Personal Expense Tracker Application

Project Report

Team ID	PNT2022TMID50619
Project Name	Project – Personal Expense Tracker Application
Team Members	**K. Vijaya Vignesh (TL) - 952819106304** **M. Esakkiraja@vishal - 9528219106007** **M. SelvaSundar - 952819106028** **M. Durai raja -952819106005** **S. Senthilvel -952819106029**

CONTENTS

1. INTRODUCTION

1.1 Project overview

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

1.2 Purpose

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

2. LITERATURE SURVEY

2.1 Existing Problem

The problem of current generation population is that they can’t remember where all of the money 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.

2.2 Reference

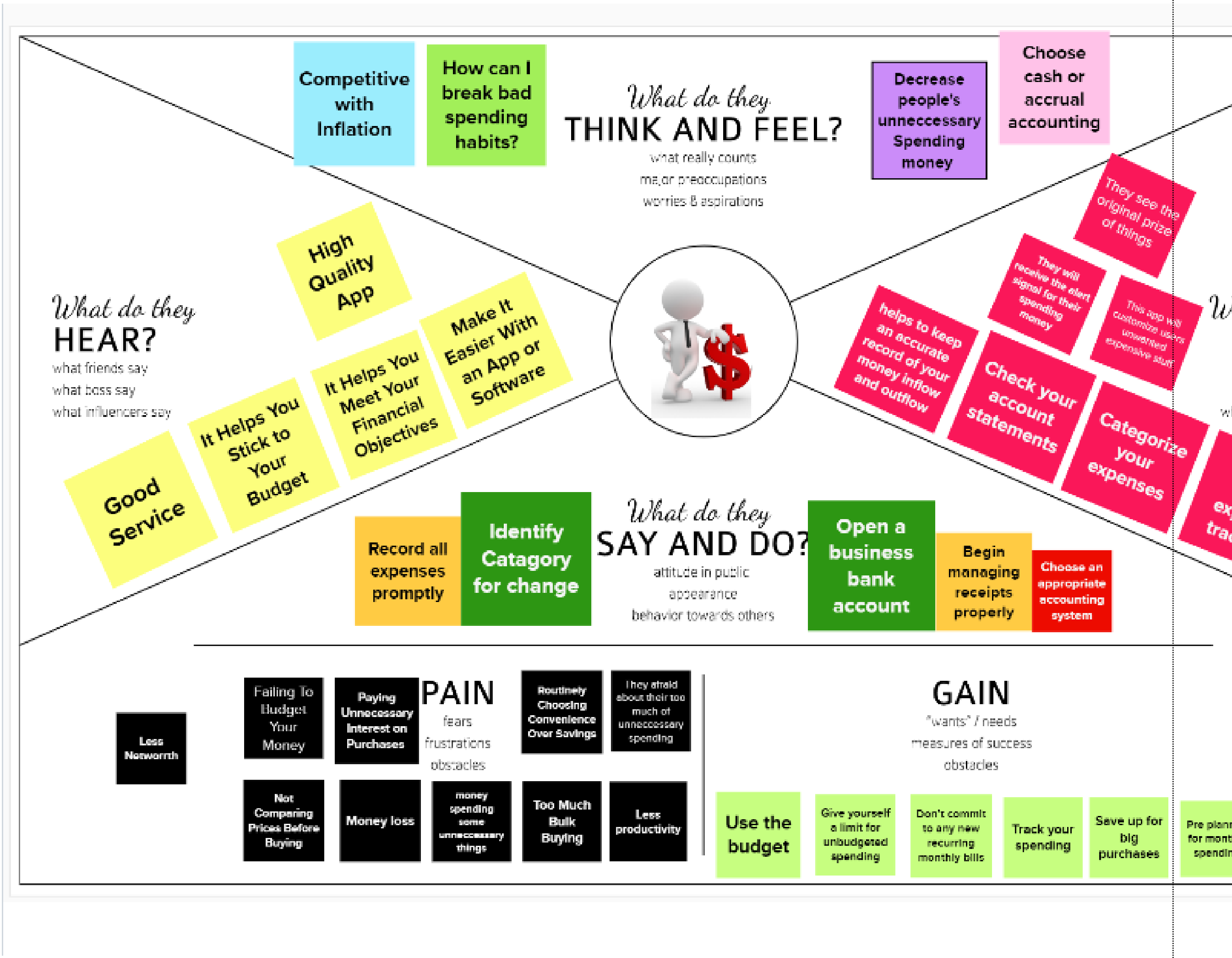
- https://nevonprojects.com/daily-expense-tracker-system/
- https://data-flair.training/blogs/expense-tracker-python/
- https://phpgurukul.com/daily-expense-tracker-using-php-and-mysql/
- https://ijarsct.co.in/Paper391.pdf
- https://kandi.openweaver.com/?landingpage=python_all_projects&utm_source=google&utm_medium=cpc&utm_campaign=promo_kandi_ie&utm_content=kandi_ie_search&utm_term=python_devs&gclid=Cj0KCQiAgribBhDkARIsAASA5bukrZgbI9UZxzpoyf0PofB1mZNxzcokUP-3TchpYMclHTYFYiqP8aAmmwEALw_wcB

2.3 Problem Statement Definition

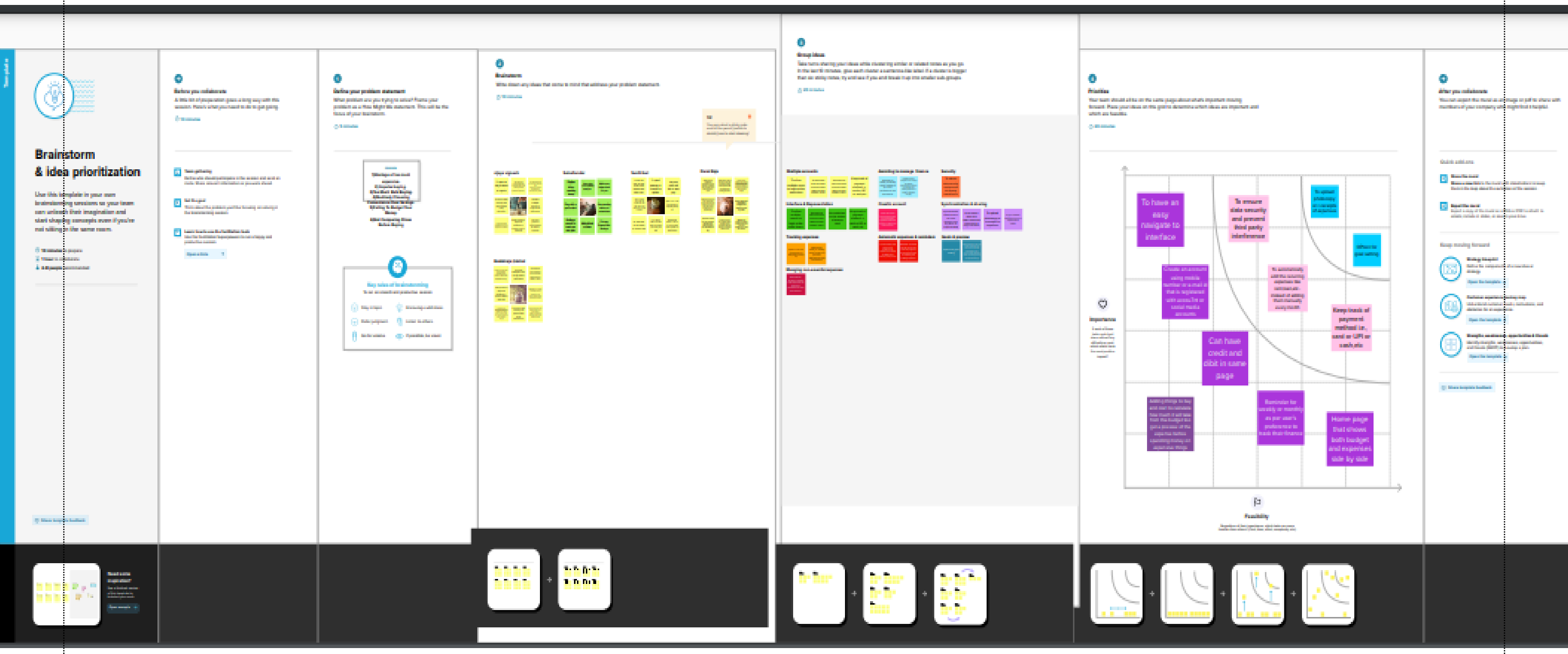
This Expense Tracker is a web application that facilitates the users to keep track and manage their personal as well as business expenses. This application helps the users to keep a digital diary. It will keep track of a user's income and expenses daily. The user will be able to add his/her expenditures 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 must enter the income and expenditures and everything else will be performed by the system. Keywords: Expense Tracker, budget, planning, savings, graphical visualization of expenditure.

3. IDEATION & PROPOSED SOLUTION

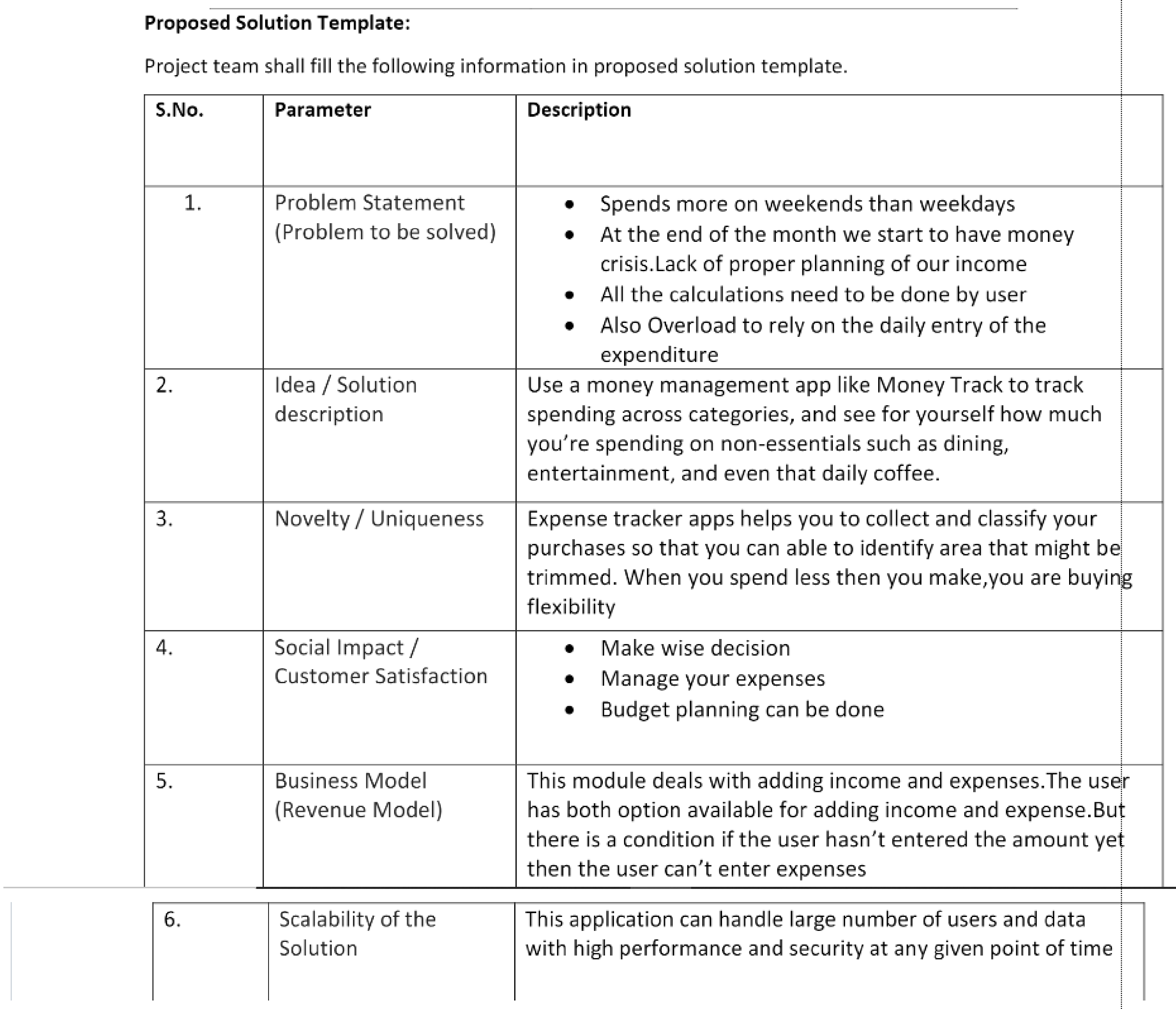
3.1 Empathy Map canvas


Competitive with Inflation
How can I break bad spending habits?
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
Decrease people's unneccessary Spending money
Choose cash or accrual accounting
They see the original prize of things
They will receive the alert signal for their spending money
This app will customize users unwanted expensive stuff
helps to keep an accurate record of your money inflow and outflow
Check your account statements
Categorize your expenses
High Quality App
What do they HEAR?
what friends say
what boss say
what influencers say
Good Service
It Helps You Stick to Your Budget
It Helps You Meet Your Financial Objectives
Make It Easier With an App or Software
Record all expenses promptly
Identify Catagory for change
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
Open a business bank account
Begin managing receipts properly
Choose an appropriate accounting system
Less Networth
Failing To Budget Your Money
Paying Unnecessary Interest on Purchases
PAIN
fears
frustrations
obstacles
Routinely Choosing Convenience Over Savings
They afraid about their too much of unneccessary spending
Not Comparing Prices Before Buying
Money loss
money spending some unneccessary things
Too Much Bulk Buying
Less productivity
GAIN
"wants" / needs
measures of success
obstacles
Use the budget
Give yourself a limit for unbudgeted spending
Don't commit to any new recurring monthly bills
Track your spending
Save up for big purchases

3.2 Ideation & Brainstorming



Proposed Solution

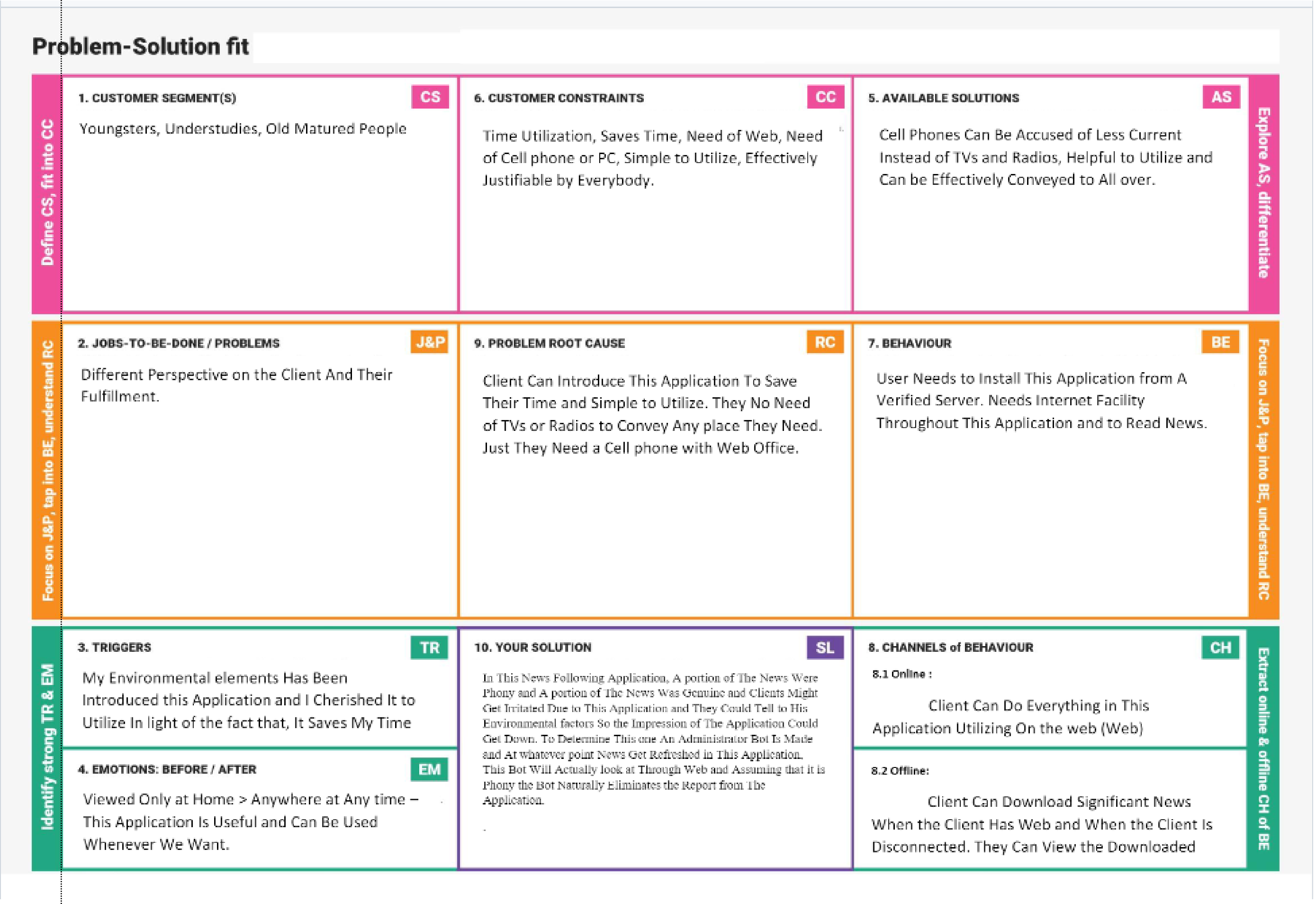


Proposed Solution Template:

Project team shall fill the following information in proposed solution template.

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	• Spends more on weekends than weekdays • At the end of the month we start to have money crisis.Lack of proper planning of our income • All the calculations need to be done by user • Also Overload to rely on the daily entry of the expenditure
2.	Idea / Solution description	Use a money management app like Money Track to track spending across categories, and see for yourself how much you're spending on non-essentials such as dining, entertainment, and even that daily coffee.
3.	Novelty / Uniqueness	Expense tracker apps helps you to collect and classify your purchases so that you can able to identify area that might be trimmed. When you spend less then you make,you are buying flexibility
4.	Social Impact / Customer Satisfaction	• Make wise decision • Manage your expenses • Budget planning can be done
5.	Business Model (Revenue Model)	This module deals with adding income and expenses.The user has both option available for adding income and expense.But there is a condition if the user hasn't entered the amount yet then the user can't enter expenses
6.	Scalability of the Solution	This application can handle large number of users and data with high performance and security at any given point of time

3.4Proposed Solution Fit



Problem-Solution fit

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Youngsters, Understudies, Old Matured People

6. CUSTOMER CONSTRAINTS CC

Time Utilization, Saves Time, Need of Web, Need of Cell phone or PC, Simple to Utilize, Effectively Justifiable by Everybody.

5. AVAILABLE SOLUTIONS AS

Cell Phones Can Be Accused of Less Current Instead of TVs and Radios, Helpful to Utilize and Can be Effectively Conveyed to All over.

Explore AS, differentiate

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

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

Different Perspective on the Client And Their Fulfillment.

9. PROBLEM ROOT CAUSE RC

Client Can Introduce This Application To Save Their Time and Simple to Utilize. They No Need of TVs or Radios to Convey Any place They Need. Just They Need a Cell phone with Web Office.

7. BEHAVIOUR BE

User Needs to Install This Application from A Verified Server. Needs Internet Facility Throughout This Application and to Read News.

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

Identify strong TR & EM

3. TRIGGERS TR

My Environmental elements Has Been Introduced this Application and I Cherished It to Utilize In light of the fact that, It Saves My Time

4. EMOTIONS: BEFORE / AFTER EM

Viewed Only at Home > Anywhere at Any time – This Application Is Useful and Can Be Used Whenever We Want.

10. YOUR SOLUTION SL

In This News Following Application, A portion of The News Were Phony and A portion of The News Was Genuine and Clients Might Get Irritated Due to This Application and They Could Tell to His Environmental factors So the Impression of The Application Could Get Down. To Determine This one An Administrator Bot Is Made and At whatever point News Get Refreshed in This Application, This Bot Will Actually look at Through Web and Assuming that it is Phony the Bot Naturally Eliminates the Report from The Application.

.

8. CHANNELS of BEHAVIOUR CH

8.1 Online :

Client Can Do Everything in This Application Utilizing On the web (Web)

8.2 Offline:

Client Can Download Significant News When the Client Has Web and When the Client Is Disconnected. They Can View the Downloaded

Extract online & offline CH of BE

4. REQUIREMENT ANALYSIS

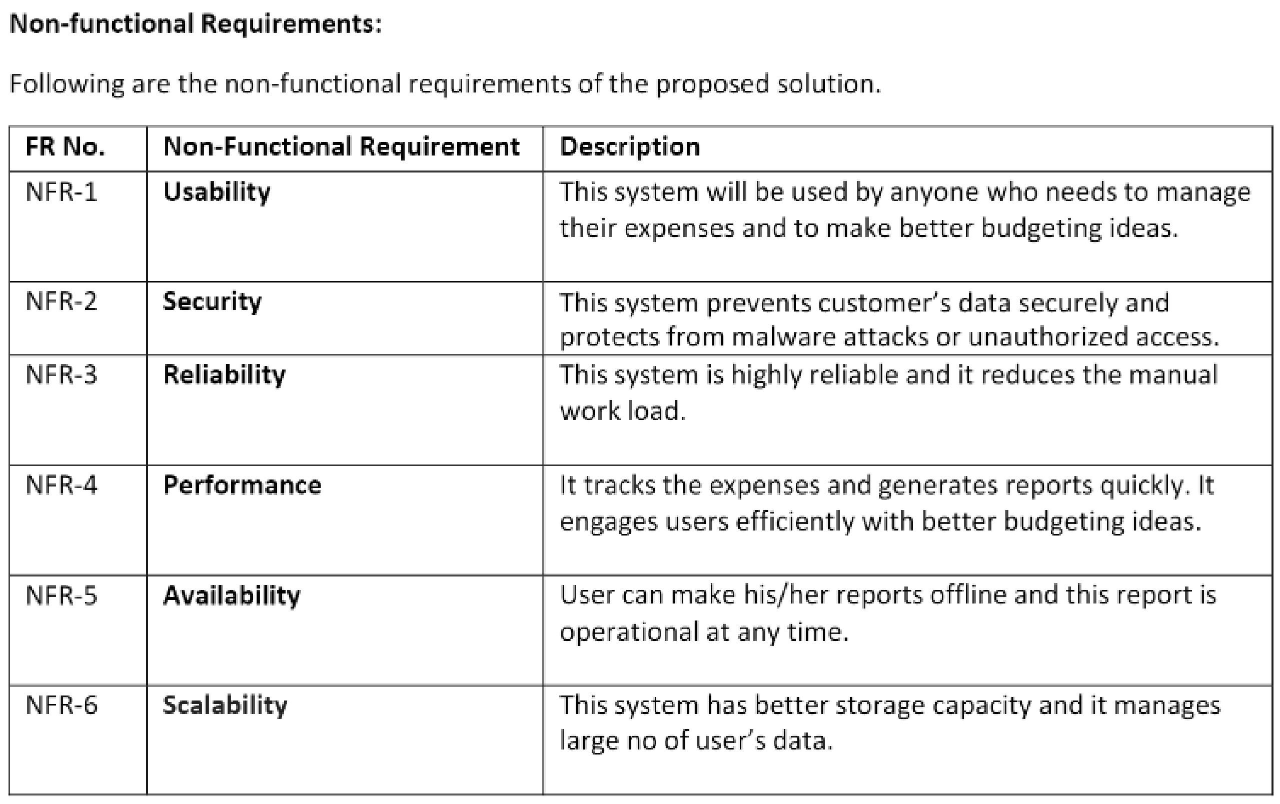
4.1 Functional requirement

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.

4.2 Non-Functional requirement



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.

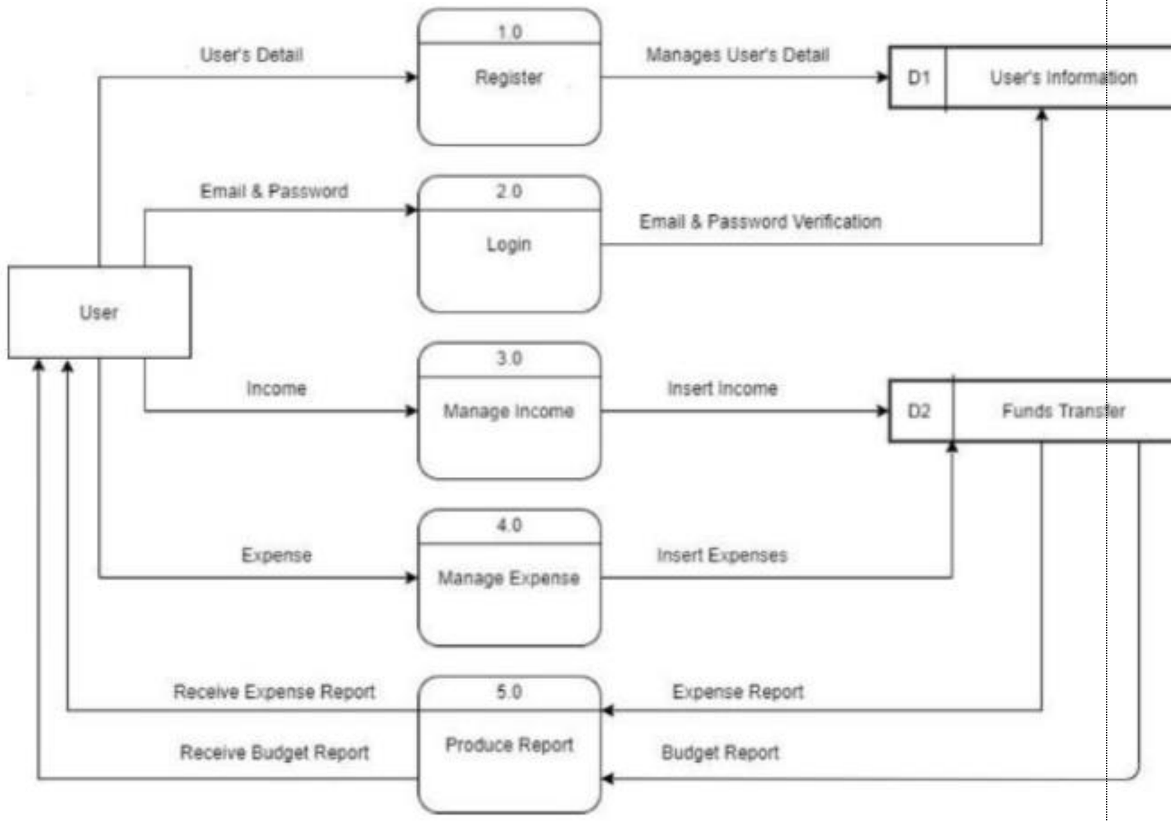
5. PROJECT DESIGN

5.1 Data Flow Diagrams

Data Flow Diagrams:

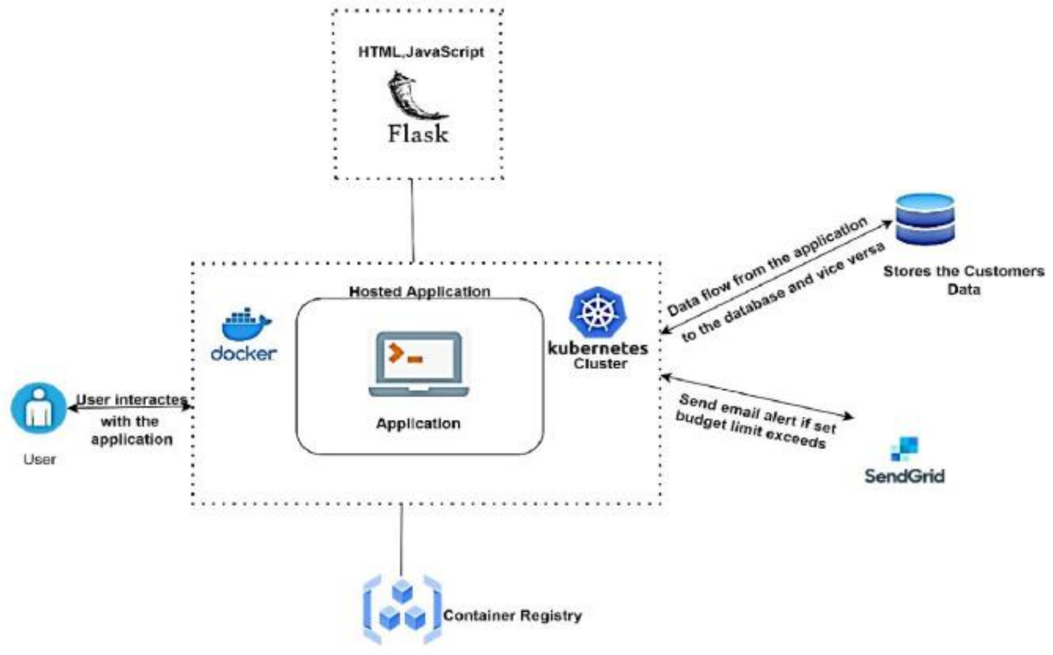
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system.. It shows how data enters and leaves the system, what changes the information, and where data is stored.

.



5.2 Solution & Technical Architecture

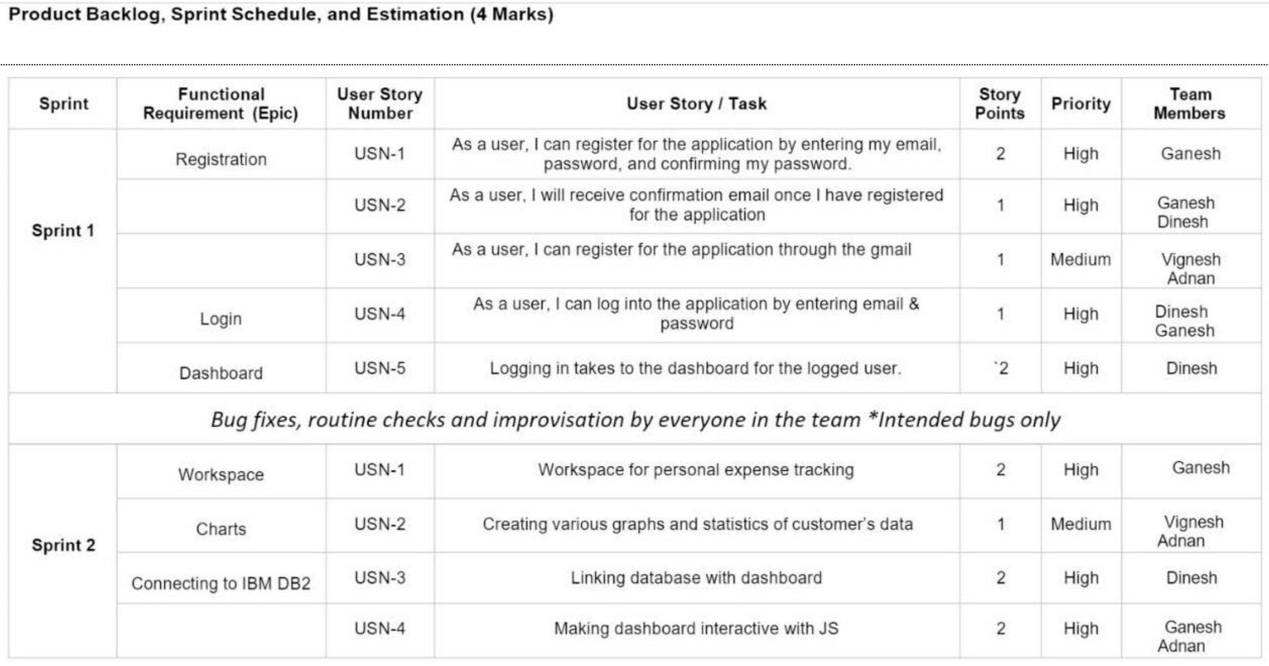
Technology Architecture:


HTML,JavaScript
Flask
Hosted Application
docker
Application
kubernetes Cluster
Data flow from the application to the database and vice versa
Stores the Customers Data
User interactes with the application
User
Send email alert if set budget limit exceeds
SendGrid
Container Registry

5.3 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, and password, and confirming my password.	I can access my account/dash board	High	Sprint-1
		USN-2	As a user, I will receive a confirmation email once I have registered for the application	I can receive a confirmation email & click confirm	High	Sprint-1
		USN-3	As a user, I can register for the application through Facebook	I can register & access the dashboard with Facebook Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through a Google account.	I can register & access the dashboard with a Google Account login.	Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering my email & Password	I can access the applicatio n.	High	Sprint-1
	Dashboard	USN-6	As a user, I can see the expenditure details and the daily expense.	I can view the daily expenses and add the expense details.	High	Sprint-1
Customer Care Executive		USN-7	As a customer care executive, I can solve the problem that customers face.	I can provide support to customers at any time 24*7.	Medium	Sprint-1
Administrator	Application	USN-8	As an administrator, I can upgrade or update the application.	I can fix any bugs raised by customers and upgrade the application.	Medium	Sprint-1

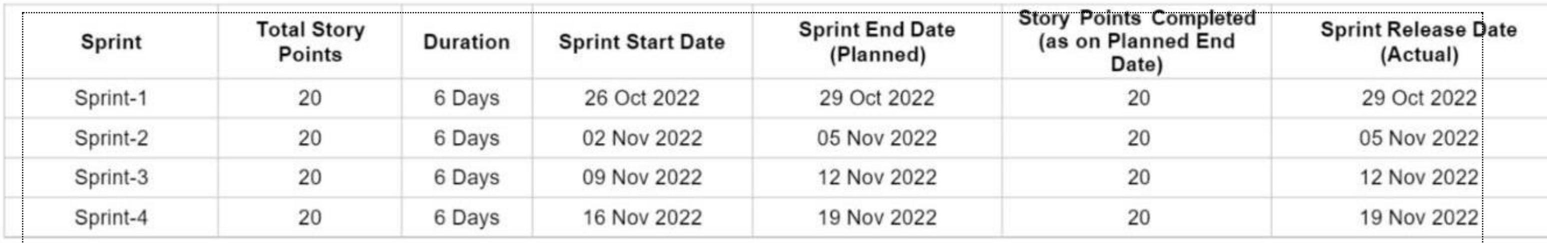

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

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.	2	High	Ganesh
		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Ganesh Dinesh
		USN-3	As a user, I can register for the application through the gmail	1	Medium	Vignesh Adnan
	Login	USN-4	As a user, I can log into the application by entering email & password	1	High	Dinesh Ganesh
	Dashboard	USN-5	Logging in takes to the dashboard for the logged user.	`2	High	Dinesh
*Bug fixes, routine checks and improvisation by everyone in the team *Intended bugs only*						
Sprint 2	Workspace	USN-1	Workspace for personal expense tracking	2	High	Ganesh
	Charts	USN-2	Creating various graphs and statistics of customer's data	1	Medium	Vignesh Adnan
	Connecting to IBM DB2	USN-3	Linking database with dashboard	2	High	Dinesh
		USN-4	Making dashboard interactive with JS	2	High	Ganesh Adnan

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

Project Tracker, Velocity & Burndown Chart: (4 Marks)

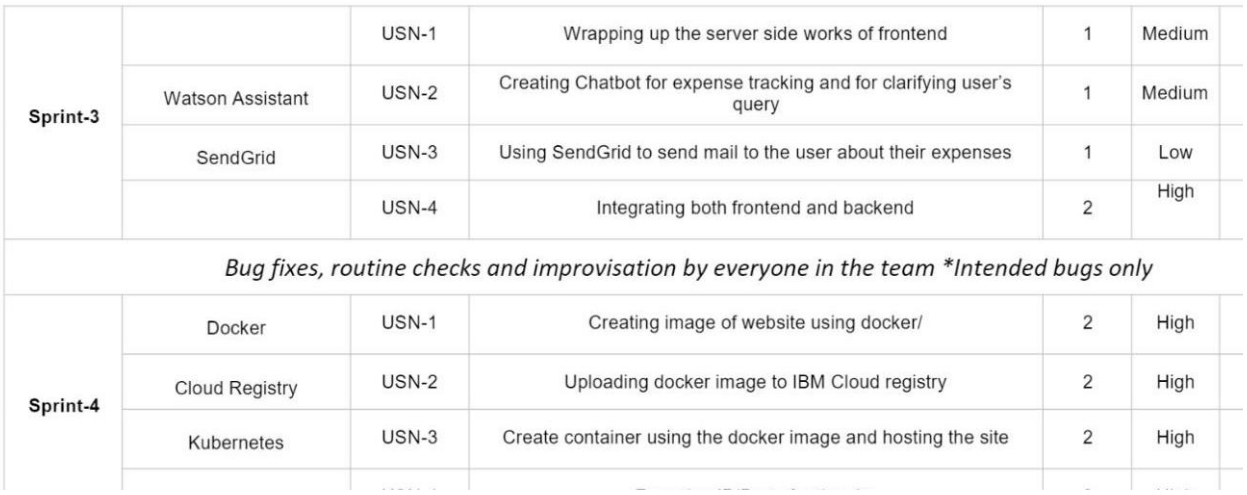


Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	26 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	02 Nov 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	6 Days	09 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	6 Days	16 Nov 2022	19 Nov 2022	20	19 Nov 2022

Velocity

We have a 6-day sprint duration, and the velocity of the team is 20 (points per sprint). Calculating the team's average velocity (AV).

$$AV = \frac{\text{sprint duration}}{\text{velocity}} = \frac{20}{6} = 3.33$$



Sprint-3		USN-1	Wrapping up the server side works of frontend	1	Medium	
	Watson Assistant	USN-2	Creating Chatbot for expense tracking and for clarifying user's query	1	Medium	
	SendGrid	USN-3	Using SendGrid to send mail to the user about their expenses	1	Low	
		USN-4	Integrating both frontend and backend	2	High	
*Bug fixes, routine checks and improvisation by everyone in the team *Intended bugs only*						
Sprint-4	Docker	USN-1	Creating image of website using docker/	2	High	
	Cloud Registry	USN-2	Uploading docker image to IBM Cloud registry	2	High	
	Kubernetes	USN-3	Create container using the docker image and hosting the site	2	High	

6.2Sprint Delivery Schedule

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

Track your expenses anywhere, anytime. Seamlessly manage your money and budget without any financial paperwork. Just click and submit your invoices and expenditures. Access, submit, and approve invoices irrespective of time and location. Avoid data loss by scanning your tickets and bills and saving in the app. Approval of bills and expenditures in real-time and get notified instantly. Quick settlement of claims and reduced human errors with an automated and streamlined billing process.

Code

```
import os
import re
import expenze_categories
```

```
from flask import request, session
from flask_session import Session
from sqlalchemy import create_engine
from sqlalchemy.orm import scoped_session, sessionmaker
from datetime import datetime
from helpers import convertSQLToDict

# Create engine object to manage connections to DB, and
scoped session to separate user interactions with DB
engine = create_engine(os.getenv("DATABASE_URL"))
db = scoped_session(sessionmaker(bind=engine))


 # Get the users budgets
def getBudgets(userID):
    results = db.execute(
       "SELECT id, name, year, amount FROM budgets WHERE
user_id = :usersID ORDER BY name ASC", {"usersID":
userID}).fetchall()

   budgets_query = convertSQLToDict(results)

   if budgets_query:
       # Create a dict with budget year as key and empty list as
value which will store all budgets for that year
```

```
        budgets = {budget['year']: [] for budget in
budgets_query}

        # Update the dict by inserting budget info as values
        for budget in budgets_query:
            budgets[budget['year']].append(
                {'amount': budget['amount'], 'id': budget['id'],
'name': budget['name']})

        return budgets
    else:
        return None


# Get a users budget by the budget ID
def getBudgetByID(budgetID, userID):
    results = db.execute(
        "SELECT name, amount, year, id FROM budgets WHERE
user_id = :usersID AND id = :budgetID", {"usersID": userID,
"budgetID": budgetID}).fetchall()

    budget = convertSQLToDict(results)

    return budget[0]
```

```
# Get total amount budgeted by year
def getTotalBudgetedByYear(userID, year=None):

    # Default to getting current years budgets
    if not year:
        year = datetime.now().year

    amount = db.execute(
        "SELECT SUM(amount) AS amount FROM budgets 
WHERE user_id = :usersID AND year = :year", {"usersID": 
userID, "year": year}).fetchone()[0]

    if amount is None:
        return 0
    else:
        return amount


# Generates a budget data structure from the users input 
when submitting a new or updated budget
def generateBudgetFromForm(formData):
    budget = {"name": None, "year": None, "amount": None, 
"categories": []}
    counter = 0
    # Loop through all of the form data to extract budgets
details and store in the budget dict
    for key, value in formData:
        counter += 1
        # First 3 keys represent the name/year/amount from the
form, all other keys represent dynamically loaded categories
from the form
        if counter <= 3:
            # Check name for invalid chars and uniqueness
            if key == "name":
                # Invalid chars are all special chars except
underscores, spaces, and hyphens (uses same regex as what's
on the HTML page)
                validBudgetName = re.search("^([a-zA-Z0-9_\s\-
]*)$", value)
                if validBudgetName:
                    budget[key] = value.strip()
                else:
                    return {"apology": "Please enter a budget name
without special characters except underscores, spaces, and
hyphens"}
            # Check if year is valid
            elif key == "year":
                budgetYear = int(value)
                currentYear = datetime.now().year
                if 2020 <= budgetYear <= currentYear:
                    budget[key] = budgetYear
                else:
                    return {"apology": f"Please select a valid budget
year: 2020 through {currentYear}"}
            # Convert the amount from string to float
            else:
                amount = float(value.strip())
                budget[key] = amount
        # All other keys will provide the *category* name /
percent budgeted
        else:
            # Skip iteration if value is empty (empty means the
user doesnt want the category in their budget)
            if value == '':
                continue

            # Need to split the key since the HTML elements are
loaded dynamically and named like 'categories.1',
'categories.2', etc.
            cleanKey = key.split(".")

            # Store the category name and associated % the user
wants budgetd for the category
            category = {"name": None, "percent": None}
            if cleanKey[0] == "categories":
                category["name"] = value.strip()

                # Get the percent value and convert to decimal
                percent = (int(formData[counter][1].strip()) / 100)
                category["percent"] = percent

                # Add the category to the list of categories within
the dict
                budget[cleanKey[0]].append(category)
            # Pass on this field because we grab the percent
above (Why? It's easier to keep these 2 lines than rewrite
many lines. This is the lowest of low pri TODOs)
            elif cleanKey[0] == "categoryPercent":
                pass
            else:
                return {"apology": "Only categories and their
percentage of the overall budget are allowed to be stored"}

    return budget


# Create a new budget
# Note: due to DB design, this is a 2 step process: 1) create a
budget (name/year/amount) in budgets table, 2) create 1:M
records in budgetCategories (budgetID + categoryID +
percentAmount)
```

```
def createBudget(budget, userID):
    # Verify the budget name is not a duplicate of an existing
budget
    uniqueBudgetName =
isUniqueBudgetName(budget["name"], None, userID)
    if not uniqueBudgetName:
        return {"apology": "Please enter a unique budget name,
not a duplicate."}

    # Insert new budget into DB
    newBudgetID = db.execute("INSERT INTO budgets (name,
year, amount, user_id) VALUES (:budgetName, :budgetYear,
:budgetAmount, :usersID) RETURNING id",
                             {"budgetName": budget["name"],
"budgetYear": budget["year"], "budgetAmount":
budget["amount"], "usersID": userID}).fetchone()[0]
    db.commit()

    # Get category IDs from DB for the new budget
    categoryIDS =
getBudgetCategoryIDS(budget["categories"], userID)

    # Insert a record for each category in the new budget
    addCategory(newBudgetID, categoryIDS)

    return budget
```

```
# When creating or updating a budget, add the spending
categories and % budgeted per category to a budgets record
in the DB
def addCategory(budgetID, categoryIDS):
    # Insert a record for each category in the new budget
    for categoryID in categoryIDS:
        db.execute("INSERT INTO budgetCategories
(budgets_id, category_id, amount) VALUES (:budgetID,
:categoryID, :percentAmount)",
                  {"budgetID": budgetID, "categoryID":
categoryID["id"], "percentAmount": categoryID["amount"]})
    db.commit()


# Update an existing budget
# Note: due to DB design, this is a 3 step process: 1) update a
budget (name/year/amount) in budgets table, 2) delete the
existing spending categories for the budget, 3) create 1:M
records in budgetCategories (budgetID + categoryID +
percentAmount)
def updateBudget(oldBudgetName, budget, userID):
    # Query the DB for the budget ID
    oldBudgetID = getBudgetID(oldBudgetName, userID)
    # Verify the budget name is not a duplicate of an existing
budget
    uniqueBudgetName = isUniqueBudgetName(
        budget["name"], oldBudgetID, userID)
    if not uniqueBudgetName:
        return {"apology": "Please enter a unique budget name,
not a duplicate."}

    # Update the budget name, year, and amount in DB
    db.execute("UPDATE budgets SET name = :budgetName,
year = :budgetYear, amount = :budgetAmount WHERE id =
:oldBudgetID AND user_id = :usersID",
               {"budgetName": budget["name"], "budgetYear":
budget["year"], "budgetAmount": budget["amount"],
"oldBudgetID": oldBudgetID, "usersID": userID})
    db.commit()

    # Delete existing category records for the budget
    db.execute("DELETE FROM budgetCategories WHERE
budgets_id = :oldBudgetID",
               {"oldBudgetID": oldBudgetID})
    db.commit()

    # Get category IDs from DB for the new budget
    categoryIDS =
getBudgetCategoryIDS(budget["categories"], userID)
```

```
    # Insert a record for each category in the new budget
    addCategory(oldBudgetID, categoryIDS)

    return budget


# Get a budgets associated category ids
def getBudgetCategoryIDS(categories, userID):
    # Get the category IDs from the DB for the updated
budget
    categoryIDS = []
    for category in categories:
        # Get the category ID
        categoryID = db.execute("SELECT categories.id FROM
userCategories INNER JOIN categories ON
userCategories.category_id = categories.id WHERE
userCategories.user_id = :usersID AND categories.name =
:categoryName",
                                {"usersID": userID, "categoryName":
category["name"]}).fetchone()[0]

        # Store the category ID and associated percent amount
into a dict
        id_amount = {"id": None, "amount": None}
        id_amount["id"] = categoryID
        id_amount["amount"] = category["percent"]

        # Add the dictionary to the list of categoryIDs
        categoryIDS.append(id_amount)

    return categoryIDS


# Delete an existing budget
def deleteBudget(budgetName, userID):
    # Query the DB for the budget ID
    budgetID = getBudgetID(budgetName, userID)

    if budgetID:
        # Delete the records for budgetCategories
        db.execute("DELETE FROM budgetCategories WHERE
budgets_id = :budgetID",
                   {"budgetID": budgetID})
        db.commit()

        # Delete the budget
        db.execute("DELETE FROM budgets WHERE id =
:budgetID",
                   {"budgetID": budgetID})
        db.commit()
        return budgetName
    else:
        return None


# Get budget ID from DB
def getBudgetID(budgetName, userID):
    # Query the DB for a budget ID based on the user and the
supplied budget name
    budgetID = db.execute("SELECT id FROM budgets WHERE
user_id = :usersID AND name = :budgetName",
                          {"usersID": userID, "budgetName":
budgetName}).fetchone()[0]

    if not budgetID:
        return None
    else:
        return budgetID


# Get and return a bool based on whether or not a
new/updated budget name already exists for the user
def isUniqueBudgetName(budgetName, budgetID, userID):
    if budgetID == None:
        # Verify the net-new created budget name is not already
existing in the users existing budgets
```

```
        results = db.execute(
            "SELECT name FROM budgets WHERE user_id = 
:usersID", {"usersID": userID}).fetchall()
        existingBudgets = convertSQLToDict(results)
    else:
        # Verify the updated budget name is not already 
existing in the users existing budgets
        results = db.execute(
            "SELECT name FROM budgets WHERE user_id = 
:usersID AND NOT id = :oldBudgetID", {"usersID": userID, 
"oldBudgetID": budgetID}).fetchall()
        existingBudgets = convertSQLToDict(results)

    # Loop through all budgets and compare names
    isUniqueName = True
    for budget in existingBudgets:
        if budgetName.lower() == budget["name"].lower():
            isUniqueName = False
            break

    if isUniqueName:
        return True
    else:
        return False
```

```
# Generate a complete, updatable budget that includes the
budget name, amount, and all categories
(selected/unselected categories and % budgeted for)
def getUpdatableBudget(budget, userID):

    # Get the users library of spend categories
    categories =
expenze_categories.getSpendCategories(userID)

    # Get the budget's spend categories and % amount for
each category
    results = db.execute("SELECT DISTINCT categories.name,
budgetCategories.amount FROM budgetCategories INNER
JOIN categories ON budgetCategories.category_id =
categories.id INNER JOIN budgets ON
budgetCategories.budgets_id = budgets.id WHERE
budgets.id = :budgetsID",
                         {"budgetsID": budget["id"]}).fetchall()
    budgetCategories = convertSQLToDict(results)

    # Add 'categories' as a new key/value pair to the existing
budget dict
    budget["categories"] = []

    # Populate the categories by looping through and adding
all their categories
```

```
    for category in categories:
        for budgetCategory in budgetCategories:
            # Mark the category as checked/True if it exists in the
budget that the user wants to update
            if category["name"] == budgetCategory["name"]:
                # Convert the percentage (decimal) into a whole
integer to be consistent with UX
                amount = round(budgetCategory["amount"] * 100)
                budget["categories"].append(
                    {"name": category["name"], "amount": amount,
"checked": True})
                break
        else:
            budget["categories"].append(
                {"name": category["name"], "amount": None,
"checked": False})

    return budget
```

8.TESTING:

8.1 TESTING:

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

8.2 User Acceptance Testing:

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of [product name] project 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 are resolved.

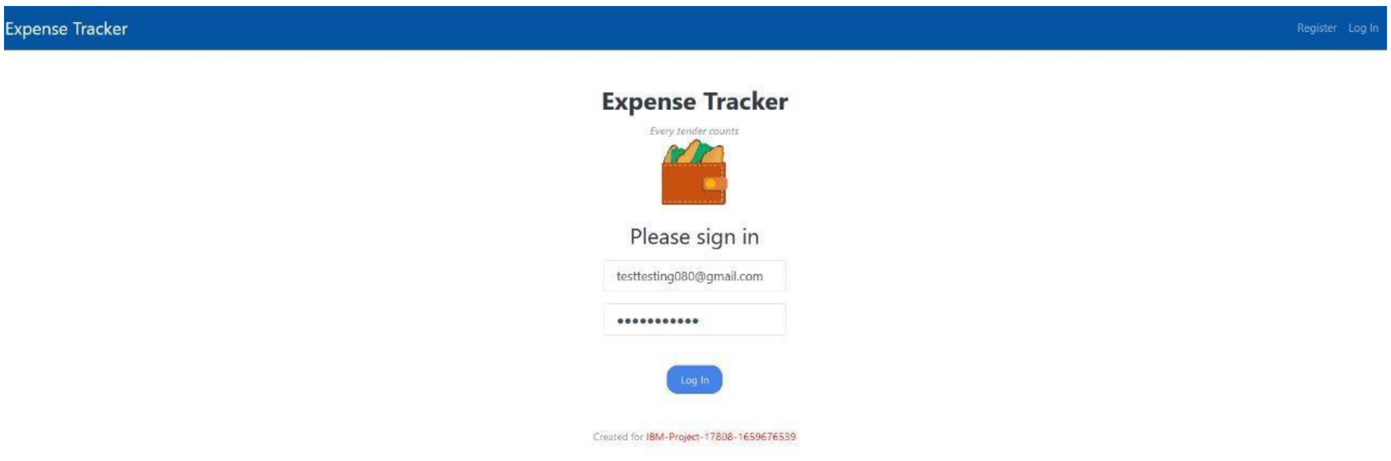
Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	2	8	15
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	9	2	4	11	20
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	0	1	8
Totals	22	14	11	22	51

3. Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Interface	7	0	0	7
Login	43	0	0	43
Logout	2	0	0	2
Limit	3	0	0	3

9.2 Login Page:


Expense Tracker

Register Log In

Expense Tracker

Every tender counts

Please sign in

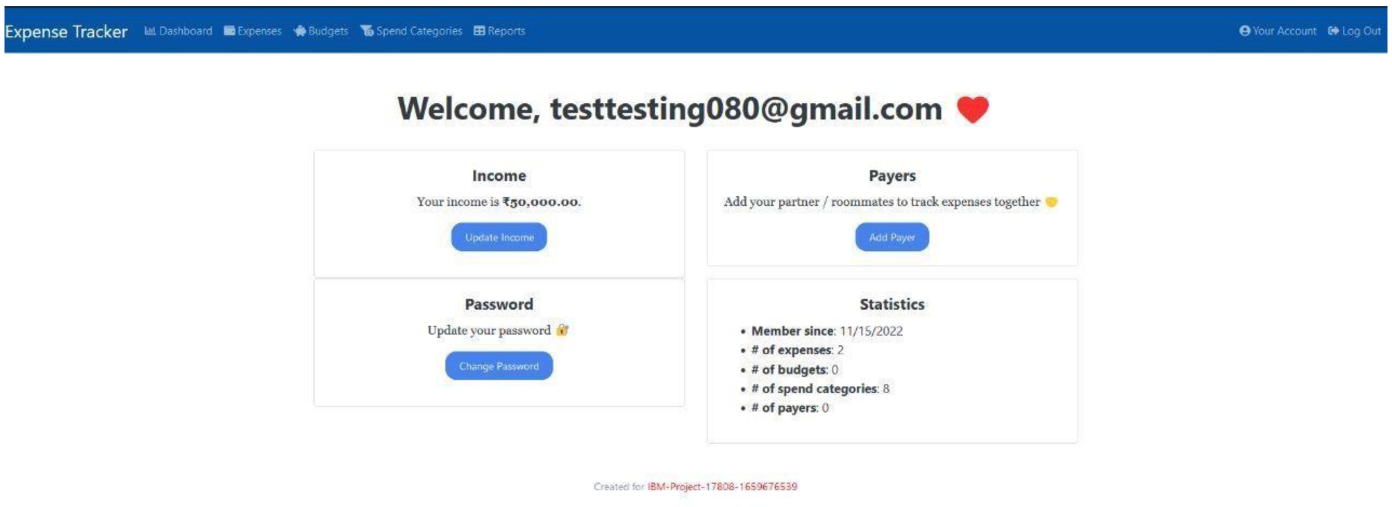
testtesting080@gmail.com

•••••••••••

Log In

Created for IBM-Project-17808-1659676539

Break down of Expense Page:



Welcome, testtesting080@gmail.com ❤️

Income

Your income is ₹50,000.00.

Payers

Add your partner / roommates to track expenses together

Password

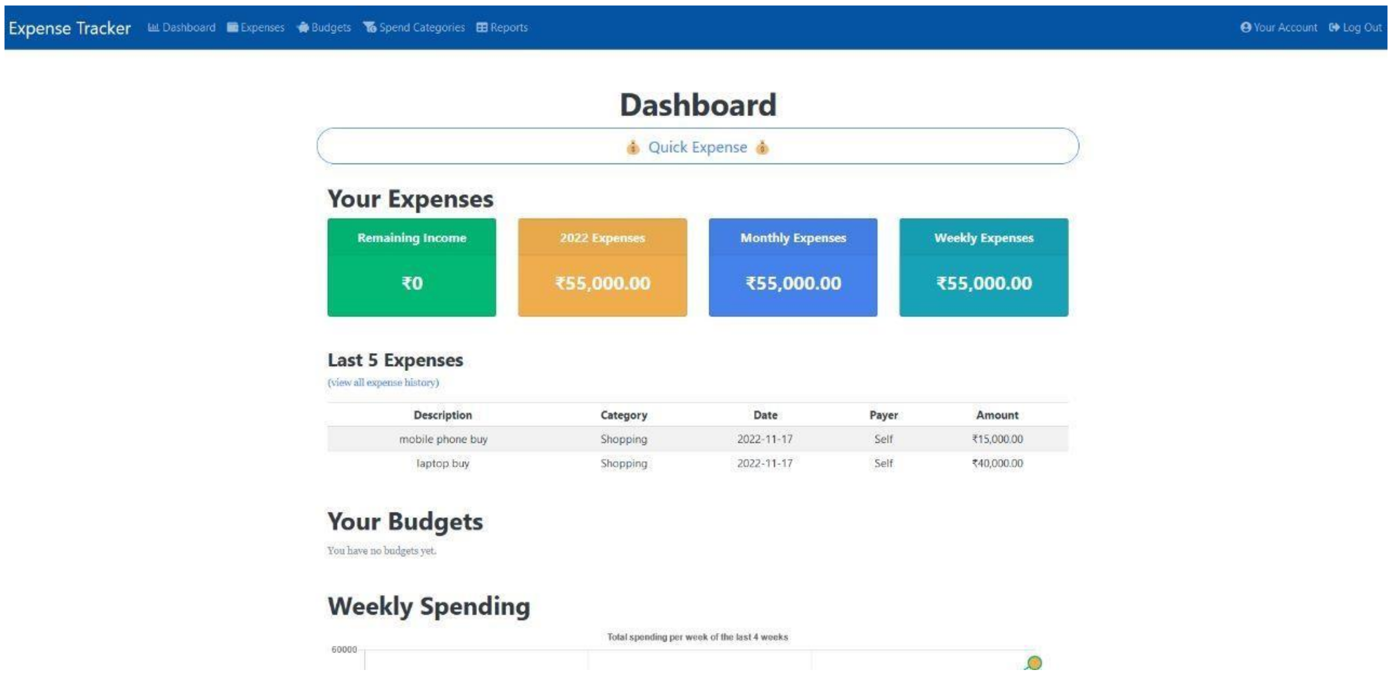
Update your password

Statistics

- **Member since**: 11/15/2022
- **# of expenses**: 2
- **# of budgets**: 0
- **# of spend categories**: 8
- **# of payers**: 0

Created for IBM-Project-17808-1659676539

10. ADVANTAGES AND DISADVANTAGES



Dashboard

Your Expenses

Remaining Income	2022 Expenses	Monthly Expenses	Weekly Expenses
₹0	₹55,000.00	₹55,000.00	₹55,000.00

Last 5 Expenses

(view all expense history)

Description	Category	Date	Payer	Amount
mobile phone buy	Shopping	2022-11-17	Self	₹15,000.00
laptop buy	Shopping	2022-11-17	Self	₹40,000.00

Your Budgets

You have no budgets yet.

Weekly Spending

ADVANTAGES:

One of the major pros of tracking spending is always being aware of the state of one's personal finances. Tracking what you spend can help you stick to your budget, not just in a general way, but in each category such as housing, food, transportation and gifts. While a con is that manually tracking all cash that is spent can be irritating as well as time consuming, a pro is that doing this automatically can be quick and simple. Another pro is that many automatic spending tracking software programs are available for free. Having the program on a hand-held device can be a main pro since it can be checked before spending occurs in order to be sure of the available budget.

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

A comprehensive money management strategy requires clarity and conviction for decision- making. You will need a defined goal and a clear vision for grasping the business and personal finances. That’s when an expense tracking app comes into the picture. An expense tracking app is an exclusive suite of services for people who seek to handle their earnings and plan their expenses and savings efficiently. It helps you track all transactions like bills, refunds, payrolls, receipts, taxes, etc., on a daily, weekly, and monthly basis.

12. FUTURE SCOPE

- Achieve your business goals with a tailored mobile app that perfectly fits your business.
- Scale-up at the pace your business is growing.
- Deliver an outstanding customer experience through additional control over the app.
- Control the security of your business and customer data.
- Open direct marketing channels with no extra costs with methods such as push notifications.
- Boost the productivity of all the processes within the organization.
- Increase efficiency and customer satisfaction with an app aligned to their needs.
- Seamlessly integrate with existing infrastructure.
- Ability to provide valuable insights.
- Optimize sales processes to generate more revenue through enhanced data collection.
- Chats: Equip your expense tracking app with a bot that can understand and answer all user queries and address their needs such as account balance, credit score, etc.
- Prediction: With the help of AI, your mobile app can predict your next purchase, according to your spending behavior. Moreover, it can recommend products and provide unique insights on saving money. It brings out the factors causing fluctuations in your expenses.