PERSONAL EXPENSE TRACKER APPLICATION

IBM NAALAIYA THIRAN

(TEAM ID: PNT2022TMID35662)

A PROJECT REPORT

Submitted by

GUHAN B (2019503015)

THILAKSURYA B (2019503057)

SANJIV PRAKASH J V (2019503557)

SUHURUTH KC (2019503566)

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE ENGINEERING

ANNA UNIVERSITY MIT CAMPUS

CHROMPET - 600025

NOVEMBER- 2022

BONAFIDE CERTIFICATE

Certified that this project report **"PERSONAL EXPENSE TRACKER APPLICATION"** is the bonafide work of **GUHAN B (2019503015), THILAKSURYA B (2019503057), SANJIV PRAKASH J V (2019503557), SUHURUTH KC (2019503566)** who carried out the **IBM NAALAIYA THIRAN** project work under our supervision.

Industry Mentor	**Faculty Mentor**	**Faculty Evaluator**
(Khushboo, IBM)	**(Dr. P. Jayashree)**	**(Dr. R. Gunasekaran)**

Head of the Department/CSE

(Dr. P. Jayashree)

ACKNOWLEDGEMENT

We take this opportunity to all those who helped towards successful completion of this IBM project. We extend our sincere thanks to our Head of the Department and guide, Dr. P. Jayashree, for her encouragement and guidance. We also thank Dr. R Gunasekaran, Project Evaluator, and Khushboo, IBM Coordinator, for their support throughout this project. We also bound to thanks to all Faculty and Non-teaching staff members of the Department of Computer Science and Engineering whose support and cooperation also contributed much to complete this project work.

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 Project Overview

Mobile applications are the most convenient for users and have surpassed online applications in popularity and usability. Numerous mobile applications exist that offer ways to manage both individual and group spending, but few of them give users a complete picture of both situations. In this article, we design a mobile application for the Android platform that maintains track of the user's expenses, his/her contribution to group expenditures, top investment alternatives, a view of the current stock market, authenticated financial news, and the best continuing offers in the market in popular categories. The proposed application will avoid untidy sticky notes, spreadsheet confusion, and inconsistencies in data management while providing the finest overview of your costs. We may handle their data using our program.

1.2 Purpose

A software or application that helps you keep an accurate record of your money intake and outflow is referred to as an expense tracker. Many individuals in India live on fixed incomes and discover that they don't have enough money left over at the end of the month to cover their expenses.

CHAPTER 2

LITERATURE SURVEY

2.1 Existing problem

The problem with today's population is that they can't remember where all of the money they earned has gone and ultimately have to live while sustaining the little money they have left for their essential needs. At this time, there is no perfect solution that helps a person track their daily expenditure easily and efficiently and notify them about the money shortage they have.

2.2 Reference

1. https://data-flair.training/blogs/expense-tracker-python/
2. https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/
3. https://ijarsct.co.in/Paper391.pdf
4. https://nevonprojects.com/daily-expense-tracker-system/
5. 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

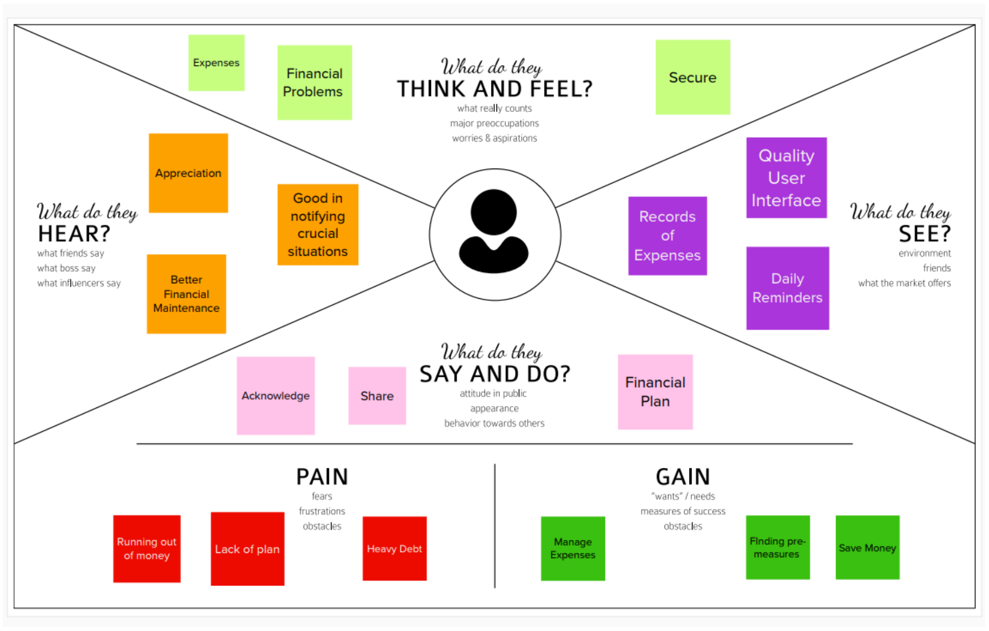
Keeping track of daily expenses and financial activites are tough job. Our daily spending are tracked using a personal expense tracker, which also alerts us through email if we go over budget. To clarify the aforementioned definition, a personal expense tracker is an application that a user may access by providing information about their expenses, based on which their expense wallet balance will be updated and made visible to the user. Additionally, users can receive a graphical analysis of their expenditures. The information we add to the programme is saved in the IBM cloud, where it is possible to update each and every expense. We can establish a cap on the cost. When we go over our daily budget estimate, the primary function of the personal spending tracker kicks in. By integrating the send grid services, the IBM DB, which was developed in Python, sends us an email alert. The Personal Expense Tracker application aids and resolves our monetary issues in this manner.

Keywords: Expense Tracker, budget, planning, savings, budget estimate

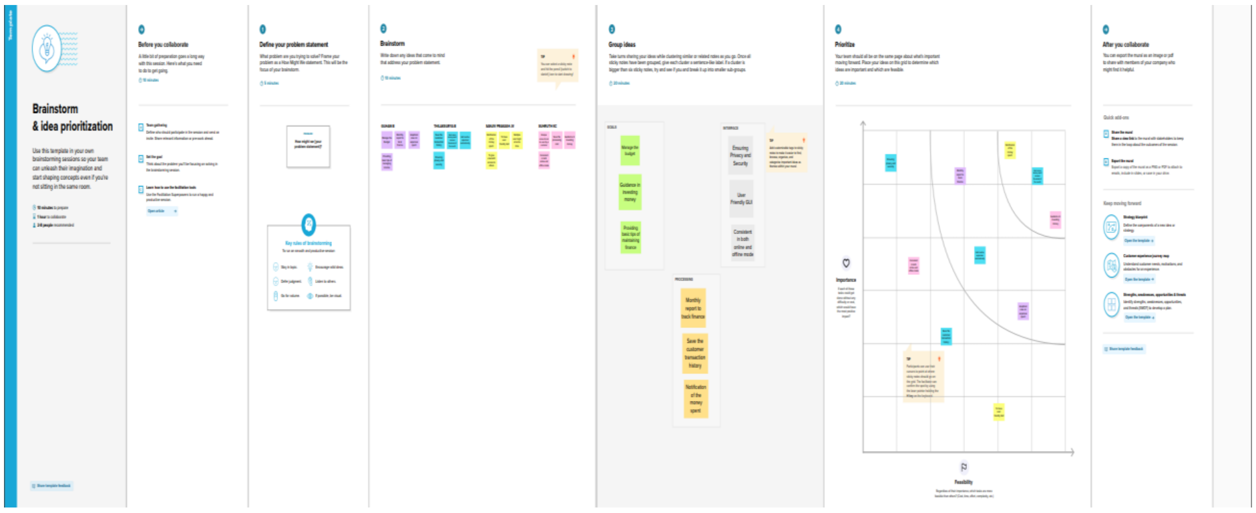
CHAPTER 3

IDEATION & PROPOSED SOLUTION

3.1 Empathy Map canvas



3.2 Ideation & Brainstorming



3.3 Proposed Solution

A cost tracker app allows you to track and categorize your spending across multiple bank and investment accounts, as well as credit cards. Some of these apps also provide budgeting tools, credit monitoring, mileage tracking, receipt keeping, and advice on how to increase your net worth. Without the use of paper receipts, this application maintains track of all of your spending. 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 get regular payments are able to track their payments and avoid 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.

- Persons from the middle class track their income and expenses with a range of methods, such as paper and ink, spreadsheets, and budgeting apps.
- Customers may effortlessly keep track of their earnings and expenses to prevent receiving loans and money from money lenders during critical situations.
- Novelty / Uniqueness Notification can be received through email.
- Social Impact / Customer Satisfaction Using this application one can track their personal expenses and frame a monthly/annual budget. If your expense exceeds the specified limit, the application will show you an

alert message. This will make an impact on Mobile Banking for Customers' Satisfaction.

- Business Model (Revenue Model) Business people can use the 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.

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

- FR-1 User Registration, Form for collecting details
- FR-2 User Confirmation, Confirmation via Email Confirmation via OTP
- FR-3 Collection and verification of account detail
- FR-4 Alert Message Consists of expenditure details and other data records
- FR-5 Category Sending alerts when the expense limit has reached and for system access

4.2 Non-Functional requirement

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

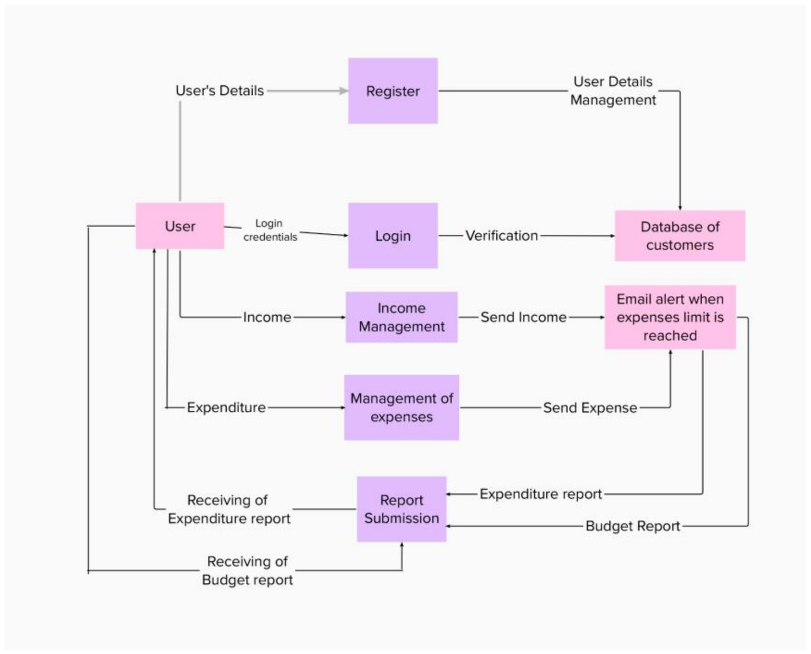
- NFR-1 Usability By using this application, the user can keep track of their expenses so that their hard-earned money does not get wasted
- NFR-2 Security More security of the customer data and bank account details.
- NFR-3 Reliability Maintaining a proper tracking of day-to-day expenses in an efficient manner

- NFR-4 Performance The software will help the user to monitor the flow of cash with at most quality and security with high performance
- NFR-5 Availability Using charts and graphs to help the user to check the budgeting and assets so that it would be easily visualized

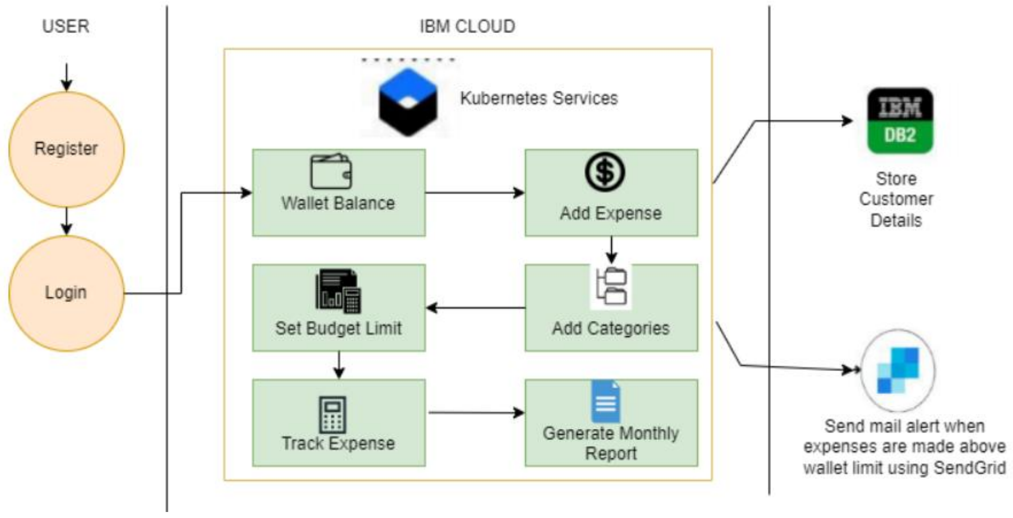
CHAPTER 5

PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

Use the below template 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, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint 1
	Login	USN-2	As a user, I can login to the application by entering the login credentials.	I can access the application	High	Sprint 1
	Dashboard	USN-3	As a user, I can enter the required details of the income and the money spent.	I can view my daily expenses	Medium	Sprint 2
		USN-4	As a customer care executive, I will be able to solve the issues faced by the user when using the application	I can provide customer support 24/7	Low	Sprint 1
	Application	USN-5	As an administrator, I will be able to upgrade or update the application.	I can fix the bugs and issues which is raised by the customers and users of the application	Medium	Sprint 1

CHAPTER 6

PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Sprint	**Functional Requirement (Epic)**	**User Story Number**	**User Story/Task**	**Story Points**	**Priority**	**Team Members**
Sprint- 1	Registration	USN-1	As user, I can register for the application by entering my email, password, and confirming my password.	2	High	Guhan B
Sprint- 1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	Medium	Thilaksurya B
Sprint- 2	Login	USN-3	As a user, I can register for the application through Facebook	2	Medium	Sanjiv Prakash J V
Sprint- 1	Dashboard	USN-4	As a user, I can register for the application through Gmail	2	High	Suhruth K C

6.2 Sprint Delivery Schedule

Sprint	**Total Story Points**	**Duration**	**Sprint Start Date**	**Sprint End Date (Planned)**	**Story Points Completed (as on Planned End Date)**	**Sprint Release Date (Actual)**
Sprint-1	20	6 Days	23Oct2022	28Oct2022	20	29Oct2022
Sprint-2	20	6 Days	30Oct2022	04Nov2022	20	05Nov2022
Sprint-3	20	6 Days	06Nov2022	11Nov2022	20	12Nov2022
Sprint-4	20	6 Days	13Nov2022	18Nov2022	20	19Nov2022

CHAPTER 7

Coding And Solutioning

7.1. Features

Feature 1: Add Expense

Feature 2: Update expense

Feature 3: Delete Expense

Feature 4: Set Limit

Feature 5: Send Alert Emails to users

7.2. Other Features:

Anywhere and anytime, keep track of your spending. Manage your finances and budget without using any paper-based records. Simply click to submit your bills and spending. No matter the time or place, you can access, send, and approve invoices. Scan your receipts and tickets, then save them in the app to prevent data loss. Real-time bill and expense approval with prompt notification. With an automated and streamlined invoicing procedure, claims may be settled quickly and human error is reduced.

Codes:

```
import os
import ibm_db
import re
import ibm_db_dbi

from flask import Flask, render_template, request, redirect, session
from flask_db2 import DB2
# from sendemail import sendgridmail,sendmail

app = Flask(__name__)
```

```
app.secret_key = 'a'

app.config['database'] = 'bludb'
app.config['hostname'] = 'fbd88901-ebdb-4a4f-a32e-
9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud'
app.config['port'] = '32731'
app.config['protocol'] = 'tcpip'
app.config['uid'] = 'qdp46216'
app.config['pwd'] = 'MGhHNGxutNYPFPfE'
app.config['security'] = 'SSL'

try:
    mysql = DB2(app)

    conn_str = 'database=bludb;hostname=fbd88901-ebdb-4a4f-a32e-
9822b9fb237b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;port=32731
;protocol=tcpip;uid=qdp46216;pwd=MGhHNGxutNYPFPfE;security=SSL'

    # conn_str = 'database=bludb;hostname=ba99a9e6-d59e-4883-8fc0-
d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;port=31321;
protocol=tcpip;uid=vmk08423;pwd=3KfJl6HGDtPdbIWy;security=SSL'

    ibm_db_conn = ibm_db.connect(conn_str,'','')
except Exception as e:
    print("IBM DB Connection error:", e)


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

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

@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']

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

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

        while dictionary != False:
            dictionary = ibm_db.fetch_assoc(res)
        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)

@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)
        account = ibm_db.fetch_row(stmt)

        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)

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

@app.route('/addexpense',methods=['GET', 'POST'])
def addexpense():
    date = request.form['date']
    expensename = request.form['expensename']
    amount = request.form['amount']
    paymode = request.form['paymode']
    category = request.form['category']
    p1 = date[0:10]
    p2 = date[11:13]
    p3 = date[14:]
    p4 = p1 + "-" + p2 + "." + p3 + ".00"
    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)

    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. " + str(s) + "/- !!!" + "\n" + "Thank you, " + "\n" +
"Team Personal Expense Tracker."
        # sendmail(msg,session['email'])

    return redirect("/display")

@app.route("/display")
def display():
    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)
        dictionary = ibm_db.fetch_assoc(res)

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

@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)
    return redirect("/display")

@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)
        dictionary = ibm_db.fetch_assoc(res)

    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")
```

```
@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)
```

```
@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)
        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]

        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]
      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["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]
    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('/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

CHAPTER 8

TESTING

8.1. TESTING:

- Login Page (Funcional)
- 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

Resolution	**Severity 1**	**Severity 2**	**Severity 3**	**Severity 4**	**Subtotal**
By Design	12	3	1	7	23
Duplicate	1	0	4	0	4
External	2	3	0	2	7
Fixed	7	3	3	11	24
Not Reproduced	0	0	1	0	1

Skipped	0	0	1	1	2
Won't Fix	0	5	0	1	8
Totals	22	14	10	22	68

3. Test Case Analysis

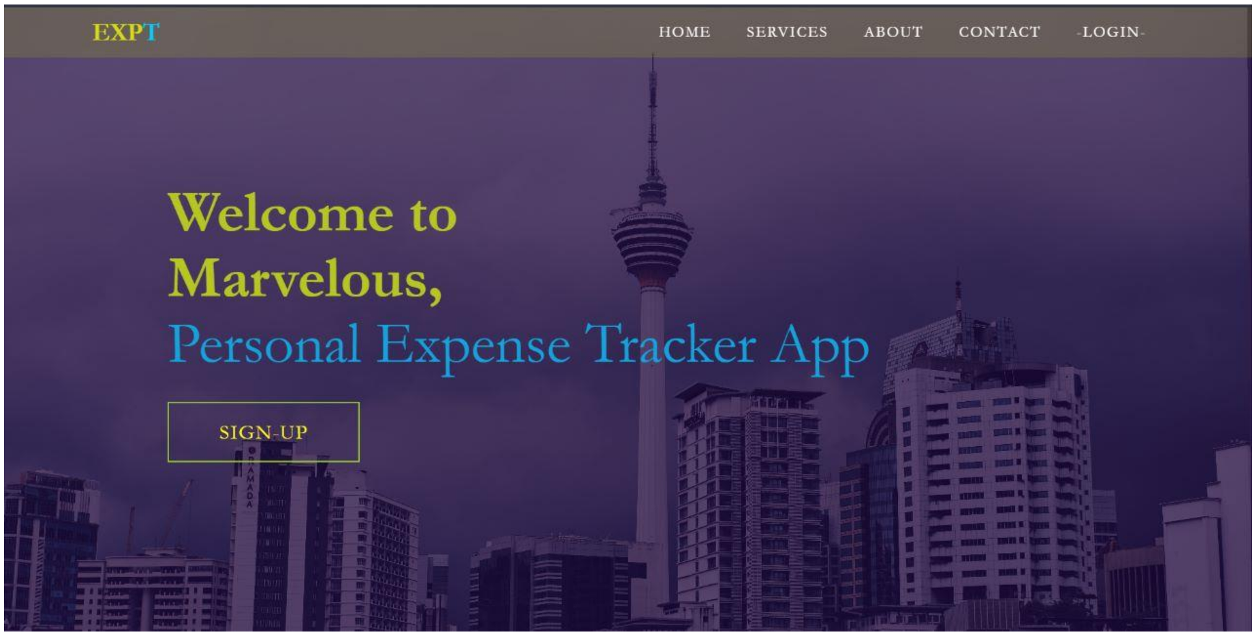
This report shows the number of test cases that have passed, failed, and untested

Section	**Total Cases**	**Not Tested**	**Fail**	**Pass**
Interface	6	0	0	6
Login	41	0	0	41
Logout	3	0	0	3

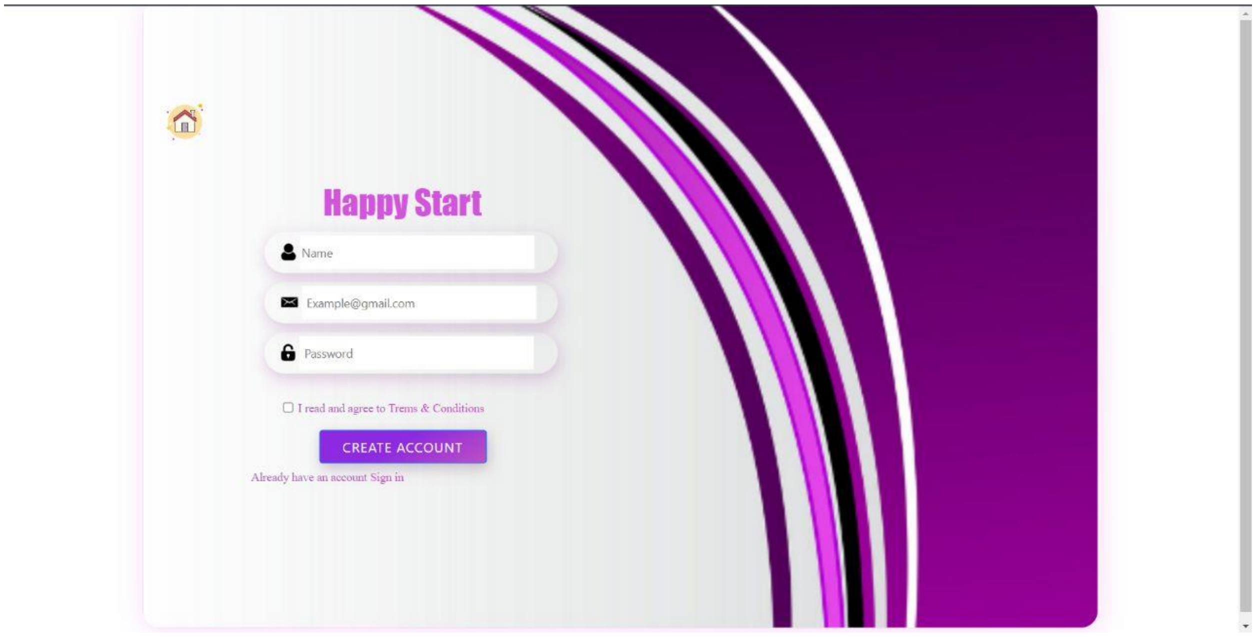
CHAPTER 9

RESULTS

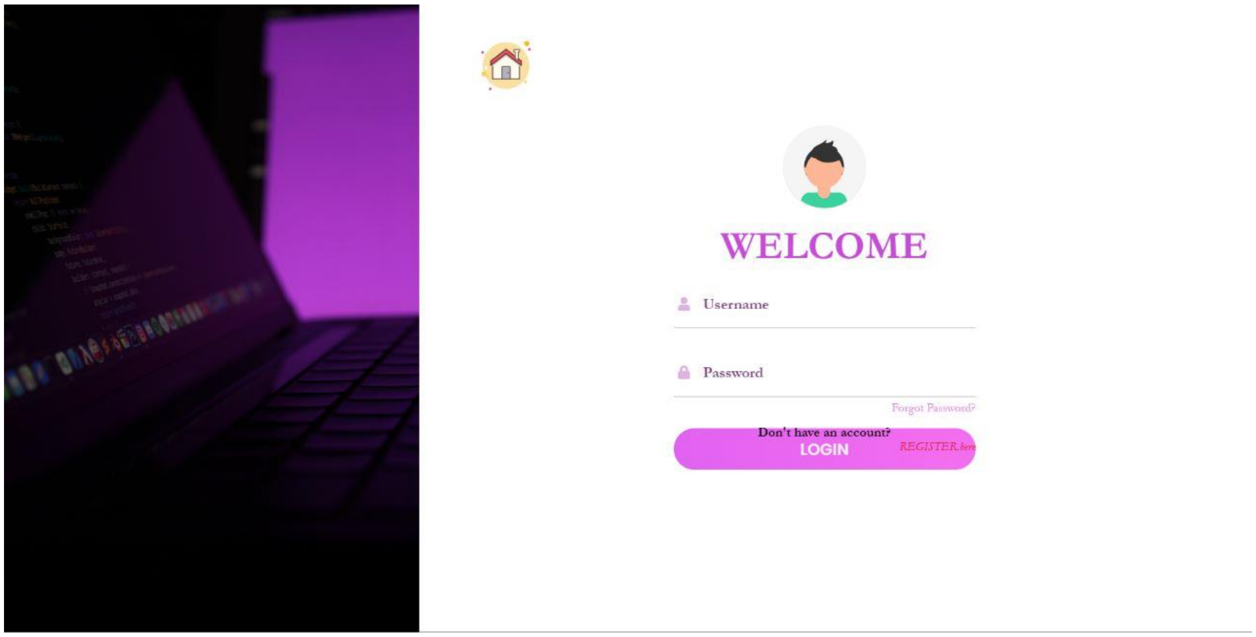
9.1 Home Page



9.2 Sign Up Page


Happy Start
Name
Example@gmail.com
Password
I read and agree to Trems & Conditions
CREATE ACCOUNT
Already have an account Sign in

9.3 Login Page



CHAPTER 10

ADVANTAGES AND DISADVANTAGES

10.1. ADVANTAGES:

One of the major benefits of spending tracking is that one is always aware of the state of one's personal finances. Tracking your spending can help you stick to your budget not only in general, but also in specific categories such as housing, food, transportation, and gifts. While manually tracking all cash spent can be inconvenient and time-consuming, doing so automatically can be quick and simple. Another advantage is that many automatic spending-tracking software programs are free. Having the program on a handheld device is a major advantage because it can be checked before spending to ensure that the available budget is not exceeded. Another advantage is that people who like to keep track of their expenditures by hand with paper and pen or by inputting data into a computer spreadsheet can do so. Some people retain a file folder or box in which they keep receipts and track how much money they spend each day. One advantage of using this easy daily monitoring technique is that it can help you become more aware of where your money is going before the conclusion of a pay period or month.

10.2. DISADVANTAGES:

A disadvantage of any system used to track spending is that it may be started, then tapered off until it is completely forgotten. However, this is a risk with any new goal, such as attempting to lose weight or stop smoking. The tracking goal can be useful if a person first creates a budget and then saves money before spending any new pay period or month. Tracking spending and ensuring all receipts are accounted for only needs to be done once or twice a month in this manner. Even with constant tracking of one's spending habits, financial goals are not guaranteed to be met. Although this can be regarded a disadvantage of spending tracking, it can be turned into a benefit if one is determined to continue attempting to handle all finances appropriately. Another disadvantage of tracking expenditure is the possibility of inaccuracy, which can be turned into a benefit if the person undertakes regular tracking. Frequent cash expenditure tracking allows one to detect and repair faults, allowing the budget plan to be followed despite the inaccuracy.

CHAPTER 11

CONCLUSION

Clarity and conviction in judgment are necessary for the implementation of a thorough money management strategy. To understand your business and personal finances, you will need a specific objective and a distinct vision. An expense tracking app enters the picture at that point. A specialized package of services called expense tracking software is available to those who want to effectively manage their income and budget their spending and savings. It enables you to keep track of all transactions on a daily, weekly, and monthly basis, including bills, refunds, wages, receipts, taxes, etc.

CHAPTER 12

FUTURE SCOPE

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

APPENDIX:

The source code and outputs are in the link (GitHub)