PROJECT REPORT

CUSTOMER CARE REGISTRY

TEAM ID - PNT2022TMID47308

TEAM LEADER : GAVASKAR S
TEAM MEMBER : MAHENDRAN E
TEAM MEMBER : POONGKANNAN P
TEAM MEMBER : PRAKASH S

CHAPTER	TITLE
1	**INTRODUCTION**
	1.1 Project Overview
	1.2 Purpose
2	**LITERATURE SURVEY**
	2.1 Existing problem
	2.2 References
	2.2 Problem Statement Definition
3	**IDEATION & PROPOSED SOLUTION**
	3.1 Empathy Map Canvas
	3.2 Ideation & Brainstorming
	3.3 Proposed Solution
	3.4 Problem Solution fit
4	**REQUIREMENT ANALYSIS**
	4.1 Functional requirement
	4.2 Non-Functional requirements
5	**PROJECT DESIGN**
	5.1 Data Flow Diagrams
	5.2 Solution & Technical Architecture
	5.3 User Stories
6	**PROJECT PLANNING & SCHEDULING**
	6.1 Sprint Planning & Estimation
	6.2 Sprint Delivery Schedule
	6.3 Reports from JIRA
7	**CODING & SOLUTIONING**
	7.1 Feature
	7.2 Feature 2
	7.3 Database Schema
8	**TESTING**
	8.1 Test Cases
	8.2 User Acceptance Testing
9	**RESULTS**
	9.1 Performance Metrics

10 **ADVANTAGES & DISADVANTAGES**

11 **CONCLUSION**

12 **FUTURE SCOPE**

13 **APPENDIX**

13.1 Source Code

INTRODUCTION

1.1 Project Overview

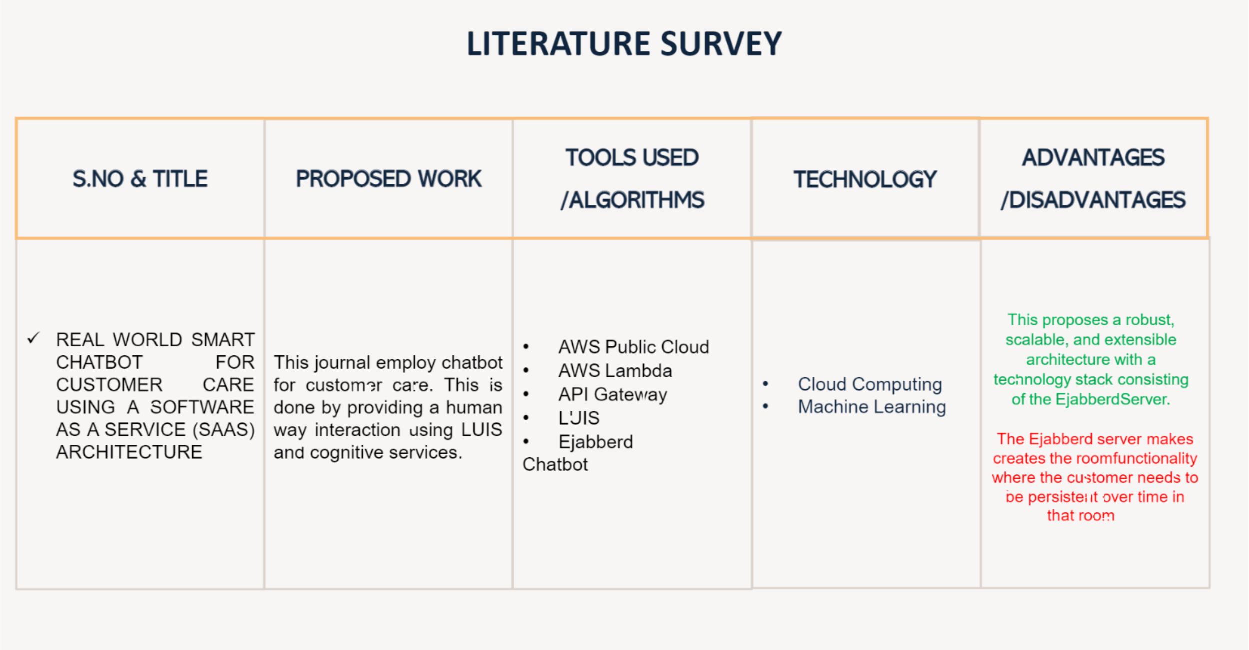
Customer is that the center of attention of each business. The terrible existence of business depends on client satisfaction. Client expects high-quality services, even willing to pay a premium for higher service. From a client perspective, smart service quality ends up in semipermanent client relationships measured by re-patronage and cross sales, additionally client advocates the service to others. Services are essentially completely different from manufacturing; this distinction contributes to the accumulated complexness of service quality. Corporations so build all efforts to produce high-quality services to please customers. However, despite best efforts, associate occasional criticism is inevitable. However, an honest recovery will flip angry, discontent customers into loyal ones, again. The key to success lies in recognizing the importance of responding fairly and effectively to client complaints. Complaints are usually a treasuring hoarded wealth of knowledge, resulting in constructive concepts for rising and upgrading services in the future. Researches show that solely many discontent customers really complain and provide the corporate a chance to correct itself. Others shift their loyalties. Hence, it becomes necessary to resolve complaints truthfully at the earliest, rather than taking a defensive approach. Structured client criticism management is one gospel for downside interference within the long run. This paper decides to develop one such customer care register model.

1.2 Purpose

The Application has been developed to help the customer in processing their complaints. The customers can raise the ticket with a detailed description of the issue. An Agent will be assigned to the Customer to solve the problem. Whenever the agent is assigned to a customer, they will be notified with an email alert. Customers can view the status of the ticket till the service is provided. The main role and responsibility of the admin are to take care of the whole process. Starting from Admin login followed by the agent creation and assigning the customer's complaints. Finally, He will be able to track the work assigned to the agent and a notification will be sent to the customer. Customer can register for an account. After the login, they can create the complaint with description of the problem they are facing. Each user will be assigned with an agent. They can view the status of their complaint.

2. LITERATURE SURVEY

2.1 Existing problem



LITERATURE SURVEY

S.NO & TITLE	PROPOSED WORK	TOOLS USED /ALGORITHMS	TECHNOLOGY	ADVANTAGES /DISADVANTAGES
✓ REAL WORLD SMART CHATBOT FOR CUSTOMER CARE USING A SOFTWARE AS A SERVICE (SAAS) ARCHITECTURE	This journal employ chatbot for customer care. This is done by providing a human way interaction using LUIS and cognitive services.	• AWS Public Cloud • AWS Lambda • API Gateway • LUIS • Ejabberd Chatbot	• Cloud Computing • Machine Learning	This proposes a robust, scalable, and extensible architecture with a technology stack consisting of the EjabberdServer. The Ejabberd server makes creates the roomfunctionality where the customer needs to be persistent over time in that room

2.2 References

[1]. M. Baye, Managerial Economics & Business Strategy McGraw-Hill Education, London, Abacus: The Undercover Economist, vol. 2013, pp. 12-23, 2017.

[2]. J. Obliquity Kay, why our goals are best achieved indirectly, London: Profile Book, pp. 15- 67, 2011.

[3]. P. Keat and P.K. Young, Managerial Economics Global Edition, London: Pearson, pp. 23- 46, 2014.

[4]. Bai changhong and Liu Chi, "study on customer loyalty of service enterprises and its determinants [J]", nankai business review, no. 06, pp. 64-69, 2002.

2.3 Problem Statement Definition

A Customer had a problem when they applied for a ticket they needed to recover a solution or result. So, the customer will contact customer care to raise this issue. After the customer complaint, the company could identify that problem and solve this issue. Now the company wants to avoid these kinds of problems and technical issues. So, the company needs customer satisfaction. Customers can create an account and log in to the dashboard and they can send a ticket along with their name, complaint's body in the webpage and also can see their complaints with the respective time sent by them. Also they can see the status of their tickets. On the other hand, the admin can create and assign agents for each customer's tickets. The email notification will be sent to the customer after their tickets are verified and solved by the agents.

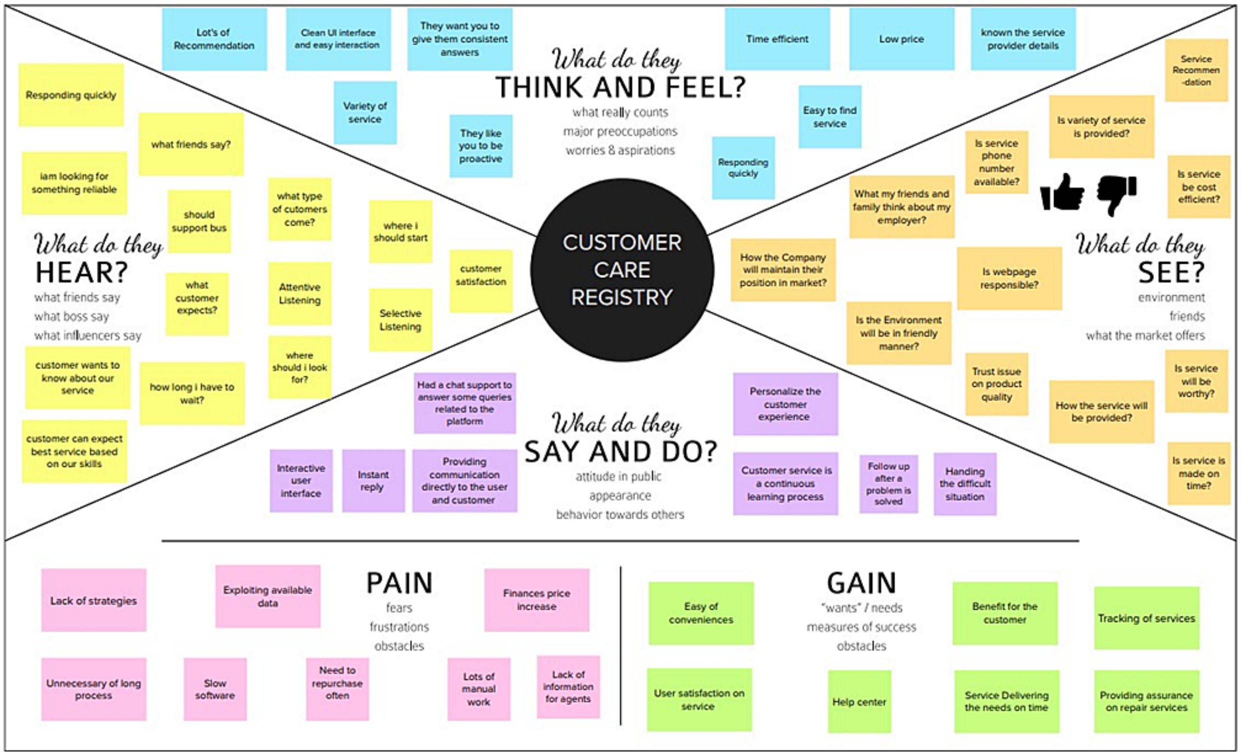
3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviorsand attitudes.

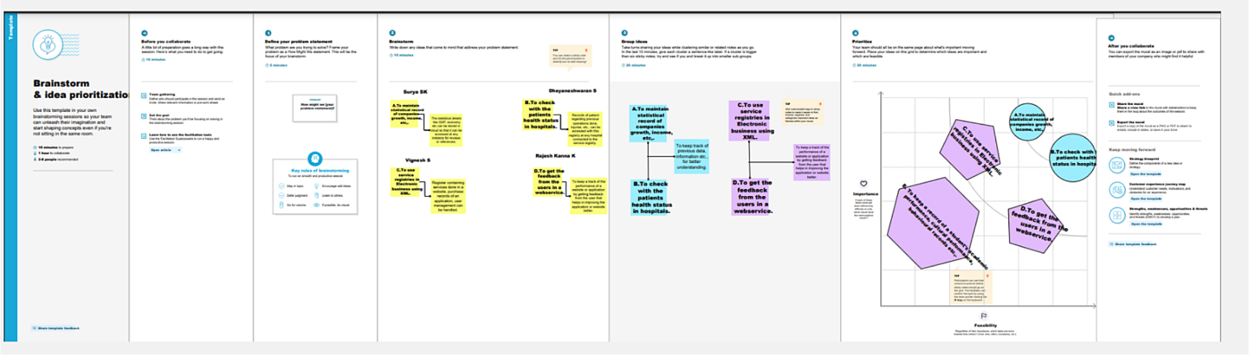
It is a usefultool to helpsteams better understand their users.

Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user’s perspective along with his or her goals and challenges



CUSTOMER CARE REGISTRY
What do they THINK AND FEEL?
what really counts
major preoccupations
worries & aspirations
Lot's of Recommendation
Clean UI interface and easy interaction
They want you to give them consistent answers
Variety of service
They like you to be proactive
Time efficient
Low price
known the service provider details
Easy to find service
Responding quickly
What do they HEAR?
what friends say
what boss say
what influencers say
Responding quickly
what friends say?
iam looking for something reliable
should support bus
what type of cutomers come?
where i should start
what customer expects?
Attentive Listening
customer satisfaction
Selective Listening
customer wants to know about our service
how long i have to wait?
where should i look for?
customer can expect best service based on our skills
What do they SEE?
environment
friends
what the market offers
Service Recommen -dation
Is variety of service is provided?
Is service phone number available?
Is service be cost efficient?
What my friends and family think about my employer?
How the Company will maintain their position in market?
Is webpage responsible?
Is the Environment will be in friendly manner?
Trust issue on product quality
How the service will be provided?
Is service will be worthy?
Is service is made on time?
What do they SAY AND DO?
attitude in public
appearance
behavior towards others
Had a chat support to answer some queries related to the platform
Interactive user interface
Instant reply
Providing communication directly to the user and customer
Personalize the customer experience
Customer service is a continuous learning process
Follow up after a problem is solved
Handing the difficult situation
PAIN
fears
frustrations
obstacles
Lack of strategies
Exploiting available data
Finances price increase
Unnecessary of long process
Slow software
Need to repurchase often
Lots of manual work
Lack of information for agents
GAIN
"wants" / needs
measures of success
obstacles
Easy of conveniences
Benefit for the customer
Tracking of services
User satisfaction on service
Help center
Service Delivering the needs on time
Providing assurance on repair services

3.2 Ideation & Brainstorming


Brainstorm & idea prioritization
Brainstorm
10 minutes
Surya SK
A.To maintain statistical record of companies growth, income, etc.,
Dheyaneshwaran S
B.To check with the patients health status in hospitals.
Vignesh S
C.To use service registries in Electronic business using XML.
Rajesh Kanna K
D.To get the feedback from the users in a webservice.
Group ideas
20 minutes
To keep track of previous data, information etc., for better understanding.
Prioritize
20 minutes
Importance
Feasibility
A.To maintain statistical record of companies growth, income, etc.,
B.To check with the patients health status in hospitals
C.To use service registries in Electronic business using XML.
D.To get the feedback from the users in a webservice.
E. To keep a record of a student's academic performance, cultural performance, behavioural records etc.,

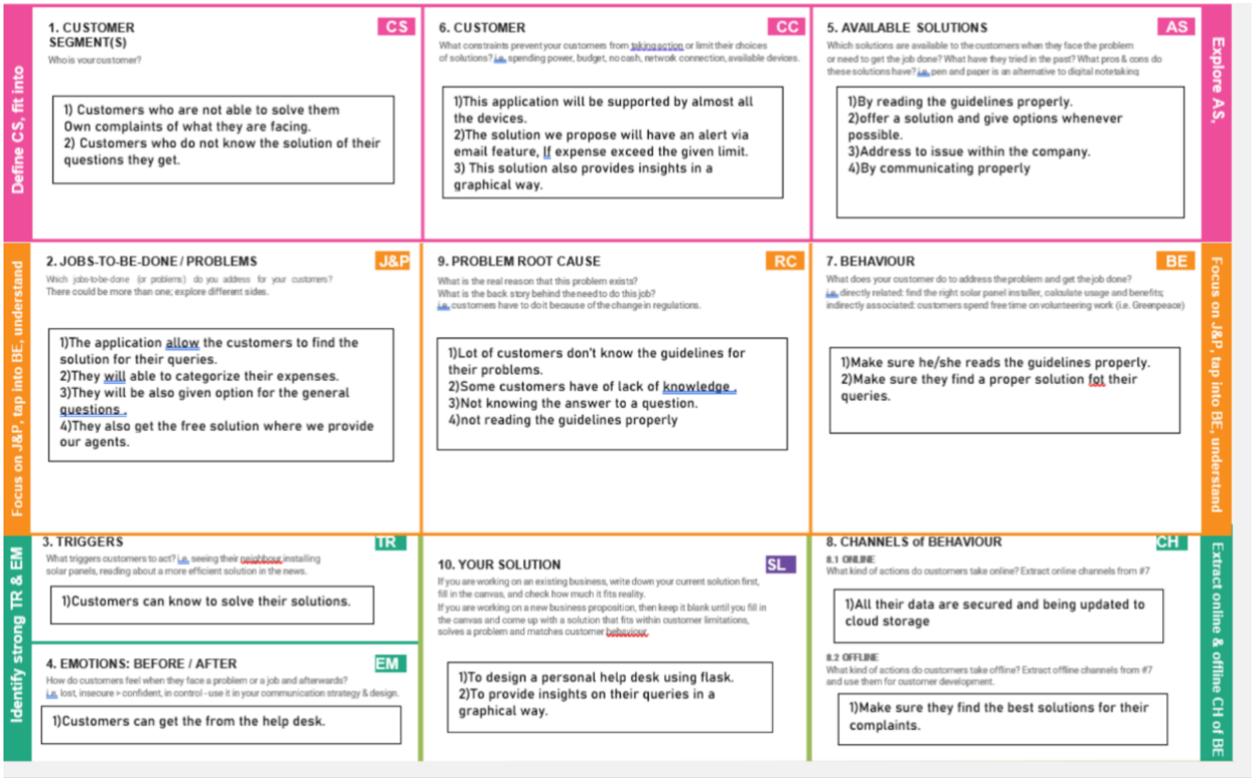
3.3 Proposed Solution

S.No	Parameter	Description
	Problem Statement(Problem to besolved	To solve customer issues using CloudApplicationDevelopment.
	Idea / Solution description	Assigned Agent routing can be solved by directlyrouting to the specific agent about theissue using the specific Email. Automated Ticket closure by using daily sync of the daily database. Status Shown to the Customer can display the status of the ticket to the customer.Regular data retrieval in the form of retrieving lost data

	Novelty / Uniqueness	Assigned Agent Routing, Automated Ticket Closure, Status Shown to the Customer, and Backup data in case of failures
	Social Impact/ Customer Satisfaction	Customer Satisfaction, Customer can track their status and Easy agent communication
	Business Model (Revenue Model)	a. Key Partners are Third-party applications, agents,and customers. b. Activities held as Customer Service,System Maintenance. c. Key Resources support Engineers,Multi-channel. d. Customer Relationship have 24/7 EmailSupport, Knowledge-based channel.

	Scalability of the Solution	The real goal of scaling customer service is providing an environment that will allow your customer service specialists to be as efficient aspossible. An environment where they will be ableto spend less time on gruntwork and more timeon actually resolving critical customer issues.

3.4 Problem Solution fit



Define CS, fit into

1. CUSTOMER SEGMENT(S) CS

Who is your customer?

1) Customers who are not able to solve them Own complaints of what they are facing.
2) Customers who do not know the solution of their questions they get.

6. CUSTOMER 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)This application will be supported by almost all the devices.
2)The solution we propose will have an alert via email feature, If expense exceed the given limit.
3) This solution also provides insights in a graphical way.

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

1)By reading the guidelines properly.
2)offer a solution and give options whenever possible.
3)Address to issue within the company.
4)By communicating properly

Explore AS,

Focus on J&P, tap into BE, understand

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.

1)The application allow the customers to find the solution for their queries.
2)They will able to categorize their expenses.
3)They will be also given option for the general questions .
4)They also get the free solution where we provide our agents.

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.

1)Lot of customers don't know the guidelines for their problems.
2)Some customers have of lack of knowledge .
3)Not knowing the answer to a question.
4)not reading the guidelines properly

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)

1)Make sure he/she reads the guidelines properly.
2)Make sure they find a proper solution fot their queries.

Focus on J&P, tap into BE, understand

Identify strong TR & EM

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.

1)Customers can know to solve their solutions.

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.

1)Customers can get the from the help desk.

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)To design a personal help desk using flask.
2)To provide insights on their queries in a graphical way.

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

1)All their data are secured and being updated to cloud storage

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

1)Make sure they find the best solutions for their complaints.

Extract online & offline CH of BE

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

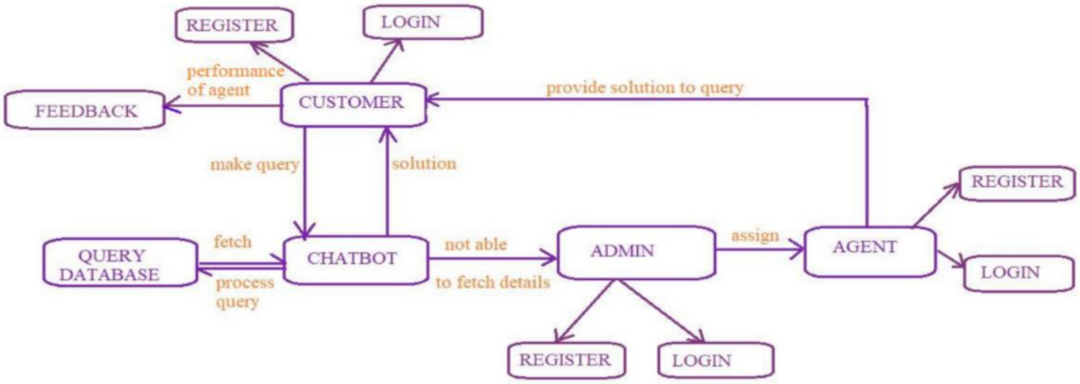
FR No	Functional Requirement(Epic)	Sub Requirement(Story/ Sub-Task)
1	User Registration	Registration through Form Registration through Gmail Registration through Google
2	User Confirmation	Confirmation via Email Confirmation via OTP
3	User Login	Login via Google Login with Email id and Password
4	Admin Login	Login via Google Login with Email id and Password
5	Query Form	Description of the issues Contact information
6	E-mail	Login alertness
7	Feedback	Customer feedback 1

4.2 Non-Functional requirements

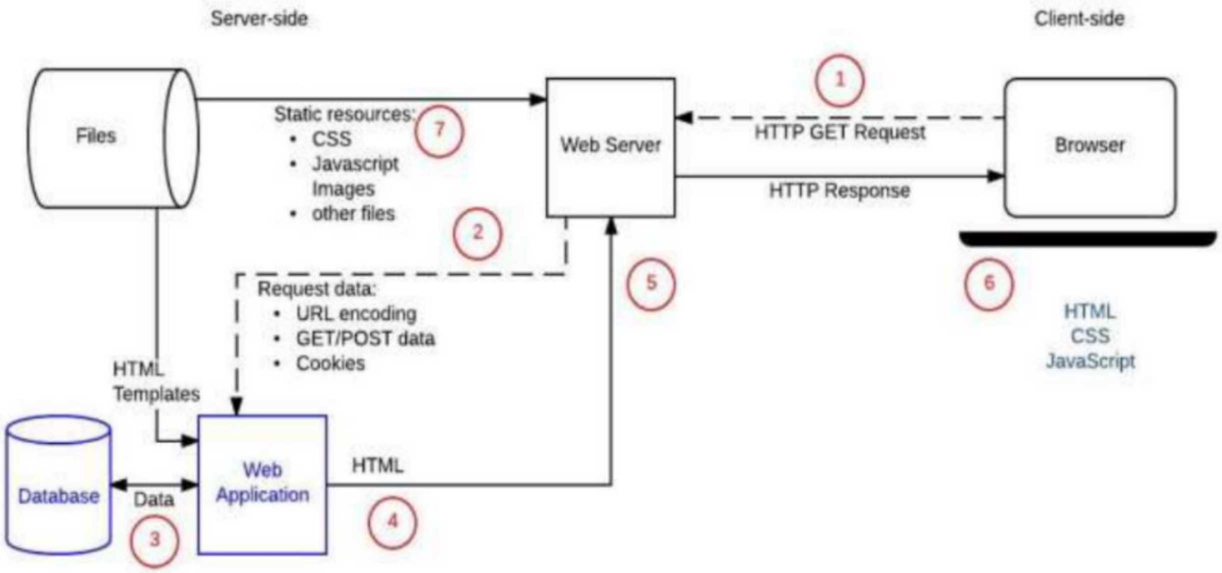
FR No	Non-Functional Requirement	Description
	Usability	To provide the solution to the problem
	Security	Track of login authentication
	Reliability	Tracking of decade status through email
	Performance	Effective development of web application
	Availability	24/7 service
	Scalability	Agents scalability as per the number of customers

5. PROJECT DESIGN

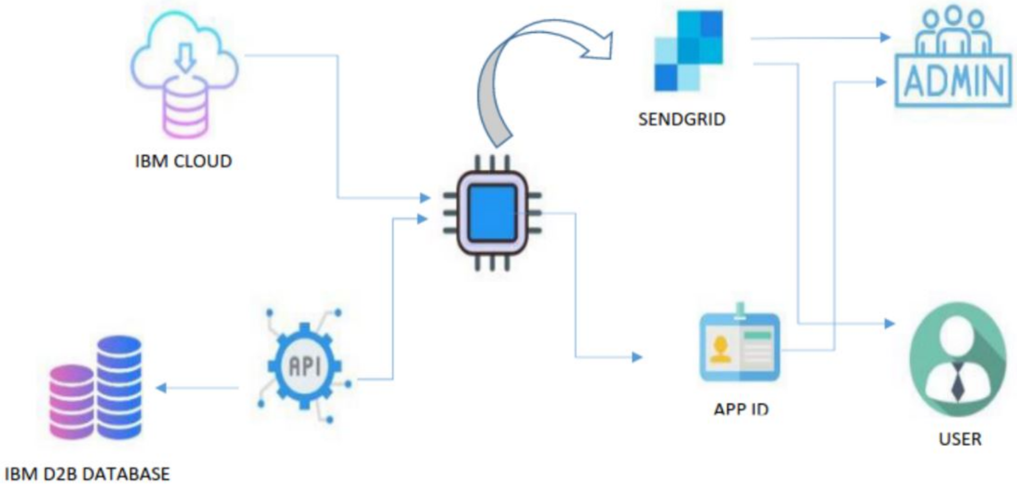
5.1 Data Flow Diagrams



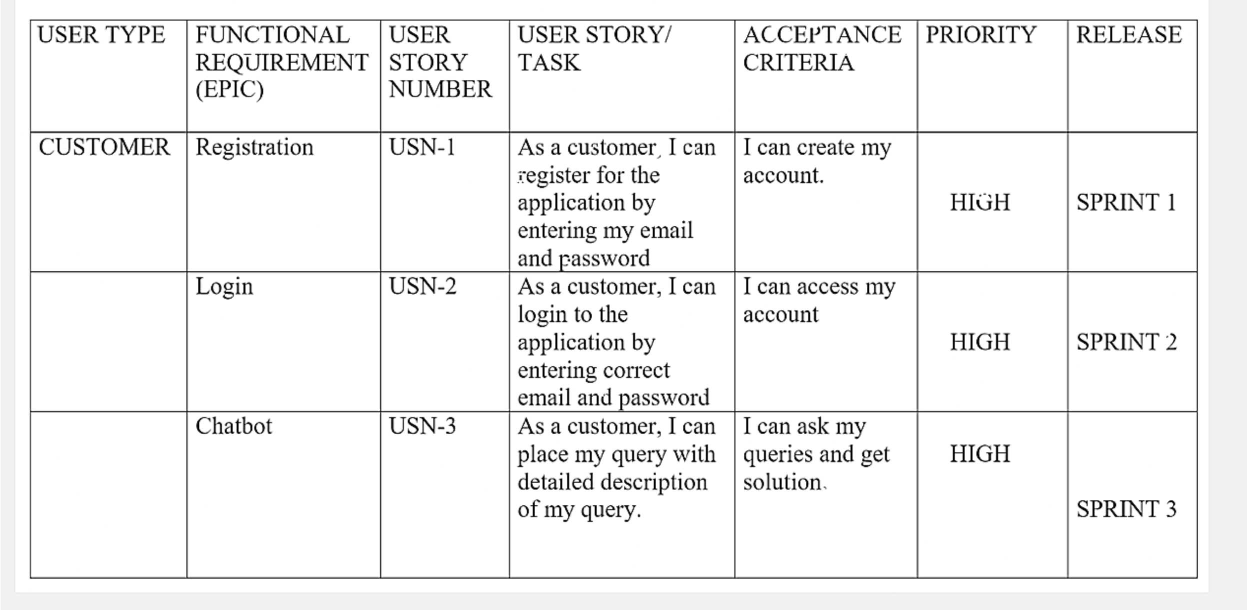
5.2 Solution & Technical Architecture

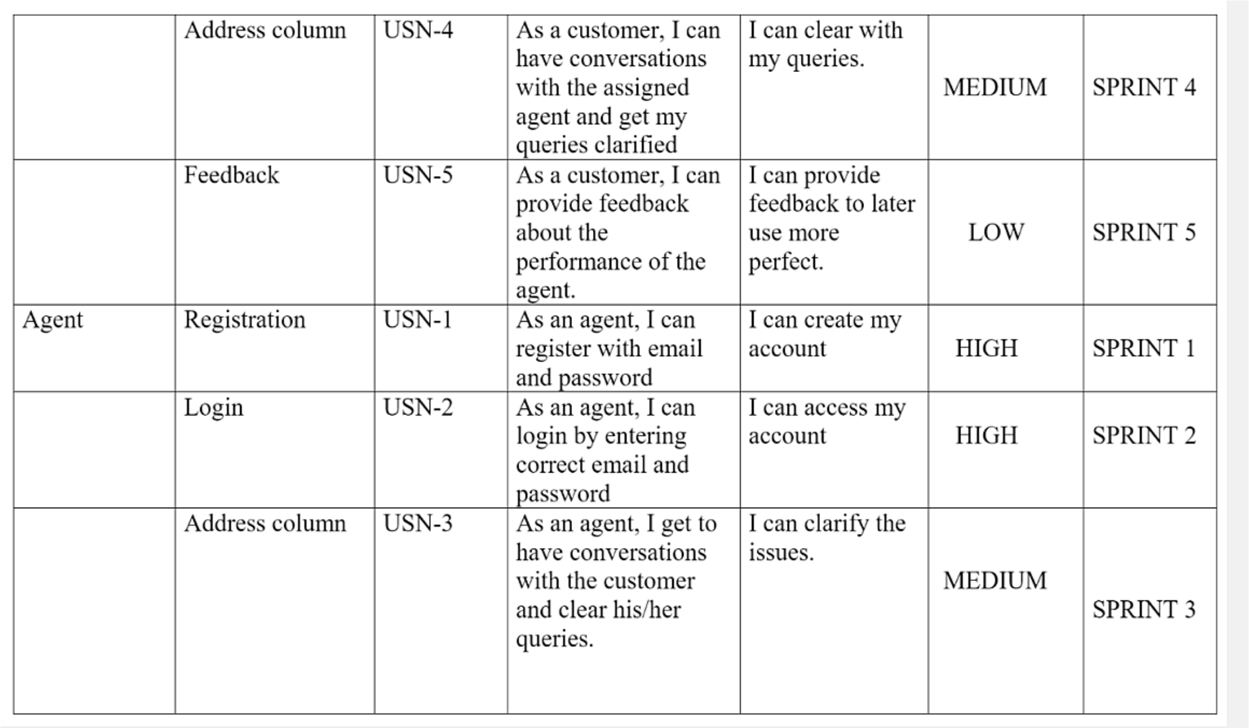


TECHNOLOGY ARCHITECTURE

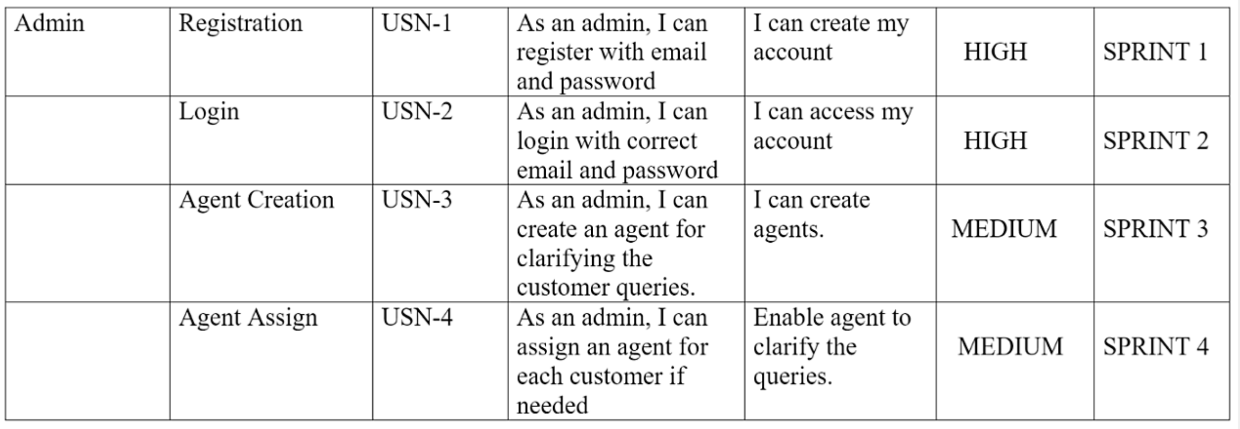


5.3 User Stories





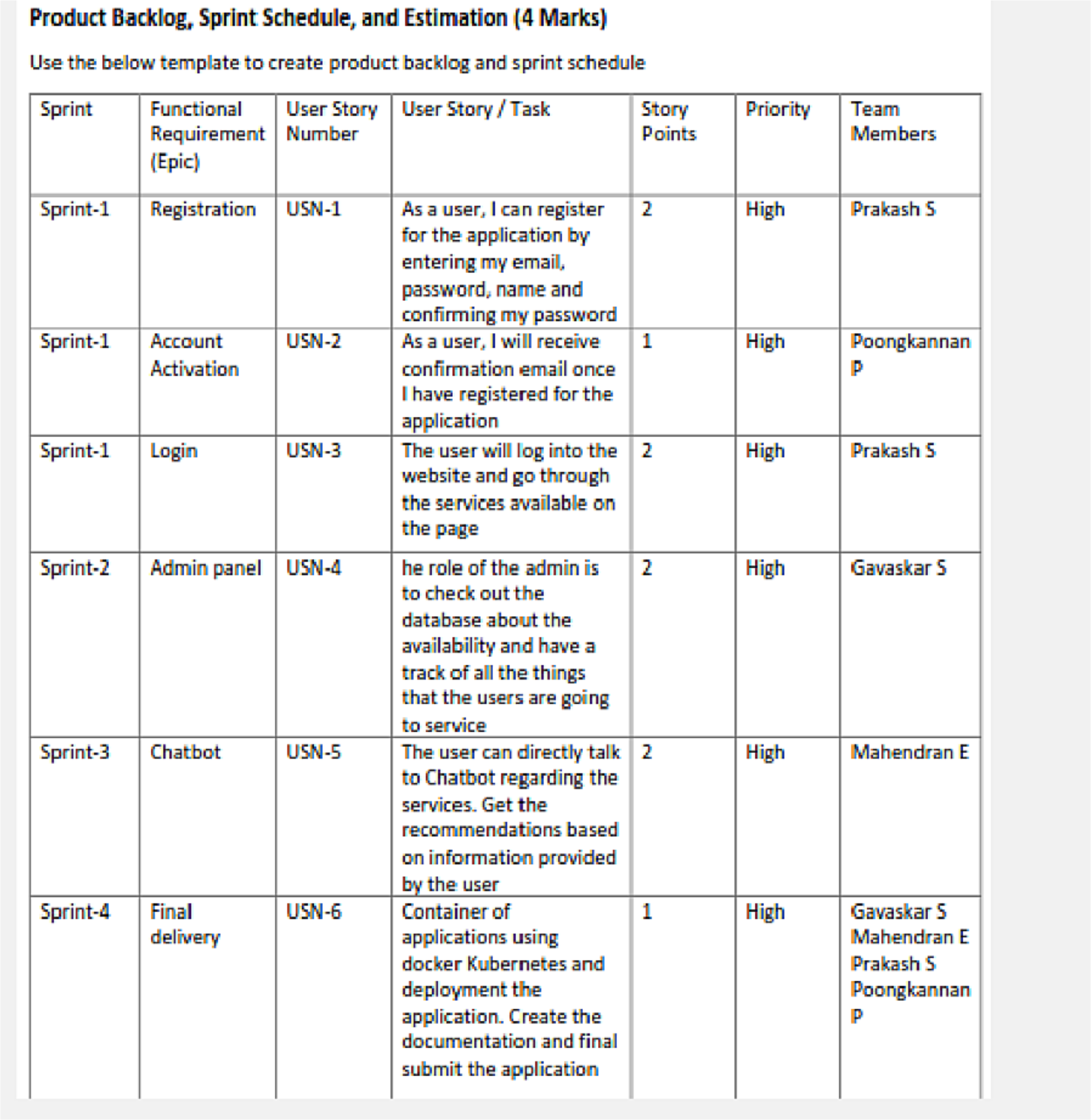
USER TYPE	FUNCTIONAL REQUIREMENT (EPIC)	USER STORY NUMBER	USER STORY/ TASK	ACCEPTANCE CRITERIA	PRIORITY	RELEASE
CUSTOMER	Registration	USN-1	As a customer, I can register for the application by entering my email and password	I can create my account.	HIGH	SPRINT 1
	Login	USN-2	As a customer, I can login to the application by entering correct email and password	I can access my account	HIGH	SPRINT 2
	Chatbot	USN-3	As a customer, I can place my query with detailed description of my query.	I can ask my queries and get solution.	HIGH	SPRINT 3
	Address column	USN-4	As a customer, I can have conversations with the assigned agent and get my queries clarified	I can clear with my queries.	MEDIUM	SPRINT 4
	Feedback	USN-5	As a customer, I can provide feedback about the performance of the agent.	I can provide feedback to later use more perfect.	LOW	SPRINT 5
Agent	Registration	USN-1	As an agent, I can register with email and password	I can create my account	HIGH	SPRINT 1
	Login	USN-2	As an agent, I can login by entering correct email and password	I can access my account	HIGH	SPRINT 2
	Address column	USN-3	As an agent, I get to have conversations with the customer and clear his/her queries.	I can clarify the issues.	MEDIUM	SPRINT 3



Admin	Registration	USN-1	As an admin, I can register with email and password	I can create my account	HIGH	SPRINT 1
	Login	USN-2	As an admin, I can login with correct email and password	I can access my account	HIGH	SPRINT 2
	Agent Creation	USN-3	As an admin, I can create an agent for clarifying the customer queries.	I can create agents.	MEDIUM	SPRINT 3
	Agent Assign	USN-4	As an admin, I can assign an agent for each customer if needed	Enable agent to clarify the queries.	MEDIUM	SPRINT 4

6.PROJECT PLANNING & SCEDULING

6.1 SPRINT PLANNING & ESTIMATION

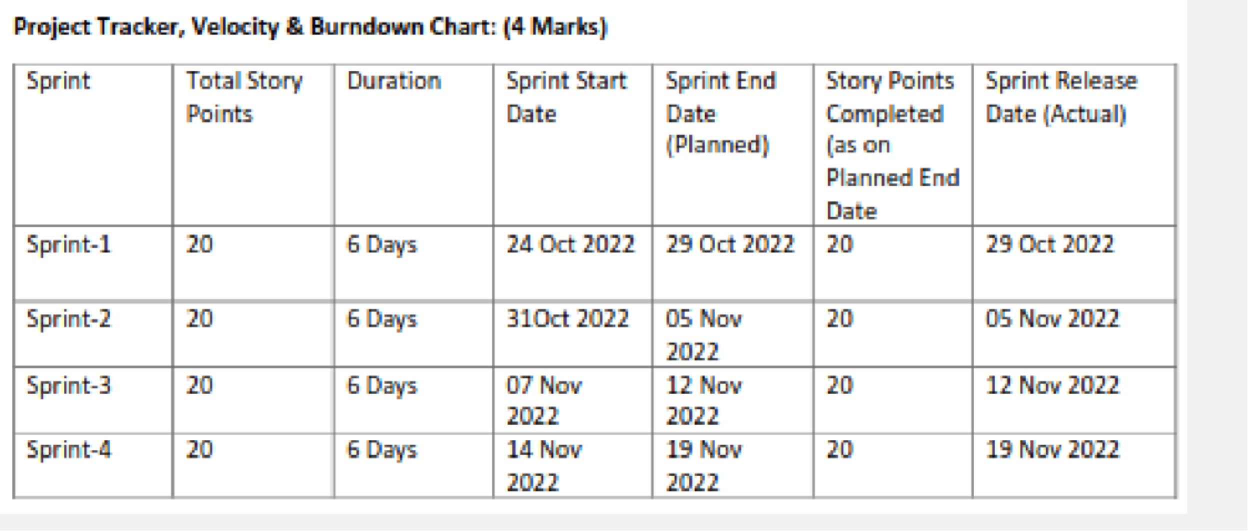


Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Use the below template to create product backlog and sprint schedule

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, name and confirming my password	2	High	Prakash S
Sprint-1	Account Activation	USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Poongkannan P
Sprint-1	Login	USN-3	The user will log into the website and go through the services available on the page	2	High	Prakash S
Sprint-2	Admin panel	USN-4	he role of the admin is to check out the database about the availability and have a track of all the things that the users are going to service	2	High	Gavaskar S
Sprint-3	Chatbot	USN-5	The user can directly talk to Chatbot regarding the services. Get the recommendations based on information provided by the user	2	High	Mahendran E
Sprint-4	Final delivery	USN-6	Container of applications using docker Kubernetes and deployment the application. Create the documentation and final submit the application	1	High	Gavaskar S Mahendran E Prakash S Poongkannan P

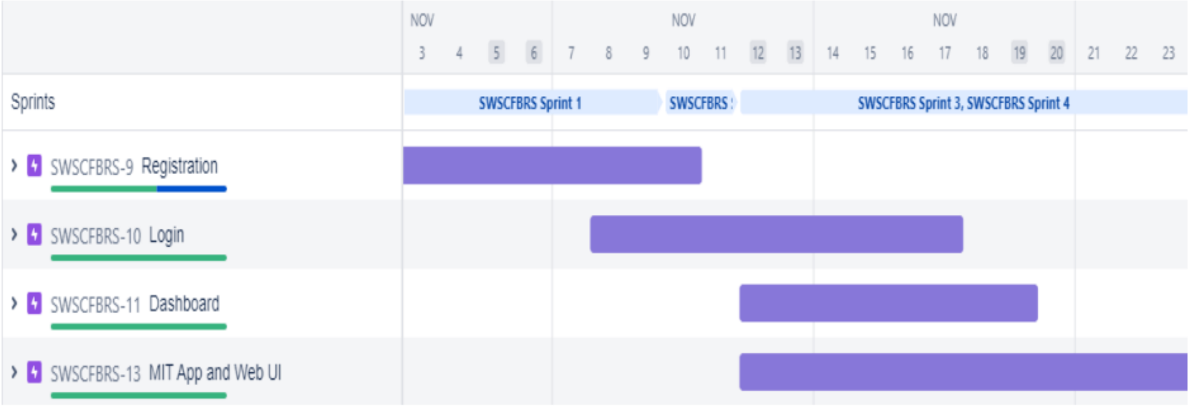
6.2 SPRINT DELIVERY SCHEDULE



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	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31Oct 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

6.3 REPORTS FROM JIRA



7. CODING & SOLUTIONING:

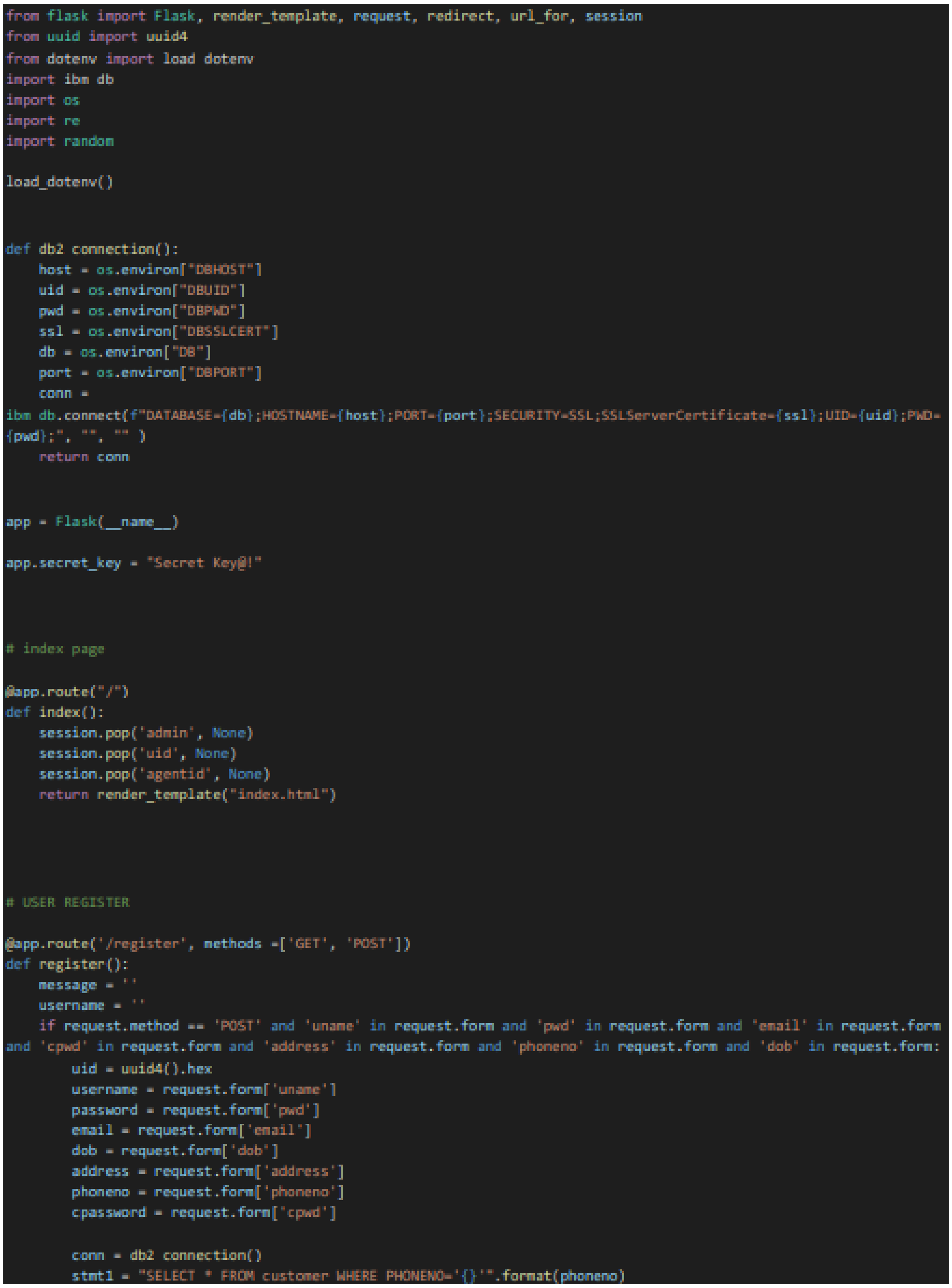
7.1 Feature 1

- Friendliness
- Empathy
- Fairness
- Control
- Alternatives
- Information
- Time

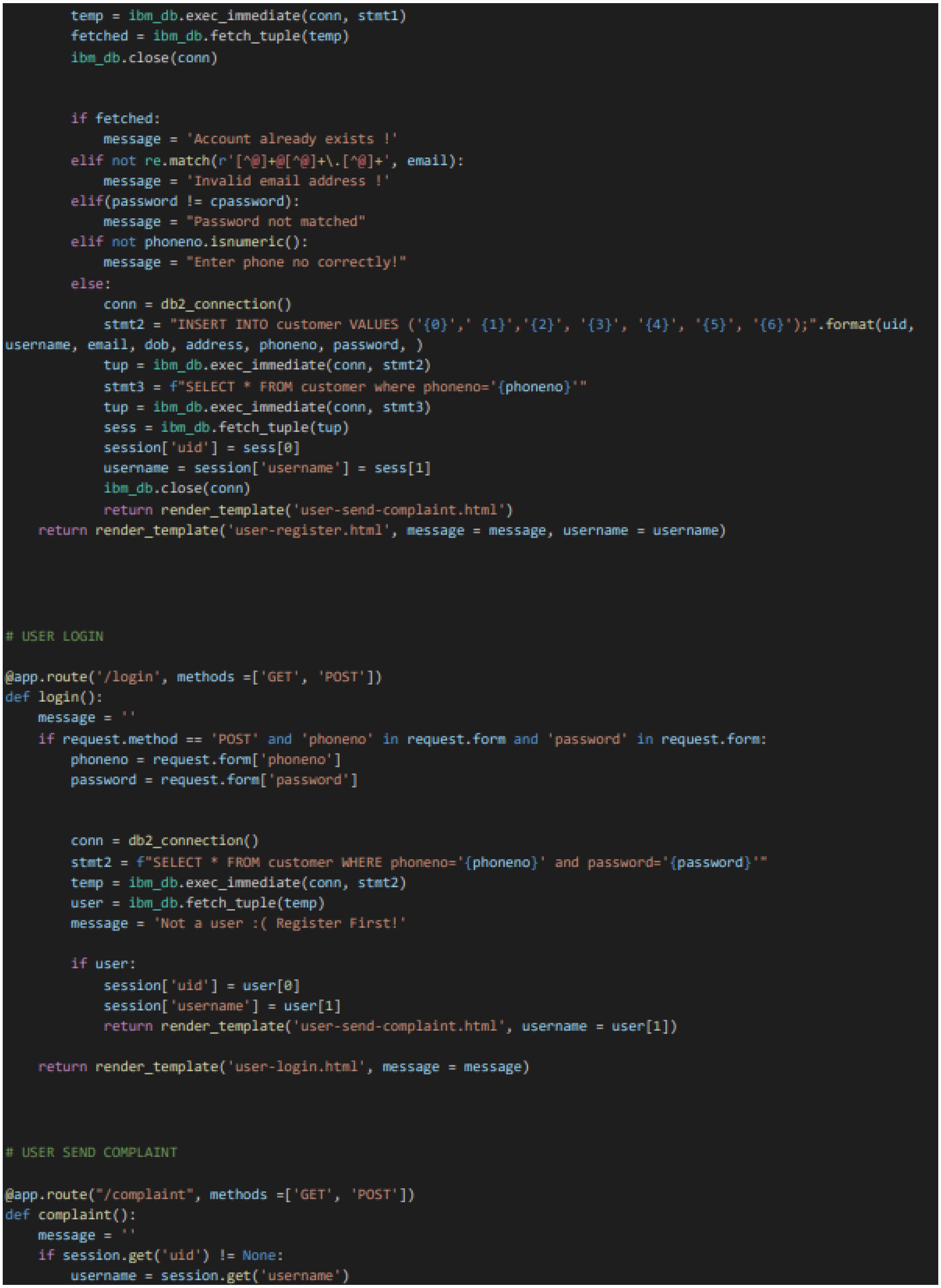
7.2 Feature 2

- 1. Unified Customer View within a Dashboard
- 2. Contextual Voice Management System
- 3. Universal Agent Management Solution
- 4.Internal Communication Mechanism

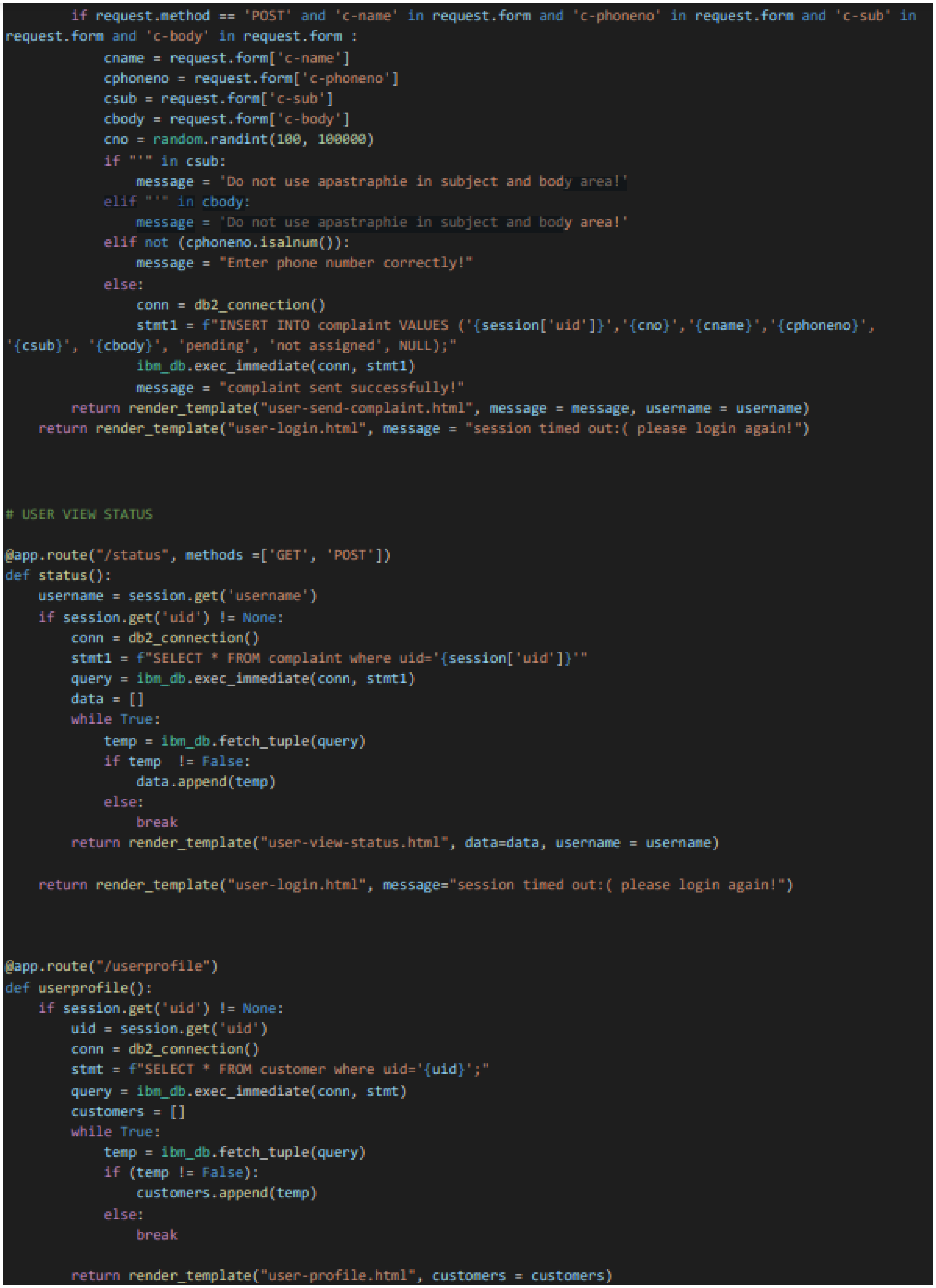
7.3 Database Schema



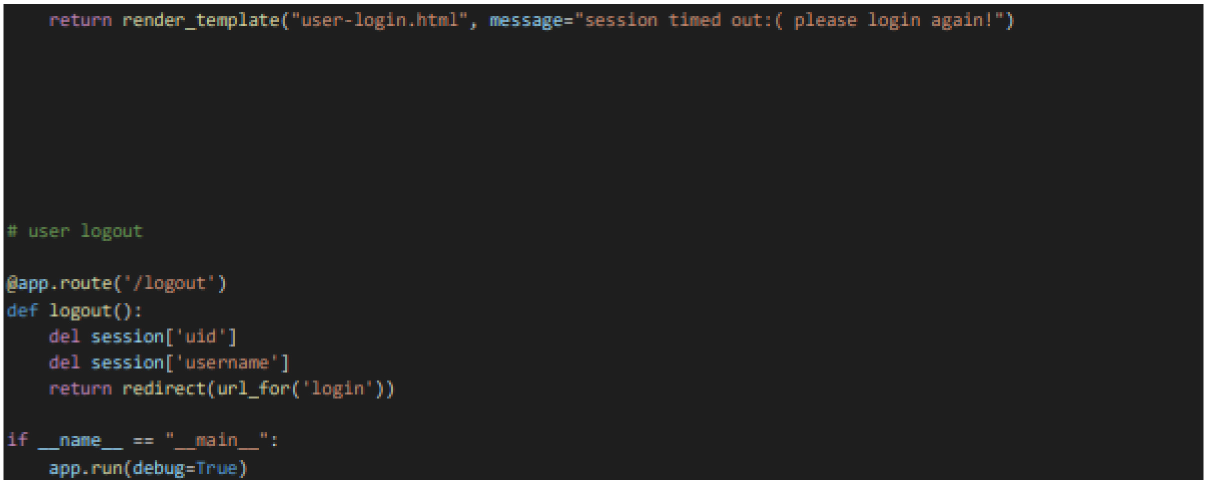
```
from flask import Flask, render_template, request, redirect, url_for, session
from uuid import uuid4
from dotenv import load dotenv
import ibm db
import os
import re
import random

load_dotenv()


def db2 connection():
    host = os.environ["DBHOST"]
    uid = os.environ["DBUID"]
    pwd = os.environ["DBPWD"]
    ssl = os.environ["DBSSLCERT"]
    db = os.environ["DB"]
    port = os.environ["DBPORT"]
    conn =
ibm db.connect(f"DATABASE={db};HOSTNAME={host};PORT={port};SECURITY=SSL;SSLServerCertificate={ssl};UID={uid};PWD=
{pwd};", "", "" )
    return conn


app = Flask(__name__)

app.secret_key = "Secret Key@!"



# index page

@app.route("/")
def index():
    session.pop('admin', None)
    session.pop('uid', None)
    session.pop('agentid', None)
    return render_template("index.html")




# USER REGISTER

@app.route('/register', methods =['GET', 'POST'])
def register():
    message = ''
    username = ''
    if request.method == 'POST' and 'uname' in request.form and 'pwd' in request.form and 'email' in request.form
and 'cpwd' in request.form and 'address' in request.form and 'phoneno' in request.form and 'dob' in request.form:
        uid = uuid4().hex
        username = request.form['uname']
        password = request.form['pwd']
        email = request.form['email']
        dob = request.form['dob']
        address = request.form['address']
        phoneno = request.form['phoneno']
        cpassword = request.form['cpwd']

        conn = db2 connection()
        stmt1 = "SELECT * FROM customer WHERE PHONENO='{}'".format(phoneno)
```



```
        temp = ibm_db.exec_immediate(conn, stmt1)
        fetched = ibm_db.fetch_tuple(temp)
        ibm_db.close(conn)


        if fetched:
            message = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            message = 'Invalid email address !'
        elif(password != cpassword):
            message = "Password not matched"
        elif not phoneno.isnumeric():
            message = "Enter phone no correctly!"
        else:
            conn = db2_connection()
            stmt2 = "INSERT INTO customer VALUES ('{0}',' {1}','{2}', '{3}', '{4}', '{5}', '{6}');".format(uid,
username, email, dob, address, phoneno, password, )
            tup = ibm_db.exec_immediate(conn, stmt2)
            stmt3 = f"SELECT * FROM customer where phoneno='{phoneno}'"
            tup = ibm_db.exec_immediate(conn, stmt3)
            sess = ibm_db.fetch_tuple(tup)
            session['uid'] = sess[0]
            username = session['username'] = sess[1]
            ibm_db.close(conn)
            return render_template('user-send-complaint.html')
    return render_template('user-register.html', message = message, username = username)




# USER LOGIN

@app.route('/login', methods =['GET', 'POST'])
def login():
    message = ''
    if request.method == 'POST' and 'phoneno' in request.form and 'password' in request.form:
        phoneno = request.form['phoneno']
        password = request.form['password']


        conn = db2_connection()
        stmt2 = f"SELECT * FROM customer WHERE phoneno='{phoneno}' and password='{password}'"
        temp = ibm_db.exec_immediate(conn, stmt2)
        user = ibm_db.fetch_tuple(temp)
        message = 'Not a user :( Register First!'

        if user:
            session['uid'] = user[0]
            session['username'] = user[1]
            return render_template('user-send-complaint.html', username = user[1])

    return render_template('user-login.html', message = message)



# USER SEND COMPLAINT

@app.route("/complaint", methods =['GET', 'POST'])
def complaint():
    message = ''
    if session.get('uid') != None:
        username = session.get('username')
```



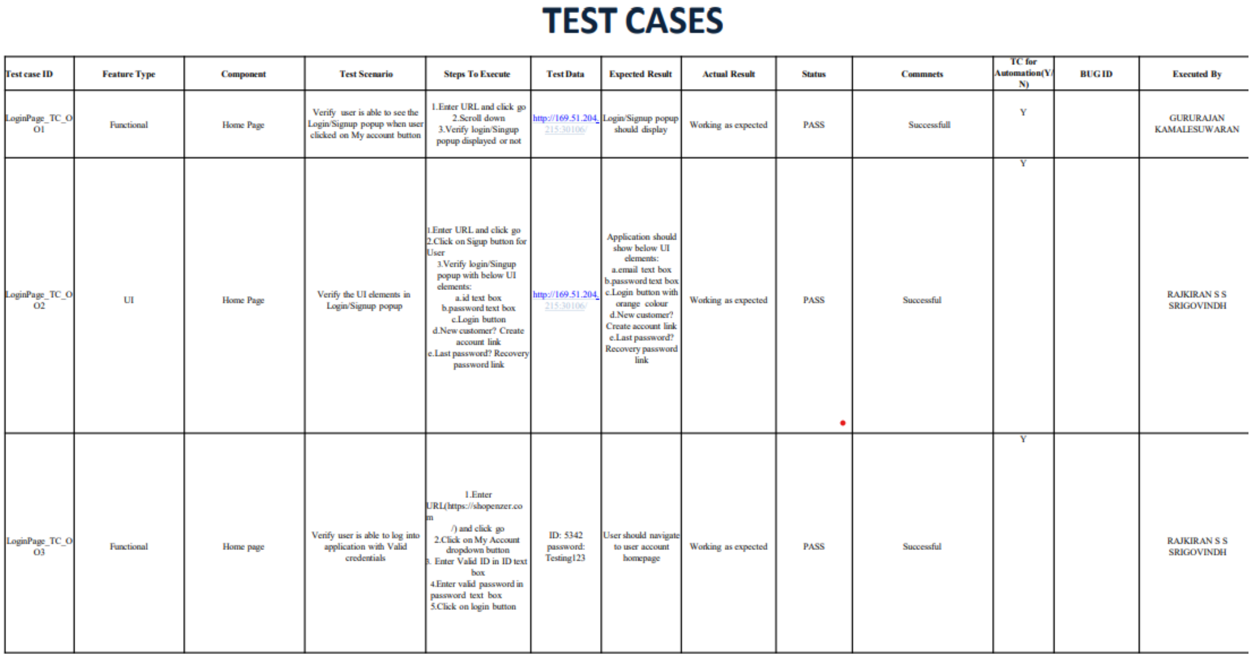
```
        if request.method == 'POST' and 'c-name' in request.form and 'c-phoneno' in request.form and 'c-sub' in
request.form and 'c-body' in request.form :
            cname = request.form['c-name']
            cphoneno = request.form['c-phoneno']
            csub = request.form['c-sub']
            cbody = request.form['c-body']
            cno = random.randint(100, 100000)
            if "'" in csub:
                message = 'Do not use apastraphie in subject and body area!'
            elif "'" in cbody:
                message = 'Do not use apastraphie in subject and body area!'
            elif not (cphoneno.isalnum()):
                message = "Enter phone number correctly!"
            else:
                conn = db2_connection()
                stmt1 = f"INSERT INTO complaint VALUES ('{session['uid']}','{cno}','{cname}','{cphoneno}',
'{csub}', '{cbody}', 'pending', 'not assigned', NULL);"
                ibm_db.exec_immediate(conn, stmt1)
                message = "complaint sent successfully!"
        return render_template("user-send-complaint.html", message = message, username = username)
    return render_template("user-login.html", message = "session timed out:( please login again!")



# USER VIEW STATUS

@app.route("/status", methods =['GET', 'POST'])
def status():
    username = session.get('username')
    if session.get('uid') != None:
        conn = db2_connection()
        stmt1 = f"SELECT * FROM complaint where uid='{session['uid']}'"
        query = ibm_db.exec_immediate(conn, stmt1)
        data = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if temp  != False:
                data.append(temp)
            else:
                break
        return render_template("user-view-status.html", data=data, username = username)

    return render_template("user-login.html", message="session timed out:( please login again!")



@app.route("/userprofile")
def userprofile():
    if session.get('uid') != None:
        uid = session.get('uid')
        conn = db2_connection()
        stmt = f"SELECT * FROM customer where uid='{uid}';"
        query = ibm_db.exec_immediate(conn, stmt)
        customers = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                customers.append(temp)
            else:
                break

        return render_template("user-profile.html", customers = customers)
```



```
    return render_template("user-login.html", message="session timed out:( please login again!")




# user logout

@app.route('/logout')
def logout():
    del session['uid']
    del session['username']
    return redirect(url_for('login'))

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

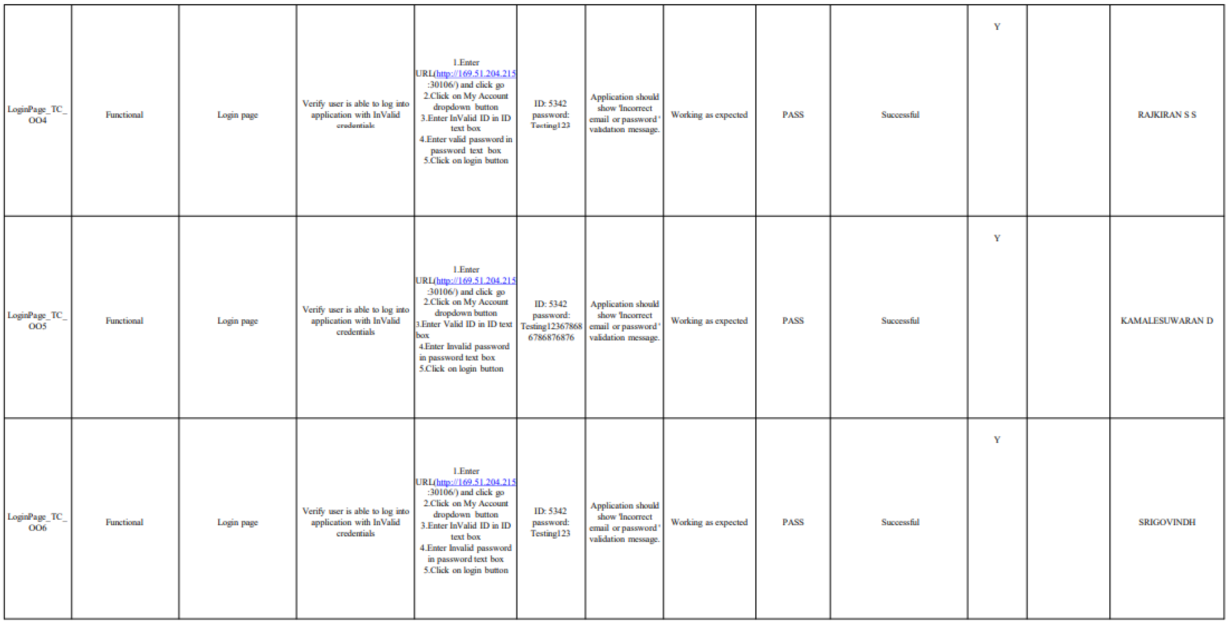
8.TESTING

8.1 Test cases



TEST CASES

Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Commnets	TC for Automation(Y/N)	BUG ID	Executed By
LoginPage_TC_OO1	Functional	Home Page	Verify user is able to see the Login/Signup popup when user clicked on My account button	1.Enter URL and click go 2.Scroll down 3.Verify login/Singup popup displayed or not	http://169.51.204.215:30106/	Login/Signup popup should display	Working as expected	PASS	Successfull	Y		GURURAJAN KAMALESUWARAN
LoginPage_TC_OO2	UI	Home Page	Verify the UI elements in Login/Signup popup	1.Enter URL and click go 2.Click on Sigup button for User 3.Verify login/Singup popup with below UI elements: a.id text box b.password text box c.Login button d.New customer? Create account link e.Last password? Recovery password link	http://169.51.204.215:30106/	Application should show below UI elements: a.email text box b.password text box c.Login button with orange colour d.New customer? Create account link e.Last password? Recovery password link	Working as expected	PASS	Successful	Y		RAJKIRAN S S SRIGOVINDH
LoginPage_TC_OO3	Functional	Home page	Verify user is able to log into application with Valid credentials	1.Enter URL(https://shopenzer.com/) and click go 2.Click on My Account dropdown button 3. Enter Valid ID in ID text box 4.Enter valid password in password text box 5.Click on login button	ID: 5342 password: Testing123	User should navigate to user account homepage	Working as expected	PASS	Successful	Y		RAJKIRAN S S SRIGOVINDH



LoginPage_TC_OO4	Functional	Login page	Verify user is able to log into application with InValid credentials	1.Enter URL(http://169.51.204.215:30106/) and click go 2.Click on My Account dropdown button 3.Enter InValid ID in ID text box 4.Enter valid password in password text box 5.Click on login button	ID: 5342 password: Testing123	Application should show 'Incorrect email or password' validation message.	Working as expected	PASS	Successful	Y		RAJKIRAN S S
LoginPage_TC_OO5	Functional	Login page	Verify user is able to log into application with InValid credentials	1.Enter URL(http://169.51.204.215:30106/) and click go 2.Click on My Account dropdown button 3.Enter Valid ID in ID text box 4.Enter Invalid password in password text box 5.Click on login button	ID: 5342 password: Testing123678686786876876	Application should show 'Incorrect email or password' validation message.	Working as expected	PASS	Successful	Y		KAMALESUWARAN D
LoginPage_TC_OO6	Functional	Login page	Verify user is able to log into application with InValid credentials	1.Enter URL(http://169.51.204.215:30106/) and click go 2.Click on My Account dropdown button 3.Enter InValid ID in ID text box 4.Enter Invalid password in password text box 5.Click on login button	ID: 5342 password: Testing123	Application should show 'Incorrect email or password' validation message.	Working as expected	PASS	Successful	Y		SRIGOVINDH

8.2 User Acceptance Testing

1. Purpose of Document

The purpose of this document is to briefly explain the test coverage and open issues of the [Customer Care Registry] project at the time of the release to User Acceptance Testing (UAT).

2. Defect Analysis

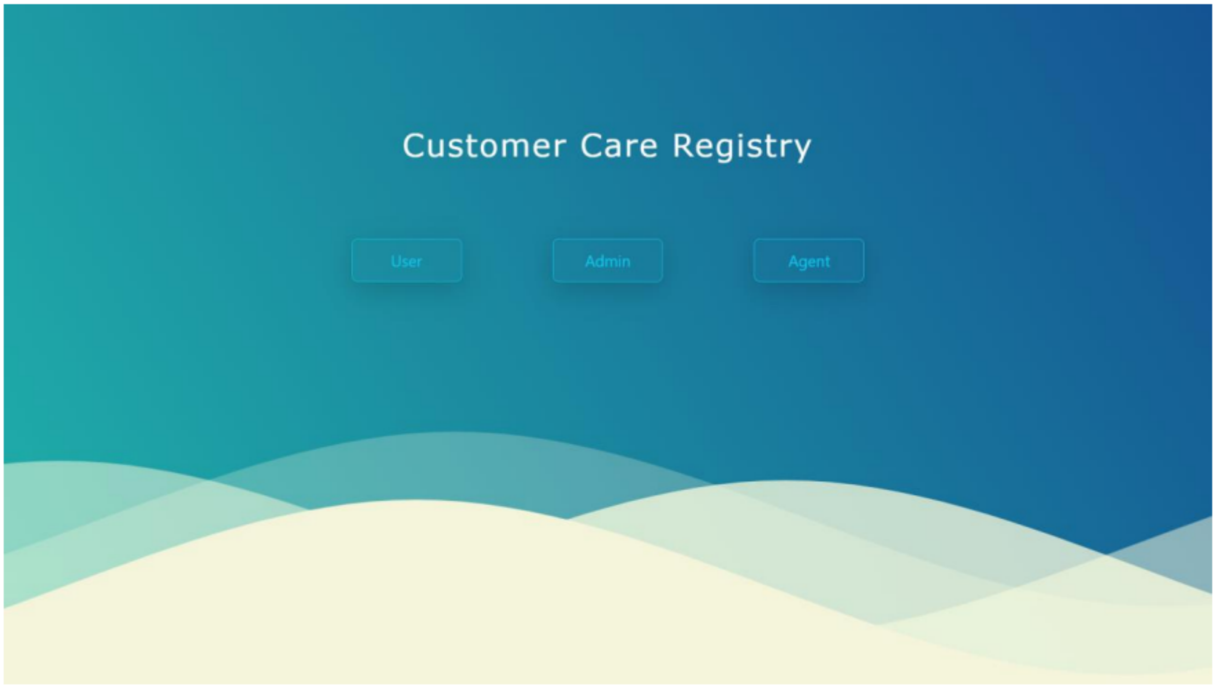
This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	4	5	5	24
Duplicate	2	0	2	0	4
External	5	3	2	1	11
Fixed	15	5	5	10	35
Not Reproduced	0	0	0	0	0
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	32	17	17	18	84

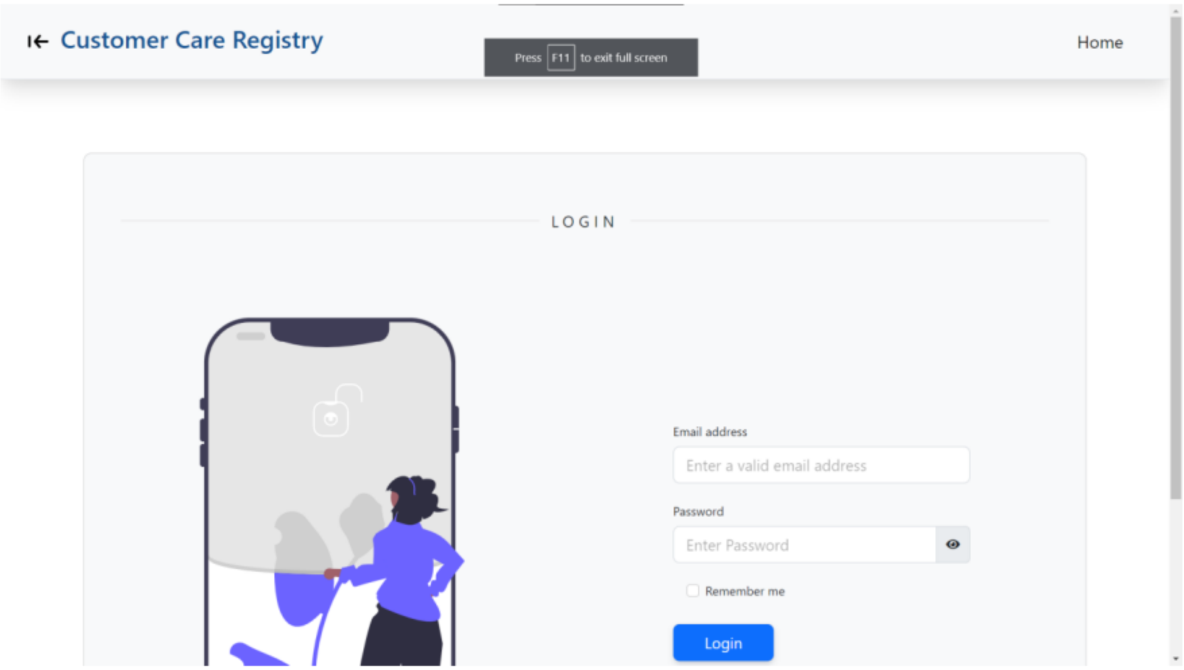
9.Results

9.1 Performance metrics

INDEX PAGE



USER LOGN



Customer Care Registry

Press F11 to exit full screen

Home

LOGIN

Email address

Enter a valid email address

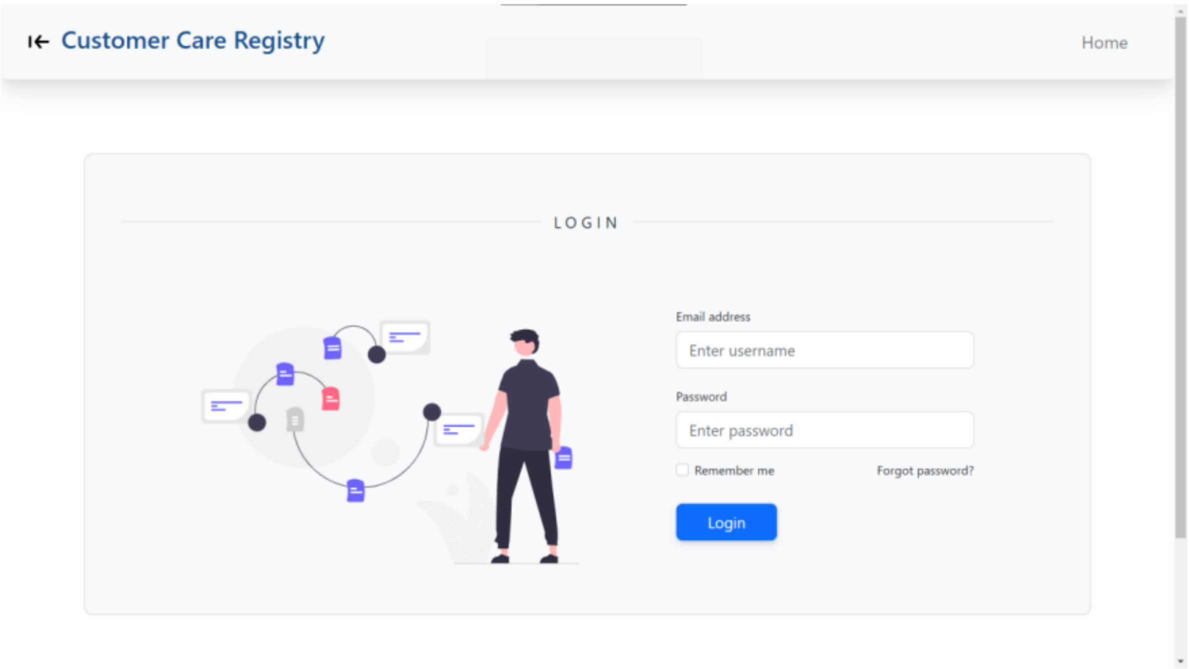
Password

Enter Password

□ Remember me

Login

ADMIN LOGIN



Customer Care Registry

Home

LOGIN

Email address

Enter username

Password

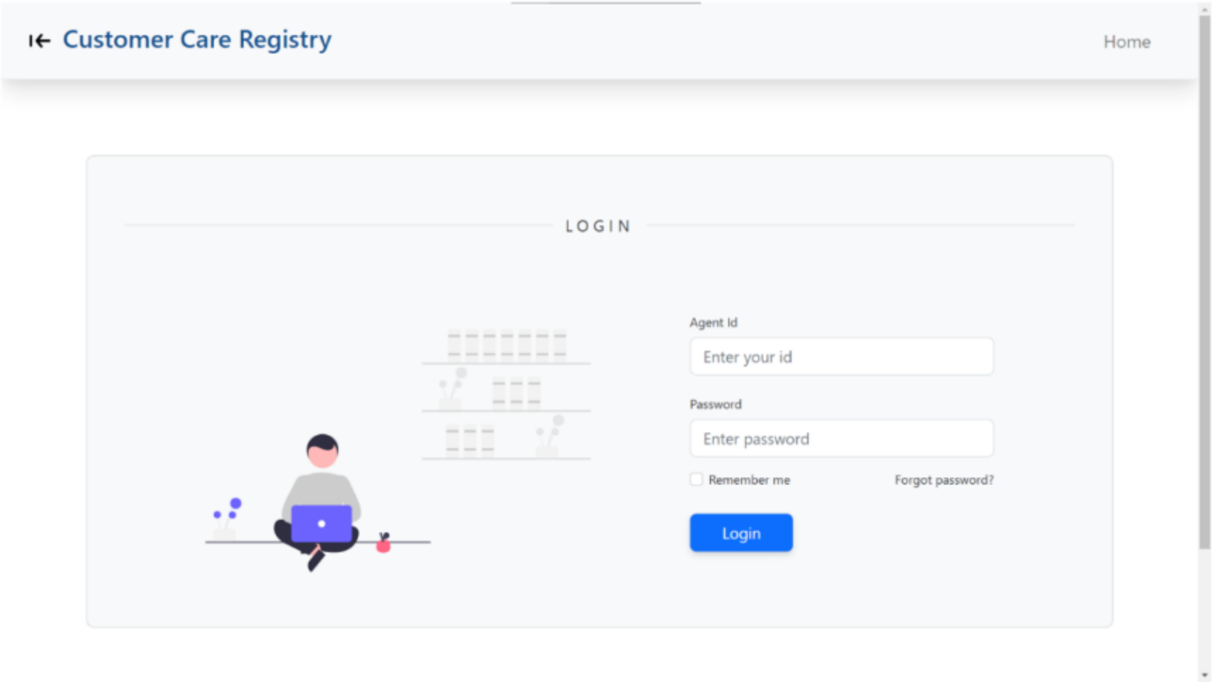
Enter password

□ Remember me

Forgot password?

Login

AGENT LOGIN



10.ADVANTAGES AND DISADVANTAGES

ADVANTAGES

1.Customer loyalty

Loyal customers have many benefits for businesses. 91% of customers say a positive customer service experience makes them more likely to make a further purchase . Also, investing in new customers is five times more expensive than retaining existing ones . Creating loyal customers through good customer service can therefore provide businesses with lucrative long-term relationships.

2. Increase profits

These long-term customer relationships established through customer service can help businesses become more profitable. Businesses can grow revenues between

4% and 8% above their market when they prioritise better customer service experiences . Creating a better customer service experience than those offered by competitors can help businesses to standout in their market place, and in turn make more sales.

3. Customer recommendations

Providing good customer service can create satisfied customers, who are then more likely to recommend the business to others. 94% of customers will recommend a company whose service they rate as “very good” . This is useful, as 90% of customers are influenced by positive reviews when buying a product. Customers recommending a company through word of mouth or online reviews can improve the credibility of the business.

4. Increase conversion

Good customer service can help businesses turn leads into sales. 78% of customers say they have backed out of a purchase due to a poor customer experience. It is therefore safe to assume that providing good customer service will help to increase customer confidence and in turn increase conversion.

Disadvantage :

The Consumer Protection Act in India has numerous restrictions and drawbacks, which are listed in this article. Only services for which a particular payment has been made are covered under the consumer protection act. However, it does not protect medical professionals, or hospitals, and covers cases when this act does not apply to free medical care. This act does not apply to mandatory services, such as water supply, that are provided by state agencies. Only two clauses related to the supply of hazardous materials are covered by this act. Consumer redress is not given any power by the consumer protection act.The consumer protection act focuses on the supply of ineffective products, but there are no strict regulations for those who produce it.

11. CONCLUSION

- It is a web-enabled project.
- This project details about the product will be given to the customers in detail with in a short span of time.
- Queries regarding the product or the services will also be clarified. It provides more knowledge about the various technologies.

12. FUTURE SCOPE

- Completion of the development process will result in a software package that will provide user-friendly environment, which is very easy to work with, even for people with very little knowledge of computer.
- Management of various tasks is incorporated in the package and will deliver the required information in a very easy to use and easy to access manner.
- This package will provide accuracy, efficiency, speed and easiness to the end user. Since the system is verified with valid as well as invalid data and is run with an insight into the necessary modifications that may require in the future, it can be maintained successfully without much.

13. APPENDIX

DEMOLINK - https://drive.google.com/file/d/1EE3GiN5PV41IxmYAABmzFeMzT5vIlL1q/view?usp=share_link

Source Code

app.py

```
from flask import Flask, render_template, request, redirect, url_for, session
from uuid import uuid4
from dotenv import load_dotenv
import ibm_db
import os
import re
import random

load_dotenv()


def db2_connection():
   host = os.environ["DBHOST"]
   uid = os.environ["DBUID"]
   pwd = os.environ["DBPWD"]
   ssl = os.environ["DBSSLCERT"]
   db = os.environ["DB"]
   port = os.environ["DBPORT"]
   conn = ibm_db.connect(f"DATABASE={db};HOSTNAME={host};PORT={port};SECURITY=SSL;SSLServerCertificate={ssl};UID={uid};PWD={pwd};", "", "" )
   return conn


app = Flask(__name__)

app.secret_key = "Secret Key@!"



# index page
```

```
@app.route("/")
def index():
    session.pop('admin', None)
    session.pop('uid', None)
    session.pop('agentid', None)
    return render_template("index.html")
```

```
# USER REGISTER

@app.route('/register', methods =['GET', 'POST'])
def register():
    message = ''
    username = ''
    if request.method == 'POST' and 'uname' in request.form and 'pwd' in request.form and 'email'
in request.form and 'cpwd' in request.form and 'address' in request.form and 'phoneno' in
request.form and 'dob' in request.form:
        uid = uuid4().hex
        username = request.form['uname']
        password = request.form['pwd']
        email = request.form['email']
        dob = request.form['dob']
        address = request.form['address']
        phoneno = request.form['phoneno']
        cpassword = request.form['cpwd']

        conn = db2_connection()
        stmt1 = "SELECT * FROM customer WHERE PHONENO='{}'".format(phoneno)
        temp = ibm_db.exec_immediate(conn, stmt1)
        fetched = ibm_db.fetch_tuple(temp)
        ibm_db.close(conn)


        if fetched:
            message = 'Account already exists !'
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            message = 'Invalid email address !'
        elif(password != cpassword):
            message = "Password not matched"
        elif not phoneno.isnumeric():
            message = "Enter phone no correctly!"
        else:
            conn = db2_connection()
            stmt2 = "INSERT INTO customer VALUES ('{0}',' {1}','{2}', '{3}', '{4}', '{5}', '{6}');".format(uid,
username, email, dob, address, phoneno, password, )
            tup = ibm_db.exec_immediate(conn, stmt2)
            stmt3 = f"SELECT * FROM customer where phoneno='{phoneno}'"
            tup = ibm_db.exec_immediate(conn, stmt3)
            sess = ibm_db.fetch_tuple(tup)
            session['uid'] = sess[0]
            username = session['username'] = sess[1]
            ibm_db.close(conn)
            return render_template('user-send-complaint.html')
    return render_template('user-register.html', message = message, username = username)




# USER LOGIN

@app.route('/login', methods =['GET', 'POST'])
def login():
    message = ''
    if request.method == 'POST' and 'phoneno' in request.form and 'password' in request.form:
        phoneno = request.form['phoneno']
        password = request.form['password']


        conn = db2_connection()
        stmt2 = f"SELECT * FROM customer WHERE phoneno='{phoneno}' and
password='{password}'"
        temp = ibm_db.exec_immediate(conn, stmt2)
        user = ibm_db.fetch_tuple(temp)
        message = 'Not a user :( Register First!'

        if user:
            session['uid'] = user[0]
            session['username'] = user[1]
            return render_template('user-send-complaint.html', username = user[1])
    return render_template('user-login.html', message = message)



# USER SEND COMPLAINT

@app.route("/complaint", methods =['GET', 'POST'])
def complaint():
    message = ''
    if session.get('uid') != None:
        username = session.get('username')
        if request.method == 'POST' and 'c-name' in request.form and 'c-phoneno' in request.form
and 'c-sub' in request.form and 'c-body' in request.form :
            cname = request.form['c-name']
            cphoneno = request.form['c-phoneno']
            csub = request.form['c-sub']
            cbody = request.form['c-body']
            cno = random.randint(100, 100000)
            if "'" in csub:
                message = 'Do not use apastraphie in subject and body area!'
            elif "'" in cbody:
                message = 'Do not use apastraphie in subject and body area!'
            elif not (cphoneno.isalnum()):
                message = "Enter phone number correctly!"
            else:
                conn = db2_connection()
                stmt1 = f"INSERT INTO complaint VALUES
('{session['uid']}','{cno}','{cname}','{cphoneno}', '{csub}', '{cbody}', 'pending', 'not assigned', NULL);"
                ibm_db.exec_immediate(conn, stmt1)
                message = "complaint sent successfully!"
        return render_template("user-send-complaint.html", message = message, username =
username)
    return render_template("user-login.html", message = "session timed out:( please login again!")



# USER VIEW STATUS

@app.route("/status", methods =['GET', 'POST'])
def status():
    username = session.get('username')
    if session.get('uid') != None:
        conn = db2_connection()
        stmt1 = f"SELECT * FROM complaint where uid='{session['uid']}'"
        query = ibm_db.exec_immediate(conn, stmt1)
        data = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if temp  != False:
                data.append(temp)
            else:
                break
        return render_template("user-view-status.html", data=data, username = username)

    return render_template("user-login.html", message="session timed out:( please login again!")



@app.route("/userprofile")
def userprofile():
    if session.get('uid') != None:
        uid = session.get('uid')
        conn = db2_connection()
        stmt = f"SELECT * FROM customer where uid='{uid}';"
        query = ibm_db.exec_immediate(conn, stmt)
        customers = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                customers.append(temp)
            else:
                break

        return render_template("user-profile.html", customers = customers)
    return render_template("user-login.html", message="session timed out:( please login again!")
# user logout

@app.route('/logout')
def logout():
    del session['uid']
    del session['username']
    return redirect(url_for('login'))



# ADMIN LOGIN

@app.route("/adminlogin", methods =['GET', 'POST'])
def adminlogin():
    username = "admin"
    password = "@12345"
    message = ""
    if request.method == "POST" and "admin-uname" in request.form and "admin-pwd" in
request.form:
        admin_uname = request.form['admin-uname']
        admin_pwd = request.form['admin-pwd']
        if admin_uname == username and admin_pwd == password:
            session['admin'] = admin_uname

            conn = db2_connection()

            # total agents
            stmt1 = "SELECT COUNT(*) FROM agent;"
            temp1 = ibm_db.exec_immediate(conn, stmt1)
            agents = ibm_db.fetch_tuple(temp1)

            # total complants
            stmt2 = "SELECT COUNT(*) FROM complaint;"
            temp2 = ibm_db.exec_immediate(conn, stmt2)
            complaints = ibm_db.fetch_tuple(temp2)

            # total assigned
            stmt3 = "SELECT COUNT(*) FROM complaint WHERE assignment='assigned';"
            temp3 = ibm_db.exec_immediate(conn, stmt3)
            assigned = ibm_db.fetch_tuple(temp3)
            # total unassigned
            stmt4 = "SELECT COUNT(*) FROM complaint WHERE assignment='not assigned';"
            temp4 = ibm_db.exec_immediate(conn, stmt4)
            unassigned = ibm_db.fetch_tuple(temp4)


            return render_template("admin-dashboard.html" , agents = agents, complaints = complaints, assigned = assigned, unassigned = unassigned)
        else:
            message = "Wrong user name and password!"
            return render_template("admin-login.html", message = message)
    else:
        return render_template("admin-login.html")



# admin dashboard
@app.route('/admindashboard')
def admindashboard():
    if session.get("admin") != None:

        conn = db2_connection()

        # total agents
        stmt1 = "SELECT COUNT(*) FROM agent;"
        temp1 = ibm_db.exec_immediate(conn, stmt1)
        agents = ibm_db.fetch_tuple(temp1)

        # total complants
        stmt2 = "SELECT COUNT(*) FROM complaint;"
        temp2 = ibm_db.exec_immediate(conn, stmt2)
        complaints = ibm_db.fetch_tuple(temp2)

        # total assigned
        stmt3 = "SELECT COUNT(*) FROM complaint WHERE assignment='assigned';"
        temp3 = ibm_db.exec_immediate(conn, stmt3)
        assigned = ibm_db.fetch_tuple(temp3)

        # total unassigned
        stmt4 = "SELECT COUNT(*) FROM complaint WHERE assignment='not assigned';"
        temp4 = ibm_db.exec_immediate(conn, stmt4)
        unassigned = ibm_db.fetch_tuple(temp4)


        return render_template('admin-dashboard.html', agents = agents, complaints = complaints,
assigned = assigned, unassigned = unassigned)
    else:
        return render_template("admin-login.html", message = "Session time out:( Please login!")




# admin add agent

@app.route("/addagent", methods =['GET', 'POST'] )
def addagent():
    message = ""
    if session.get("admin") != None:
        if request.method == "POST" and 'agentid' in request.form and 'adob' in request.form and
'afname' in request.form and 'aemail' in request.form and 'aphoneno' in request.form and
'aaddress' in request.form and 'apwd' in request.form and 'acpwd' in request.form:
            agentid = request.form['agentid']
            a_dob = request.form['adob']
            a_fullname = request.form['afname']
            a_email = request.form['aemail']
            a_phoneno = request.form['aphoneno']
            a_address = request.form['aaddress']
            a_password = request.form['apwd']
            a_cpassword = request.form['acpwd']

            # checks agent already exists
            conn = db2_connection()
            stmt1 = "SELECT * FROM agent WHERE agentid='{}'".format(agentid)
            temp = ibm_db.exec_immediate(conn, stmt1)
            fetched = ibm_db.fetch_tuple(temp)
            ibm_db.close(conn)

            if fetched:
                message = 'Account already exists !'
            elif not re.match(r'[^@]+@[^@]+\.[^@]+', a_email):
                message = 'Invalid email address!'
            elif(a_password != a_cpassword):
                message = "Password not matched!"
            elif not a_phoneno.isnumeric():
                message = "Enter phone no correctly!"
            else:
                conn = db2_connection()
                stmt2 = "INSERT INTO agent VALUES ('{0}',' {1}','{2}', '{3}', '{4}', '{5}', '{6}');".format(agentid,
a_fullname, a_dob, a_email, a_phoneno, a_address, a_password)
                ibm_db.exec_immediate(conn, stmt2)
                message = "Agent created successfully!"
                ibm_db.close(conn)
                return render_template('admin-add-agent.html', message = message)
        return render_template("admin-add-agent.html", message = message)
    return render_template("admin-login.html", message = "Session time out:( Please login!")




# view agents

@app.route("/viewagent")
def viewagent():
    if session.get("admin") != None:
        conn = db2_connection()
        stmt = "SELECT * FROM agent;"
        query = ibm_db.exec_immediate(conn, stmt)
        data = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if temp != False:
                data.append(temp)
            else:
                break
        print(data)
        return render_template("admin-view-agents.html", data = data)

    else:
        return render_template("admin-login.html", message = "session timed out:( please login
again!")
```

```
 # remove agents

@app.route("/viewagent/remove")
def remove():
    if session.get("admin") != None:
        uid = request.args.get("id")
        conn = db2_connection()

        # delete agent
        stmt = f"DELETE FROM agent where agentid='{uid}'"
        ibm_db.exec_immediate(conn, stmt)

        message = "Agent deleted successfully!"

        stmt1 = "SELECT * FROM agent;"
        query = ibm_db.exec_immediate(conn, stmt1)
        data = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if temp != False:
                data.append(temp)
            else:
                break
        return render_template("admin-view-agents.html", data = data, message = message)
    return render_template("admin-login", message = "session timed out:( please login again!")



# assign tasks to agent

@app.route("/assigntasks")
def assigntasks():
    if session.get("admin") != None:
        conn = db2_connection()
        stmt1 = "SELECT * FROM complaint where assignment='not assigned';"
        query1 = ibm_db.exec_immediate(conn, stmt1)
        complaint = []
        while True:
            temp1 = ibm_db.fetch_tuple(query1)
            if temp1 != False:
                complaint.append(temp1)
            else:
                break
        stmt2 = "SELECT * FROM agent;"
        query2 = ibm_db.exec_immediate(conn, stmt2)
        agents = []
        while True:
            temp2 = ibm_db.fetch_tuple(query2)
            if temp2 != False:
                agents.append(temp2)
            else:
                break
        return render_template("admin-assign-tasks.html", complaint = complaint, agents = agents)
    else:
        return render_template("admin-login.html", message = "session timed out:( please login
again!")



# tasks assignment

@app.route("/assigntasks/assign")
def assign():
    if session.get("admin") != None:

        aid = request.args.get('aid')
        cno = request.args.get('cno')

        if ( aid == 'Choose Agent'):
            conn = db2_connection()
            stmt1 = "SELECT * FROM complaint where assignment='not assigned';"
            query1 = ibm_db.exec_immediate(conn, stmt1)
            complaint = []
            while True:
                temp1 = ibm_db.fetch_tuple(query1)
                if temp1 != False:
                    complaint.append(temp1)
                else:
                    break
            stmt2 = "SELECT * FROM agent;"
            query2 = ibm_db.exec_immediate(conn, stmt2)
            agents = []
            while True:
                temp2 = ibm_db.fetch_tuple(query2)
                if temp2 != False:
                    agents.append(temp2)
                else:
                    break
            message = "Choose agent properly!"

        else:
        # assign table
            conn = db2_connection()
            stmt1 = f"update complaint set assignment='assigned', agentid='{aid}' where cno='{cno}';"
            ibm_db.exec_immediate(conn, stmt1)

            stmt1 = "SELECT * FROM complaint where assignment='not assigned';"
            query1 = ibm_db.exec_immediate(conn, stmt1)
            complaint = []
            while True:
                temp1 = ibm_db.fetch_tuple(query1)
                if temp1 != False:
                    complaint.append(temp1)
                else:
                    break
            stmt2 = "SELECT * FROM agent;"
            query2 = ibm_db.exec_immediate(conn, stmt2)
            agents = []
            while True:
                temp2 = ibm_db.fetch_tuple(query2)
                if temp2 != False:
                    agents.append(temp2)
                else:
                    break
            message = "Task assigned successfully!"
        return render_template("admin-assign-tasks.html", complaint = complaint, agents = agents,
message = message)
    else:
        return render_template("admin-login.html", message = "session timed out:( please login
again!")
```

```
# View assigned tasks

# @app.route('/viewassigned')
# def viewassigned():
#     if session.get("admin") != None:
#         conn = db2_connection()

#         # fetching assigned complaints
#         stmt = "SELECT * FROM complaint where assignment='assigned';"
#         query = ibm_db.exec_immediate(conn, stmt)
#         complaint = []
#         while True:
#             temp = ibm_db.fetch_tuple(query)
#             if (temp != False):
#                 complaint.append(temp)
#             else:
#                 break

#         # finding agentid by complaint no
#         for i in complaint:
#             stmt1 = f"SELECT * FROM assign where cno='{i[1]}'"
#             query1 = ibm_db.exec_immediate(conn, stmt1)
#             assignment = []
#             while True:
#                 temp1 = ibm_db.fetch_tuple(query1)
#                 if (temp1 != False):
#                     assignment.append(temp1)

#         # finding respective agents by agentid
#         for j in assignment:
#             stmt2 = f"SELECT * FROM agent where agentid='{j[0]}'"
#             query2 = ibm_db.exec_immediate(conn, stmt2)
#             agents = []
#             while True:
#                 temp2 = ibm_db.fetch_tuple(query2)
#                 if (temp2 != False):
#                     agents.append(temp2)

#         print(complaint)
#         print(assignment)
#        print(agents)
#        return render_template('admin-view-assigned-tasks.html')
#    else:
#        return render_template("admin-login.html", message = "session timed out:( please login
again!")



# ADMIN LOGOUT

@app.route('/adminlogout')
def adminlogout():
    session.pop('admin', None)
    return redirect(url_for('adminlogin'))




# agent login

@app.route("/agentlogin", methods = ['GET', 'POST'])
def agentlogin():
    message = ''
    if request.method == 'POST':
        agentid = request.form['agentid']
        a_password = request.form['apassword']
        conn = db2_connection()
        stmt2 = f"SELECT * FROM agent WHERE agentid='{agentid}' and apassword='{a_password}'"
        temp = ibm_db.exec_immediate(conn, stmt2)
        user = ibm_db.fetch_tuple(temp)

        if user:
            session['agentid'] = agentid
            session['agentname'] = user[1]
            conn = db2_connection()
            stmt = f"SELECT * FROM complaint WHERE agentid='{agentid}' and status='pending';"
            query = ibm_db.exec_immediate(conn, stmt)
            complaints = []
            while True:
                temp = ibm_db.fetch_tuple(query)
                if (temp != False):
                    complaints.append(temp)
                else:
                    break

            return render_template('agent-dashboard.html', agentname = f"{user[1]}", complaints = complaints)
        else:
            message = "Wrong agentid and password!"
            return render_template('agent-login.html', message = message)

    return render_template('agent-login.html', message = message)




# agent dashboard

@app.route("/agentdashboard")
def agentdashboard():
    if session.get('agentid') != None:
        agentid = session.get('agentid')
        agentname = session.get('agentname')
        conn = db2_connection()
        stmt = f"SELECT * FROM complaint WHERE agentid='{agentid}' and status='pending';"
        query = ibm_db.exec_immediate(conn, stmt)
        complaints = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                complaints.append(temp)
            else:
                break
        print(complaints)

        return render_template("agent-dashboard.html" , agentname = agentname, complaints = complaints)
    else:
        return render_template("agent-login.html", message = "session timed out:( please login again!")


@app.route('/agentprocess')
```

```
def agentprocess():
    if session.get('agentid') != None:
        agentid = session.get('agentid')
        agentname = session.get('agentname')
        csid = request.args.get('csid')
        cfid = request.args.get('cfid')
        conn = db2_connection()

        # success
        if csid:
            stmt =  f"UPDATE complaint SET status='success' where agentid='{agentid}'and
cno='{csid}';"
            ibm_db.exec_immediate(conn, stmt)

        # failure
        elif cfid:
            stmt1 = f"UPDATE complaint SET status='failure' where agentid='{agentid}' and
cno='{cfid}';"
            ibm_db.exec_immediate(conn, stmt1)

        message = "Work completed!"

        stmt = f"SELECT * FROM complaint WHERE agentid='{agentid}' and status='pending';"
        query = ibm_db.exec_immediate(conn, stmt)
        complaints = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                complaints.append(temp)
            else:
                break

        return render_template("agent-dashboard.html", agentname = agentname, complaints =
complaints, message = message )
    else:
        return render_template("agent-login.html", message = "session timed out:( please login
again!")



# agent history
```

```
@app.route("/agenthistory")
def agenthistory():
    if session.get('agentid') != None:
        conn = db2_connection()

        agentid = session.get('agentid')
        agentname = session.get('agentname')

        stmt = f"SELECT * FROM complaint WHERE agentid='{agentid}';"
        query = ibm_db.exec_immediate(conn, stmt)
        complaints = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                complaints.append(temp)
            else:
                break
        return render_template("agent-history.html", agentname = agentname, complaints =
complaints)
    else:
        return render_template("agent-login.html", message = "session timed out:( please login
again!")




@app.route("/agentprofile")
def agentprofile():
    if session.get('agentid') != None:
        aid = session.get('agentid')
        print(aid)
        conn = db2_connection()
        stmt = f"SELECT * FROM agent where agentid='{aid}';"
        query = ibm_db.exec_immediate(conn, stmt)
        agents = []
        while True:
            temp = ibm_db.fetch_tuple(query)
            if (temp != False):
                agents.append(temp)
            else:
                break
        print(agents)
        return render_template("agent-profile.html", agents = agents)
    return render_template("agent-login.html", message="session timed out:( please login again!")




# agent logout

@app.route("/agentlogout")
def agentlogout():
    del session['agentid']
    return render_template("agent-login.html")




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