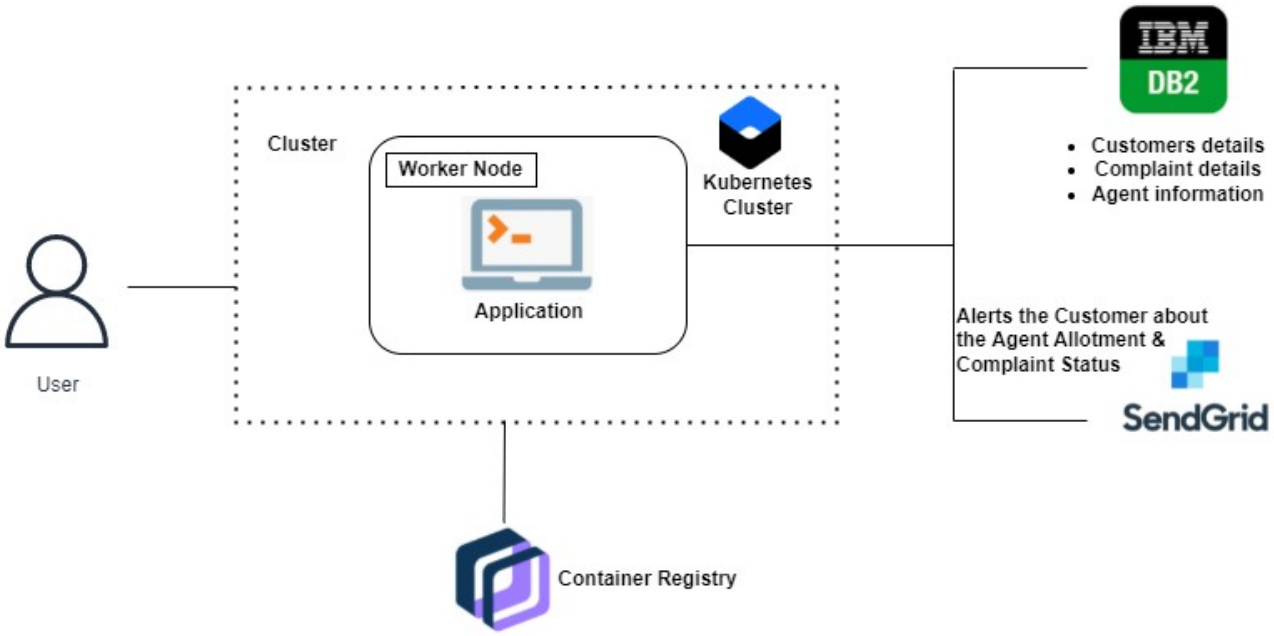
CUSTOMER CARE REGISTRY



Team Details	
Team ID	PNT2022TMID34671
Team Leader	Shagish Bergin V
Team Member 1	Shamitha Sheffrin S
Team Member 2	Shahina Rizvana S M
Team Member 3	Sreeja J
Faculty Mentor	Merbin Jose P

1. INTRODUCTION

1.1 PROJECT OVERVIEW

Customer service is the support you offer your customers both before and after they buy and use your products or services that helps them have an easy and enjoyable experience with you. Offering amazing customer service is important if you want to retain customers and grow your business. Today’s customer service goes far beyond the traditional telephone support agent. It’s available via email, web, text message, and social media. Many companies also provide self-service support, so customers can find their own answers at any time day or night. Customer support is more than just providing answers; it’s an important part of the promise your brand makes to its customers.

Customer service is critical to competing effectively.In the past, people chose which companies they did business with based on price, or the product or service offered, but today the overall experience is often the driver.

It’s often said that it’s cheaper to keep existing customers than to find new ones. (It’s even been estimated that acquiring customers costs 6–7x more.) And it’s true: Bad customer service is a key driver of churn. The U.S. Small Business Administration reports that 68% of customers leave because they’re upset with the treatment they've received. Don’t let that happen to you. Prioritising customer service support helps you attract and retain loyal customers, and can have a big impact on your company’s bottom line.

It’s no surprise that as today’s social, mobile consumers have grown accustomed to getting what they want, when they want it, their expectations have risen accordingly. In fact, in a recent poll, 82% of CEOs reported that customer expectations of their companies were “somewhat” or “much” higher than they were three years ago. What’s more, today’s customers are quick to share negative experiences online, where they can quickly reach large audiences. It’s more important than ever to support customers on every channel from day one and establish what good customer service looks like internally and externally.

1.2 PURPOSE:

The aim of the project is to provide a sleek interface to the user without any complications,to notify the customers about the review and providing tickets to be accessed during the review process . This helps the customer for the easy access and ease of access to the application.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

- Slow response time
- Rude communication of customer service staff
- Lack of real time engagement
- Being transferred from one agent to another
- Excessive customer service automation
- No unified customer view
- Incompetent customer service staff
- Offering a wrong product
- Fail to meet commitments
- No or poor after-sales support

2.2 REFERENCES

Complaint management system:

this project is aimed at implementing a complaint management system which will exclusively:
• Allow customer to registered and obtain username and password to login into the system and lodge in their complain as well as to view previous compla
• Simultaneously update changes made to any data, item in the entire databas
• Efficiently provide a medium through which authorize personel can attend to those complain from customers from any location.

Email support:

Email is the classic, common, and widespread way customers communicate with companies. With the right email management software, email can be one of the easiest ways to organize, prioritize, and delegate customer support interactions in one place. What’s so great about email? It’s hard to find someone without an email account, and customers can reach out anytime to log an inquiry. Email is also usually the first form of support a business will offer.

Email is also a common type of internal support. Human Resources, Payroll, and IT teams can use email to handle issues or answer questions for employees.

Self-Service:

Support teams know a lot about customer issues—and the best way to solve them. But agents also spend a lot of time looking for information. A collective knowledge base can help tap into that institutional knowledge and aid your customer service team with the information they need to better

serve customers. It can also help your business to understand and fill the knowledge gaps your company might have.

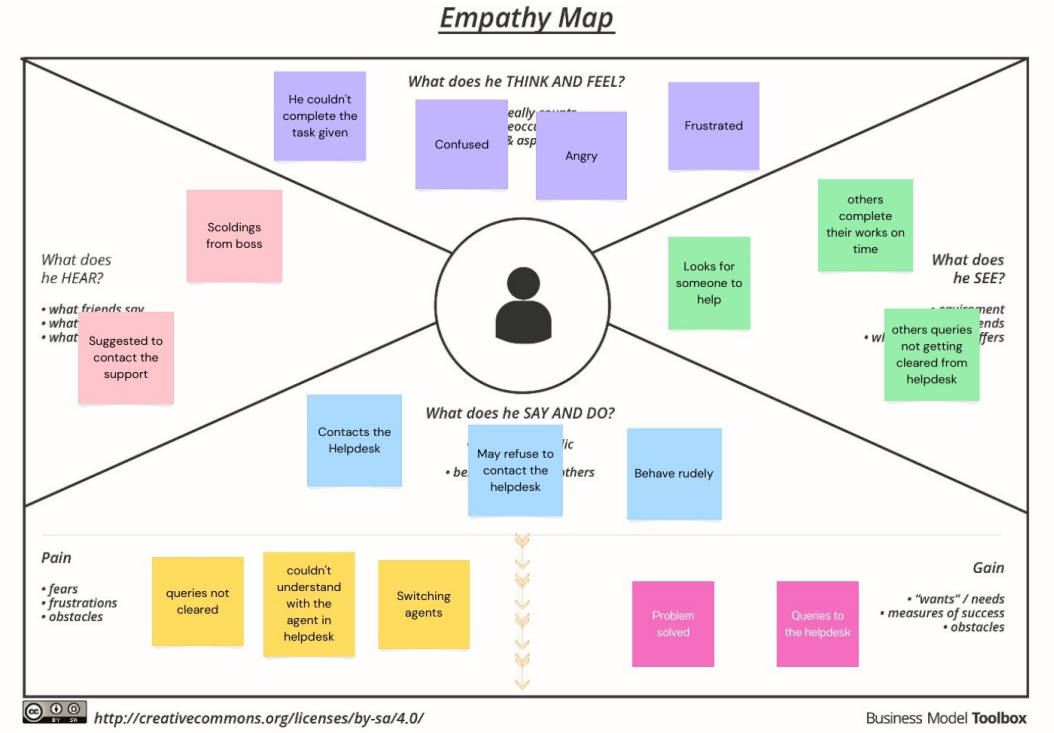
Besides, customers prefer self-service because it offers the least amount of interaction friction. By letting customers help themselves through a help center, online community, or customer service portal, you can reduce customer friction while also improving efficiency and delivering faster resolutions. Offering self-service is a baseline for excellent customer service and a great self-service experience can boost customer satisfaction, reduce support costs, and increase agent engagement.

2.3 PROBLEM STATEMENT DEFINITION

To create a customer care support application which is easy for the customer to use and getting the queries cleared effectively.

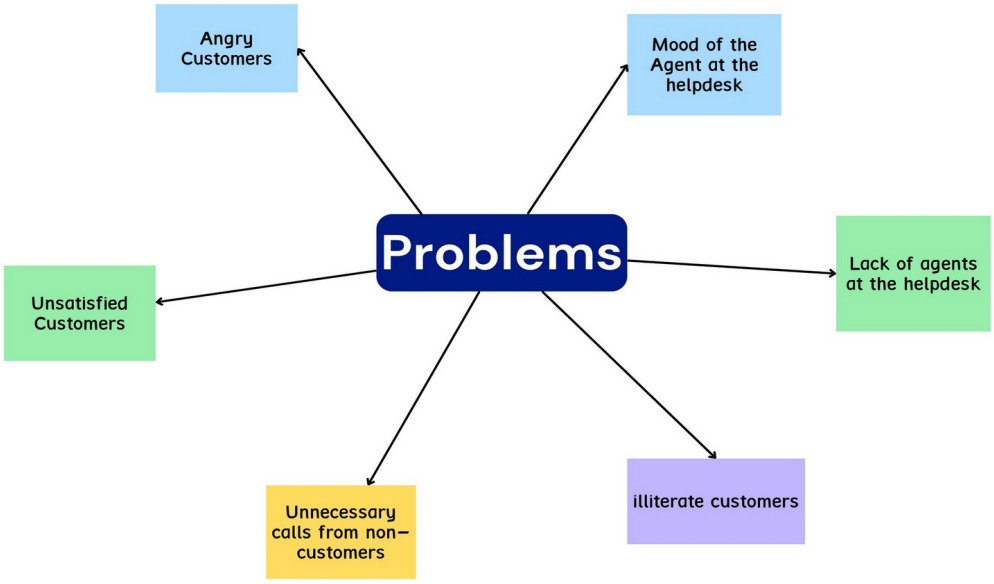
3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

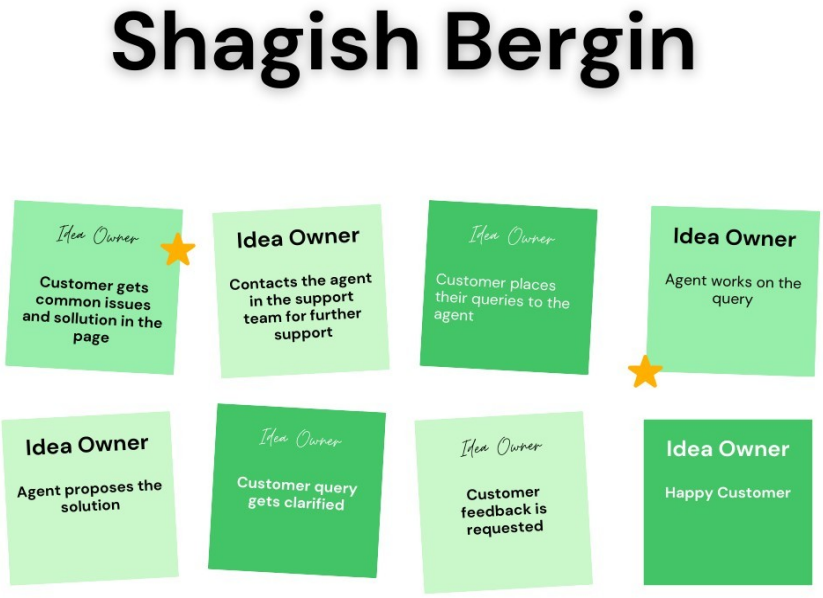


3.2 IDEATION AND BRAINSTORMING

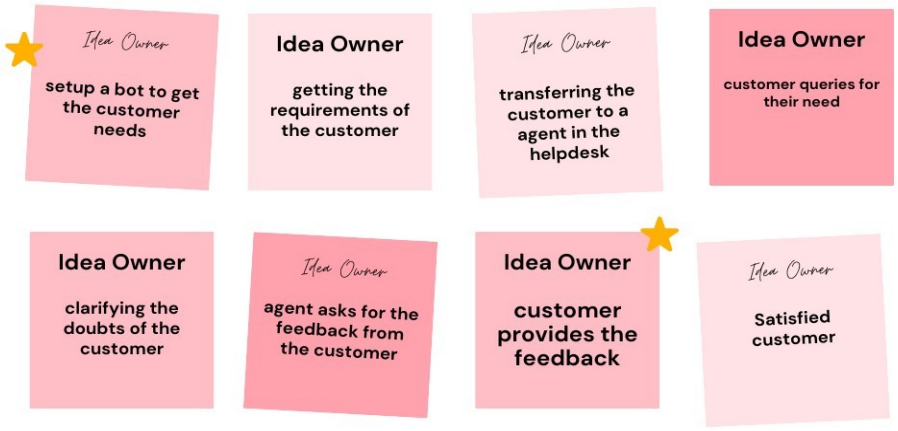
Problems



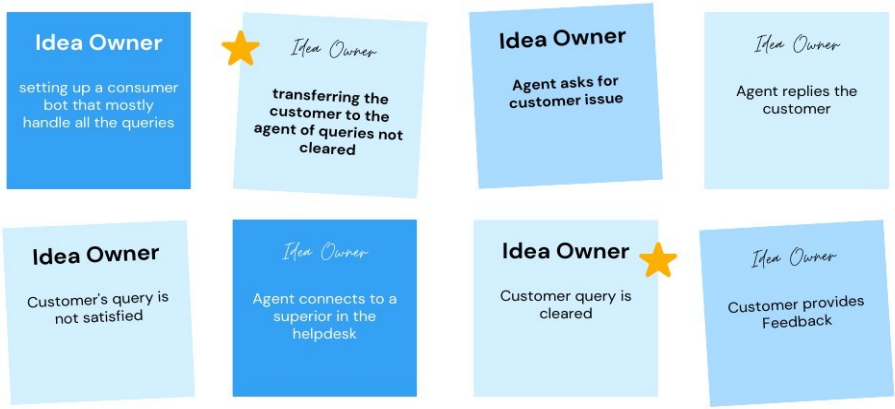
Steps to overcome the problem



Shamitha Sheffrin



Shahina Rizvana

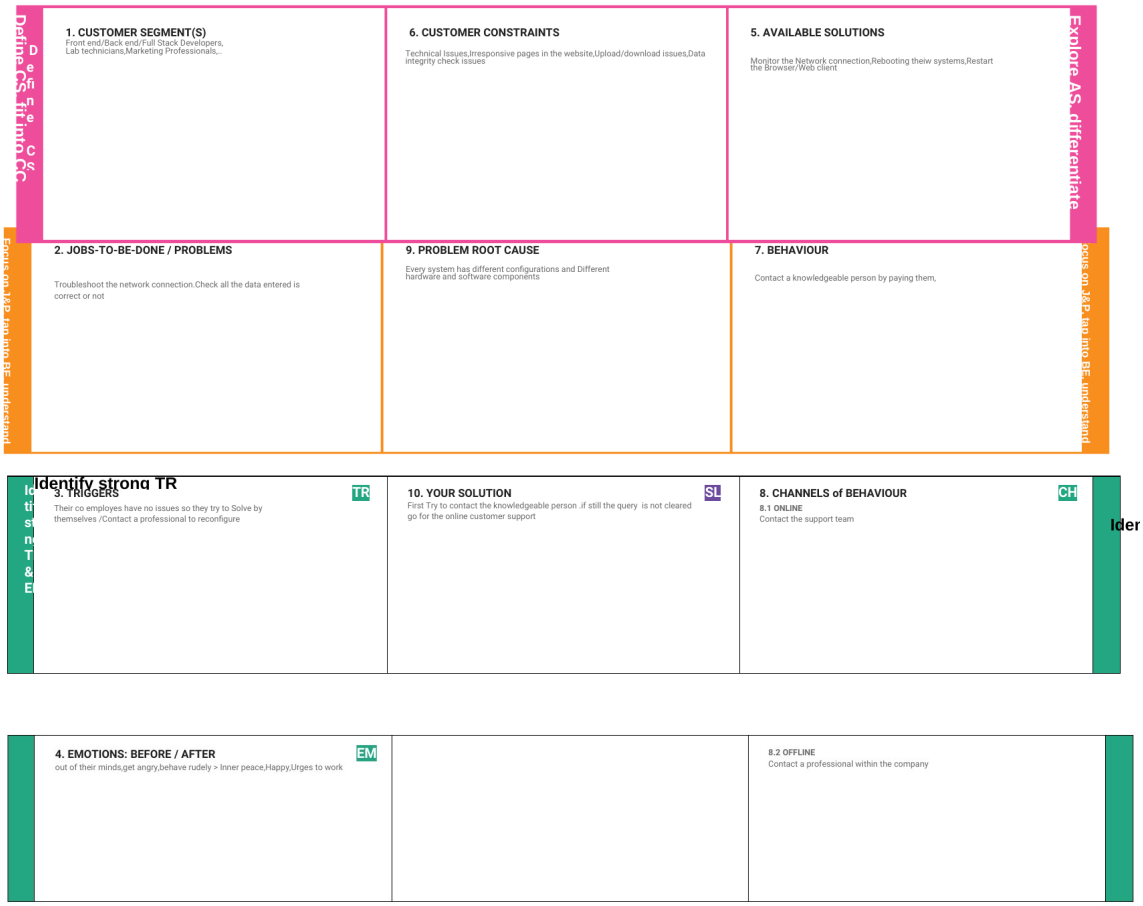


Idea Prioritization

3.3 PROPOSED SOLUTION

- To introduce the ticket concept in the application to overcome the user credentials exposure
- To make a sleek application possible
- To deploy the application in a fast medium to access it faster

3.4 PROBLEM SOLUTION FIT



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

4.1.1 SYSTEM REQUIREMENTS

Software Required

Python, Flask ,flask_mail , Docker

System Required:

8GB RAM,Intel Core i3,OS-Windows/Linux/MAC ,Laptop or Desktop

4.1.2 SOFTWARE REQUIREMENTS

Registration page:

Users can register for an ID to contact the support.

Login page:

Users , Agents , Admin can use this page to login in to their respective dashboards.

Customer Dashboard:

Customer’s after login page.Here the customers can query and can check whether their query is replied or not.

Agent Dashboard:

Agent’s after login page.It contains the details of the assigned customers.Here the agent can reply to the customer queries.

Admin Dashboard:

Admin’s after login page.It contains the customer’s Queries , list of available agents and the option to assign customers to agents.

4.2 NON-FUNCTIONAL REQUIREMENTS

Usability:

Users can access the application with a minimal spec Devices and no additional dependencies a re required to access the aplication

Security:

The cloud vendor provides security to the
deployed application

Reliability:

The User’s credentials are protected by the secured database provided by the cloud vendor.So customers can trust the application.

Performance:

Since the application is deployed in the Docker container,Docker provides a smooth performance to the clients/customers.

Availability:

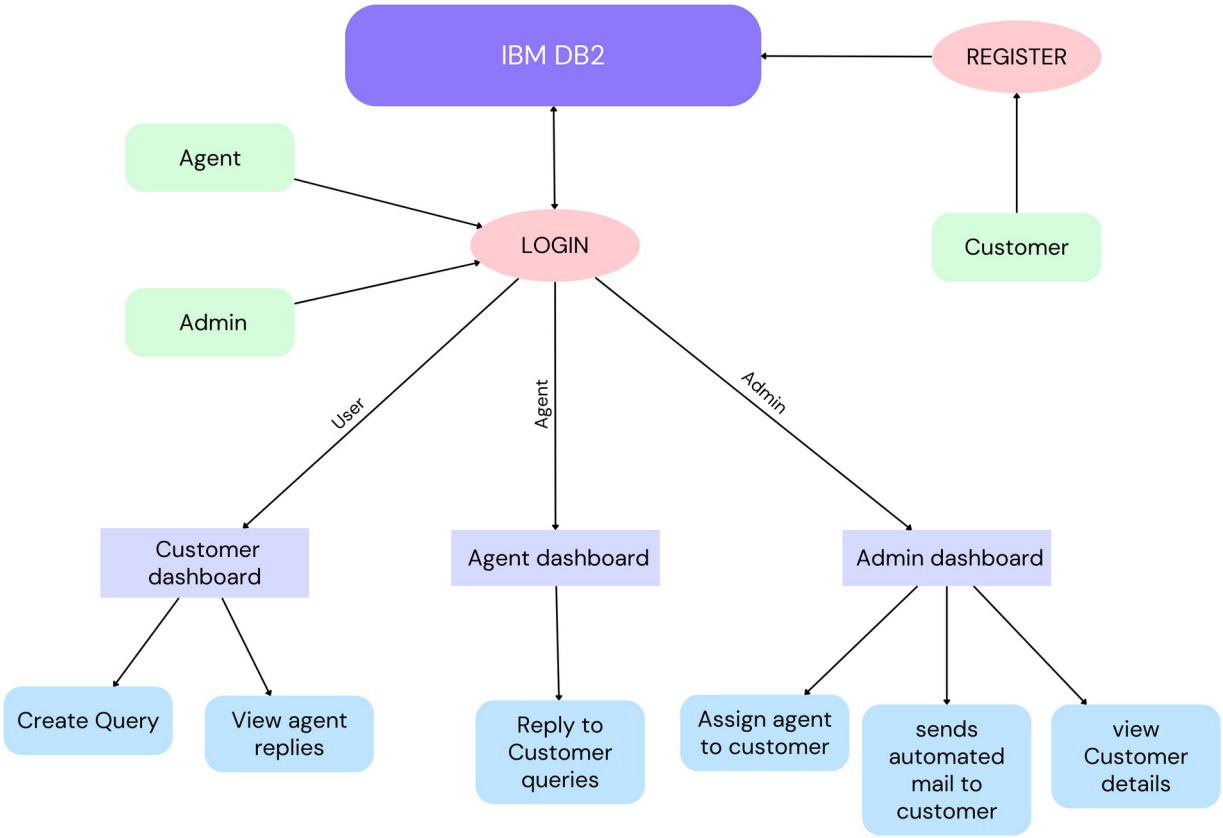
Since the application is deployed in cloud The cloud vendor provides the availability of the application

Scalability:

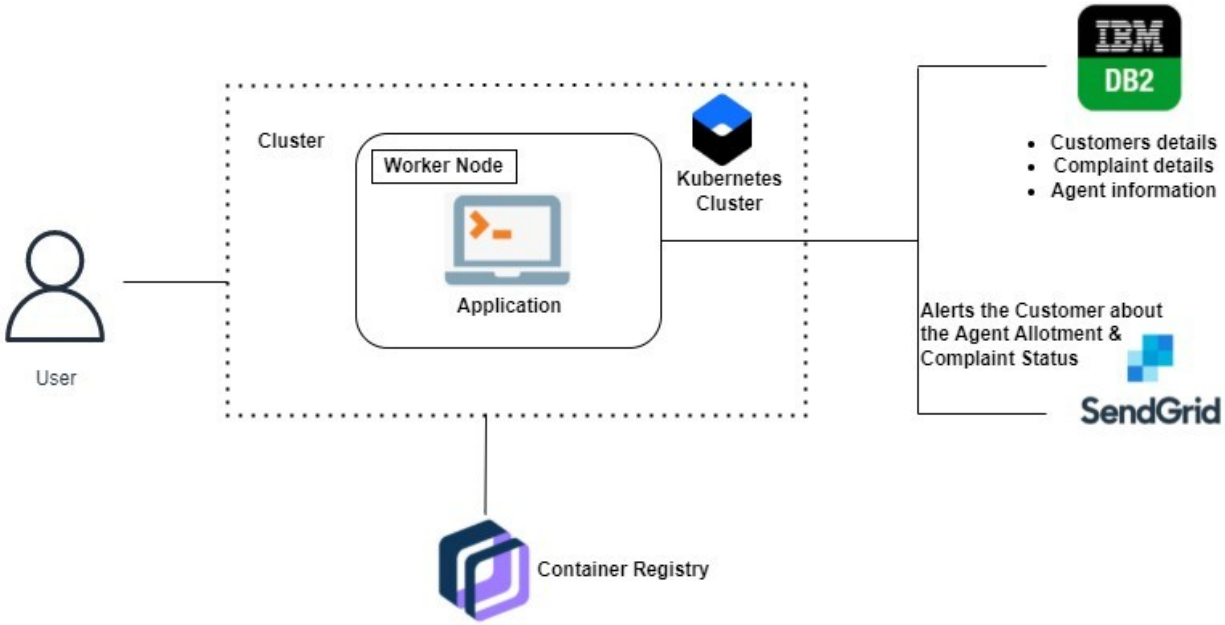
If the no of customers are increased,the application can be Scaled through the cloud vendor

5. PROJECT DESIGN

5.1 DATAFLOW DIAGRAM



5.2 SOLUTION AND TECHNICAL ARCHITECTURE



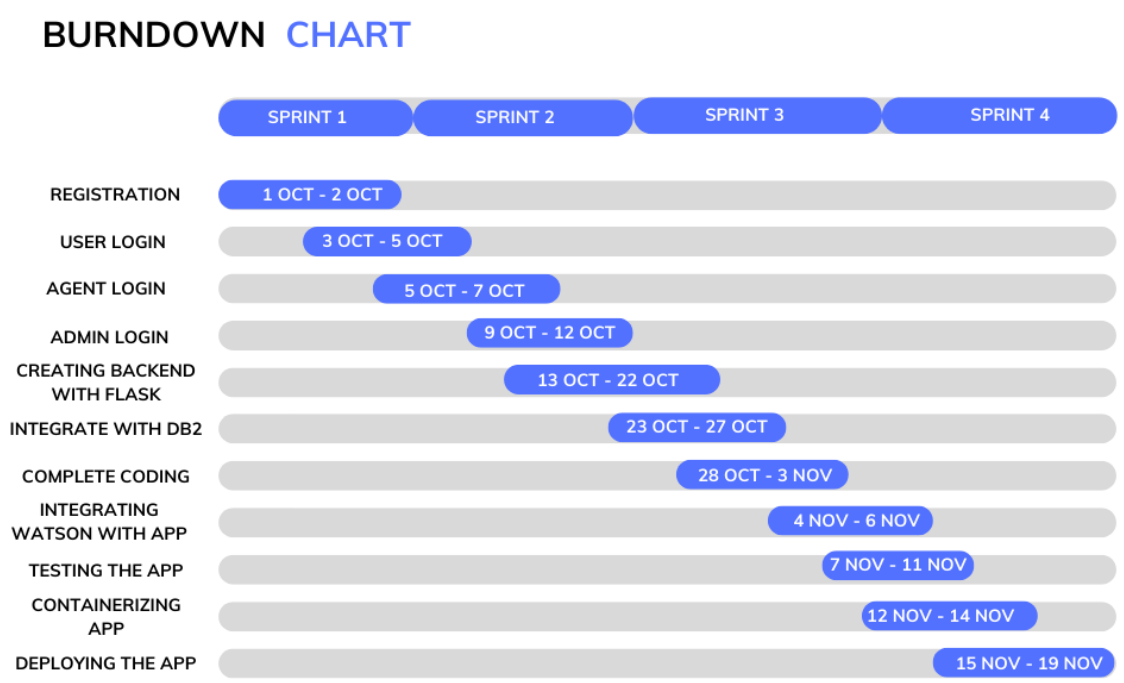
6. PROJECT PLANNING & SCHEDULING

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	Shahina Rizvana
Sprint-1	Login	USN-2	As a user, I can log into the application by entering email & password	1	High	Shamitha Sheffrin
Sprint-2	Agent	USN-3	As an agent, I can see the customers assigned to me.	2	High	Shagish Bergin- Team Lead
Sprint-3	Admin	USN-4	As an admin, I can assign the agents to the customers	2	High	Shagish Bergin v
Sprint-4	Sendgrid	USN-5	As an admin, while i assign customers It will be intimated through sendgrid email service	2	High	Sreeja
Sprint-4	Virtual Assistant	USN-6	The virtual assistant is set up to assist the customer with the options in the web page	2	Medium	Shamitha Sheffrin

6.2 SPRINT DELIVERY AND 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	9 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	14 Days	31 Oct 2022	05 Nov 2022	20	05 Nov 2022
Sprint-3	20	13 Days	07 Nov 2022	12 Nov 2022	20	12 Nov 2022
Sprint-4	20	12 Days	14 Nov 2022	19 Nov 2022	20	19 Nov 2022

Burndown Chart



7. CODING & SOLUTIONING

7.1 TICKET CREATION

For every Query creation a ticket is generated and it is used to hide the identity to others.

CODE GENERATION SNIPPET

```
def Upper_Lower_string(length):
    result = ''.join((random.choice(string.ascii_uppercase) for x in
range(length)))
    return result

def success():
    if request.method == "POST":
        ticket = session['ticket'] = Upper_Lower_string(16)
        print(ticket, session['ticket'])
        query = request.form['query']
        sql = "UPDATE user SET QUERY=?,TICKET=?,REVIEW_STATUS=0 WHERE
USERNAME=?"
        out = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(out, 1, query)
        ibm_db.bind_param(out, 2, session['ticket'])
        ibm_db.bind_param(out, 3, session['customer']['USERNAME'])
        status = ibm_db.execute(out)
        if status:
            msg = 'Success ! Your Ticket Nno is :', ticket, 'You can now return
to the home page'
            return render_template('success.html', msg=msg)
        else:
            msg = 'Error Submitting your Query'
```

```
            return render_template('success.html', msg=msg)
```

7.2 DYNAMIC USER MATCHING

This is a feature that enables anyone (customer,agent,admin) to login using a single logon page.

LOGIN SNIPPET

```
def login():
    if request.method == "POST":
        customer = list()
        agent = list()
        name = request.form['name']
        password = request.form['password']
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM user WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            session['role'] = 'user'
            session['customer'] = rs
            print(rs)
            return render_template('dashboard.html')
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM agent WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            cms = ibm_db.exec_immediate(conn, 'SELECT * FROM user')
            agt = ibm_db.exec_immediate(conn, 'SELECT * FROM agent')
            customers = ibm_db.fetch_assoc(cms)
            agents = ibm_db.fetch_assoc(agt)
            while customers:
                customer.append(customers)
                customers = ibm_db.fetch_assoc(cms)
            while agents:
                agent.append(agents)
                agents = ibm_db.fetch_assoc(agt)
            print(customer)
            print(agent)
            session['role'] = 'agent'
            session['name'] = rs['USERNAME']
            session['customer'] = customer
            session['agent'] = agent
            return render_template('dashboard.html')
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM admin WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            cms = ibm_db.exec_immediate(conn, 'SELECT * FROM user')
            agt = ibm_db.exec_immediate(conn, 'SELECT * FROM agent')
            customers = ibm_db.fetch_assoc(cms)
            agents = ibm_db.fetch_assoc(agt)
```

```
            while customers:
                customer.append(customers)
                customers = ibm_db.fetch_assoc(cms)
            while agents:
                agent.append(agents)
                agents = ibm_db.fetch_assoc(agt)
            print(customer)
            print(agent)
            session['role'] = 'admin'
            session['customer'] = customer
            session['agent'] = agent

            return render_template('dashboard.html', agent=agent,
customer=customer)
        else:
            msg = 'UID/Password is incorrect'
            return render_template('login.html', msg=msg)
    else:
        return render_template('login.html')
```

7.3 AUTOMATED MAILING

By using this feature ,whenever the admin puts an agent to a customer,The customer will be notified with a mail alert.This is made use of the python package ‘flask_mail’.

MAILING SNIPPET

```
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'shagish.111937@sxcce.edu.in'
app.config['MAIL_PASSWORD'] = 'SX@09/07/2002'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)


msg = Message(
                    f'Hello {usr_name[i]}',
                    sender='shagish.111937@sxcce.edu.in',
                    recipients=[f'{emails[i]}']
                )
                msg.body = f'Agent named {agent[i]} alloted to your query.
{agent[i]} will be responding you soon within 24 hrs.'
                mail.send(msg)
                msg = 'Allotments updated Successfully'
            except:
                msg = "Error saving allotments/sending emails"
```

7.4 DATABASE SCHEMA

```
--create the user table

CREATE TABLE USER (
USERNAME VARCHAR(32) NOT NULL,
EMAIL VARCHAR(32) NOT NULL,
PASSWORD    VARCHAR(16) NOT NULL,
QUERY VARCHAR(200),
REVIEW_STATUS     INTEGER,
TICKET      VARCHAR(16),
```

```
ASSIGNED_AGENT    VARCHAR(32),
REPLY VARCHAR(250)
);

--create the agent table

CREATE TABLE AGENT(
USERNAME VARCHAR(32),
ASSIGNED_CUSTOMERS VARCHAR(3000),
PASSWORD VARCHAR(16)
);

--create the admin table

CREATE TABLE ADMIN(
USERNAME VARCHAR(32),
PASSWORD VARCHAR(16)
);

--inserting to agent table

INSERT
 INTO  "WVN94274"."AGENT" ("USERNAME","ASSIGNED_CUSTOMERS","PASSWORD")
 VALUES(
'agent',                --USERNAME  VARCHAR(32)
NULL,                   --ASSIGNED_CUSTOMERS  VARCHAR(3000)
'agent'                 --PASSWORD  VARCHAR(16)
);

--inserting to admin table

INSERT
  INTO  "WVN94274"."ADMIN" ("USERNAME","PASSWORD")
  VALUES(
'admin',                --USERNAME  VARCHAR(32)
'admin'                 --PASSWORD  VARCHAR(16)
);
```

8. RESULTS

8.1 PERFORMANCE METRICS

The overall performance of the application depends on the following factors

- Internet connectivity
- Updated Browser Software

When Deployment the overall memory usage is just 3% and required 1.53 cores to run it.

9. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- very fast UI.
- since python is used as backend .The performance is also good.
- Can use the application with a medium network speeds.
- The application runs 24/7.
- All user credentials are safely stored.

DISADVANTAGES

- Users can’t place multiple queries

10. CONCLUSION

From here we conclude our project.This application is really good at its performance as the backend program used is lighter.Today, we are seeing that, more often than not, exceptional customer service makes the infrequent issues not as painful as they once were. Our customers expect and deserve the best, and consistently excellent customer service helps to generate not only customer retention and renewals, but also referrals.

More and more managers are playing an active role in making sure their staff expresses and practices good customer-service actions on a daily basis. The exceptional manager will go as far as personally reaching out to the customers, as well as extending the proverbial “customer service branch” to make sure expectations are not only met but exceeded.

11. FUTURE SCOPE

The application is really safe and can be expanded in the future .These kind of application can be run in a minimal storage and memory.There are future plans to build more components to improve the features , quality and performance of the application.

12. APPENDIX

SOURCE CODE

==> templates/base.html <==

```
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>
        {%block title%}

        {%endblock%}
    </title>
    <!-- Bootstrap -->
    <link rel="stylesheet"
href="{{url_for('static',filename='css/styles.css')}}">
    <link rel="stylesheet" href="{{
url_for('static',filename='css/bootstrap.min.css') }}">
    <script src="{{ url_for('static',filename='js/script.js') }}"></script>
    <script>
        window.watsonAssistantChatOptions = {
            integrationID: "09da12b3-a730-473b-bef4-8a122b7e4b9e", // The ID of
this integration.
            region: "us-south", // The region your integration is hosted in.
            serviceInstanceID: "845b2a44-fbd0-462c-aedc-d15aa31eb452", // The ID
of your service instance.
```

```
            onLoad: function (instance) { instance.render(); }
        };
        setTimeout(function () {
            const t = document.createElement('script');
            t.src =
"https://web-chat.global.assistant.watson.appdomain.cloud/versions/" +
(window.watsonAssistantChatOptions.clientVersion || 'latest') +
"/WatsonAssistantChatEntry.js";
            document.head.appendChild(t);
        });
    </script>
</head>

<body style="background-image: url({{ url_for('static',
filename='img/bg.jpg') }})">
    <nav class="navbar navbar-primary">
        <div class="container-fluid">
            <!-- Brand and toggle get grouped for better mobile display -->
            <div class="navbar-header">
                <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse"
                    data-target="#bs-example-navbar-collapse-1" aria-
expanded="false">
                    <span class="sr-only">Toggle navigation</span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                <a class="navbar-brand" href="/">Customer Care Registry</a>
            </div>

            <ul class="nav navbar-nav navbar-right">
                <li><a href="{{url_for('login')}}">Login</a></li>
                <li><a href="{{url_for('register')}}">Register</a></li>

            </ul>
        </div>

    </nav>
    {%block content%}

    {%endblock%}
</body>

</html>
```

==> templates/dashboard.html <==

```
{%extends 'base.html'%}
{%block title%}
Dashboard
{%endblock%}
{%block content%}
<center>
    <div class="container container-pd-5">
        <h1>Welcome to Dashboard</h1>
        <div>You have successfully logged in as {{ session['role'] }}</div>
        <hr>
    </div>
</center>
{% if session['role'] =='user' %}
<center>
    {% set customer = session['customer'] %}
```

```
    <h1 class="heading">Hello {{ customer['USERNAME'] }}</h1>
    {% if customer['QUERY'] =='none' %}
    <h1>You Haven't made any Queries</h1>
    {% else %}
    <div>
        <h1>Your Previous Query :</h1>
        <div>{{ customer['QUERY'] }} </div>
        <h1>Your Ticket:</h1>
        <div>{{ customer['TICKET'] }}</div>
        {% if customer['REVIEW_STATUS'] == 1 %}
        <h1>Reply from the Agent:</h1>
        <div>{{ customer['REPLY'] }}</div>
        {% else %}
        <h1>Review Status:</h1>
        <div>Not Yet Reviewed</div>
        {% endif %}
    </div>
    <hr>
    {% endif %}
    <div>
        <div id="add_elem">
            <button id="form" class="btn btn-success"
onclick="form_create()">Create Query</button>
            <a class="btn btn-primary" href="{{ url_for('redir') }}">Back to
Home</a>
        </div>
    </div>
</center>

{% elif session['role'] == 'agent' %}
<hr>
<center>
    <form action="{{ url_for('agent_submit_reply') }}" method="POST">
        <table class="table table-hover">
            <tr>
                <th scope="col">Name</th>
                <th scope="col">Ticket No</th>
                <th scope="col">Query</th>
                <th scope="col">Reply</th>
            </tr>
            {% for cus in session['customer'] %}
            {% if cus['ASSIGNED_AGENT'] == session['name'] and cus['REPLY'] ==
'none' and cus['REVIEW_STATUS'] != 1 %}

            <tr>

                <td>
                    <select name="name">
                        <option value="{{ cus['USERNAME'] }}">{{ cus['USERNAME']
}}</option>
                    </select>
                </td>
                <td>
                    {{ cus['TICKET'] }}
                </td>
                <td>
                    {{ cus['QUERY'] }}
                </td>
                <td><textarea name="text" cols="50" rows="4"></textarea></td>
            </tr>
            {% endif %}
            {% endfor %}
            <tr>
                <td colspan="4">
```

```
                    <center><input type="submit" value="submit"></center>
                </td>

            </tr>

        </table>
    </form>




</center>
{% elif session['role']=='admin' %}
<center>
    <div class="container">
        <form action="{{ url_for('admin_query') }}" method="POST">
            <h1>All Customers</h1>
            <table class="table table-hover">
                <tr>
                    <th scope="col">Name</th>
                    <th scope="col">Ticket No</th>
                    <th scope="col">Email</th>
                    <th scope="col">Query</th>
                    <th scope="col">Query Status</th>
                    <th scope="col">Assigned to</th>
                    <th scope="col">Assign To</th>
                </tr>

                {% for cus in session['customer'] %}
                {% if cus['REVIEW_STATUS'] == 0 and not cus['QUERY'] == 'none'
%}

                <tr>

                    <td>
                        <select name="cus_name">
                            <option
value="{{ cus['USERNAME'] }}">{{ cus['USERNAME'] }}</option>
                        </select>
                    </td>
                    <td>
                        {{ cus['TICKET'] }}
                    </td>
                    <td>
                        <select name="email">
                            <option
value="{{ cus['EMAIL'] }}">{{ cus['EMAIL'] }}</option>
                        </select>
                    </td>

                    <td>
                        {{ cus['QUERY'] }}
                    </td>
                    <td>
                        {% if cus['REVIEW_STATUS'] == 1 %}
                        Reviewed
                        {% else %}
                        Not yet Reviewed
                        {% endif %}
                    </td>
                    <td>{{ cus['ASSIGNED_AGENT'] }}</td>
                    <td>
                        <select name="agent_name">
                            <option value="none">Select agent</option>
```

```
                            {% for agt in session['agent'] %}
                            <option
value="{{ agt['USERNAME'] }}">{{ agt['USERNAME'] }}</option>
                            {% endfor %}

                        </select>
                    </td>
                </tr>
                {% endif %}
                {% endfor %}
                <tr>
                    <td colspan="7">
                        <center><input type="submit" value="submit"></center>
                    </td>

                </tr>

            </table>
        </form>
    </div>
</center>
<center>
    <h1>Available Agents</h1>

    <table>
        {% for agt in agent %}
        <tr>
            <td>
                {{ agt['USERNAME'] }}
            </td>
        </tr>
        {% endfor %}


    </table>
</center>
{% else %}
<h1>UnIdentified User</h1>
{% endif %}
{%endblock%}
```

==> templates/done.html <==

```
{% extends 'base.html'%}
{% block title %}
Done
{%endblock%}
{%block content%}
{% if session['role'] == 'agent' %}
<center>
    <h1>{{msg}}</h1>
</center>
{% endif %}
{% if session['role'] == 'admin' %}
<center>
    <h1>{{msg}}</h1>
</center>
{% endif %}

{% endblock %}
==> templates/index.html <==
{%extends 'base.html'%}
{%block title%}
```

```
HOME | Customer Care Registry
{%endblock%}
{%block content%}
<center>
    <div class="container">
        <h1>Welcome to Customer Care Registry</h1>
        <br><br><br><br><br><br>
        <h2>Customer service is the support that organizations offer to
customers
            before and after purchasing a product or service. In customer
service,
            the organization's representative values both potential and existing
customers equally.
            Customer service representatives are the main line of contact
between an
            organization and its customers, making CX a
            critical facet and the main priority of customer service teams</h2>
    </div>
</center>


{%endblock%}
```

==> templates/login.html <==

```
{%extends 'base.html'%}
{%block title%}
Login Page
{%endblock%}
{%block content%}
<div class="container pd-20">
    <h1 style="margin-top:100px;">
        Welcome to Login page
    </h1>
    <h2>{{ msg }}</h2>
    <form action="{{url_for('login')}}" method="POST">
        <div class="form-group">
        <label class="col-md-5">User Name</label><input class="form-control col-
md-10" name="name" type="text">
        <label class="col-md-5">Password:</label><input class="form-control col-
md-10" name="password" type="password">
            <input class="btn btn-success" type="submit" value="submit">
            <input class="btn btn-danger" type="reset" value="clear">
        </div>
        <div>
            <a href="{{url_for('home')}}" class="btn btn-primary">< Back to
Home</a>

        </div>
    </form>
</div>
{%endblock%}
```

==> templates/register.html <==

```
{% extends 'base.html' %}
{%block title%}
Registration page
{%endblock%}

{%block content%}
<div class="container pd-20">
```

```
    <h1 style="margin-top:100px;">
        Welcome to Registration page
    </h1>
    <h2>{{ msg }}</h2>
    <form name="register" action="{{url_for('register')}}" method="POST"
onsubmit="submitted()">
        <div class="form-group">
            <label class="col-md-5">User Name</label><input class="form-control
col-md-5" name="name" type="text"
                required>
            <label class="col-md-5">Email</label><input class="form-control col-
md-5" name="email" type="text" required>
            <label class="col-md-5">New Password:</label><input class="form-
control col-md-5" id="p1" name="password1"
                type="password" required>
            <label class="col-md-5">confirm Password:</label><input class="form-
control col-md-5" id="p2"
                name="password" type="password" required>
            <input name="submit" class="btn btn-success" type="submit"
value="submit">
            <input class="btn btn-danger" type="reset" value="clear">
        </div>
        <div>
            <a class="btn btn-primary" href="{{ url_for('login') }}">Login
instead?</a>
        </div>
    </form>
</div>
{%endblock%}
```

==> templates/success.html <==

```
{% extends 'base.html' %}
{% block title %}
Result
{% endblock %}

{% block content %}
<h1>{{ msg }}</h1>
<a class="btn btn-primary" href="{{ url_for('home') }}">Home</a>
{% endblock %}
```

==> static/css/styles.css <==

```
.heading {
    text-align: center;
    display: inline-block;
}

body{
    /* background-image: url('{{ url_for(\'static\',
filename=\'img/bg.jpg\')}}'); */
    background-size: cover;
```

==> app.py <==

```
#!/bin/python3

from flask_mail import Mail, Message
import random
import string
from flask import Flask, render_template, request, redirect, url_for, session
import ibm_db


def Upper_Lower_string(length):
    result = ''.join((random.choice(string.ascii_uppercase) for x in
range(length)))
    return result


app = Flask(__name__)
app.config['SECRET_KEY'] = 'helloworld'
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=125f9f61-9715-46f9-9399-
c8177b21803b.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30426;SECURITY=
SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=wvn94274;PWD=2K5Z7ZiQuEV2e
dmQ", '', '')
app.config['MAIL_SERVER'] = 'smtp.gmail.com'
app.config['MAIL_PORT'] = 465
app.config['MAIL_USERNAME'] = 'shagish.111937@sxcce.edu.in'
app.config['MAIL_PASSWORD'] = 'SX@09/07/2002'
app.config['MAIL_USE_TLS'] = False
app.config['MAIL_USE_SSL'] = True
mail = Mail(app)


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


@app.route('/register', methods=['POST', 'GET'])
def register():
    if request.method == "POST":
        global rs
        name = request.form.get('name')
        email = request.form.get('email')
        password = request.form.get('password')
        stmt = ibm_db.prepare(conn, 'SELECT * FROM user WHERE username=?')
        ibm_db.bind_param(stmt, 1, name)
        ibm_db.execute(stmt)
        rs = ibm_db.fetch_assoc(stmt)
        print(rs)
        if rs:
            msg = 'Account already Exists'
            return render_template('register.html', msg=msg)
        else:
            reg_stmt = ibm_db.prepare(
                conn, 'INSERT INTO user ("USERNAME","EMAIL","PASSWORD")
VALUES(?,?,?)')
            ibm_db.bind_param(reg_stmt, 1, name)
            ibm_db.bind_param(reg_stmt, 2, email)
            ibm_db.bind_param(reg_stmt, 3, password)
            ibm_db.execute(reg_stmt)
            msg = 'Successfully Registered'
            return render_template('register.html', msg=msg)
    else:
        return render_template('register.html')
```

```
@app.route('/login', methods=['POST', 'GET'])
def login():
    if request.method == "POST":
        customer = list()
        agent = list()
        name = request.form['name']
        password = request.form['password']
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM user WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            session['role'] = 'user'
            session['customer'] = rs
            print(rs)
            return render_template('dashboard.html')
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM agent WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            cms = ibm_db.exec_immediate(conn, 'SELECT * FROM user')
            agt = ibm_db.exec_immediate(conn, 'SELECT * FROM agent')
            customers = ibm_db.fetch_assoc(cms)
            agents = ibm_db.fetch_assoc(agt)
            while customers:
                customer.append(customers)
                customers = ibm_db.fetch_assoc(cms)
            while agents:
                agent.append(agents)
                agents = ibm_db.fetch_assoc(agt)
            print(customer)
            print(agent)
            session['role'] = 'agent'
            session['name'] = rs['USERNAME']
            session['customer'] = customer
            session['agent'] = agent
            return render_template('dashboard.html')
        log_stmt = ibm_db.prepare(
            conn, 'SELECT * FROM admin WHERE username=? and password=?')
        ibm_db.bind_param(log_stmt, 1, name)
        ibm_db.bind_param(log_stmt, 2, password)
        ibm_db.execute(log_stmt)
        rs = ibm_db.fetch_assoc(log_stmt)
        if rs:
            cms = ibm_db.exec_immediate(conn, 'SELECT * FROM user')
            agt = ibm_db.exec_immediate(conn, 'SELECT * FROM agent')
            customers = ibm_db.fetch_assoc(cms)
            agents = ibm_db.fetch_assoc(agt)
            while customers:
                customer.append(customers)
                customers = ibm_db.fetch_assoc(cms)
            while agents:
                agent.append(agents)
                agents = ibm_db.fetch_assoc(agt)
            print(customer)
            print(agent)
            session['role'] = 'admin'
            session['customer'] = customer
            session['agent'] = agent
```

```
            return render_template('dashboard.html', agent=agent,
customer=customer)
        else:
            msg = 'UID/Password is incorrect'
            return render_template('login.html', msg=msg)
    else:
        return render_template('login.html')


@app.route('/dashboard', methods=['POST', 'GET'])
def dashboard():
    return render_template('dashboard.html')


@app.route('/success', methods=['POST', 'GET'])
def success():
    if request.method == "POST":
        ticket = session['ticket'] = Upper_Lower_string(16)
        print(ticket, session['ticket'])
        query = request.form['query']
        sql = "UPDATE user SET QUERY=?,TICKET=?,REVIEW_STATUS=0 WHERE
USERNAME=?"
        out = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(out, 1, query)
        ibm_db.bind_param(out, 2, session['ticket'])
        ibm_db.bind_param(out, 3, session['customer']['USERNAME'])
        status = ibm_db.execute(out)
        if status:
            msg = 'Success ! Your Ticket Nno is :', ticket, 'You can now return
to the home page'
            return render_template('success.html', msg=msg)
        else:
            msg = 'Error Submitting your Query'
            return render_template('success.html', msg=msg)


@app.route('/redirect')
def redir():
    return redirect(url_for('home'))


@app.route('/querying', methods=['POST'])
def admin_query():
    msg = ""
    agent = request.form.getlist('agent_name')
    usr_name = request.form.getlist('cus_name')
    emails = request.form.getlist('email')
    for i in range(0, len(agent)):
        if agent[i] != 'none':
            try:
                qr = ibm_db.prepare(
                    conn, "UPDATE USER SET ASSIGNED_AGENT=? WHERE USERNAME=?")
                ibm_db.bind_param(qr, 1, agent[i])
                ibm_db.bind_param(qr, 2, usr_name[i])
                result = ibm_db.execute(qr)
                print(agent[i], usr_name[i], emails[i])
                msg = Message(
                    f'Hello {usr_name[i]}',
                    sender='shagish.111937@sxcce.edu.in',
                    recipients=[f'{emails[i]}']
                )
                msg.body = f'Agent named {agent[i]} alloted to your query.
{agent[i]} will be responding you soon within 24 hrs.'
```

```
                mail.send(msg)
                msg = 'Allotments updated Successfully'
            except:
                msg = "Error saving allotments/sending emails"
    return render_template('done.html', msg=msg)


@app.route('/executing...', methods=['POST', 'GET'])
def agent_submit_reply():
    names = request.form.getlist('name')
    text = request.form.getlist('text')
    print(names)
    print(text)
    for i in range(0, len(names)):
        if not text[i] == '':
            try:
                sql = 'UPDATE USER SET REPLY=?,REVIEW_STATUS=1 WHERE USERNAME=?'
                query = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(query, 1, text[i])
                ibm_db.bind_param(query, 2, names[i])
                ibm_db.execute(query)

                msg = 'Replies sent successfully'
            except:
                msg = 'Error Sending replies'
    return render_template('done.html', msg=msg)


if __name__ == '__main__':
    app.run(debug=True, host="0.0.0.0")
```

==> **deployment.yaml** <==

```
 apiVersion: apps/v1
 kind: Deployment
 metadata:
   name: customercareregistry
 spec:
   replicas: 1
   selector:
     matchLabels:
       app: customercareregistry
   template:
     metadata:
       labels:
         app: customercareregistry
     spec:
       containers:
       - name: customercareregistry
         image: icr.io/bergin/customercareregistry:v1
         imagePullPolicy: Always
         ports:
         - containerPort: 5000
```

==> **service.yaml** <==

```
apiVersion: v1
 kind: Service
 metadata:
   name: ccr-deployment
 spec:
   ports:
   - port: 9000
```

```
    targetPort: 9000
  selector:
    app: customer_care_registry
```

==> IBM-DB2-schema.sql <==

```
--create the user table

CREATE TABLE USER (
USERNAME VARCHAR(32) NOT NULL,
EMAIL VARCHAR(32) NOT NULL,
PASSWORD    VARCHAR(16) NOT NULL,
QUERY VARCHAR(200),
REVIEW_STATUS     INTEGER,
TICKET      VARCHAR(16),
ASSIGNED_AGENT    VARCHAR(32),
REPLY VARCHAR(250)
);

--create the agent table

CREATE TABLE AGENT(
USERNAME VARCHAR(32),
ASSIGNED_CUSTOMERS VARCHAR(3000),
PASSWORD VARCHAR(16)
);

--create the admin table

CREATE TABLE ADMIN(
USERNAME VARCHAR(32),
PASSWORD VARCHAR(16)
);

--inserting to agent table

INSERT
 INTO  "WVN94274"."AGENT" ("USERNAME","ASSIGNED_CUSTOMERS","PASSWORD")
 VALUES(
'agent',                --USERNAME  VARCHAR(32)
NULL,                   --ASSIGNED_CUSTOMERS  VARCHAR(3000)
'agent'                 --PASSWORD  VARCHAR(16)
);

--inserting to admin table

INSERT
  INTO  "WVN94274"."ADMIN" ("USERNAME","PASSWORD")
  VALUES(
'admin',                --USERNAME  VARCHAR(32)
'admin'                 --PASSWORD  VARCHAR(16)
);
```

==> requirements.txt <==

```
ibm_db
flask
flask_mail
```

LINKS

GITHUB LINK: https://github.com/IBM-EPBL/IBM-Project-42259-1660657571

DEMO LINK: https://vimeo.com/772293484