**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** |  |
|  | 1.1Project 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**

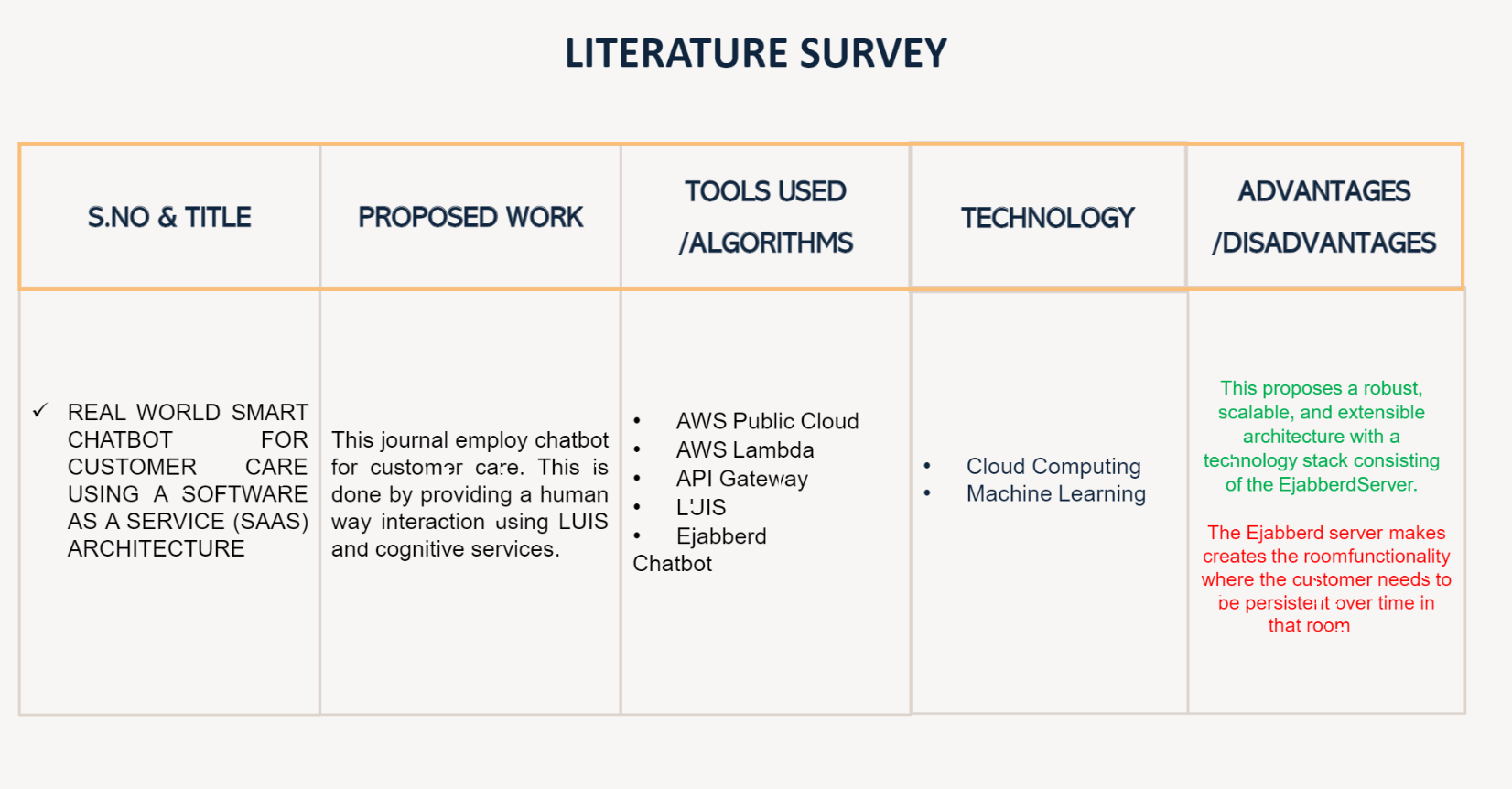
Customer is that the center of attention of each business. The terrible existence of business depends on client satisfaction. Client expects high-quality services, even willing to pay a premium for higher service. From a client perspective, smart service quality ends up in semipermanent client relationships 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**



**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**

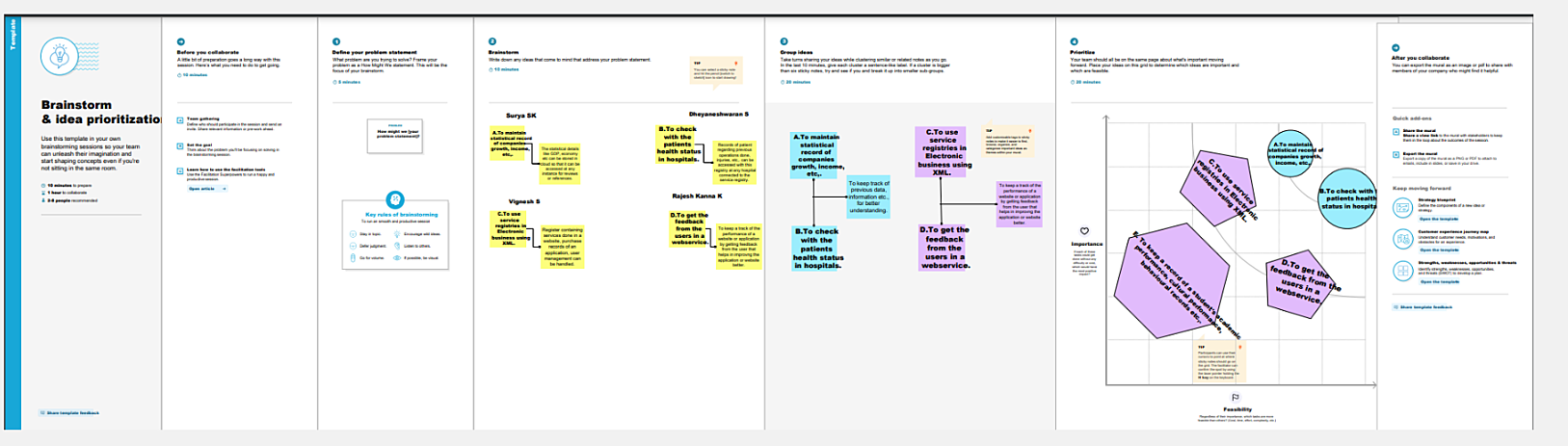
An empathy map is a simple, easy-to-digest visual that captures knowledge about a user’s behaviorsand attitudes.

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



**3.2 Ideation & Brainstorming**



**3.3 Proposed Solution**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Parameter** | **Description** |
|  | Problem Statement(Problem to be solved | To solve customer issues using CloudApplication Development. |
|  | Idea / Solution description | Assigned Agent routing can be solved by directlyrouting to the specific agent about the issue 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) | 1. Key Partners are Third-party applications, agents,and customers. 2. Activities held as Customer Service,System Maintenance. 3. Key Resources support Engineers,Multi-channel. 4. 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 as possible. An environment where they will be able to spend less time on gruntwork and more time on actually resolving critical customer issues. |

**3.4 Problem Solution fit**



**4. REQUIREMENT ANALYSIS**

**4.1 Functional requirement**

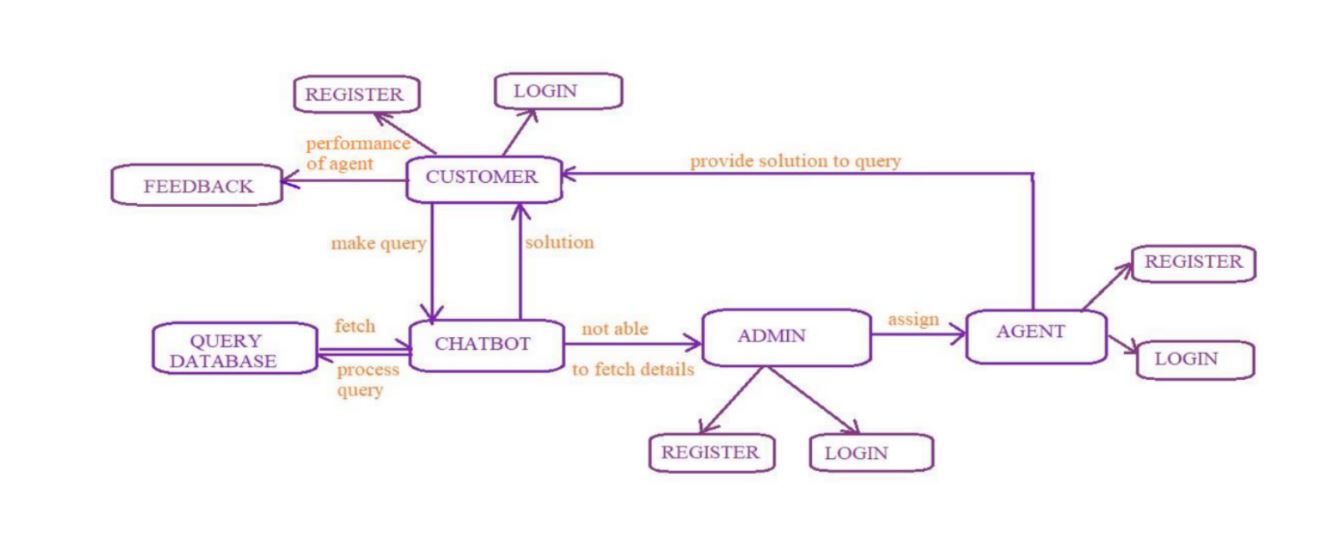
|  |  |  |
| --- | --- | --- |
| FR No | Functional Requirement(Epic) | Sub Requirement(Story/ Sub-Task) |
| **1** | User Registration | Registration through Form Registration through Gmail Registration through Google |
| **2** | User Confirmation | Confirmation via Email Confirmation via OTP |
| **3** | 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**

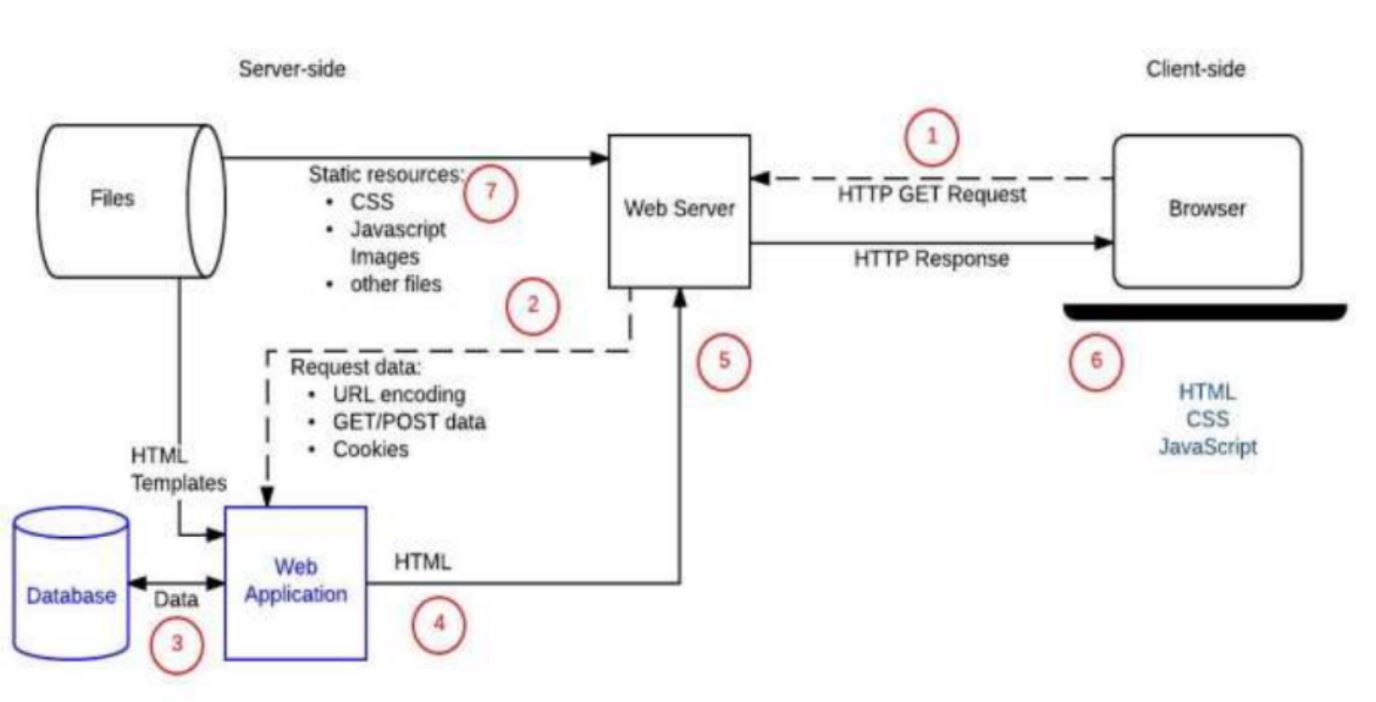
|  |  |  |
| --- | --- | --- |
| FR No | Non-Functional Requirement | Description |
|  | Usability | To provide the solution to the problem |
|  | Security | Track of login authentication |
|  | Reliability | Tracking of decade status through email |
|  | Performance | Effective development of web application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

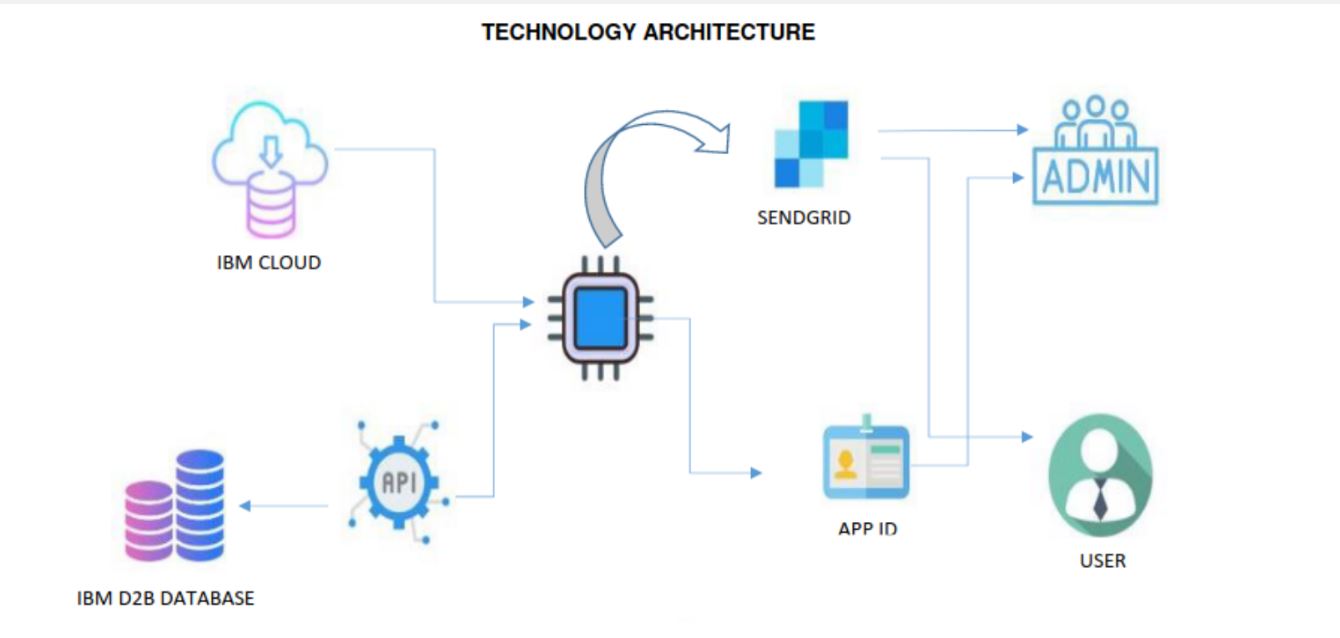
**5. PROJECT DESIGN**

**5.1 Data Flow Diagrams**

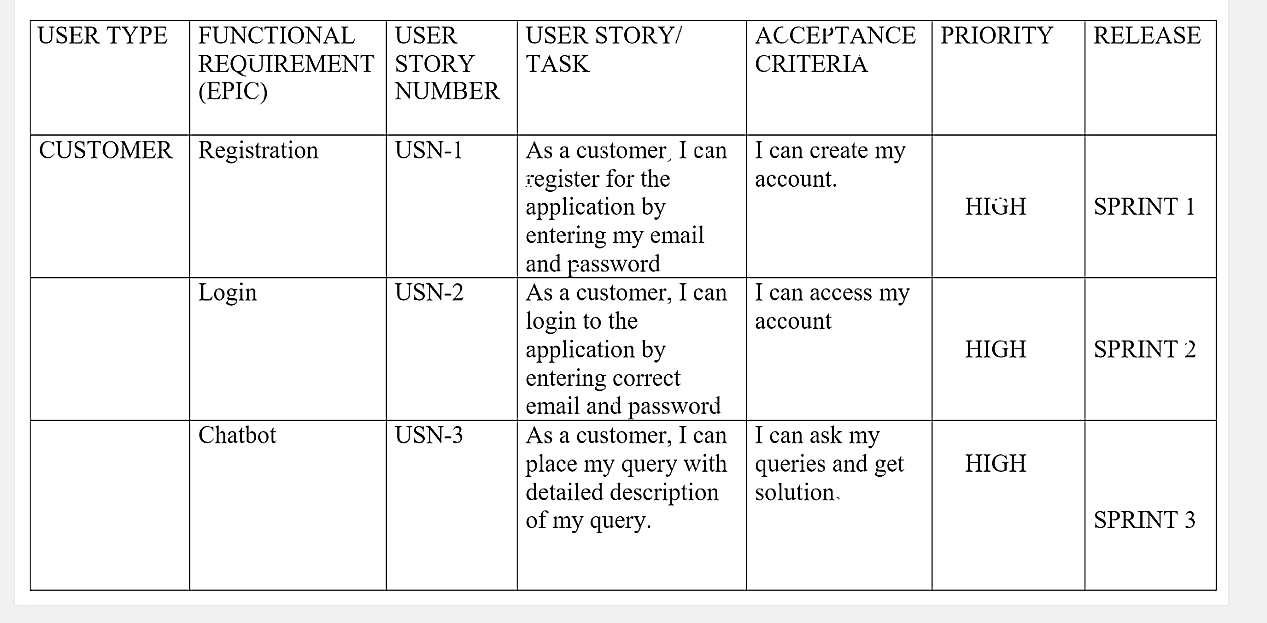


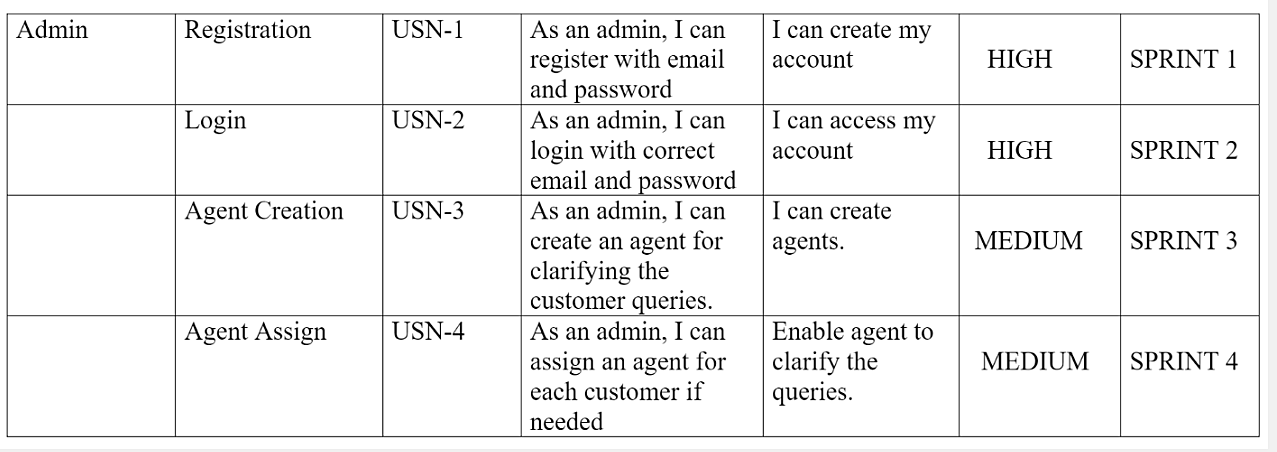
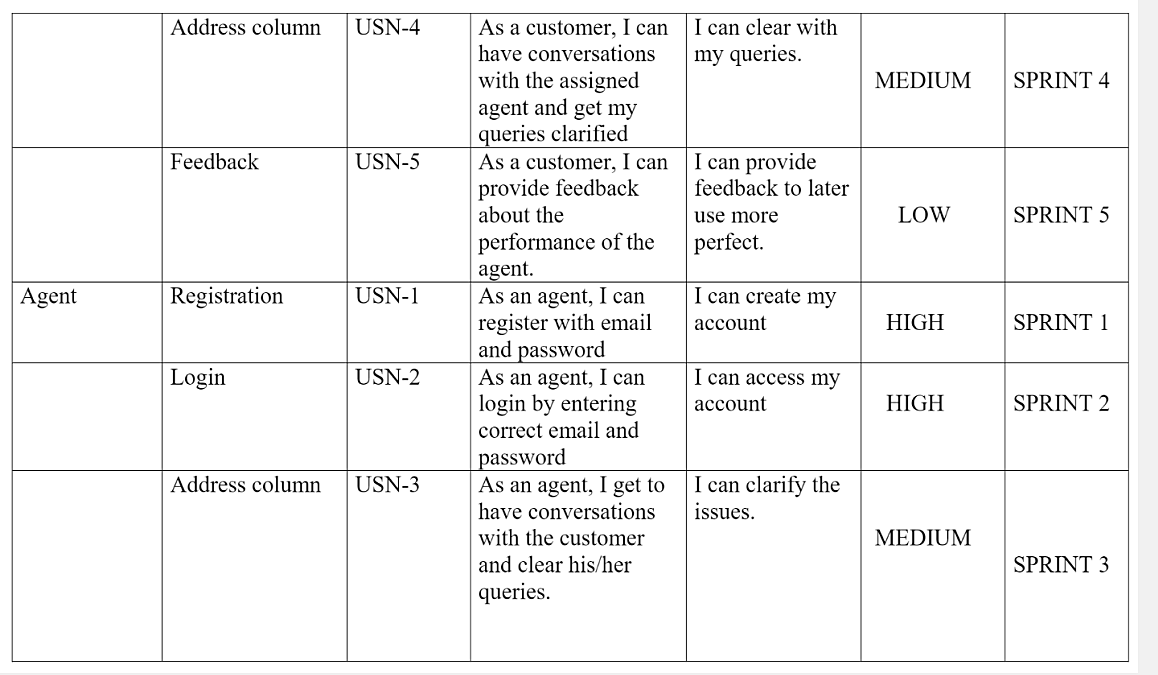
**5.2 Solution & Technical Architecture**





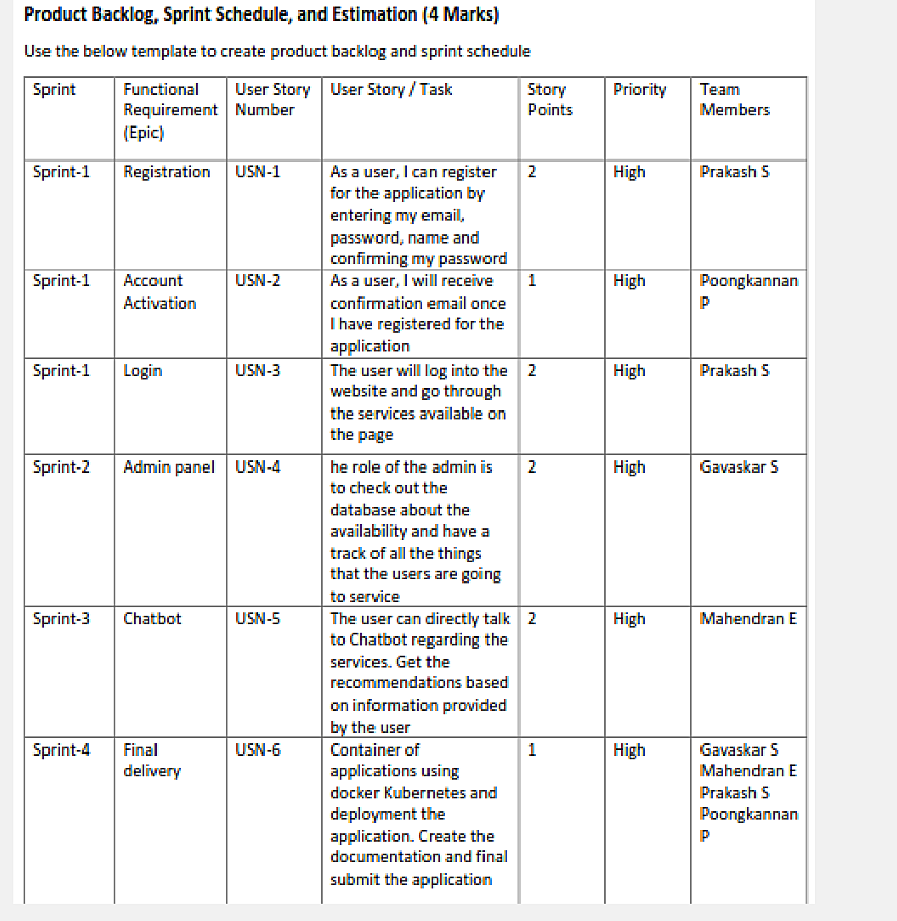
**5.3 User Stories**



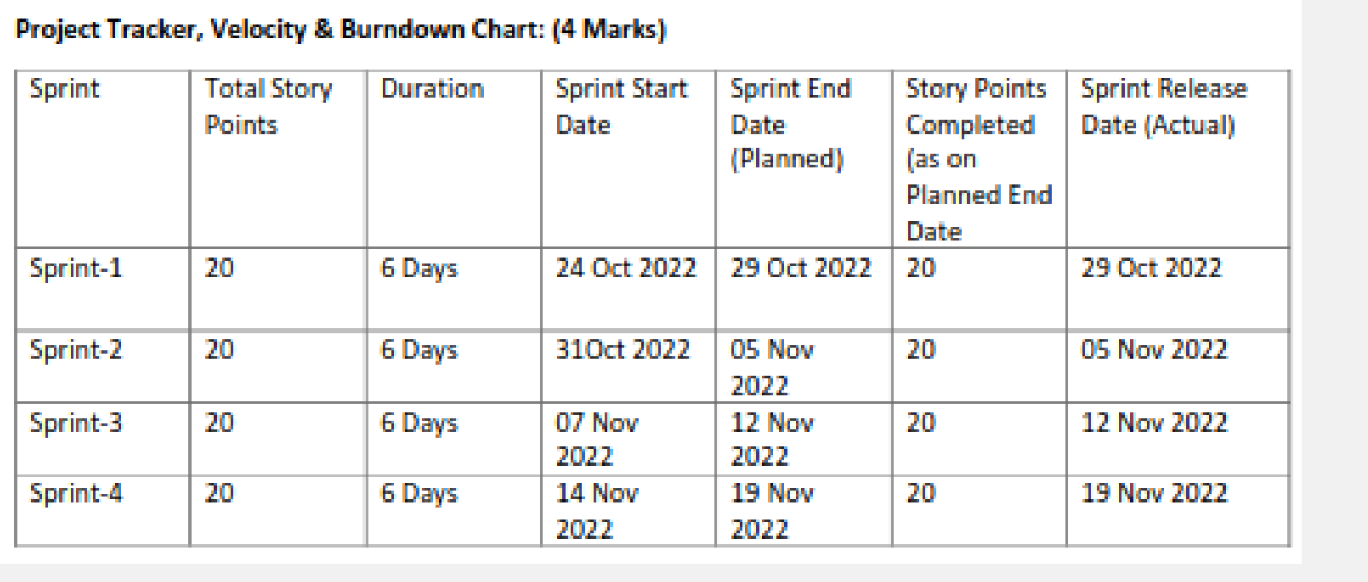


**6.PROJECT PLANNING & SCEDULING**

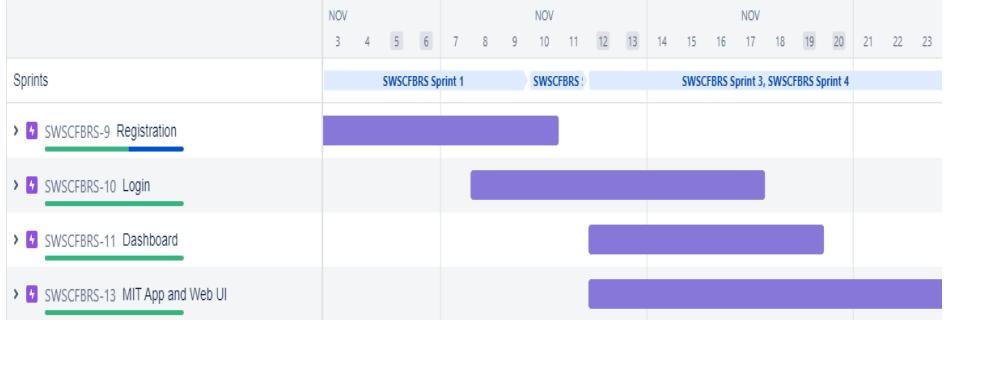
**6.1 SPRINT PLANNING & ESTIMATION**



**6.2 SPRINT DELIVERY SCHEDULE**



**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

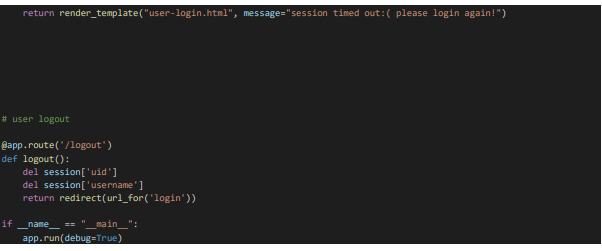
### 3. Universal Agent Management Solution

### 4.Internal Communication Mechanism

**7.3 Database Schema**

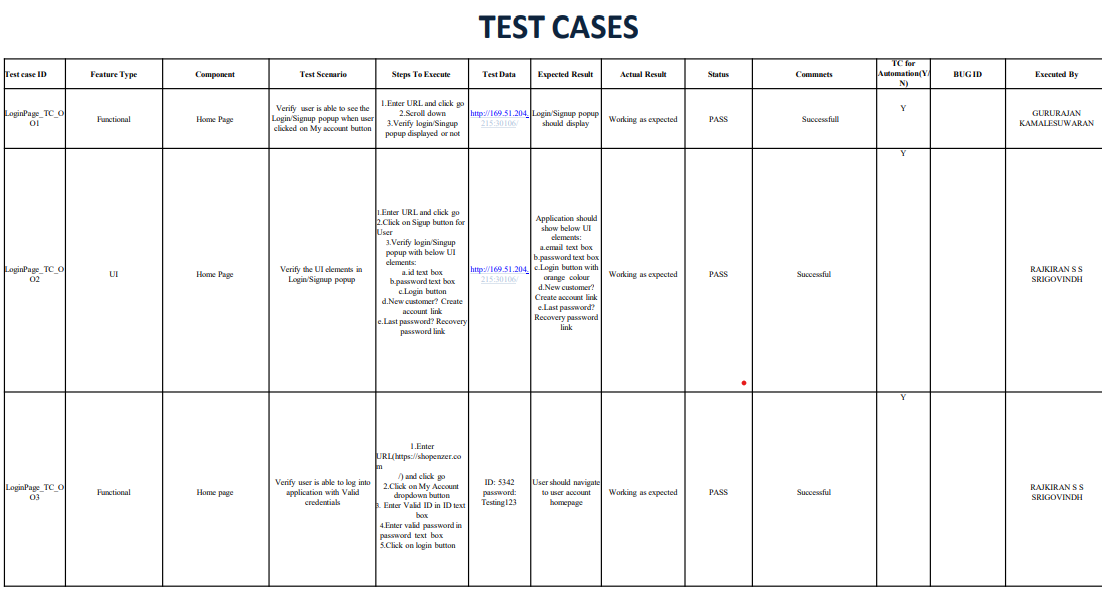


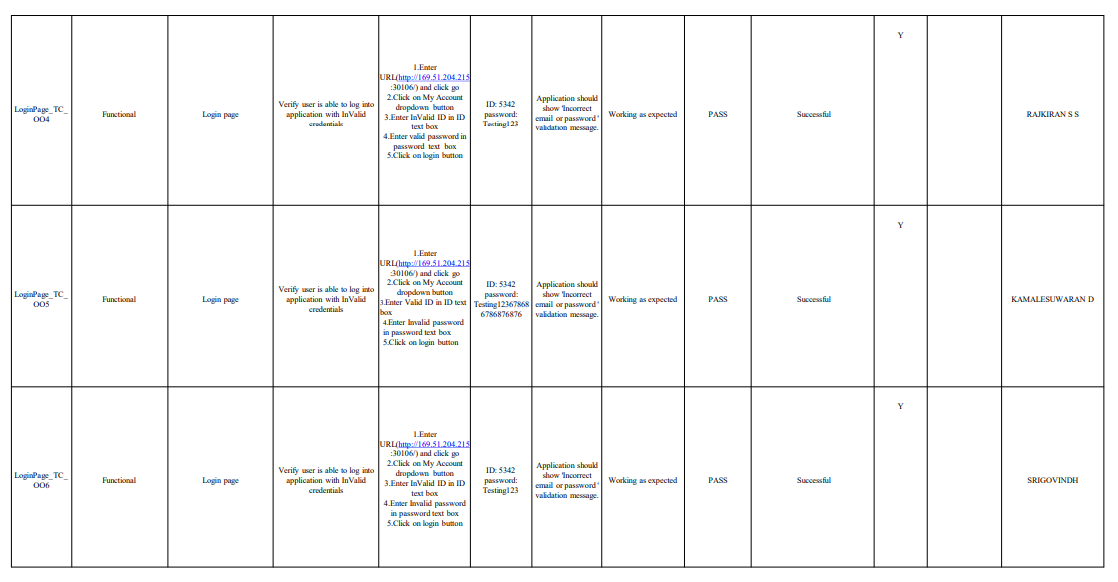




**8.TESTING**

**8.1 Test cases**





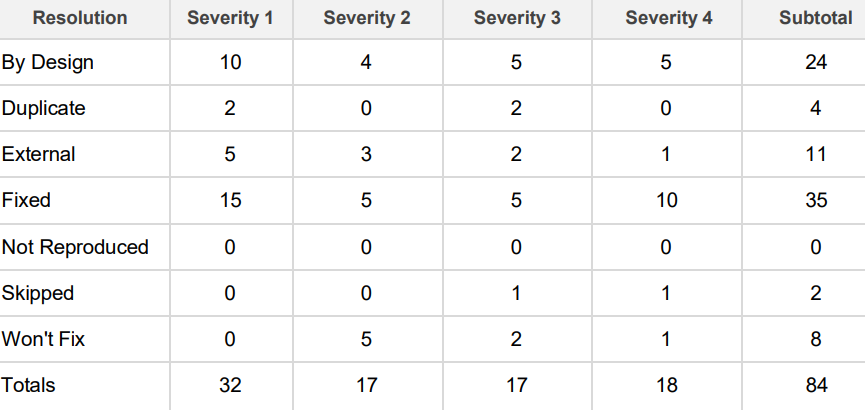
**8.2 User Acceptance Testing**

**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).

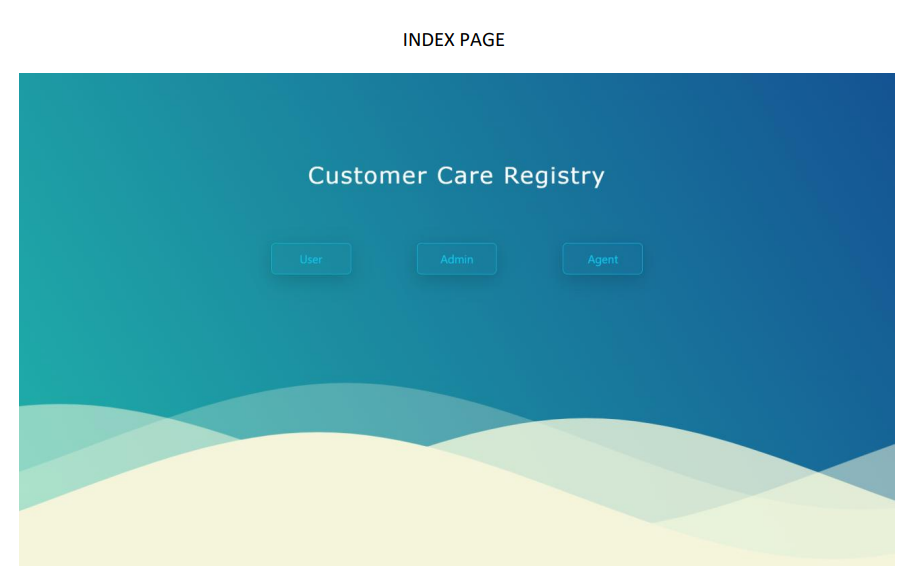
**2. Defect Analysis**

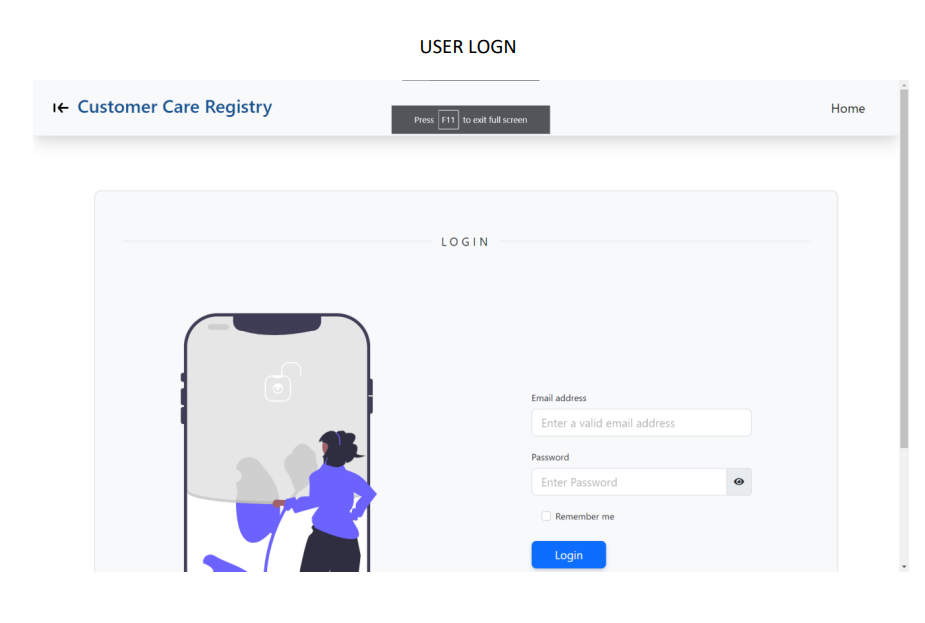
This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

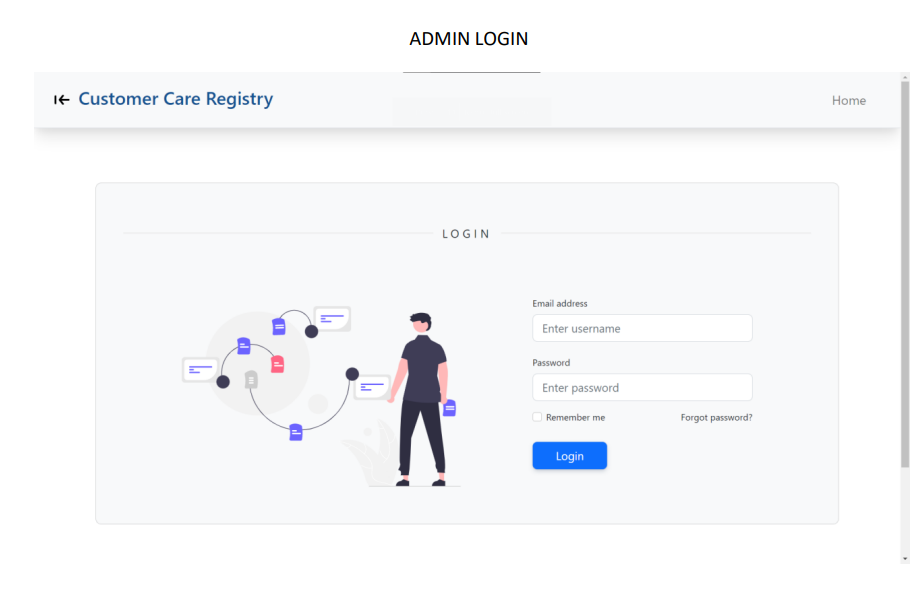


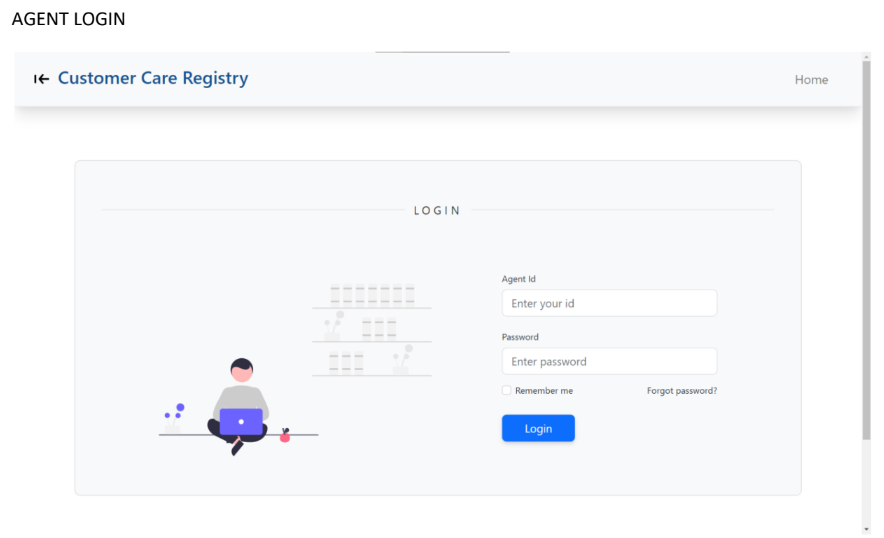
**9.Results**

**9.1 Performance metrics**









**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)