

PERSONAL EXPENSE TRACKER APPLICATION

A PROJECT REPORT

Submitted by

**AARTHI.G (950019106702)
SNEHA.R (950019106701)
VARSHA.R(950019106046)
NILAVARASI.A(950019106029)**

TEAM ID : PNT2022TMID49649

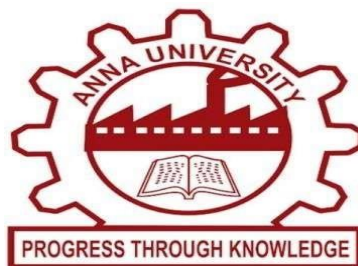
In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

ENGINEERING



S.NO	TABLE OF CONTENT
1	INTRODUCTION Project Overview Purpose
2	LITERATURE SURVEY Existing problem Reference Problem Statement Definition
3	IDEATION & PROPOSED SOLUTION Empathy Map Canvas Ideation & Brainstorming Proposed Solution Problem Solution Fit
4	REQUIREMENT ANALYSIS Functional requirements Non-Functional requirements
5	PROJECT DESIGN Data Flow Diagrams Solution & Technical Architecture User Stories
6	PROJECT PLANNING & SCHEDULING Sprint Planning & Estimation Sprint Delivery Schedule Reports from JIRA
7	CODING & SOLUTIONING (Explain the features added in the project along with code) Feature 1 Feature 2 Database Scheme (if applicable)
8	TESTING Test Case

User Acceptance Testing

9 RESULT

Performance Metrics

10 ADVANTAGES &DISADVANTAGES

11 CONCLUSION

12 FUTURE SCOPE

13 APPENDIX

Source Code

GitHub & Project Demo Link

INTRODUCTION

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. A personal finance application 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.

PROJECT OVERVIEW

Expense Tracker is a web application that facilitates the users to keep track and manage their personal expenses. This application helps the users to keep a digital diary. The monthly, and year-wise comparison of expenditures will be done by the app which will let the user know the area where he is spending the most. With the help of this application user can maintain a digital record of their own expenses and alert them with a email if the expenses exceed.

PURPOSE:

An expense tracker 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.

1.

LITERATURE SURVEY

EXISTING METHOD:

MINT

- The app provides several features to monitor and analyze your personal finances, including personalized insights, customizable budgets and subscription monitoring.
- The standout features of Mint are the customizable budgets and alerts. Additionally, Mint provides alerts for bank fees or upcoming bills due.

Pros:

- Customizable budgets
Free credit score tracking
Customizable alerts for upcoming bills and potential bank fees
Desktop platform available

Cons:

- Connectivity issues with some financial accounts
- Ads within the app can be bothersome

SPENDEE

- Spendee is a mobile application for managing personal and family finances.
- Interactive graphs and infographics clearly show input and output financial flows.

Pros:

- Convenient, one device management of all your financial data, including cryptocurrency
- User-friendly interface with colored infographic charts to visually represent and breakdown your spending habits
Customizable features- set up alerts to notify you when you have gone over budget or monthly reminders to pay your bills
Affordable and customizable for however you choose to budget

Option to connect multiple checking accounts, like a family plan but for your finances so you know who paid for what and how much was spent in total

Cons:

Cannot be synchronized with all bank accounts: It depends if the financial institutions allow Spendee to do so.

REFERENCES:

<http://expense-manager.com/how-expense software/>

<https://www.splitwise.com/terms>

<http://code.google.com/p/socialauthandroid/wiki/Facebook>

<http://code.google.com/p/socialauth-android>

https://ijirt.org/master/publishedpaper/IJIRT150860_PAPER.pdf

<http://www.appbrain.com/app/expensemanager/ com.expensemanager>

<http://dspace.daffodilvarsity.edu.bd:8080/handle/123456789/4026>

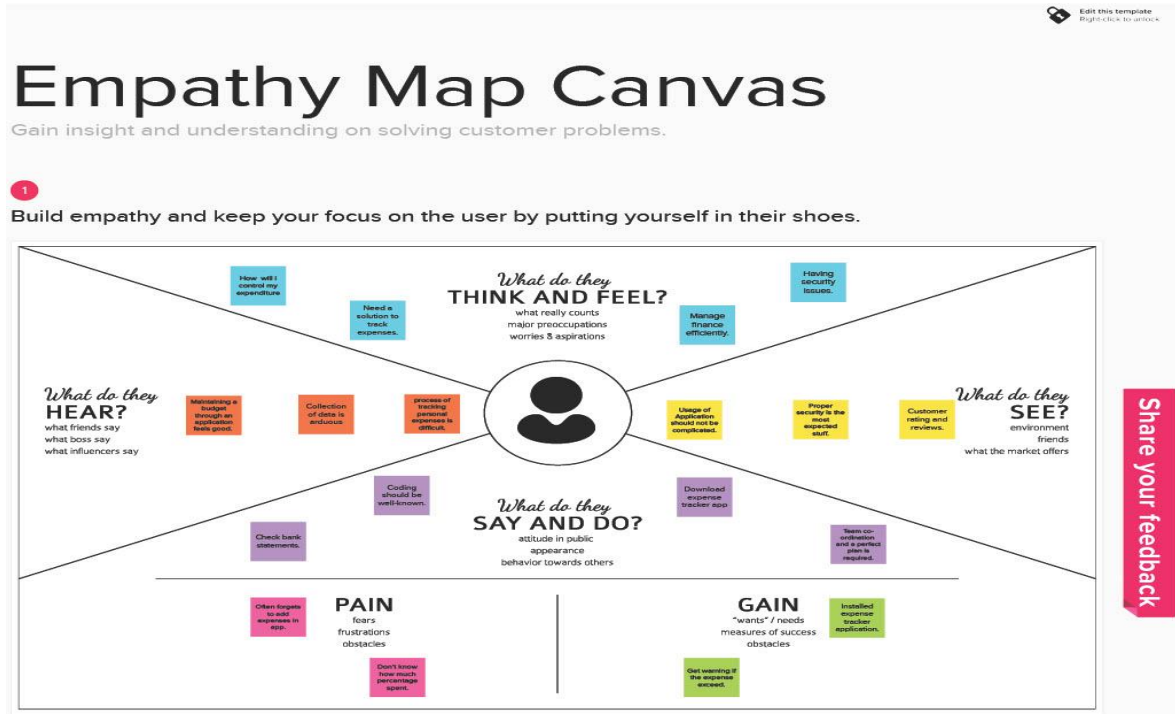
<http://expense-manager.com/how-expense software/>

PROBLEM STATEMENT DEFINITION

At the instant , there is no as such complete solution present easily or we should say free of cost which enables a person to keep a track of its daily expenditure easily. To do so a person has to keep a log in a diary or in a computer, also all the calculations needs to be done by the user which may sometimes results in errors leading to losses. Due to lack of a complete tracking system, there is a constant overload to rely on the daily entry of the expenditure and total estimation till the end of the month.

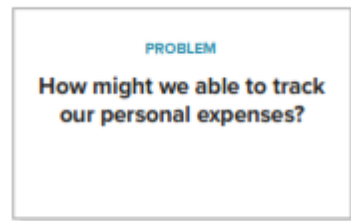
IDEATION & PROPOSED SOLUTION

EMPATHY MAP CANVAS



BRAINSTORMING :

Problem Statements:



Brainstorm:

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!

AARTHI.G

NILAVARASIA

SNEHA.R

VARSHA.R



Group ideas:

Financial
Application.

Simple
track as per
the needs.

Secured
Application.

Offered
strategies to
lower the
burden.

Boost
productivity.

Incredibly
useful
application.

Prioritize:



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



PROPOSED SOLUTION:

Project Design Phase-I Proposed Solution


Date	24 September 2022
Team ID	PNT2022TMID49649
Project Name	Personal Expense Tracker Application

Proposed Solution :

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Expense Tracker Application helps the users to keep a digital diary . With the help of this application user can maintain a digital record of their own expenses and alert them with a email if the expenses exceed.
2.	Idea / Solution description	<ol style="list-style-type: none">1. The ability to manage their money which is coming is a major hurdle for many individuals. Very few people have the discipline and control on their expenses , leaving a gap that this application will fill.2. Expense tracker application uses software to categorize and keep expense all in one place.
3.	Novelty / Uniqueness	Rely on the budgeting app to track expenses and automate all recurrent expenses and remind you on a timely basis.
4.	Social Impact / Customer Satisfaction	Expense tracking app development is one of the best trends today and many people are using them already.
1.	Business Model (Revenue Model)	<ol style="list-style-type: none">1. This application acts as a record book for small business.2. Subscription to enable better features by the users..
2.	Scalability of the Solution	<ol style="list-style-type: none">1. Expense tracking application ensures security and secures your personal data.2. Usage of this application is more in number.

PROPOSED SOLUTION FIT:

Problem-Solution fit canvas 2.0		Purpose / Vision	
Define CS, fit into CC	1. CUSTOMER SEGMENT(S) CS Who is your customer? i.e. working parents of 0-5 y.o. kids People who feel difficult to track their earnings and plan their expenses and savings efficiently.	6. CUSTOMER CONSTRAINTS CC What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices. 1. Availability of categories is limited. 2. Unable to spend more than the fixed limit.	5. AVAILABLE SOLUTIONS AS Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking. Manual calculations being an alternative to online applications but may have some errors.
	2. JOBS-TO-BE-DONE / PROBLEMS J&P Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one, explore different sides. Expenses spent should be updated often which being the major issue.	9. PROBLEM ROOT CAUSE RC What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations. Spending money on unwanted stuffs.	7. BEHAVIOUR BE What does your customer do to address the problem and get the job done? i.e. directly related: find the right solar panel installer; calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace) People start avoiding unwanted expenses and focuses on spending money efficiently.
Focus on J&P, tap into BE, understand RC	3. TRIGGERS TR What triggers customers to act? i.e. seeing their neighbour installing solar panels; reading about a more efficient solution in the news. Seeing others tracking expenses on a daily basis while it's a major hurdle.	10. YOUR SOLUTION SL If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behaviour. 1. People can track their expenses in the form of graph. 2. People will be notified with an email if the limit set is reached	8. CHANNELS OF BEHAVIOUR CH 8.1 ONLINE What kind of actions do customers take online? Extract online channels from #7 Tracking expense through online helps to avoid unwanted expenses. 8.2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development. Physical mode of calculation is tedious process and time consuming.
Identify strong TR & EM	4. EMOTIONS: BEFORE / AFTER EM How do customers feel when they face a problem or a job and afterwards? i.e. lost, insecure > confident, in control - use it in your communication strategy & design. Unable to save the expenses on daily basis/clear on spending money on valuable things.		


 Problem-Solution fit canvas is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 license
 Created by Daria Nepriakhina / Amaltama.com


AMALTAMA
 Problem-Solution Fit Canvas

3. REQUIREMENT ANALYSIS

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 Form Registration through Gmail
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Data	Data gathered in the application server is saved in the high security cloud server.
FR-4	Alert Notification	Alert messages through the Email or SMS.
FR-5	User Monthly Budget Plan	Setting Monthly budget to manage their expenses.
FR-6	Cloud Data Storage	To save the user valuable data high security cloud storage are used (IBM, GOOGLE, etc)

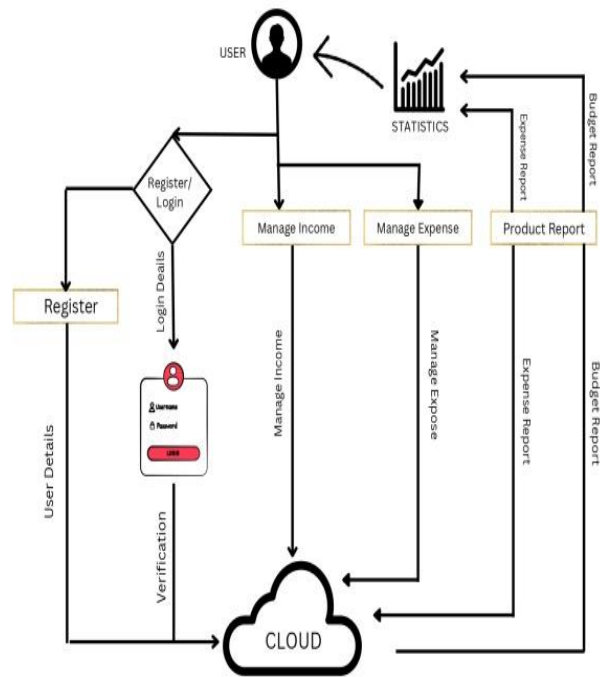
NON-FUNCTIONAL REQUIREMENTS

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

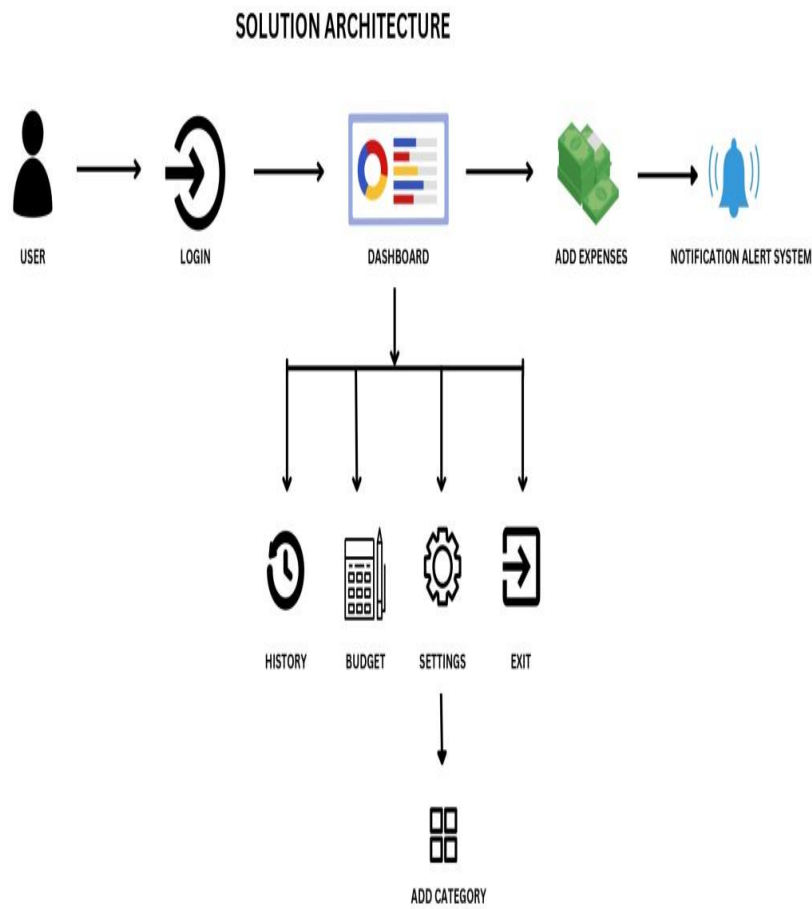
FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Effectiveness, efficiency, and overall experience of the user interacting with our application should be maximized.
NFR-2	Security	Authentication, authorization, encryption of the data must be done in the application.
NFR-3	Reliability	Probability of error in the operations in a specified environment for a specified time should be minimized.
NFR-4	Performance	How the application is functioning accurately and effectively the application is to the end-users.
NFR-5	Availability	Using Cloud Storage and database, application reliability and the user satisfaction will affect the solution
NFR-6	Scalability	Capacity of the application to handle growth, especially in handling more users.

4.PROJECT DESIGN

DATA FLOW DIAGRAMS:



SOLUTION AND TECHNICAL ARCHITECTURE:



USER STORIES:

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
		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-2	As a user, I can register for the application through Form	I can register & access the dashboard with Form Login	Low	Sprint-2
		USN-4	As a user, I can register for the application through Gmail		Medium	Sprint-1
	Login	USN-5	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard		As a user, I can access my detail, manage the expense, add budget, expense report from the app etc.		High	Sprint-1
Customer (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
	Login	USN-2	As a user, I can log into the application by entering email & password		High	Sprint-1
	Dashboard	USN-3	As a user, I can access my detail, manage the expense, add budget, expense report from the application etc..		High	Sprint-1
Customer Care Executive	Email or Customer Care no		As a user, I can contact the service administration for the support.	I can solve the Issue.	High	Sprint-3
Administrator	Email or Customer Care no		As a user, I can contact the service administration for the support.	I can solve the Issue.	High	Sprint-1

5.

PROJECT PLANNING & SCHEDULING:

SPRINT PLANNING & ESTIMATION:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint 1	Login	USN-1	As a user, I can log into the application by entering email & password	20	High	Aarthi G Sneha R Nilavarasi A Varsha R
Sprint2	Registration	USN-1	As a user, I can register for the application by entering my email,password, and confirming my password.	18	High	Aarthi G Sneha R Nilavarasi A Varsha R
Sprint3	Dashboard	USN-1	Logging in takes to the dashboard for the logged user	15	High	Aarthi G Sneha R Nilavarasi A Varsha R
Sprint4	Mail Alert	USN-1	Sending mail to the user when the limit exceed	19	High	Aarthi G Sneha R Nilavarasi A Varsha R

MILESTONE & ACTIVITY LIST:

Activity Number	Activity Name	Detailed Activity Description	Task Assigned	Status
1.1	Access Resources	Access the resources(courses)inproject dashboard.	All Members	COMPLETED
1.2	Rocket chat registration	Join the mentoring channel via platform& rocket-chat mobile app.	All Members	COMPLETED
1.3	Access workspace	Access the guided project workspace.	All Members	COMPLETED
1.4	IBM Cloud registration	Register on IBM Academic Initiative &Apply Feature code for IBM Cloud Credits.	All Members	COMPLETED

1.5	Project Repository Creation	Create GitHub account & collaborate with Project Repository in project workspace.	All Members	COMPLETED
1.6	Environment Setup	Set-up the Laptop / Computers based on the pre-requisites for each technology track.	All Members	COMPLETED
2.1	Literature survey	Literature survey on the selected project & Information Gathering.	All Members	COMPLETED
2.2	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED
2.3	Empathy Map	Prepare Empathy Map Canvas to capture the user Pains & Gains, Prepare list of problem statements	All Members	COMPLETED
2.4	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED
2.5	Brainstorming	List the ideas (at least 4 per each team member) by organizing the brainstorm session and prioritize the ideas	All Members	COMPLETED

2.6	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED
3.1	Proposed Solution Document	Prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.	All Members	COMPLETED
3.2	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED

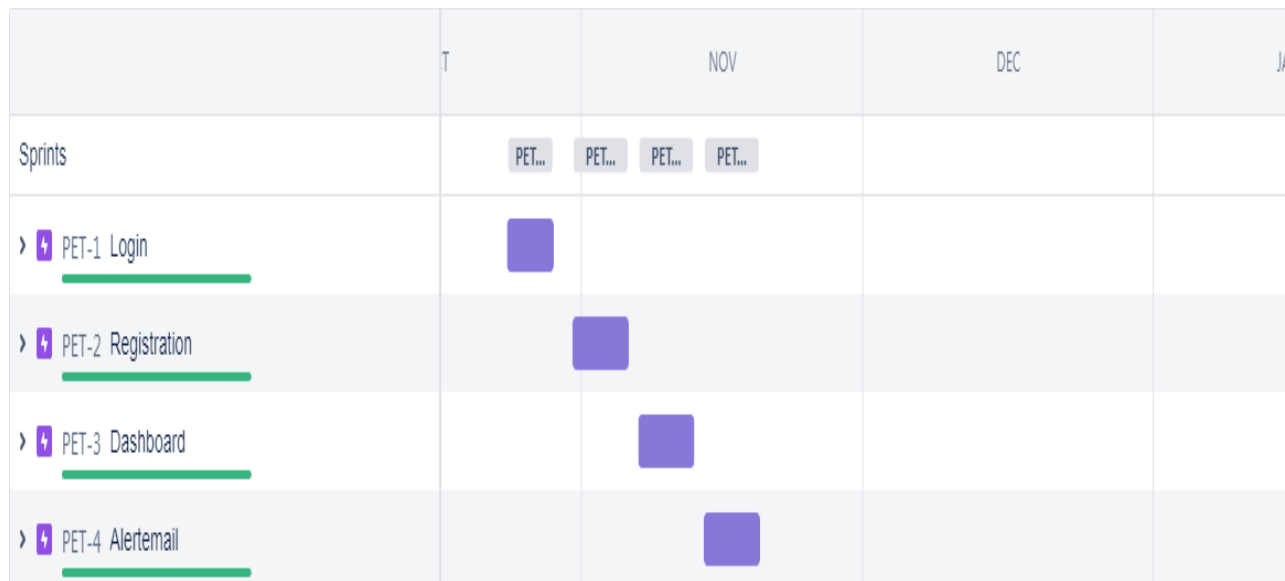
3.3	Problem - Solution fit & SolutionArchitecture	Prepare problem - solution fit document& Solution Architecture.	All Members	COMPLETED
3.4	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED
4.1	Customer Journey Map	Prepare the customer journey maps to understand the user interactions & experiences with the application (entry to exit).	All Members	COMPLETED
4.2	Technology Training	Attend the technology trainings as perthe training Calendar.	All Members	COMPLETED
4.3	Functional Requirements& Data Flow Diagrams	Prepare theFunctionalRequirement Document &DataFlow Diagrams.	All Members	COMPLETED
4.4	Technology Architecture	Prepare Technology Architecture of the solution.	All Members	COMPLETED
4.5	Technology Training	Attend the technology trainings as per the training Calendar.	All Members	COMPLETED
5.1	Milestone&Activity List	Prepare Milestone & Activity List.	All Members	COMPLETED
5.2	Sprint Delivery Plan	Prepare Sprint Delivery Plan.	All Members	COMPLETED

5.3	Login	As a user, I can log into the application by entering email & password	All Members	COMPLETED
5.4	Registration	As a user, I can register for the application by entering myemail ,password, and confirming my password.	All Members	COMPLETED
5.5	Dashboard	Logging in takes to the dashboard for the logged user	All Members	COMPLETED
6.1	Mail Alert	Logging in takes to the dashboard for the logged user	All Members	COMPLETED

SPRINT DELIVERY SCHEDULE:

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	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

REPORTS FROM JIRA:



6.

CODING & SOLUTIONING

FEATURES:

1. Python Flask

We have used python flask to develop our project . Python flask is a web framework and a python module that lets you develop web applications easily.

```
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db
import re
from random import *
from flask_mail import Mail, Message
app = Flask(__name__)
mail = Mail(app) # instantiate the mail class

# configuration of mail
app.config['MAIL_SERVER']='smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'nilavarasi2002@gmail.com'
app.config['MAIL_PASSWORD'] = 'ipergvvhzblqfnpd'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
otp = randint(000000,999999)

def test_predict_image_url():
    stub = service_pb2_grpc.V2Stub(ClarifaiChannel.get_grpc_channel())

    req = service_pb2.PostModelOutputsRequest(
        model_id=GENERAL_MODEL_ID,
        inputs=[
            resources_pb2.Input(
                data=resources_pb2.Data(image=resources_pb2.Image(url=DOG_IMAGE_URL))
            )
        ],
    )

    response = post_model_outputs_and_maybe_allow_retries(stub, req, metadata=metadata())
    print(response)
    raise_on_failure(response)

    assert len(response.outputs[0].data.concepts) > 0
app.secret_key = 'a'
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=ba99a9e6-d59e-4883-8fc0-d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=31321;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UID=pvw86690;PWD=zJNTZLGkY3yuoy3X","")
```

```

@app.route("/")
@app.route('/home')
def home():
    return render_template('index.html')
@app.route('/login', methods=['GET', 'POST'])
def login():
    global userid
    msg = ""
    if request.method == 'POST' and 'email' in request.form and 'password1' in request.form:
        email = request.form['email']
        password1 = request.form['password1']
        stmt = ibm_db.prepare(conn, 'SELECT * FROM regi WHERE email = ? AND password1 = ?')
        ibm_db.bind_param(stmt, 1, email)
        ibm_db.bind_param(stmt, 2, password1)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_tuple(stmt)
        if account:
            session['id'] = account[0]
            userid = account[0]
            session['email'] = account[1]
            msg = 'Logged in successfully !'
            return redirect(url_for('button'))
        else:
            msg = 'Incorrect username / password !'
    return render_template('login.html', msg = msg)
@app.route('/button')
def button():
    return render_template('button.html')

@app.route('/limit', methods=["POST", "GET"])
def limit():

    if 'id' in session and 'email' in session:
        uid = session['id']
        exist = ibm_db.prepare(conn, 'SELECT uid, limit FROM limit WHERE uid = ?')
        ibm_db.bind_param(exist, 1, session['id'])
        ibm_db.execute(exist)
        exist = ibm_db.fetch_tuple(exist)
        if request.method == "POST":
            print("Executing INSERT into LIMIT")
            uid = session['id']
            stmt = ""
            if exist == False:
                print("Creating New")
                stmt = ibm_db.prepare(conn, 'INSERT INTO limit (limit, uid) VALUES (?, ?)')

            else:
                print("Updating")
                stmt = ibm_db.prepare(conn, 'UPDATE limit SET \
limit = ? \
WHERE uid = ?')

```

```

        ibm_db.bind_param(stmt, 1, request.form['limit'])
        ibm_db.bind_param(stmt, 2, uid)
        ibm_db.execute(stmt)

        return render_template('limit.html', status="Success", limit=int(request.form['limit']))
    else:
        if exist == False:
            return render_template('limit.html', limit=limit)
        else:
            return render_template('limit.html', limit=int(exist[1]))

    return 'Not Authed'

@app.route('/stat')
def stat():
    if 'id' in session:
        stmt = ibm_db.prepare(conn, 'SELECT expense, expensedate FROM expense WHERE uid = ?')
        ibm_db.bind_param(stmt,1,session['id'])
        ibm_db.execute(stmt)
        tb = ibm_db.fetch_tuple(stmt)

        months = { }
        while tb != False:
            sliced = tb[1].strftime("%b")
            if sliced in months:
                months[sliced] += tb[0]
            else:
                months[sliced] = tb[0]
            tb = ibm_db.fetch_tuple(stmt)

        return render_template('stat.html', data=months)
    return 'Not Authed Please Login'

@app.route('/display')
def display():
    if 'id' in session :
        stmt = ibm_db.prepare(conn, 'SELECT amount, income, expense, expensedate, category FROM
expense WHERE uid = ?')
        ibm_db.bind_param(stmt,1,session['id'])
        ibm_db.execute(stmt)

        tb = ibm_db.fetch_assoc(stmt)
        data = []
        while tb != False:
            data.append(tb)
            tb = ibm_db.fetch_assoc(stmt)
        print(data)
        return render_template('display.html',data=data)

    return 'Not Authed'

```

```

@app.route('/wallet')
def wallet():
    return render_template('design.html')
@app.route('/income', methods=['GET', 'POST'])
def income():
    if 'id' in session and request.method == 'POST' and ('amount' in request.form) and ('income' in
request.form) and ('expense' in request.form) and ('expensedate' in request.form )and ('category' in
request.form):
        msg = "
        uid = session['id']
        amount = request.form['amount']
        income = request.form['income']
        expense = request.form['expense']
        expensedate = request.form['expensedate']
        category = request.form['category']

        limit = ibm_db.prepare(conn, "SELECT limit from limit WHERE uid = ?")
        ibm_db.bind_param(limit, 1, session['id'])
        ibm_db.execute(limit)

        data = ibm_db.fetch_tuple(limit)
        sum = 0
        if data != False:
            data = data[0]

            all_expenses = ibm_db.prepare(conn, "SELECT expense from expense WHERE uid = ?")
            ibm_db.bind_param(all_expenses, 1, session['id'])
            ibm_db.execute(all_expenses)
            expense_data = ibm_db.fetch_tuple(all_expenses)

            while expense_data != False:
                sum += int(expense_data[0])
                expense_data = ibm_db.fetch_tuple(all_expenses)

            sum += int(expense)
            email = (session['email'] + '@gmail.com').lower()
            if sum >= data:
                #Limit Exceeded
                msg = Message('Limit Exceeded', sender='nilavarasi2002@gmail.com', recipients=[email])
                msg.body = "You Have Exceeded Your Expense Limit"
                mail.send(msg)
                msg = "Exceeded Limit"
                return render_template('add.html', a = msg)

            prep_stmt = ibm_db.prepare(conn, "INSERT INTO
expense(uid,amount,income,expense,expensedate,category) VALUES (?, ?, ?, ?, ?, ?)")
            ibm_db.bind_param(prepare_stmt, 1, uid)
            ibm_db.bind_param(prepare_stmt, 2, amount)
            ibm_db.bind_param(prepare_stmt, 3, income)
            ibm_db.bind_param(prepare_stmt, 4, expense)
            ibm_db.bind_param(prepare_stmt, 5, expensedate)
            ibm_db.bind_param(prepare_stmt, 6, category)

```



```

        ibm_db.execute(prepare_stmt)
        msg = 'You have successfully added !'
        return render_template('add.html', a = msg)
    return render_template('add.html')

```

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

```
def register():
```

```
    msg = "
```

```
    if request.method == 'POST':
```

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

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

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

```
        password1 = request.form['password1']
```

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

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

```
        phonenumber = request.form['onenumber']
```

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

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

```
        sql = "SELECT * FROM regi WHERE email = ? "
```

```
        stmt = ibm_db.prepare(conn,sql)
```

```
        ibm_db.bind_param(stmt,1,email)
```

```
        ibm_db.execute(stmt)
```

```
        account = ibm_db.fetch_assoc(stmt)
```

```
        print(account)
```

```
        if account:
```

```
            msg = 'Account already exists !'
```

```
        elif not re.match(r'^@+@[^@]+\.[^@]+', email):
```

```
            msg = 'Invalid email address !'
```

```
        elif not re.match(r'[A-Za-z0-9]+', firstname):
```

```
            msg = 'firstname must contain only characters and numbers !'
```

```
        elif not email or not firstname or not password1:
```

```
            msg = 'Please fill out the form !'
```

```
        else:
```

```
            insert_sql = "INSERT INTO
```

```
regi(firstname,lastname,email,password1,cpassword,birthdate,onenumber,job,income) VALUES (?,
?, ?, ?, ?, ?, ?, ?)"
```

```
            stmt = ibm_db.prepare(conn,insert_sql)
```

```
            ibm_db.bind_param(stmt, 1, firstname)
```

```
            ibm_db.bind_param(stmt, 2, lastname)
```

```
            ibm_db.bind_param(stmt, 3, email)
```

```
            ibm_db.bind_param(stmt, 4, password1)
```

```
            ibm_db.bind_param(stmt, 5, cpassword)
```

```
            ibm_db.bind_param(stmt, 6, birthdate)
```

```
            ibm_db.bind_param(stmt, 7, ononenumber)
```

```
            ibm_db.bind_param(stmt, 8, job)
```

```
            ibm_db.bind_param(stmt, 9, income)
```

```
            ibm_db.execute(stmt)
```

```
            msg = 'You have successfully registered !'
```

```
            return render_template('email.html')
```

```
    elif request.method == 'POST':
```

```
        msg = 'Please fill out the form !'
```

```

    return render_template('register.html', msg = msg)
@app.route('/userprofile', methods=['GET', 'POST'])
def userprofile():
    if 'id' in session:
        stmt = ibm_db.prepare(conn, 'SELECT firstname,email,income FROM regi WHERE id = ?')
        ibm_db.bind_param(stmt,1,session['id'])
        ibm_db.execute(stmt)
        tb = ibm_db.fetch_assoc(stmt)
        data = []
        while tb != False:
            data.append(tb)
            tb = ibm_db.fetch_assoc(stmt)
        print(data)
        return render_template('userprofile.html',data=data)
    return render_template('userprofile.html')
@app.route('/verify', methods=['GET', 'POST'])
def verify():
    if request.method == 'POST':
        email1 = request.form['email1']
        sql = "SELECT * FROM validate WHERE email1 = ? "
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,email1)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Account already exists !'
        else:
            insert_sql = "INSERT INTO validate VALUES (?)"
            stmt = ibm_db.prepare(conn,insert_sql)
            ibm_db.bind_param(stmt, 1, email1)
            ibm_db.execute(stmt)
            msg = Message('Hello',sender ='nilavarasi2002@gmail.com',recipients = [email1])
            msg.body = str(otp)
            mail.send(msg)
            return render_template('verify.html')
    return render_template('verify.html')
@app.route('/validate',methods=['GET', 'POST'])
def validate():
    user_otp = request.form['otp']
    if otp == int(user_otp):
        return render_template('index.html')
    return render_template('index.html')

@app.route('/logout', methods=['GET'])
def logout():
    if 'email' in session:
        del session['email']
        del session['id']
        return redirect('/')
    return redirect('/')

```

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

2.IBM Cloud db2

When the new user registers into the application and the details of the details of the user gets stored in IBM Cloud DB2. We have connected the DB2 with our project using the below code

```
app.config['MAIL_SERVER']='smtp.gmail.com'  
app.config['MAIL_PORT'] = 465  
app.config['MAIL_USERNAME'] = 'nilavarasi2002@gmail.com'  
app.config['MAIL_PASSWORD'] = 'ipergvvhzblqfnpd'  
app.config['MAIL_USE_TLS'] = False  
app.config['MAIL_USE_SSL'] = True  
mail = Mail(app)  
otp = randint(000000,999999)  
  
def test_predict_image_url():  
    stub = service_pb2_grpc.V2Stub  
  
(ClarifaiChannel.get_grpc_channel())  
  
req = service_pb2.PostModelOutputsRequest(  
    model_id=GENERAL_MODEL_ID,  
    inputs=[  
        resources_pb2.Input(  
            data=resources_pb2.Data  
  
(image=resources_pb2.Image(url=DOG_IMAGE_URL))  
        )  
    ],  
)
```

```

response = post_model_outputs_and_maybe_allow_retries

(stub, req, metadata=metadata())

print(response)

raise_on_failure(response)

assert len(response.outputs[0].data.concepts) > 0

app.secret_key = 'a'

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=ba99a9e6-

d59e-4883-8fc0-

d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.clou

d;PORT=31321;SECURITY=SSL;SSLServerCertificate=DigiCertGlo

balRootCA.crt;UID=pvw86690;PWD=zJNTZLGkY3yuoy3X",",")

```

3.Python Flask

When the new user registers ,the confirmation mail is sent to the user's mail using the python flask and when the user exceeds the limit the alert email is sent. Usin the code

4.Deploying flask app in Docker

We have deployed our flask app in Docker where they packae all the code,libraries And dependencies together to make it possible for multiple containers to run in the same host and we can run our flask app using this Docker Desktop.

5.IBM Cloud Container Registry

We have deployed our app as Docker image at IBM Cloud Registry

6.Kubernetes

These containers are managed by Kubernetes which automates the operational tasks of container

8. TESTING

Test Cases:

User Acceptance Testing:

Purpose of Document:

The purpose of this document is to briefly explain the test coverage and open issues of the [Early detection of forest fire using Deep Learning] project at the time of the release to User Acceptance Testing (UAT).

Defect Analysis:

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

Resolution	Severit 1	Severit 2	Severit 3	Severit 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

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	40	0	0	40
Logout	2	0	0	2
Sinup	3	0	0	3
Limit	9	0	0	9
Final Report Output	4	0	0	4

9. RESULTS

9.1.PERFORMANCE METRICS:

The new system has overcome most of the limitations of the existing system and works according to the design specification given. The project what we have developed is work more efficient than the other income and expense tracker. The project successfully avoids the manual calculation for avoiding calculating the income and expense per month. The modules are developed with efficient and also in an attractive manner. The developed systems dispense the problem and meet the needs of by providing reliable and comprehensive information. All the requirements projected by the user have been met by the system. The newly developed system consumes less processing time and all the details are updated and processed immediately

10.ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- User can have a control over their money and expenses
- Users are alerted with an email when they exceed their limit
- Reports are generated based on the users expenses

DISADVANTAGES:

- Less secured
- Limited Accessibility

11. CONCLUSION

Personal Expense Tracker is an web based application. We created this application so that a user can accurately calculate his daily cost. Using this application, the user will see the amount of his income and how much a user is spending and a notification will be sent to the user's if he/she exceeds the limit and also report is generated..

FUTURE SCOPE

In future more features may be added , Some are listed below

- Multiple currency support
- Include currency converter

Our Github link - [IBM-EPBL/IBM-Project-30033-1660138776](https://github.com/IBM-EPBL/IBM-Project-30033-1660138776)

DEMO VIDEO:

Demo video link - <https://drive.google.com/file/d/1F9whyZmX3EqVpV4eufEnAe0DhV-ZtKzL/view?usp=drivesdk>

SAMPLE CODE:

app.py

```
from flask import Flask, render_template, request,
redirect, url_for, session
import ibm_db
import re
from random import *
from flask_mail import Mail, Message
app = Flask(__name__)
mail = Mail(app) # instantiate the mail class

# configuration of mail
app.config['MAIL_SERVER']='smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'nilavarasi2002@gmail.com'
app.config['MAIL_PASSWORD'] = 'ipergvvhzblqfnpd'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)
otp = randint(000000,999999)

def test_predict_image_url():
    stub = service_pb2_grpc.V2Stub

(ClarifaiChannel.get_grpc_channel())

req = service_pb2.PostModelOutputsRequest(
    model_id=GENERAL_MODEL_ID,
    inputs=[
        resources_pb2.Input(
            data=resources_pb2.Data
```

```

(image=resources_pb2.Image(url=DOG_IMAGE_URL))
    )
    ],
)

response = post_model_outputs_and_maybe_allow_retries

(stub, req, metadata=metadata())
    print(response)
    raise_on_failure(response)

    assert len(response.outputs[0].data.concepts) > 0
app.secret_key = 'a'
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=ba99a9e6-
d59e-4883-8fc0-
d6a8c9f7a08f.c1ogj3sd0tgtu0lqde00.databases.appdomain.clou
d;PORT=31321;SECURITY=SSL;SSLServerCertificate=DigiCertGlo
balRootCA.crt;UID=pvw86690;PWD=zJNTZLGkY3yuoy3X",",")
@app.route("/")
@app.route('/home')
def home():
    return render_template('index.html')
@app.route('/login', methods =['GET', 'POST'])
def login():
    global userid
    msg = ""
    if request.method == 'POST' and 'email' in request.form
and 'password1' in request.form:
        email = request.form['email']
        password1 = request.form['password1']
        stmt = ibm_db.prepare(conn,'SELECT * FROM regi
WHERE email = ? AND password1 = ?')
        ibm_db.bind_param(stmt,1,email)
        ibm_db.bind_param(stmt,2,password1)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_tuple(stmt)
        if account:

```

```

        session['id'] = account[0]
        userid = account[0]
        session['email'] = account[1]
        msg = 'Logged in successfully !'
        return redirect(url_for('button'))
    else:
        msg = 'Incorrect username / password !'
        return render_template('login.html', msg = msg)
@app.route('/button')
def button():
    return render_template('button.html')

@app.route('/limit', methods=["POST", "GET"])
def limit():

    if 'id' in session and 'email' in session:
        uid = session['id']
        exist = ibm_db.prepare(conn, 'SELECT uid, limit

FROM limit WHERE uid = ?')
        ibm_db.bind_param(exist,1, session['id'])
        ibm_db.execute(exist)
        exist = ibm_db.fetch_tuple(exist)
        if request.method == "POST":
            print("Executing INSERT into LIMIT")
            uid = session['id']
            stmt = ""
            if exist == False:
                print("Creating New")
                stmt = ibm_db.prepare(conn, 'INSERT INTO

limit (limit, uid) VALUES (?, ?)')
            else:
                print("Updating")
                stmt = ibm_db.prepare(conn, 'UPDATE limit

SET \

        limit = ? \
        WHERE uid = ?)

        ibm_db.bind_param(stmt, 1, request.form

['limit'])

```

```

        ibm_db.bind_param(stmt, 2, uid)
        ibm_db.execute(stmt)

        return render_template('limit.html',

status="Success", limit=int(request.form['limit']))
    else:
        if exist == False:
            return render_template('limit.html',

limit=limit)
        else:
            return render_template('limit.html',

limit=int(exist[1]))

    return 'Not Authed'

@app.route('/stat')
def stat():
    if 'id' in session:
        stmt = ibm_db.prepare(conn, 'SELECT expense,

expensedate FROM expense WHERE uid = ?')
        ibm_db.bind_param(stmt,1,session['id'])
        ibm_db.execute(stmt)
        tb = ibm_db.fetch_tuple(stmt)

        months = { }
        while tb != False:
            sliced = tb[1].strftime("%b")
            if sliced in months:
                months[sliced] += tb[0]
            else:
                months[sliced] = tb[0]
            tb = ibm_db.fetch_tuple(stmt)

        return render_template('stat.html', data=months)
    return 'Not Authed Please Login'

@app.route('/display')
def display():
    if 'id' in session :
        stmt = ibm_db.prepare(conn, 'SELECT amount,

```

income, expense, expensedate, category FROM expense WHERE

```
uid = ?')
    ibm_db.bind_param(stmt,1,session['id'])
    ibm_db.execute(stmt)

    tb = ibm_db.fetch_assoc(stmt)
    data = []
    while tb != False:
        data.append(tb)
        tb = ibm_db.fetch_assoc(stmt)
    print(data)
    return render_template('display.html',data=data)

return 'Not Authed'
```

```
@app.route('/wallet')
def wallet():
    return render_template('design.html')
@app.route('/income', methods =['GET', 'POST'])
def income():
    if 'id' in session and request.method == 'POST' and

('amount' in request.form) and ('income' in request.form)

and ('expense' in request.form) and ('expensedate' in

request.form )and ('category' in request.form):
    msg = "
    uid = session['id']
    amount = request.form['amount']
    income = request.form['income']
    expense = request.form['expense']
    expensedate = request.form['expensedate']
    category = request.form['category']

    limit = ibm_db.prepare(conn, "SELECT limit

from limit WHERE uid = ?")
    ibm_db.bind_param(limit, 1, session['id'])
    ibm_db.execute(limit)

    data = ibm_db.fetch_tuple(limit)
    sum = 0
```

```

    if data != False:
        data = data[0]

    all_expenses = ibm_db.prepare(conn,

"SELECT expense from expense WHERE uid = ?")
    ibm_db.bind_param(all_expenses, 1,

session['id'])
    ibm_db.execute(all_expenses)
    expense_data = ibm_db.fetch_tuple

(all_expenses)

    while expense_data != False:
        sum += int(expense_data[0])
        expense_data = ibm_db.fetch_tuple

(all_expenses)

    sum += int(expense)
    email = (session['email'] +

'@gmail.com').lower()
    if sum >= data:
        #Limit Exceeded
        msg = Message('Limit Exceeded',

sender='nilavarasi2002@gmail.com', recipients=[email])
        msg.body = "You Have Exceeded Your Expense

Limit"
        mail.send(msg)
        msg = "Exceeded Limit"
        return render_template('add.html', a =

msg)

    prep_stmt = ibm_db.prepare(conn, "INSERT INTO

expense(uid,amount,income,expense,expensedate,category)

VALUES (?, ?, ?, ?, ?, ?)")
    ibm_db.bind_param(prepare_stmt, 1, uid)

```

```

        ibm_db.bind_param(prepare_stmt, 2, amount)
        ibm_db.bind_param(prepare_stmt, 3, income)
        ibm_db.bind_param(prepare_stmt, 4, expense)
        ibm_db.bind_param(prepare_stmt, 5, expensedate)
        ibm_db.bind_param(prepare_stmt, 6, category)
        ibm_db.execute(prepare_stmt)
        msg = 'You have successfully added !'
        return render_template('add.html', a = msg)
    return render_template('add.html')

```

```

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

```

```

def register():

```

```

    msg = "

```

```

    if request.method == 'POST':

```

```

        firstname = request.form['firstname']

```

```

        lastname = request.form['lastname']

```

```

        email = request.form['email']

```

```

        password1 = request.form['password1']

```

```

        cpassword = request.form['cpassword']

```

```

        birthdate = request.form['birthdate']

```

```

        phonenumber = request.form['phonenumber']

```

```

        job = request.form['job']

```

```

        income = request.form['income']

```

```

        sql = "SELECT * FROM regi WHERE email = ? "

```

```

        stmt = ibm_db.prepare(conn,sql)

```

```

        ibm_db.bind_param(stmt,1,email)

```

```

        ibm_db.execute(stmt)

```

```

        account = ibm_db.fetch_assoc(stmt)

```

```

        print(account)

```

```

        if account:

```

```

            msg = 'Account already exists !'

```

```

        elif not re.match(r'^[a-zA-Z0-9]+@[a-zA-Z0-9]+\.[a-zA-Z0-9]+$', email):

```

```

            msg = 'Invalid email address !'

```

```

        elif not re.match(r'[A-Za-z0-9]+$', firstname):

```

```

            msg = 'firstname must contain only characters

```

```

and numbers !'

```

```

        elif not email or not firstname or not password1:

```

```

            msg = 'Please fill out the form !'

```

```

        else:

```

```

            insert_sql = "INSERT INTO regi

```

```

(firstname,lastname,email,password1,cpassword,birthdate,ph

```

```

onenumber,job,income) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
    stmt = ibm_db.prepare(conn,insert_sql)
    ibm_db.bind_param(stmt, 1, firstname)
    ibm_db.bind_param(stmt, 2, lastname)
    ibm_db.bind_param(stmt, 3, email)
    ibm_db.bind_param(stmt, 4, password1)
    ibm_db.bind_param(stmt, 5, cpassword)
    ibm_db.bind_param(stmt, 6, birthdate)
    ibm_db.bind_param(stmt, 7, phonenumber)
    ibm_db.bind_param(stmt, 8, job)
    ibm_db.bind_param(stmt, 9, income)
    ibm_db.execute(stmt)
    msg = 'You have successfully registered !'
    return render_template('email.html')
elif request.method == 'POST':
    msg = 'Please fill out the form !'

    return render_template('register.html', msg = msg)
@app.route('/userprofile', methods=['GET', 'POST'])
def userprofile():
    if 'id' in session:
        stmt = ibm_db.prepare(conn, 'SELECT
        firstname,email,income FROM regi WHERE id = ?')
        ibm_db.bind_param(stmt,1,session['id'])
        ibm_db.execute(stmt)
        tb = ibm_db.fetch_assoc(stmt)
        data = []
        while tb != False:
            data.append(tb)
            tb = ibm_db.fetch_assoc(stmt)
        print(data)
        return render_template
        ('userprofile.html',data=data)
    return render_template('userprofile.html')
@app.route('/verify', methods=['GET', 'POST'])
def verify():
    if request.method == 'POST':
        email1 = request.form['email1']
        sql = "SELECT * FROM validate WHERE email1 = ? "
        stmt = ibm_db.prepare(conn,sql)
        ibm_db.bind_param(stmt,1,email1)
        ibm_db.execute(stmt)

```



```

account = ibm_db.fetch_assoc(stmt)
print(account)
if account:
    msg = 'Account already exists !'
else:
    insert_sql = "INSERT INTO validate VALUES (?)"
    stmt = ibm_db.prepare(conn,insert_sql)
    ibm_db.bind_param(stmt, 1, email1)
    ibm_db.execute(stmt)
    msg = Message('Hello',sender

='nilavarasi2002@gmail.com',recipients = [email1])
    msg.body = str(otp)
    mail.send(msg)
    return render_template('verify.html')
    return render_template('verify.html')
@app.route('/validate',methods=['GET', 'POST'])
def validate():
    user_otp = request.form['otp']
    if otp == int(user_otp):
        return render_template('index.html')
    return render_template('index.html')

@app.route('/logout', methods=['GET'])
def logout():
    if 'email' in session:
        del session['email']
        del session['id']
        return redirect('/')
    return redirect('/')

if __name__ == '__main__':
    app.run(debug = True)

```

login.html

```

<!DOCTYPE html>
<html lang="en" >
<head>
<style>
body

```

```

.box-form {
  margin-top: 1%;
  margin-left: auto;
  margin-right: auto;
  width: 30%;
  height: 500px;
  background:white;
  border-radius: 10px;
  overflow: hidden;
  display: flex;
  flex: 1 1 100%;
  align-items: stretch;
  justify-content: space-between;
  box-shadow: 0 0 20px 6px #090b6f85;
}
@media (max-width: 750px) {
  .box-form {
    flex-flow: wrap;
    text-align: center;
    align-content: center;
    align-items: center;
  }
}
.box-form div {
  height: auto;
  top: 20%;
}

.box-form .right {
  padding: 40px;
  overflow: hidden;
  background-image: linear-gradient(135deg, #b6bbb8 40%,
purple 100%);
}
@media (max-width: 980px) {
  .box-form .right {
    width: 300px;
    height: auto;
  }
}
.box-form .right h5 {
  font-size: 3vmax;
  line-height: 0;
}

```

```
}  
.box-form .right p {  
  font-size: 14px;  
  color: black ;  
}  
.box-form .right .inputs {  
  overflow: hidden;  
}  
.box-form .right input {  
  width: 100%;  
  padding: 8px;  
  margin-top: 20px;  
  font-size: 14px;  
  border: none;  
  outline: none;  
  border-bottom: 2px solid green;  
}  
  
.box-form .right button {  
  color: #fff;  
  font-size: 16px;  
  padding: 12px 35px;  
  border-radius: 50px;  
  display: inline-block;  
  border: 0;  
  outline: 0;  
  box-shadow: 0px 4px 20px 0px purple;  
  background-color: black;  
}  
label {  
  display: block;  
  position: relative;  
  margin-left: 30px;  
}  
  
label::before {  
  content: '\f00c';  
  position: absolute;  
  font-family: FontAwesome;  
  background: transparent;  
  border: 3px solid purple;  
  border-radius: 4px;  
  color: transparent;  
  left: -30px;
```

```
    transition: all 0.2s linear;
}
```

```
label:hover::before {
    font-family: FontAwesome;
    content: '\f00c';
    color: purple;
    cursor: pointer;
    background: #70F570;
}
```

```
label:hover::before .text-checkbox {
    background: #70F570;
}
```

```
label span.text-checkbox {
    display: inline-block;
    height: auto;
    position: relative;
    cursor: pointer;
    transition: all 0.2s linear;
}
```

```
label input[type="checkbox"] {
    display: none;
}
```

```
</style>
```

```
<meta charset="UTF-8">
```

```
<title>Login Page</title>
```

```
<link href="https://fonts.googleapis.com/css?
```

```
family=Open+Sans" rel="stylesheet">
```

```
<link href="https://maxcdn.bootstrapcdn.com/font-
```

```
awesome/4.7.0/css/font-awesome.min.css" rel="stylesheet"
```

```
integrity="sha384-wvfXpqpZZVQGK6TAh5PVIGOfQNHSoD2xbE
```

```
+QkPxCAFINEevoEH3Sl0sibVcOQVnN"
```

```
crossorigin="anonymous"><link rel="stylesheet"
```

```
href="/style.css">
```

```

</head>
<body>
<form action="{{ url_for('login') }}" method="post">
<!-- partial:index.partial.html -->
<div class="box-form">
  <div class="left">

    <div class="right">
      <h5>Login</h5>
      <p>Don't have an account? <a href="{{ url_for(
('register') }}">Creat Your Account</a> </p>
      <div class="inputs">
        <input type="text" name="email" placeholder="email">
        <br>
        <input type="password" name="password1"
placeholder="password">
      </div>

      <br>
      <button>Login</button>
    </div>

  </div>
<!-- partial -->
</form>
</body>
</html>

```

Registration.html

```

<html>
<head>
  <link rel="stylesheet" href="{{ url_for('static',
filename='css/reg.css') }}">
  <link
href="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootst

```

```

rap.min.css" rel="stylesheet" id="bootstrap-css">
<script

src="//maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstra

p.min.js"></script>
<script

src="//cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.

min.js"></script>
<!------- Include the above in your HEAD tag ----->
<link

href="//netdna.bootstrapcdn.com/bootstrap/3.2.0/css/bootst

rap.min.css" rel="stylesheet" id="bootstrap-css">
<script

src="//netdna.bootstrapcdn.com/bootstrap/3.2.0/js/bootstra

p.min.js"></script>
<script src="//code.jquery.com/jquery-

2.1.3.min.js"></script>

</head>
<body>
<form action="{ { url_for('register') } }" method="post">
<!------- Include the above in your HEAD tag ----->
<div class="container" >
<form class="form-horizontal" role="form">
<h2>Registration</h2>
<div class="form-group">
<label for="firstname" class="col-sm-3 control-label"

>First Name*</label>
<div class="col-sm-9">
<input type="text" id="firstname" name="firstname"

placeholder="First Name" minlength="5" required

class="form-control" autofocus>

```

```

</div>
</div>
<div class="form-group">
<label for="lastname" class="col-sm-3 control-label">Last
Name</label>
<div class="col-sm-9">
<input type="text" id="lastname" name="lastname"
placeholder="Last Name" class="form-control" autofocus>
</div>
</div>
<div class="form-group">
<label for="email" class="col-sm-3 control-label">Email*
</label>
<div class="col-sm-9">
<input type="email" id="email" name="email"
placeholder="Email" class="form-control" name="email"
required>
</div>
</div>
<div class="form-group">
<label for="password" class="col-sm-3 control-
label">Password*</label>
<div class="col-sm-9">
<input type="password" id="password" name="password1"
minlength="4" required placeholder="Password"
class="form-control" >
</div>
</div>
<div class="form-group">
<label for="cpassword" class="col-sm-3 control-
label">Confirm Password*</label>
<div class="col-sm-9">
<input type="password" id="cpassword" name="cpassword"
placeholder="Password" class="form-control" required>

```

```
</div>
</div>
<div class="form-group">
<label for="birthdate" class="col-sm-3 control-label">Date
of Birth* </label>
<div class="col-sm-9">
<input type="date" id="birthdate" name="birthdate"
class="form-control" required>
</div>
</div>
<div class="form-group">
<label for="phonenummer" class="col-sm-3 control-
label">Phone number* </label>
<div class="col-sm-9">
<input type="phonenummer" id="phonenummer"
name="phonenummer" pattern="[0-9]{10}" required
placeholder="Phone number" class="form-control">
<span class="help-block"> </span>
</div>
</div>
<div class="form-group">
<label for="job" class="col-sm-3 control-label">Job*
</label>
<div class="col-sm-9">
<input type="text" id="job" name="job" placeholder="Please
write your job title" class="form-control" required>
</div>
</div>
<div class="form-group">
<label for="income" class="col-sm-3 control-
label">Income(in Rs)* </label>
<div class="col-sm-9">
<input type="number" id="income" name="income"
placeholder="Please write your income per month"
```



```
class="form-control" min="0" required>
</div>
</div>
<div class="form-group">
<div class="col-sm-9 col-sm-offset-3">
<span class="help-block">*Required fields</span>
</div>
</div>
<button type="submit" class="btn btn-primary btn-block"
```

```
action="{ { url_for('register') } }"
```

```
method="post">Register</button>
</div>
</form> <!-- /form -->
</div> <!-- ./container -->
</form>
</body>

</html>
```