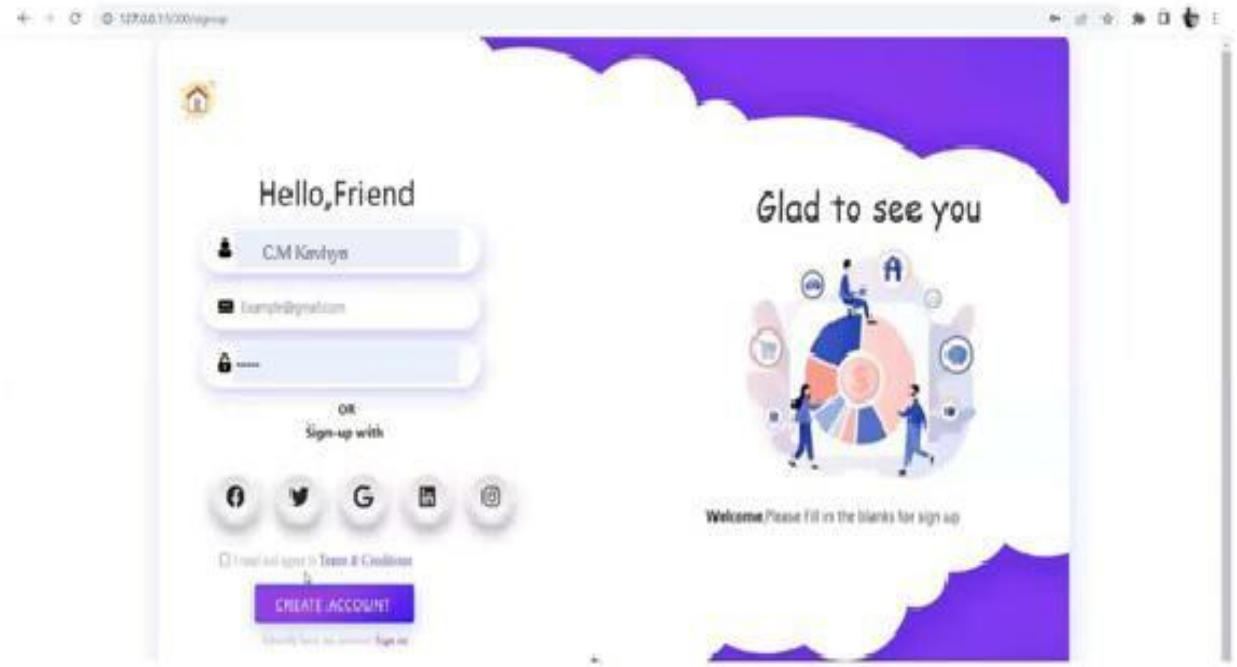
TABLE 8.2.2

Section	Total Cases	Not Tested	Fail	Pass
Interface	7	0	0	7
Login	43	0	0	43
Logout	2	0	0	2

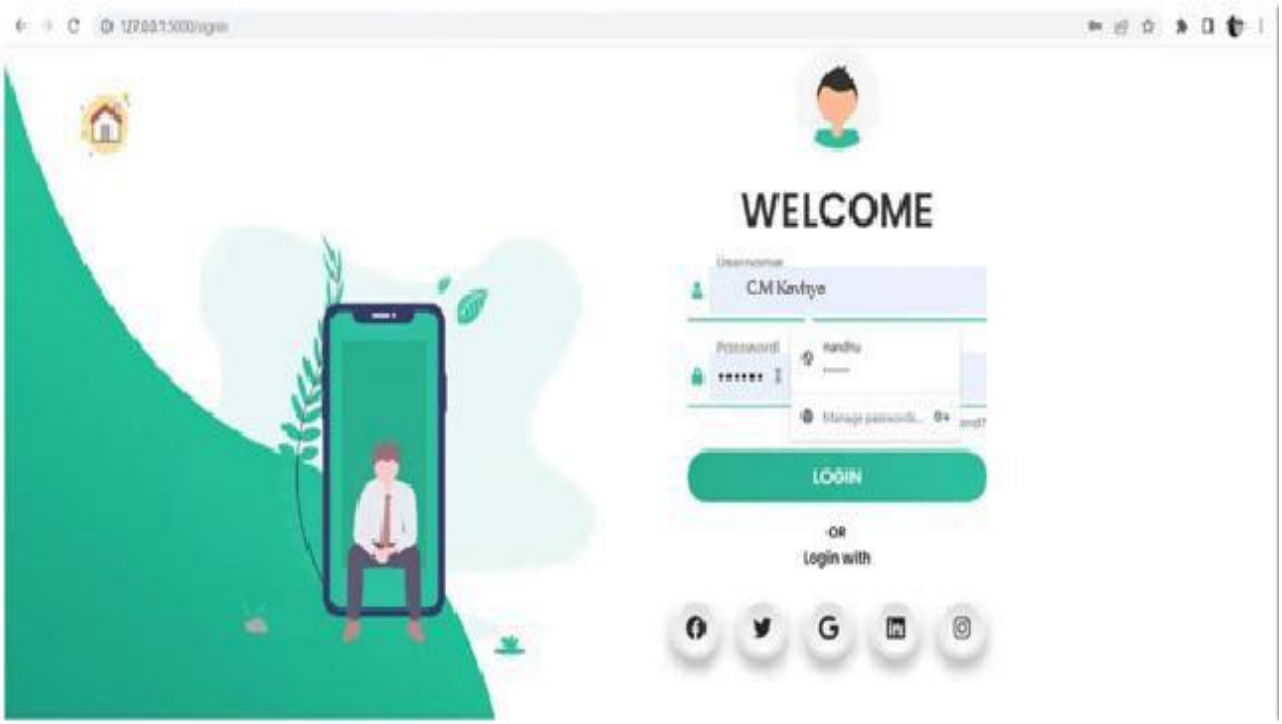
9. RESULTS

Home Page:

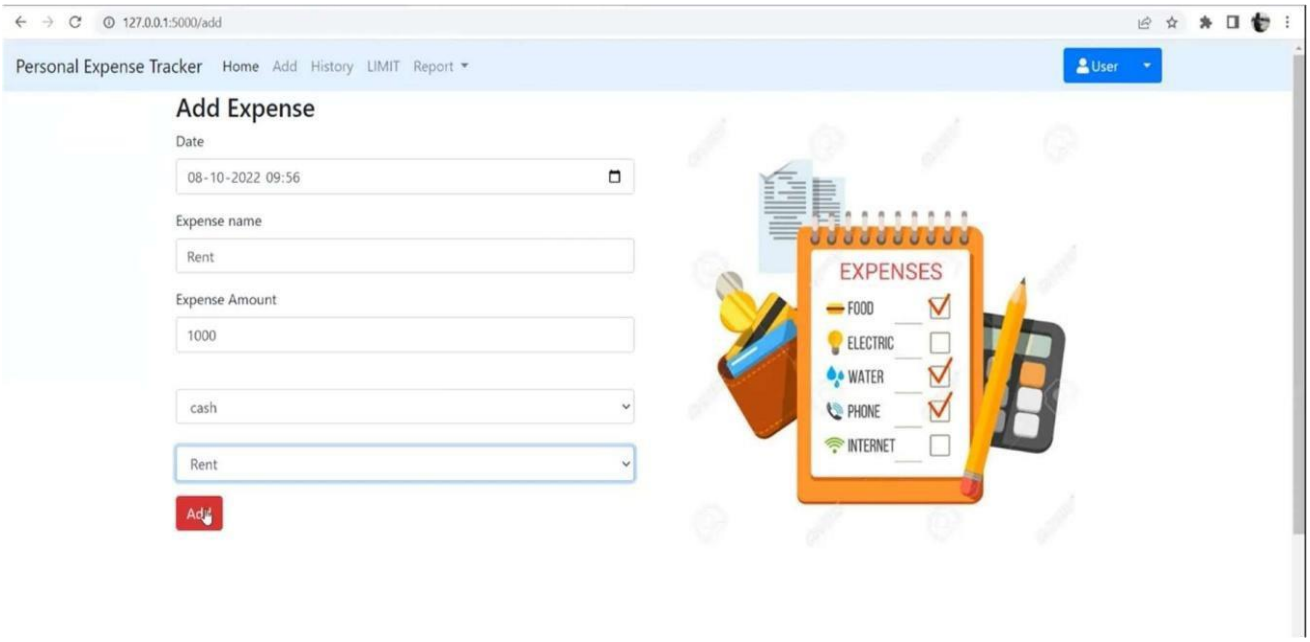
Sign Up Page:



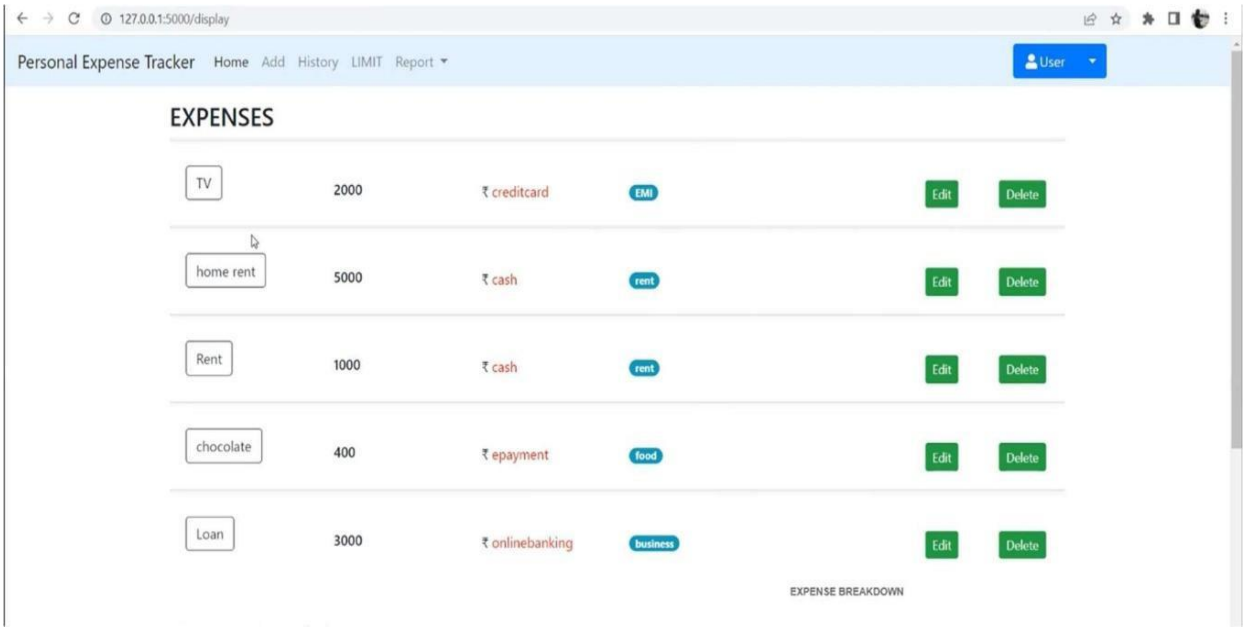
Login Page:



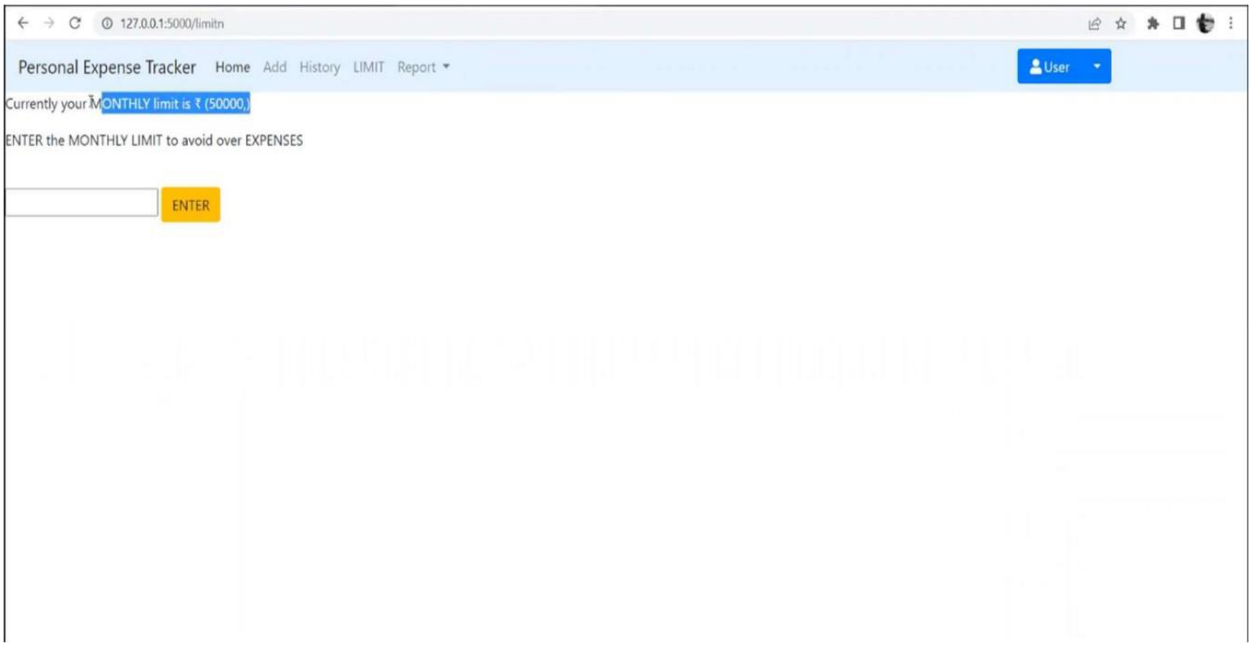
Add Expense Page:



Breakdown of Expenses Page:


127.0.0.1:5000/display
Personal Expense Tracker
Home
Add
History
LIMIT
Report
User
EXPENSES
TV
2000
₹ creditcard
EMI
Edit
Delete
home rent
5000
₹ cash
rent
Edit
Delete
Rent
1000
₹ cash
rent
Edit
Delete
chocolate
400
₹ epayment
food
Edit
Delete
Loan
3000
₹ onlinebanking
business
Edit
Delete
EXPENSE BREAKDOWN

Limit Page:


127.0.0.1:5000/limitn
Personal Expense Tracker
Home
Add
History
LIMIT
Report
User
Currently your MONTHLY limit is ₹ (50000.)
ENTER the MONTHLY LIMIT to avoid over EXPENSES
ENTER

10. ADVANTAGES AND 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. 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.

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. Another con that may occur when spending is being tracked is an error, but this may also be able to be changed.

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

- 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.
- Achieve your business goals with a tailored mobile app that perfectly fits your business.
- Scale-up at the pace your business is growing
- 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.
- Robo Advisors: Get expert investment advice and solutions with the Robo-advisors feature. This feature will analyze, monitor, optimize, and improve diversification in investments by turning data into actionable insights in real-time.
- 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. It brings out the factors causing fluctuations in your expenses.
- Employee Travel Budgeting: Most businesses save money with a travel budgeting app as it helps prepare a budget for an employee's entire business trip. The feature will predict the expenses and allocate resources according to the prediction.

13. APPENDIX

SOURCE CODE GITHUB LINK:

https://github.com /IBM-EPBL/IBM-Project-53852-1661502144

PROJECT DEMO LINK:

https://drive.google.com/drive/folders/1IzTgjcQd85QwOziGrZqoclVS_KF2_33G?usp=share_link