PROJECT REPORT

Project Title	PERSONAL EXPENSE TRACKER APPLICATION
Team ID	PNT2022TMID14560
Team Members	Gokul Raam M (Leader) - 19BCS026 Serjeel Ranjan - 19BCS009 Gokul S - 19BCS033 Ashok Kaushik - 19BCS059
College Name	Kumaraguru College of Technology

CONTENTS

1. INTRODUCTION

1.1 Project Overview

Keeping tabs on your spending is sometimes the first step in organizing your finances. You can determine exactly where your money is going and where you can make savings by knowing what you buy and how much you spend. Knowing how you spend your money is the first step, if you've ever determined to take control of your cash flow. That is something that is incredibly boring to do. The majority of people are aware of the value of retaining business receipts in case of audit. Paper receipts have the drawback that it is simple to misplace such a significant record.

Instead of having to dig through a pile full of receipts, you can save space and time by putting these right away into an expense tracker app. With the aid of mobile spending tracker applications, you can easily include this into your daily routine. These apps definitely overlap with budgeting tools, however while the latter gives you a broad overview of your income, cost tracker apps focus more on your spending. These apps typically classify your expenses and give you a clear picture of your shopping habits. To gather and categorize your purchases and find areas where you may cut costs, use expense tracker apps. Alternatively, if you're trying to increase your net worth, consider investing more of your income or setting aside more money. You may keep track of your spending for a while to get a sense of where your money is going, or it may be a first step in creating and adhering to a budget.

This solution service can be used by any end user who wishes to plan their expenses on the go. This includes majority of people from working class who wants to understand what they spend money on and how much they spend, they can see exactly where their cash is draining and areas where they can cut back to get their finances in order.

1.2 Purpose

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. We don’t have to go through the loads of paper receipts, you may save time by entering them into an expense tracker application straight immediately. Use expense tracker application to collect and classify your purchases and identify areas where you may save money. For a while, you may keep track of your spending to get a feel of where your money is going, or it may be the first step towards making and following a budget. The most of the middle-class families want to understand what they spend money on and how much they spend so that they can see exactly where their money is going and where they may reduce them to get their finances in order.

2. LITERATURE SURVEY

2.1 Existing problem

We have collected, studied some series of articles and research papers about various proposed solutions for developing a Personal Expense Tracker Application.

DAILY EXPENSE TRACKER MOBILE APPLICATION [1]

In this report, the Author used Least Square Method which helps in a successful budget planned with the prediction of the outcome of the budget based on expenses for his Daily Expense Tracker Mobile Application. This Application integrates the least square method in statistics for a function which generates a monthly expense report and budget outcome predictions.

As the least square method is very sensitive to even small deviations, it could drastically affect the outcome. This is considered as one of the major disadvantages of this application. And another being this is limited to being only a Mobile Application as of now.

EXPENSE TRACKER [2]

In this android application, use to track and update their daily costs so they are always aware of how much they are spending and they the user can establish their own spending categories like food, clothing, rent and bills, and then enter the amount spent. And there is notes feature to specify more information about the expenses. The advantage in the application is there is a pie chart of costs feature to visualize the expenses in an efficient way.

EXPENSE TRACKER MOBILE APPLICATION [3]

This application is similar to the above mentioned except this is for iPhone users implemented using iOS SDK. After when the user enters the expenses, dates and other details, the user is able to see the expense details daily, weekly, monthly, and yearly in figures, graphs, PDF format, and can print them as well if a printer is detected or scanned nearby.

As the user can only enter the expense amount in United States Dollars Currency, this application is limited to only for US people. And also there is no search option in this application, so if a user wants to see the expense he made on Rent for the past 2 months he has to scroll through the calendar to find it.

Expense Manager Application [4]

In this mobile application, which is for 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.

This application can also help digital marketing agencies in rolling out their advertising campaigns more effectively. A feature called “splitbill” is available used to split bills among friends and like for storing any information about money lent or borrowed.

Expense Tracker : A Smart Approach to Track Everyday Expense [5]

This research paper is about a Windows based application specifically for windows users. This Expense Tracker is a day-to-day expense management system designed to easily and efficiently track the daily expenses of unpaid and unpaid staff through a computerized system that eliminates the need for manual paper tasks that systematically maintains records and easily accesses data stored by the user.

2.2 References

[1] MUSTAFA, N. N. B. DAILY EXPENSE TRACKER MOBILE APPLICATION.

[2] Jadhav, N. J., Chakor, R. V., Gunjal, T. M., & Pawar, D. D. EXPENSE TRACKER.

[3] Manchanda, A. (2012). Expense Tracker Mobile Application (Doctoral dissertation, San Diego State University).

[4] Velmurugan, A., Mayan, J. A., Niranjana, P., & Francis, R. (2020, December). Expense manager application. In Journal of Physics: Conference Series (Vol. 1712, No. 1, p. 012039). IOP Publishing.

[5] Gupta, H., Singh, A. P., Kumar, N., & Blessy, J. A. (2020). Expense Tracker: A Smart Approach to Track Everyday Expense (No. 4809). EasyChair.

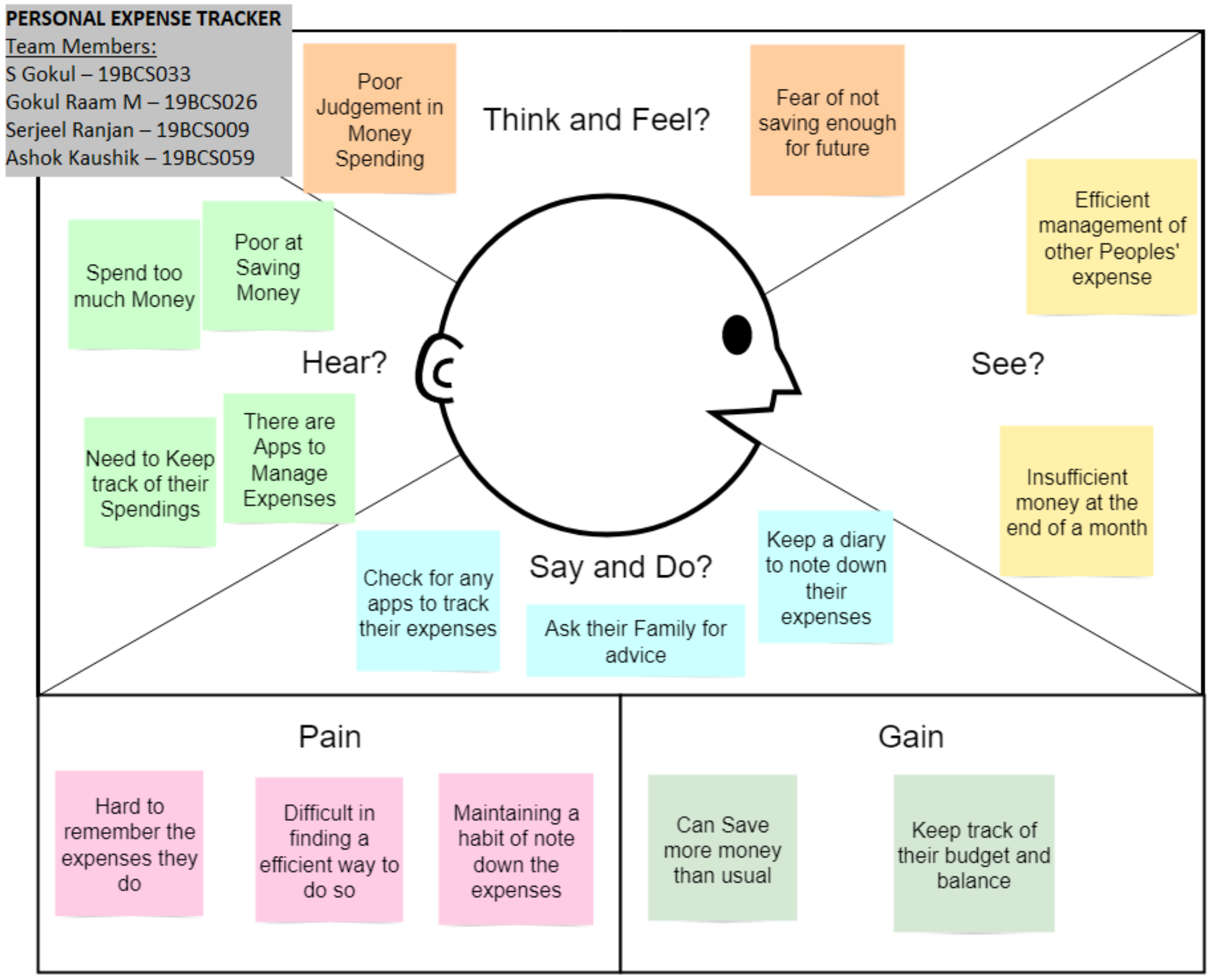
2.3 Problem Statement Definition

In simple words, personal finance entails all the financial decisions and activities that a Finance app makes your life easier by helping you to manage your finances efficiently. A personal finance app will not only help you with budgeting and accounting but also give you helpful insights about money management.

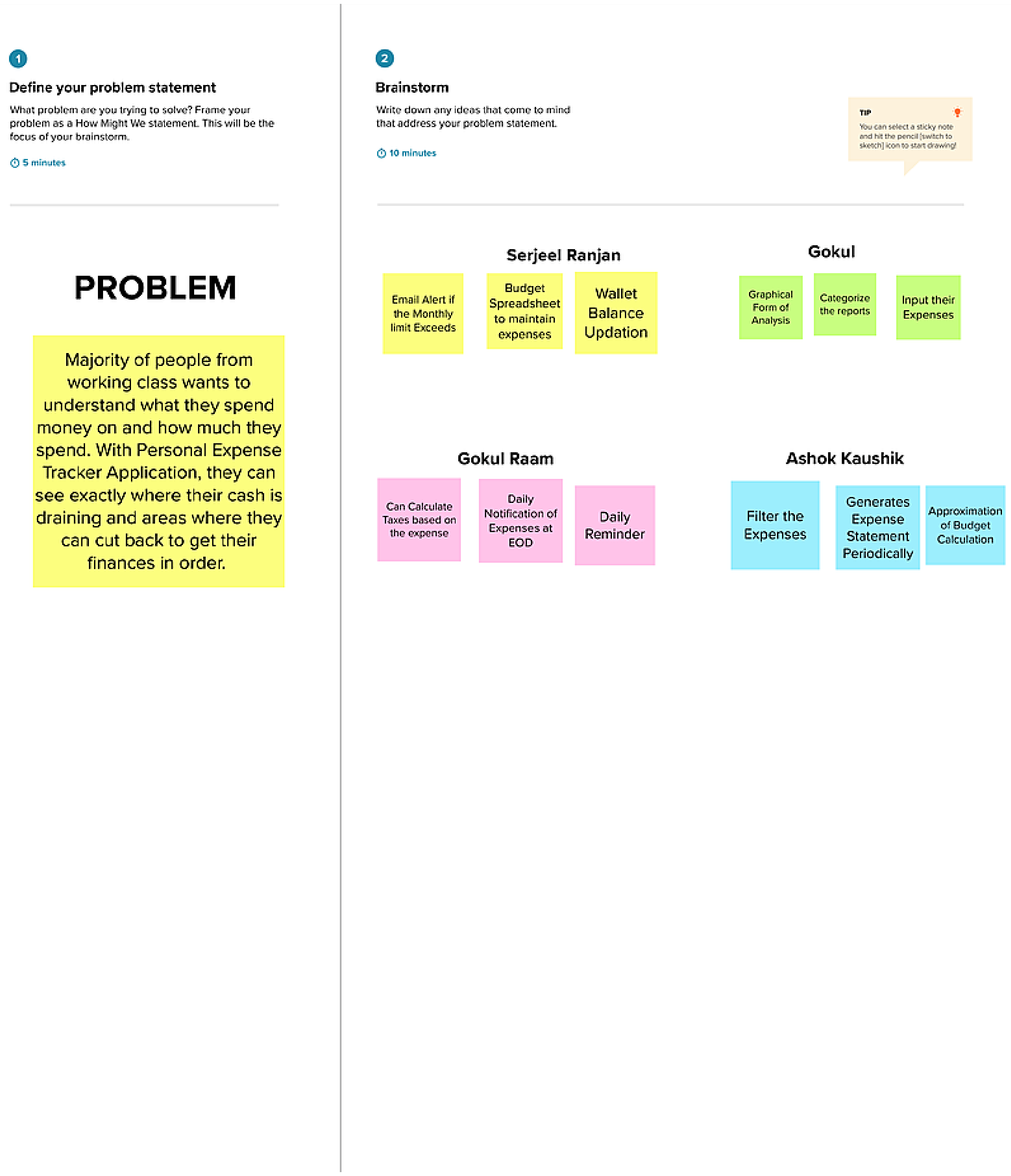
Personal finance applications will ask users to add their expenses and based on their expenses wallet balance will be updated which will be visible to the user. Also, users can get an analysis of their expenditure in graphical forms. They have an option to set a limit for the amount to be used for that particular month if the limit is exceeded the user will be notified with an email alert

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas


PERSONAL EXPENSE TRACKER
Team Members:
S Gokul – 19BCS033
Gokul Raam M – 19BCS026
Serjeel Ranjan – 19BCS009
Ashok Kaushik – 19BCS059
Poor Judgement in Money Spending
Think and Feel?
Fear of not saving enough for future
Spend too much Money
Poor at Saving Money
Efficient management of other Peoples' expense
Hear?
See?
Need to Keep track of their Spendings
There are Apps to Manage Expenses
Insufficient money at the end of a month
Check for any apps to track their expenses
Say and Do?
Keep a diary to note down their expenses
Ask their Family for advice
Pain
Gain
Hard to remember the expenses they do
Difficult in finding a efficient way to do so
Maintaining a habit of note down the expenses
Can Save more money than usual
Keep track of their budget and balance

3.2 Ideation & Brainstorming



1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

Majority of people from working class wants to understand what they spend money on and how much they spend. With Personal Expense Tracker Application, they can see exactly where their cash is draining and areas where they can cut back to get their finances in order.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

Serjeel Ranjan

- Email Alert if the Monthly limit Exceeds
- Budget Spreadsheet to maintain expenses
- Wallet Balance Updation

Gokul

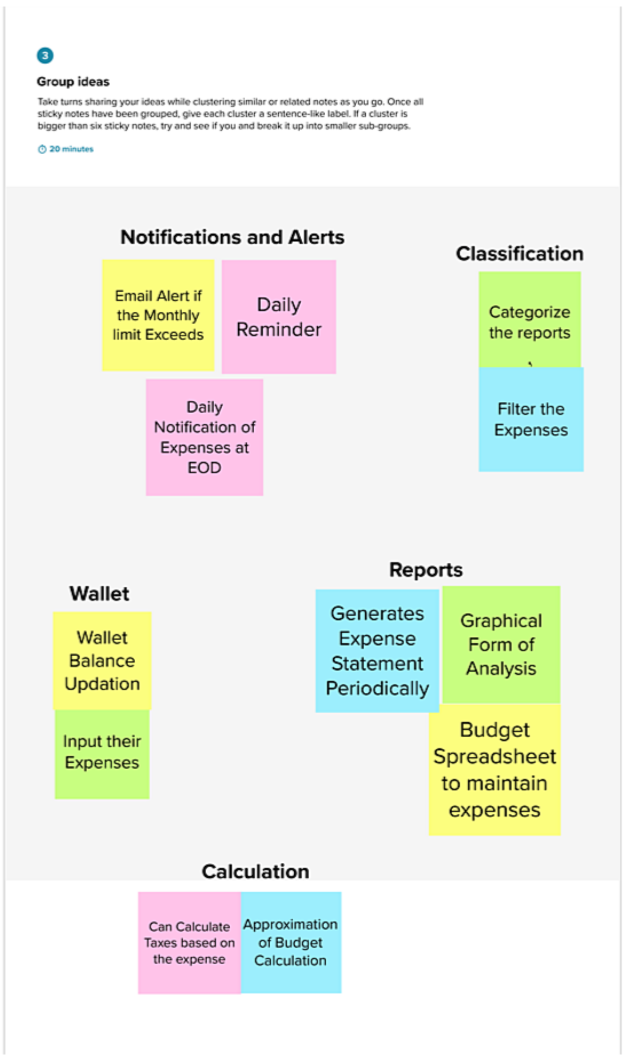
- Graphical Form of Analysis
- Categorize the reports
- Input their Expenses

Gokul Raam

- Can Calculate Taxes based on the expense
- Daily Notification of Expenses at EOD
- Daily Reminder

Ashok Kaushik

- Filter the Expenses
- Generates Expense Statement Periodically
- Approximation of Budget Calculation



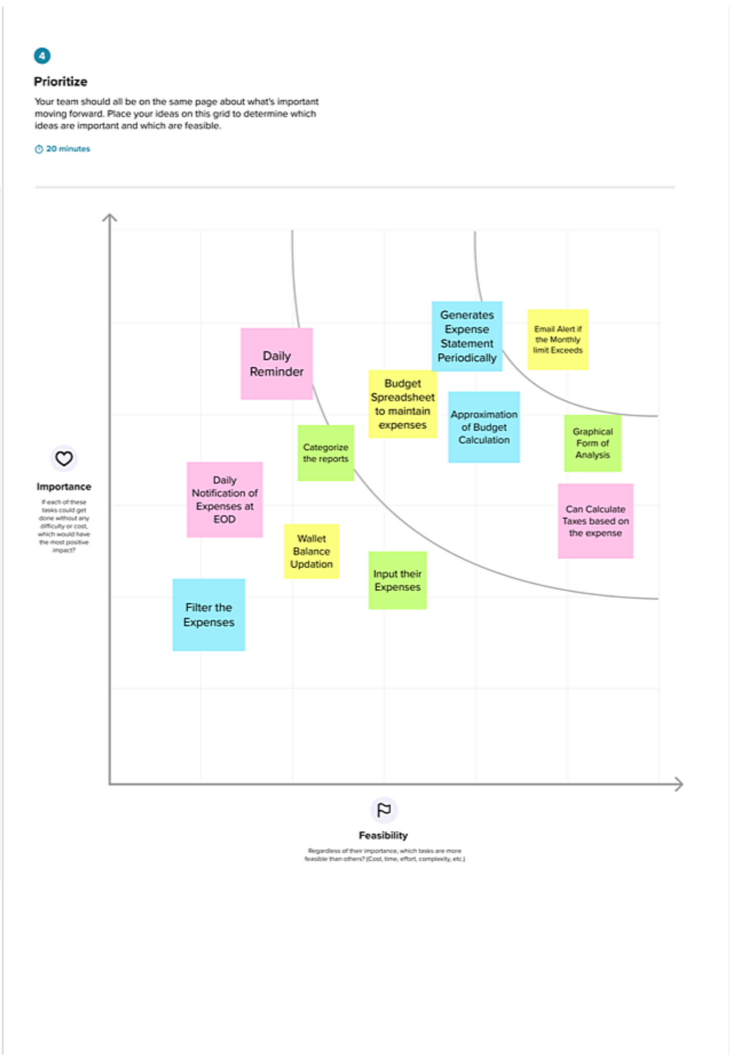
3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, 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

Notifications and Alerts
Email Alert if the Monthly limit Exceeds
Daily Reminder
Daily Notification of Expenses at EOD
Classification
Categorize the reports
Filter the Expenses
Wallet
Wallet Balance Updation
Input their Expenses
Reports
Generates Expense Statement Periodically
Graphical Form of Analysis
Budget Spreadsheet to maintain expenses
Calculation
Can Calculate Taxes based on the expense
Approximation of Budget Calculation



4

Prioritize

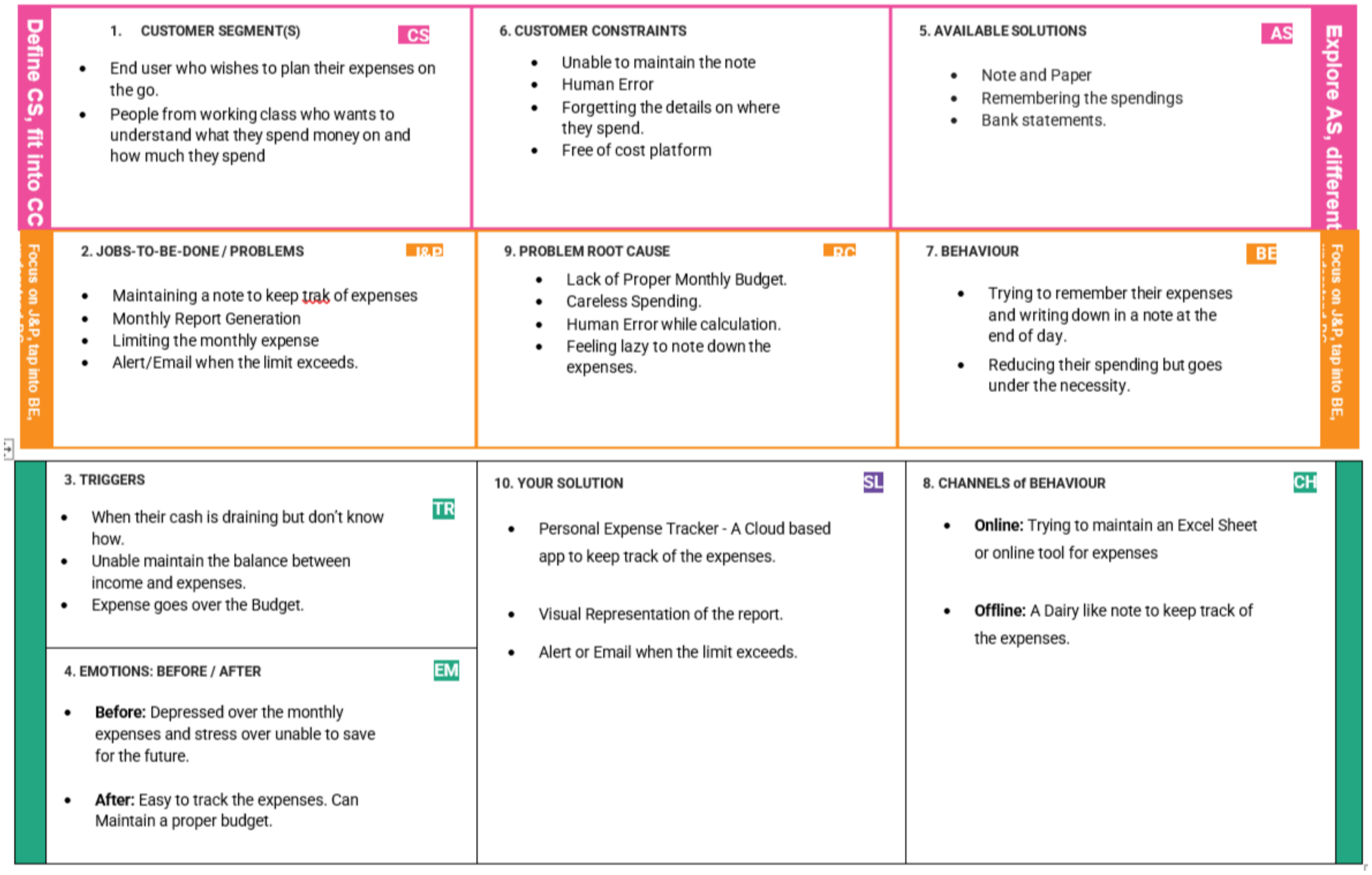
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes

3.3 Proposed Solution

S.No.	Parameter	Description
1.	**Problem Statement (Problem to be solved)**	Majority of people from working class wants to understand what they spend money on and how much they spend, they can see exactly where their cash is draining and areas where they can cut back to get their finances in order.
2.	**Idea / Solution description**	A Personal Expense Tracker makes your life easier by helping you to manage your finances efficiently. A Tracker app will not only help you with budgeting and accounting but also give you helpful insights about money management.
3.	**Novelty / Uniqueness**	-> 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. -> Graphical form of Report.
1.	**Social Impact / Customer Satisfaction**	With this app, people can save money efficiently and help to make a budget for a month. It can reduce the debt problems and also to gain more financial knowledge.
2.	**Business Model (Revenue Model)**	Integration with the Bank Account for Premium/Subscribed Users.
3.	**Scalability of the Solution**	The collected data from the users can be used to take census nationwide for Annual Budget to help the Government but in a secure way.

3.4 Problem Solution fit



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

- End user who wishes to plan their expenses on the go.
- People from working class who wants to understand what they spend money on and how much they spend

6. CUSTOMER CONSTRAINTS

- Unable to maintain the note
- Human Error
- Forgetting the details on where they spend.
- Free of cost platform

5. AVAILABLE SOLUTIONS AS

- Note and Paper
- Remembering the spendings
- Bank statements.

Explore AS, different

Focus on J&P, tap into BE,

2. JOBS-TO-BE-DONE / PROBLEMS J&P

- Maintaining a note to keep trak of expenses
- Monthly Report Generation
- Limiting the monthly expense
- Alert/Email when the limit exceeds.

9. PROBLEM ROOT CAUSE RC

- Lack of Proper Monthly Budget.
- Careless Spending.
- Human Error while calculation.
- Feeling lazy to note down the expenses.

7. BEHAVIOUR BE

- Trying to remember their expenses and writing down in a note at the end of day.
- Reducing their spending but goes under the necessity.

Focus on J&P, tap into BE,

3. TRIGGERS TR

- When their cash is draining but don't know how.
- Unable maintain the balance between income and expenses.
- Expense goes over the Budget.

4. EMOTIONS: BEFORE / AFTER EM

- **Before:** Depressed over the monthly expenses and stress over unable to save for the future.
- **After:** Easy to track the expenses. Can Maintain a proper budget.

10. YOUR SOLUTION SL

- Personal Expense Tracker - A Cloud based app to keep track of the expenses.
- Visual Representation of the report.
- Alert or Email when the limit exceeds.

8. CHANNELS of BEHAVIOUR CH

- **Online:** Trying to maintain an Excel Sheet or online tool for expenses
- **Offline:** A Dairy like note to keep track of the expenses.

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Email
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login through email and password
FR-4	Forgot Password	Send OTP to Email
FR-5	Dashboard Panel	Input Expense Input Budget
FR-6	Report Generation	Generate Statement for a specific Time Period (Custom, Weekly, Monthly)
FR-7	Graphical Report	Generate Report in a Graphical Form (Pie Chart, Bar Chart, Line Chart)
FR-8	Alert/Notification	Email when the limit exceeds.

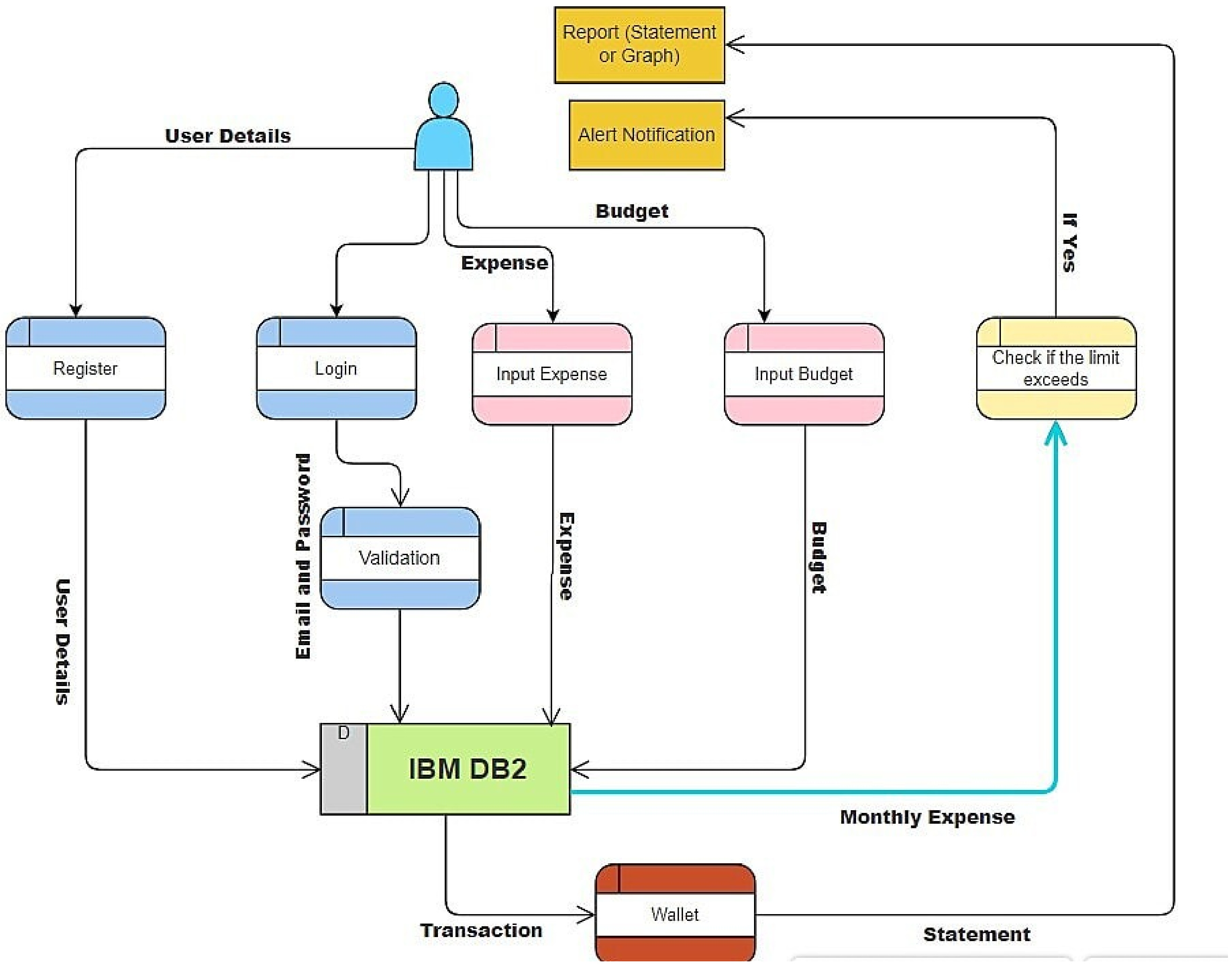
4.2 Non-Functional requirements

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

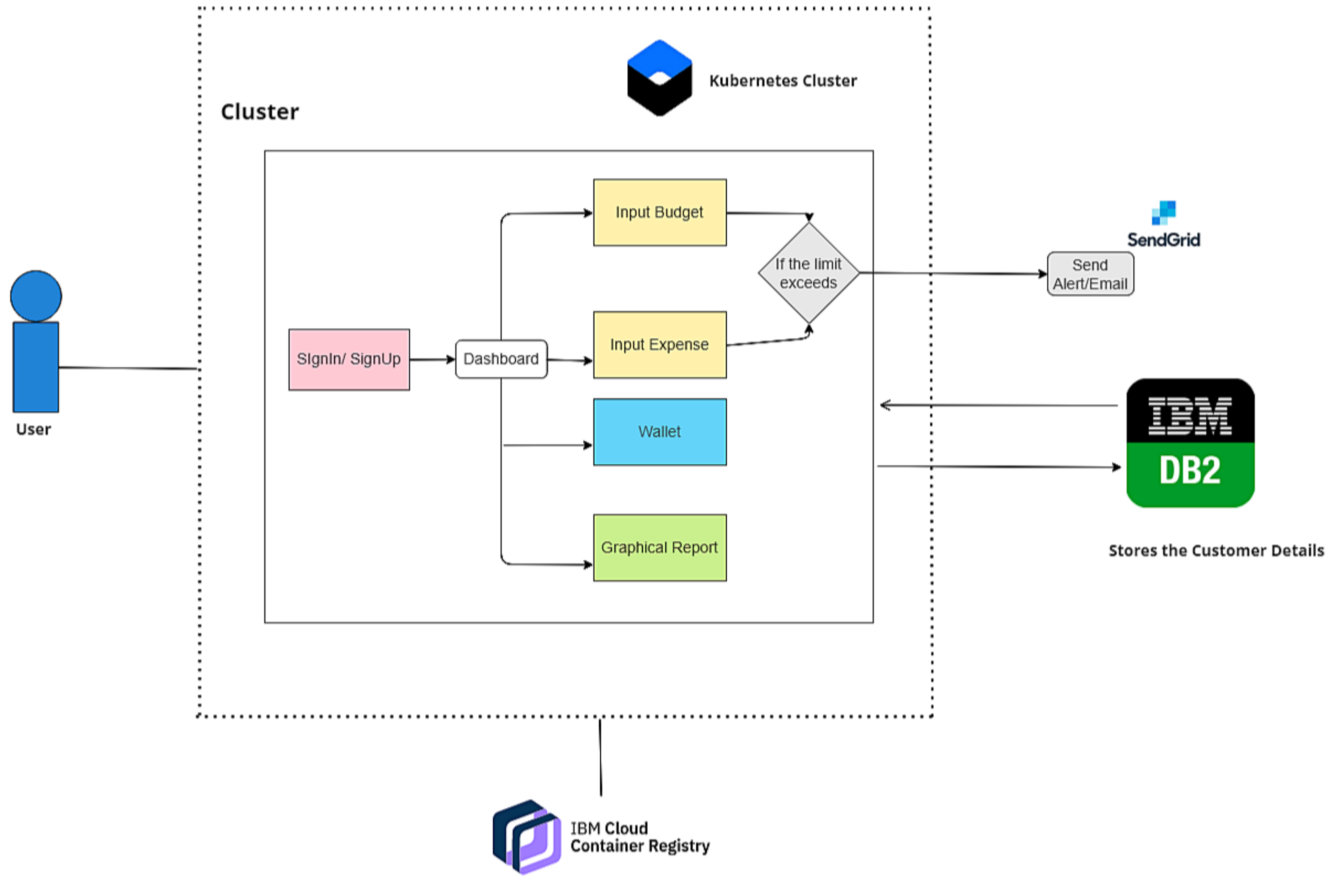
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Compatibility of the tracker application in any devices and a webapp version too.
NFR-2	**Security**	Encryption of Passwords stored in the Database.
NFR-3	**Reliability**	No fear of data leakage or security issue since trustworthy database and its structure.
NFR-4	**Performance**	Optimised Categorisation of the expenses can increase the efficiency of the application.
NFR-5	**Availability**	Can access the application anytime and anywhere since the support of IBM cloud.
NFR-6	**Scalability**	The Potential to handle exponential growth in users

5. PROJECT DESIGN

5.1 Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
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	I can access my account/dashboard with my Gmail	Medium	Sprint-1
Login	USN-4	As a user, I can log into the application by entering email & password	I can access my account / dashboard	High	Sprint-1
Dashboard	USN-5	As a user, I can enter my income and expenses	I can access my report.	High	Sprint-2
	USN-6	As a user, I can enter a monthly limit to get an alert	I can get alert when the monthly limit exceeds.	High	Sprint-2
Wallet	USN-7	As a user, I can add money in my wallet.	I can access my savings.	Medium	Sprint-3
Using Sendgrid	USN-8	As a user, I will get an alert mail when my monthly limit exceeds.	I will get the alert email saying my limit exceeds.	High	Sprint-3
Report Generation	USN-9	As a user, I can get my periodical report on my expenses.	I can access my statement in either Spreadsheet	Medium	Sprint-4
Graph Charts	USN-10	As a user, I can get periodical report in Graph format using the data.	A graph form report for the selcected time period will be shown.	Low	Sprint-4
Deployment	USN-11	As a user, I can access my data from anywhere. Deploying the application in cloud.	I can access the application from any browser from anywhere on the world with my data.	High	Sprint-4

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 a user, I can register for the application by entering my email, password, and confirming my password.	7	High	Serjeel, Gokul Raam
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	3	High	Ashok, Gokul
Sprint-1		USN-3	As a user, I can register for the application through Gmail	7	Medium	Ashok, Gokul
Sprint-1	Login	USN-4	As a user, I can log into the application by entering email & password	3	Medium	Serjeel, Gokul Raam
Sprint-2	Dashboard	USN-5	As a user, I can enter my income and expenses.	13	High	Serjeel

Sprint-2		USN-6	As a user, I can enter a limit to set for the expenses	7	Medium	Gokul
Sprint-3	Wallet	USN-7	As a user, I can add money in my wallet	7	Medium	Ashok
Sprint-3	Using Sendgrid	USN-8	As a user, I will get an alert mail when my monthly limit exceeds.	13	High	Gokul Raam
Sprint-4	Report Generation	USN-9	As a user, I can get my monthly report in a spreadsheet format.	7	Medium	Ashok
Sprint-4	Graph Charts	USN-10	As a user, I can get periodical report in Graph format using the data.	10	Low	Gokul Raam, Gokul
Sprint-4	Deployment	USN-11	As a user, I can access my data from anywhere. Deploying the application in cloud.	3	High	Serjeel

6.2 Sprint Delivery Schedule

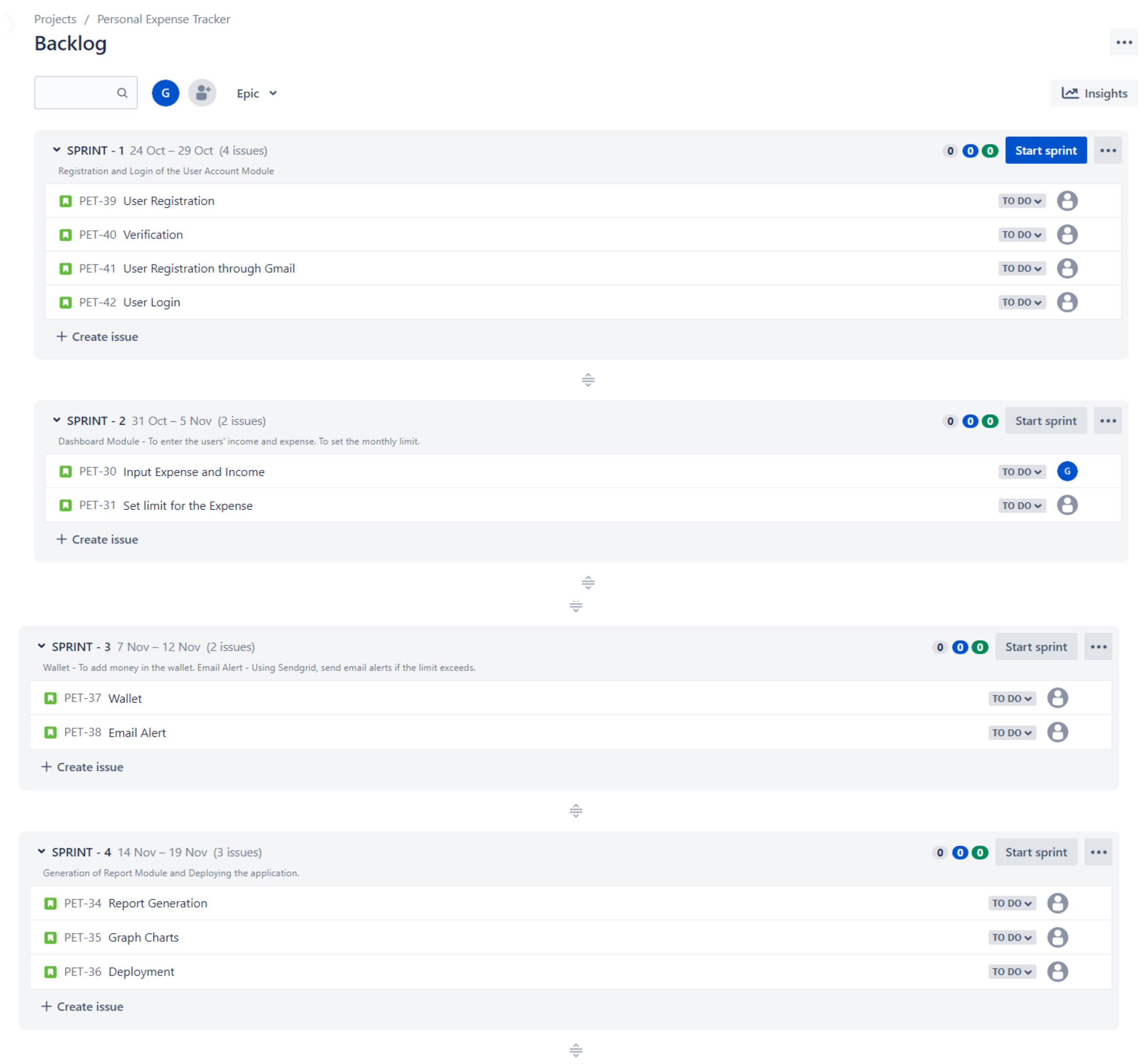
Sprint	**Total Story Points**	**Duration**	**Sprint Start Date**	**Sprint End Date**	**Story Points Completed**	**Sprint Release Date (Actual)**
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

Velocity:

Here we have a sprint of duration 6 days, and the velocity of the team is 20(Total Story Points per Sprint)

AV = 20/6 = 3.33

6.3 Reports from JIRA



Projects / Personal Expense Tracker

Backlog

G Epic Insights

SPRINT - 1 24 Oct – 29 Oct (4 issues) 0 0 0 Start sprint

Registration and Login of the User Account Module

PET-39	User Registration	TO DO
PET-40	Verification	TO DO
PET-41	User Registration through Gmail	TO DO
PET-42	User Login	TO DO

+ Create issue

SPRINT - 2 31 Oct – 5 Nov (2 issues) 0 0 0 Start sprint

Dashboard Module - To enter the users' income and expense. To set the monthly limit.

PET-30	Input Expense and Income	TO DO
PET-31	Set limit for the Expense	TO DO

+ Create issue

SPRINT - 3 7 Nov – 12 Nov (2 issues) 0 0 0 Start sprint

Wallet - To add money in the wallet. Email Alert - Using Sendgrid, send email alerts if the limit exceeds.

PET-37	Wallet	TO DO
PET-38	Email Alert	TO DO

+ Create issue

SPRINT - 4 14 Nov – 19 Nov (3 issues) 0 0 0 Start sprint

Generation of Report Module and Deploying the application.

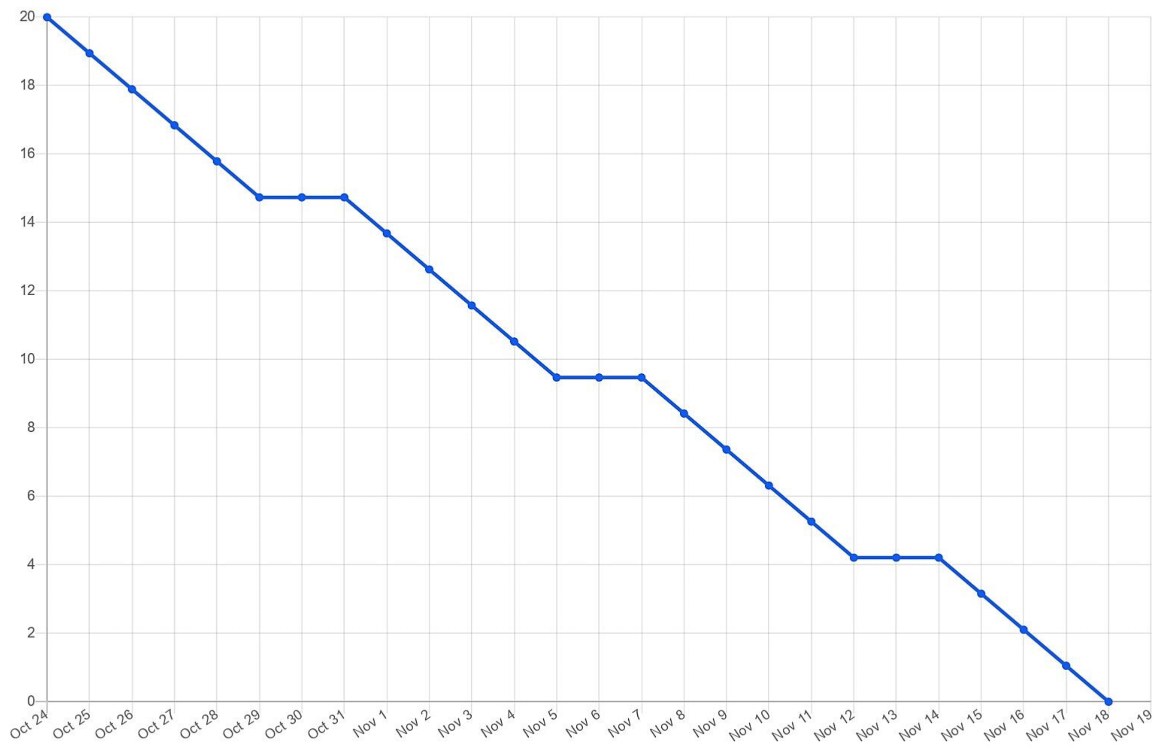
PET-34	Report Generation	TO DO
PET-35	Graph Charts	TO DO
PET-36	Deployment	TO DO

+ Create issue

Burndown Chart:

11/11/22, 5:56 PM (Team ID: PNT2022TMID14560) Personal Expense Tracker - Sprint Burndown Chart - Generated by https://easyretro.io

(Team ID: PNT2022TMID14560) Personal Expense Tracker - Sprint Burndown Chart



https://easyretro.io/burndown-chart-generator/ 1/2

7. CODING & SOLUTIONING

7.1 Feature 1 - Expense and Income Tracker

i. Expense Tracker:

To keep track our expenses. The expense we enter as input can be categorised with user-defined categories. We can add, remove and update our expenses. And have an option to filter the expenses while searching.

Coding - Frontend:

```
import { loadChart } from "../chart.js";
import { send_usage_alert } from "./alert.js";
import { endpoint } from "./endpoint.js";
import { fetchSplitIncome, updateBalance } from "./income.js";
import { expense_data_template } from "./template.js";
import { user } from "./user_data.js";

let is_income = false;
let label = "Food & Drinks";

const labelDropDown = document.querySelector("#amount-label");
const radioTypeBtn = document.querySelectorAll(".radio-type-expense");
const expenseForm = document.querySelector(".expense-add-form");

const updateIsIncome = (e) => {
   is_income = e.currentTarget.dataset.value === "true" ? true : false;
   if(is_income){
      radioTypeBtn[0].classList.add("active");
      radioTypeBtn[1].classList.remove("active");
   }
   else{
      radioTypeBtn[1].classList.add("active");
      radioTypeBtn[0].classList.remove("active");
   }
   console.log(is_income);
}
const updateLabelValue = (e) => {
   label =e.target.value;
   console.log(label);
}
```

```
const addExpense = async (e) => {
    e.preventDefault();
    const amountInp = document.querySelector("#amount-inp");
    const amount = +amountInp.value;
    const timestamp = Date.now();
    const data = {
        amount,
        label,
        is_income,
        timestamp
    }
    const res = await fetch(endpoint.add_expense, {
        method:"POST",
        credentials: 'include',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    });
    if(res.status === 200){
        amountInp.value = "";
        user.updateUserExpenseData(data);
        if(isFilterPresent === false){
            console.log('in', filterData)
            updateExpenseData(user.getData('expenseData'));
        }
        updateBalance();
        fetchSplitIncome();
        send_usage_alert();
        loadChart();
    }
}

export const updateExpenseData = (expenseData) => {
    const expenseValuesCnt = document.querySelector(".expense-his-values");
    const expenseValueCnt = document.querySelectorAll(".expense-his-value");
    expenseValueCnt.forEach(child => {
        expenseValuesCnt.removeChild(child);
    })
    if(expenseData.length === 0){
        document.querySelector(".expense-msg").classList.remove("none");
        return;
        }
        document.querySelector(".expense-msg").classList.add("none");
        expenseData.forEach(eachData => {
            const valueDiv = expense_data_template(eachData);
            expenseValuesCnt.appendChild(valueDiv)
        })
    }

    // Filter expense
    const fromDateInp = document.querySelector("#from-date");
    const toDateInp = document.querySelector("#to-date");
    const filterLabelInp = document.querySelector("#filter-label");
    const updateBtn = document.querySelector('.filter-update-btn');
    const resetBtn = document.querySelector('.filter-reset-btn');
    let filterData = false, isFilterPresent = false;
    const getTimestampFromDate = (dateStr) => {
        const datePart = dateStr.split('-');
        const date = new Date(datePart[0], datePart[1]-1, datePart[2]);
        return date.getTime();
    }

    let prevToTimestamp = 0, currToTimestamp = 0, prevFromTimestamp = 0,
currFromTimestamp = 0;

    const setDate = (is_toDate, e) => {
        const dateStr = e.target.value;
        is_toDate ?
            (currToTimestamp = getTimestampFromDate(dateStr) + (60 * 60 * 24 * 1000) -
1000) :
            (currFromTimestamp = getTimestampFromDate(dateStr))
    }

    const filterExpenseLabel = () => {
        const label = filterLabelInp.value;
        isFilterPresent = filterData === false ? false : true;
        const toUpdateData = filterData === false ? user.getData('expenseData') : filterData;
        if(label === "None"){
            updateExpenseData(toUpdateData);
            return;
        }
        isFilterPresent = true;
        const newFilterData = toUpdateData.filter(eachData => eachData["LABEL"] ===
label)
        updateExpenseData(newFilterData)
    }

    let isFilterProcessing = false;
    const getFilterExpense = async (e) => {
        e.preventDefault();
        if(isFilterProcessing || (currToTimestamp == 0 || currFromTimestamp == 0) ||
currFromTimestamp >= currToTimestamp || (currToTimestamp == prevToTimestamp &&
currFromTimestamp == prevFromTimestamp) ){
            filterExpenseLabel(user.getData('expenseData'));
            return;
        }
        isFilterProcessing = true;
        const bodyData = {
            fromTimestamp: currFromTimestamp,
            toTimestamp: currToTimestamp
        }
        const res = await fetch(endpoint.expense_filter, {
            method:"POST",
            credentials: 'include',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(bodyData)
        });
        if(res.status === 200){
            const resData = await res.json();
            filterData = resData["expense_data"]
            console.log(filterData)
            filterExpenseLabel()
            prevFromTimestamp = currFromTimestamp;
            prevToTimestamp =currToTimestamp;
            isFilterProcessing = false;
        }
    }

    const resetFilter = (e) => {
        e.preventDefault();
        isFilterPresent = false;
        toDateInp.value = fromDateInp.value = "";
        filterLabelInp.value = "None";
```

```
    filterData = false;
    filterExpenseLabel();
}

export const loadExpenseFunction = () => {
    console.log(radioTypeBtn)
    labelDropDown.addEventListener("change", updateLabelValue);
    radioTypeBtn.forEach(btn => {
        btn.addEventListener("click", updateIsIncome)
    });
    expenseForm.addEventListener("submit", addExpense);
    fromDateInp.addEventListener("change", setDate.bind(null, false));
    toDateInp.addEventListener("change", setDate.bind(null, true));
    updateBtn.addEventListener("click", getFilterExpense);
    resetBtn.addEventListener("click", resetFilter);
}
```

Coding - Backend:

```
from flask import request
from flask_restful import Resource
from ..utils import validate, general, db
from ..utils.general import token_required

class Expense(Resource):
    @token_required
    def post(payload, self):
        user_data = request.json
        print(request.json)
        validate_result = validate.validate_add_expense(user_data=user_data)
        print(validate_result)
        if(validate_result):
            print('exp')
            return validate_result["error"]
        sql_query = "INSERT INTO expense (user_id, amount, is_income, label, 
timestamp) values(?, ?, ?, ?, ?)"
        params = (payload["id"], user_data["amount"], user_data["is_income"], 
user_data["label"], user_data["timestamp"])
        run_status = db.run_sql_update(sql_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200
    @token_required
    def delete(payload, self, id):
        sql_query = "DELETE FROM expense WHERE id=?"
        params = (id)
        run_status = db.run_sql_delete(sql_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200

class ExpenseFilter(Resource):
    @token_required
    def post(payload, self):
        user_date = request.json
        print(user_date)
        sql_query = "SELECT * FROM expense WHERE user_id = ? AND timestamp
BETWEEN ? AND ?"
        params = (payload["id"], user_date["fromTimestamp"],
user_date["toTimestamp"])
        expense_data = db.run_sql_select(sql_query, params=params)
        return {"expense_data": expense_data}, 200
```

ii. Income Tracker:

This feature is to keep track our income. When we input our income, it will be stored in our Database. And also the income can be split and balance can be calculated from the expenses.

Coding - Frontend:

```
import { endpoint } from "./endpoint.js";
import { labelId } from "./label.js";
import { split_data_template } from "./template.js";
import { user } from "./user_data.js";

const incomeForm = document.querySelector(".user-income-form");
const editIncomeBtn = document.querySelector(".edit-income-ic");
const tickBtn = document.querySelector(".accept-income-ic");

let isTrigger = false;
let prevValue = 0;
const incomeInp = document.querySelector("#income");
const editIncome = (e) => {
    e.preventDefault();
    console.log(prevValue)
    if(isTrigger && +incomeInp.value !== prevValue){
        updateIncome(+incomeInp.value)
    }
    prevValue = incomeInp.value ? +incomeInp.value : 0;
    incomeInp.readOnly = isTrigger;
    incomeInp.focus();
    editIncomeBtn.classList.toggle("none");
    tickBtn.classList.toggle("none");
    isTrigger = !isTrigger;
}

const updateIncome = async (amount) => {
    const timestamp = Date.now();
    const data = {
        amount,
        timestamp
    };
    const res = await fetch(endpoint.add_income, {
        method:"POST",
        credentials: 'include',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify(data)
    });
    const status = await res.json();
    if(res.status === 200){
        user.setData('balance', amount);
        document.querySelector(".balance span").innerText = amount;
    }
}

export const updateBalance = () => {
    const balanceEle = document.querySelector(".balance span");
    balanceEle.innerText = user.getData('balance');
}

export const updateSplitData = () => {
    const split_data_cnt = document.querySelector(".split-values");
    const split_value_cnt = split_data_cnt.querySelectorAll(".split-value");
    split_value_cnt.forEach((ele, idx) => {
        if(idx == 0)  return;
        split_data_cnt.removeChild(ele);
    })
    console.log(user.getData('splitData'))
    user.getData('splitData').forEach(data => {
        const split_value_div = split_data_template(data);
        console.log(split_value_div);
        split_data_cnt.appendChild(split_value_div);
        split_value_div.querySelector(".split-edit").addEventListener("click",
removeSplitData);
    });
}

// income split
let splitAmount = 0;
let label = "Food & Drinks";
const updateLabelValue = (e) => {
    label =e.target.value;
}
const updateSplitPreview = (amount) => {
    const splitCnt = document.querySelector(".split-preview span");
    const isPercent = document.querySelector('input[name="split-type"]:checked').value;
    console.log('hi')
    if(isPercent === "percent"){
        amount = amount > 100 ? 100 : amount;
        splitAmountInp.value = amount;
        amount = calculateAmount(amount)
    }
    else{
        if(amount > user.getData('balance')){
            amount = user.getData('balance');
            splitAmountInp.value = amount;
        }
    }
    splitCnt.innerText = `Rs ${amount}`;
    splitAmount = amount
}

const calculateAmount = (percentage) => {
    const balance = user.getData('balance');
```

```
   const amount = ((balance / 100) * percentage).toFixed(2);
   return amount
}

const splitAmountInp = document.querySelector("#split-amount-inp");
const radioOptions = document.querySelectorAll('input[name="split-type"]');
const labelDropDown = document.querySelector("#split-label");
const splitIncomeForm = document.querySelector(".split-income-form")
const changeSplitOnUpdate = (e) => {
   updateSplitPreview(+splitAmountInp.value)
}

let isSplitProgress = false;
const addSplitIncome = async (e) => {
   e.preventDefault();
   if(splitAmount === 0 || isSplitProgress){
      return;
   }
   isSplitProgress = true;
   const data = {
      amount: splitAmount,
      label
   };
   const res = await fetch(endpoint.split_income, {
      method:"POST",
      credentials: 'include',
      headers: {
         'Content-Type': 'application/json'
      },
      body: JSON.stringify(data)
   });
   const msg = await res.json();
   isSplitProgress = false;
   if(res.status === 200){
      user.updateSplitData(data);
      updateSplitData();
      splitAmountInp.value = ""
   }
}

export const fetchSplitIncome = async () => {
   const res = await fetch(endpoint.get_split_income(user.getData('timestamp')), {
        method:"GET",
        credentials: 'include',
    });
    const resData = await res.json();
    console.log(resData)
    if(res.status === 200){
        // user.setSplitData(resData["data"]);
        user.setSplitData(user.getData('splitData'), resData['balance_data']);
        updateSplitData();
    }
}

let isRemoveTriggered = false;
const removeSplitData = async (e) => {
    if(isRemoveTriggered){
        return;
    }
    isRemoveTriggered = true;
    const label = e.currentTarget.parentElement.dataset.value;
    const id = labelId[label]
    const res = await fetch(endpoint.split_income_del(id), {
        method:"DELETE",
        credentials: 'include',
    });
    if(res.status === 200){
        user.removeSplitData(label);
        updateSplitData();
    }
    isRemoveTriggered = false;
}

export const loadIncomeFunction = () => {
    editIncomeBtn.addEventListener("click", editIncome);
    tickBtn.addEventListener("click", editIncome);
    incomeForm.addEventListener("submit", editIncome);
    radioOptions.forEach(ele => {
        ele.addEventListener("change", changeSplitOnUpdate);
    });
    labelDropDown.addEventListener("change", updateLabelValue);
    splitAmountInp.addEventListener("change", changeSplitOnUpdate);
    splitIncomeForm.addEventListener("submit", addSplitIncome);
}
```

Coding - Backend:

```
from flask import request
from flask_restful import Resource
import ibm_db
from ..utils import validate, general, db
from ..utils.general import token_required

class Income(Resource):
    @token_required
    def post(payload, self):
        user_data = request.json
        validate_result = validate.validate_add_income(user_data=user_data)

        if(validate_result):
            return validate_result["error"]
        sql_query = "UPDATE user SET total_amount=?, timestamp=? WHERE id=?"
        params = (user_data["amount"], user_data["timestamp"], payload["id"])
        run_status = db.run_sql_update(sql_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200

class SplitIncome(Resource):
    @token_required
    def get(payload, self, id):
        timestamp = id
        # sql_query = "SELECT label, sum(amount) as amount FROM split_income
WHERE user_id=? GROUP BY label ORDER BY LABEL"
        sql_balance = "select label, sum(case when is_income = true then amount else -
amount end) as balance from expense where user_id = ? AND timestamp >= ? group by
label"
        params = (payload["id"])
        # split_data = db.run_sql_select(sql_query, params)
        params = (payload["id"], timestamp)
        balance_data = db.run_sql_select(sql_balance, params)
        return {"balance_data": balance_data}, 200
    @token_required
    def post(payload, self):
        user_data = request.json
        print(user_data)
        validate_result = validate.validate_split_income(user_data=user_data)
        if(validate_result):
            return validate_result["error"]
        sql_query = "INSERT INTO split_income (user_id, amount, label) VALUES(?, ?,
?)"
        params = ( payload["id"], user_data["amount"], user_data["label"])
        run_status = db.run_sql_insert(sql_query, params=params)
        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200

    @token_required
    def delete(payload, self, id):
        labelId = {
                    0: "Food & Drinks",
                    1:"Entertainment",
                    2:"Shopping",
                    3:"Transportation",
                    4:"Vehicle",
                    5:"Trip",
                    6:"General Expense",
                    7:"Financial Expense",
                    8:"Income"
                }
        label = labelId[id]
        sql_query = "DELETE FROM split_income WHERE user_id=? AND label=?"
        params = (payload["id"], label)
        run_status = db.run_sql_delete(sql_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200
```

7.2 Feature 2 - Mail Alert from SendGrid

When our monthly expense is going out of control, this feature is what we need. We can set an amount as limit and when our expense becomes higher than the limit, it will send an alert email regarding the limit exceed. For this feature, we have used SendGrid for mail service.

Coding:

Send Mail:

```
from sendgrid.helpers.mail import Mail
from ..config.mail_config import get_mail_config
from os import getenv

def send_mail(email, data, templateID):
    # try:
    print(email, data, templateID)
    sg = get_mail_config()

    FROM_EMAIL = getenv("FROM_MAIL")
    TO_EMAIL = [(email, 'User')]
    message = Mail(
        from_email=FROM_EMAIL,
        to_emails=TO_EMAIL)
    message.dynamic_template_data = data
    message.template_id = templateID

    response = sg.send(message)
    code, body, headers = response.status_code, response.body, response.headers
    print(f"Response code: {code}")
    print(f"Response headers: {headers}")
    print(f"Response body: {body}")
    print("Dynamic Messages Sent!")
    return True
```

Mail Connection:

```
from dotenv import load_dotenv
from os import getenv

load_dotenv()

def get_mail_config():
    import sendgrid

    sg = sendgrid.SendGridAPIClient(api_key=getenv('SENDGRID_API_KEY'))

    return sg
```

Mail Alert:

```
from flask import request
from flask_restful import Resource
from ..utils import validate, general, db
from ..utils.general import token_required
from ..utils.mail import send_mail

class Alert(Resource):
    @token_required
    def post(payload, self):
        user_data = request.json

        data = {
        "total_amount": user_data["total_amount"],
        "pending_amount": user_data["pending_amount"],
        "percentage": user_data["percentage"],
        "date": user_data["date"]
        }
        templateID = "d-24f02e45da0b4852a23550a0ab1a2478"
        res = send_mail(payload["email"], data, templateID)
        print(res)
        if(not res):
            return {"message": "Error Occured"}, 400
        sql_update_query = "UPDATE user SET is_send=? where id=?"
        params = (user_data["is_send"], payload["id"])
        run_status = db.run_sql_update(sql_update_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "mail sent"}, 200
    @token_required
    def put(payload, self):
        user_data = request.json
        sql_update_query = "UPDATE user SET alert = ?, is_send = ? where id=?"
        params = (user_data["amount"], user_data["is_send"], payload["id"])
        run_status = db.run_sql_update(sql_update_query, params=params)

        if(not run_status):
            return {"message": "Error Occured"}, 400
        return {"message": "Successful"}, 200
```

7.3 Database

DB2 Connection:

```
from dotenv import load_dotenv
from os import getenv
from ibm_db import connect

load_dotenv()

def get_db_credential():
    db_hostname = getenv("DB_HOSTNAME")
    db_uid = getenv("DB_USERNAME")
    db_pwd = getenv("DB_PASSWORD")
    db_db = getenv("DB_DB")
    db_port = getenv("DB_PORT")
    db_protocol = getenv("DB_PROTOCOL")
    db_cert_path = getenv("DB_CERT_PATH")

    db_crediential = (
        "DATABASE={0};"
        "HOSTNAME={1};"
        "PORT={2};"
        "PROTOCOL={3};"
        "UID={4};"
        "PWD={5};"
        "SECURITY=SSL;"
        "SSLServerCertificate={6}"
    ).format(db_db, db_hostname, db_port, db_protocol, db_uid, db_pwd, db_cert_path)

    return db_crediential
```

DB2 Manipulation:

```
import ibm_db
from ..config.db_config import get_db_credential

conn=ibm_db.connect(get_db_credential(),"","")

def run_sql_select(query,params=None):
    try:
        stmt=ibm_db.prepare(conn,query)
        if(params==None):
            ibm_db.execute(stmt)
        else:
            ibm_db.execute(stmt,params)
        row = ibm_db.fetch_assoc(stmt)
        data = []
        while(row):
            data.append(row)
            row = ibm_db.fetch_assoc(stmt)
        return data
    except:
        return False

def run_sql_insert(query,params):
    try:
        stmt=ibm_db.prepare(conn,query)
        ibm_db.execute(stmt,params)
        print('true')
        return True
    except:
        print('false')
        return False

def run_sql_update(query, params):
    try:
        stmt=ibm_db.prepare(conn, query)
        ibm_db.execute(stmt, params)
        print('true')
        return True
    except:
        return False

def run_sql_delete(query, params):
    try:
        stmt=ibm_db.prepare(conn, query)
        ibm_db.execute(stmt, params)
        print('true')
        return True
    except:
        return False
```

8. TESTING

8.1 Test Cases

1. Login Page (Functional)
2. Login Page (UI)
3. Add Expense (Functional)
4. Add Income (Functional)
5. Expense Stats (UI)

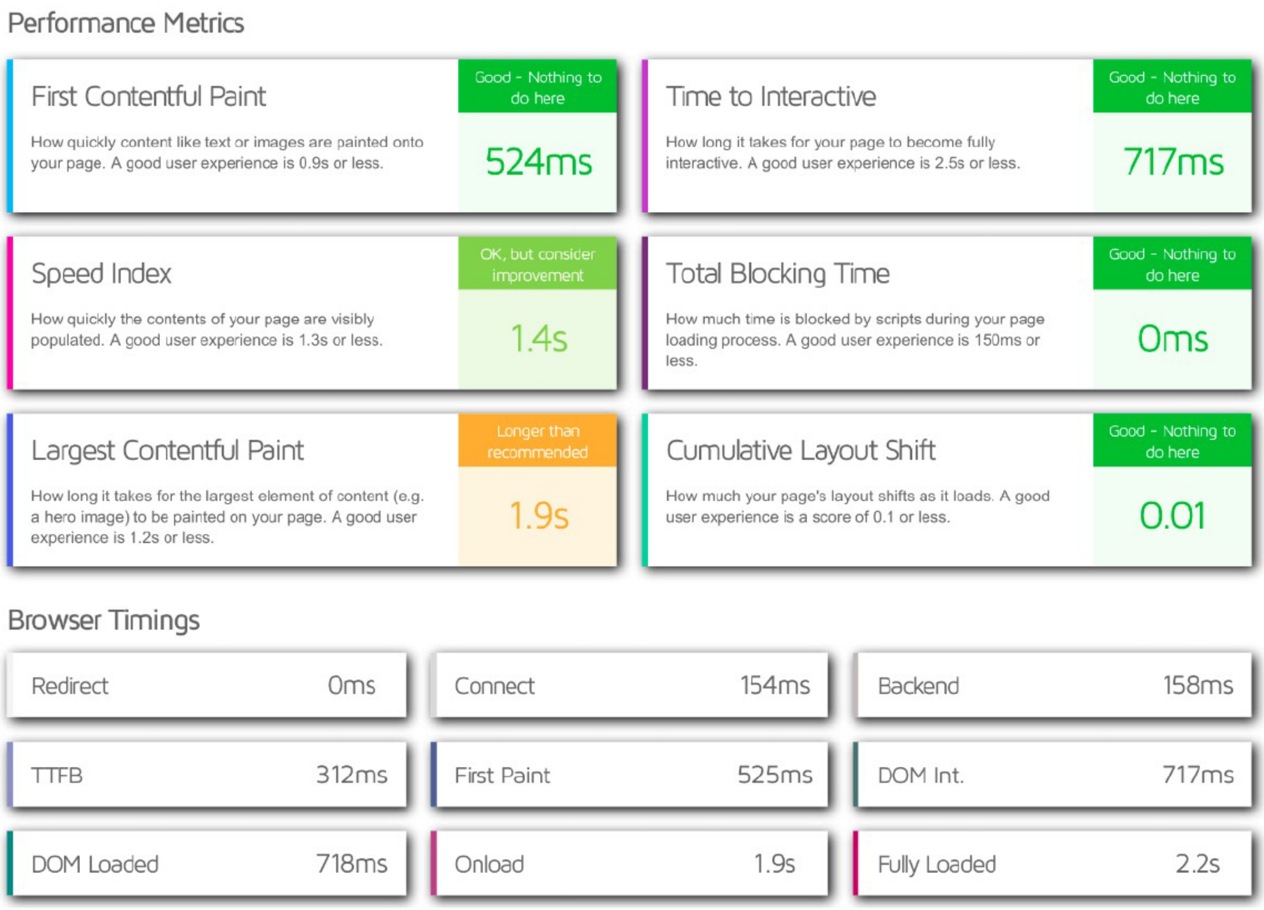
8.2 User Acceptance Testing

User Story Number	User Story / Task	Acceptance Criteria
USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	- There should be a register page. - the page should contain input field for fields like email, password and confirm password. - Form should contain a sign up button to submit details.
USN-2	As a user, I will receive confirmation email once I have registered for the application	- The user must receive the confirmation mail on their repective email which should contain link to confirm mail. - the link should open a webpage which gives confirmation for mail.
USN-3	As a user, I can register for the application through Gmail	- the sign up page should have option for users to select their gmail for registration.
USN-4	As a user, I can log into the application by entering email & password	- there should be a page for users to login. - the page should contain the input fields for email and password fields. - a login button which logs the user in.
USN-5	As a user, I can enter my income and expenses.	- the main homepage of application should show the income and expenses of user. - the page should contain buttons for adding income and expenses.

USN-6	As a user, I can enter a limit to set for the expenses	- the homepage should show the limit entered by the user. - the page should also contain button to edit this limit.
USN-7	As a user, I can add money in my wallet	- the page should display the money currently in wallet. - the page should also contain a button to edit and update the value in the wallet.
USN-8	As a user, I will get an alert mail when my monthly limit exceeds.	- the user is expected to receive a mail from the app if the limit ie exceeded. - the mail should contain information related to expenses
USN-9	As a user, I can get my monthly report in a spreadsheet format.	- the user should have provision in the end of mon th to view the montly expenses and income in a excel sheet report.
USN-10	As a user, I can get periodical report in Graph format using the data.	- the user should have provision to view the report in graph format as well.
USN-11	As a user, I can access my data from anywhere. Deploying the application in cloud.	- the data should be available from anywhere.

9. RESULTS

9.1 Performance Metrics



Performance Metrics

First Contentful Paint

How quickly content like text or images are painted onto your page. A good user experience is 0.9s or less.

Good - Nothing to do here

524ms

Time to Interactive

How long it takes for your page to become fully interactive. A good user experience is 2.5s or less.

Good - Nothing to do here

717ms

Speed Index

How quickly the contents of your page are visibly populated. A good user experience is 1.3s or less.

OK, but consider improvement

1.4s

Total Blocking Time

How much time is blocked by scripts during your page loading process. A good user experience is 150ms or less.

Good - Nothing to do here

0ms

Largest Contentful Paint

How long it takes for the largest element of content (e.g. a hero image) to be painted on your page. A good user experience is 1.2s or less.

Longer than recommended

1.9s

Cumulative Layout Shift

How much your page's layout shifts as it loads. A good user experience is a score of 0.1 or less.

Good - Nothing to do here

0.01

Browser Timings

Redirect	0ms	Connect	154ms	Backend	158ms
TTFB	312ms	First Paint	525ms	DOM Int.	717ms
DOM Loaded	718ms	Onload	1.9s	Fully Loaded	2.2s

10. ADVANTAGES & DISADVANTAGES

Advantages:

- A Personal expense tracker application helps you decide between short-term and long-term spending. It will give more control over our money spending.
- When we forget to add expenses in the app, this application can send a reminder so that we would not forget.
- The Limit on our monthly expenses will send email when the limit exceeds which keeps us on budget every month.
- The application can make us able to save some money for our future and can be helpful in investments too.

Disadvantages:

- **Human Error:** Since we have to input our expense manually, it is not possible that we can remember all the expenses that we made during weekends outing.
- **Security issues:** Eventhough cloud-based IBM Db2 can protect our information, any data connected to the Internet, in theory, can be breached. So lack of security for our personal data.

11. CONCLUSION

Tracking our expenses will creae financial goals for our future even more than we can imagine instead of just saving money for a month. When we get to know where we spend most of our expenses on, it will really be helpful for us to make us realize on our spendings so we can cut off on our expenditure. The project helped us learn about the fundamentals of cloud services. We interacted with many IBM services and learnt about their applications and the tools provided to us by IBM helped us build this Personal Expense Tracker Application.

Apart from IBM services we also learnt about the fundamentals of creating the flask application and some other 3rd party services such as SendGrid to send mails from flask server. We also learnt the power of Docker and Kubernetees in development and deployment of web applications. In our team's personal opinion, overall it was a nice experience getting to build this project by learning the new things.

12. FUTURE SCOPE

Our Personal Expense Tracker Application can be integrated with the UPI Id so that it can automatically add our expenses and our debits transactions from the UPI apps such as GPay, Paytm,etc,. which reduces our manual input. By using Machine Learning techniques, we can implement the feature to the app to suggest some spending advises about we are spending more money on this category so that we can reduce it if we find it unnecessary.

13. APPENDIX

Github Repo Link: **https://github.com/IBM-EPBL/IBM-Project-11152-1659270700**

Demo Video Link: **Demo Video**

Source Code: **Source Code**