PROJECT REPORT

PERSONAL EXPENSE TRACKER

TEAM ID: PNT2022TMID53373

TEAM MEMBERS:

NAME	ROLL NUMBER
PRATEEP NS	2127190501091
KARTHIKEYAN JV	2127190501055
SUNIL KUMAR Y	2127190501310
KIRUPHANIDHI	2127190501308

INDEX

1. **INTRODUCTION**
    1. Project Overview

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

    2. Purpose

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

2. **LITERATURE SURVEY**
    1. Existing problem

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

    2. References

• https://nevonprojects.com/daily-expense-tracker-system/
• https://data-flair.training/blogs/expense-tracker-python/

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

• https://ijarsct.co.in/Paper391.pdf

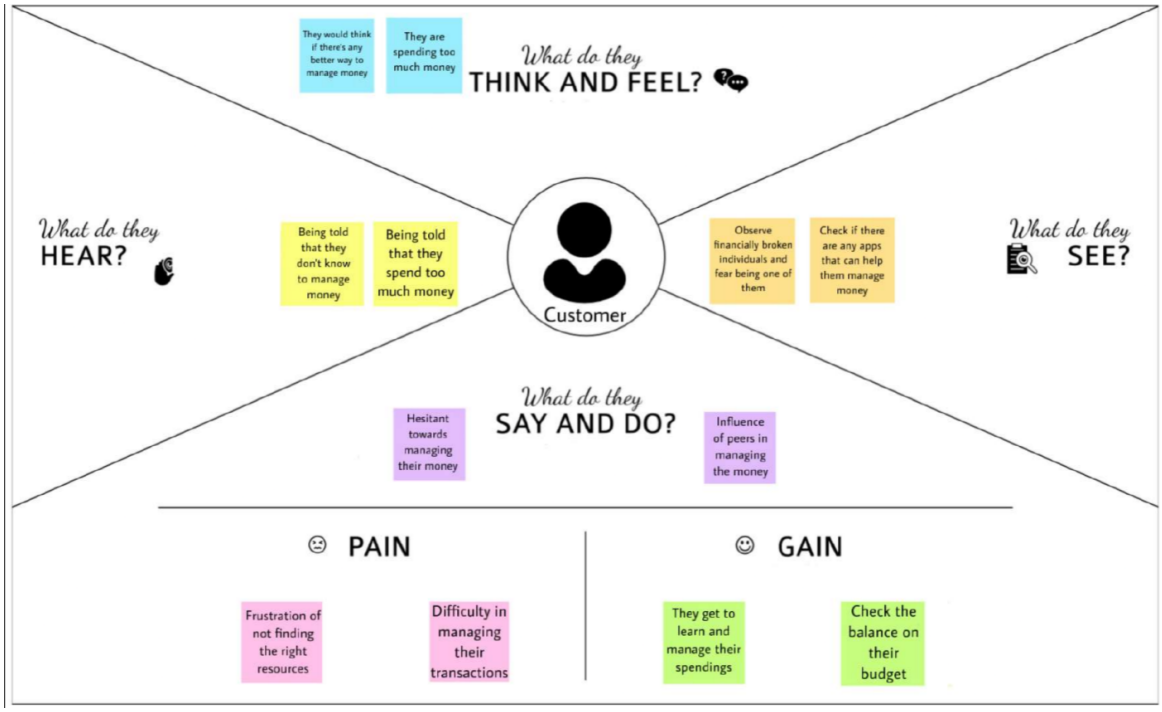
• https://kandi.openweaver.com/?landingpage=python_all_projects&utm_source=google&utm_medium=cpc&utm_campaign=promo_kandi_ie&utm_content=kandi_ie_search&utm_term=python_devs&gclid=Cj0KCQiAgribBhDkARIsAASA5bukrZgbI9UZxzpoyf0PofB1mZNxzcokUP3TchpYMclHTYFYiqP8aAmmwEALw_wcB

3. Problem Statement Definition

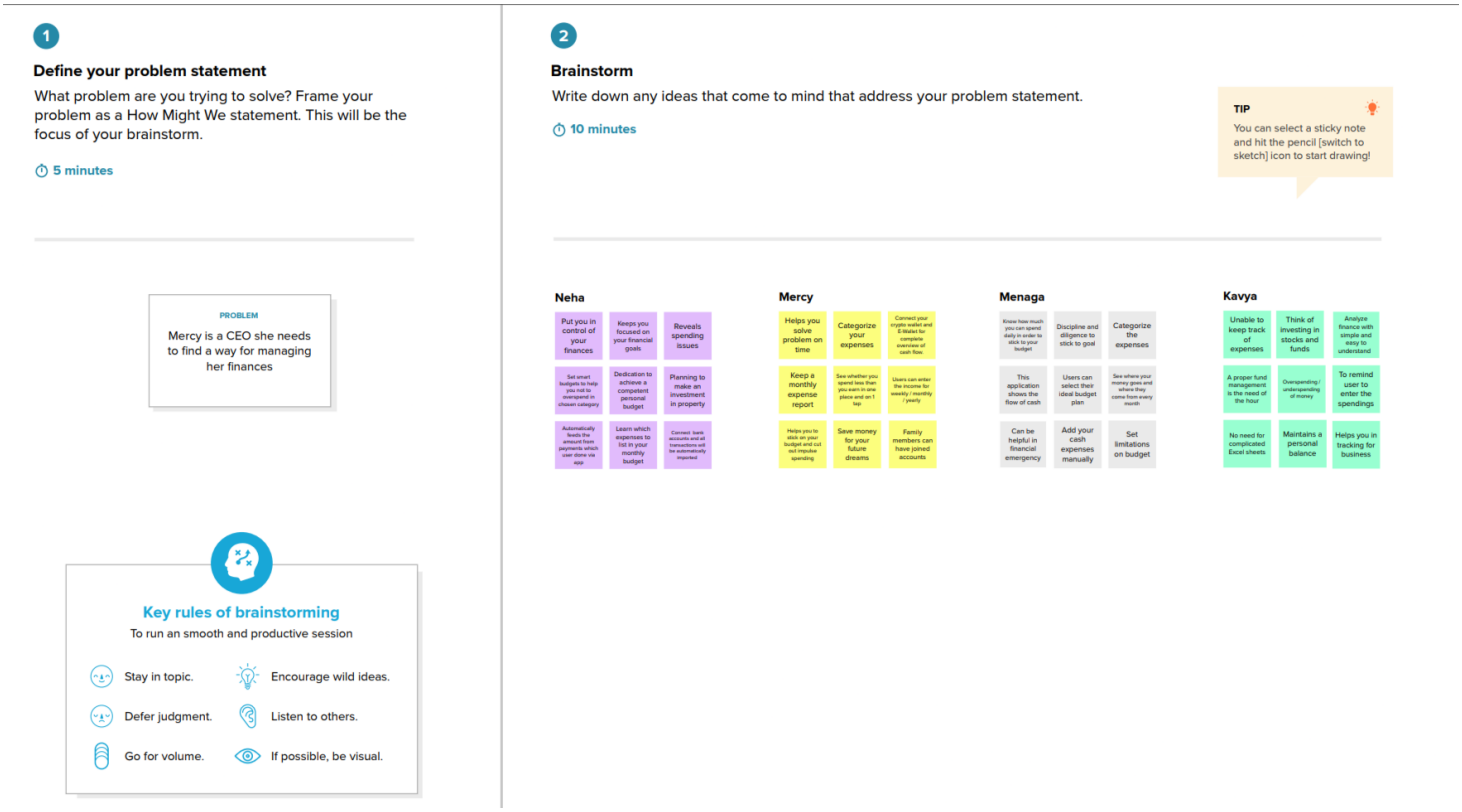
This Expense Tracker is a web application that facilitates the users to keep track and manage their personal as well as business expenses. This application helps the users to keep a digital diary. It will keep track of a user's income and expenses on a daily basis. The user will be able to add his/her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

3. IDEATION & PROPOSED SOLUTION

1. Empathy Map Canvas



2. Ideation & Brainstorming

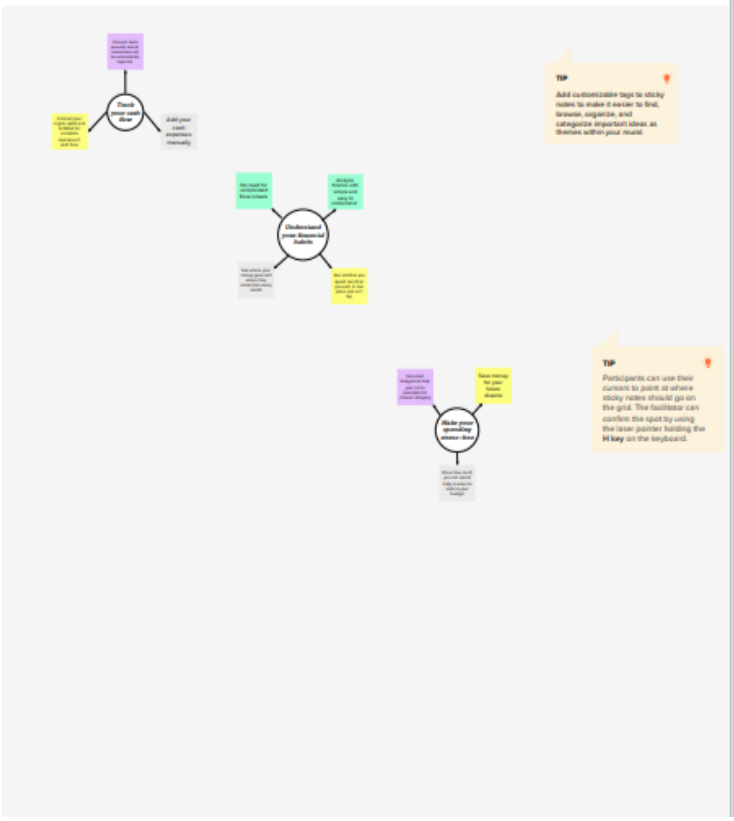


3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes



4

Prioritize

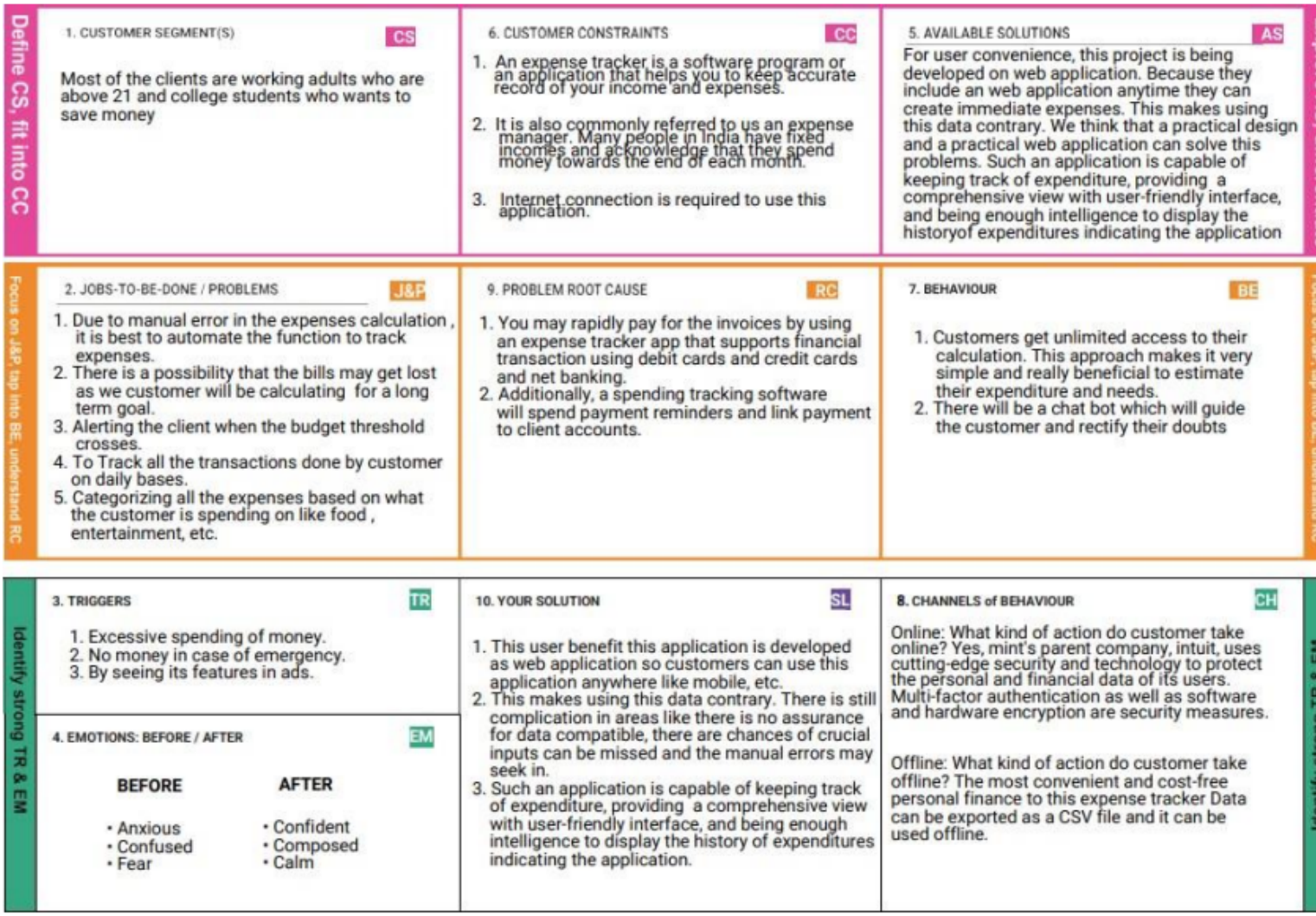
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

3. Proposed Solution

S.No.	Parameter	Description
1.	**Problem Statement (Problem to be solved)**	At the moment, there is no such simple or free solution available that allows a person to effortlessly keep track of his or her daily expenses. In order to accomplish this, a person must maintain a journal in a diary or on a computer. Additionally, all computations must be performed by the user, which might occasionally result in mistakes that result in losses. Because there is no comprehensive tracking system, it is constantly burdensome to rely on the daily entry of expenditures and total estimates through the end of the month.
2.	**Idea / Solution description**	We're going to create a web application to make it simple to track spending and to provide useful insights into money management in order to manage expenses effortlessly.
3.	**Novelty / Uniqueness**	When the user's spending limit is exceeded, they will receive emails and text messages, and if they neglect to enter their expenses, a remainder will be set. Additionally, we will add automated ideas that will aid in budget planning, and we will group spending according to categories like entertainment, shopping, etc.
4.	**Social Impact / Customer Satisfaction**	It will aid consumers in keeping track of their spending and warn them when they go over their budget's allotment.
5.	**Business Model (Revenue Model)**	We can provide the application in a monthly subscription plan.
6.	**Scalability of the Solution**	This application can handle large number of users.

4. Problem Solution fit



4. REQUIREMENT ANALYSIS

1. Functional requirement

Functional Requirements:

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Phone number. Registration through Gmail. Registration through Username and Gmail.
FR-2	User Confirmation	Confirmation via Email. Confirmation via OTP.
FR-3	User Login	Login using Gmail. Login through Username.
FR-4	Manage Expenses	Create or update new budget/expense limit. Manage expenses by categorizing the priority ones.
FR-5	Expense Tracker	Analyze the level of expenses in graphical report format and graphical representation of expenses based on daily, monthly, yearly usage and categorize the based on what customer is using for.
FR-6	Manage income and expenditure	Create or update income and expenditure details, then the app suggests better ideas for budgeting. Provides built-in plans for some certain budget goals.

2. Non-Functional requirements

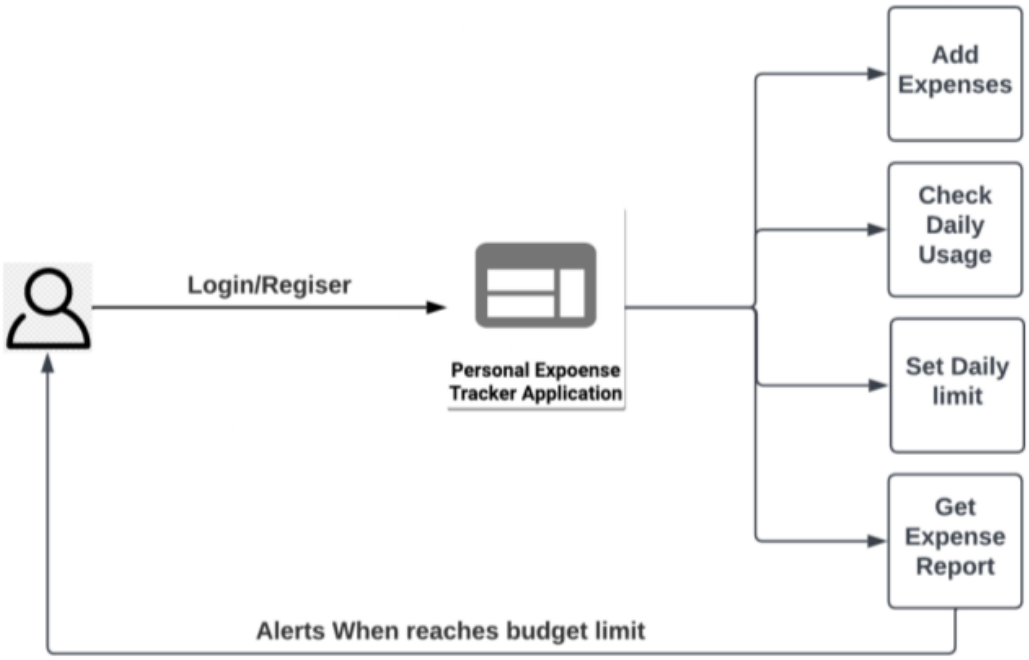
Non-functional Requirements:

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

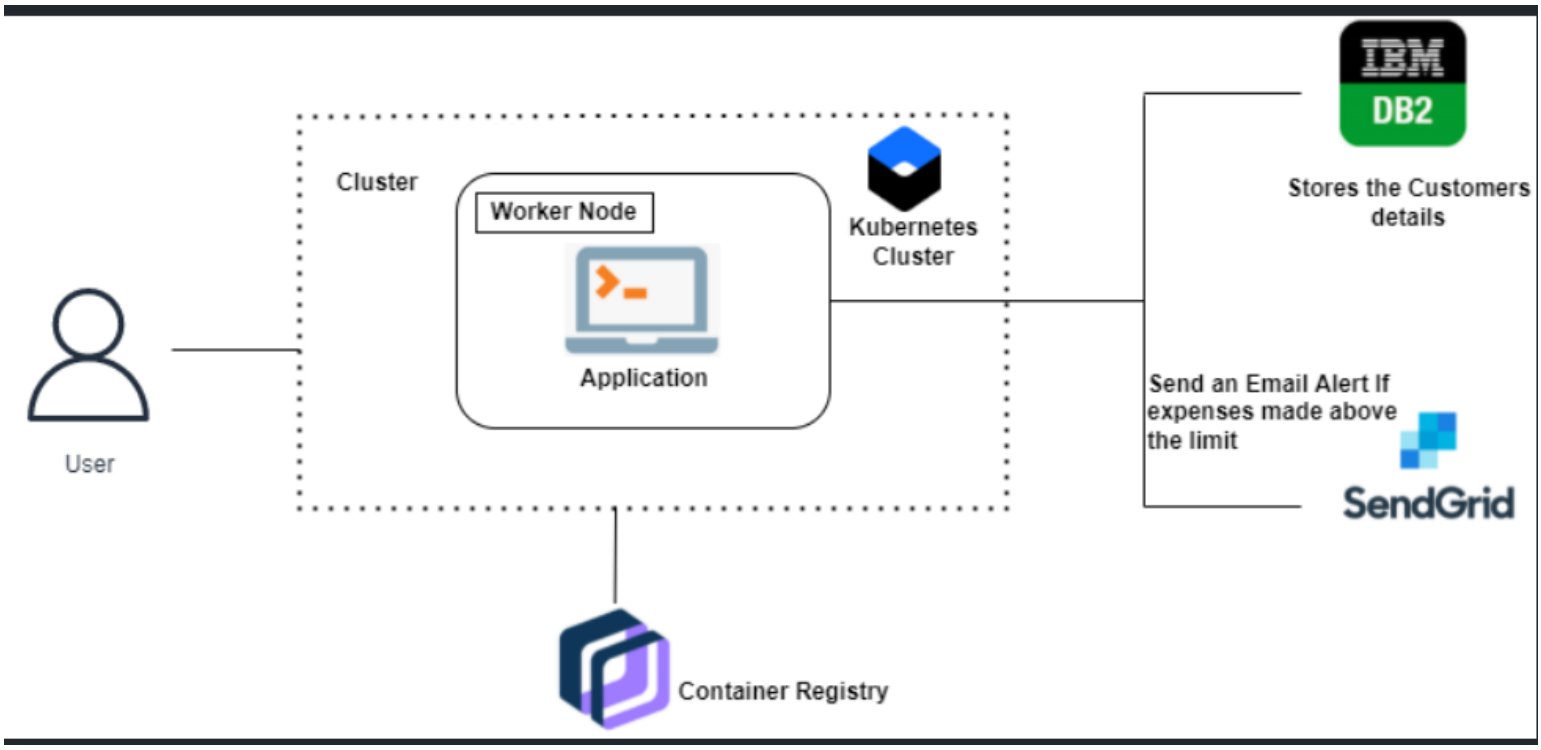
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	This system will be used by anyone who needs to manage their expenses and to make better budgeting ideas.
NFR-2	**Security**	This system prevents customer's data securely and protects from malware attacks or unauthorized access.
NFR-3	**Reliability**	This system is highly reliable and it reduces the manual work load.
NFR-4	**Performance**	It tracks the expenses and generates reports quickly. It engages users efficiently with better budgeting ideas.
NFR-5	**Availability**	User can make his/her reports offline and this report is operational at any time.
NFR-6	**Scalability**	This system has better storage capacity and it manages large no of user's data.

5. PROJECT DESIGN

1. Data Flow Diagrams



2. Solution & Technical Architecture



3. User Stories

User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user/ Web user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		USN-2	As a user, I will receive confirmation email once I have registered for the application	I can receive confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Gmail / Phone Number	I can confirmation using OTP	Medium	Sprint-2
Customer (Web user/ Mobile User)	Login	USN-4	As a user, I can log into the application by entering email / username & password	I can access my account / dashboard	High	Sprint-1
Customer (Web user/ Mobile User)	Dashboard	USN-5	As a user, I can register for the application by Bank account number or using UPI id or manually upload expense or upload csv file	I can access my account / dashboard	Medium	Sprint-2
		USN-6	As a user, I will receive confirmation mail and OTP, once I have registered for the application	I can confirmation using OTP	High	Sprint-3
Customer (Mobile user)		USN-7	As a user, I can use all the services provided by the application in both online and offline		Medium	Sprint-2
Customer (Web User)		USN-8	As a user, I can use all the services provided by the application only in offline and platform independent		Medium	Sprint-3
Customer Care Executive	Customer support	USN-9	As a user, I can support for customers to automatic logging of recurring transactions.		High	Sprint-3
Administrator	Responsibility	USN-10	As a system administrator, provide security to the data and provide support to the user		High	Sprint-3

6. PROJECT PLANNING & SCHEDULING

1. Sprint Planning & Estimation

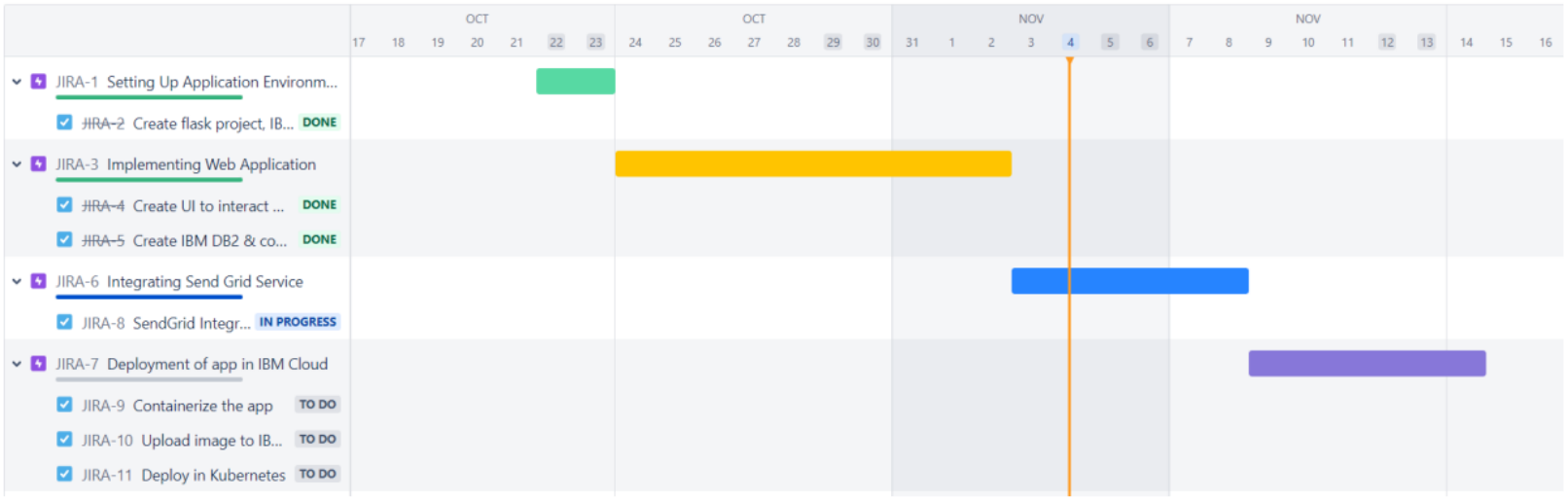
Sprintnumber	Sprint	FunctionalRequirement (Epic)	UserStory Number	UserStory/Task	StoryPoints (Total)	Priority	TeamMembers
1	Setting UpApplicationEnvironment	Create flask project,IBM account,Dockerand SendGrid	USN-1	Getting started with flask, creating IBM cloudaccountand installingIBMcloudCLI,docker	5	High	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.
2	Implementing WebApplication	CreateUltointeractwithapplication	USN-2	Designing user friendly UI to make humancomputerinteractionsmoothly.Homepage,login, log out, expense report page are theconsole.	20	High	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.
		Create IBM DB2andconnect withpython	USN-3	Creating IBM DB2 connecting it with flask MVCmicroframeworktostoretheusercredentialsandexpenses.	5	High	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.

3	Integrating SendGrid Service	SendGridintegrationwithpythoncode	USN-4	GeneratingSendGridAPIEtonotifyuserthatheyreachthe limitofspending.	5	medium	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.
4	Deployment ofapp inIBMCloud	Containerizethe app	USN-1	Creatingdockerfileandtestingtherequirementsforthe flaskapplication.	5	medium	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.
		Uploadimagetol BM containerregistry	USN-2	Uploadimagesusedinthefrontendto Containerregistry.	3	medium	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.
		Deploy inKubernetes	USN-3	Deploytheapplicationaftertestingitsfunctionality on Kubernetes.	7	High	Prateep, Karthikeyan, Sunilkumar, Kirupanidhi.

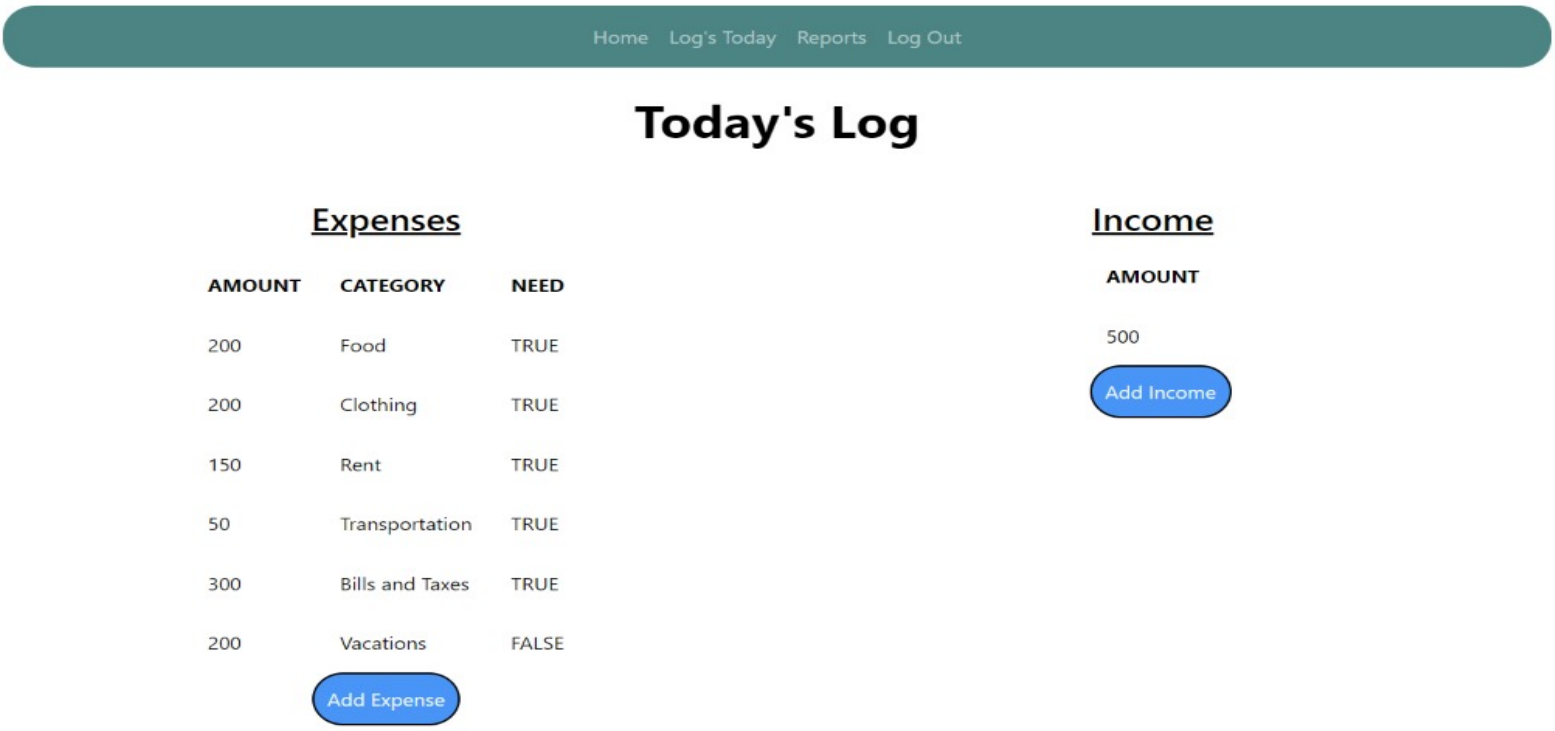
2. Sprint Delivery Schedule

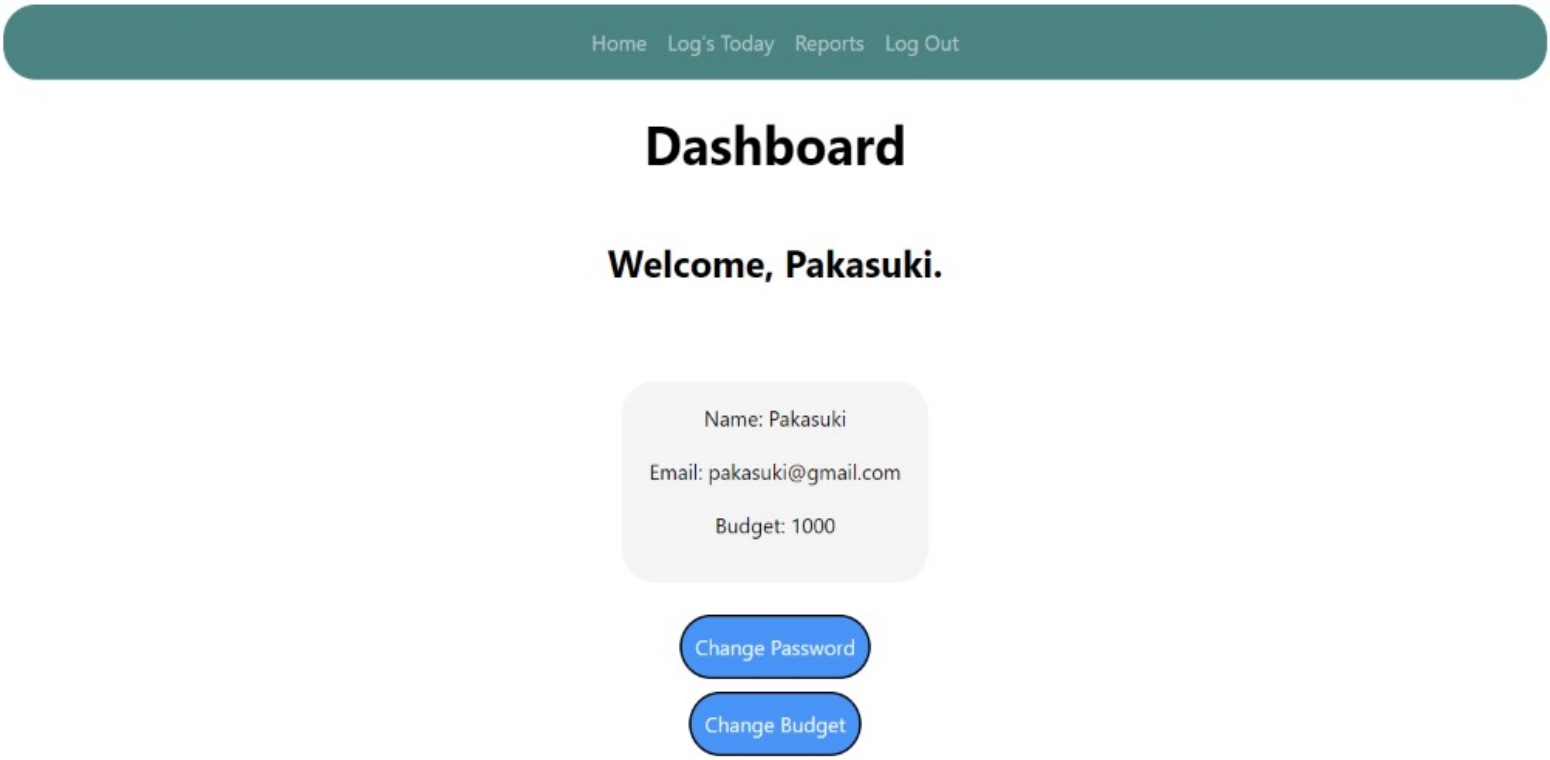
Sprint	Total StoryPoints	Duration	SprintStartDate	SprintEndDate(Planned)	Story PointsCompleted (as onPlannedEndDate)	SprintReleaseDate(Actual)
Sprint-1	5	2Days	22Oct2022	23Oct2022	5	25Oct2022
Sprint-2	25	10Days	24Oct2022	02Nov2022	20	02Nov2022
Sprint-3	5	6 Days	03Nov2022	08Nov2022	5	08Nov2022
Sprint-4	15	6Days	09Nov2022	14Nov2022	15	15Nov2022

3. Reports from JIRA



7. **CODING & SOLUTIONING**
    1. Update Expense
    2. Add Income
    3. Change Budget





8. **TESTING**

1. Test Cases

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	10	0	0	10
Client Application	50	0	0	50
Security	1	0	0	1
Outsource Shipping	3	0	0	3

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	3	1	2	16
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	13	12	25	74

2. User Acceptance Testing

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

Exception Reporting	8	0	0	8
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9. **RESULTS**

1. Performance Metrics

● Tracking income and expenses: Monitoring the income and tracking all expenditures (through bank accounts, mobile wallets, and credit & debit cards).

● 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 the appropriate 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 sendsremindersfor 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.,

● E-commerce integration: Integrate your expense tracking app with your eCommerce store and track your sales through payments received via multiple payment methods.

● Vendors and Contractors: Manage and track all the payments to the vendors and contractors added to the mobile app.

● Access control: Increase your team productivity by providing access control to particular users through custom permissions.

● Track Projects: Determine project profitability by tracking labor costs, payroll, expenses, etc., of your ongoing project.

● Inventory tracking: An expense tracking app can do it all. Right from tracking products orthe cost of goods, sending alert notifications when the product is running out of stock or the product is not selling, to purchase orders.

● In-depth insights and analytics: Provides in-built tools to generate reports with easy-to- understand visuals and graphics to gain insights about the performance of your business.

● Recurrent Expenses: Rely on your budgeting app to track, streamline, and automate all the recurrent expenses and remind you on a timely basis.

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

One of the major pros of tracking spending is always being aware of the state of one's personal finances. Tracking what you spend can help you stickto 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 trackingall cash that is spent can be irritating as well as time consuming, a pro is that doing this automatically can be quick and simple. Another pro is that many automatic spending tracking software programs are available for free. Having the program on a hand-held device can be a main pro since it can be checked before spending occurs in order to be sure of the available budget.

DISADVANTAGES:

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

11. CONCLUSION

From this project, we are able to manage and keep tracking the daily expenses as well as income. While making this project, we gained a lot of experience of working as a team. We discovered various predicted and unpredicted problems and we enjoyed a lot solving them as a team. We adopted things like video tutorials, text tutorials, internet and learning materials to make our project complete.

12. FUTURE SCOPE

The project assists well to record the income and expenses in general.However, this project has some limitations:

- The application is unable to maintain the backup of data once it isuninstalled.
- This application does not provide higher decision capability.

To further enhance the capability of this application, we recommend the followingfeatures to be incorporated

into the system:

- Multiple language interface.
- Provide backup and recovery of data.
- Provide better user interface for user.
- Mobile apps advantage.

13. APPENDIX

Source Code

```
from flask import (
    Flask,
    render_template,
    send_file,
    request,
    redirect,
    url_for,
```

```
    session,
    flash,
)
import ibm_db
import re
from matplotlib import pyplot as plt
from matplotlib.backends.backend_agg import FigureCanvasAgg as FigureCanvas
from matplotlib.figure import Figure
from io import BytesIO

app = Flask(__name__)
app.secret_key = "Zenik"

conn = ibm_db.connect(
    "DATABASE=bludb;"
                                                    "HOSTNAME=2d46b6b4-cbf6-40eb-bbce-
     6251e6ba0300.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;"
    "PORT=32328;"
    "SECURITY=SSL;"
    "SSLServerCertificate=DigiCertGlobalRootCA.crt;"
    "UID=fpj20933;"
    "PWD=ELH6dqXE1OBE0MGC;",
    "",
    "",
)



@app.route("/", methods=["POST", "GET"])
@app.route("/home")
def home():
    return render_template("home.html")
```

```
@app.route("/login", methods=["GET", "POST"])
def login():
    msg = ""
    if request.method == "POST":
        username = request.form["username"]
        password = request.form["password"]
         sql = "SELECT clients.*,budgets.MAXBUDGET FROM clients LEFT JOIN BUDGETS ON
     CLIENTs.ID=BUDGETS.ID WHERE username =? AND password =?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        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["ID"]
            session["email"] = account["EMAIL"]
            session["username"] = account["USERNAME"]
            session["budget"] = account["MAXBUDGET"]
            print(session["Loggedin"])
            return redirect("/dashboard")
        else:
            msg = "Incorrect login credentials"
    flash(msg)
    return render_template("login.html", title="Login")



@app.route("/register", methods=["GET", "POST"])
```

```
def register():
    msg = ""
    if request.method == "POST":
        username = request.form["username"]
        email = request.form["email"]
        password = request.form["password"]
        password1 = request.form["password1"]
        sql = "SELECT * FROM CLIENTS WHERE username =? or email=? "
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, email)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = "Account already exists"
        elif password1 != password:
            msg = "re-entered password doesnt match"
        elif not re.match(r"[A-Za-z0-9]+", username):
            msg = "Username should be only alphabets and numbers"
        else:
            sql = "INSERT INTO clients(EMAIL,USERNAME,PASSWORD) VALUES (?,?,?)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, email)
            ibm_db.bind_param(stmt, 2, username)
            ibm_db.bind_param(stmt, 3, password)
            ibm_db.execute(stmt)
            return redirect("/dashboard")
    flash(msg)
    return render_template("register.html", title="Register")
```

```
@app.route("/logout")
def logout():
    session.clear()
    return redirect("/")


def isLogged():
    return session["Loggedin"]


@app.route("/dashboard")
def dashboard():
    if isLogged:
        return render_template("dashboard.html", title="Dashboard")
    else:
        flash("Login to go to dashboard")
        return redirect("/login")


@app.route("/changePassword/", methods=["POST", "GET"])
def changePassword():
    msg = "Enter the new password"
    if request.method == "POST":
        pass1 = request.form["pass1"]
        pass2 = request.form["pass2"]
        if pass1 == pass2:
            sql = "UPDATE CLIENTS SET password=? where id=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, pass1)
            ibm_db.bind_param(stmt, 2, session["id"])
```

```
            if ibm_db.execute(stmt):
                msg = "Successfully Changed Password!!!!"


        else:
            msg = "Passwords not equal"
    flash(msg)
    return redirect(url_for("dashboard"))



@app.route("/changeBudget/", methods=["POST", "GET"])
def changeBudget():
    msg = "Enter the new budget"
    if request.method == "POST":
        budgetAmount = request.form["budgetAmount"]
        sql = "UPDATE BUDGETS SET maxBudget=? where id=?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, budgetAmount)
        ibm_db.bind_param(stmt, 2, session["id"])
        if ibm_db.execute(stmt):
            session["budget"] = budgetAmount
            msg = "Successfully Changed Budget!!!!"
        else:
            msg = "Budget not changed"
    flash(msg)
    return redirect(url_for("dashboard"))



@app.route("/addBudget/", methods=["POST", "GET"])
def addBudget():
    msg = "Enter the budget"
    if request.method == "POST":
```

```
        budgetAmount = request.form["budgetAmountToAdd"]
        sql = "INSERT INTO BUDGETS(id,maxbudget) VALUES(?,?)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, session["id"])
        ibm_db.bind_param(stmt, 2, budgetAmount)
        if ibm_db.execute(stmt):
            session["budget"] = budgetAmount
            msg = "Successfully Set The Budget!!!!"
        else:
            msg = "Budget not set yet"
    flash(msg)
    return redirect(url_for("dashboard"))


def fetchall(stmt):
    ibm_db.bind_param(stmt, 1, session["id"])
    ibm_db.execute(stmt)
    results = []
    result_dict = ibm_db.fetch_assoc(stmt)
    results.append(result_dict)
    while result_dict is not False:
        result_dict = ibm_db.fetch_assoc(stmt)
        results.append(result_dict)
    results.pop()
    return results


def getTotal(table):
    sql = "SELECT SUM(AMOUNT) FROM " + table + " where USER_ID=?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, session["id"])
```

```
    ibm_db.execute(stmt)
    result = ibm_db.fetch_assoc(stmt)
    print(result)
    return result["1"]



@app.route("/log_today", methods=["GET"])
def logToday():
    if isLogged():
        sql = "SELECT AMOUNT,CATEGORY,NEED FROM transacations WHERE USER_ID=?
     AND DATEADDED=CURRENT_DATE"
        stmt = ibm_db.prepare(conn, sql)
        expenseData = fetchall(stmt)
        print(expenseData)
        expenseTotal = getTotal("transacations")
                        sql = "SELECT AMOUNT FROM income WHERE ID=? AND
     DATEADDED=CURRENT_DATE"
        stmt = ibm_db.prepare(conn, sql)
        incomeData = fetchall(stmt)
        print(incomeData)
        return render_template(
            "logtoday.html",
            title="Today's Log",
            expenseData=expenseData,
            incomeData=incomeData,
            expenseTotal=expenseTotal,
        )
    else:
        flash("Login First")
        return redirect("/login")
```

```
@app.route("/addExpense/", methods=["POST", "GET"])
def addExpense():
    msg = ""
    if request.method == "POST":
        amount = request.form["Amount"]
        need = request.form["Need/Want"]
        category = request.form["category"]
          sql = "INSERT INTO transacations(USER_ID,AMOUNT,NEED,CATEGORY,DATEADDED)
     VALUES(?,?,?,?,CURRENT_DATE)"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, session["id"])
        ibm_db.bind_param(stmt, 2, amount)
        ibm_db.bind_param(stmt, 3, need)
        ibm_db.bind_param(stmt, 4, category)
        if ibm_db.execute(stmt):
            msg = "Successfully Added Expense!!!!"
        else:
            msg = "Expense not added"

    flash(msg)
    return redirect(url_for("logToday"))



@app.route("/addIncome/", methods=["POST", "GET"])
def addIncome():
    msg = ""
    if request.method == "POST":
        amount = request.form["AmountIncome"]
        sql = "INSERT INTO INCOME(ID,AMOUNT,DATEADDED) VALUES(?,?,CURRENT_DATE)"
        stmt = ibm_db.prepare(conn, sql)
```

```
        ibm_db.bind_param(stmt, 1, session["id"])
        ibm_db.bind_param(stmt, 2, amount)
        if ibm_db.execute(stmt):
            msg = "Successfully Added Income!!!!"
        else:
            msg = "Income not added"

    flash(msg)
    return redirect(url_for("logToday"))



# @app.route("/Edit")
###Visualization functions



@app.route("/reports")
def reports():
    return render_template("reports.html", title="Reports")



@app.route("/needVwant/")
def needVwant():
        sql = "SELECT Sum(amount) AS amount, need FROM transacations WHERE
     DAYS(CURRENT_DATE)-DAYS(DATEADDED)<29 AND  user_id = ? GROUP BY NEED
     ORDER BY need"
    stmt = ibm_db.prepare(conn, sql)
    transacations = fetchall(stmt)
    values = []
    labels = []
    print(transacations)
    for transaction in transacations:
```

```
        values.append(transaction["AMOUNT"])
        labels.append(transaction["NEED"])
    fig = plt.figure(figsize=(10, 7))
    plt.pie(values)
    plt.title("Need v Want")
    plt.legend(["WANT", "NEED"])
    canvas = FigureCanvas(fig)
    img = BytesIO()
    fig.savefig(img)
    img.seek(0)
    return send_file(img, mimetype="image/png")


@app.route("/categoriesChart/")
def categoriesChart():
        sql = "SELECT Sum(amount) AS amount, category FROM transacations WHERE
     DAYS(CURRENT_DATE)-DAYS(DATEADDED)<29 AND  user_id = ? GROUP BY category
     ORDER BY category"
    stmt = ibm_db.prepare(conn, sql)
    transacations = fetchall(stmt)
    values = []
    labels = []
    print(transacations)
    for transaction in transacations:
        values.append(transaction["AMOUNT"])
        labels.append(transaction["CATEGORY"])
    fig = plt.figure(figsize=(10, 7))
    plt.pie(values, labels=labels)
    plt.title("Categories")
    plt.legend()
    canvas = FigureCanvas(fig)
```

```
    img = BytesIO()
    fig.savefig(img)
    img.seek(0)
    return send_file(img, mimetype="image/png")



##edit the legend... all visualizations workkkkkk!!!!!!!
@app.route("/dailyLineChart/")
def dailyLineChart():
        sql = "SELECT Sum(amount) AS amount, DAY(dateadded) as dateadded FROM
     transacations WHERE DAYS(CURRENT_DATE)-DAYS(DATEADDED)<29 AND  user_id = ?
     GROUP BY dateadded ORDER BY dateadded"
    stmt = ibm_db.prepare(conn, sql)
    transacations = fetchall(stmt)
    x = []
    y = []
    print(transacations)
    for transaction in transacations:
        y.append(transaction["AMOUNT"])
        x.append(transaction["DATEADDED"])
        ##get budget
    sql = "SELECT MAXBUDGET FROM budgets WHERE id = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, session["id"])
    ibm_db.execute(stmt)
    budget = ibm_db.fetch_assoc(stmt)
    print(budget)
    fig = plt.figure(figsize=(10, 7))
    plt.scatter(x, y)
    plt.plot(x, y, "-")
    if budget:
```

```
        plt.axhline(y=budget["MAXBUDGET"], color="r", linestyle="-")
    plt.xlabel("Day")
    plt.ylabel("Transaction")
    plt.title("Daily")
    plt.legend()
    canvas = FigureCanvas(fig)
    img = BytesIO()
    fig.savefig(img)
    img.seek(0)
    return send_file(img, mimetype="image/png")


if __name__ == "__main__":
    app.debug = True
    app.run(host='0.0.0.0')
```

GitHub Repository

Project Demo Link