CUSTOMER CARE REGISTRY

TEAM ID: PNT2022TMID42197

TEAM MEMBERS:

ARRUSH N (TEAM LEAD)	**710019104005**
KANISHKAR R	**710019104015**
POOJASHREE R	**710019104029**
PRADAP V	**710019104031**

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 answering commonly 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. That awesome experience starts from the very first touchpoint, whether it be web, email, brick and mortar, or Instagram, and carries through to when they're wearing our product

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 Accessed: 11/02/2004.

[3] UDDI V3 Specification. http://uddi.org/pubs/uddi-v3.00-published-20020719.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 ,

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.
[13] LIANG1 Q, CHUNG2 J A Federated UDDI System for Concurrent Access to Service Data, IEEE International Conference on e-
Business 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

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.

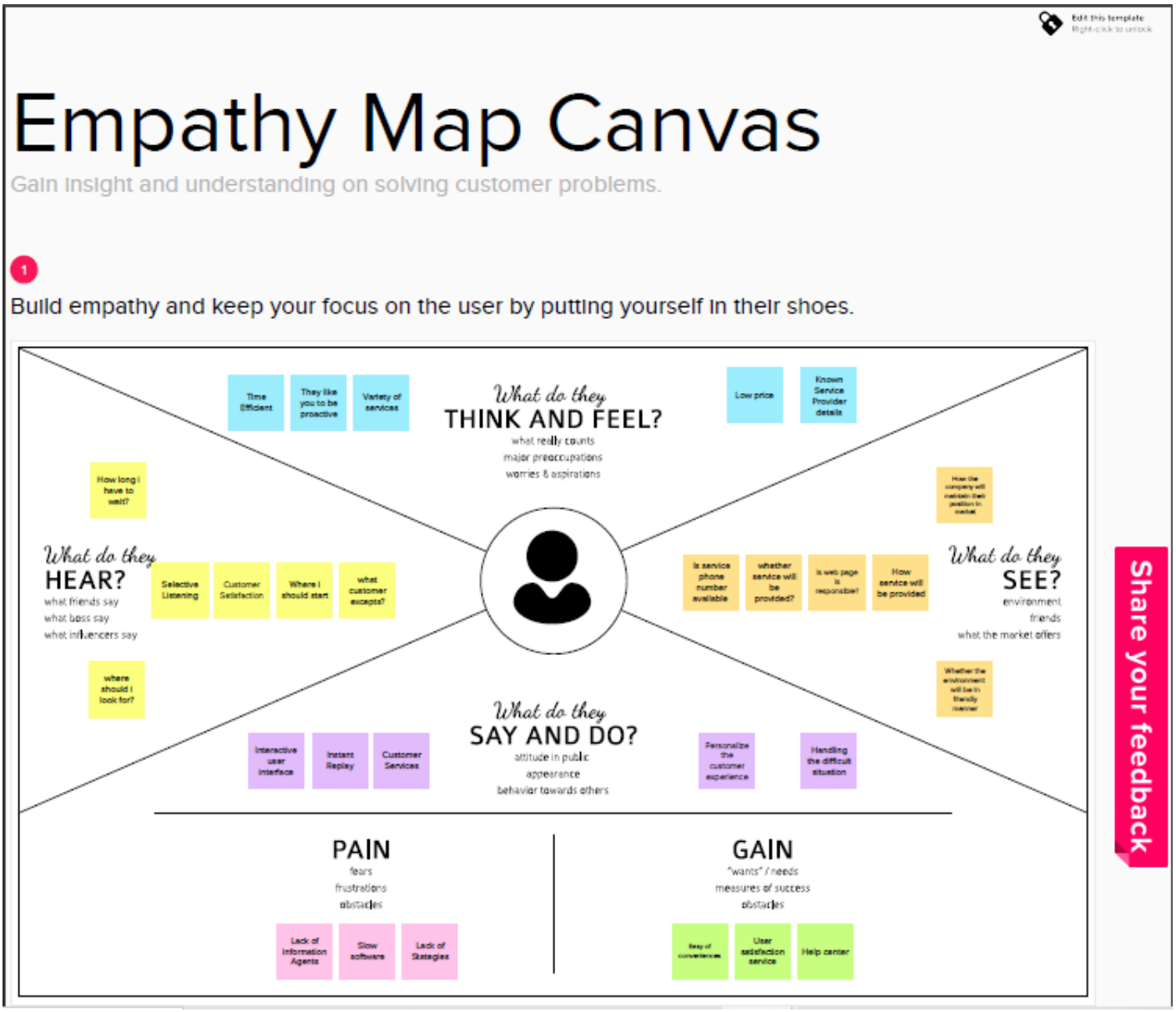
A well-articulated customer problem statement allows you and your team to find the ideal solution for the challenges your customers face. Throughout the process, you'll also be able to empathize with your customers, which helps you better understand how they perceive your product or service.

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.

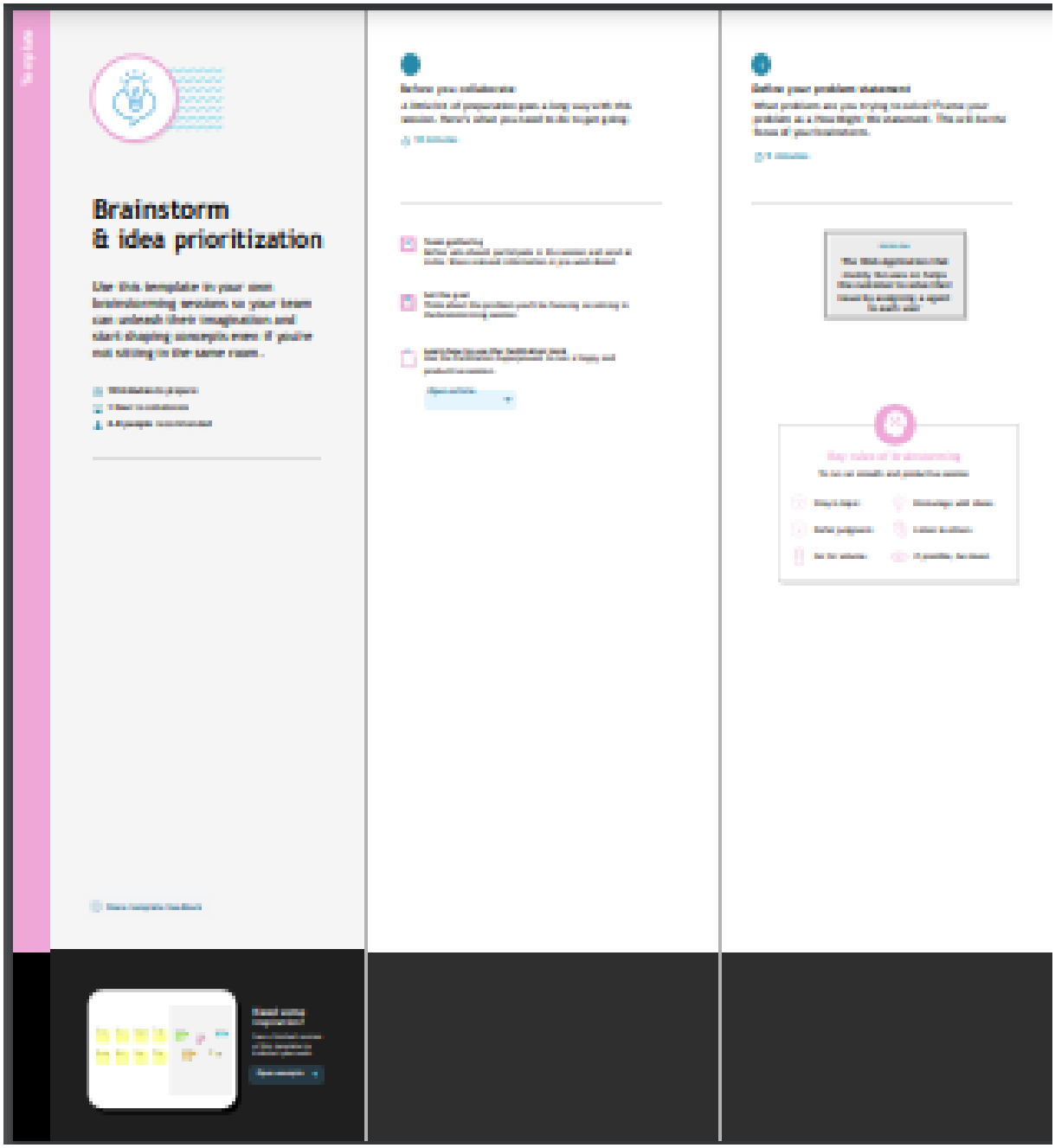
The customer problem statement is a critical component of a project. It benefits everyone involved with the project because it helps people understand why they're working on the project, providing clarity on the reasons behind the product or service. Team members will consider how your customers will 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.

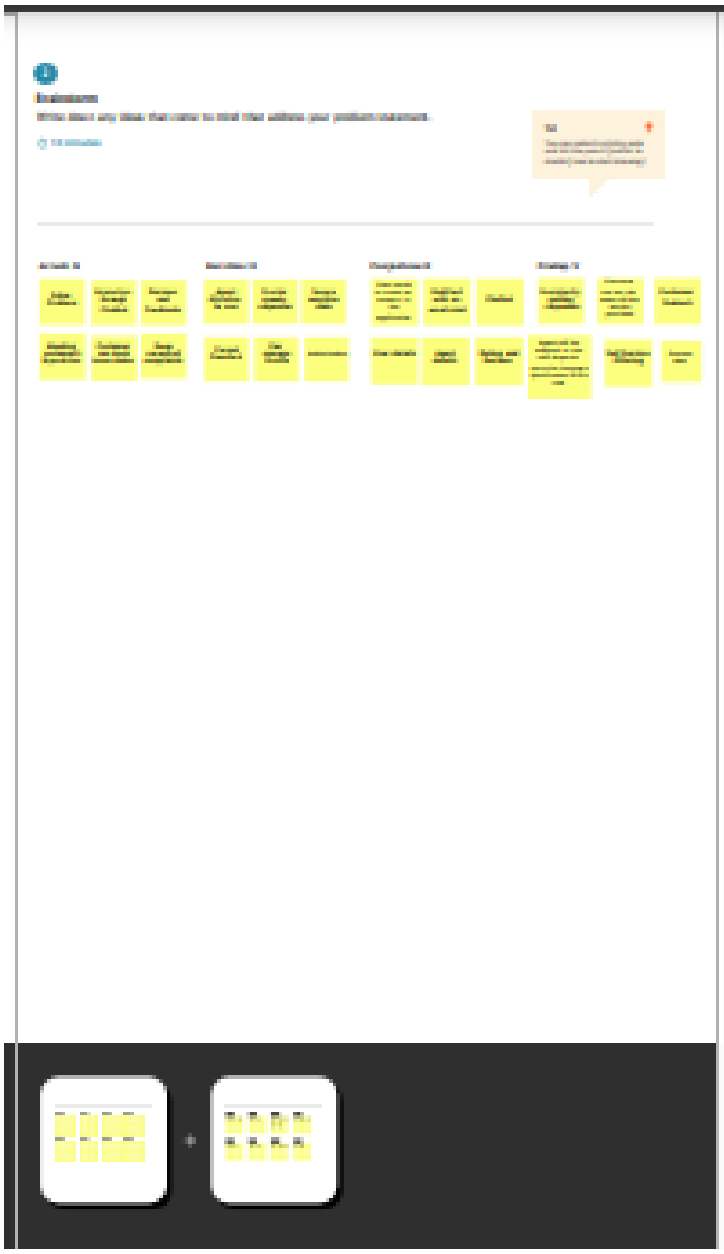
3.IDEATION & PROPOSED SOLUTION

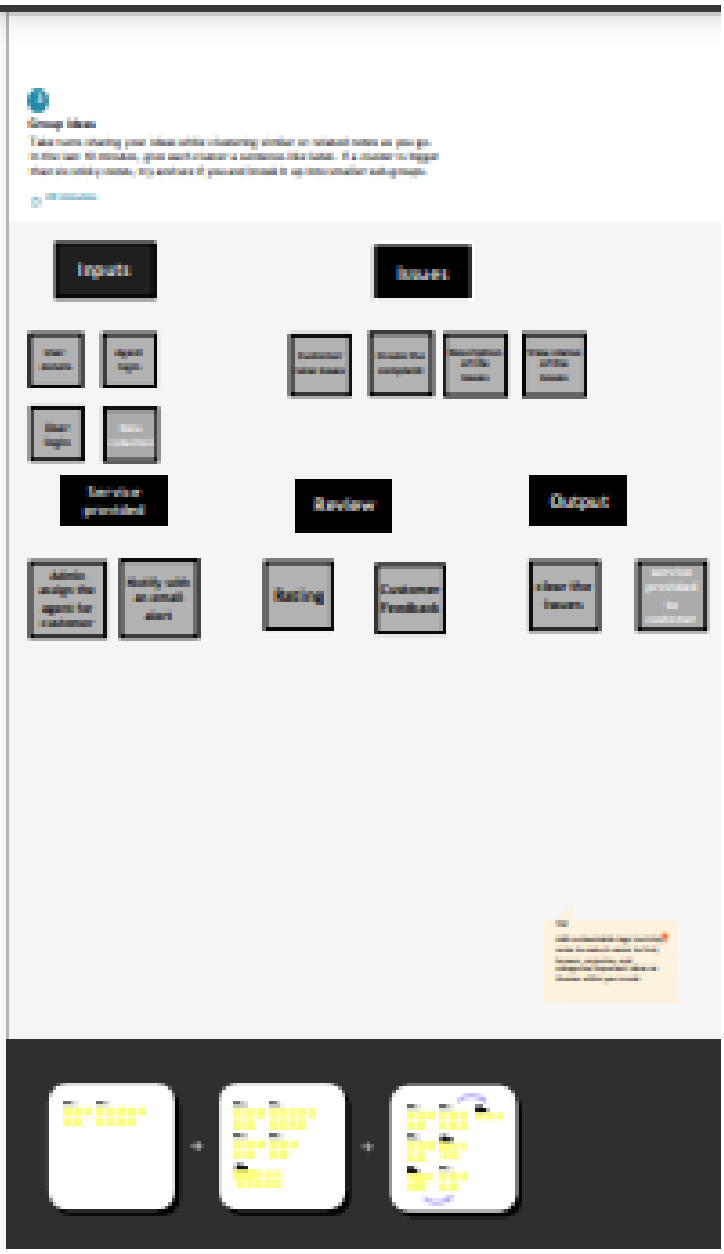
3.1 EMPATHY MAP CANVAS

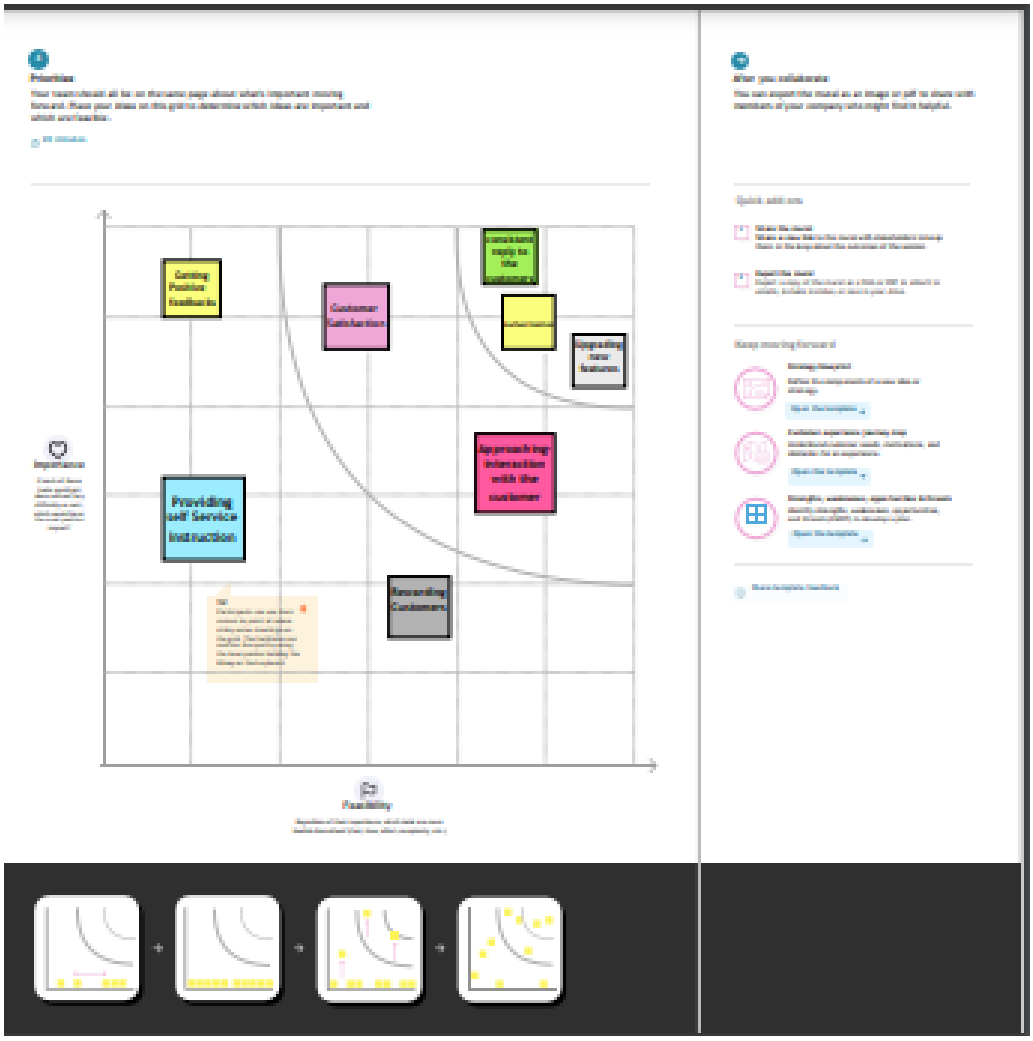


3.2 IDEATION & BRAINSTORMING






Inputs
Issues
Service provided
Review
Output
Rating
Customer Feedback



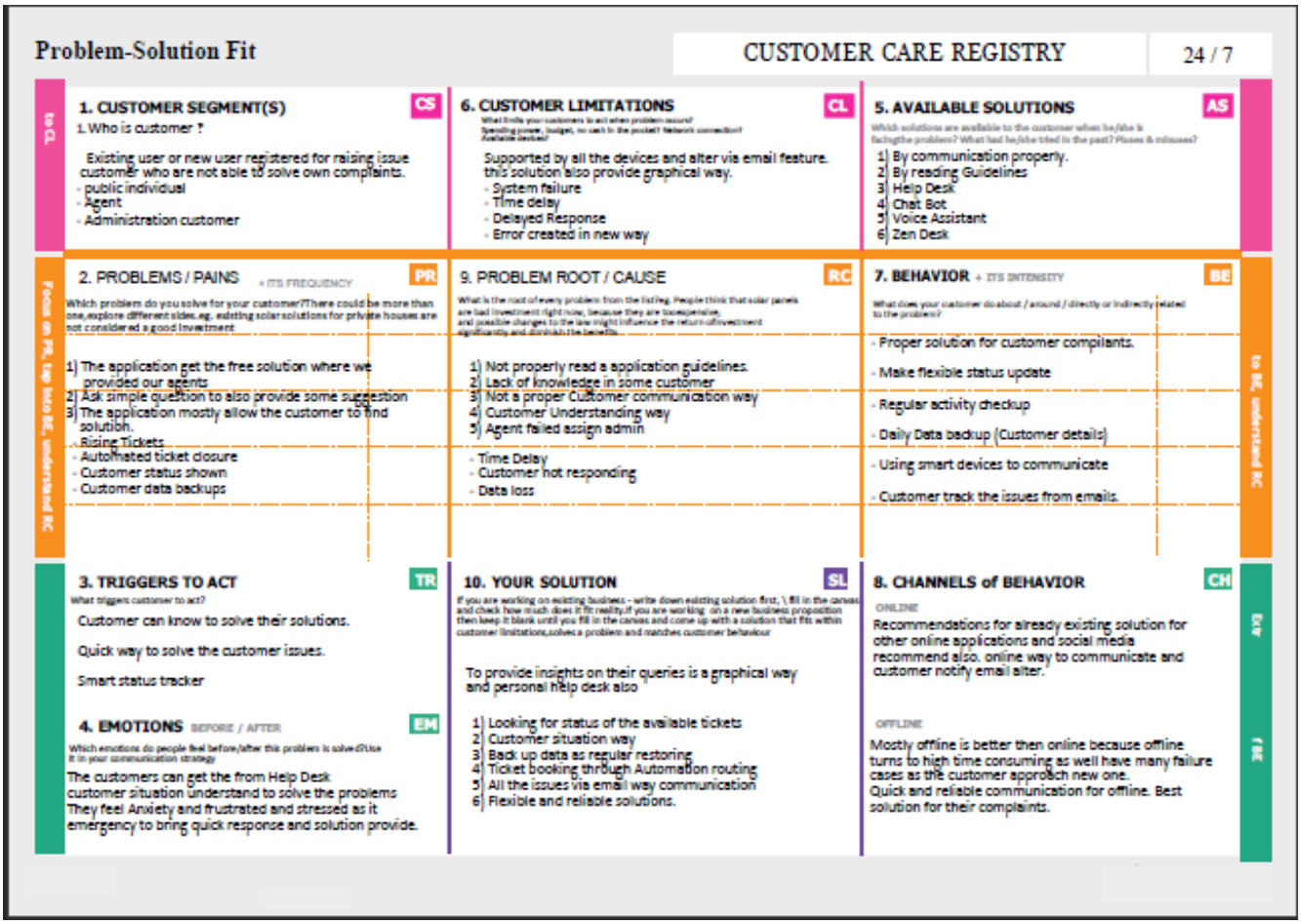
3.3 PROPOSED SOLUTION

S.NO.	PARAMETER	DESCRIPTION
01	Problem Statement (Problem to be solved)	To solve customer issues using Cloud Application Development.
02	Idea / Solution description	Assigned Agent routing can be solved by directly routing to the specific agent about theissue using the specific Email. Automated Ticket closure by using daily sync of the daily database. Status Shown to the Customer can display the status of the ticket to the customer. Regular data retrieval in the form of retrieving lost data.
03	Novelty / Uniqueness	Assigned Agent Routing, Automated Ticket Closure, Status Shown to the Customer, and Backup data in case of failures.

S.NO.	PARAMETER	DESCRIPTION
04	Social Impact / Customer Satisfaction	Customer Satisfaction, Customer can track their status and Easy agent communication.
05	Business Model (Revenue Model)	• Key Partners are Third-party applications, agents, and customers. • Activities held as Customer Service, System Maintenance. • Key Resources support Engineers, Multi-channel. • Customer Relationship have 24/7 Email Support, Knowledge-based channel. • Cost Structure expresses Cloud Platform, Offices

S.NO.	PARAMETER	DESCRIPTION
06	Scalability of the Solution	The real goal of scaling customer service is providing an environment that will allow your customer service specialists to be as efficient as possible. An environment where they will be able to spend less time on gruntwork and more time on actually resolving critical customer issues

3.4 PROBLEM SOLUTION FIT



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

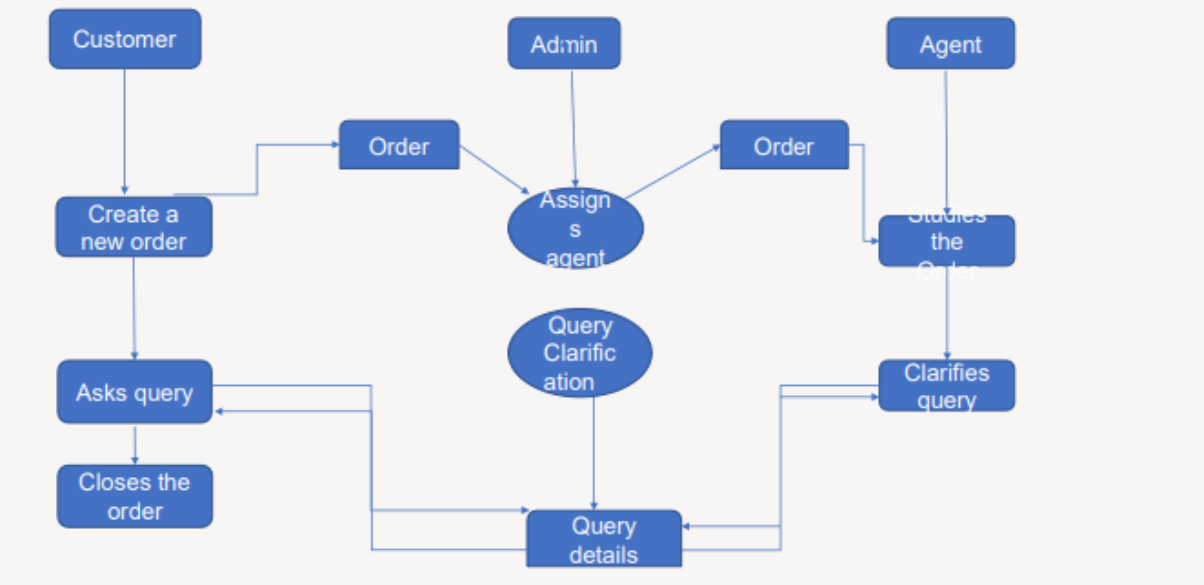
FR No	Functional Requirement(Epic)	Sub Requirement(Story/ Sub-Task)
1	User Registration	Registration through Form Registration through Gmail Registration through Google
2	User Confirmation	Confirmation via Email Confirmation via OTP
3	User Login	Login via Google Login with Email id and Password
4	Admin Login	Login via Google Login with Email id and Password
5	Query Form	Description of the issues Contact information
6	E-mail	Login alertness
7	Feedback	Customer feedback

4.2 NON-FUNCTIONAL REQUIREMENT

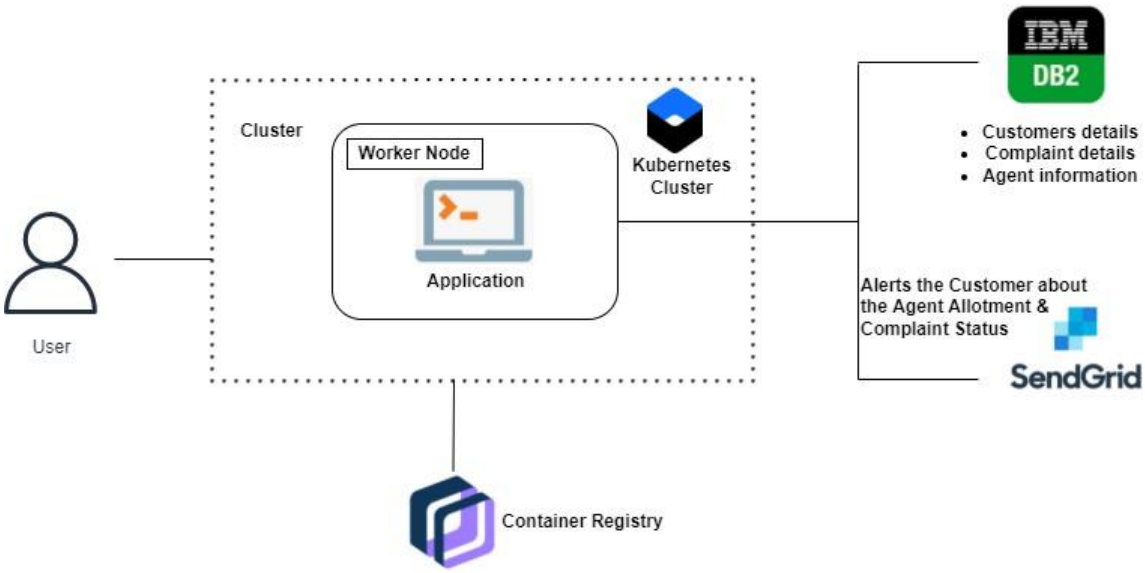
FR No	Non-Functional Requirement	Description
1	Usability	To provide the solution to the problem
2	Security	Track of login authentication
3	Reliability	Tracking of decade status through email
4	Performance	Effective development of web application
5	Availability	24/7 service
6	Scalability	Agents scalability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

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 (Mobile user)	Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my 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 orders raised by me.	I get all the info needed in my dashboard.	Low	Sprint-2
	Order creation	USN-4	As a customer, I can place my order with the detailed description of my query	I can ask my query	Medium	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 incase I forgot my old password.	I get access to my account again	Medium	Sprint-4
	Order details	USN-7	As a Customer ,I can see the current stats of order.	I get abetter 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 the order details assigned to me by 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/er dobuts	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 (Mobile user)	Login	USN-1	As a admin, I can login to the appliaction 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 orders raised in the entire system and lot more	I can assign agents by seeing those order.	High	Sprint-1
	Agent creation	USN-3	As an admin I can create an agent for clarifying the customers queries	I can create agents.	High	Sprint-2
	Assignment agent	USN-4	As an admin I can assign an agent for each order created by the customer.	Enable agent to clarify the queries.	High	Sprint-1
	Forgot password	USN-5	As an admin I can reset my password by this option in case I forgot my old password.	I get access to my account.	High	Sprint-1

6. PROJECT PLANNING & SCHEDULE

6.1 SPRINT PLANNING &ESTIMATION

Product Backlog, Sprint Schedule, and Estimation (4 Marks)

Sprint	User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Customer (Web User)	Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my password.	2	High	Pradap , Poojashree
Sprint-1		Login	USN-2	As a customer, I can login to the application by entering correct email and password	1	High	Arrush , Kanishkar
Sprint-1		Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more	3	High	Pradap , Arrush
Sprint-2		Ticket creation	USN-4	As a customer, I can create a new ticket with the detailed description of my query	2	High	Kanishkar , Pradap
Sprint-3		Address Column	USN-5	As a customer, I can have conversations with the assigned agent and get my queries clarified	3	High	Arrush , Poojashree
Sprint-4		Forgot password	USN-6	As a customer, I can reset my password by this option in case I forgot my old password	2	Medium	Arrush , Kanishkar
Sprint-4		Ticket details	USN-7	As a customer, I can see the current status of my tickets	2	Medium	Pradap , Poojashree

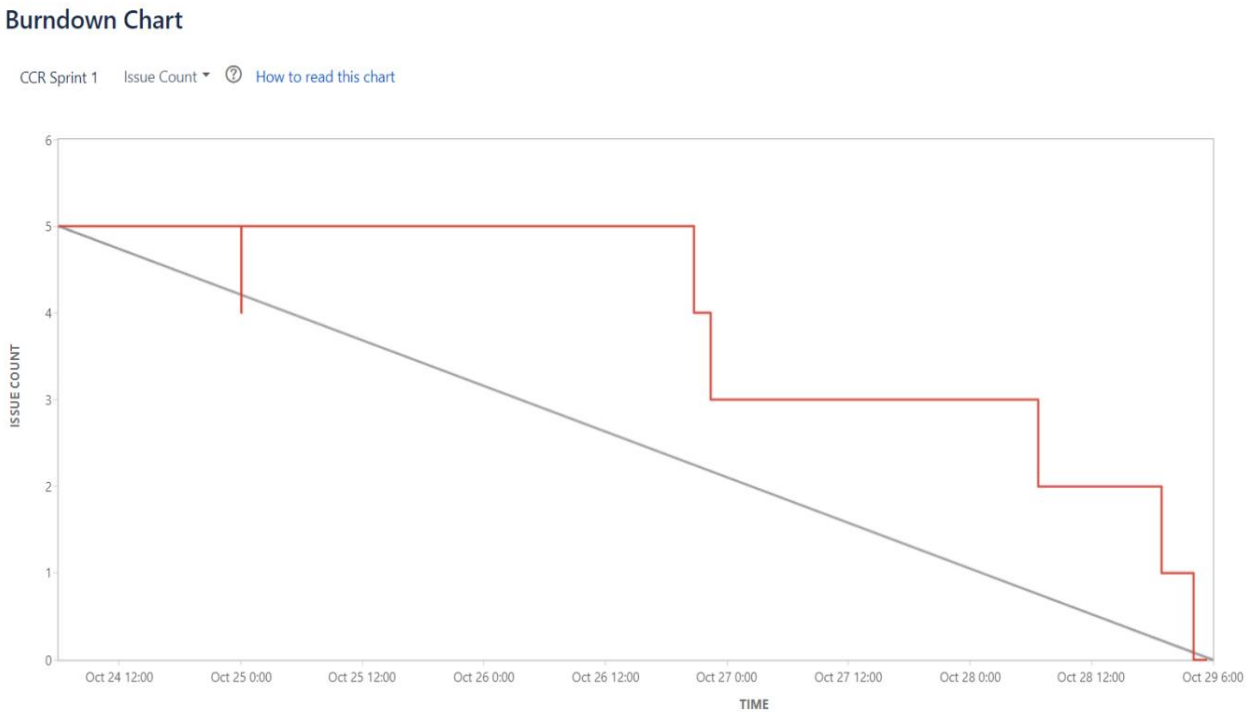
Sprint	User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-3	Agent (Web user)	Login	USN-1	As an agent, I can login to the application by entering correct email and password	2	High	Arrush
Sprint-3		Dashboard	USN-2	As an agent, I can see all the tickets assigned to me by the admin	3	High	Poojashree
Sprint-3		Address Column	USN-3	As an agent, I get to have conversations with the customer and clear his/her queries	3	High	Kanishkar , Arrush
Sprint-4		Forgot password	USN-4	As an agent, I can reset my password by this option in case I forgot my old password	2	Medium	Kanishkar , Pradap
Sprint-1	Admin (Web user)	Login	USN-1	As an admin, I can login to the application by entering correct email and password	1	High	Pradap , Poojashree
Sprint-1		Dashboard	USN-2	As an admin, I can see all the tickets raised in the entire system and lot more	3	High	Arrush
Sprint-2		Agent creation	USN-3	As an admin, I can create an agent for clarifying the customer's queries	2	High	Pradap
Sprint-2		Assigning agent	USN-4	As an admin, I can assign an agent for each ticket created by the customer	3	High	Arrush , Kanishkar
Sprint-4		Forgot password	USN-4	As an admin, I can reset my password by this option in case I forgot my old password	2	Medium	Arrush , Poojashree

6.2 SPRINT DELIVERY SCHEDULE

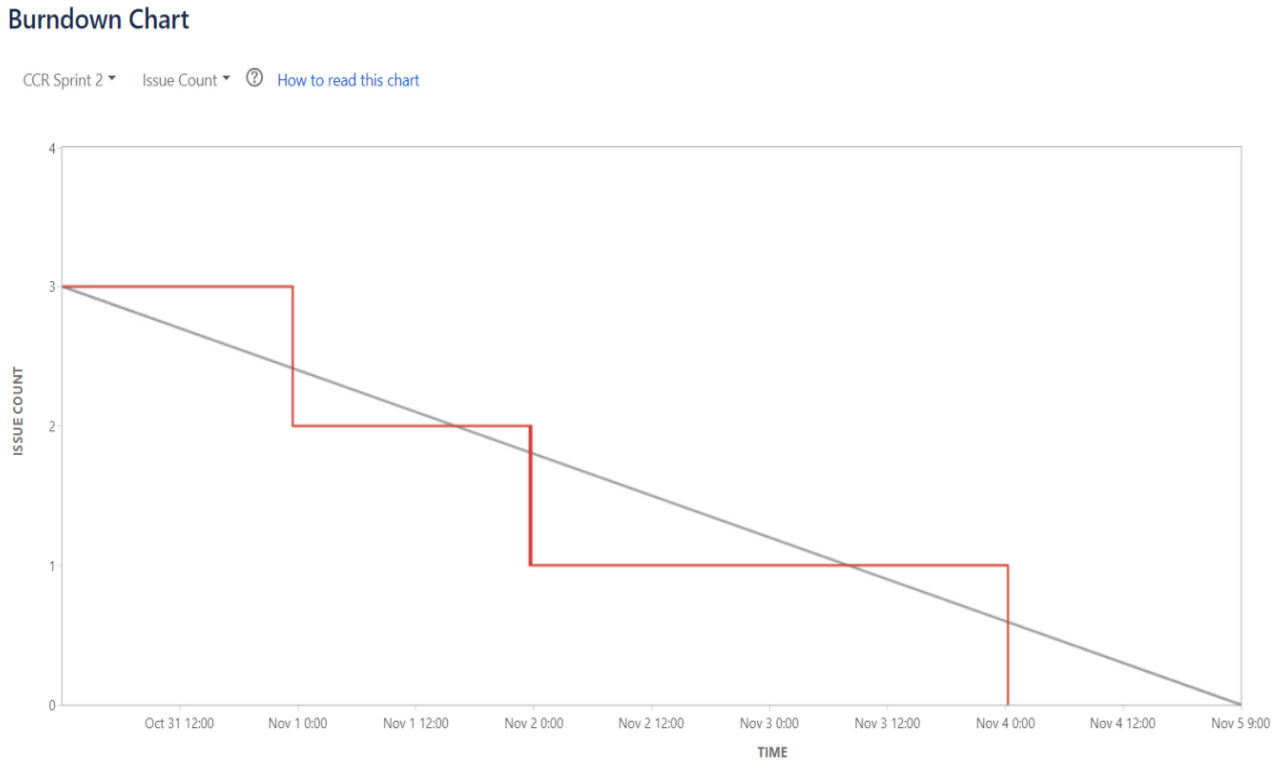
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	10	3 Days	8 Nov 2022	10 Nov 2022	10	10 Nov 2022
Sprint-2	7	3 Days	11 Nov 2022	13 Nov 2022	7	13 Nov 2022
Sprint-3	11	3 Days	14 Nov 2022	16 Nov 2022	11	16 Nov 2022
Sprint-4	8	3 Days	17 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 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'))
```

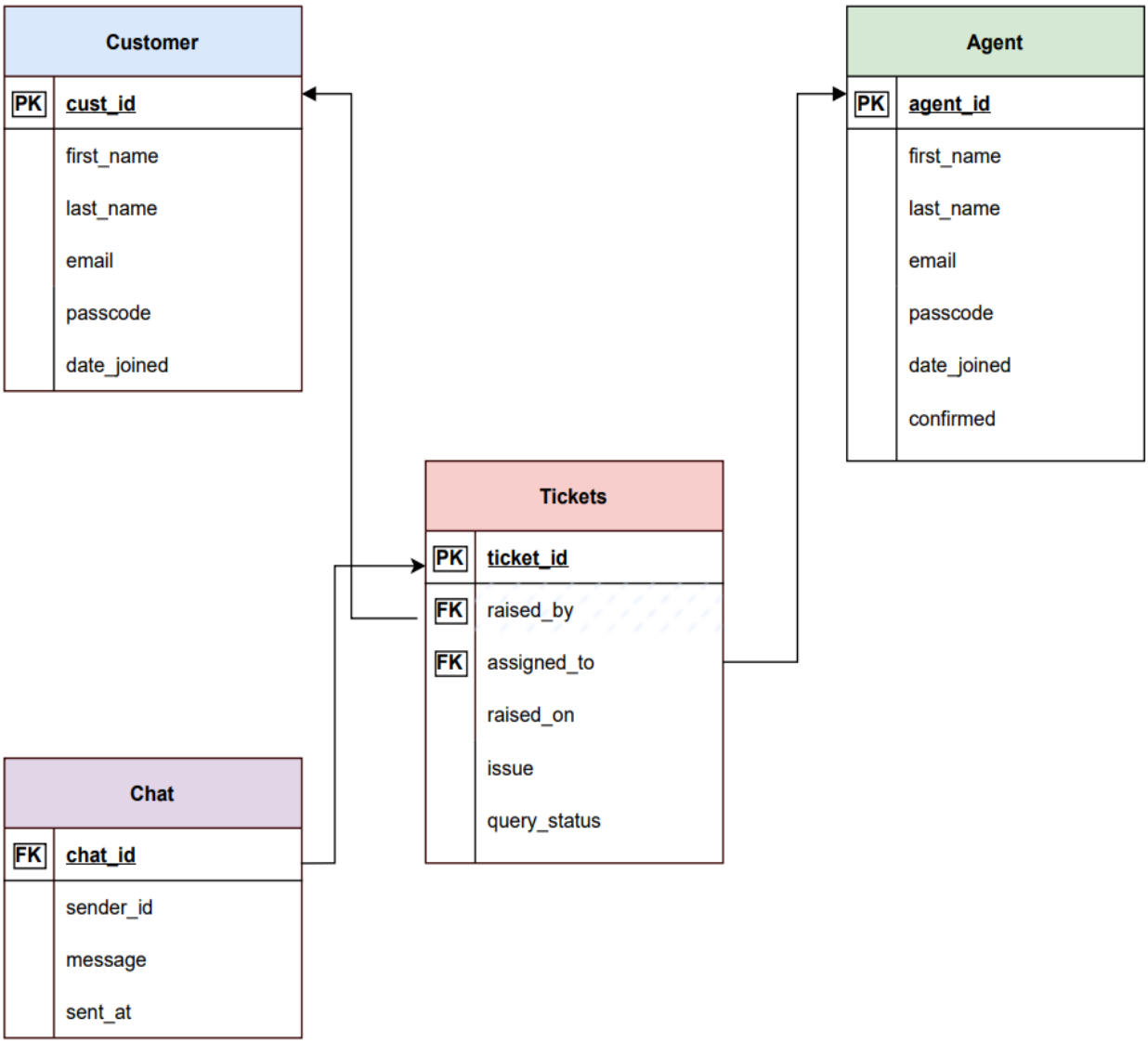
Explanation:

- 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.

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

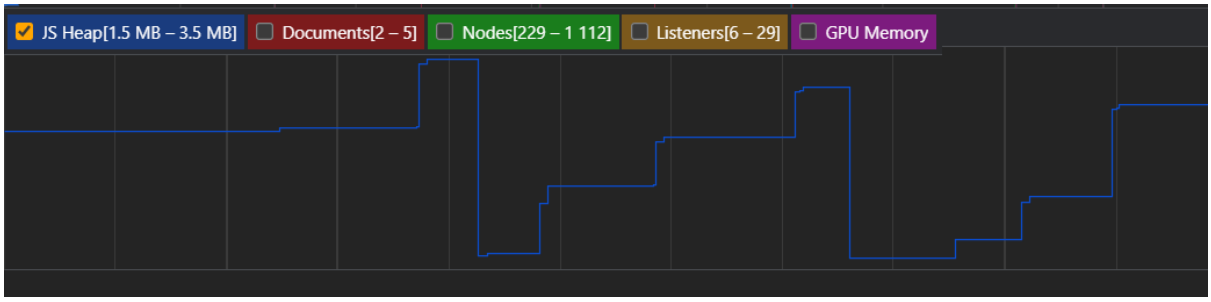
Section	Total Cases	Not Tested	Fail	Pass
Client Application	72	0	0	72
Security	7	0	0	7
Exception Reporting	5	0	0	5
Final Report Output	4	0	0	4

9. RESULTS

9.1 Performance Metrics:

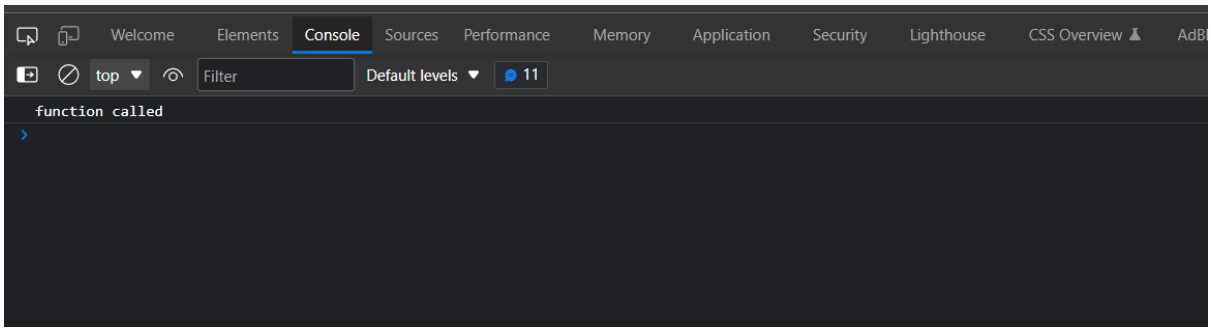
CPU usage:

- ✓ 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.



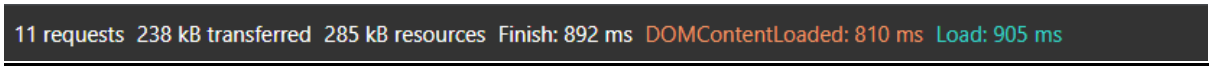
Errors:

- ✓ 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 AND PROJECT DEMO LINK

Github Rep Link:

https://github.com/IBM-EPBL/IBM-Project-1237-1658380072

Project Demo Link:

https://drive.google.com/drive/folders/1Y83eMqZYpms8yq65qDwBeHrUIRI6Nu4R?usp=sharing