IBM NALAIYATHIRAN PROJECT REPORT

Domain: Retails and E-Commerce (R&E)

Title: Customer Care Registry

Submitted by

Team ID: PNT2022TMID14438

Team Members:

Team Leader: SARAN MK

Team Member: VP PRAVIN

Team Member: UNNIKRISHNAN J

Team member: VARUN T

Industry Mentor(s) Name : Vasudeva Hanush

Faculty Mentor(s) Name : Naresh Sammeta

Project Overview

1. INTRODUCTION

A comprehensive online Client Care Solution is used to manage customer interactions and complaints with Service Providers via phone or e-mail. The system must be able to integrate with any service provider from any field or sector, including banking, telecommunications, insurance, etc. The provision of service to consumers, commonly referred to as client service, has varying significance depending on the product, business, and domain. When making a purchase of a service rather than a product, customer service is frequently of greater importance. People or Sales & Service Representatives may offer customer service. The customer value proposition of a business typically includes excellent customer service.

1.1 PROJECT OVERVIEW

The Customer Service Desk project is online. The provision of service to customers is known as customer service or client service. The importance varies depending on the product, industry, and domain. When information relates to a service rather than a customer, customer service is frequently more crucial. A service representative might offer customer service. The customer value proposition of a business typically includes excellent customer service. Software like Flask, Docker, SendGrid, and IBM Watson are used to implement this.

1.2 PURPOSE

The project's goal is to create customer engagement, address customer issues, and offer a helpful service. It is an essential component of all businesses.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

The information is now saved in disc drives in the form of excel sheets in a semi-automated approach. Only the mailing feature is used to share information with Volunteers, Group members, etc. In this system, information maintenance and storage are increasingly crucial. It is a hard task to monitor the members' activities and the development of the work here. This system is unable to offer information exchange every day of the week.

2.2 REFERENCES

a) In this article, a Chabot from the AWS cloud is deployed for customer care. Real world smart Chabot for customer care using SaaS architecture. This is done to offer cognitive and LUIS services to humans.

b) Chat bots for customer service are used in this paper in place of human customer service representatives. It makes decisions and offers services using AI.

c) Client service chatbot - In this paper, the customer gives the chatbot the information it needs based on the information it gives the customer service.

d) A clever cloud-based customer relationship management system that uses adjustable pricing to maintain customers. This essay analyses historical patterns to suggest consumer behavior that might be used for marketing.

2.3 PROBLEM STATEMENT DEFINITION

A problem statement is a concise description of the problem or issues a project seeks to address. The problem statement identifies the current state, the desired future state and any gaps between the two. A problem statement is an important communication tool that can help ensure everyone working on a project knows what the problem they need to address is and why the project is important.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a straightforward, simple-to-understand picture that summarizes information about a user's actions and views. It is a helpful tool that enables teams to comprehend their users more fully. It's important to comprehend both the actual issue and the person who is experiencing it in order to develop a workable solution. Participants learn to think about situations from the user's perspective, including goals and challenges, through the exercise of creating the map.

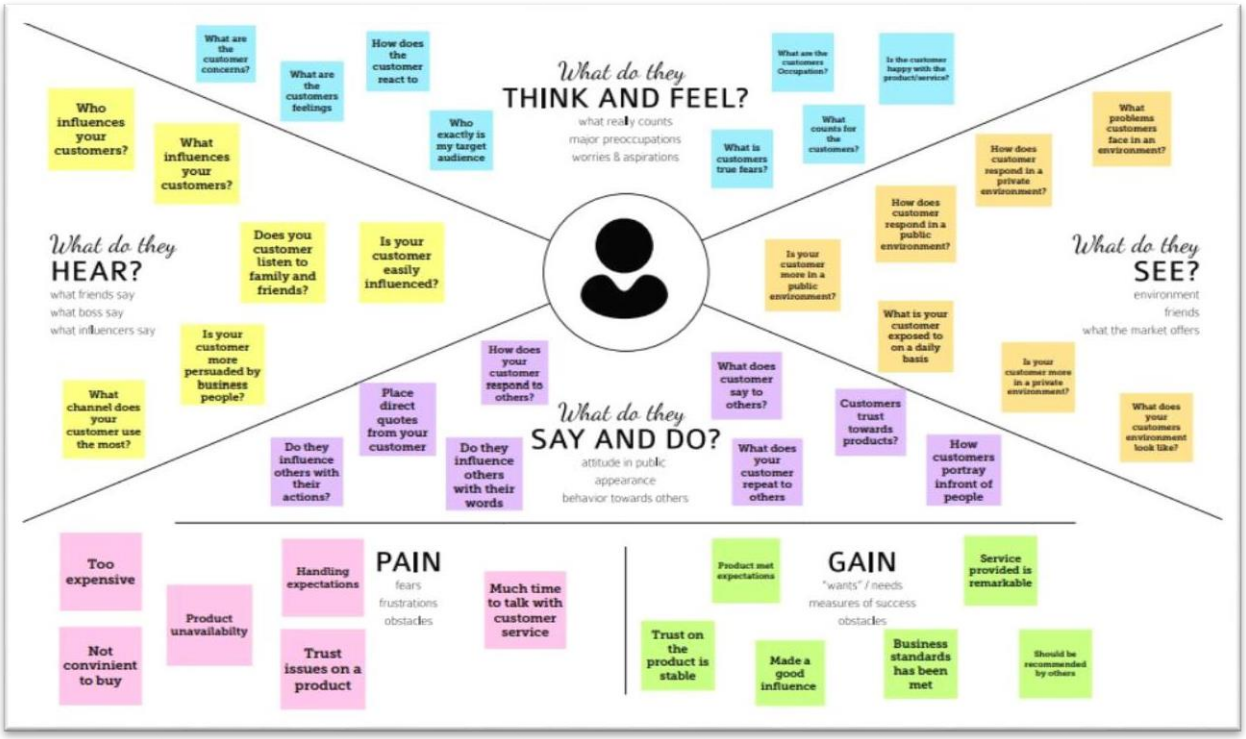


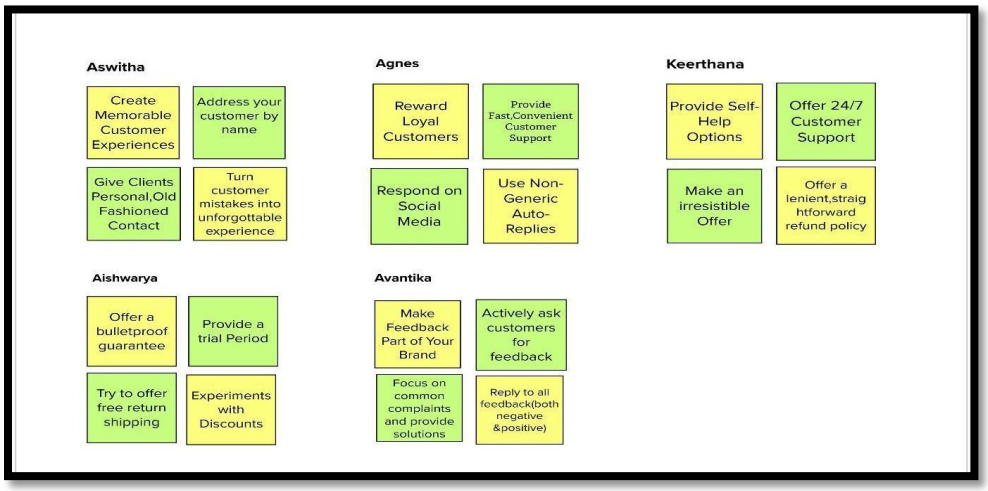
Fig 3.1: Empathy Map Canvas

3.2 IDEATION AND BRAINSTORMING

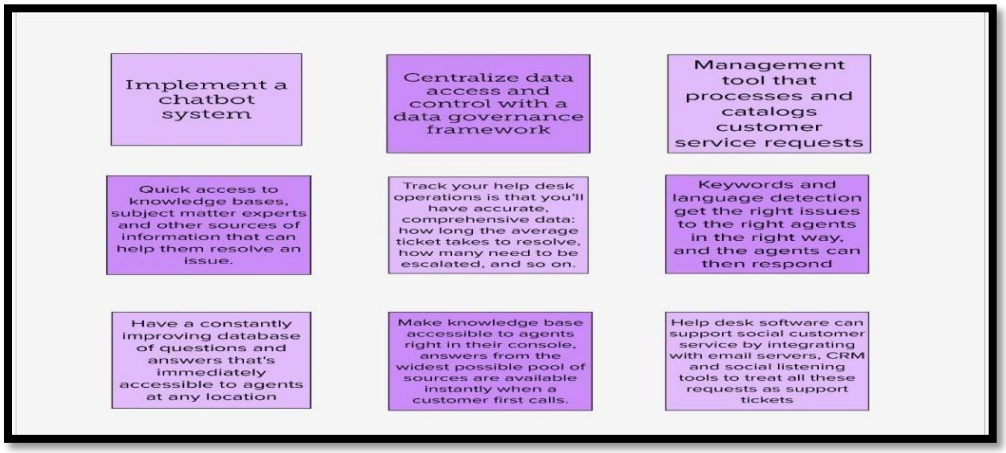
1. What issues are you attempting to address? Describe the issues.

2. How might we address the issue? Which does the customer raise?

Brainstorming:



Group Ideas:



3.3 PROPOSED SOLUTION

Allotted By sending the email directly to a specific agent about the problem, agent routing can be fixed. automated ticket closure by cloud database sync. The customer's status display may include their tickets. The platform that will enable the customer specialist to be effective is what the customer care service aims to give. And it takes less time to find the answer.

4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIn
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	Customer Query	Access through email and chatbot from the chosen website
FR-4	Database	preserving the modelled item
FR-5	Feedback	Customer's Feedback
FR-6	E-Mail	Login alertness

NON-FUNCTIONAL REQUIREMENTS

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	To offer a remedy for the issue, user friendly.
NFR-2	Security	Logging and authentication history
NFR-3	Reliability	Tracking the status of the decade via email
NFR-4	Performance	responsive and adaptable
NFR-5	Availability	24/7 Support
NFR-6	Scalability	Scalability of agents according to consumer volume

5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAM

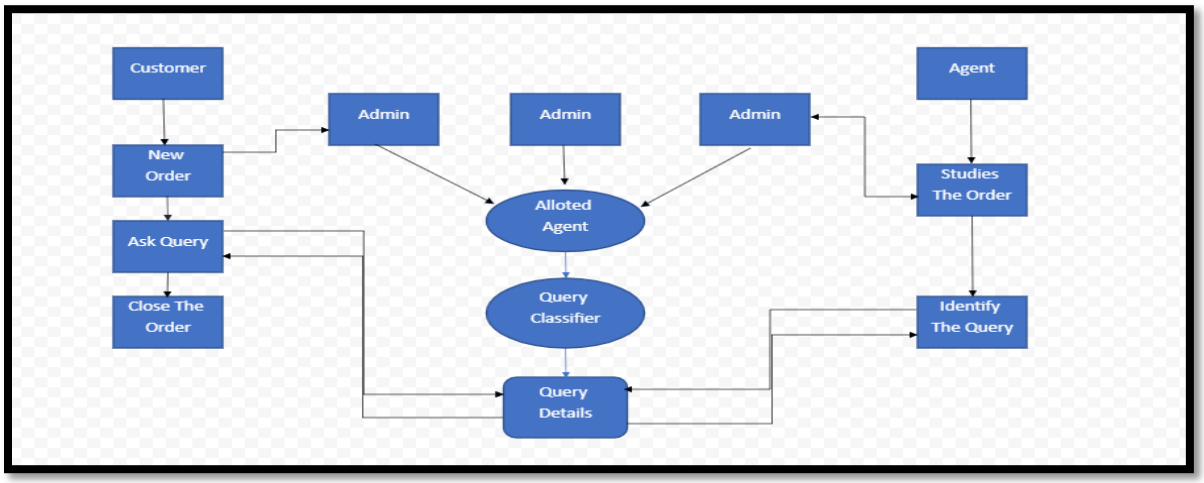


Fig 5.1: Data Flow Diagram

5.2 SOLUTION AND TECHNICAL ARCHITECTURE

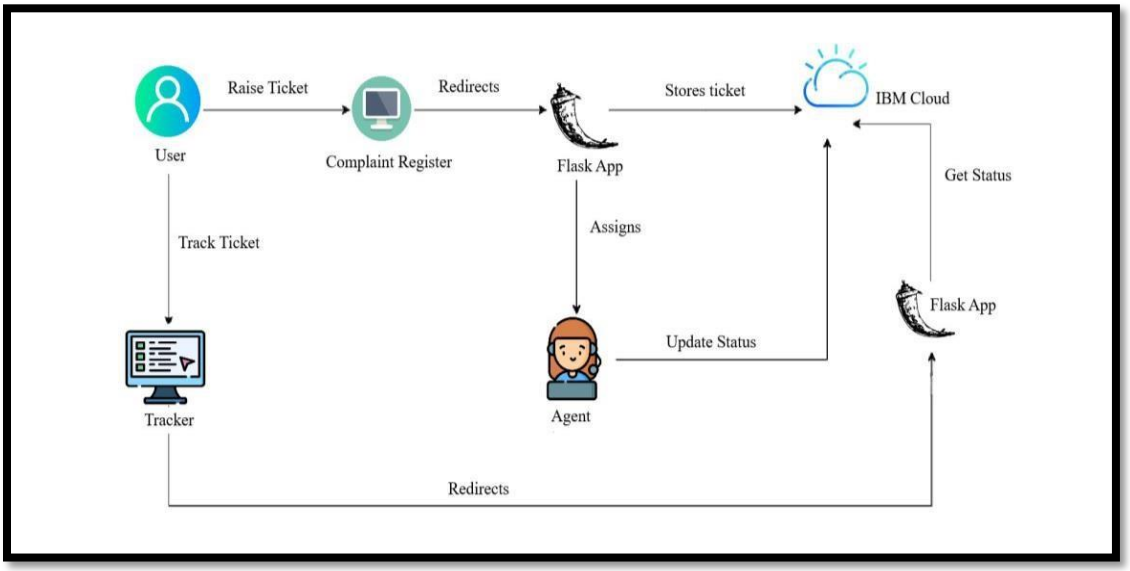


Fig 5.2: Solution and Technical Architecture

6. PROJECT PLANNING & SCHEDULING

6.1 SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	2	High	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
Sprint-1		USN-2	As a user, I will receive confirmation email once I have registered for the application	1	High	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
Sprint-2		USN-3	As a user, I can register for the application through Facebook	2	Low	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
Sprint-1		USN-4	As a user, I can register for the application through Gmail	2	Medium	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
Sprint-1	Login	USN-5	As a user, I can log into the application by entering email & password	1	High	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
	Dashboard	USN-6	Create a model set that contains those models, then assign it to a role.		High	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.
Sprint-3		USN-7	Open, public access, User- aut,1enticated access, Employee- restricted access		High	Aswitha,Aishwarya, Agnes Sharon,Avantika, Keerthana.

6.2 SPRINT DELIVERY SCHEDULE

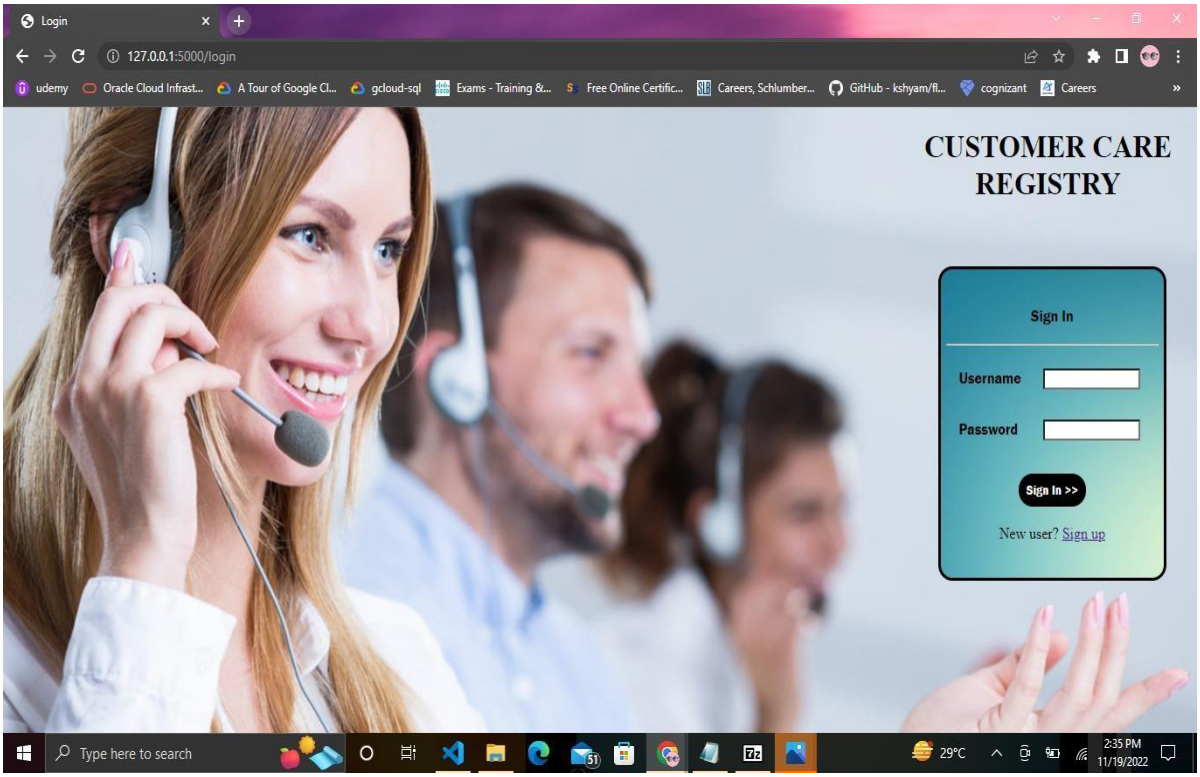
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022		
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022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

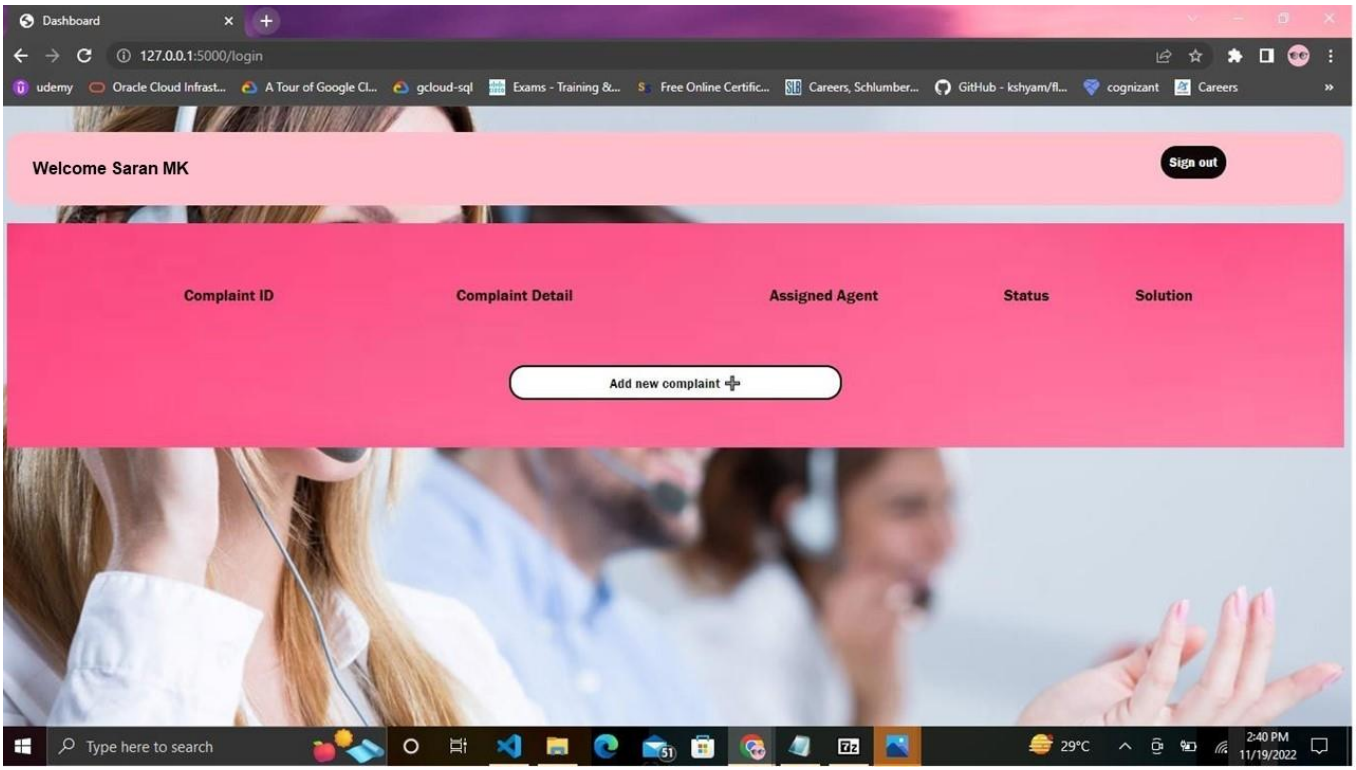
6.3 REPORTS FROM JIRA

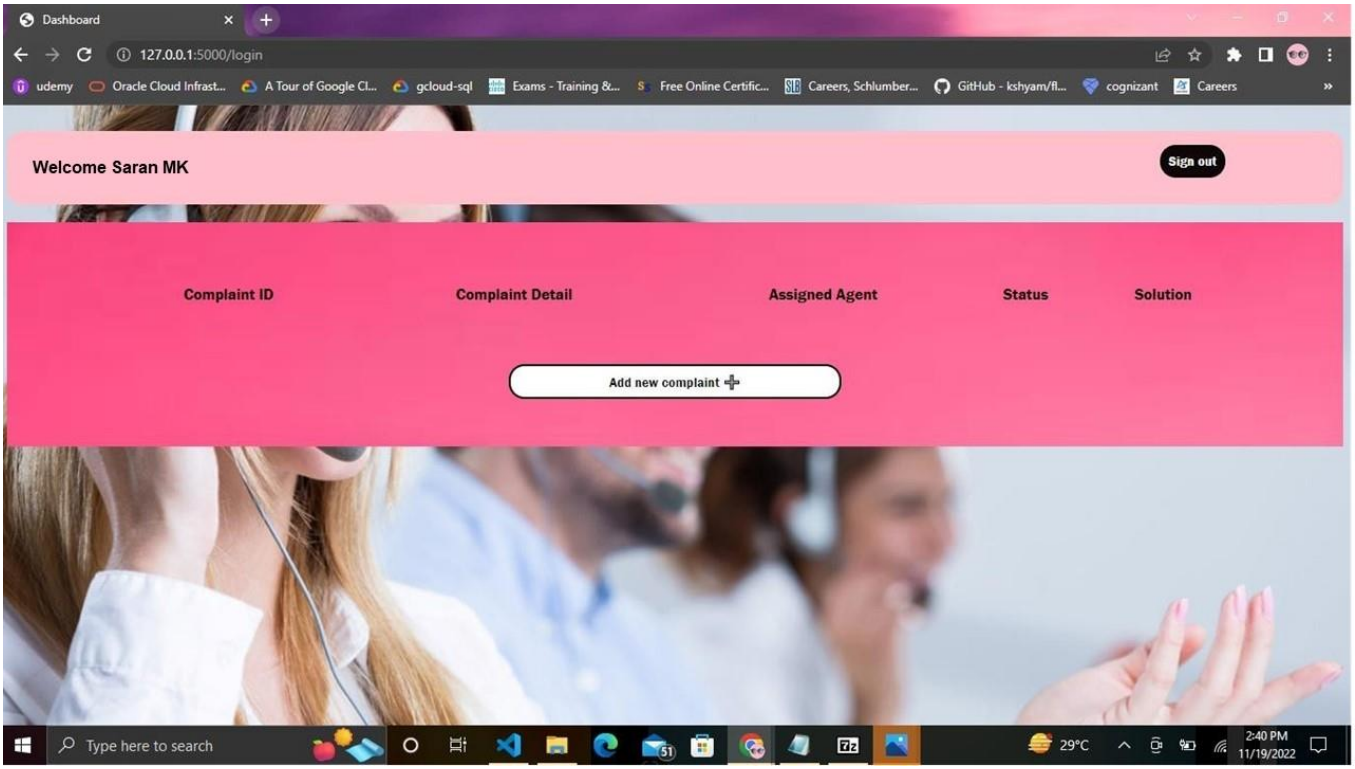
~~CCR-3~~	The user will login into the website and go throug...	DONE
~~CCR-4~~	The role of the agent is to check out the complaint...	DONE
~~CCR-5~~	The role of the admin is to check out the database...	DONE
~~CCR-6~~	he user can directly talk to Chatbot regarding the ...	DONE
~~CCR-7~~	Container of applications using docker kubernetes...	DONE

7. CODING & SOLUTION

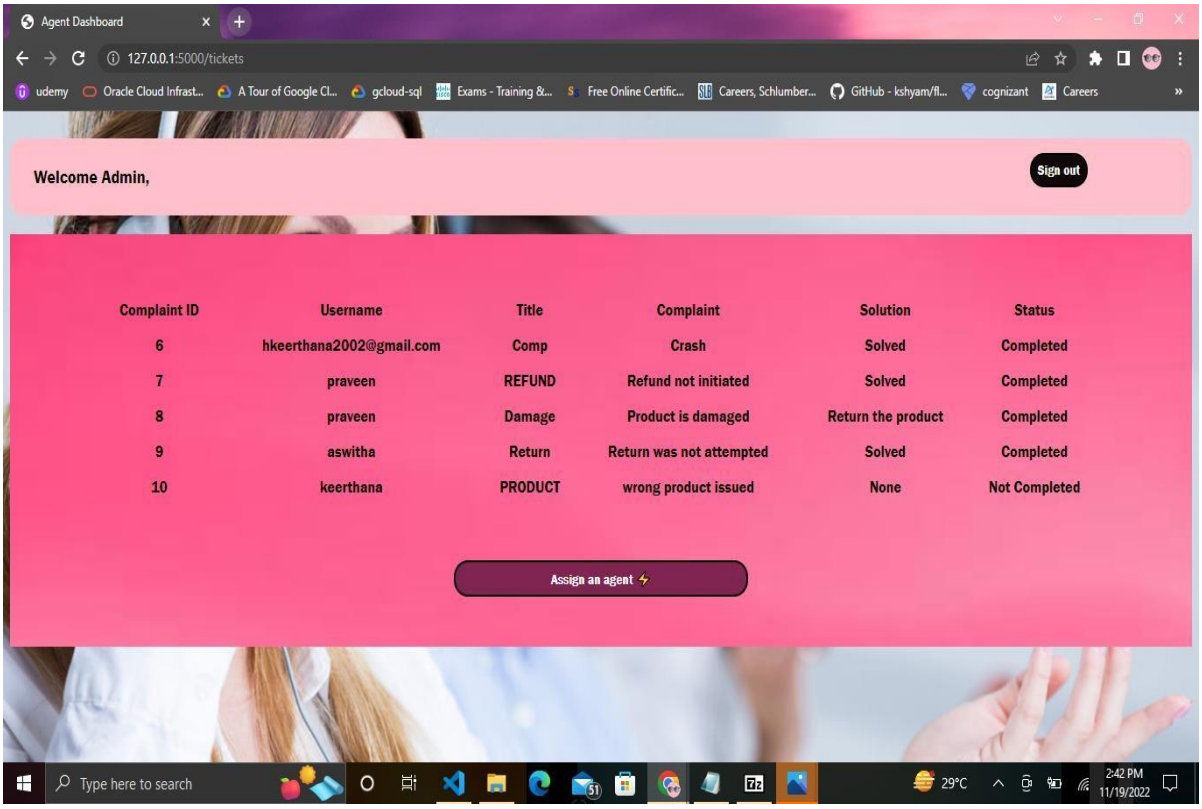
7.1 FEATURE

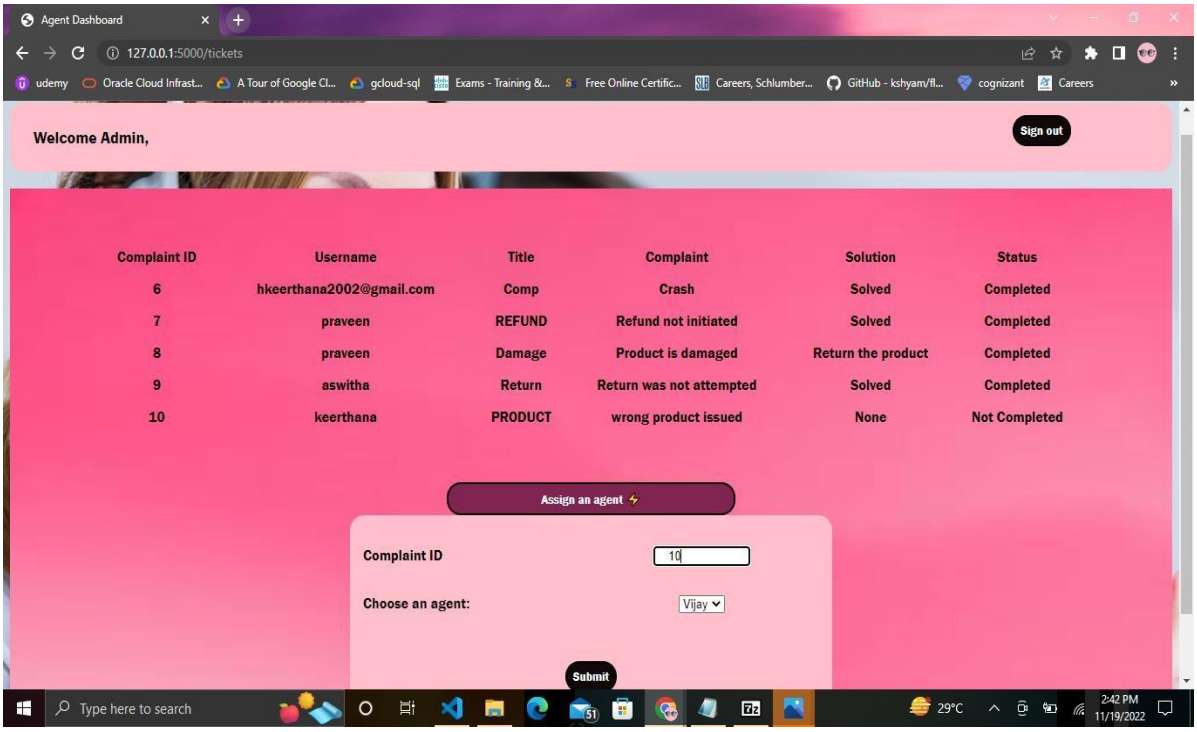


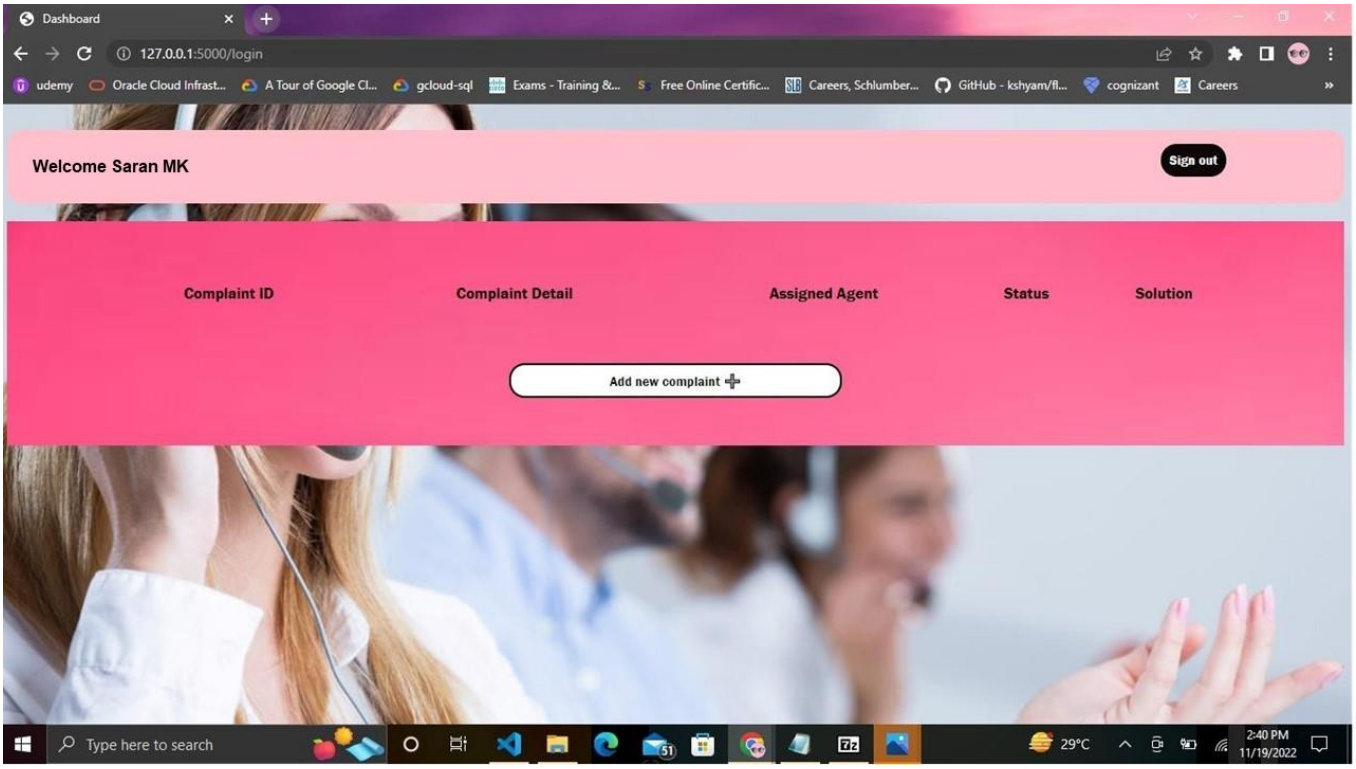



Dashboard
127.0.0.1:5000/login
Welcome Saran MK
Sign out
Complaint ID
Complaint Detail
Assigned Agent
Status
Solution
Add new complaint

7.2 FEATURE

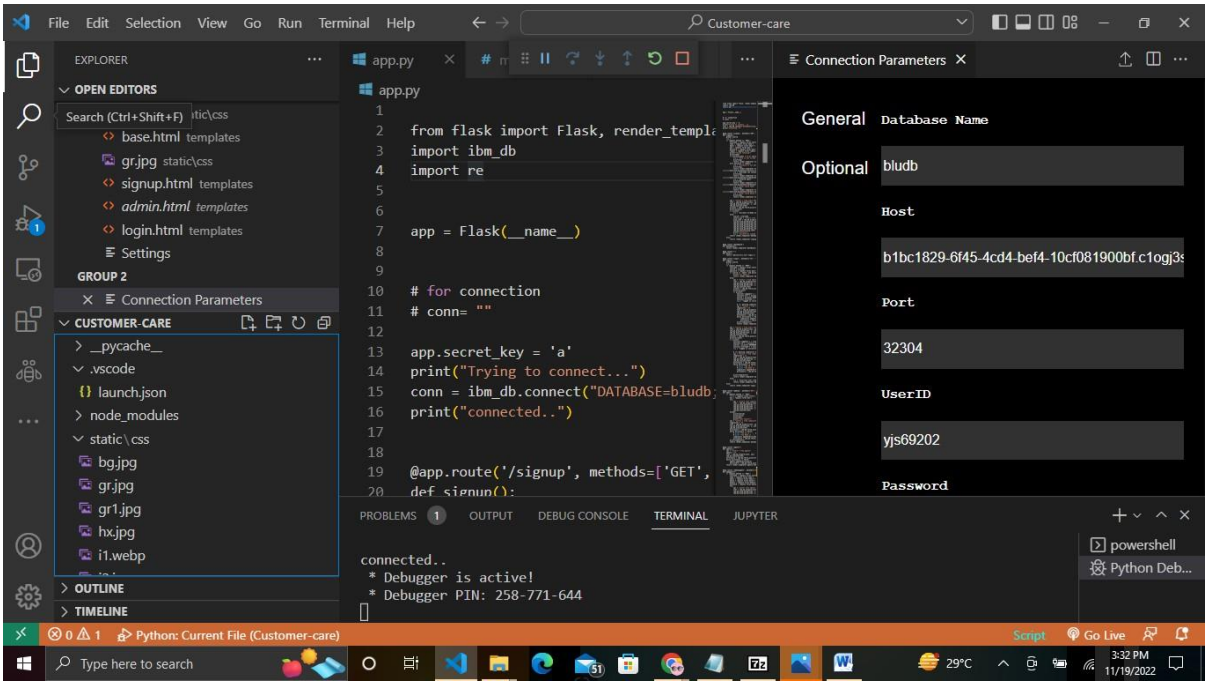

Agent Dashboard
127.0.0.1:5000/tickets
Welcome Admin,
Sign out
Complaint ID | Username | Title | Complaint | Solution | Status
6 | hkeerthana2002@gmail.com | Comp | Crash | Solved | Completed
7 | praveen | REFUND | Refund not initiated | Solved | Completed
8 | praveen | Damage | Product is damaged | Return the product | Completed
9 | aswitha | Return | Return was not attempted | Solved | Completed
10 | keerthana | PRODUCT | wrong product issued | None | Not Completed
Assign an agent


Agent Dashboard
127.0.0.1:5000/tickets
Welcome Admin,
Sign out
Complaint ID
Username
Title
Complaint
Solution
Status
6
hkeerthana2002@gmail.com
Comp
Crash
Solved
Completed
7
praveen
REFUND
Refund not initiated
Solved
Completed
8
praveen
Damage
Product is damaged
Return the product
Completed
9
aswitha
Return
Return was not attempted
Solved
Completed
10
keerthana
PRODUCT
wrong product issued
None
Not Completed
Assign an agent
Complaint ID
10
Choose an agent:
Vijay
Submit
Type here to search
29°C
2:42 PM
11/19/2022

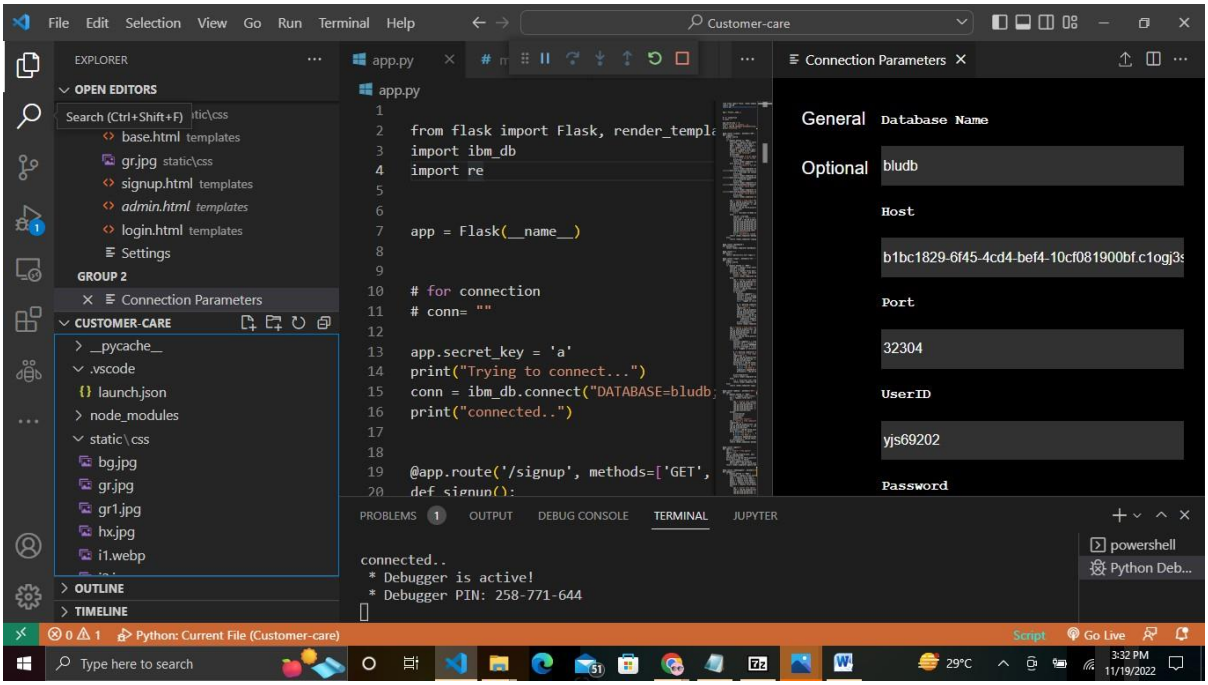

Dashboard
127.0.0.1:5000/login
Welcome Saran MK
Sign out
Complaint ID
Complaint Detail
Assigned Agent
Status
Solution
Add new complaint
Type here to search
29°C
2:40 PM
11/19/2022

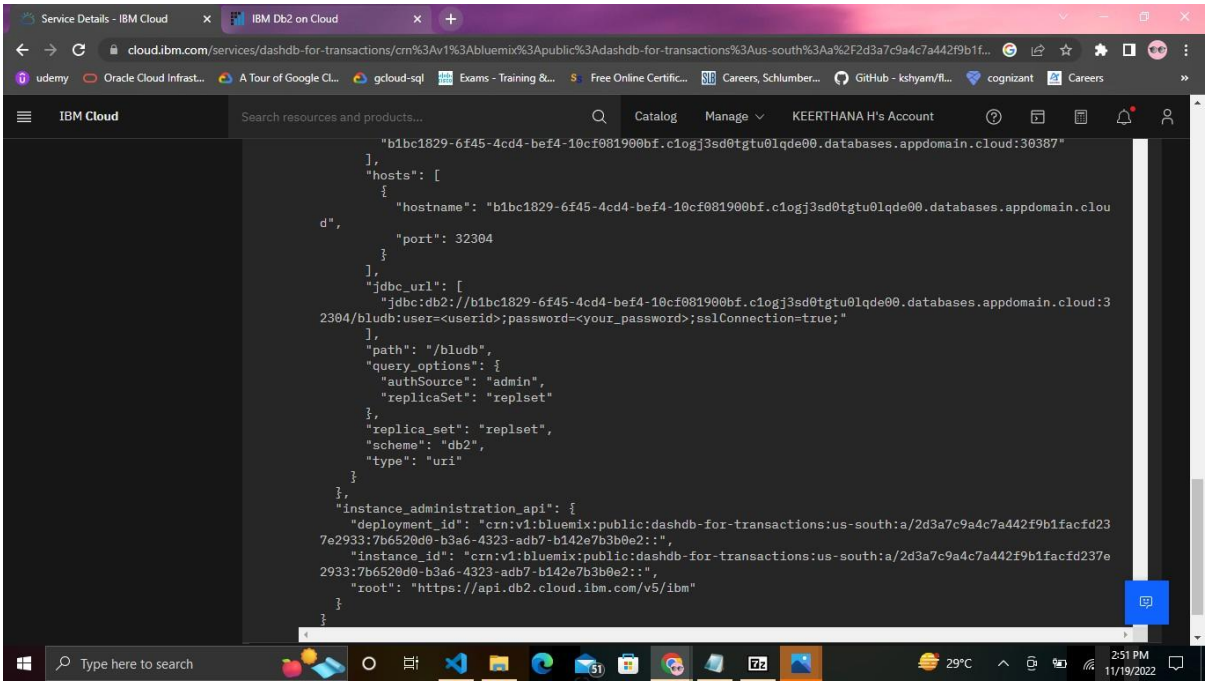
7.3 DATABASE SCHEMA

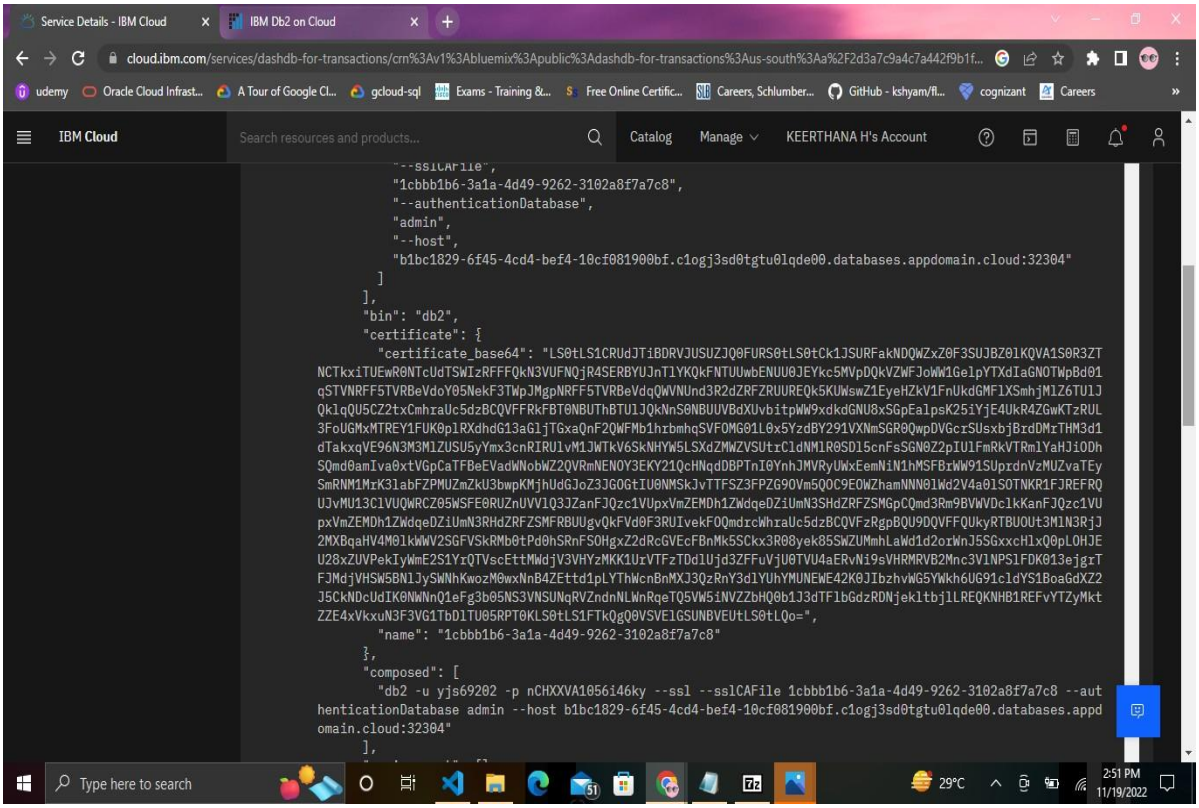
CONNECTING TO DATABASE:



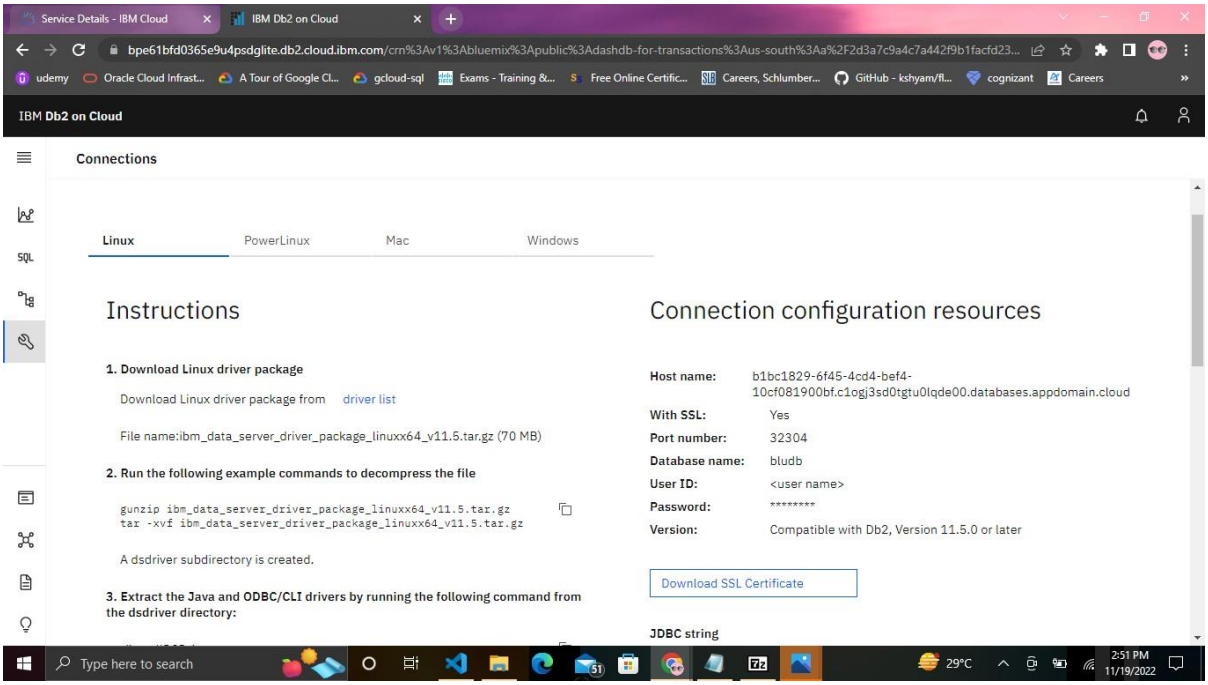
SERVICE CEREDENTIALS:

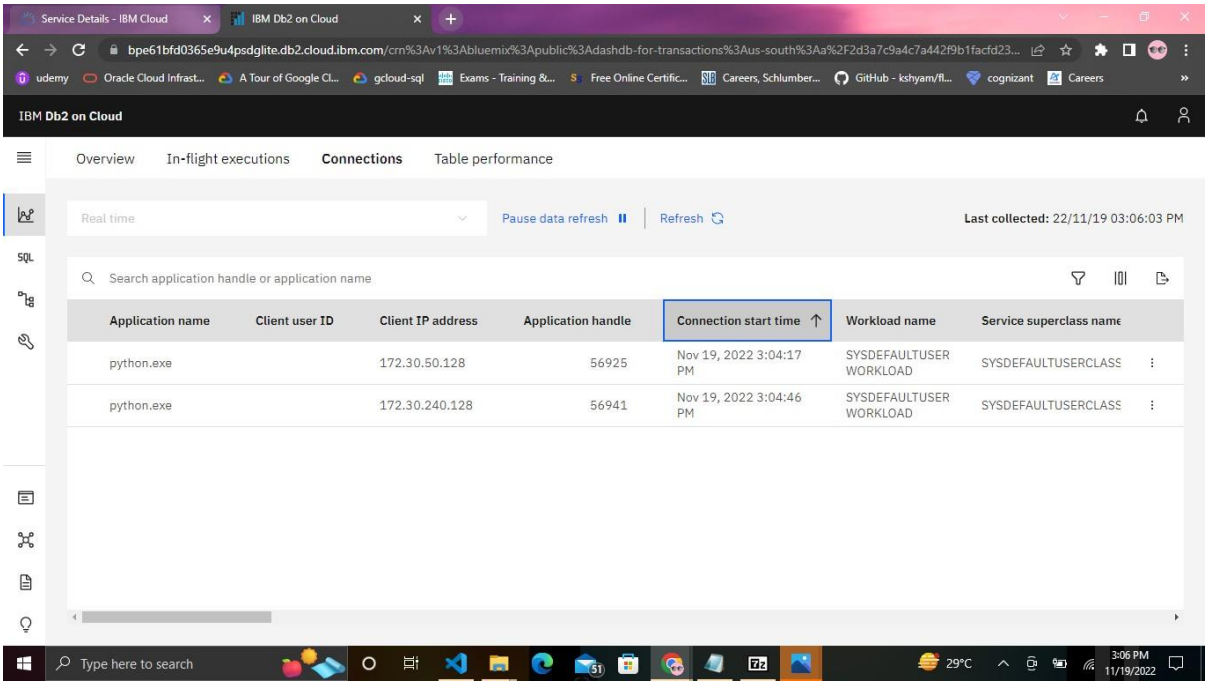




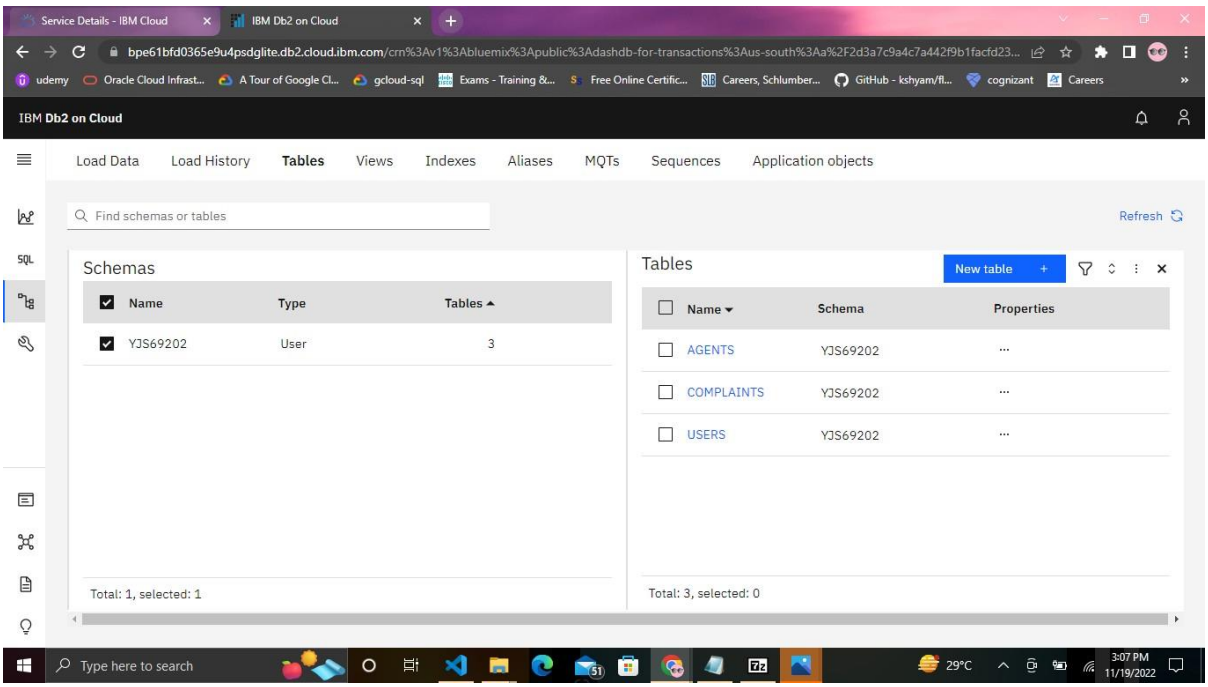


CONNECTIONS:

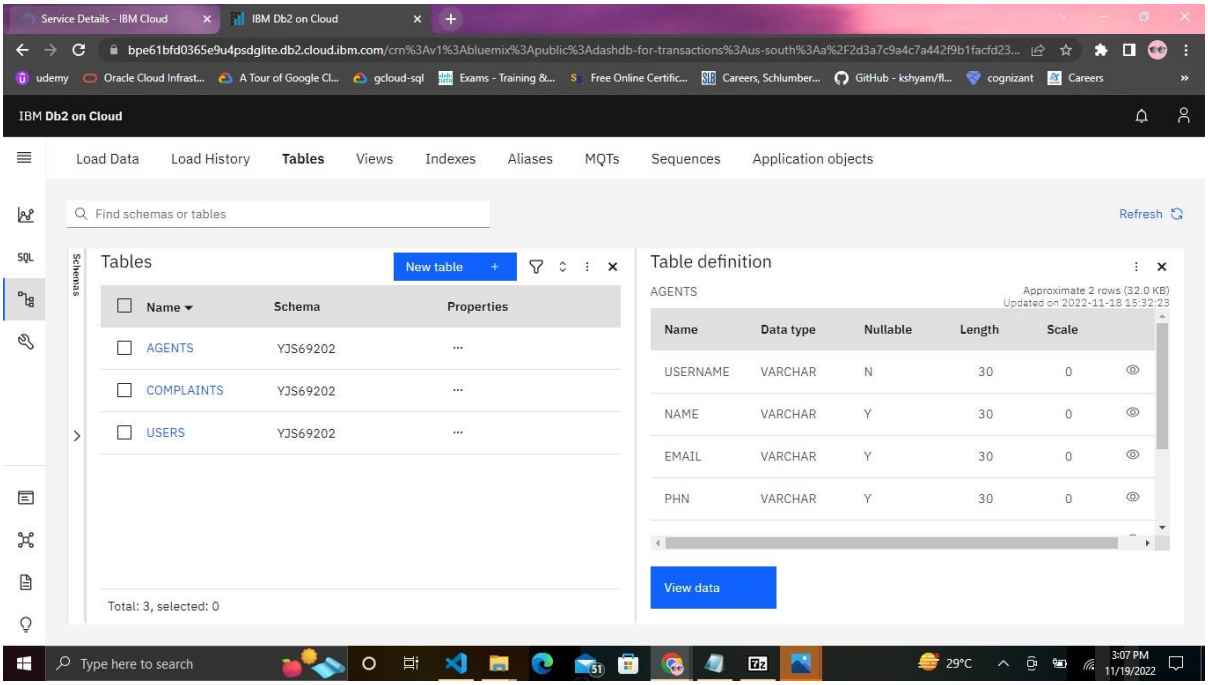



Service Details - IBM Cloud
IBM Db2 on Cloud
IBM Db2 on Cloud
Overview
In-flight executions
Connections
Table performance
Real time
Pause data refresh
Refresh
Last collected: 22/11/19 03:06:03 PM
Search application handle or application name
Application name
Client user ID
Client IP address
Application handle
Connection start time
Workload name
Service superclass name
python.exe
172.30.50.128
56925
Nov 19, 2022 3:04:17 PM
SYSDEFAULTUSER WORKLOAD
SYSDEFAULTUSERCLASS
python.exe
172.30.240.128
56941
Nov 19, 2022 3:04:46 PM
SYSDEFAULTUSER WORKLOAD
SYSDEFAULTUSERCLASS
Type here to search
29°C
3:06 PM
11/19/2022

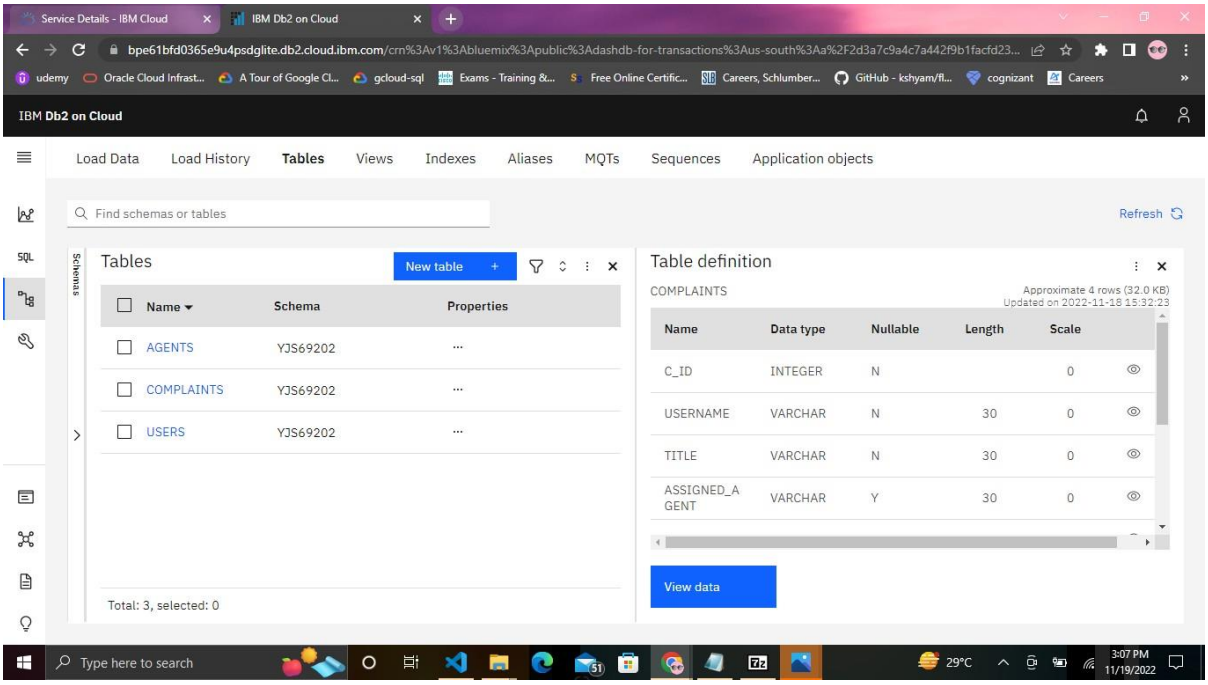
DATA TABLES:


Service Details - IBM Cloud
IBM Db2 on Cloud
IBM Db2 on Cloud
Load Data
Load History
Tables
Views
Indexes
Aliases
MQTs
Sequences
Application objects
Find schemas or tables
Refresh
Schemas
Name
Type
Tables
YJS69202
User
3
Total: 1, selected: 1
Tables
New table
Name
Schema
Properties
AGENTS
YJS69202
COMPLAINTS
YJS69202
USERS
YJS69202
Total: 3, selected: 0
Type here to search
29°C
3:07 PM
11/19/2022

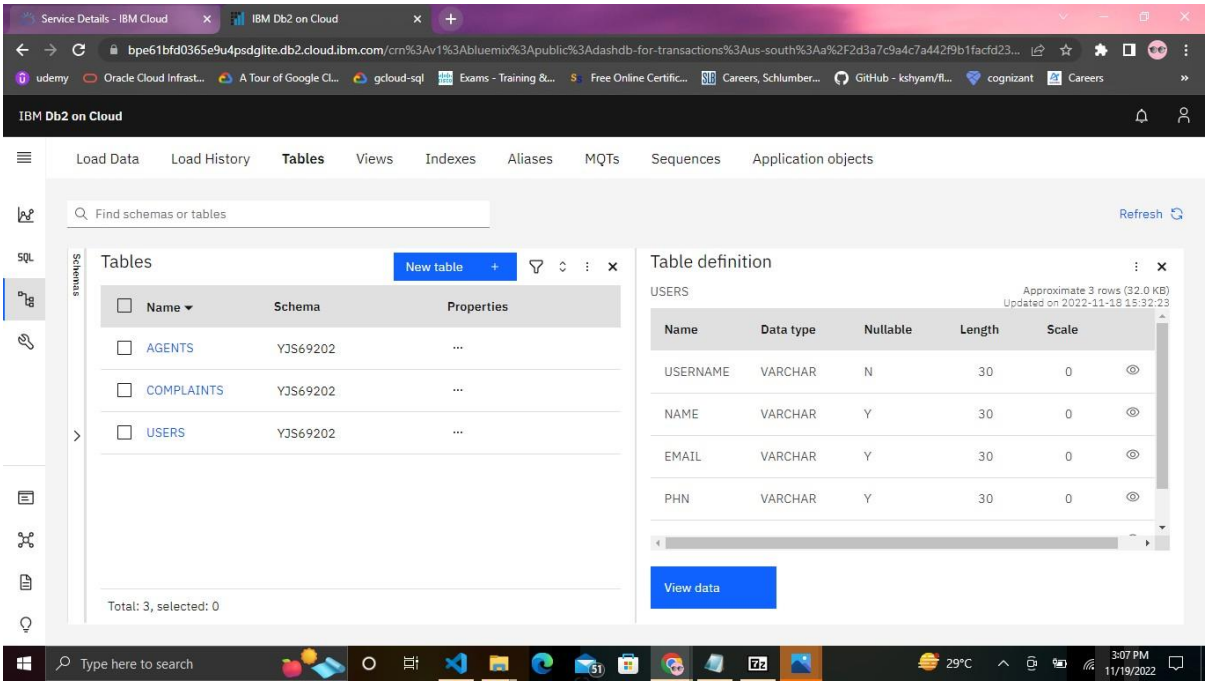
AGENT TABLE:



COMPLAINT TABLE:



USER TABLE:



Job Calendar:

When a user chooses a day from the job calendar, a list of jobs that are open on that day is displayed on the same page. Ajax features were used to construct this functionality from scratch.

8. TESTING

8.1 TEST CASES

Test Case ID	Feature Type	Component	Test Scenario	Expected Result	Result Status
01	Functional	user page	**Verify user is able to see show complain popup**	**Show complain popup displayed**	**Pass**
02	**UI**	user page	**Verify user has no complain**	**No complain displayed**	**Pass**
03	**UI**	**AGENT login**	**Text field visible to enter email**	**Text field visible**	**Pass**
04	**UI**	**User login**	**Text field visible to enter email**	**Text field visible**	**Pass**
05	**UI**	**Agent login**	**Text field visible to enter password**	**Text field visible**	**Pass**

8.2 USER ACCEPTANCE TESTING

Test Case ID	Feature Type	Component	Test Scenario	Expected Result	Result Status
01	**Functional**	**Home page**	**Verify user is able to see the login/signup popup when clicked on my account**	**Login/Signup displayed**	**pass**
02	**Ui**	**Home page**	**Verify the UI elements in the Login/Signup**	**The UI elements Working accordingly**	**pass**
03	**Functional**	**Home page**	**Verify user is able to login to the application with valid credentials**	**Login successful**	**pass**
04	**Functional**	**login page**	**Verify user cannot login to the application without valid credentials**	**Login unsuccessful**	**pass**

9. RESULT

This project is designed to solve the customer queries and achieve customer satisfaction. It is a web-enabled project. With this project the details about the product will be given to the customers in detail within a short span of time. Queries regarding the product or the services will also be clarified. It provides more knowledge about the various technologies.

10. ADVANTAGES AND DISADVANTAGES

Advantage

- Provides total population data reporting with no chart abstraction.
- Generates revenue (it shows when services are needed).
- Provides outreach information at fingertips.
- Flow sheet is a powerful tool to monitor clinical data and track trends.
- Provides a dashboard of who needs what.
- Improves team-based care.
- Smaller software package than EHRs.

Disadvantage

- Parallel documentation system
- Does not include information necessary for billing.
- Requires hardware, software and maintenance.
- Requires data entry and data maintenance.
- Experience burnout and stress.

11. CONCLUSION

The goal of this project is to satisfy customers by providing answers to their questions. The project is web-enabled. With the help of this project, clients will quickly receive detailed information about the product. Questions about the goods or services will also be answered. More information about the various technologies is provided.

12. FUTURE SCOPE

1. Answering each customer's question individually.

2. It is a pivotal moment for marketing.

3. It will bring about a major revolution.

14.APPENDIX

Source Code:

```
from flask import Flask, render_template, request, redirect, session, url_for
import ibm_db
import re
app = Flask(__name__)


# for connection
# conn= ""

app.secret_key = 'a'
print("Trying to connect...")
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-
4cd4-bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32304;SE
CURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=yjs69202;P
WD=nCHXXVA1056i46ky;", ", ")
print("connected..")


@app.route('/signup', methods=['GET', 'POST'])
def signup():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phn = request.form['phn']
        password = request.form['pass']
        repass = request.form['repass']
        print("inside checking")
        print(name)
        if len(username) == 0 or len(name) == 0 or len(email) == 0 or len(phn) == 0
or len(password) == 0 or len(repass) == 0:
            msg = "Form is not filled completely!!"
```

```
        print(msg)
        return render_template('signup.html', msg=msg)
    elif password != repass:
        msg = "Password is not matched"
        print(msg)
        return render_template('signup.html', msg=msg)
    elif not re.match(r'[a-z]+', username):
        msg = 'Username can contain only small letters and numbers'
        print(msg)
        return render_template('signup.html', msg=msg)
    elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
        msg = 'Invalid email'
        print(msg)
        return render_template('signup.html', msg=msg)
    elif not re.match(r'[A-Za-z]+', name):
        msg = "Enter valid name"
        print(msg)
        return render_template('signup.html', msg=msg)
    elif not re.match(r'[0-9]+', phn):
        msg = "Enter valid phone number"
        print(msg)
        return render_template('signup.html', msg=msg)

    sql = "select * from users where username = ?"
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.bind_param(stmt, 1, username)
    ibm_db.execute(stmt)
    account = ibm_db.fetch_assoc(stmt)
    print(account)
    if account:
        msg = 'Acccount already exists'
    else:
        userid = username
        insert_sql = "insert into users values(?,?,?,?,?)"
        prep_stmt = ibm_db.prepare(conn, insert_sql)
        ibm_db.bind_param(prep_stmt, 1, username)
        ibm_db.bind_param(prep_stmt, 2, name)
        ibm_db.bind_param(prep_stmt, 3, email)
        ibm_db.bind_param(prep_stmt, 4, phn)
        ibm_db.bind_param(prep_stmt, 5, password)
```

```
            ibm_db.execute(prep_stmt)
            print("successs")
            msg = "succesfully signed up"
        return render_template('dashboard.html', msg=msg, name=name)
    else:
        return render_template('signup.html')


@app.route('/dashboard')
def dashboard():
    return render_template('dashboard.html')

@app.route('/')
def base():
    return redirect(url_for('login'))

@app.route('/login', methods=["GET", "POST"])
def login():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        userid = username
        password = request.form['pass']
        if userid == 'admin' and password == 'admin':
            print("its admin")
            return render_template('admin.html')
        else:
            sql = "select * from agents where username = ? and password = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, password)
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            print(account)
            if account:
                session['Loggedin'] = True
                session['id'] = account['USERNAME']
                userid = account['USERNAME']
                session['username'] = account['USERNAME']
```

```
            msg = 'logged in successfully'

            # for getting complaints details
            sql = "select * from complaints where assigned_agent = ?"
            complaints = []
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            while dictionary != False:
                complaints.append(dictionary)
                dictionary = ibm_db.fetch_assoc(stmt)
            print(complaints)
            return render_template('agentdash.html', name=account['USERNAME'],
complaints=complaints)

        sql = "select * from users where username = ? and password = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['USERNAME']
            userid = account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'logged in successfully'

            # for getting complaints details
            sql = "select * from complaints where username = ?"
            complaints = []
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            while dictionary != False:
                # print "The ID is : ",  dictionary["EMPNO"]
                # print "The Name is : ", dictionary[1]
```

```
            complaints.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)

        print(complaints)
        return render_template('dashboard.html', name=account['USERNAME'],
complaints=complaints)
    else:
        msg = 'Incorrect user credentials'
        return render_template('dashboard.html', msg=msg)
else:
    return render_template('login.html')


@app.route('/addnew', methods=["GET", "POST"])
def add():
    if request.method == 'POST':
        title = request.form['title']
        des = request.form['des']
        try:
            sql = "insert into complaints(username,title,complaint) values(?,?,?)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.bind_param(stmt, 2, title)
            ibm_db.bind_param(stmt, 3, des)
            ibm_db.execute(stmt)
        except:
            print(userid)
            print(title)
            print(des)
            print("cant insert")
        sql = "select * from complaints where username = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            # print "The ID is : ",  dictionary["EMPNO"]
            # print "The Name is : ", dictionary[1]
            complaints.append(dictionary)
```

```
            dictionary = ibm_db.fetch_assoc(stmt)
        print(complaints)
        return render_template('dashboard.html', name=userid,
complaints=complaints)


@app.route('/agents')
def agents():
    sql = "select * from agents"
    agents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        agents.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template('agents.html', agents=agents)


@app.route('/addnewagent', methods=["GET", "POST"])
def addagent():
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        domain = request.form['domain']
        password = request.form['password']
        try:
            sql = "insert into agents values(?,?,?,?,?,?,2)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, name)
            ibm_db.bind_param(stmt, 3, email)
            ibm_db.bind_param(stmt, 4, phone)
            ibm_db.bind_param(stmt, 5, password)
            ibm_db.bind_param(stmt, 6, domain)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
```

```
        sql = "select * from agents"
        agents = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            agents.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)

        return render_template('agents.html', agents=agents)


@app.route('/updatecomplaint', methods=["GET", "POST"])
def updatecomplaint():
    if request.method == 'POST':
        cid = request.form['cid']
        solution = request.form['solution']
        try:
            sql = "update complaints set solution =?,status=1 where c_id = ? and
assigned_agent=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, solution)
            ibm_db.bind_param(stmt, 2, cid)
            ibm_db.bind_param(stmt, 3, userid)
            ibm_db.execute(stmt)
            sql = "update agents set status =3 where username=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from complaints where assigned_agent = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            complaints.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)
```

```
        # print(complaints)
        return render_template('agentdash.html', name=userid,
complaints=complaints)


@app.route('/tickets')
def tickets():
    sql = "select * from complaints"
    complaints = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        complaints.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)

    sql = "select username from agents where status <> 1"
    freeagents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        freeagents.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    print(freeagents)
    return render_template('tickets.html', complaints=complaints,
freeagents=freeagents)


@app.route('/assignagent', methods=['GET', 'POST'])
def assignagent():
    if request.method == "POST":
        ccid = request.form['ccid']
        agent = request.form['agent']
        print(ccid)
        print(agent)
        try:
            sql = "update complaints set assigned_agent =? where c_id = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, agent)
```

```
            ibm_db.bind_param(stmt, 2, ccid)
            ibm_db.execute(stmt)
            sql = "update agents set status =1 where username = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant update")
        return redirect(url_for('tickets'))

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

GitHub Repository Link:

https://github.com/IBM-EPBL/IBM-Project-34470-1660236275