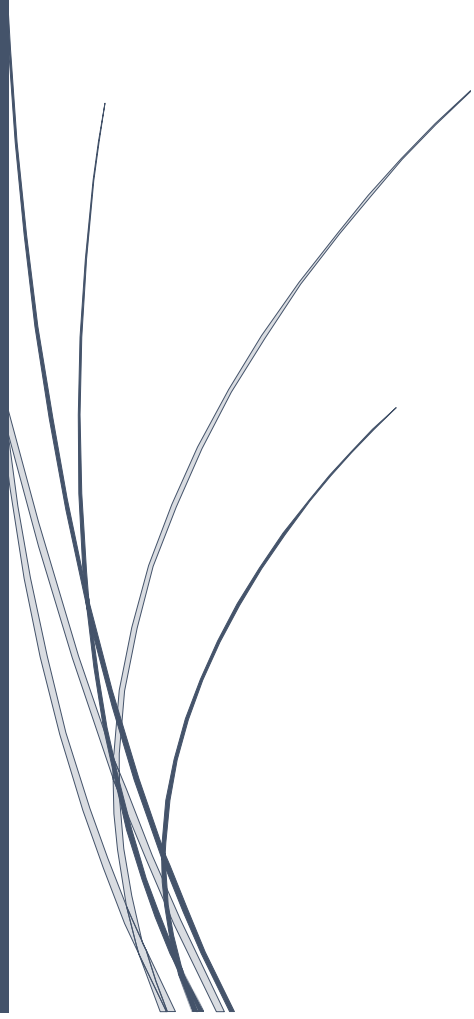
11/19/2022

PROJECT REPORT

Customer Care Registry



PNT2022TMID10745
IFET COLLEGE OF ENGINEERING

Customer Care Registry

Project Name	:	Customer Care Registry
Project Domain	:	Cloud Application Development
College	:	IFET COLLEGE OF ENGINEERING, VILLUPURAM
Team ID	:	**PNT2022TMID10745**
Team Size	:	4
Team Members	:	[1] GAYATHRI JD [2] CHITRA D [3] LOGAPRIYA P [4] LOGITHA K
Team Mentor	:	Mrs. GOMATHI R
GitHub Link	:	https://github.com/IBM-EPBL/IBM-Project-20286-1659716511

TABLE OF CONTENTS

1. INTRODUCTION

1.1 PROJECT OVERVIEW

Short Description:

This 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.

Admin: 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.

User: They can register for an account. After the login, they can create the complaint with a description of the problem they are facing. Each user will be assigned with an agent. They can view the status of their complaint.

1.2 PURPOSE

The purpose of the whole project is to:

- Provide a common platform to the customers to clarify their queries
- Having expert agents in the platform for better answering
- Customer's tickets (queries) are answered quickly by the agents
- Customers and Agents can chat with one another for better understanding
- While doing so, the former asks questions
- Later, answers those questions as quickly and as legitimately as possible
- Customers can raise as many tickets as they want
- Customers and Agents can also submit their feedbacks to the Admin, for the betterment of the platform

2. LITERATURE SURVEY

2.1 Existing Problem

- The ratings and reviews on e-commerce websites are unreliable.
- Furthermore, they are frequently provided by the manufacturers themselves.
- Reviews are not provided by real people.
- After purchasing the goods, I am without a way to get my questions answered.
- We, the customers, do not have access to a unified platform to get our questions answered.
- If it exists, we are not receiving prompt responses. By the time the response is sent, the problem might have been solved or may not have been worth solving for the clients.

2.2 References

https://doi.org/10.3115/v1/D14-1181

https://doi.org/10. 18653/v1/N19-1423

https://doi.org/10.1145/3511808.3557718

https://doi.org/10.29244/jcs.7.1.68-82

2.3 Problem Statement Definition

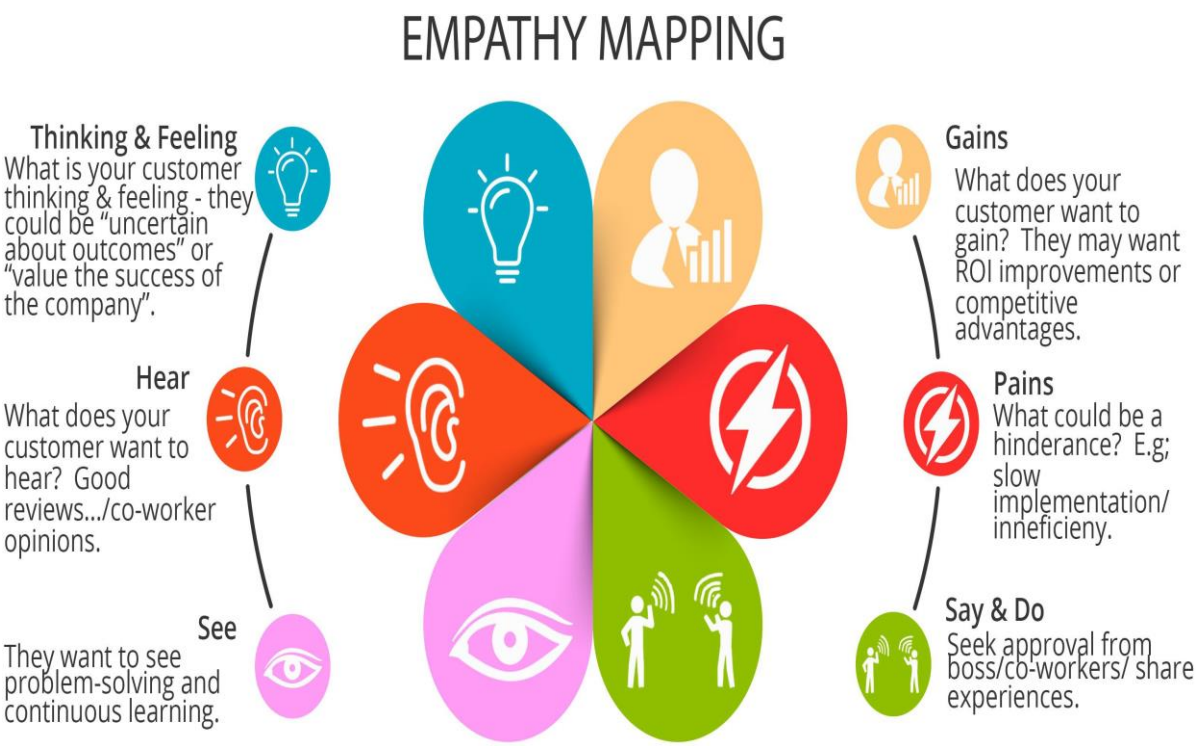
I frequently shop at well-known e-commerce sites like Meesho. I often place orders. The issue I have now is that I frequently lack access to trustworthy sources to allay my concerns over some of the things I purchase.

These websites provide reviews and consumer ratings, but I don't think they are genuine or legitimate. If the responses were from a true professional and I could get all of my questions answered on one platform, that would rock my world. Obviously, I should require prompt responses from a true authority who is familiar with the goods I am requesting.

3. IDEATION AND PROPOSED SOLUTION

3.1 Empathy Map Canvas

- Empathy Map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes
- It is a useful tool to help teams to better understand their users
- Creating an effective solution requires understanding the true problem and the person who us 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 and Brainstorming

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome and built upon, and all participants are encouraged to collaborate, helping each other develop a rich number of creative solutions

Step-1: Team Gathering, Collaboration and Select the Problem Statement

Team Gathering:

Team Members	
Team Leader	Gayathri JD
Team Members	Chitra D
	Logitha K
	Logapriya P

Problem Statement:

Customer seek a resolution to their complaint because the response time was too long.

Customer service agents do not pay close attention to what the customer requires.

The majority of the times, customer are unable to talk with a real human, alternatively with automated chats.

Customers do not know the current status of problem and at present how much percent is completed.

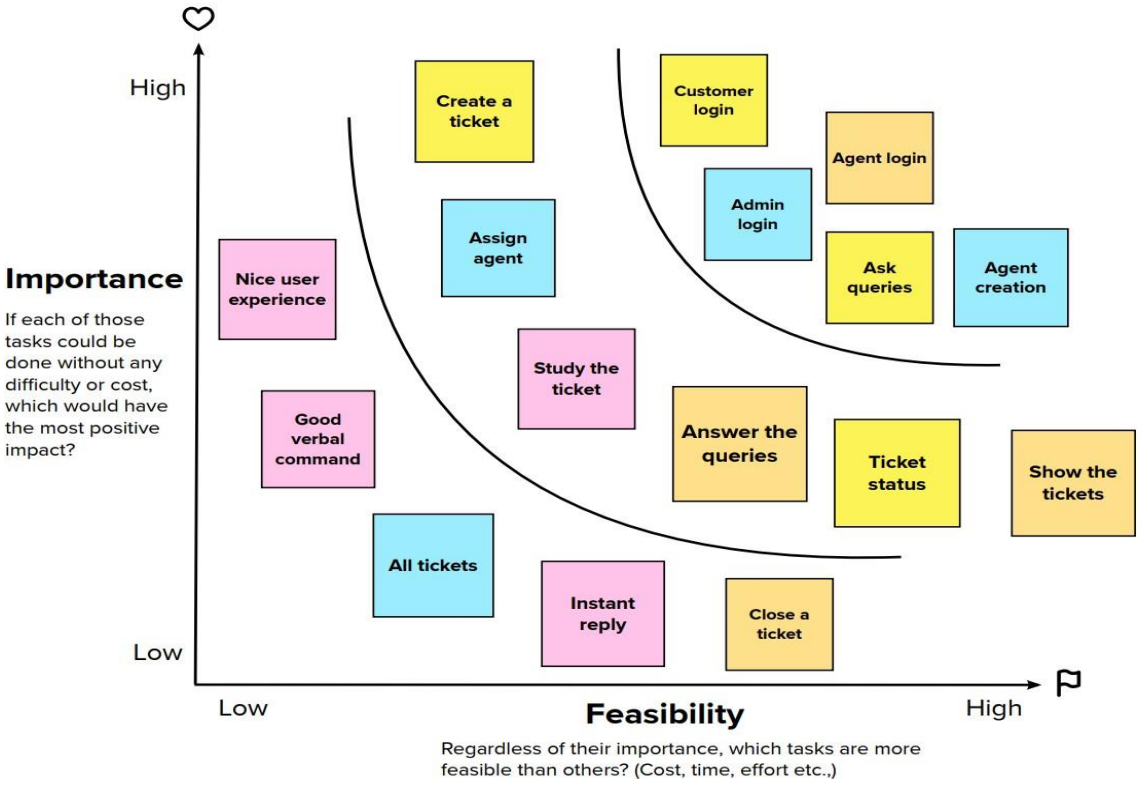
The customer needs a method to set up an account so they can log in and report a problem.

For example, customer who perform their banking on their smart phones complain that entering the data for each bill is time-consuming, challenging, and susceptible to mistakes. So we caneasily help the customers by using the application.

Step-2: Brainstorm, Idea Listing and Grouping

Step-3: Idea Prioritization

Prioritization



3.3 Proposed Solution

S. No.	Parameter	Description
•	Problem Statement (Problem to be solved)	The process of entering the data for each bill is time-consuming, challenging, and prone to mistakes, according to customers who use their mobile phones for banking. As a result, we can easily assist clients by using the application.
•	Idea / Solution description	This application's goal is to help customers resolve their issues. Customers can submit a ticket outlining the issue in detail. An Agent will be assigned to the Customer to handle the problem. Each time the agent is assigned to that person, an email notification will be sent to the customer. Customers can check the status of their tickets before the service is provided.
•	Novelty / Uniqueness	Routing of the assigned agent, an automated system for closing tickets, displaying the customer's status, and data backup in case of errors.
•	Social Impact / Customer Satisfaction	The assumption that businesses know what their customers want is improper. Instead, it's important to appreciate the viewpoint of the client by using methods like polling, focus groups, and customer surveys. By adopting these techniques to gain in-depth insights into what their customers want, businesses may better alter their services and products to meet or surpass client expectations.
•	Business Model (Revenue Model)	● Third-party apps, agents, and clients are examples of Key Partners. ● Cost Structure Identifies Offices and Cloud Platforms. ● Activities held as Customer Service, System Maintenance. ● Key Resources support Engineers, Multi-channel. ● Knowledge -based channels and 24 -hour email support are provided for customer relationships.
•	Scalability of the Solution	The main goal of scaling customer service is to create an environment that will allow your customer service employees to function as effectively as possible. A situation where they can devote more of their time to resolving crucial client issues rather than performing tedious activities.

3.4 Proposed Solution Fit

1. CUSTOMER SEGMENT

- Customers who are unable to understand the answers to their enquiries and not able to solve their own issues.
- Customers do not know the current status of problem and at present how much percent is completed.

2. CUSTOMER CONSTRAINT

- If costs surpass the specified limit, the suggested fix will have a feature that will send an email notice or notification
- The mobile app we provide is adapt to any device, easy to understand and access by anyone.

3. AVAILABLE SOLUTIONS

- By giving suggestion properly and make them clear.
- By Go through the guidelines properly before complains.

4. PROBLEMS

- Customers seek a speedy resolution to their complaint because the response time was too long.
- The majority of the times, customers are unable to talk with the real human, alternatively with automated chats.

5. ROOT / CAUSE

- Most of the customer does not know the solution for the problem.
- Some of them are not properly reading the firm guidelines.

6. BEHAVIOUR

- Ensure that they receive the appropriate response to their questions.
- Make sure the regularly check the email for latest updates for solution of the problem.

7. TRIGGERS

Customer have to learn how to solve their problem and make themselves aware of the problems.

8. EMOTIONS

Once the problem occurs, the customer need to approach the helpdesk and know the prerequisites of the issue.

9. SOLUTIONS

- To offer visuals about the answers to their enquiries.
- Customers can raise a ticket outlining the issue in detail. An Agent will be assigned to the Customer to handle the problem.
- Each time the agent is engaged to that customer, an email notification will be sent to the customer.

10. CHANNELS OF BEHAVIOR

- To ensure high security data, that keeping customer data is secured manner and it is stored in the cloud services.
- To make sure that they will aware of the solutions for the problem and issues are fully cleared.

4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

- A functional requirement establishes the relationship between the behaviour of inputs and outputs in a system or a system component. It specifies “what should the software system do?”
- Specified at the component system
- Easy to establish

S. No	Functional requirements	Description
1	Guidelines	The steps for registration have been conveyed to customer through message or pdf files.
2	Registration Process	Registration is done by following ways: ➢ Google Forms ➢ Application ➢ Official links
3	Conformation Process	• OTP confirmation for verification • Confirmation of registration sends through emails and notification in the apps.
4	Account	Created account is login through the customer user id and password.
5	Documentation	The issues and the requirement of the customer is stated in their account and customer requested to submit their response for issue clearance.
6	Current status	Customer can follow their status of request through “Track flow” option in their account.
8	Feedback	For further queries and suggestions need to fill feedback form or convey through mails.

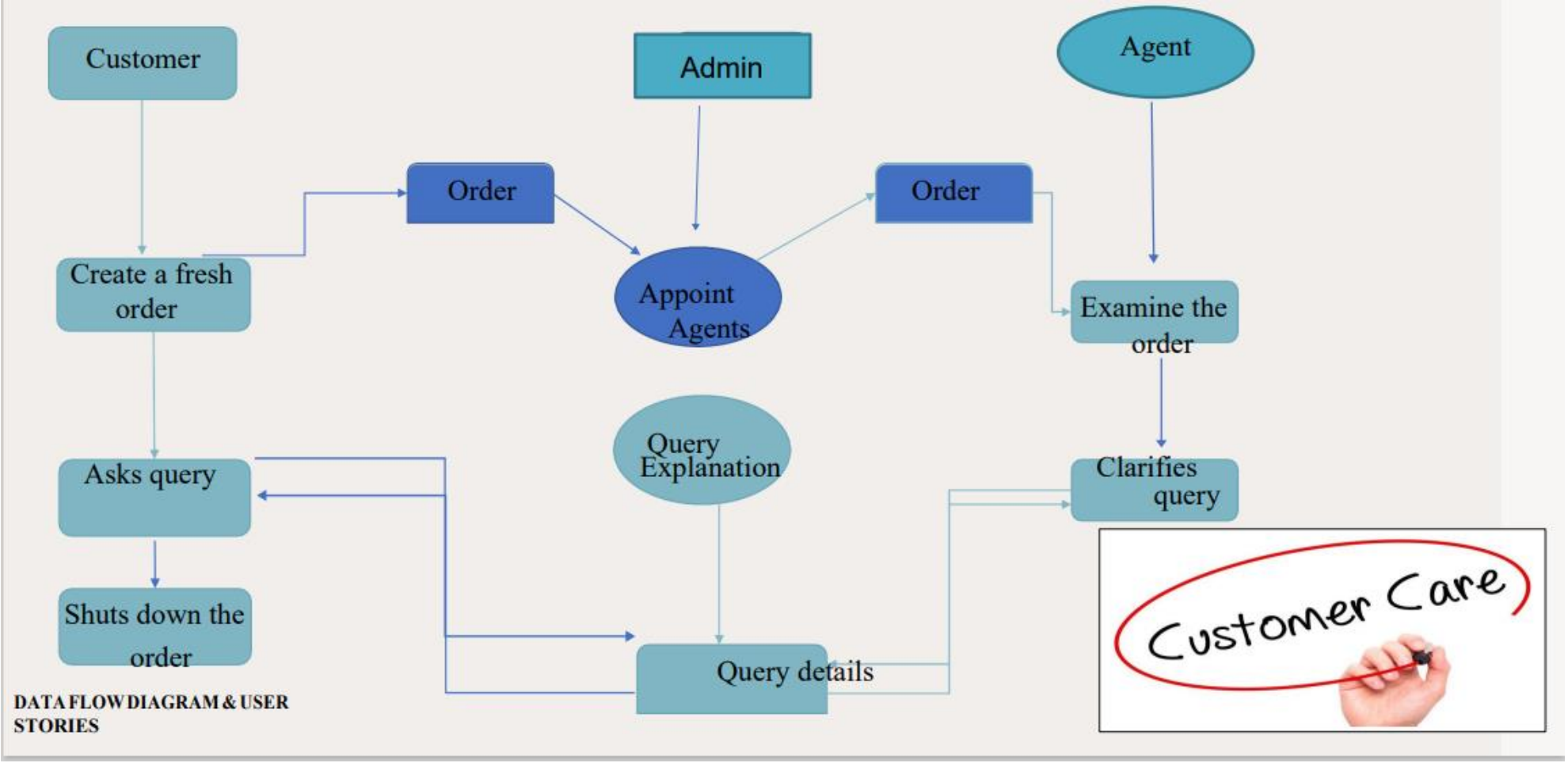
4.2 Non-functional Requirements

- A non-functional requirement identifies a software system's quality characteristic.
- It is not necessary
- It imposes limitations on the question of "How should the software system satisfy the functionality?
- It apply to entire system

S. No	Non-Functional requirements	Description
1	Security	Customer information is highly secured and comes under security policy.
2	Compatibility	Mutual response between the customer and the agent via phone calls and messages.
3	Usability	Guidelines and solution have been given in a simple manner.
4	Flexibility	Based on the severity of issues and the work load
5	Reliability	Updating the current status of issue through mail or notification in the apps.
6	Availability	Service available - 24/7.
7	Maintainability	Immediate response by the agent through email or calls and fix the problem within 24 hours.

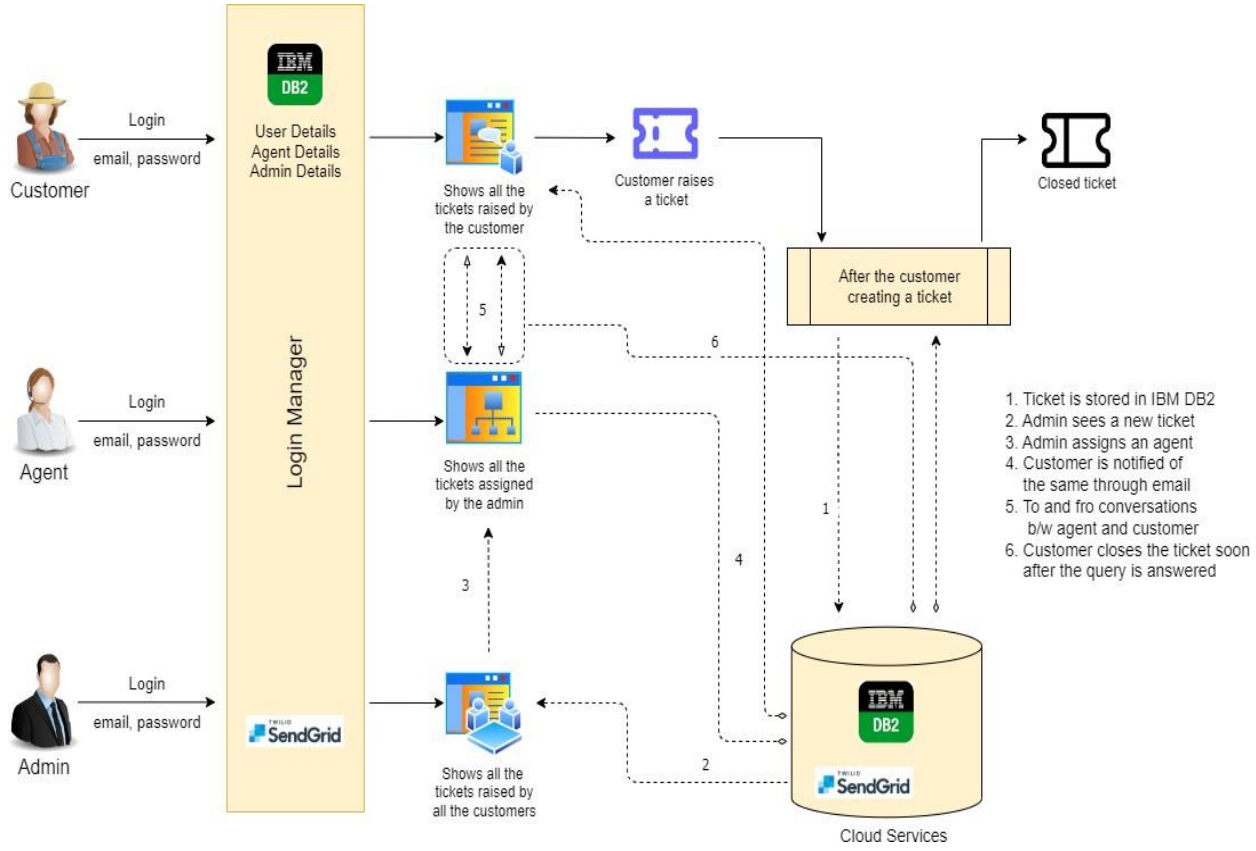
5. PROJECT DESIGN

5.1 Dataflow Diagram

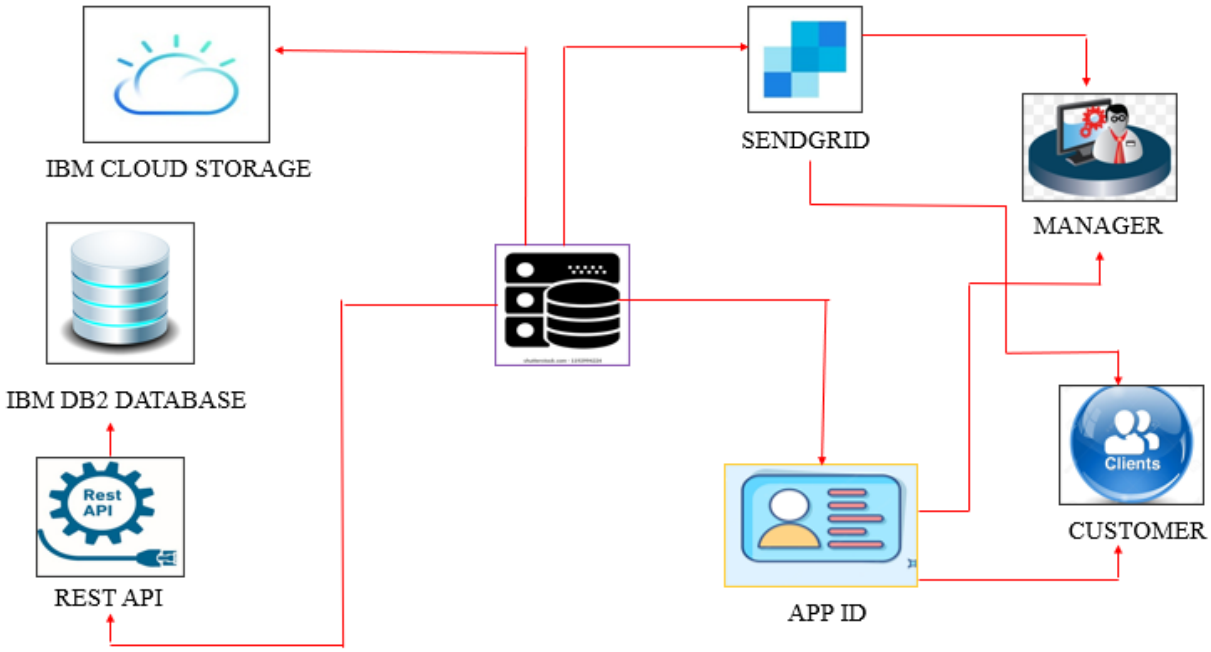


5.2 Solution and Technical Architecture

Solution Architecture



Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	By providing my email address, a password, and a password confirmation, I can sign up for the application as a customer.	I can access my dashboard or account.	High	Sprint-1
	Login	USN-2	I can access the application as a customer by entering my correct email address and password.	I can access my dashboard or account.	Low	Sprint-2
	Dashboard	USN-3	I can view every order I've placed as a customer.	My dashboard provides all the necessary information.	Medium	Sprint-2
	Ticket creation	USN-4	I can place my order as a customer by describing my inquiry in full.	I can make a request.	High	Sprint-3
	Address Column	USN-5	I may speak with the designated agent and get my questions answered as a client.	My concerns have been answered.	Low	Sprint-3
	Forgot password	USN-6	If I forget my old password, I can reset it using this option as a client.	My account is accessible once more.	Low	Sprint-4
	Ticket details	USN-7	I can view the most recent order statistics as a customer.	I am more able to grasp	High	Sprint-3
Agent (Web user)	Login	USN-1	I may access the application as an agent by entering the proper email address and password.	I can get to my dashboard or account.	Medium	Sprint-3
	Dashboard	USN-2	I can view the order details that admin has given me as an agent.	I am able to see the tickets that require responses.	Low	Sprint-4

	Address Column	USN-3	As an agent, I get to speak with the client and address his or her concerns.	I can explain the problems.	High	Sprint-4
	Forgot password	USN-4	In the event that I forget my old password, I can use this option to reset it as an agent.	I can access my account once more..	Low	Sprint-4
Admin (Web user)	Login	USN-1	As an administrator, I may access the application by entering the proper email address and password.	I can access my dashboard and account.	High	Sprint-1
	Dashboard	USN-2	As an administrator, I have access to all orders submitted across the board and much more.	By seeing that order, I can allocate agents.	High	Sprint-2
	Agent creation	USN-3	As an administrator, I can designate an agent to answer customers' questions.	I'm able to make agents.	High	Sprint-2
	Assigning agent	USN-4	I may assign an agent to each order that a customer creates in my capacity as an admin.	Allow the agent to elaborate on the queries.	High	Sprint-1
	Forgot password	USN-5	In the event that I forget my old password, I can reset it as an admin using this option.	I am able to log into my account.	High	Sprint-2

6. PROJECT DESIGN AND PLANNING

6.1 Sprint Planning and Estimation

To design a product backlog and sprint schedule, use the table below

Sprint	Functional Requirement	User Story Number	Task	Story Points	Priority	Team Members
Sprint-1	User Panel	USN-1	In order to access the website's features, the customer must first log in.	20	High	LOGAPRIYA P LOGITHA K
Sprint-2	Admin panel	USN-2	The administrator's job is to monitor the database's functionality and keep track of everything that consumers are getting to serve.	20	High	CHITRA D GAYATHRI JD
Sprint-3	Chat Bot	USN-3	The customer can speak with the chatbot directly to inquire about the services. Obtain suggestions based on the identity data.	20	High	CHITRA D LOGAPRIYA P
Sprint-4	final delivery	USN-4	Application deployment and server containers for apps. Make the necessary data, then submit the form.	20	High	LOGITHA K GAYATHRI JD

6.2 Sprint Delivery Plan

Sprint	Total StoryPoints	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	26 Oct 2022	31 Oct 2022		31 Oct 2022
Sprint-2	20	6 Days	2 Nov 2022	7 Nov 2022		7 Nov 2022
Sprint-3	20	6 Days	9 Nov 2022	14 Nov 2022		14 Nov 2022
Sprint-4	20	6 Days	16 Nov 2022	19 Nov 2022		21 Nov 2022

Velocity:

Consider a scenario in which the sprint will last 10 days and the team's velocity is 20. Let's determine the group's average velocity (AV) for each iteration.

$$\text{Average Velocity} = \frac{\text{Sprint Duration}}{\text{Velocity}} = 20/10 = 2$$

6.3 Reports from JIRA

Sprint 1 – Burndown Chart

Sprint 2 – Burndown Chart

Sprint 3 – Burndown Chart

Sprint 4 – Burndown Chart

7. CODING AND SOLUTIONING

7.1 Admin assigning an agent to a ticket

Code:

```
@admin.route('/admin/update/<agent_id>/<ticket_id>')
@login_required
def assign(agent_id, ticket_id):
    '''
        Assigning an agent to the ticket
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # query to update the ASSIGNED_TO of a ticket
        assign_agent_query = '''
            UPDATE tickets SET assigned_to = ? WHERE ticket_id = ?
        '''

        stmt = ibm_db.prepare(conn, assign_agent_query)
        ibm_db.bind_param(stmt, 1, agent_id)
        ibm_db.bind_param(stmt, 2, ticket_id)

        ibm_db.execute(stmt)

        return "None"

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))
```

Explanation:

- User creates a ticket by describing the query
- Admin views the newly created ticket in the dashboard
- In the dropdown given, admin selects an agent
- Once selected, using fetch() the request is sent to the server
- The request URL contains both the Ticket ID and the selected Agent ID
- Using the shown SQL query, the assigned_to column of the tickets table is set to agent_id where the ticket_id column = ticket_id
- Then, the dashboard of the admin gets refreshed

7.2 Customer closing a ticket

Code:

```
@cust.route('/customer/close/<ticket_id>/')
@login_required
def close(ticket_id):
    '''
        Customer can close the ticket
        :param ticket_id ID of the ticket that should be closed
    '''
    from .views import customer

    if(hasattr(customer, 'uuid')):
        # query to close the ticket
        close_ticket = '''
            UPDATE tickets SET query_status = ? WHERE ticket_id = ?
        '''

        stmt = ibm_db.prepare(conn, close_ticket)
        ibm_db.bind_param(stmt, 1, "CLOSED")
        ibm_db.bind_param(stmt, 2, ticket_id)
        ibm_db.execute(stmt)

        return redirect(url_for('customer.tickets'))

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))
```

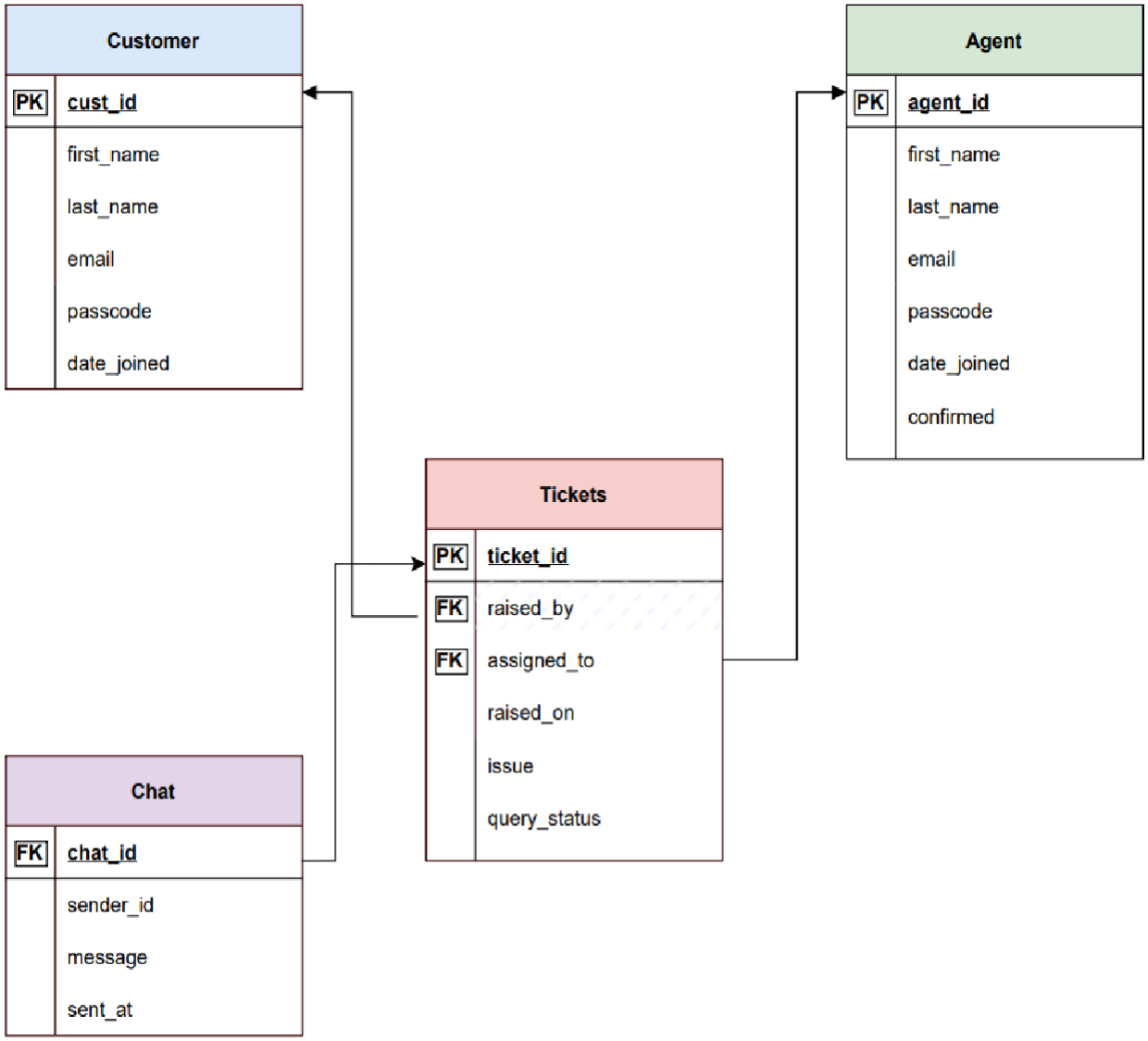
Explanation:

- User creates a ticket by describing the query
- Admin assigns an agent to this ticket
- The customer and the agent, chat with each other, in the view of clearing the customer's doubts
- Once the customer is satisfied, the customer decides to close the ticket
- Using fetch() the request is sent to the server. The requested URL contains the Ticket ID
- Using the shown SQL query, the status of the ticket is set to "CLOSED"
- Thus the ticket is closed
- Then the customer gets redirected to the all-tickets page

7.3 Database Schema

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



8. TESTING

8.1 Test Cases

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Test Cases Performed:

1. Sprint 1

   Click Here
2. Sprint 2

   Click Here
3. Sprint 3

   Click Here
4. Sprint 4

   Click Here
5. Test Cases Report

   Click Here

8.2 User Acceptance Testing

1. Purpose of Document

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

2. Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	5	0	0	2	7
External	0	2	0	0	2
Fixed	12	11	35	45	103
Not Reproduced	0	5	0	0	5
Skipped	0	0	0	0	0
Totals	17	18	35	47	117

3. Test Case Analysis

This report shows the number of test cases that have passed, failed, and untested

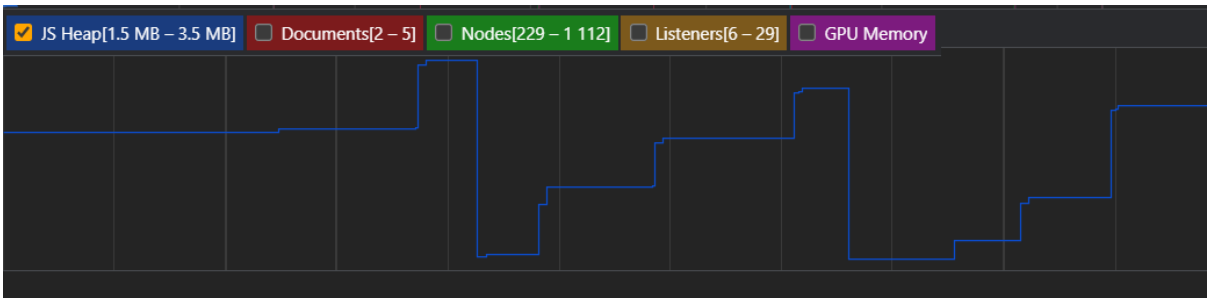
Section	Total Cases	Not Tested	Fail	Pass
Client Application	72	0	0	72
Security	7	0	0	7
Exception Reporting	5	0	0	5
Final Report Output	4	0	0	4

9. RESULTS

9.1 Performance Metrics:

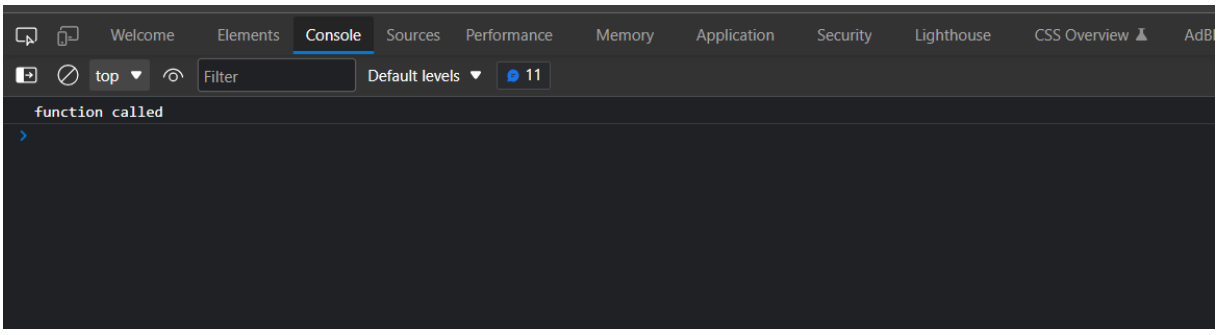
CPU usage:

- ✓ Since all the operations run using Flask is in server-side, the client (browser) need not worry about the CPU usage. Just rendering the page, static contents take place in the client-side.
- ✓ Memory for client-side functions (Javascript) is allocated using heap. It can be either increased based upon the requirement or removed from the heap.



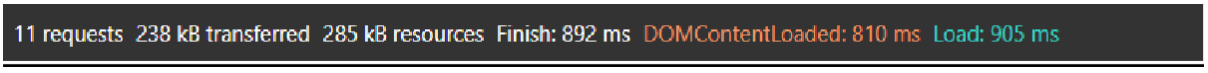
Errors:

- ✓ Since all the backend functions are done using flask, any exceptions / errors rising are well-handled. Though they appear, user's interaction with the site is not affected in any way



Latency and Response time:

It takes less than a second to load a page in the client. From this it is evident that there is low latency



10. ADVANTAGES AND DISADVANTAGES

Advantages:

- ✓ Customers can clarify their doubts just by creating a new ticket
- ✓ Customer gets replies as soon as possible
- ✓ Not only the replies are faster, the replies are more authentic and practical
- ✓ Customers are provided with a unique account, to which the latter can login at any time
- ✓ Very minimal account creation process
- ✓ Customers can raise as many tickets as they want
- ✓ Application is very simple to use, with well-known UI elements
- ✓ Customers are given clear notifications through email, of all the processes related lo login, ticket creation etc.,
- ✓ Customers' feedbacks are always listened
- ✓ Free of cost

Disadvantages:

- × Only web application is available right now (as of writing)
- × UI is not so attractive, it's just simple looking
- × No automated replies
- × No SMS alerts
- × Supports only text messages while chatting with the Agent
- × No tap to reply feature
- × No login alerts
- × Cannot update the mobile number
- × Account cannot be deleted, once created
- × Customers cannot give feedback to the agent for clarifying the queries

11. CONCLUSION

Thus, there are many customer service applications available on the internet. Noting down the structural components of those applications and we built a customer care registry application. It will be a web application build with Flask (Python micro-web framework), HTML, JavaScript. It will be a ticket-based customer service registry.

Customers can register into the application using their email, password, first name and last name. Then, they can login to the system, and raise as tickets as they want in the form of their tickets.

These tickets will be sent to the admin, for which an agent is assigned. Then, the assigned agent will have a one-to-one chat with the customer and the latter's queries will be clarified. It is also the responsibility of the admin, to create an agent.

12. FUTURE SCOPE

Our application is not finished yet. There are many rooms for improvement. Some of them will be improved in the future versions

- ✓ Attracting and much more responsive UI throughout the application
- ✓ Releasing cross-platform mobile applications
- ✓ Incorporating automatic replies in the chat columns
- ✓ Deleting the account whenever customer wishes to
- ✓ Supporting multi-media in the chat columns
- ✓ Creating a community for our customers to interact with one another
- ✓ Call support
- ✓ Instant SMS alerts

13. APPENDIX

Flask:

- ✓ Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.
- ✓ It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions

JavaScript:

- ✓ JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS
- ✓ As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, often incorporating third-party libraries

IBM Cloud:

- ✓ IBM cloud computing is a set of cloud computing services for business offered by the information technology company IBM

Kubernetes:

- ✓ Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management

Docker:

- ✓ Docker is a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers

SOURCE CODE (Only Samples)

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=1.0">
    <title>{% block title %}{% endblock %}</title>
    <link rel="icon" type="image" href="{{ url_for('static', filename='images/cart logo white-modified.png') }}">

    <!-- Linking css, js, Google fonts -->
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link rel="stylesheet" href="{{ url_for('static', filename='css/style.css') }}"/>
    <link
href="https://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400
;1,500;1,700;1,900&display=swap" rel="stylesheet">
    <script src="{{ url_for('static', filename='js/pass.js') }}"></script>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

    <!-- Linking Watson Assistant -->
    {% block watson %}
    {% endblock %}
</head>
<body>
    {% block alert %}
        {% if to_show %}
            <script>
                alert('{{ message }}')
            </script>
        {% endif %}
    {% endblock %}

    {% block main %}
    {% endblock %}
</body>
</html>
```

login.html:

```
{% extends 'base.html' %}


{% block title %}
    Login
{% endblock %}


{% block main %}
    <div class="bg-main-div">
        <section class="login-section">
            <div class="login-div">
                <div class="login-header">
                    <img src="{{ url_for('static', filename='images/cart logo white.png') }}" class="login-img" alt="logo" />
                    <h2>Sign in</h2>
                    <p>Use your Registry Account</p>
                </div>
                <div class="login-remind">
                    <form action="{{ url_for('blue_print.login') }}" method="POST" class="login-form">
                        <label>Email</label>
                        <input type="email" required value="{{ email }}" name="email" placeholder="Enter your email"/>


                        <label>Password</label>
                        <input type="password" required value="{{ password }}" name="password" id="password-input"
placeholder="Enter your password"/>


                        <div class="show-pass-div">
                            <input type="checkbox" onclick="showPassword()" style="height: 20px;"/>
                            <p>Show Password</p>
                        </div>


                        <div class="role-div">
                            <p>Role : </p>
                            <div>
                                <div>
                                    <input type="radio" style="height: 20px;" value="Customer" checked name="role-check"/>
                                    <p>Customer</p>
                                </div>
                                <div>
                                    <input type="radio" style="height: 20px;" value="Agent" name="role-check"/>
                                    <p>Agent</p>
                                </div>
```

```
                </div>
            </div>

            <button class="submit-btn" type="submit">Login</button>

            <div>
                <!-- {{ url_for('blue_print.forgot') }} -->
                <a href="{{ url_for('blue_print.forgot') }}" class="links">Forgot Password?</a> <br>
                <div>
                    <a href="{{ url_for('blue_print.register') }}" class="links">Don't have an account yet? Register</a>
                </div>
            </div>
        </form>
    </div>
  </div>
 </section>
</div>
{% endblock %}
```

address.html:

```
{% extends 'base.html' %}

{% block title %}
  Address Column
{% endblock %}

{% block main %}
  <div class="dashboard-div">
    <nav>
      <div class="dash-nav">
        <div>
          <div class="dash-img-text">
            {% if user == "AGENT" %}
              <a href="{{ url_for('agent.assigned') }}">
                <i class="fa fa-arrow-left" aria-hidden="true"></i>
              </a>
              <img src="{{ url_for('static', filename='images/cust profile.png') }}" class="img-in-nav" alt="logo"/>
            {% else %}
              <a href="{{ url_for('customer.tickets') }}">
                <i class="fa fa-arrow-left" aria-hidden="true"></i>
```

```
                </a>
                <img src="{{ url_for('static', filename='images/agent.png') }}" class="img-in-nav" alt="logo"/>
            {% endif %}
            <h3>{{ name }}</h3>
        </div>
    </div>
    <div>
        <div style="align-items: center;">
            {% if value == "True" %}
                {% if user == "CUSTOMER" %}
                    <a href="/customer/close/{{ id }}"><button class="logout-btn">CLOSE TICKET</button></a>
                {% endif %}
            {% endif %}
        </div>
    </div>
  </div>
</nav>

<div class="chat-body">
  <div class="chat-contents" id="content">
    {% if msgs_to_show %}
      {% for chat in chats %}
        {% if chat['SENDER_ID'] == sender_id %}
          <div class="message-sent">{{ chat['MESSAGE'] }}</div>
        {% else %}
          <div class="message-sent received">{{ chat['MESSAGE'] }}</div>
        {% endif %}
      {% endfor %}
    {% endif %}
  </div>
  <div class="chat-input-div">
    {% if value == "True" %}
      <form method="POST" action="{{ post_url }}">
        <input name="message-box" class="chat-input" type="text" placeholder="Type something" required/>
        <button type="submit" class="chat-send">
          <i class="fa fa-paper-plane-o" aria-hidden="true"></i>
        </button>
      </form>
    {% else %}
      <div>
        {% if user == "CUSTOMER" %}
```

```
                    <h4>You closed this ticket. Chats are disabled</h4>
                {% else %}
                    <h4>{{ name }} closed this ticket. Chats are disabled</h4>
                {% endif %}
            </div>
        {% endif %}
    </div>
  </div>
</div>
{% endblock %}
```

chat.py:

```
from flask import render_template, Blueprint, request, session, redirect, url_for
import ibm_db
from datetime import datetime
import time

chat = Blueprint("chat_bp", __name__)

@chat.route('/chat/<ticket_id>/<receiver_name>/', methods = ['GET', 'POST'])
def address(ticket_id, receiver_name):
    '''
        Address Column - Agent and Customer chats with one another

        : param ticket_id ID of the ticket for which the chat is being opened
        : param receiver_name Name of the one who receives the texts, may be Agent / Customer
    '''
    # common page for both the customer and the agent
    # so cannot use login_required annotation
    # so to know who signed in, we have to use the session
    user = ""
    sender_id = ""
    value = ""
    can_trust = False
    post_url = f'/chat/{ticket_id}/{receiver_name}/'

    if session['LOGGED_IN_AS'] is not None:
        if session['LOGGED_IN_AS'] == "CUSTOMER":
            # checking if the customer is really logged in
            # by checking, if the customer has uuid attribute
```

```
    from .views import customer

    if(hasattr(customer, 'uuid')):
        user = "CUSTOMER"
        sender_id = customer.uuid
        can_trust = True

    else:
        # logging out the so called customer
        return redirect(url_for('blue_print.logout'))

elif session['LOGGED_IN_AS'] == "AGENT":
    # checking if the agent is really logged in
    # by checking, if the agent has uuid aatribute
    from .views import agent

    if (hasattr(agent, 'uuid')):
        user = "AGENT"
        sender_id = agent.uuid
        can_trust = True

else:
    # Admin is the one who logged in
    # admin should not see the chats, sp directly logging the admin out
    return redirect(url_for('blue_print.logout'))

to_show = False
message = ""

if can_trust:
    # importing the connection string
    from .views import conn

    if request.method == 'POST':
        # chats are enabled, only if the ticket is OPEN
        # getting the data collected from the customer / agent
        myMessage = request.form.get('message-box')

        if len(myMessage) == 0:
            to_show = True
            message = "Type something!"
```

```
    else:
        # inserting the message in the database

        # query to insert the message in the database
        message_insert_query = '''
            INSERT INTO chat
                (chat_id, sender_id, message, sent_at)
            VALUES
                (?, ?, ?, ?)
        '''

        try:
            stmt = ibm_db.prepare(conn, message_insert_query)
            ibm_db.bind_param(stmt, 1, ticket_id)
            ibm_db.bind_param(stmt, 2, sender_id)
            ibm_db.bind_param(stmt, 3, myMessage)
            ibm_db.bind_param(stmt, 4, datetime.now())

            ibm_db.execute(stmt)

        except:
            to_show = True
            message = "Please send again!"

    return redirect(post_url)

else:
    # method is GET
    # retrieving all the messages, if exist from the database
    msgs_to_show = False

    # query to get all the messages for this ticket
    get_messages_query = '''
        SELECT * FROM chat
            WHERE chat_id = ?
        ORDER BY sent_at ASC
    '''

    # query to check if the ticket is still OPEN
    query_status_check = '''
```

```
    SELECT query_status FROM tickets WHERE ticket_id = ?
'''

try:
    # first checking if the ticket is OPEN
    check = ibm_db.prepare(conn, query_status_check)
    ibm_db.bind_param(check, 1, ticket_id)
    ibm_db.execute(check)

    value = "True" if ibm_db.fetch_assoc(check)['QUERY_STATUS'] == "OPEN" else "False"

    # getting all the messages concerned with this ticket
    stmt = ibm_db.prepare(conn, get_messages_query)
    ibm_db.bind_param(stmt, 1, ticket_id)
    ibm_db.execute(stmt)

    messages = ibm_db.fetch_assoc(stmt)
    messages_list = []

    while messages != False:
        messages_list.append(messages)
        print(messages)

        messages = ibm_db.fetch_assoc(stmt)

    # then some messages exist in this chat
    if len(messages_list) > 0:
        msgs_to_show = True

    elif len(messages_list) == 0 and value == "True":
        # ticket is OPEN
        # but no messages are sent b/w the customer and the agent
        msgs_to_show = False
        to_show = True
        message = f'Start the conversation with the {"Customer" if user ==  "AGENT" else "Agent"}'

except:
    to_show = True
    message = "Something happened! Try Again"

return render_template(
```

```
                'address.html',
                to_show = to_show,
                message = message,
                id = ticket_id,
                chats = messages_list,
                msgs_to_show = msgs_to_show,
                sender_id = sender_id,
                name = receiver_name,
                user = user,
                post_url = post_url,
                value = value
            )

    else:
        # logging out whoever came inside the link
        return redirect(url_for('blue_print.logout'), user = user)
```

__init__.py:

```
from flask import Flask, session
from flask_login import LoginManager

def create_app():
    app = Flask(__name__)
    app.config['SECRET_KEY'] = "PHqtYfAN2v@CCR2022"

    # registering the blue prints with the app
    from .routes.views import views
    app.register_blueprint(views, appendix='/')

    from .routes.cust import cust
    app.register_blueprint(cust, appendix='/customer/')

    from .routes.admin import admin
    app.register_blueprint(admin, appendix='/admin/')

    from .routes.agent import agent
    app.register_blueprint(agent, appendix='/agent/')

    from .routes.chat import chat
    app.register_blueprint(chat, appendix='/chat/')
```

```
    # setting up the login manager
    login_manager = LoginManager()
    login_manager.login_view = "blue_print.login"
    login_manager.init_app(app)

    @login_manager.user_loader
    def load_user(id):
        if session.get('LOGGED_IN_AS') is not None:
            if session['LOGGED_IN_AS'] == "CUSTOMER":
                from .routes.views import customer

                if hasattr(customer, 'first_name'):
                    return customer

            elif session['LOGGED_IN_AS'] == "AGENT":
                from .routes.views import agent

                if hasattr(agent, 'first_name'):
                    return agent

            elif session['LOGGED_IN_AS'] == "ADMIN":
                from .routes.views import admin

                if hasattr(admin, 'email'):
                    return admin

        else:
            return None

    return app
```

GITHUB LINK

Github Rep Link:
https://github.com/IBM-EPBL/IBM-Project-20286-1659716511