CUSTOMER CARE REGISTRY

A PROJECT REPORT

Submitted by

DHEYANESHWARAN S	**(731619205012)**
RAJESH KANNA K	**(731619205039)**
SURYA S K	**(731619205052)**
VIGNESH S	**(731619205058)**

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KSR INSTITUTE FOR ENGINEERING AND TECHNOLOGY,

TIRUCHENGODE

ANNA UNIVERSITY : CHENNAI 600 025

Customer Care Registry

Project Name	:	Customer Care Registry
Project Domain	:	Cloud Application Development
College	:	KSR Institute for Engineeering and Technology
College SPOC	:	Mrs. Keerthana P
Team ID	:	**PNT2022TMID32111**
Team Size	:	4
Team Members	:	A. Surya SK B. Dheyaneshwaran S C. Rajesh Kanna K D. Vignesh S
Team Mentor	:	Mr. SelvaPrabhu T
Team Evaluator	:	Mr. Vasudeva Hanush
Github Link	:	click here
Project Demo Link	:	click here

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 theCustomer 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 descriptionof the problem they are facing. Each user will be assigned with an agent. They can view the status oftheir complaint.

1.2 Project 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 feed backs to the Admin, for the betterment of theplatform

2. LITERATURE SURVEY

2.1 Existing Problem

- Reviews and rating in the e-commerce websites are not reliable
- Even more so, they are often been given by the manufactures themselves
- Reviews are not from the authentic individuals
- After buying the products, I am left with no option to clear my doubts
- There is no common platform available to us, the customers, to have our doubts cleared
- If it is existing, we are not getting fast replies. By the time, the reply comes, the issuemight have been cleared or of not worth of being cleared to the customers

2.2 References

https://www.helpdesk.com/

https://freshdesk.com/helpdesk-software

https://freshdesk.com/resources/case-study/hamleys

https://pulsedesk.com/

https://www.redpoints.com/blog/amazon-fake-reviews/

2.3 Problem Statement Definition

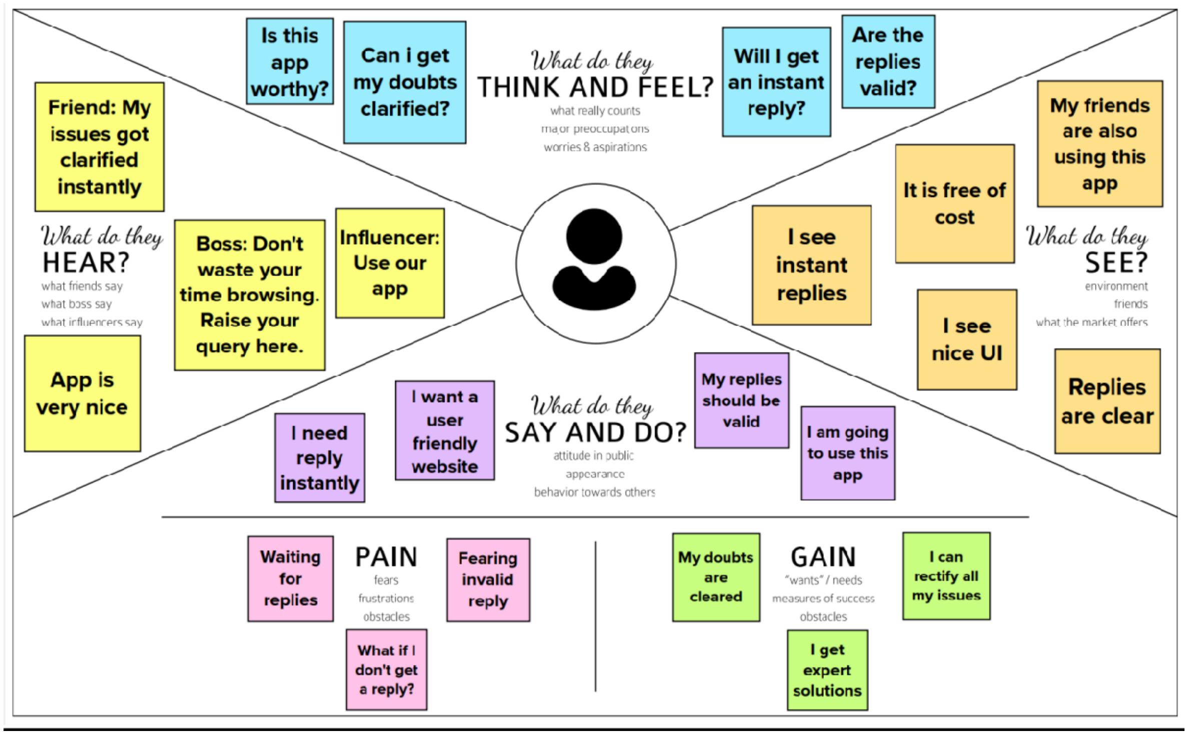
I am Surya and I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don't have any reliable sources to clear my doubts in some of the products I buy.

There are reviews and customer ratings in those websites, but somehow, I don't feel they areauthentic and real. It would make my world if those replies were from a real expert, and I could clarifyall my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for.

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 behaviours 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 personwho 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 teamto 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:

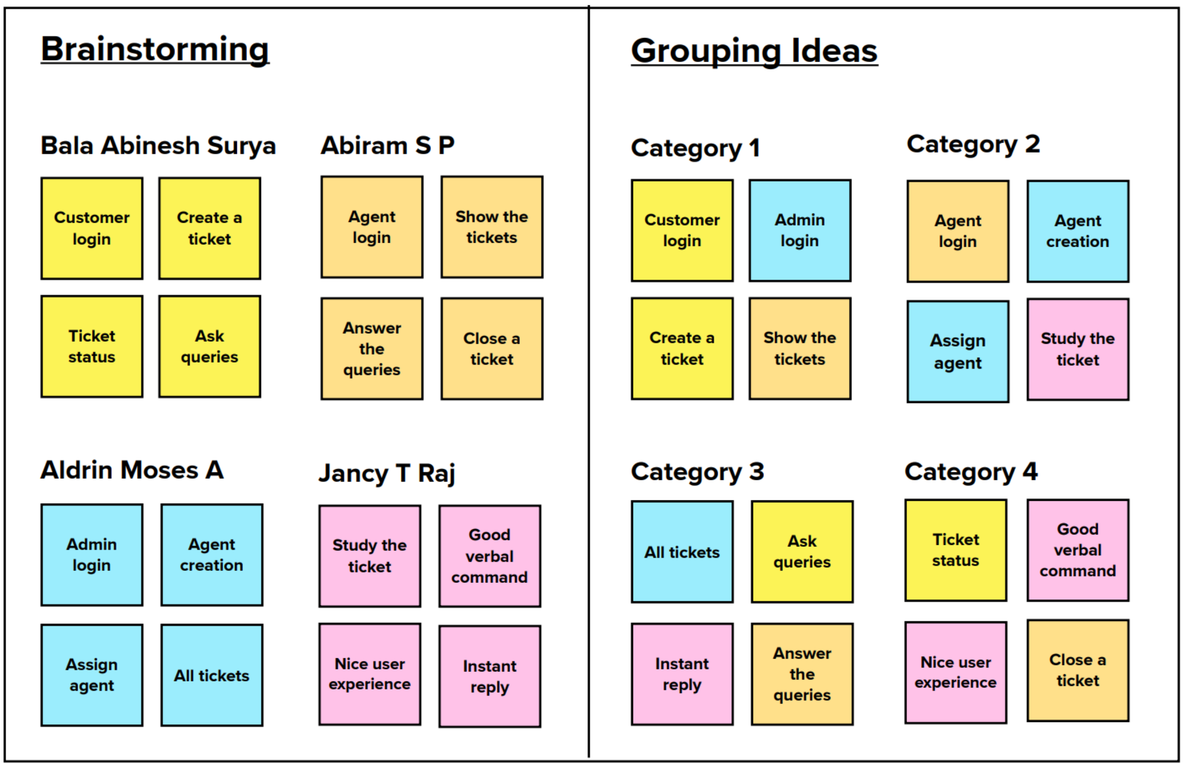
Team Members	
Team Leader	Surya S K
Team Members	Dheyaneshwaran S
	Rajesh Kanna K
	Vignesh S

Problem Statement:

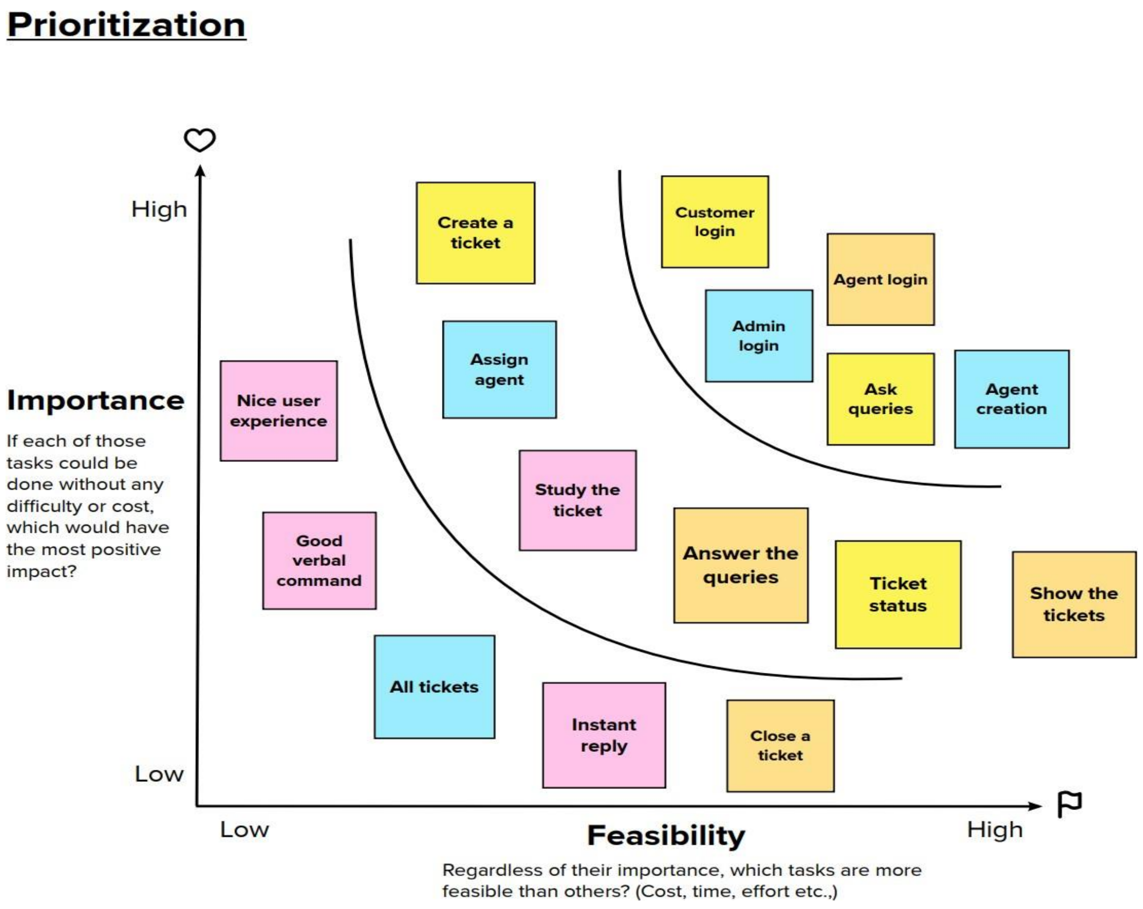
I am Surya and I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don’t have any reliable sources to clear my doubts in some of the products I buy.

There are reviews and customer ratings in those websites, but somehow, I don’t feel they areauthentic and real. It would make my world if those replies were from a real expert, and I could clarifyall my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for.

Step-2: Brainstorm, Idea Listing and Grouping



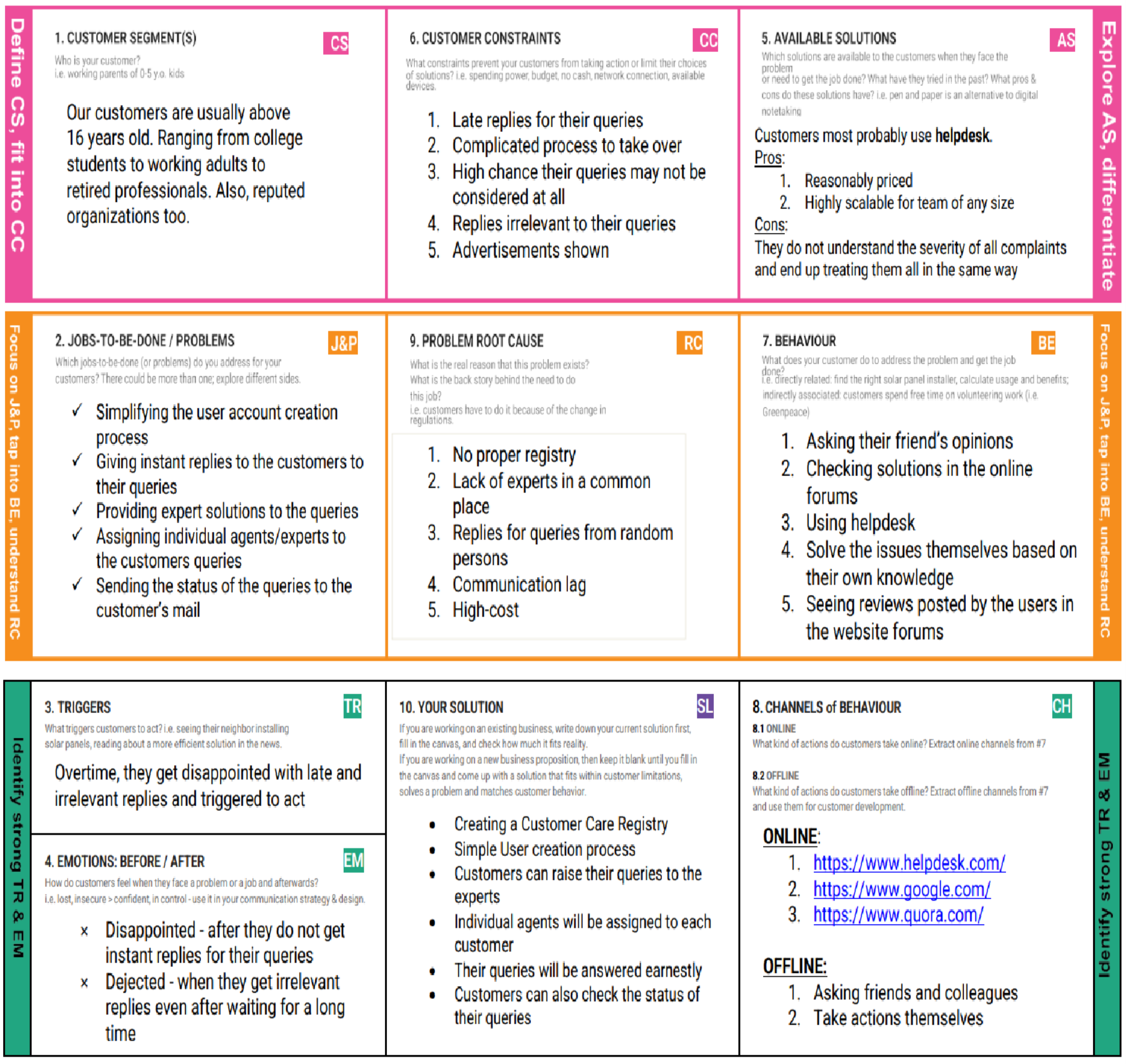
Step-3: Idea Prioritization



3.3 Proposed Solution

S. No.	Parameter	Description
▯	Problem Statement (Problem to be solved)	I am Surya and I am a regular customer in famous e- commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don’thave any reliable sources to clear my doubts in some of the products I buy. There are reviews and customer ratings in those websites,but somehow, I don’t feel they are authentic and real. It would make my world if those replies are from a real expert and I could clarify all my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for.
▯	Idea / Solution description	Creating a Customer Care Registry, where the customerscan raise their queries in form of tickets. An agent will be assigned to them for replying/clarifying their issue.
▯	Novelty / Uniqueness	The agents are experts in the product domain and theywill communicate well with the customers
▯	Social Impact / Customer Satisfaction	Customers will be satisfied with the instant and valid replies. Also, it creates a doubtless society, that boostssales.
▯	Business Model (Revenue Model)	Customers can be charged a minimal amount based on the number of queries (tickets) they can rise in a said period of time.
▯	Scalability of the Solution	May be in the future, may be a cross-platform mobile application may be developed, making this customer care registry much more accessible to the users.

3.4 Problem Solution Fit



Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

Who is your customer?
i.e. working parents of 0-5 y.o. kids

Our customers are usually above 16 years old. Ranging from college students to working adults to retired professionals. Also, reputed organizations too.

6. CUSTOMER CONSTRAINTS CC

What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices.

1. Late replies for their queries
2. Complicated process to take over
3. High chance their queries may not be considered at all
4. Replies irrelevant to their queries
5. Advertisements shown

5. AVAILABLE SOLUTIONS AS

Which solutions are available to the customers when they face the problem
or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking

Customers most probably use **helpdesk.**

Pros:

1. Reasonably priced
2. Highly scalable for team of any size

Cons:

They do not understand the severity of all complaints and end up treating them all in the same way

Explore AS, differentiate

Focus on J&P, tap into BE, understand RC

2. JOBS-TO-BE-DONE / PROBLEMS J&P

Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides.

- ✓ Simplifying the user account creation process
- ✓ Giving instant replies to the customers to their queries
- ✓ Providing expert solutions to the queries
- ✓ Assigning individual agents/experts to the customers queries
- ✓ Sending the status of the queries to the customer's mail

9. PROBLEM ROOT CAUSE RC

What is the real reason that this problem exists?
What is the back story behind the need to do this job?
i.e. customers have to do it because of the change in regulations.

1. No proper registry
2. Lack of experts in a common place
3. Replies for queries from random persons
4. Communication lag
5. High-cost

7. BEHAVIOUR BE

What does your customer do to address the problem and get the job done?
i.e. directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace)

1. Asking their friend's opinions
2. Checking solutions in the online forums
3. Using helpdesk
4. Solve the issues themselves based on their own knowledge
5. Seeing reviews posted by the users in the website forums

Focus on J&P, tap into BE, understand RC

Identify strong TR & EM

3. TRIGGERS TR

What triggers customers to act? i.e. seeing their neighbor installing solar panels, reading about a more efficient solution in the news.

Overtime, they get disappointed with late and irrelevant replies and triggered to act

4. EMOTIONS: BEFORE / AFTER EM

How do customers feel when they face a problem or a job and afterwards?
i.e. lost, insecure > confident, in control - use it in your communication strategy & design.

- × Disappointed - after they do not get instant replies for their queries
- × Dejected - when they get irrelevant replies even after waiting for a long time

10. YOUR SOLUTION SL

If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality.
If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behavior.

- Creating a Customer Care Registry
- Simple User creation process
- Customers can raise their queries to the experts
- Individual agents will be assigned to each customer
- Their queries will be answered earnestly
- Customers can also check the status of their queries

8. CHANNELS of BEHAVIOUR CH

8.1 ONLINE
What kind of actions do customers take online? Extract online channels from #7

8.2 OFFLINE
What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development.

ONLINE:

1. https://www.helpdesk.com/
2. https://www.google.com/
3. https://www.quora.com/

OFFLINE:

1. Asking friends and colleagues
2. Take actions themselves

Identify strong TR & EM

4. REQUIREMENT ANALYSIS

4.1 Functional Requirements

- A functional requirement defines a function of a system or its component, where a function is described as a specification of behaviour between inputs and outputs.
- It specifies "what should the software system do?"
- Defined at a component level
- Usually easy to define
- Helps you verify the functionality of the software

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

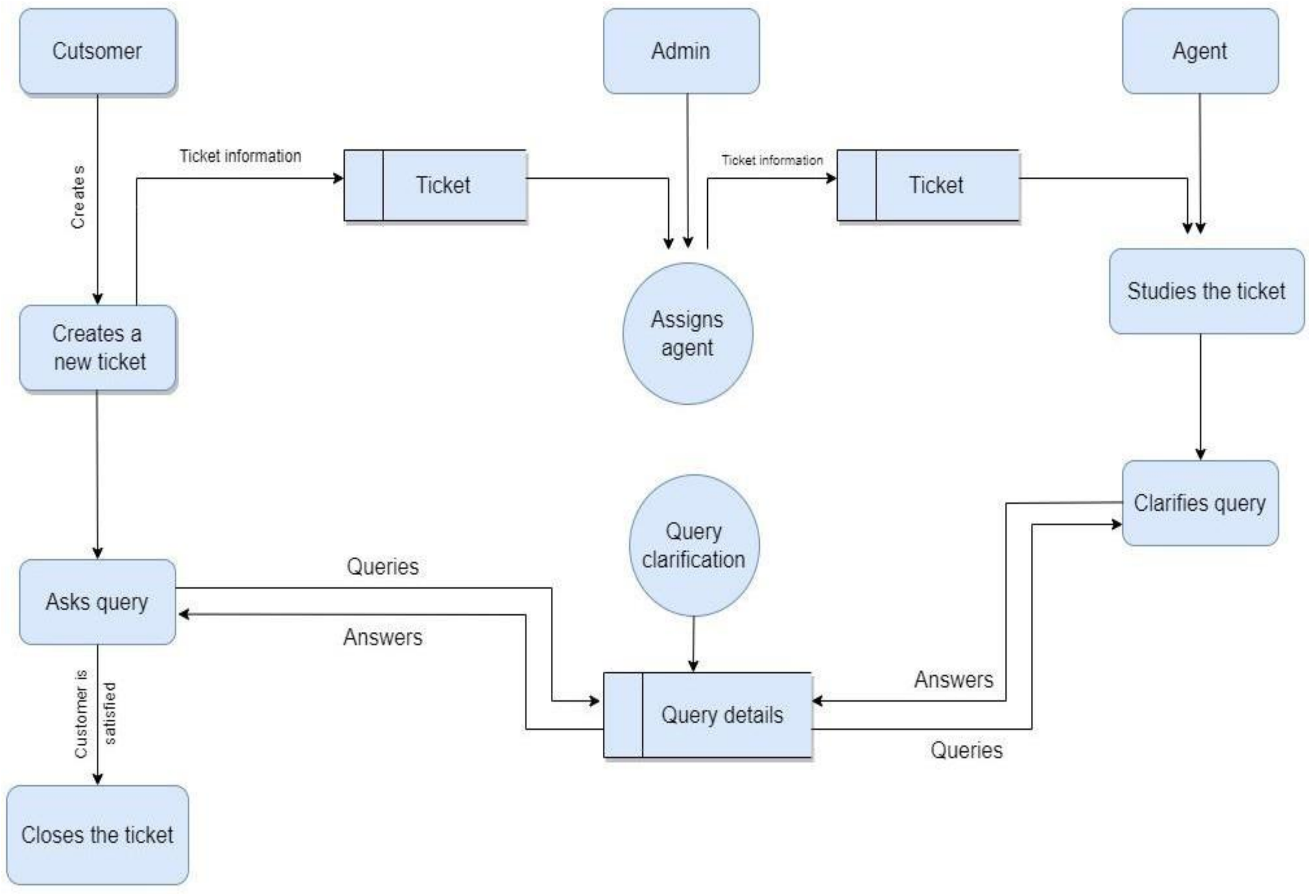
4.2 Non-functional Requirements

- A non-functional requirement defines the quality attribute of a software system
- It places constraint on “How should the software system fulfil the functional requirements?”
- It is not mandatory
- Applied to system as a whole
- Usually more difficult to define
- Helps you verify the performance of the software

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 detailedUI, 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 an8 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.

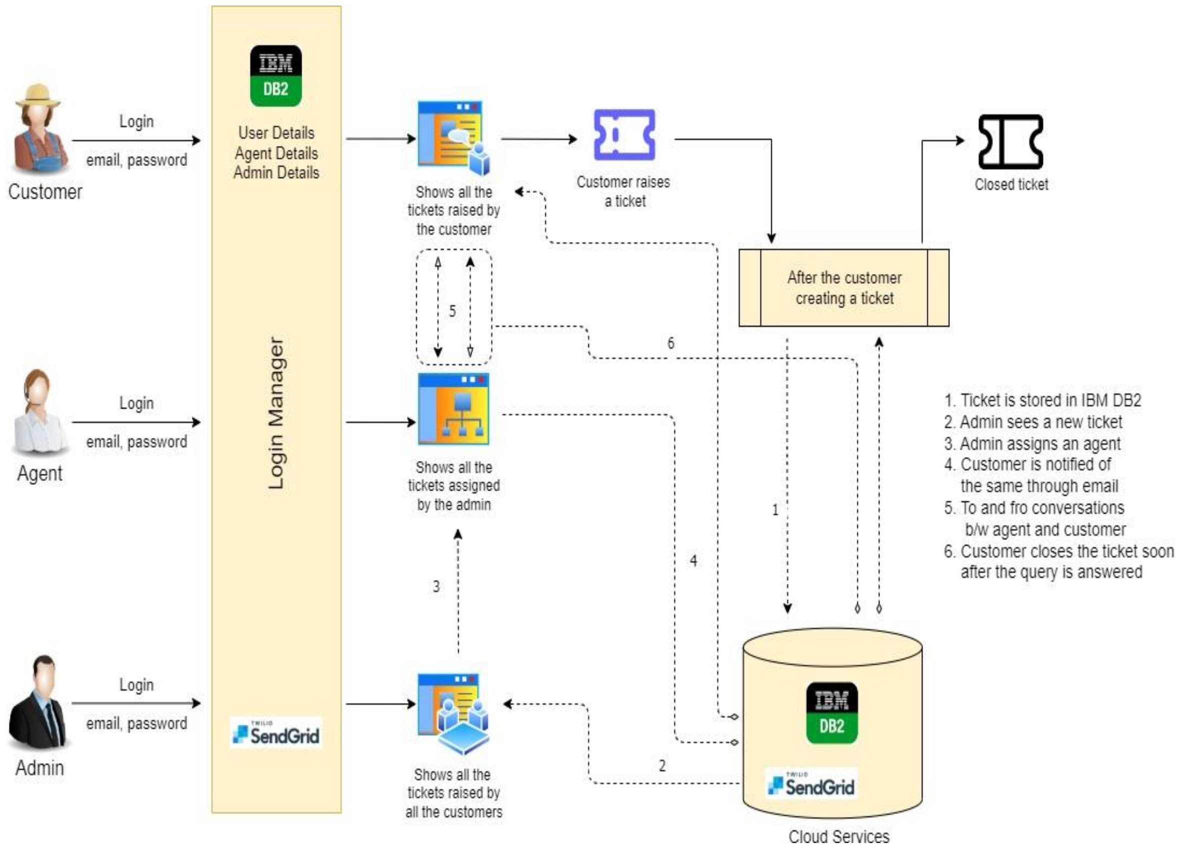
5. PROJECT DESIGN

5.1 Dataflow Diagram



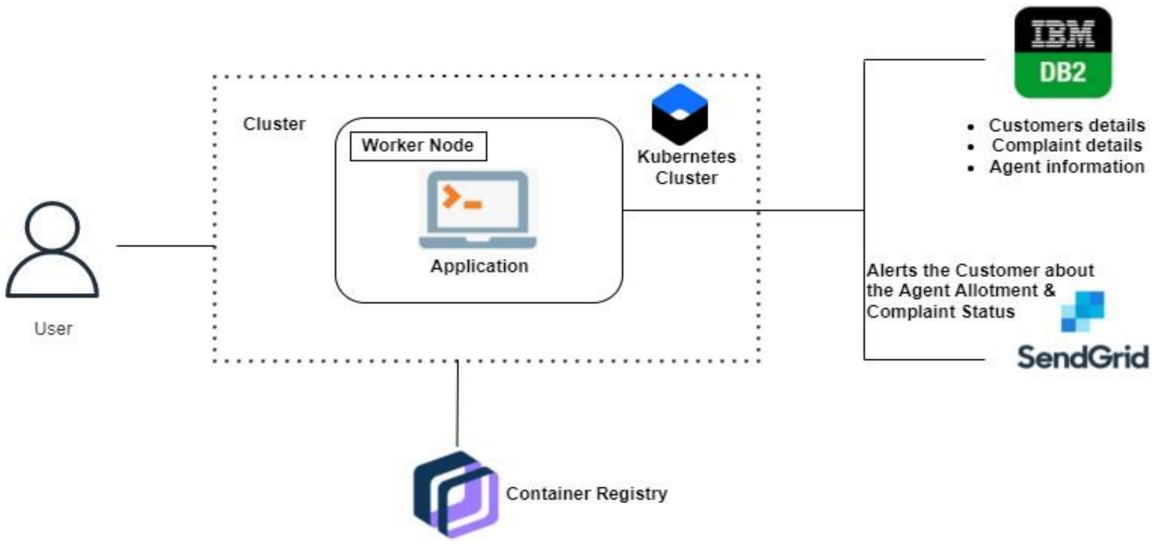
5.2 Solution and Technical Architecture

Solution Architecture



Customer
Login
email, password
Agent
Login
email, password
Admin
Login
email, password
User Details
Agent Details
Admin Details
Login Manager
SendGrid
Shows all the tickets raised by the customer
Customer raises a ticket
Closed ticket
After the customer creating a ticket
Shows all the tickets assigned by the admin
Shows all the tickets raised by all the customers
Cloud Services
1
2
3
4
5
6
1. Ticket is stored in IBM DB2
2. Admin sees a new ticket
3. Admin assigns an agent
4. Customer is notified of the same through email
5. To and fro conversations b/w agent and customer
6. Customer closes the ticket soon after the query is answered

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 applicationby 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 ticketsraised by me and lot more	I get all the info needed inmy dashboard	High	Sprint-1
	Ticket creation	USN-4	As a customer, I can create a new ticketwith the detailed description of my query	I can ask my query	High	Sprint-2
	Address Column	USN-5	As a customer, I can have conversationswith 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 passwordby this option in case I forgot my old password	I get access to my accountagain	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 byentering correct email and password	I can access my account / dashboard	High	Sprint-3
	Dashboard	USN-2	As an agent, I can see all the ticketsassigned 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 thisoption in case I forgot my old password	I get access to my account again	Mediu m	Sprint -4
Admin	Login	USN-1	As an admin, I can login to the application byentering 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 seeingthose tickets	High	Sprint -1
	Agent creation	USN-3	As an admin, I can create an agent forclarifying the customer's queries	I can create agents	High	Sprint -2
	Assigning agent	USN-4	As an admin, I can assign an agent for eachticket created by the customer	Enables agent to clarifythe queries	High	Sprint -2
	Forgot password	USN-4	As an admin, I can reset my password by thisoption in case I forgot my old password	I get access to my account again	Mediu m	Sprint -4

6. PROJECT DESIGN AND PLANNING

6.1 Sprint Planning and Estimation

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

Sprint	User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority
Sprint-4		Forgot password	USN -4	As an agent, I can reset my password by this option incase I forgot my old password	2	Medium
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
Sprint-1		Dashboard	USN -2	As an admin, I can see all the tickets raised in the entire system and lot more	3	High
Sprint-2		Agent creation	USN -3	As an admin, I can create an agent for clarifying the customer's queries	2	High
Sprint-2		Assigning agent	USN -4	As an admin, I can assign an agent for each ticket created by the customer	3	High
Sprint-4		Forgot password	USN -4	As an admin, I can reset my password by this option incase I forgot my old password	2	Medium

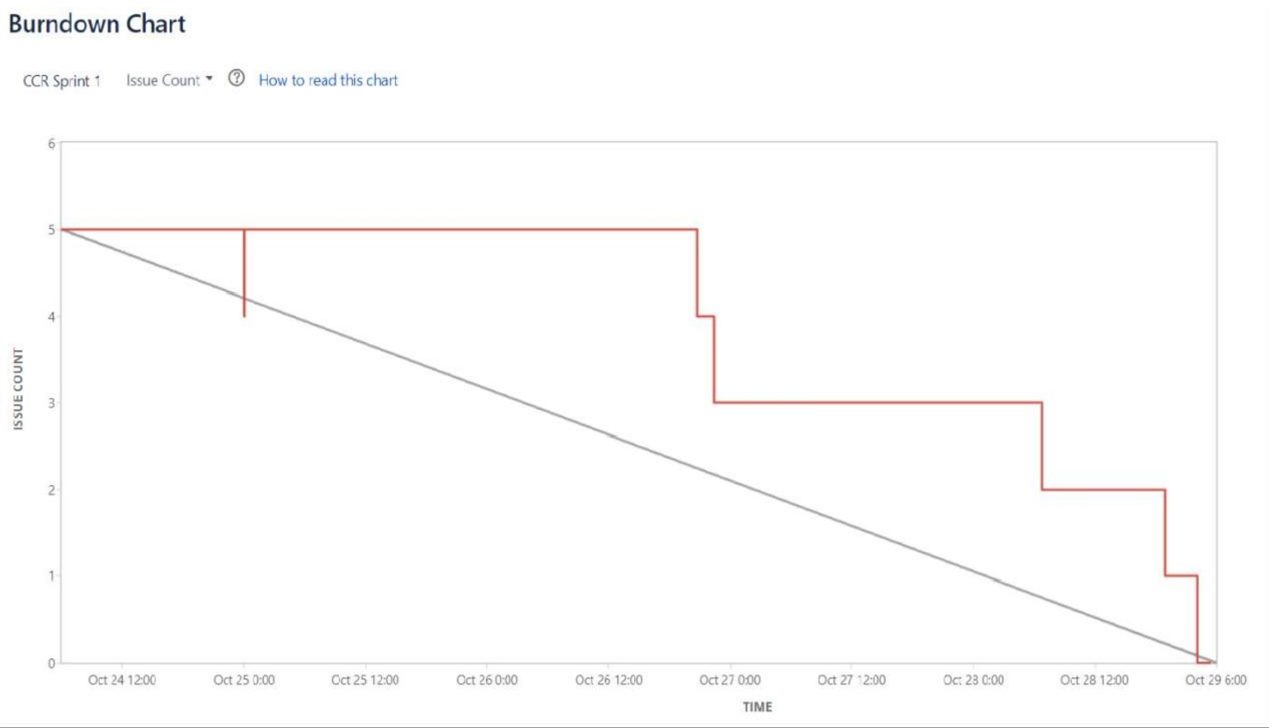
6.2 Sprint Delivery Plan

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date	Story Points Completed	Sprint Release Date (Actual)
Sprint-1	10	6 Days	24 Oct 2022	29 Oct 2022	10	29 Oct 2022

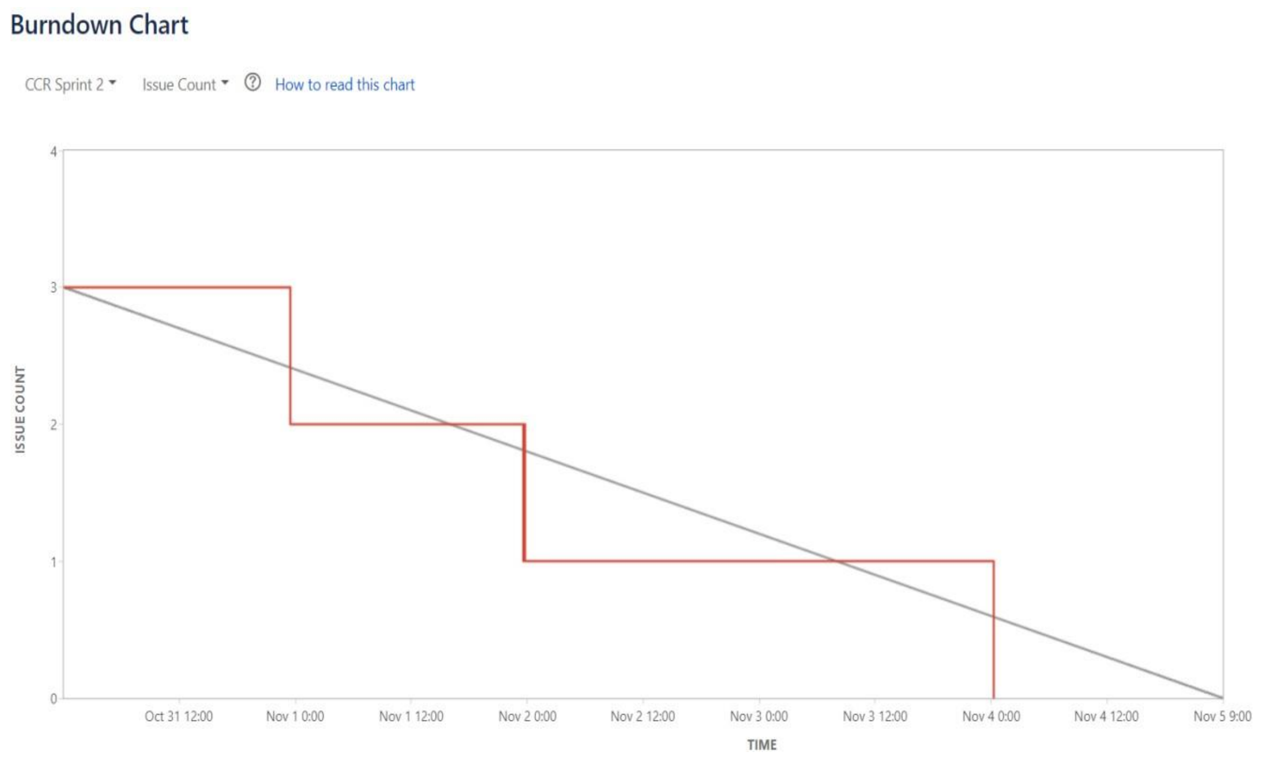
Sprint-2	7	6 Days	31 Oct 2022	05 Nov 2022	7	05 Nov 2022
Sprint-3	11	4 Days	06 Nov 2022	11 Nov 2022	11	09 Nov 2022
Sprint-4	8	4 Days	10 Nov 2022	15 Nov 2022	8	13 Nov 2022

6.3 Reports from JIRA

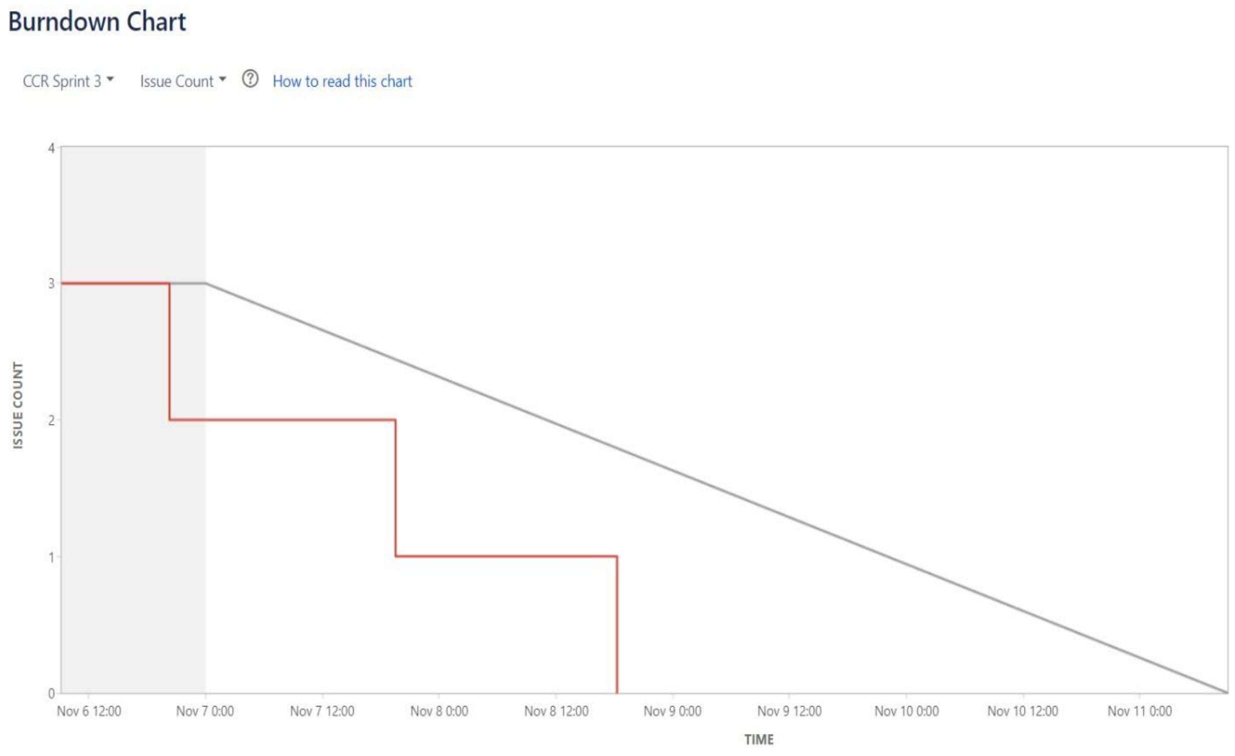
Sprint 1 – Burndown Chart



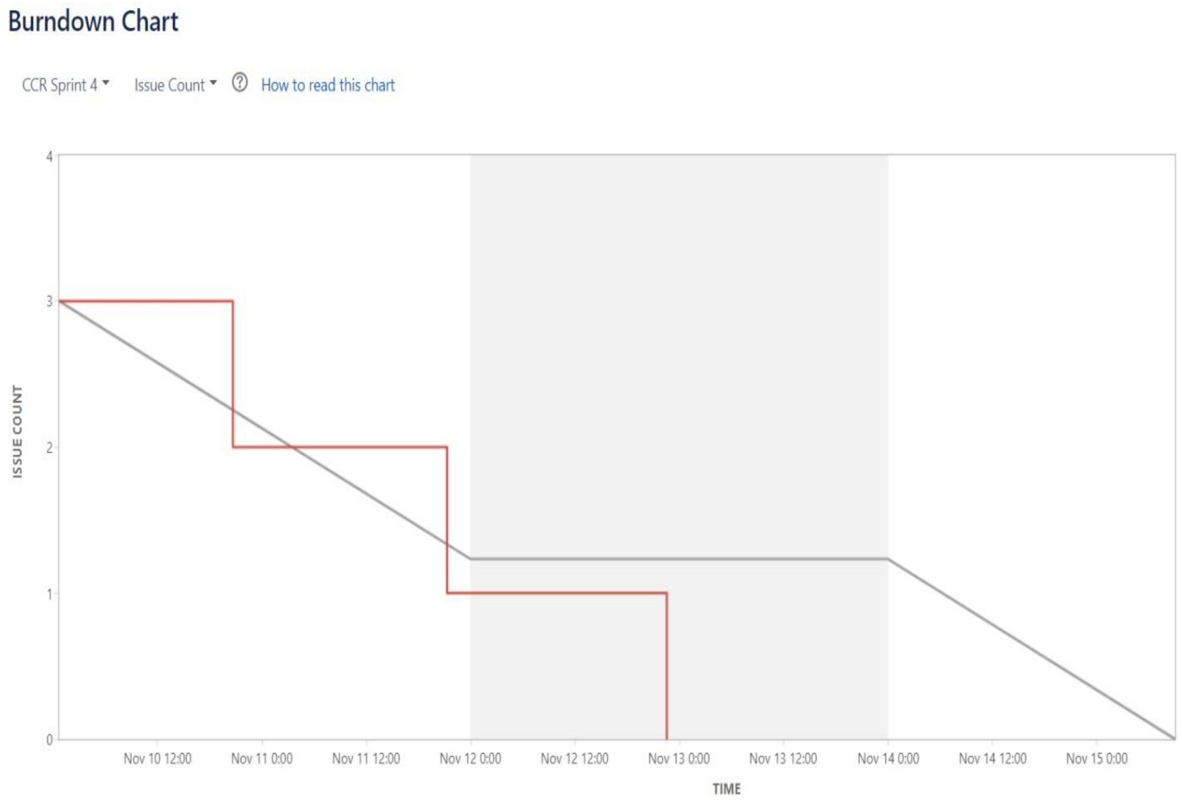
Sprint 2 – Burndown Chart



Sprint 3 – Burndown Chart



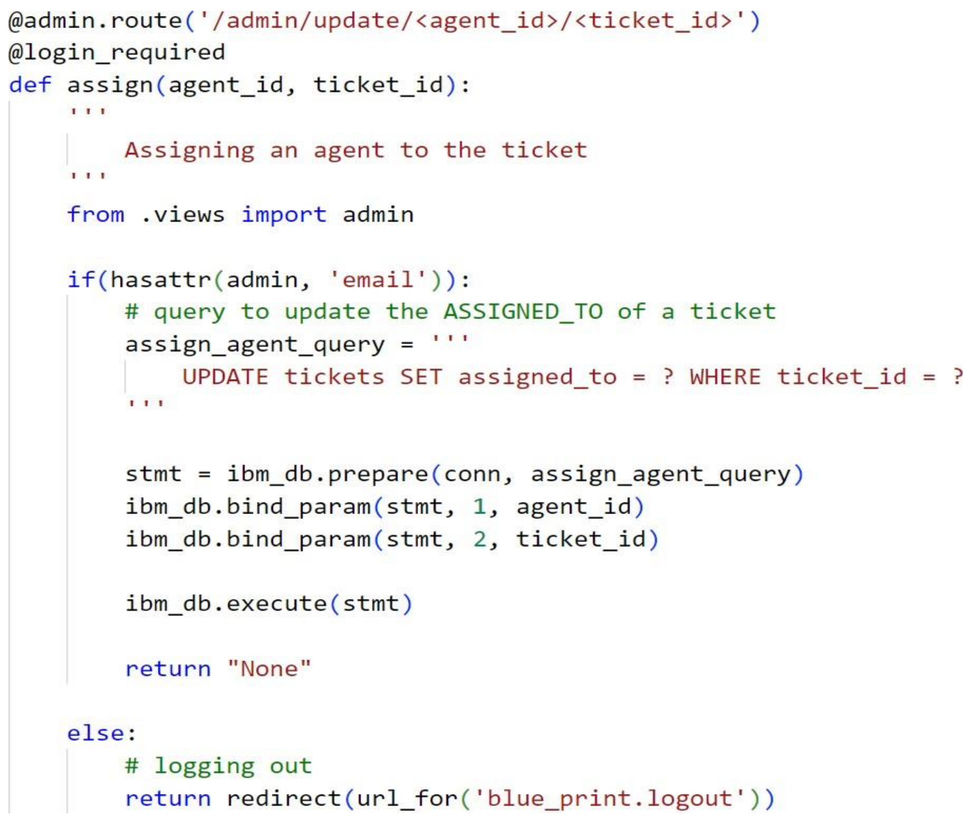
Sprint 4 – Burndown Chart



7. CODING AND SOLUTIONING

7.1 Admin assigning an agent to a ticket

Code:



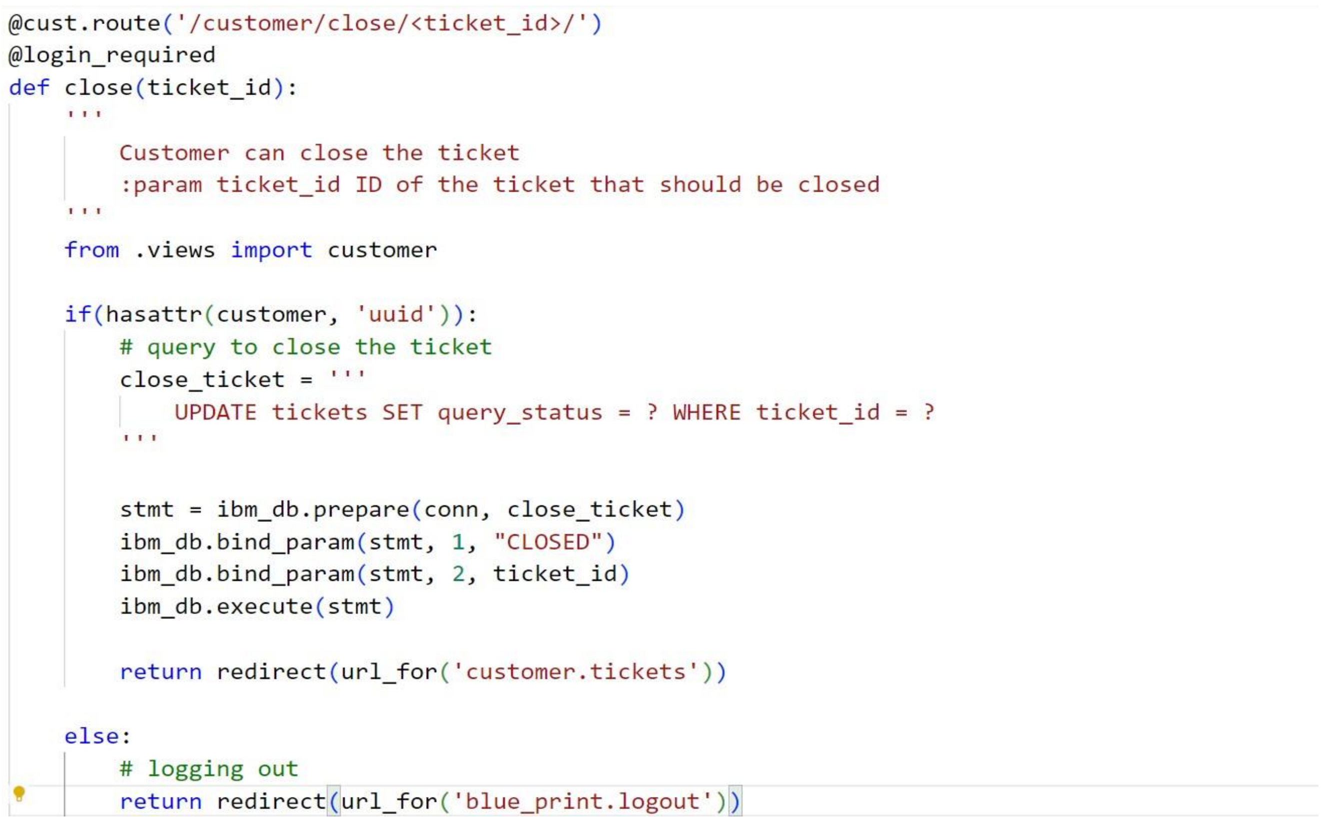
```python
@admin.route('/admin/update/<agent_id>/<ticket_id>')
@login_required
def assign(agent_id, ticket_id):
    '''
        Assigning an agent to the ticket
    '''
    from .views import admin

    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_idwhere 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'))
```

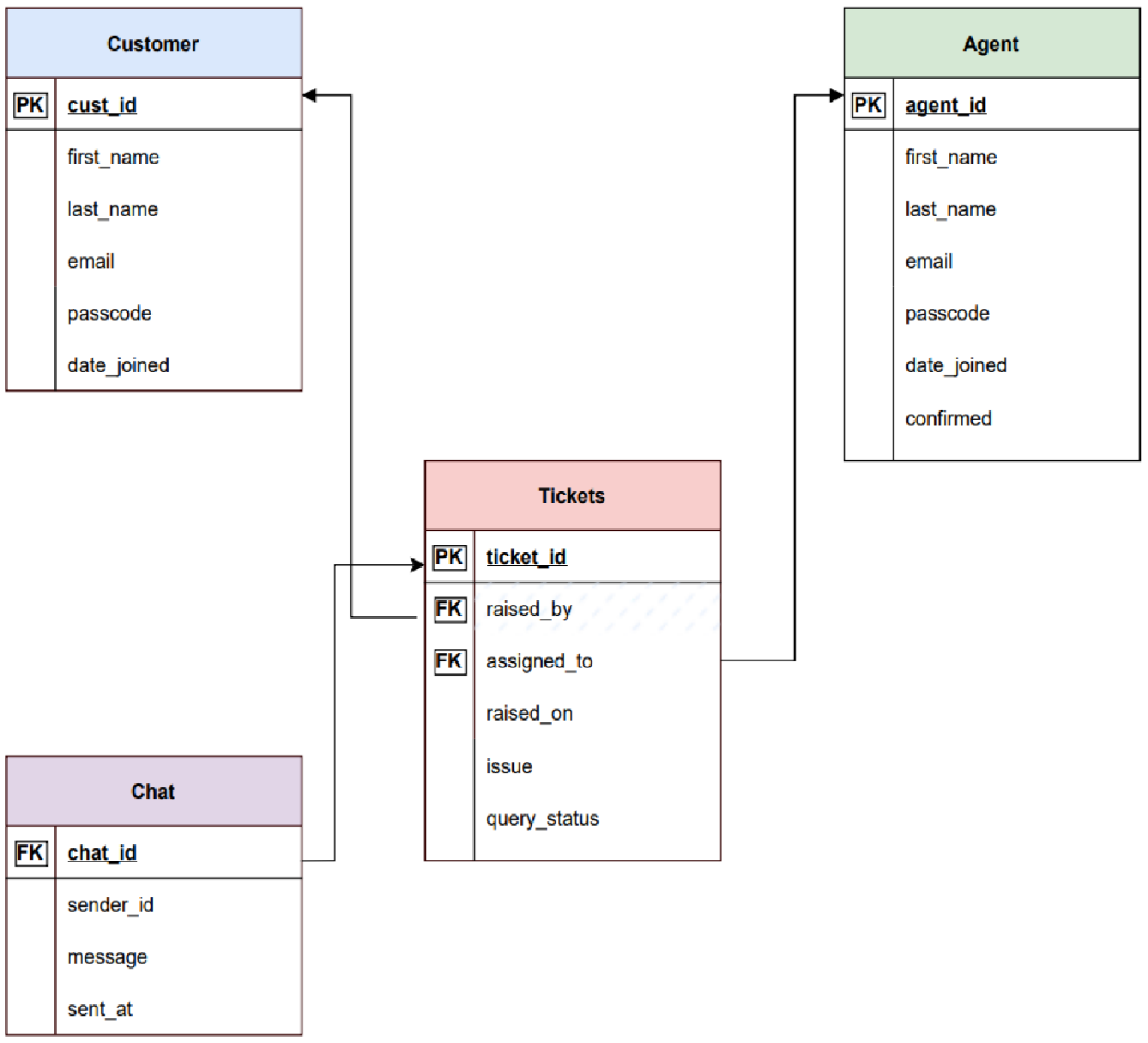
Explanation:

- 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'sdoubts
- 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 adescriptive 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 makeit useful.



8. TESTING

8.1 Test Cases

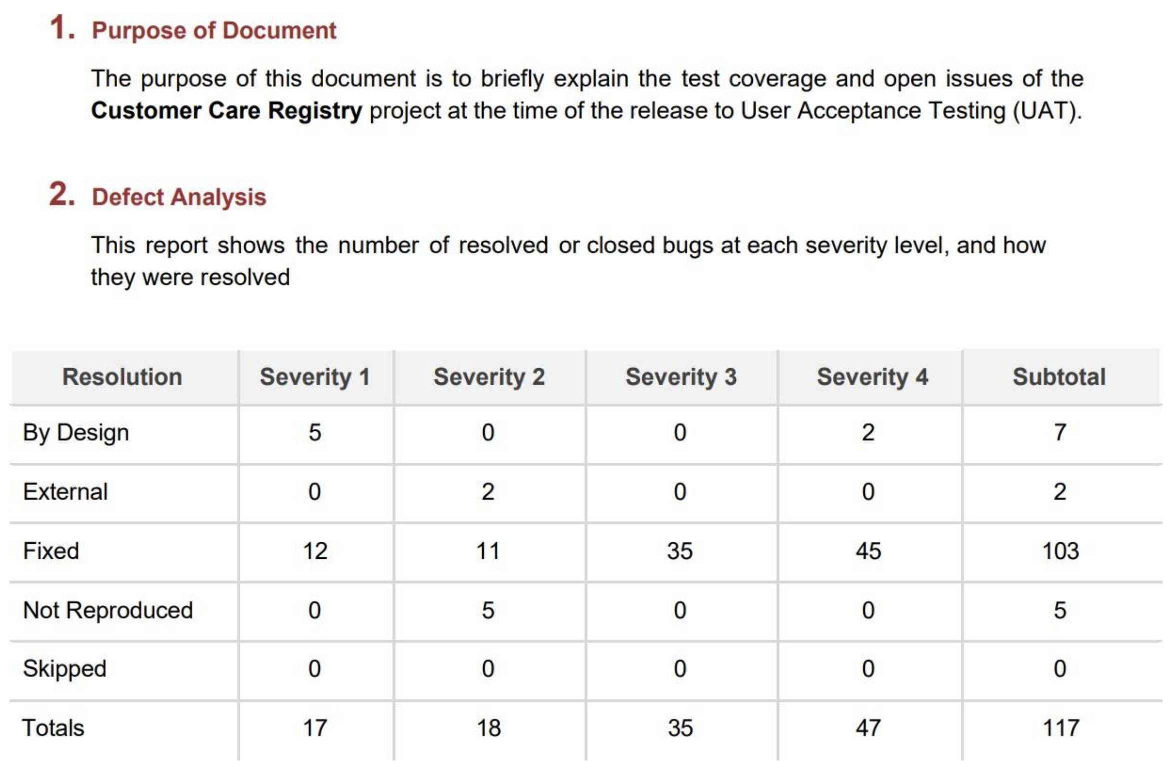
The test case is defined as a group of conditions under which a tester determines whethera software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived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

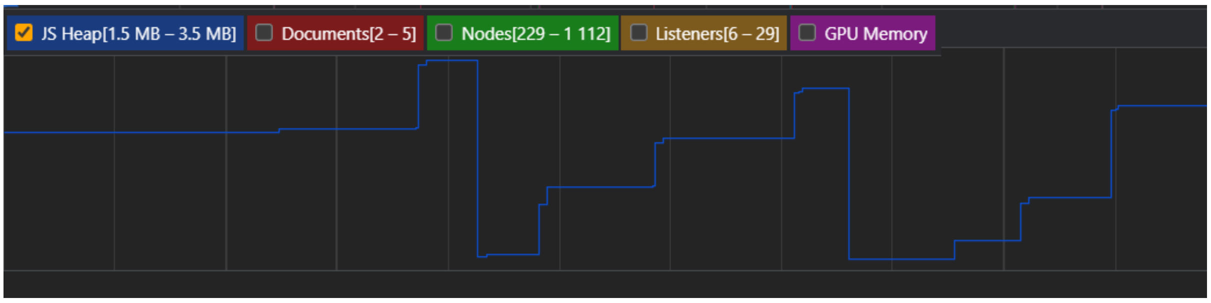
Section	Total Cases	Not Tested	Fail	Pass
Client Application	72	0	0	72
Security	7	0	0	7
Exception Reporting	5	0	0	5
Final Report Output	4	0	0	4

9. RESULTS

9.1 Performance Metrics:

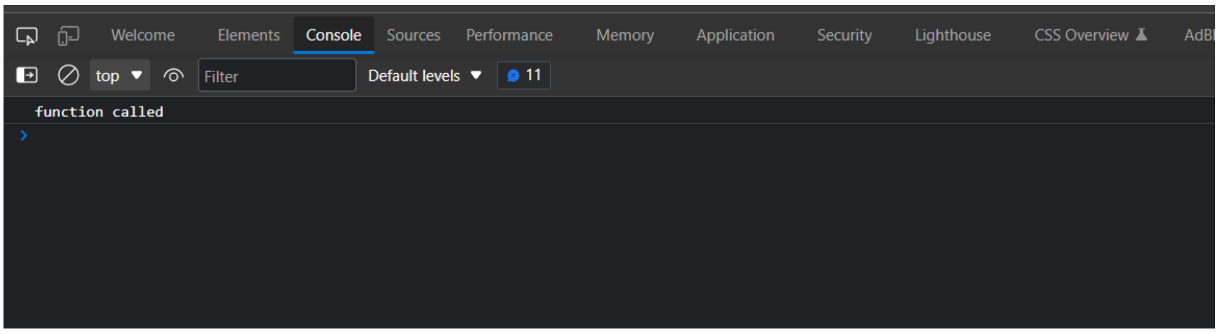
CPU usage:

- ✓ Since all the operations run using Flask is in server-side, the client (browser) need not worryabout 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:

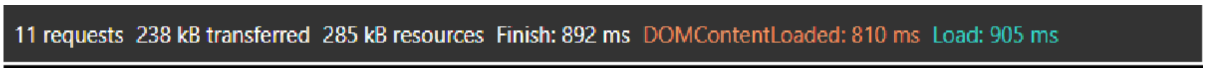
- ✓ Since all the backend functions are done using flask, any exceptions / errors rising are well-handled. Though they appear, user's interaction with the site is not affected in any way



```
function called
>
```

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



```
11 requests  238 kB transferred  285 kB resources  Finish: 892 ms  DOMContentLoaded: 810 ms  Load: 905 ms
```

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 downthe 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 willbe 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 alsothe responsibility of the admin, to create an agent.

12. FUTURE SCOPE

Our application is not finished yet. There are many rooms for improvement. Some ofthem 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

APPENDIX

Flask:

- ✓ Flask is a micro web framework written in Python. It is classified as a microframework because itdoes not require particular tools or libraries
- ✓ It has no database abstraction layer, form validation, or any other components where pre-existingthird-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 deliversoftware in packages called containers

<u>**SOURCE CODE (Only Samples)**</u>

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_forimport
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 sessionuser =
    ""
    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 aatributefrom
    .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 outreturn
    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 ticketstmt
    = 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 chatif
    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_loaderdef
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-16368-1659612100

Project Demo Link:

Demonstration video Link