CUSTOMER CARE REGISTRY

TEAM ID : PNT2022TMID06484
BATCH NUMBER : B4-4M6E
DOMAIN : CLOUD APPLICATION DEVELOPMENT

TEAM MEMBERS :

Team Leader	JANAKRISHNAMOORTHY R
Team Member 1	JIBIN M
Team Member 2	SANJAY S
Team Member 3	THANUSH M

1. INTRODUCTION

1.1 PROJECT OVERVIEW

Customer care and customer service together help create a positive customer experience, or the overall impression a person has when interacting with your company. Both are vital, but there are subtle differences in how they are implemented. High-quality customer care is proactive. The needs of customers throughout the buyer's journey are anticipated, making customers feel supported. That, in turn, helps create an emotional connection between the customer and the company. Customer service is reactive. Here, the focus is on helping customers solve problems or answer questions before purchase, either in a self-serve fashion or via the customer support team. Customer care is more than just providing great customer service. It's a proactive approach to providing information, tools, and services to customers at each point they interact with a brand. If a company neglects customer care, it can negatively impact the customer service experience. For example, when a website chatbot can't provide key information about a product, customers are more likely to get frustrated and reach out to a customer service agent for help. Consumer expectations are extremely high, putting increased pressure on companies to improve their customer relationships. This can lead to lost information when the same person reaches out via multiple channels. When a customer service agent doesn't know the whole story and the customer has to repeatedly share the problem, it leaves both people frustrated. They can register for an account. After the login, they can create a complaint with a description of the problem they are facing. Each user will be assigned an agent. They can view the status of their complaint.

- Customers get the insights they need to make an informed purchase.
- Customer satisfaction can increase and customer loyalty can improve.
- Customer service agents spend less time on routine-tasks and answeringcommonly asked questions, enabling agents to do more meaningful task.

1.2 PURPOSE

There are two sides to customer service objectives. First, there are the goals and KPIs customer service teams attempt to achieve. Then, there's customer service resume objectives. It's important to understand the connection between the two: Writing a strong customer service resume objective starts with understanding the objectives of the field and its depth and possibilities. To provide insight into both levels of customer service objectives. The prime objective of customer service is to answer customer questions quickly and effectively, resolve issues with empathy and care, document pain points to share with internal teams, nurture relationships, and improve brand credibility. Great customer service can make people loyal to your brand, products, and services for years to come. A strong customer service resume objective underscores your skills and experiences in contributing to customer service's overall goals and objectives. Meeting key customer service KPIs doesn't just involve answering phones and emails. It's a whole world of solutions development, intuition, empathy, brand management, time management-and the soft skills that help connect people and create trust. I guide my team toward giving the best service possible. Sometimes, we're not delivering good news. But the objective is to do that with compassion and empathy and in a way that we give the customer constructive next steps to move forward. We also know that as a newer, younger brand, customers may be wary of our credibility. It usually takes a few consistently excellent customer experiences to feel connected and loyal to the brand.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

A strong customer problem statement should provide a detailed description of your customer's current situation. Consider how they feel, the financial and emotional impact of their current situation, and any other important details about their thoughts or feelings. Customer Satisfaction is an attitude that is decided based on the experience obtained. Satisfaction is an assessment of the characteristics or privileges of a product or service, or the product itself, that provides a level of consumer pleasure with regard to meeting consumer consumption needs. Customer Satisfaction is the customer's response to the evaluation of perception of differences in initial expectations prior to purchase (or other performance standards) and the actual performance of the product as perceived after wearing or consuming the product in question. The level of complaint is how high the complaint or delivery of dissatisfaction, discomfort, irritation, and anger over the service of the service or product. The dimension or indicator of complaint level is the high level of complaint. Product Quality affects Customer Satisfaction, where the dimensions or indicators of Product Quality are quality products, in accordance with the price offered, and ease of use affects the dimensions or indicators of Customer Satisfaction in relation to subscription decisions.

2.2 REFERENCES

[1] Uddi Executive Overview: Enabling Service Oriented Architecture, 2004 Oct.
[2] "Web Services Architecture", http://www.w3.org/TR/ws-arch/. Date 6 Accessed: 11/02/2004.
[3] UDDI V3 Specification. http://uddi.org/pubs/uddi-v3.00-published20020719.html.19/07/2002.
[4] Christensen E, Curbera F, Meredith G, Weerawarana S, Web Services Description Language (WSDL) 1.1, W3C Note, 2001
[5] Ali A S, Rana O F, Ali R A, Walker D W, UDDIe: an extended registry for Web services, SAINT-w'03 Proceedings of the 2003 Symposium on Applications and the Internet Workshops, 2003 Jan, pp .85-89.
[6] Tsai W T, Paul R, Cao Z, Yu L, Saimi A, Xiao B, Verification of Web Services Using an Enhanced UDDI Server ,Proceedings of The Eighth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems ,2003 Jan, pp 131-38.
[7] Liu J, Gu N, Zong Y Ding Z, Zhang S, Zhang Q Service Registration and Discovery in a Domain-Oriented UDDI Registry , 6 Proceedings of the 2005 The Fifth International Conference on

Computer and Information Technology (CIT'05),2005, pp .276-83.
[8] Jian W, Zhaohui W, Similarity-based Web Service Matchmaking Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), 2005.
[9] Tretola G, Zimeo E , Structure Matching for Enhancing UDDI Queries Results , IEEE International Conference on Service-Oriented Computing and Applications(SOCA'07),2007.
[10] Ayorak E, Bener a, Super Peer Web Service Discovery Architecture, IEEE Proceedings - International Conference on Data Engineering, 2007 pp .287- 94.
[11] Libing He W, Wu Y, Jianqun D A novel interoperable model of distributed UDDI - proceedings of the 2008 IEEE International Conference on Networking, Architecture, and Storage - IEEE NAS 2008 Jun, pp .153-54.
[12] Nawaz F, Qadir K, Ahmad H F, and SEMREG-Pro: A Semantic based Registry for Proactive Web Service Discovery using Publish Subscribe Model, Fourth International Conference on Semantics, Knowledge and Grid, IEEE Xplore, 2008 Dec, pp .301-08. 7
[13] LIANG1 Q, CHUNG2 J A Federated UDDI System for Concurrent Access to Service Data, IEEE International Conference on eBusiness Engineering, ICEBE'08 - Workshops: AiR'08, EM2I'08, SOAIC'08, SOKM'08, BIMA'08, and DKEEE'08, 2008 Oct, pp .71- 78.
[14] Vandan T, Nirmal D, InderjeetGarg S, Garg N, Soni P. An Improved Discovery Engine for Efficient and Intelligent discovery of Web Service with publication facility, SERVICES 2009 - 5th 2009 World Congress on Services, 2009, pp .63-70.
[15] Ma C, Song M, Xu K, Zhang X.Web Service Discovery Research and Implementation Based on Semantic Search Engine, IEEE 2nd Symposium on Web Society, Beijing. 2010 Aug 16-17, pp .672-77.
[16] Rajendran T.Balasubramanie P, Flexible and Intelligent Architecture for Quality Based Web Service Discovery with an Agent- Based Approach, International Conference on Communication and Computational Intelligence (INCOCCI), Erode. 2010 Dec 27-29, pp .617- 22.
[17] Johnsen F T, Hafsøe T, Eggen A, Griwodz C, Halvorsen P, Web Services Discovery across Heterogeneous Military Networks, IEEE Communications Magazine, 2010 Oct, 48(10), pp. 84-90.
[18] Ourania H, Georgios B, Mara N, Dimosthenis A, "A Specialized Search Engine for Web Service Discovery" IEEE 19th International Conference on Web Services,USA. 2012 June 24-29, pp. 448-55.
[19] Raj R J R, Sasipraba T., Web Service Recommendation Framework Using Qos Based Discovery and Ranking Process 3rd 7 International Conference on Advanced Computing, ICoAC. Chennai. 2011Dec 14-16, pp .371-77.
[20] Ren X, Hou R, Extend UDDI Using Ontology for Automated Service Composition 2nd International Conferences on Mechanic Automation and Control Engineering, MACE, 2011 July, pp .298-301.

2.3 PROBLEM STATEMENT

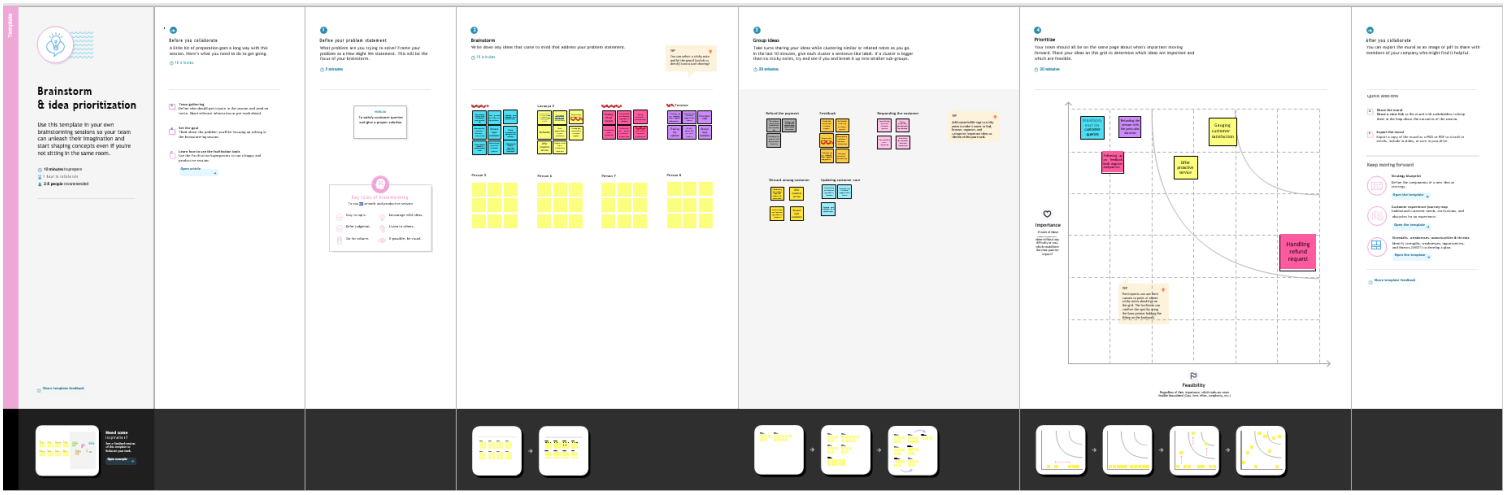
DEFINITION

A customer problem statement outlines problems that your customers face. It helps you figure out how your product or service will solve this problem for them. The statement helps you understand the experience you want to offer your customers. It can also help you understand a new audience when creating a new product or service. A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service. A Customer Problem Statement is a detailed description of an issue that needs to be addressed. This document thoroughly elaborates on the problem that your product or your service solves for your particular customers. It takes into consideration your customer's unique pain points and how your product goals about solving their situation. A customer problem statement helps you and your team understand the detailed experience you are attempting to transform by analyzing and empathizing with your customers. The customer problem statement is a critical component of a project. It benefits everyone involved with the project because it helps people understand why they're working on the project, providing clarity on the reasons behind the product or service. Team members will consider how your customers will be impacted by your project, what their thoughts and needs are, and thus come up with truly effective and valuable ways to improve their experience.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

3.2 IDEATION & BRAINSTORMING

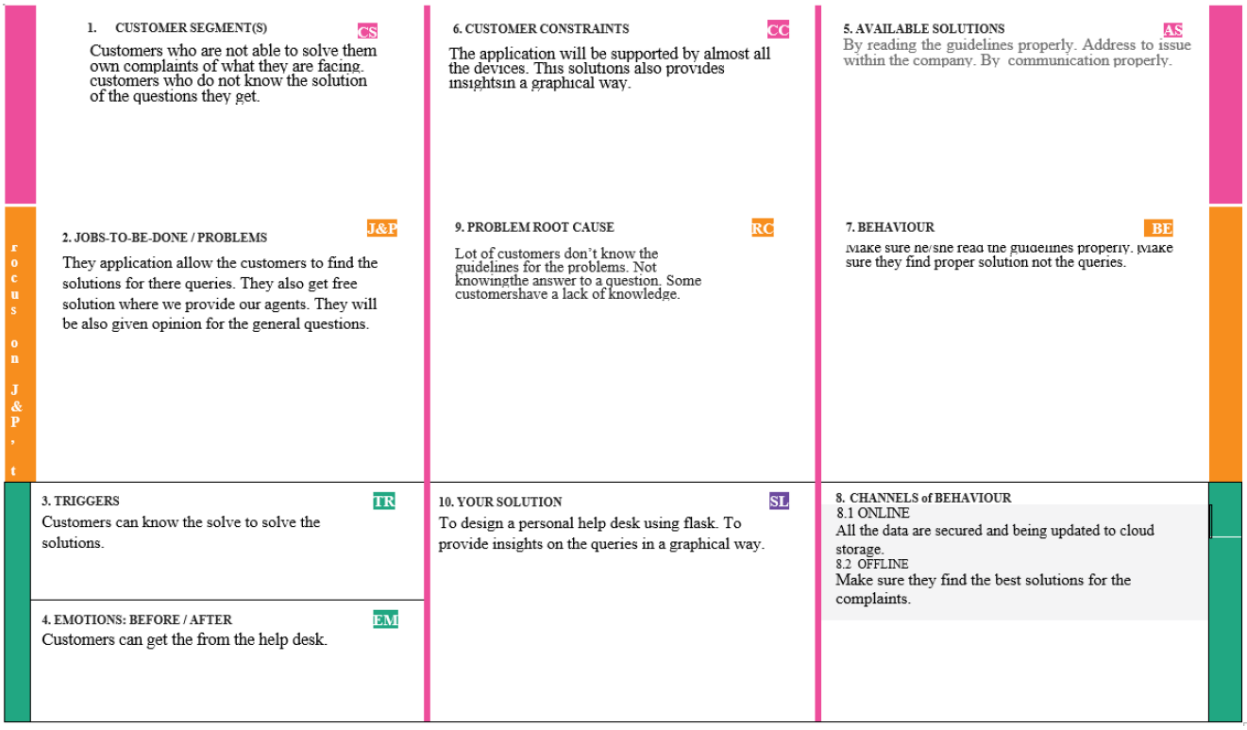


3.3 PROPOSED SOLUTION

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To solve customer issues using cloud application Development.
2.	Idea / Solution description	Assigned Agent routing can be solved by directly routing to the specific agent about the issue using the specific Email. Automated Ticket closure by using daily sync of the daily database. Regular data retrieval in the form of retrieving lost data.
3.	Novelty / Uniqueness	Assigned Agent Routing, Automated Ticket Closure, Status Shown to the customer, and Backup data in case of failures.
4.	Social Impact / Customer Satisfaction	Customer satisfaction, customer can track theirstatus and easy agent communication.
5.	Business Model (Revenue Model)	• Key Resources support Engineers,Multi-channel. • Customer Relationship have 24/7 Emailsupport,Knowledge-based channel. • Activities held as Customer service, system Maintenance • Cost Structure express Cloud Platform, Offices

3.4 PROBLEM SOLUTION FIT

PROJECT DESIGN PHASE I:



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

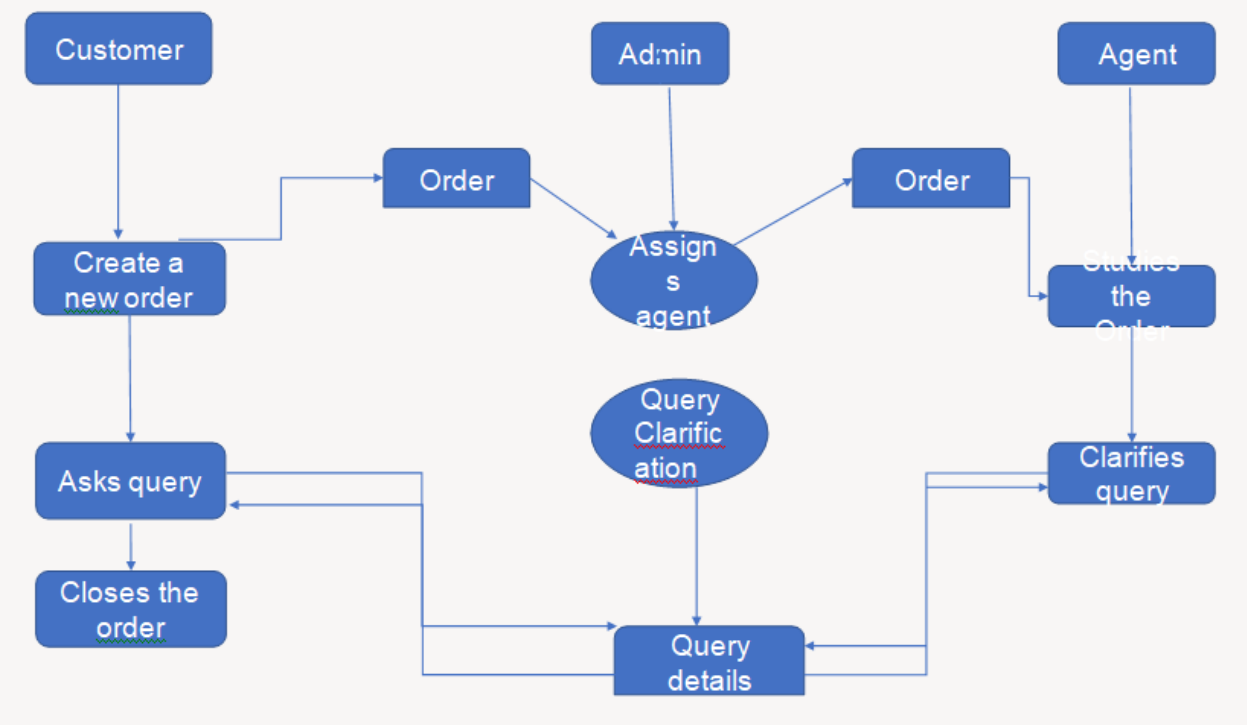
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Signup form (customer)
FR-2	Forgot Password	Resetting the password by sending an OTP to user's mail (customer, agent, admin)
FR-3	User Login	Login through Login form (customer, agent, user)
FR-4	Agent creation (admin)	Create an agent profile with username, email and password
FR-5	Dashboard (customer)	Show all the tickets raised by the customer
FR-6	Dashboard (agent)	Show all the tickets assigned to the agent by admin
FR-7	Dashboard (Admin)	Show all the tickets raised in the entire system
FR-8	Ticket creation (customer)	Customer can raise a new ticket with the detailed description of his/her query
FR-9	Assign agent (admin)	Assigning an agent for the created ticket
FR-10	Ticket details (customer)	1. Showing the actual query, status, assigned agent details 2. Status of the ticket
FR-11	Address Column	Agent clarifies the doubts of the customer

4.2 NON-FUNCTIONAL REQUIREMENT

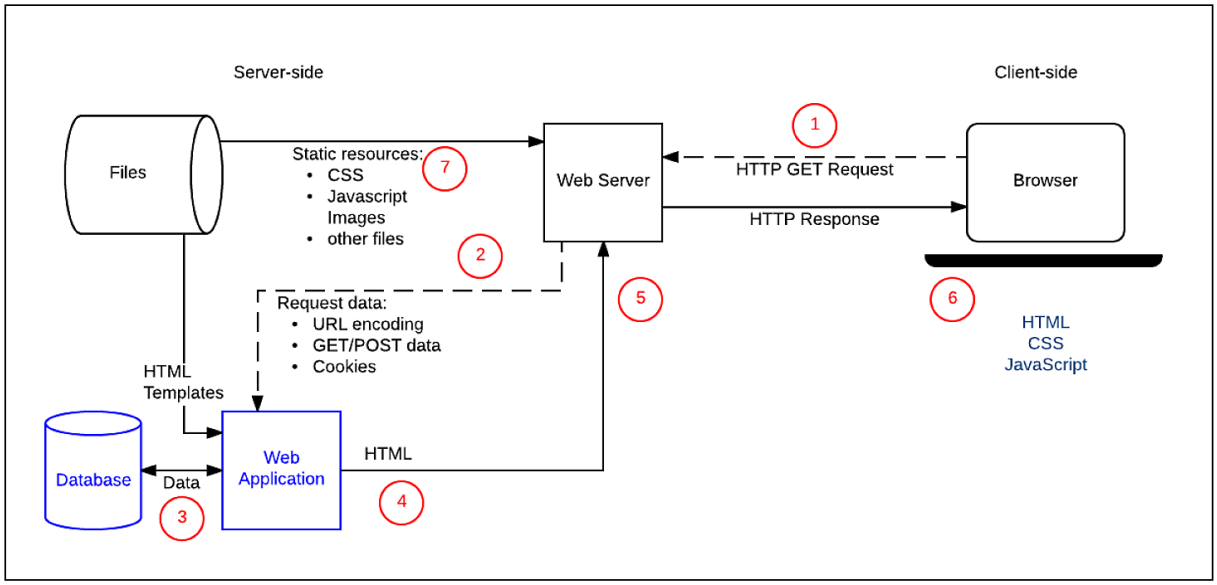
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Customers can use the application in almost all the web browsers. Application is with good looking and detailed UI, which makes it more friendly to use.
NFR-2	**Security**	Customers are asked to create an account for themselves using their email which is protected with an 8 character-long password, making it more secure.
NFR-3	**Reliability**	Customers can raise their queries and will be replied with a valid reply, as soon as possible, making the application even more reliable and trust-worthy.
NFR-4	**Performance**	Customers will have a smooth experience while using the application, as it is simple and is well optimised.
NFR-5	**Availability**	Application is available 24/7 as it is hosted on IBM Cloud
NFR-6	**Scalability**	In future, may be cross-platform mobile applications can be developed as the user base grows.

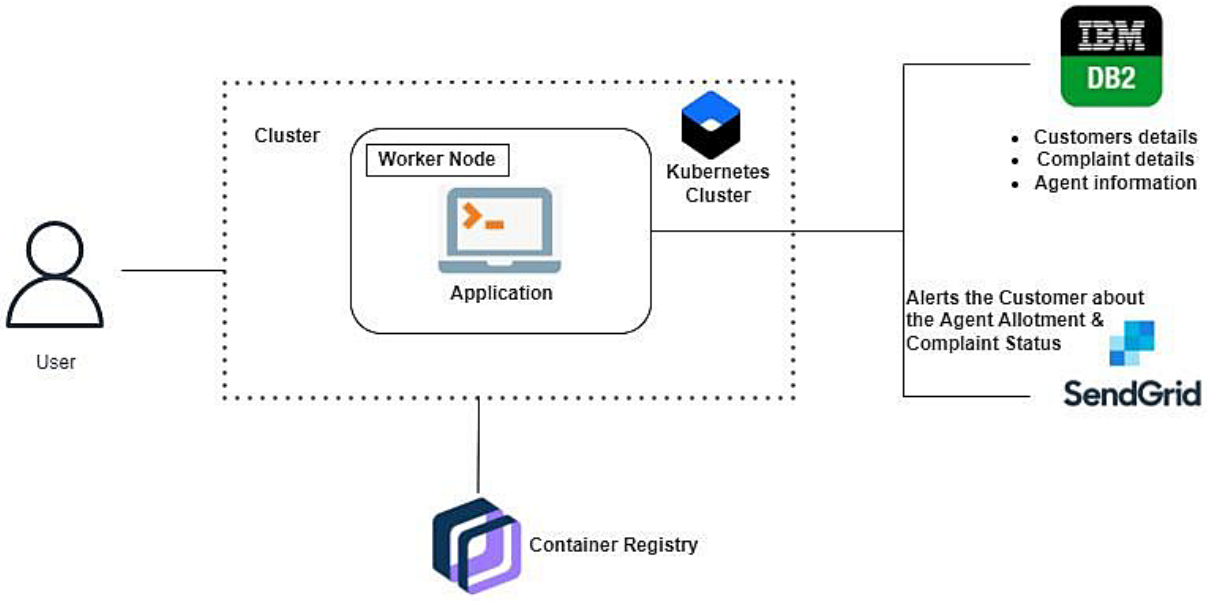
5. PROJECT DESIGN

5.1 DATA FLOW DIAGRAMS



5.2 SOLUTION AND TECHNICAL ARCHITECTURE





5.3 USER STORIES

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a customer, I can register for the application byentering my email, password, and confirmingmy password.	I can access my account / dashboard	High	Sprint-1
	Login	USN-2	As a customer, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more	I get all the info needed in my dashboard	High	Sprint-1
	Ticket creation	USN-4	As a customer, I can create a new ticket with the detailed description of my query	I can ask my query	High	Sprint-2
	Address Column	USN-5	As a customer, I can have conversations with the assigned agent and get my queries clarified	My queries are clarified	High	Sprint-3
	Forgot password	USN-6	As a customer, I can reset my password by this option in case I forgot my old password	I get access to my account again	Medium	Sprint-4
	Ticket details	USN-7	As a customer, I can see the current status of my tickets	I get better understanding	Medium	Sprint-4
Agent (Web user)	Login	USN-1	As an agent, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-3
	Dashboard	USN-2	As an agent, I can see all the tickets assigned to me by the admin	I can see the tickets to which I could answer	High	Sprint-3
	Address Column	USN-3	As an agent, I get to have conversations with the customer and clear his/her queries	I can clarify the issues	High	Sprint-3
	Forgot password	USN-4	As an agent, I can reset my password by this option in case I forgot my old password	I get access to my account again	Medium	Sprint-4
Admin (Web user)	Login	USN-1	As an admin, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-2	As an admin, I can see all the tickets raised in the entire system and lot more	I can assign agents by seeing those tickets	High	Sprint-1
	Agent creation	USN-3	As an admin, I can create an agent for clarifying the customer's queries	I can create agents	High	Sprint-2
	Assigning agent	USN-4	As an admin, I can assign an agent for each ticket created by the customer	Enables agent to clarify the queries	High	Sprint-2
	Forgot password	USN-4	As an admin, I can reset my password by this option in case I forgot my old password	I get access to my account again	Medium	Sprint-4

6. PROJECT DESIGN AND PLANNING

6.1 Sprint Planning and Estimation

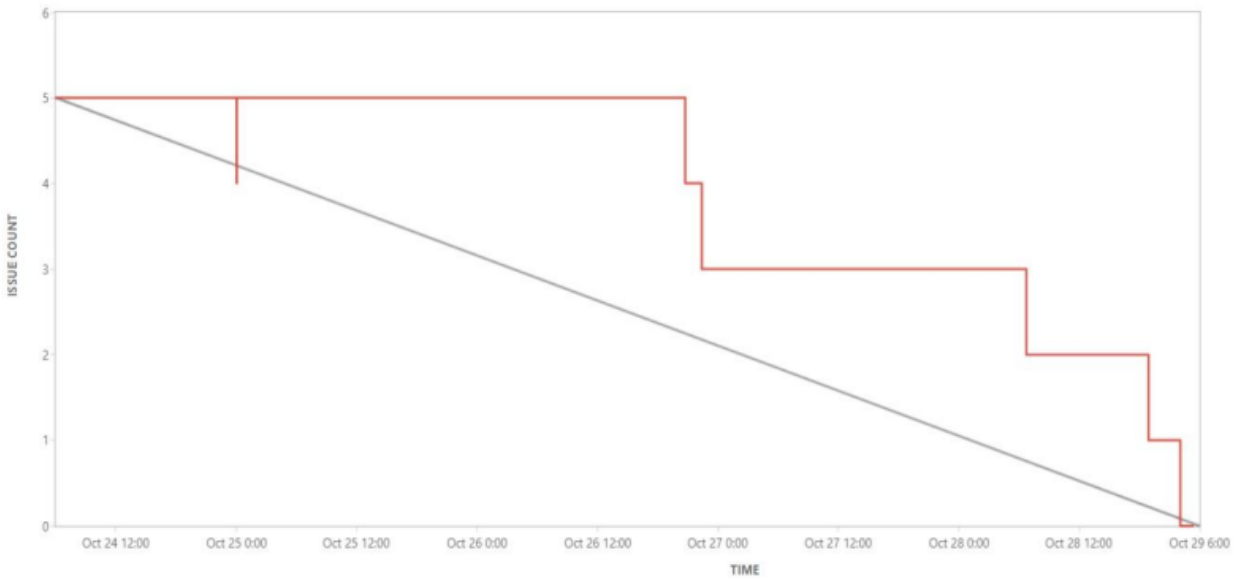
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Panel	USN-1	The user will login into the website and gothrough the services available on the webpage	20	High	JANAKRISHNAMOORTHY JIBIN SANJAY
Sprint-2	Admin panel	USN-2	The role of the admin is to check out the database about the availability and have a trackof all the things that the users are going to service	20	High	JIBIN SANJAY THANUSH
Sprint-3	Chat Bot	USN-3	The user can directly talk to Chatbot regarding the services. Get the recommendations based on information provided by the user.	20	High	JANAKRISHNAMOORTHY SANJAY THANUSH
Sprint-4	final delivery	USN-4	Container of applications using docker kubernetes anddeployment the application.Create the documentation and final submit the application	20	High	JANAKRISHNAMOORTHY JIBIN THANUSH

6.2 Sprint Delivery Plan

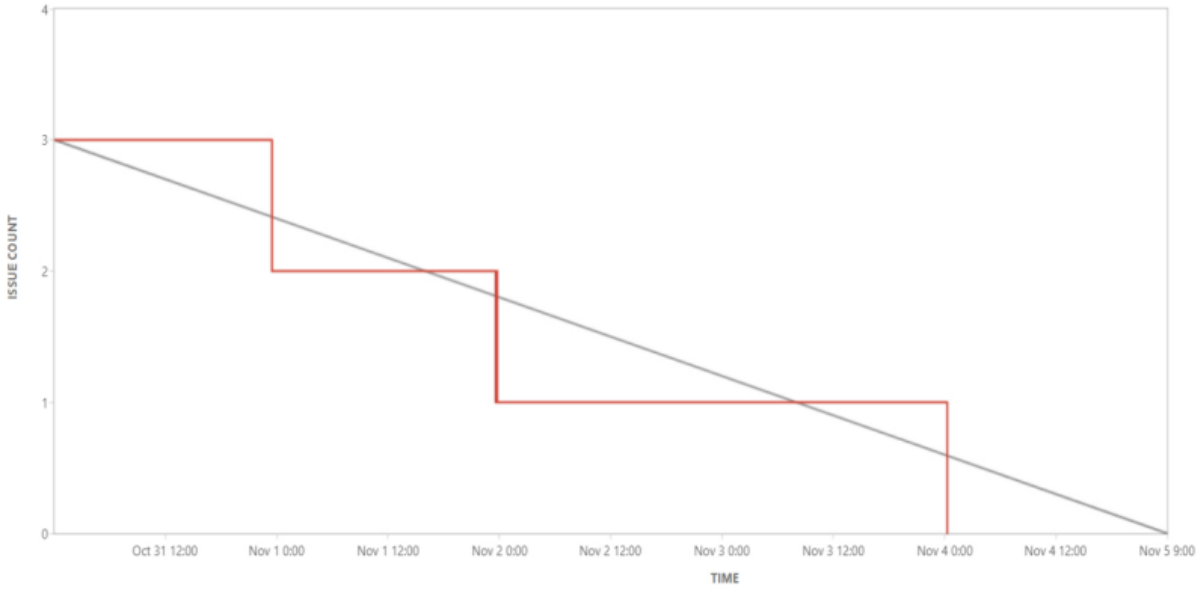
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	10	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022	7	05 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022	11	12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022	8	19 Nov 2022

6.3 Reports from JIRA

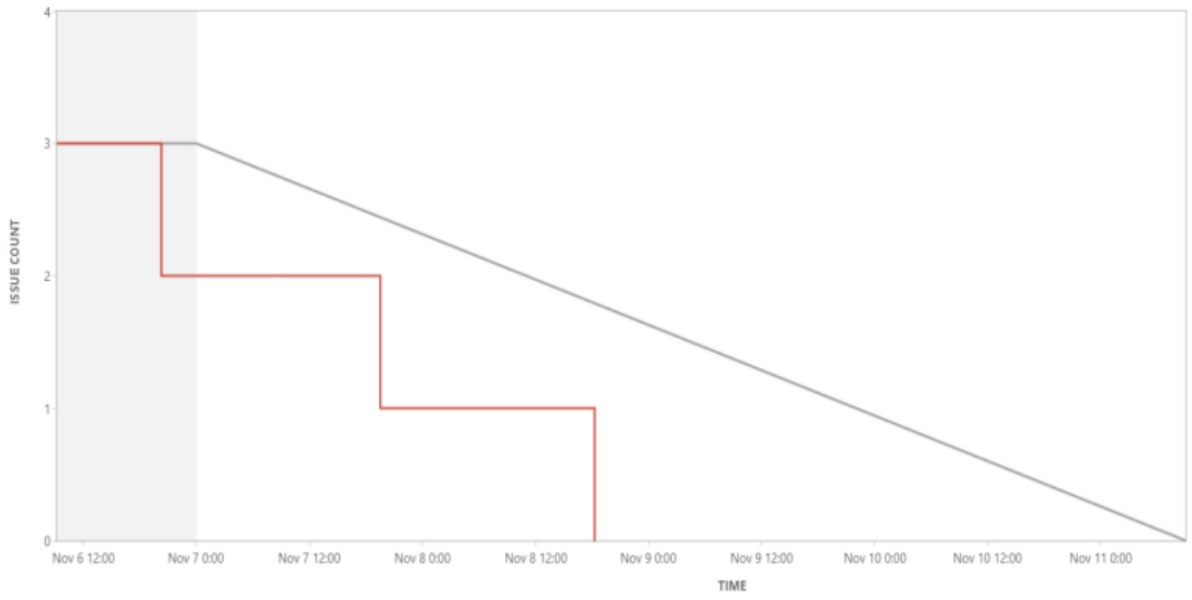
Sprint 1 – Burndown Chart



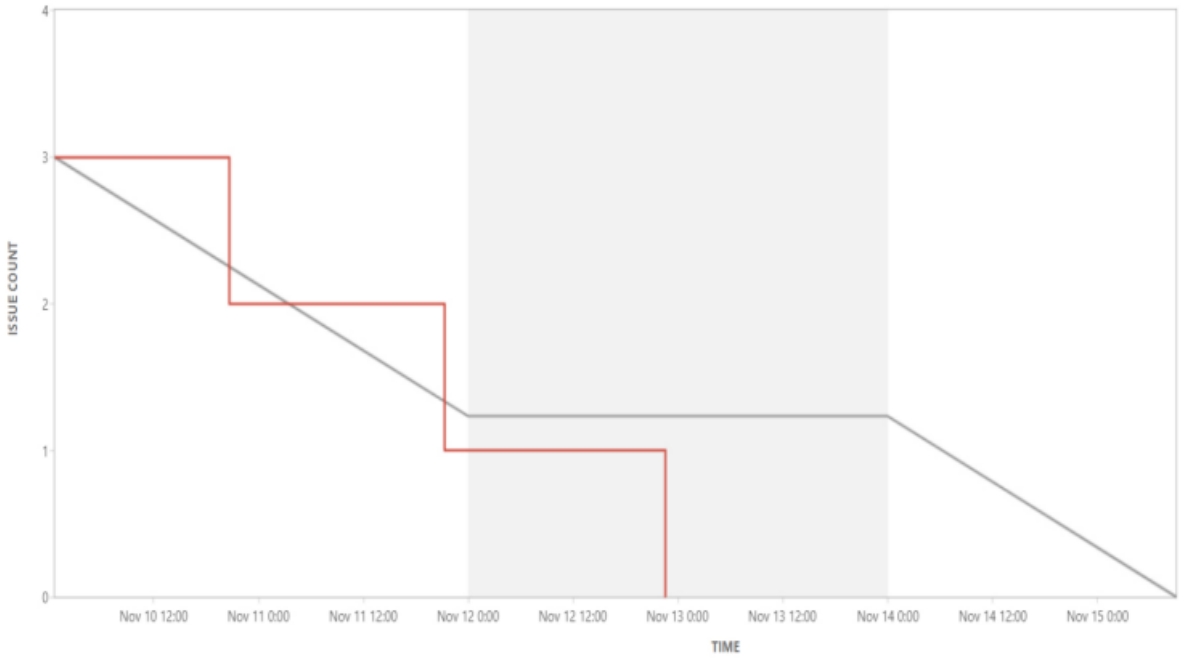
Sprint 2 – Burndown Chart



Sprint 3 – Burndown Chart



Sprint 4 – Burndown Chart



7. CODING AND SOLUTIONING

7.1 Feature 1

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 Feature 2

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 pag

8. TESTING

8.1 TEST CASES

8.1.1 FUNCTIONAL TESTING

Functional test can be defined as testing two or more modules together with the intent of finding defects, demonstrating that defects are not present, verifying that the module performs its intended functions as stated in the specification and establishing confidence that a program does what it is supposed to do.

8.1.2 WHITE BOX TESTING

Testing based on an analysis of internal workings and structure of a piece of software. This testing can be done sing the percentage value of load and energy. The tester should know what exactly is done in the internal program. Includes techniques such as Branch Testing and Path Testing. Also known as Structural Testing and Glass Box Testing.

8.1.3 BLACK BOX TESTING

Testing without knowledge of the internal workings of the item being tested. Tests are usually functional. This testing can be done by the user who has no knowledge of how the shortest path is found.

8.2 USER ACCEPTANCE TESTING

Acceptance testing can be defined in many ways, but a simple definition is the succeeds when the software functions in a manner that can be reasonable expected by the customer. After the acceptance test has been conducted, one of the two possible conditions exists. This is to fine whether the inputs are accepted by the database or other validations. For example accept only numbers in the numeric field, date format data in the date field. Also the null check for the not null fields. If any error occurs then 24 show the error messages. The function of performance characteristics to specification and is accepted. A deviation from specification is uncovered and a deficiency list is created. User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

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 wellhandled. 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

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

11. CONCLUSION

Companies today are modernizing customer care, using advanced AI to ensure a positivecustomer experience startingfrom the first interaction and throughout the buyer's journey. To properly manage customer care, companies must understand how they are succeeding and what needs improvement. This requires establishing key performance indicators (KPIs) for customer service and creating a system of gathering metrics across channels. In conclusion, customer care, involves the use of basic ethics and any company who wants to have success and grow, needsto remember, that in order to do so, it must beginwith establishing a code of ethics in regards to how each employee is to handle the dealing with customers. Customers are at the heart of the company and its growth or decline. Customer care involves, the treatment, care, loyalty, trust the employee should extend to the consumer, as well in life. This concept can be applied to so much more than just customer care.People need to treat otherswith respect and kindness; people should try to take others into consideration when making any decision. If more people were to practice this policy, chances are the world would be a better, more understanding place for all to exist. Thereby, the customer care registry would be far helpful and approachable. It offers easy tracking, recording and notification than any other means.

12. FUTURE SCOPE

The current state of customercare registery, in so many companies, lookssomething like this:
● Customer acquisition is prioritised over retention
● Customer service investment projects are sidelined.
● Departmental efficiency is of highestpriority.
● Businesses see employees in the customer service department as short-term anddisposable. They are there to fulfil a specific, repetitive, purpose.
● Employees are considered unskilled and leadershire accordingly.
● New agents view customer service as a ‘last resort’ or ‘short term’ job. Peopleoften see careers in customersupport as unambitious.
● Agent training rarelygoes beyond productand people skills. In the next 3-5 years, we expect to see these future customer care registrytrends:
● The shift from a primarily 'cost centre' to primarily 'growthcentre' worldview.
● The job desk for a customer care registry director will focus more on leadership,innovation, and ability to drive company-wide improvement.

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;0,70
0;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 AND PROJECT DEMO LINK

Github Rep Link:
https://github.com/IBM-EPBL/IBM-Project-27325-1660054120.git