CUSTOMER CARE REGISTRY
PROJECT REPORT

1.INTRODUCTION

1.1. Project Overview

This Application has been developed to help the customer in processing their complaints. The customers can raise the ticket with a detailed description of the issue. An Agent will be assigned to the Customer to solve the problem. Whenever the agent is assigned to a customer they will be notified with an email alert. Customers can view the status of the ticket till the service is provided.

1.2. Purpose

The purpose of this application is to create an effective solution which requires understanding the true problem and the person who is experiencing it.This application is develop to provide advice and help to the customers by raising tokens. Customer care is more than just delivering the services that customer expect and providing the right technical support. It’s about meeting their emotional needs and fostering relationships. To do so, we must treat the customers how they wanted to be treated. We need to listen to each individual’s needs and find the best solutions.

2. LITERATURE SURVEY

2.1. Existing System

The customer's complaint often not documented because the customer services record all complaint manually one by one and the amount of complaint increases day by day. The customer service often answers the same question rom different customer. There is no information for the customer about the progress of the complaint and it is difficult to monitor the complaint and report. Cloud-based solution framework, user found it difficult to communicate with customer service representsative during faulty experience, and follows traditional way of acquiring and managing data or information.

2.2. References

1. A Proposed Cloud Based Solution for Customer Satisfaction in Telecommunication Industry,Nurulhuda Mustafa, Lew Sook Ling, Siti Fatimah Abdul Razak,2019.
2. Using SMS and Web Technology in Mobile Government Information Services Platform Hua Zhang ,Fayu Wang 2010.
3. Real World Smart Chatbot for Customer Care using a Software as a Service (SaaS) Architecture Godson Michael D'silva, Sanket Thakare,Sharddha More and Jeril Kuriakose 2017.
4. Virtual Customer Service Agents: Using Social Presence and Personalization to Shape Online Service Encounter Tibert Verhagen, Jaap van Nes, Frans Feldberg, Willemijn van Dolen, Ph.D 2014.
5. Online Complaint Registration System to Municipality A.Prassana, Dr. A.V. Senthil Kumar 2020.
6. Implementation Of 'ASR4CRM': An Automated Speech Enabled Customer Care Service System Aderemi A. Atayero, Charles K. Ayo, Ikhu-Omoregbe Nicholas and Azeta Ambrose 2009.
7. A Blockchain and AutoML Approach for Open and Automated Customer Service Zhi Li, Hanyang Guo, Wai Ming Wang, Yijiang Guan, Ali Vatankhah Barenji, George Q. Huang, Kevin S. McFall, and Xin Chen 2019.
8. Using Authentic Leadership and Mindfulness as Internal Marketing Mechanism for Enhancing Proactive Customer Service Performanc C. M. Wu ,T. J. Chen , Y. D. Lee , T. F. Chen 2016.
9. An Application of SMS Technology for Customer Service Centre Ariff Idris, Abd. Samad Hasan Basari, Nur Hanisah Zubir, 2009.
10. Online Helpdesk Support System for Handling Complaints and Service Cadelina Cassandra, Sugiarto Hartono, Marisa Karsen 2019.

2.3 Problem Statement Definition

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. For organizations, and for product and design teams, there can be a number of reasons why a

product could fail. But not taking the time to consider a customer’s onditions and their current situation could potentially harm your product’s future. By working with a problem statement you can make sure you are defining a customer’s experience and attempting to transform your product for the better.

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

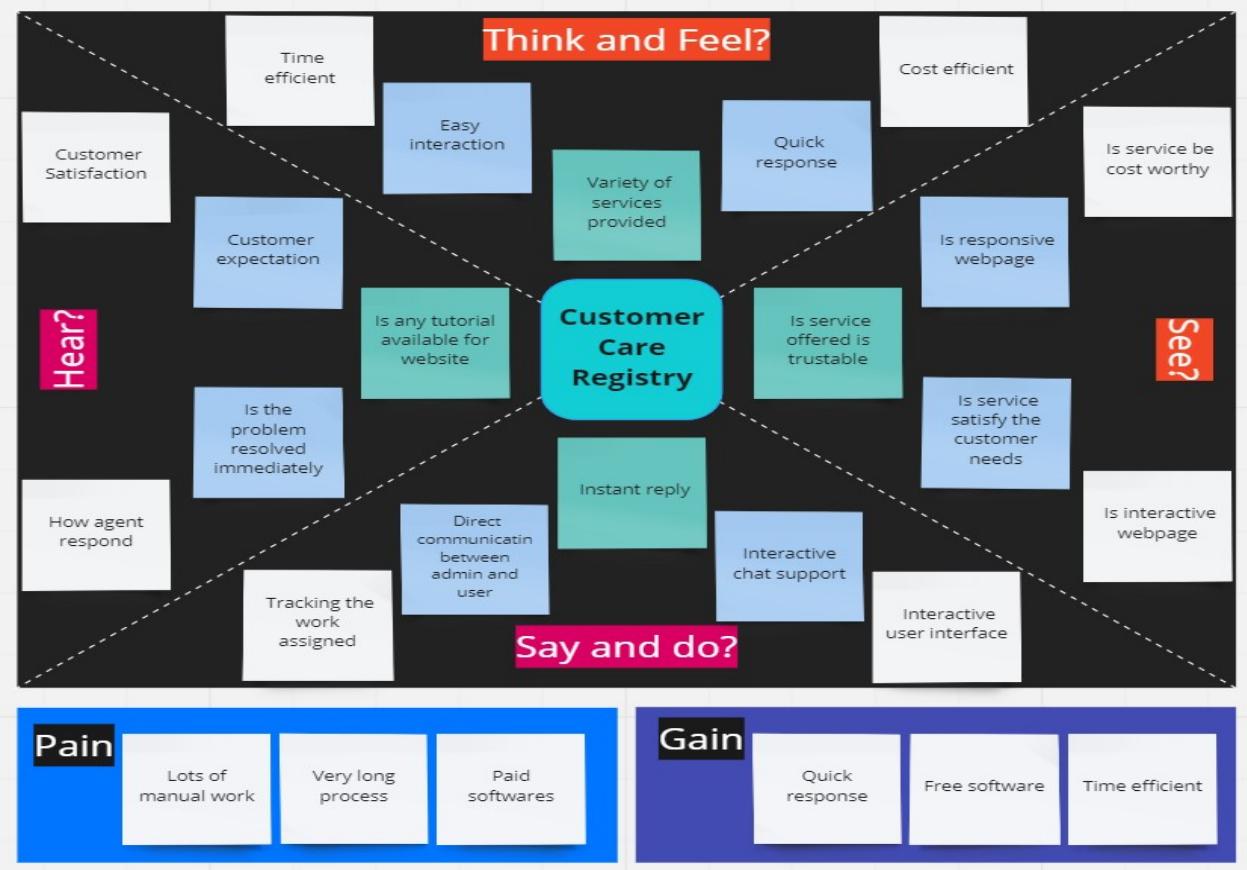
The user can register for an account. After the login, they can create the complaint with a description of the problem they are facing. Each user will be assigned with an agent.

The user can view the status of their complaint through email.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas

An 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 helps teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is 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 & Brainstorming

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

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

10 minutes to prepare
1 hour to collaborate
2-8 people recommended

Share template feedback

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

10 minutes

A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B **Set the goal**
Think about the problem you'll be focusing on solving in the brainstorming session.

C **Learn how to use the facilitation tools**
Use the Facilitation Superpowers to run a happy and productive session.

Open article →

1 Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM

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. For organizations, and for product and design teams, there can be a number of reasons why a product could fail. But not taking the time to consider a customer's conditions and their current situation could potentially harm your product's future. By working with a problem statement you can make sure you're defining a customer's experience and attempting to transform your product for the better.

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2 Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!

3 Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Customer

- Customer satisfaction
- Customer expectation
- Customer rating
- Customer queries

Chat box

- Provide chat box
- Live chat
- Quick chat response
- Interactive chat

Services

- Tracking of services
- Provides service details
- Stores customer detais
- Stores agent details

Feedback

- Customer satisfaction
- Customer feedback
- Customer rating
- Custome review

Information & Security

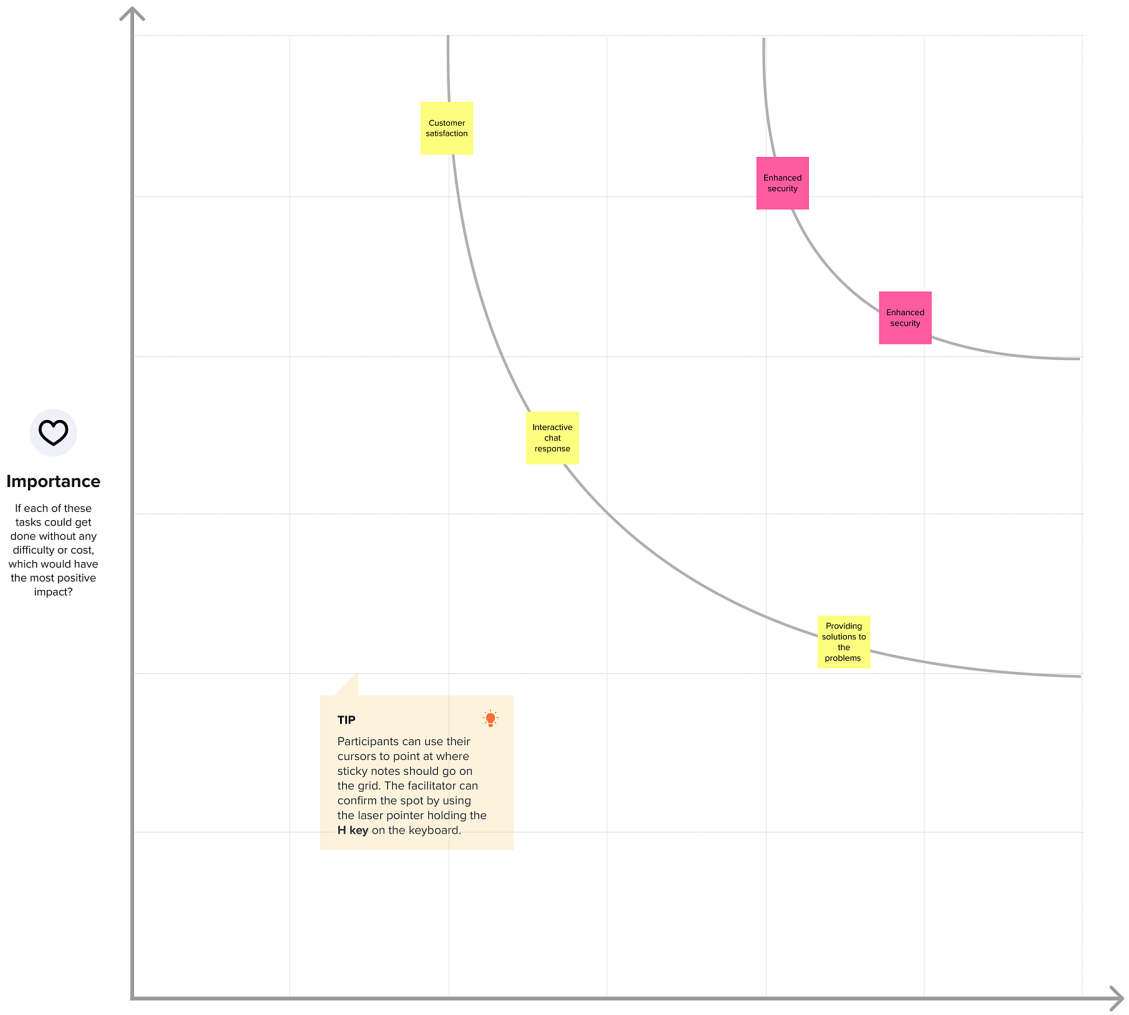
- Stores data
- Data privacy
- Enhanced security
- Data protection

4

Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

20 minutes



Importance

If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?

Feasibility

Regardless of their importance, which tasks are more feasible than others? (Cost, time, effort, complexity, etc.)

3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	Problem phase describes that the 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. For organizations, and for product and design teams, there can be a number of reasons why a product could fail. But not taking the time to consider a customer's conditions and their current situation could potentially harm your product's future. By working with a problem statement you can make sure you are defining a customer's experience and attempting to transform your product for the better. So the problem statement mainly defines to solve customer issues using Cloud Application Development.
2.	Idea / Solution description	Solution phase describes the web application that has been developed to help the customer in processing their complaints. The customers can raise the ticket with a detailed description of the issue. An Agent will be assigned to the Customer to solve the problem. Whenever the agent is assigned to a customer they will be notified with an email alert. Customers can view the status of the ticket till the service is provided.
3.	Novelty / Uniqueness	Customer care registry provides instant reply and the assigned work can be tracked at any time and provides tutorial for website.

4.	Social Impact / Customer Satisfaction	Customer care registry provides direct communication between admin and user and provides variety of services.
5.	Business Model (Revenue Model)	Customer care registry can be linked with industrial organizations to provide customer care support.
6.	Scalability of the Solution	Customer care registry provides an environment which has both time and cost efficient.

3.4 Problem Solution fit

Project Title: Customer Care Registry **Project Design Phase-I - Solution Fit Template** **Team ID:** PNT2022TMID02641

4. REQUIREMENT ANALYSIS

4.1 Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail
FR-2	User Confirmation	Confirmation via Email. Confirmation via OTP.
FR-3	User Login	Login via google with Email ID and password.
FR-4	Admin Login	Login via google with Email ID and password.
FR-5	Agent Login	Login via google with Email ID and password.
FR-6	User Request	Description of the queries.
FR-7	Agent Replay	Solving the customers queries.

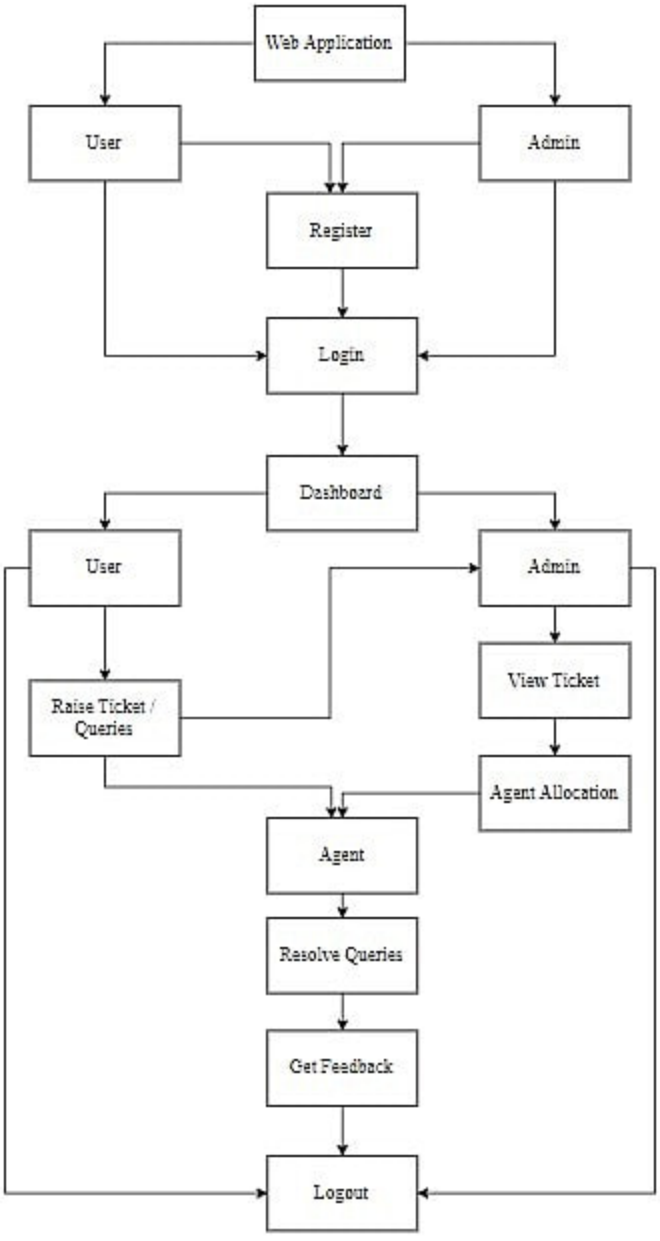
4.2 Non-Functional requirements

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Easily used by everyone, to provide a solution to a problem and our web application is flexible for all type of devices
NFR-2	**Security**	High end security to track of login authentication.
NFR-3	**Reliability**	Increased reliability and measure portability.
NFR-4	**Performance**	Every user is allotted a individual agent by the admin, effective development of web application.
NFR-5	**Availability**	User can interact with their respective agents 24*7 by following proper user-agent guidelines.

NFR-6	**Scalability**	Increase in user's request results in increase in allotment of agent therefore data storage also increases.

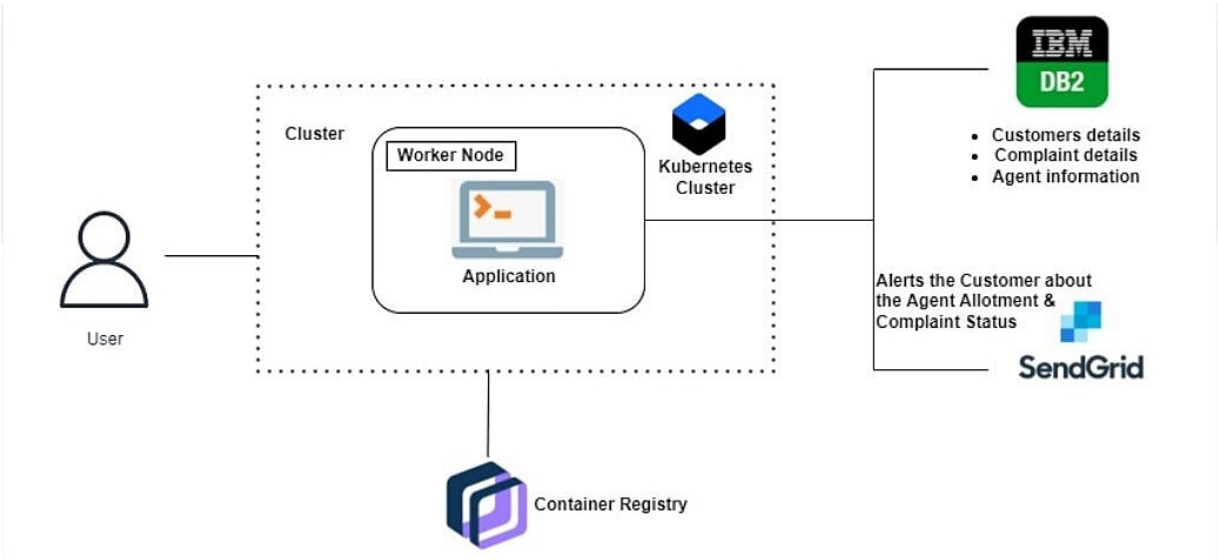
5. PROJECT DESIGN

5.1 Data Flow Diagrams

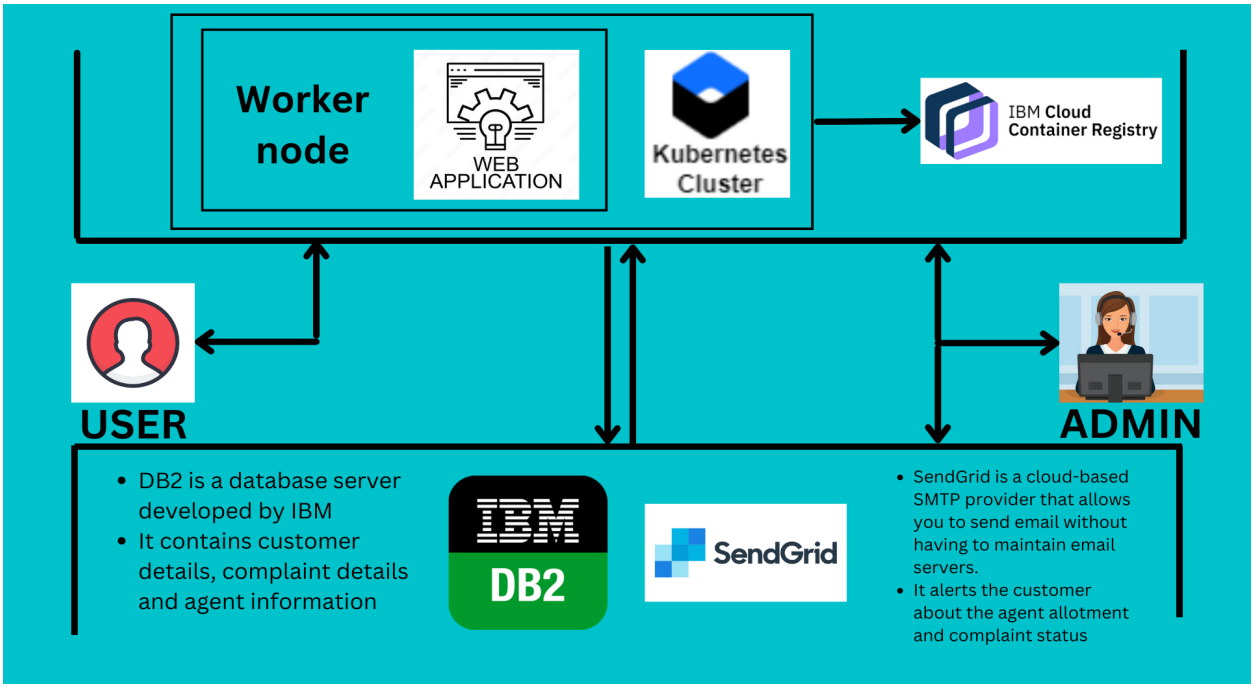


5.2 Solution & Technical Architecture

Solution Architecture



Technical Architecture:



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web or 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 register and access my account / dashboard with email and password	High	Sprint-1
		USN-2	As a customer, I can register for the application through Facebook	I can register and access the dashboard with Facebook Login	Low	Sprint-1
		USN-3	As a customer, I can register for the application through Gmail	I can register and access the dashboard with Gmail Login	Medium	Sprint-1
		USN-4	As a customer, I will receive confirmation email once I have registered for the application	I can receive confirmation email after registering the application	High	Sprint-1

	Login	USN-5	As a customer, I can log into the application by entering email & password	I can login and access the dashboard with email and password	High	Sprint-1
	Forget password	USN-6	As a customer, I can reset my password if I forgot my old password	I can get access to my dashboard after resetting my password	Medi um	Sprint-1
	Dashboard	USN-7	As a customer, I can view all the queries raised by me	I can get all the information needed in my dashboard	Medi um	Sprint-2
	Raise ticket	USN-8	As a customer, I can raise the ticket with a detailed description of my issue	I can able to raise my queries	High	Sprint-2
	Ticket details	USN-9	As a customer, I can see the current status of my raised ticket	I can able to view the ticket status at any time	Medi um	Sprint-2
	Notification	USN-10	As a customer, I could be receiving the email notification about the agent assigned to me	I can able to track the agent assigned to me	Medi um	Sprint-2

Agent (Web or mobile user)	Registration	USN-1	As a agent, I can register for the application by entering my email, password, and confirming my password.	I can register and access my account / dashboard with email and password	High	Sprint-2
		USN-2	As a agent, I can register for the application through Gmail	I can register and access the dashboard with Gmail Login	Medi um	Sprint-2
		USN-3	As an agent, I will receive confirmation email once I have registered for the application	I can receive confirmation email after registering the application	High	Sprint-2
	Login	USN-4	As an agent, I can log into the application by entering email & password	I can login and access the dashboard with email and password	High	Sprint-3
	Forget password	USN-5	As an agent, I can reset my password if I forgot my old password	I can get access to my dashboard after resetting my password	Medi um	Sprint-3

	Dashboard	USN-6	As an agent, I can view all the queries raised by the customers which was assigned to me	I can view all the information and queries raised by the customers	Medi um	Sprint-3
	Resolve queries	USN-7	As an agent, I can give efficient solution to the queries raised by customers	I can able to provide precise solution	High	Sprint-3
	Ticket details	USN-8	As an agent, I can see the current status of the ticket raised by customers which was assigned to me	I can able to view the ticket status at any time	High	Sprint-3
Administrat or (Web or mobile user)	Registration	USN-1	As an administrator, I can register for the application by entering my email, password, and confirming my password.	I can register and access my account / dashboard with email and password	High	Sprint-3
		USN-2	As an administrator, I can register for the application through Gmail	I can register and access the dashboard with Gmail Login	Medi um	Sprint-3

		USN-3	As an administrator, I will receive confirmation email once I have registered for the application	I can receive confirmation email after registering the application	High	Sprint-3
	Login	USN-4	As an administrator, I can log into the application by entering email & password	I can login and access the dashboard with email and password	High	Sprint-4
	Forget password	USN-5	As an administrator, I can reset my password if I forgot my old password	I can get access to my dashboard after resetting my password	Medi um	Sprint-4
	Dashboard	USN-6	As an administrator, I can view all the queries raised by the customers	I can view all the information and queries raised by the customers	Medi um	Sprint-4
	Manage tickets	USN-7	As an administrator, I can able to assign the tickets to the agent raised by customers	I can able to assign raised tickets to the agent	High	Sprint-4

	Ticket details	USN-8	As an administration, I can able to track the work assigned to the agent	I can able to track the ticket status at any time	High	Sprint-4
	Notification	USN-9	As an administrator, I can able to send email notification to the customers about their assigned agents to solve their queries	I can able to send email notification to the customers about their assigned agents	High	Sprint-4

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

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	9	6 Days	24 Oct 2022	29 Oct 2022	9	29 Oct 2022
Sprint-2	9	6 Days	31 Oct 2022	05 Nov 2022	9	05 Nov 2022
Sprint-3	8	6 Days	07 Nov2022	12 Nov 2022	8	12 Nov 2022
Sprint-4	6	6 Days	14 Nov2022	19 Nov 2022	6	19 Nov 2022

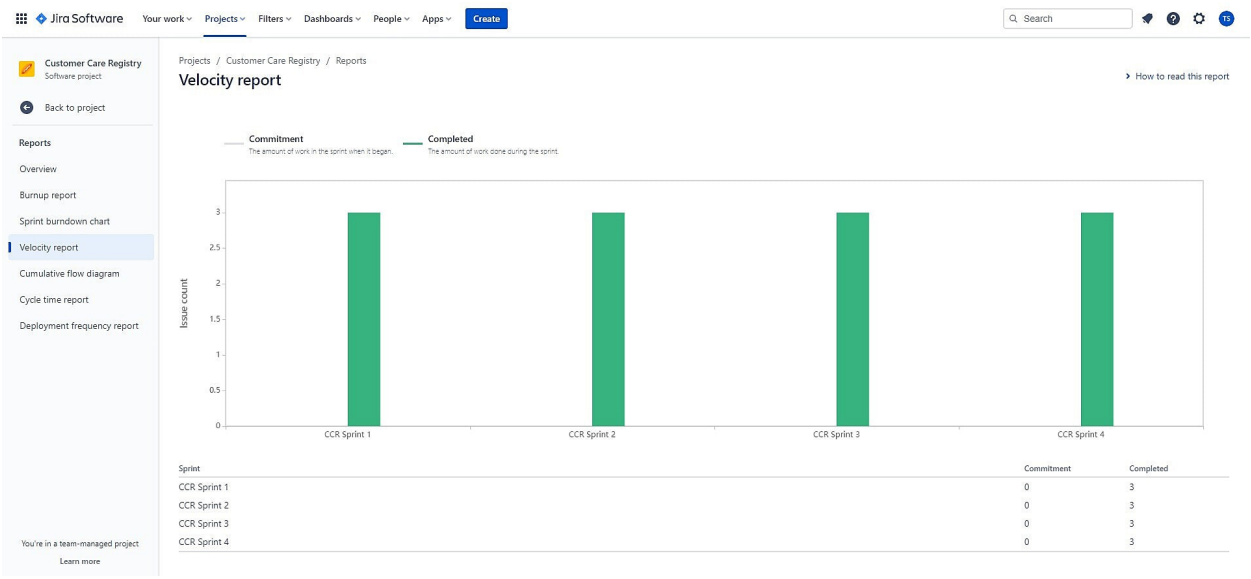
6.2 Sprint Delivery Schedule

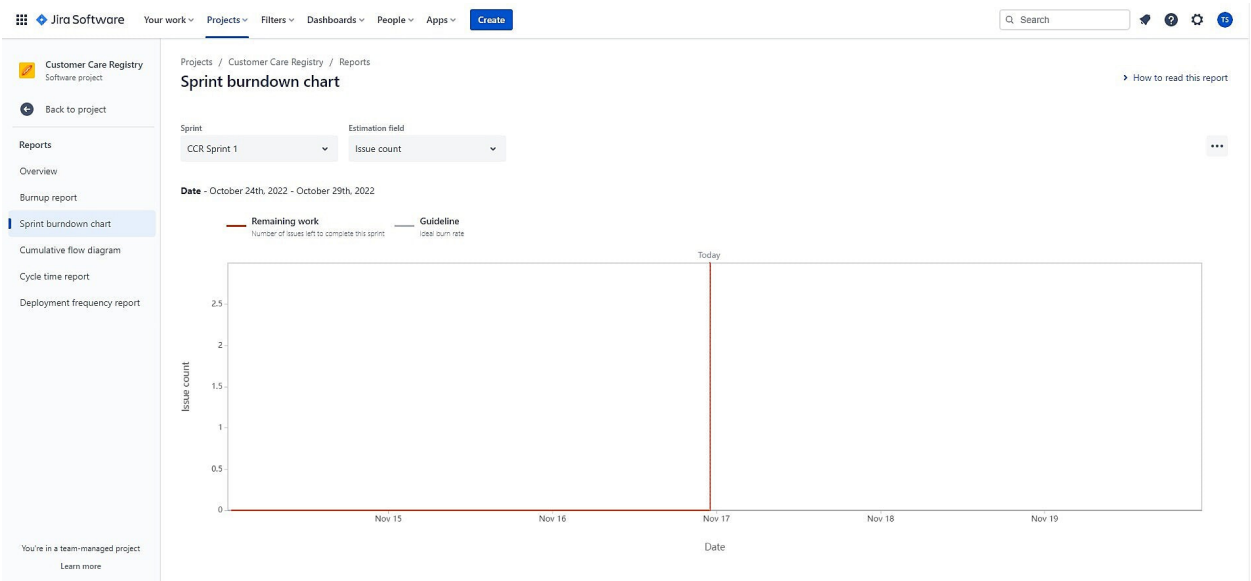
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint - 1	Registration	USN-1	Registration Page – User, Admin, Agent	3	High	SHARMILA M
	Login	USN-2	Login Page – User, Admin, Agent	3	Medium	SREEVARSHINI S
	Forget password	USN-3	Forgot Password Page – User, Admin, Agent	3	Medium	SNEHA T
Sprint - 2	Dashboard	USN-1	Dashboard – User can raise a ticket with a detailed description of their issue and can view the current status of the raised ticket and can view all the raised tickets	3	High	TAMILARASAN S

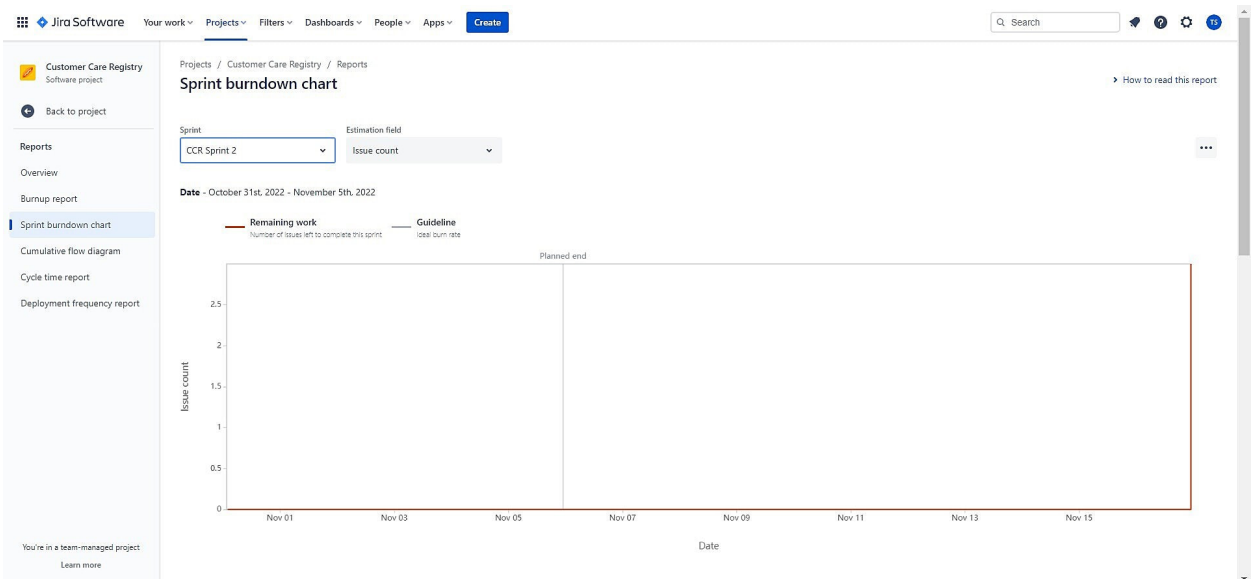
		USN-2	Dashboard – Admin can view all the tickets raised by the users and can able to assign the tickets to the agent raised by users and can able to track the work assigned to the agent	3	High	SHARMILA M
		USN-3	Dashboard – Agent can view all the assigned tickets raised by the user and can give efficient solution to the queries raised by users and can able to see the current status of the tickets assigned by admin	3	High	SREEVARSHINI S
Sprint-3	IBM DB2	USN-1	Connect IBM DB2 with python	3	High	SNEHA T
	Watson Assistant	USN-2	Chatbot – User, Admin, Agent	3	Medium	TAMILARASA S
	SendGrid	USN-3	SendGrid enables admin to send emails to the user about the assigned agent to solve the customer queries	2	Medium	SHARMILA M
Sprint-4	Docker image	USN-1	Create docker image for flask app	2	Medium	SREEVARSHINI S
	IBM Container Registry	USN-2	Upload the docker image to the IBM	2	Medium	SNEHA T

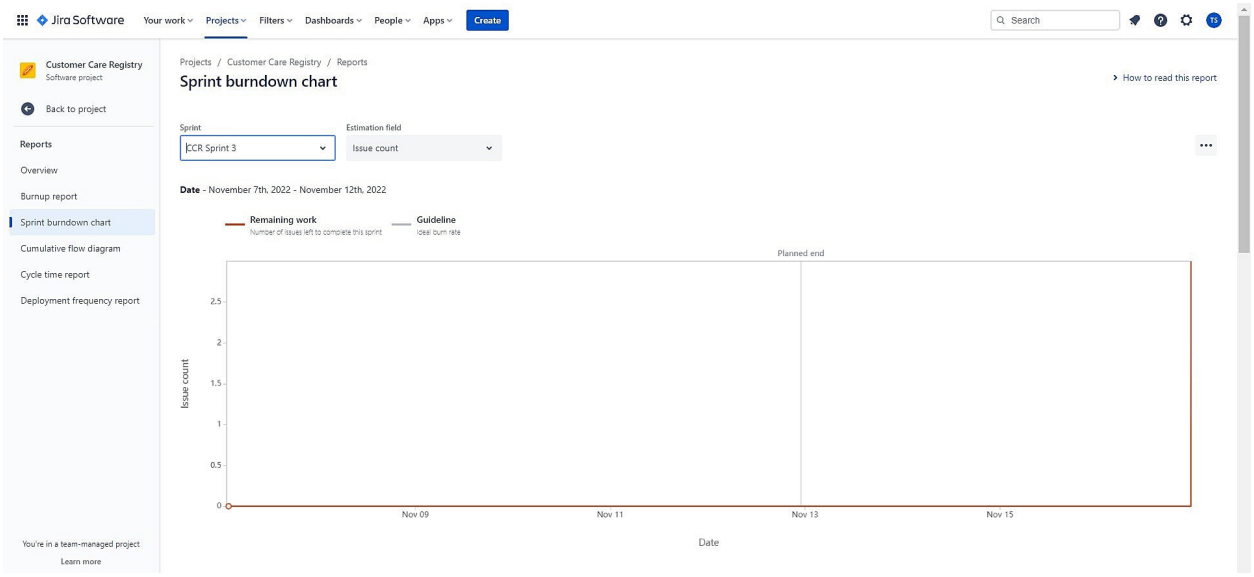
			Container Registry			
	Kubernetes	USN-3	Deploy the docker image on Kubernetes cluster	2	Medium	TAMILARASAN S

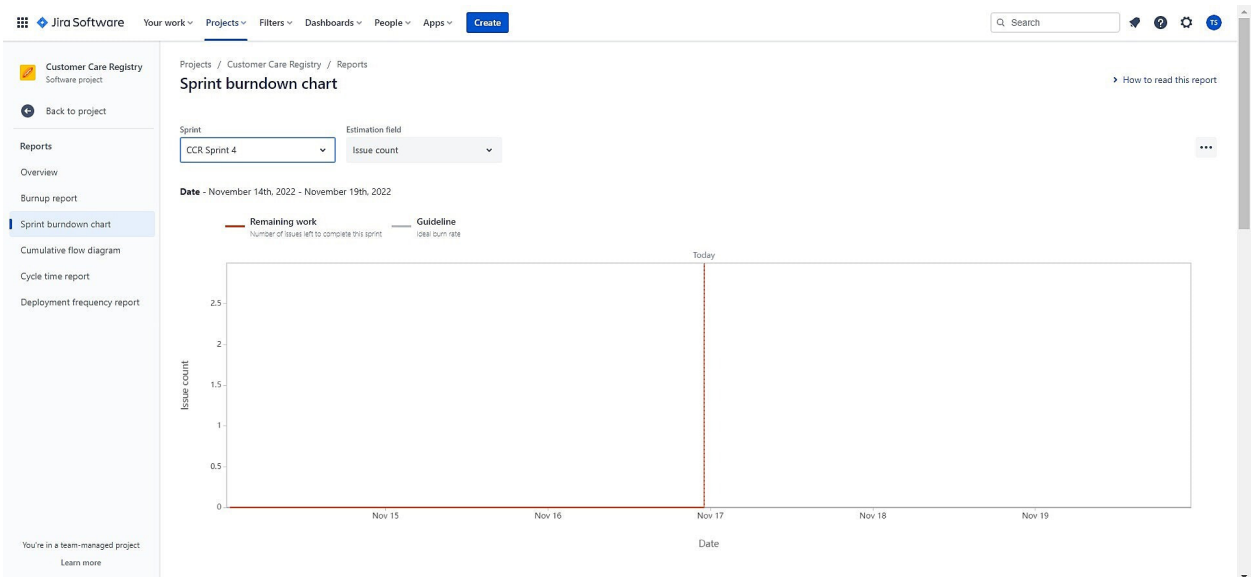
6.3 Reports from JIRA






Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Customer Care Registry
Software project
Back to project
Reports
Overview
Burnup report
Sprint burndown chart
Cumulative flow diagram
Cycle time report
Deployment frequency report
You're in a team-managed project
Learn more
Projects / Customer Care Registry / Reports
Sprint burndown chart
How to read this report
Sprint
CCR Sprint 2
Estimation field
Issue count
Date - October 31st, 2022 - November 5th, 2022
Remaining work
Number of issues left to complete this sprint
Guideline
Ideal burn rate
Planned end
Issue count
2.5
2
1.5
1
0.5
0
Nov 01
Nov 03
Nov 05
Nov 07
Nov 09
Nov 11
Nov 13
Nov 15
Date


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Customer Care Registry
Software project
Back to project
Reports
Overview
Burnup report
Sprint burndown chart
Cumulative flow diagram
Cycle time report
Deployment frequency report
You're in a team-managed project
Learn more
Projects / Customer Care Registry / Reports
Sprint burndown chart
How to read this report
Sprint
CCR Sprint 3
Estimation field
Issue count
Date - November 7th, 2022 - November 12th, 2022
Remaining work
Number of issues left to complete this sprint
Guideline
Ideal burn rate
Planned end
Issue count
2.5
2
1.5
1
0.5
0
Nov 09
Nov 11
Nov 13
Nov 15
Date



7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 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 Feature-Customer closing a ticket

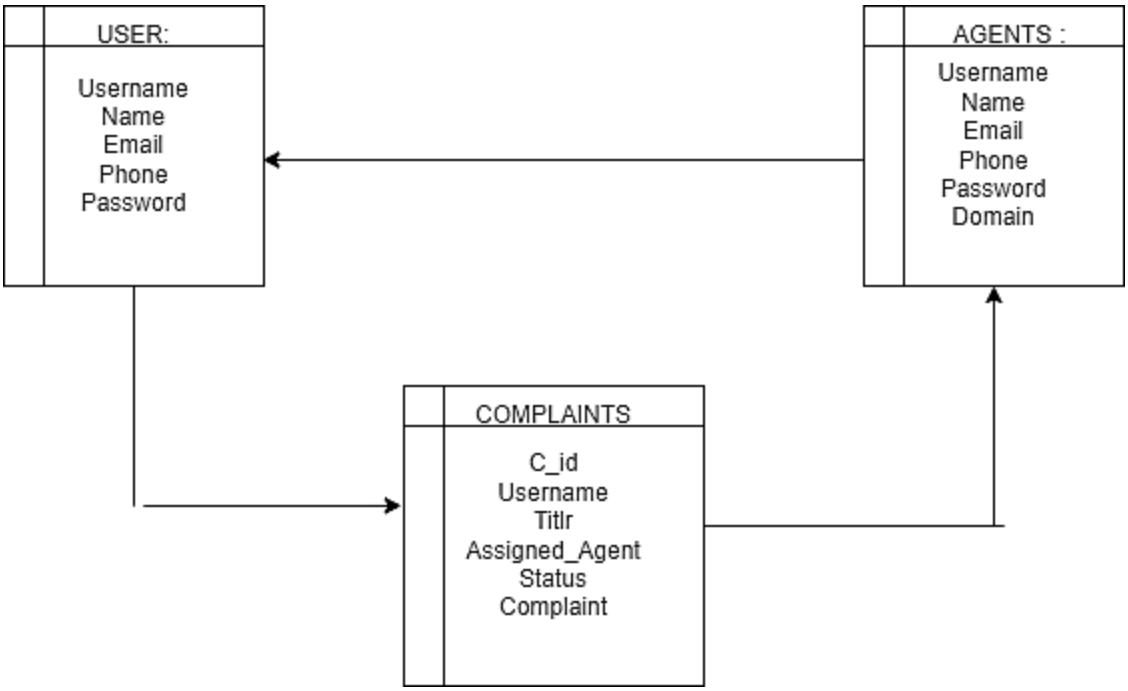
Code:

```
@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



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.

2.Defect Analysis

This report shows the number of resolved or closed bugs at each serverity 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

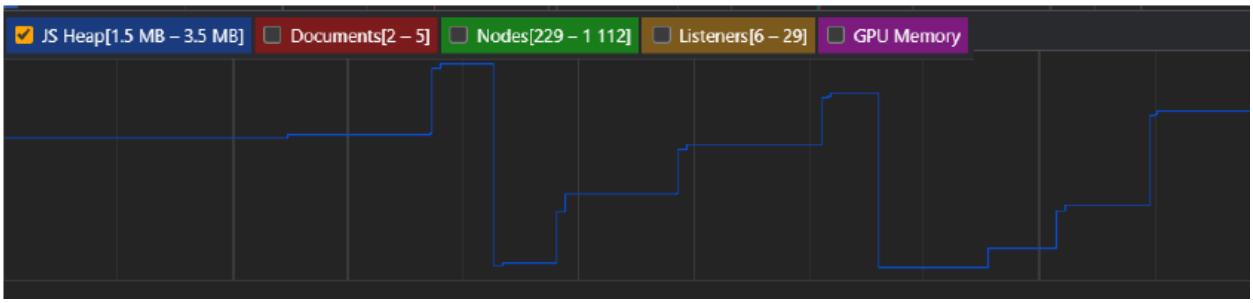
This report shows the number of test cases that are passed, failed, and untested.

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

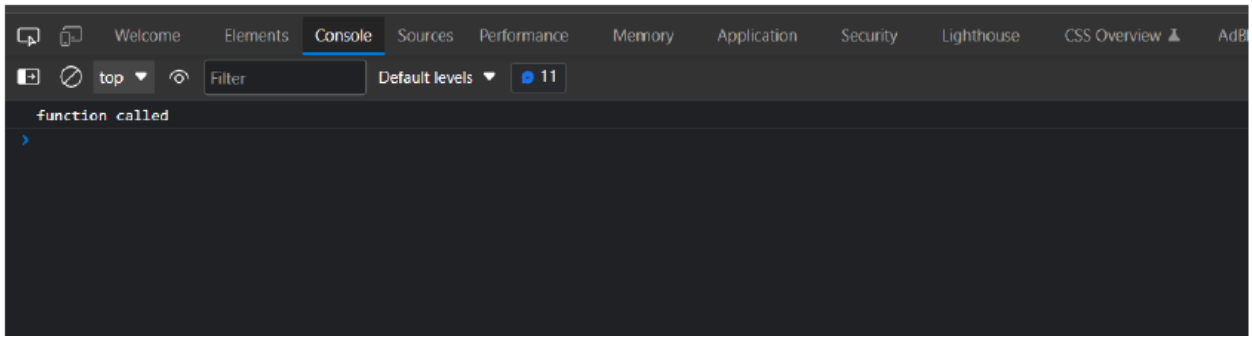
9.RESULTS

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.

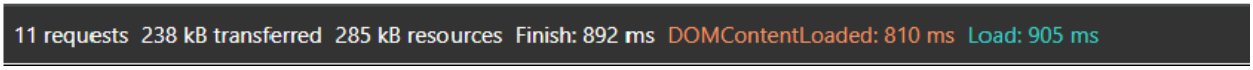


Errors:
Since all the backend functions are done using flask, any exceptions / errors rising are well-handled. Though they appear, user's interaction with the site is not affected in any way



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 & DISADVANTAGES

ADVANTAGES

- The advantage is you can learn 10 times faster than any other employee who works in any other field. As this role requires System Work, Lots of patience, General knowledge, Good listening skills, Problem Solving Attitude and much more thing. But the best part is you'll get the chance to meet and speak with a new customer (person) every day, You can contact 10 to 50 customers a day.
- It will make you a person, a professional person who can easily handle any situation.
- It'll quite easy for you to handle any situation where a customer or anyone is angry or disappointed also you'll learn some new things, will get new contacts..

DISADVANTAGES

- Customer service representatives to work in irregular schedule.
- Many customer service representatives have significant responsibility within their organization to assist customers and ensure their satisfaction. For some, this may be stressful to try to balance the level of responsibility and the workload.
- Customer service representatives often deal with frequent changes in policies, procedures, products and services.

11. CONCLUSION

Customer care, involves the use of basic ethics and any company whowants to have success and grow, needs to remember, that in order to do so, it must begin withestablishing a code of ethics in regards to how each employee is to handle the dealing withcustomers. Customers are at the heart of the company and its growth or decline. Customer careinvolves, the treatment, care, loyalty, trust the employee should extend to the consumer, as wellin life. This concept can be applied to so much more than just customer care. People need to treatothers with respect and kindness, people should try to take others into consideration whenmaking any decision. If more people were to practice this policy, chances are the world would bea better, more understanding place for all to exist.

12. FUTURE SCOPE

- The shift from a primarily 'cost centre' to primarily 'growth centre' worldview.
- The job desc for a customer service director will focus more on leadership, innovation, and ability to drive company-wide improvement.
- Customer service will shift to become a strategic partner of marketing, sales, and product development. CS will help with direction, project prioritisation, and impact.
- A need for customer service leaders to take a highly strategic seat at the table. They'll need to argue for investment in talent, technology, and innovation.
- A shift in performance metrics. Forget # of resolved tickets. In the future, we'll measure performance based on # of customers saved from the precipice of churn.
- A career in customer service will not be a last resort. Top graduates will prioritise getting an education in strategic customer interaction.
- Focus on ticket deflection will reduce because brands will view each customer interaction as an opportunity to learn, build a relationship, and grow profits. They deserve a well-trained, human touch.

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

Source Code

bass.html

```
<!DOCTYPE html>

<head>

    <link rel="stylesheet" href="static/css/main.css"/>
    {% block head %}
    {% endblock %}

</head>

<body>
    {% block body %}

    {% endblock %}
    <script>
        var coll = document.getElementsByClassName("collapsible");
        var i;

        for (i = 0; i < coll.length; i++) {
            coll[i].addEventListener("click", function () {
                this.classList.toggle("active");
                var content = this.nextElementSibling;
                if (content.style.display === "block") {
```

```
                content.style.display = "none";
            } else {
                content.style.display = "block";
            }
        });
    }
</script>

</body>

</html>
```

Signup.html

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

{% block head %}

<title>
    Sign Up
</title>
{% endblock %}

{% block body %}

<div class="forpadding">

    <!-- for box of the signup form -->
    <div class="sign">
        <div>
            <p class="fortitle">
                Register Now!!
            </p>
            <hr>
            <form action="/signup" method="post">
                <div class="forform">
                    <div class="textinformleft">
                        Username
                    </div>
```

```
    <div class="textinformright">
      <input type="name" name="username">
    </div>
  </div>
  <div class="forform">
    <div class="textinformleft">
      Name
    </div>
    <div class="textinformright">
      <input type="name" name="name">
    </div>
  </div>
  <div class="forform">
    <div class="textinformleft">
      E - mail
    </div>
    <div class="textinformright">
      <input type="name" name="email">
    </div>
  </div>
  <div class="forform">
    <div class="textinformleft">
      Phone Number
    </div>
    <div class="textinformright">
      <input type="name" name="phn">
    </div>
  </div>
  <div class="forform">
    <div class="textinformleft">
      Password
    </div>
    <div class="textinformright">
      <input type="password" name="pass">
    </div>
  </div>
  <div class="forform">
```

```
                <div class="textinformleft">
                    Re - enter Password
                </div>
                <div class="textinformright">
                    <input type="password" name="repass">
                </div>
            </div>
            <br>
            <div>
                <button class="forbutton" type="submit"> Sign up >></button>
            </div>
        </form>
        <br>
        <div>
            {{msg}}
        </div>
        <br>
        <div>
            Already have an account? <a href="/login">Sign in</a>
        </div>
        <br>
 </div>

    </div>
</div>

{% endblock %}
```

login.html

```
{% extends 'base.html' %}
{% block head %}
<title>
    Login
</title>

{% endblock %}
```

```
{% block body %}
<div class="forpadding">

   <!-- for box of the signup form -->
   <div class="sign">
      <div>
         <p class="fortitle">
            Sign In
         </p>
         <hr>
         <form action="/login" method="post">
            <div class="forform">
               <div class="textinformleft">
                  Username
               </div>
               <div class="textinformright">
                  <input type="name" name="username">
               </div>
            </div>

            <div class="forform">
               <div class="textinformleft">
                  Password
               </div>
               <div class="textinformright">
                  <input type="password" name="pass">
               </div>
            </div>

            <br>
            <div>
               <button class="forbutton" type="submit"> Sign In >></button>
            </div>
         </form>
         <br>
```

```
            <div>
                New user? <a href="/signup">Sign up</a>
            </div>
            <br>
        </div>

    </div>
</div>

{% endblock %}
```

dashboard.html

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

{% block head %}

<title>
    Dashboard
</title>

{% endblock %}
{% block body %}



<!-- things

    div 1
welcome jetson,    sign out

    div 2
your complaints status

add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
```

```
{% endfor %}

<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
   {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
   <div class="fordashboardtopelements1">
      Welcome {{ name }},
   </div>
   <div class="fordashboardtopelements2">
      <a href="/login"><button class="forbutton">Sign out</button></a>
   </div>

</div>
<br>
<div class="outerofdashdetails">

   <div class="fordashboarddetails">
      <br>
      <!-- table of customers complaints -->
      <table class="fortable">
         <thead>
            <th>Complaint ID</th>
            <th class="pad">Complaint Detail</th>
            <th>Assigned Agent</th>
            <th>Status</th>
            <th>Solution</th>
         </thead>
         <tbody>
            {% for i in complaints %}
```

```
        <tr>
            <td>
                {{ i['C_ID'] }}
            </td>
            <td class="pad">
                {{ i['TITLE'] }}
            </td>
            <td>
                {{ i['ASSIGNED_AGENT'] }}
            </td>
            <td>
                {% if i['STATUS'] == 1 %}
                Completed
                {% elif i['STATUS'] == 0 %}
                Not completed
                {% else %}
                In progress
                {% endif %}
            </td>
            <td>
                {{ i['SOLUTION'] }}
            </td>
        </tr>
        {% endfor %}
    </tbody>

</table>

<br>
<center>

    <div class="fordashboarddetails">

        <button type="button" class="collapsible">Add new complaint    </button>
        <div class="content">
            <br>
            <form action="/addnew" method="post">
```

```
                    <div class="forform">
                        <div class="textinformleft">
                            Title
                        </div>
                        <div class="textinformright">
                            <input type="name" name="title">
                        </div>
                    </div>

                    <div class="forform">
                        <div class="textinformleft">
                            Complaint
                        </div>
                        <div class="textinformright">
                            <textarea name="des"
                                    style="border-radius: 1rem;width: 90%;height: 150%;background-color:
black;color: white;"></textarea>
                        </div>
                    </div>

                    <br>
                    <br>
                    <div>
                        <button class="forbutton" type="submit"> Submit </button>
                    </div>
                </form>
                <br>
            </div>


        </div>
    </center>
</div>

</div>

{% endblock %}
```

admin.html

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

{% block head %}

<title>
   Admin Dashboard
</title>
{% endblock %}



{% block body %}



<!-- things

   div 1
welcome jetson,   sign out

   div 2
your complaints status

add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}

<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
   {{ j }}
{% endfor %}
```

```
<br>
{% endfor %} -->

<div class="fordashboardtop">
   <div class="fordashboardtopelements1">
      Welcome Admin,
   </div>
   <div class="fordashboardtopelements2">
      <a href="/login"><button class="forbutton">Sign out</button></a>
   </div>

</div>
<br>
<div class="outerofdashdetails">


    <div class="fordashboarddetails">
       <br>
       <!-- table of customers complaints -->
       <table class="fortable">
          <thead>
          </thead>
          <tbody>
             <tr>
                <td class="pad">
                   <a href="/agents">Agent Details</a>
                </td>
                <td class="pad">
                   <a href="/tickets">Customer Ticket Details</a>
                </td>
             </tr>
          </tbody>


        </table>

       <br>
```

```
    </div>

</div>

{% endblock %}
```

agent.html

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

<title>
    Dashboard
</title>



{% endblock %}



{% block body %}


<!-- things

    div 1
welcome jetson,   sign out

    div 2
your complaints status

add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}
```

```
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
   {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
   <div class="fordashboardtopelements1">
      Welcome Admin,
   </div>
   <div class="fordashboardtopelements2">
      <a href="/login"><button class="forbutton">Sign out</button></a>
   </div>

</div>
<br>
<div class="outerofdashdetails">

   <div class="fordashboarddetails">
      <br>
      <!-- table of customers complaints -->
      <table class="fortable">
         <thead>
            <th class="pad">Name</th>
            <th>Username</th>
            <th>Email</th>
            <th>Phone</th>
            <th>Domain</th>
            <th>Status</th>
         </thead>
         <tbody>
            {% for i in agents %}
            <tr>
```

```
            <td class="pad">
                {{ i['NAME'] }}
            </td>
            <td class="pad">
                {{ i['USERNAME'] }}
            </td>
            <td>
                {{ i['EMAIL'] }}
            </td>
            <td>
                {{ i['PHN'] }}
            </td>
            <td>
                {{ i['DOMAIN'] }}
            </td>
            <td>
                {% if i['STATUS'] == 1 %}
                Assigned to job
                {% elif i['STATUS'] == 0 %}
                not Available
                {% else %}
                Available
                {% endif %}
            </td>
        </tr>
        {% endfor %}
    </tbody>

</table>

<br>
<center>

    <div class="fordashboarddetails">

        <button type="button" class="collapsible">Add new agent    </button>
        <div class="content">
```

```
<br>
<form action="/addnewagent" method="post">
    <div class="forform">
        <div class="textinformleft">
            Username
        </div>
        <div class="textinformright">
            <input type="name" name="username">
        </div>
    </div>
    <div class="forform">
        <div class="textinformleft">
            Name
        </div>
        <div class="textinformright">
            <input type="name" name="name">
        </div>
    </div>
    <div class="forform">
        <div class="textinformleft">
            Email
        </div>
        <div class="textinformright">
            <input type="name" name="email">
        </div>
    </div>
    <div class="forform">
        <div class="textinformleft">
            Phone
        </div>
        <div class="textinformright">
            <input type="name" name="phone">
        </div>
    </div>
    <div class="forform">
        <div class="textinformleft">
            Domain
```

```
                    </div>
                    <div class="textinformright">
                        <input type="name" name="domain">
                    </div>
                </div>
                <div class="forform">
                    <div class="textinformleft">
                        Password
                    </div>
                    <div class="textinformright">
                        <input type="password" name="password">
                    </div>
                </div>

                <br>
                <br>
                <div>
                    <button class="forbutton" type="submit"> Submit </button>
                </div>
            </form>
            <br>
        </div>


    </div>
  </center>
 </div>

</div>

{% endblock %}
```

tickets.html

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



{% block head %}
```

```
<title>
   Agent Dashboard
</title>



{% endblock %}



{% block body %}


<!-- things

   div 1
welcome jetson,   sign out

   div 2
your complaints status

add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}

<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
   {{ j }}
{% endfor %}
<br>
```

```
{% endfor %} -->

<div class="fordashboardtop">
    <div class="fordashboardtopelements1">
        Welcome Admin,
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button class="forbutton">Sign out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">

    <div class="fordashboarddetails">
        <br>
        <!-- table of customers complaints -->
        <table class="fortable">
            <thead>
                <th>Complaint ID</th>
                <th class="pad">Username</th>
                <th>Title</th>
                <th>Complaint</th>
                <th>Solution</th>
                <th>Status</th>
            </thead>
            <tbody>
                {% for i in complaints %}
                <tr>
                    <td>{{ i['C_ID'] }}</td>
                    <td class="pad">
                        {{ i['USERNAME'] }}
                    </td>
                    <td>
                        {{ i['TITLE'] }}
                    </td>
                    <td>
```

```
            {{ i['COMPLAINT'] }}
          </td>
          <td>
            {{ i['SOLUTION'] }}
          </td>
          <td>
            {% if i['STATUS'] == 1 %}
            Completed
            {% else %}
            Not Completed
            {% endif %}
          </td>
        </tr>
        {% endfor %}
    </tbody>

</table>

<br>
<center>

    <div class="fordashboarddetails">

        <button type="button" class="collapsible">Assign an agent ⚡</button>
        <div class="content">
            <br>
            <form action="/assignagent" method="post">
                <div class="forform">
                    <div class="textinformleft">
                        Complaint ID
                    </div>
                    <div class="textinformright">
                        <input type="name" name="ccid">
                    </div>
                </div>
                <div class="forform">
                    <div class="textinformleft">
```

```
                    <label for="agent">Choose an agent:</label>
                </div>
                <div class="textinformright">
                    <select name="agent" id="agent">
                        {% for i in freeagents %}
                        <option value={{ i['USERNAME'] }}>{{ i['USERNAME'] }}</option>
                        {% endfor %}
                    </select>
                </div>
            </div>

            <br>
            <br>
            <div>
                <button class="forbutton" type="submit"> Submit </button>
            </div>
        </form>
        <br>
    </div>


    </div>
  </center>
</div>

</div>

{% endblock %}
```

agentsdash.html

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

{% block head %}

<title>
  Agent Dashboard
</title>
```

```
{% endblock %}



{% block body %}


<!-- things

   div 1
welcome jetson,   sign out

   div 2
your complaints status

add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}

<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
   {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
   <div class="fordashboardtopelements1">
      Welcome {{ name }},
```

```
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button class="forbutton">Sign out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">

    <div class="fordashboarddetails">
        <br>
        <!-- table of customers complaints -->
        <table class="fortable">
            <thead>
                <th>Complaint ID</th>
                <th class="pad">Username</th>
                <th>Title</th>
                <th>Complaint</th>
                <th>Solution</th>
                <th>Status</th>
            </thead>
            <tbody>
                {% for i in complaints %}
                <tr>
                    <td class="pad">
                        {{ i['C_ID'] }}
                    </td>
                    <td class="pad">
                        {{ i['USERNAME'] }}
                    </td>
                    <td>
                        {{ i['TITLE'] }}
                    </td>
                    <td>
                        {{ i['COMPLAINT'] }}
                    </td>
                    <td>
```

```
            {{ i['SOLUTION'] }}
          </td>
          <td>
            {% if i['STATUS'] == 1 %}
            Completed
            {% else %}
            Not Completed
            {% endif %}
          </td>
        </tr>
        {% endfor %}
    </tbody>

</table>

<br>
<center>

    <div class="fordashboarddetails">

        <button type="button" class="collapsible">Solve an Issue ⚡</button>
        <div class="content">
            <br>
            <form action="/updatecomplaint" method="post">
                <div class="forform">
                    <div class="textinformleft">
                        Complaint ID
                    </div>
                    <div class="textinformright">
                        <input type="name" name="cid">
                    </div>
                </div>
                <div class="forform">
                    <div class="textinformleft">
                        Solution
                    </div>
                    <div class="textinformright">
```

```
                        <input type="text" name="solution">
                    </div>
                </div>

                <br>
                <br>
                <div>
                    <button class="forbutton" type="submit"> Submit </button>
                </div>
            </form>
            <br>
        </div>


    </div>
  </center>
</div>

</div>

{% endblock %}
```

main.css

```
.sign {
  border-radius: 1rem;
  background-color: lightblue;
  text-align: center;
  padding: 1%;
}

.fortitle {
  font-size: medium;
  font-weight: 500;
  font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
  padding: 5px;
}

.forp {
```

```
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

.textinformleft {
    text-align: left;
    padding-left: 5%;
    width: 50%;
    border-radius: 1rem;
    font-size: medium;
    font-weight: 500;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

.textinformright {
    width: 50%;
    padding-right: 10px;
    border-radius: 1rem;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}
.textinformright2 {
    width: 100%;
    text-align: center;
    padding-right: 10px;
    border-radius: 1rem;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

input {
    border-radius: 1rem;
    color: white;
    background-color: black;
    padding-left: 15px;
}

input:focus {
    border-color: yellow;
}
```

```
.forform {
    display: flex;
    padding: 15px;
    border-radius: 1rem;

}

.forpadding {
    padding-top: 5%;
    padding-left: 25%;
    padding-right: 25%;
}

body {
    background-image: url('/static/images/background.jpg');
    background-repeat: no-repeat;
    background-size: 1540px 715px;
    /* background-color: black; */
    /* background-image: url('F:\Own\IBM project\Sample2\static\css\bg.png'); */
}

.forbutton {
    background-color: black;
    color: white;
    border-radius: 1rem;
    padding: 7px;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

button:hover {
    background-color: white;
    color: black;
    box-shadow: white;
    cursor: pointer;
}
```

```
/* for dashboard */

.fordashboardtop {
    border-radius: 1rem;
    display: flex;
    background-color: lightblue;
}

.fordashboardtopelements1 {
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
    width: 90%;
    font-size: large;
    padding: 2%;
}

.fordashboardtopelements2 {
    width: 10%;
    padding-top: 1%;
    padding-bottom: 1%;
}

.fordashboarddetails {
    padding: 2%;
    border-radius: 1rem;
    background-color: rgb(102, 150, 184);
}

.outerofdashdetails {
    /* padding-top: 2%; */
    padding-left: 5%;
    padding-right: 5%;
}

.fortable {
    width: 100%;
    padding: 1%;
```

```
    text-align: center;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

.pad {
    padding: 7px;
}

.forbutton2 {
    background-color: black;
    color: white;
    border-radius: 1rem;
    padding: 7px;
    width: 200%;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

.foraddbutton{
    /* width: 30%; */
    background-color: black;
    color: white;
    border-radius: 1rem;
    padding: 7px;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
}

.collapsible {
    background-color: black;
    color: white;
    border-radius: 1rem;
    padding: 7px;
    width: 30%;
    font-family: 'Franklin Gothic Medium', 'Arial Narrow', Arial, sans-serif;
    /* background-color: #777; */
    /* color: white; */
    cursor: pointer;
    /* padding: 18px; */
```

```
    /* width: 100%; */
    /* border: none;
    text-align: left; */
    /* outline: none;
    font-size: 15px; */
}

.collapsible:hover {
    background-color: white;
}

.content {
    /* padding: 0 18px; */
    display: none;
    border-radius: 1rem;
    background-color: rgb(89, 131, 160);
    width: 50%;
    /* overflow: hidden; */
    /* background-color: #f1f1f1; */
}
```

app.py

```
from flask import Flask, render_template, request, redirect, session, url_for
import ibm_db
import re
app = Flask(__name__)
# for connection
# conn= ""

app.secret_key = 'a'
print("Trying to connect...")
conn=ibm_db.connect("DATABASE=bludb;HOSTNAME=ea286ace-86c7-4d5b-8580-
3fbfa46b1c66.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31505;SECURITY=SS
L;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=rrv63214;PWD=tZj4yo9dMQNoZ9d3
","","")
print("connected..")
```

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phn = request.form['phn']
        password = request.form['pass']
        repass = request.form['repass']
        print("inside checking")
        print(name)
        if len(username) == 0 or len(name) == 0 or len(email) == 0 or len(phn) == 0 or
len(password) == 0 or len(repass) == 0:
            msg = "Form is not filled completely!!"
            print(msg)
            return render_template('signup.html', msg=msg)
        elif password != repass:
            msg = "Password is not matched"
            print(msg)
            return render_template('signup.html', msg=msg)
        elif not re.match(r'[a-z]+', username):
            msg = 'Username can contain only small letters and numbers'
            print(msg)
            return render_template('signup.html', msg=msg)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email'
            print(msg)
            return render_template('signup.html', msg=msg)
        elif not re.match(r'[A-Za-z]+', name):
            msg = "Enter valid name"
            print(msg)
            return render_template('signup.html', msg=msg)
        elif not re.match(r'[0-9]+', phn):
            msg = "Enter valid phone number"
```

```
            print(msg)
            return render_template('signup.html', msg=msg)

        sql = "select * from users where username = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Acccount already exists'
        else:
            userid = username
            insert_sql = "insert into users values(?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn, insert_sql)
            ibm_db.bind_param(prep_stmt, 1, username)
            ibm_db.bind_param(prep_stmt, 2, name)
            ibm_db.bind_param(prep_stmt, 3, email)
            ibm_db.bind_param(prep_stmt, 4, phn)
            ibm_db.bind_param(prep_stmt, 5, password)
            ibm_db.execute(prep_stmt)
            print("successs")
            msg = "succesfully signed up"
        return render_template('dashboard.html', msg=msg, name=name)
    else:
        return render_template('signup.html')


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

@app.route('/')
def base():
    return redirect(url_for('login'))

@app.route('/login', methods=["GET", "POST"])
```

```
def login():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        userid = username
        password = request.form['pass']
        if userid == 'admin' and password == 'admin':
            print("its admin")
            return render_template('admin.html')
        else:
            sql = "select * from agents where username = ? and password = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, password)
            ibm_db.execute(stmt)
            account = ibm_db.fetch_assoc(stmt)
            print(account)
            if account:
                session['Loggedin'] = True
                session['id'] = account['USERNAME']
                userid = account['USERNAME']
                session['username'] = account['USERNAME']
                msg = 'logged in successfully'

                # for getting complaints details
                sql = "select * from complaints where assigned_agent = ?"
                complaints = []
                stmt = ibm_db.prepare(conn, sql)
                ibm_db.bind_param(stmt, 1, username)
                ibm_db.execute(stmt)
                dictionary = ibm_db.fetch_assoc(stmt)
                while dictionary != False:
                    complaints.append(dictionary)
                    dictionary = ibm_db.fetch_assoc(stmt)
                print(complaints)
                return render_template('agentdash.html', name=account['USERNAME'],
```

```
complaints=complaints)

        sql = "select * from users where username = ? and password = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            session['Loggedin'] = True
            session['id'] = account['USERNAME']
            userid = account['USERNAME']
            session['username'] = account['USERNAME']
            msg = 'logged in successfully'

            # for getting complaints details
            sql = "select * from complaints where username = ?"
            complaints = []
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.execute(stmt)
            dictionary = ibm_db.fetch_assoc(stmt)
            while dictionary != False:
                # print "The ID is : ",  dictionary["EMPNO"]
                # print "The Name is : ", dictionary[1]
                complaints.append(dictionary)
                dictionary = ibm_db.fetch_assoc(stmt)

            print(complaints)
            return render_template('dashboard.html', name=account['USERNAME'],
complaints=complaints)
        else:
            msg = 'Incorrect user credentials'
            return render_template('dashboard.html', msg=msg)
    else:
        return render_template('login.html')
```

```
@app.route('/addnew', methods=["GET", "POST"])
def add():
    if request.method == 'POST':
        title = request.form['title']
        des = request.form['des']
        try:
            sql = "insert into complaints(username,title,complaint) values(?,?,?)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.bind_param(stmt, 2, title)
            ibm_db.bind_param(stmt, 3, des)
            ibm_db.execute(stmt)
        except:
            print(userid)
            print(title)
            print(des)
            print("cant insert")
        sql = "select * from complaints where username = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            # print "The ID is : ",  dictionary["EMPNO"]
            # print "The Name is : ", dictionary[1]
            complaints.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)
        print(complaints)
        return render_template('dashboard.html', name=userid, complaints=complaints)


@app.route('/agents')
def agents():
    sql = "select * from agents"
```

```
    agents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        agents.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    return render_template('agents.html', agents=agents)


@app.route('/addnewagent', methods=["GET", "POST"])
def addagent():
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        domain = request.form['domain']
        password = request.form['password']
        try:
            sql = "insert into agents values(?,?,?,?,?,?,2)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, name)
            ibm_db.bind_param(stmt, 3, email)
            ibm_db.bind_param(stmt, 4, phone)
            ibm_db.bind_param(stmt, 5, password)
            ibm_db.bind_param(stmt, 6, domain)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from agents"
        agents = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
```

```
            agents.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)

        return render_template('agents.html', agents=agents)


@app.route('/updatecomplaint', methods=["GET", "POST"])
def updatecomplaint():
    if request.method == 'POST':
        cid = request.form['cid']
        solution = request.form['solution']
        try:
            sql = "update complaints set solution =?,status=1 where c_id = ? and assigned_agent=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, solution)
            ibm_db.bind_param(stmt, 2, cid)
            ibm_db.bind_param(stmt, 3, userid)
            ibm_db.execute(stmt)
            sql = "update agents set status =3 where username=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from complaints where assigned_agent = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            complaints.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)
        # print(complaints)
        return render_template('agentdash.html', name=userid, complaints=complaints)
```

```
@app.route('/tickets')
def tickets():
    sql = "select * from complaints"
    complaints = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        complaints.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)

    sql = "select username from agents where status <> 1"
    freeagents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        freeagents.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    print(freeagents)
    return render_template('tickets.html', complaints=complaints, freeagents=freeagents)


@app.route('/assignagent', methods=['GET', 'POST'])
def assignagent():
    if request.method == "POST":
        ccid = request.form['ccid']
        agent = request.form['agent']
        print(ccid)
        print(agent)
        try:
            sql = "update complaints set assigned_agent =? where c_id = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, agent)
            ibm_db.bind_param(stmt, 2, ccid)
            ibm_db.execute(stmt)
            sql = "update agents set status =1 where username = ?"
```

```
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant update")
        return redirect(url_for('tickets'))

if __name__ == '__main__':
  app.debug = True
  app.run(host='0.0.0.0', port=5000)
```

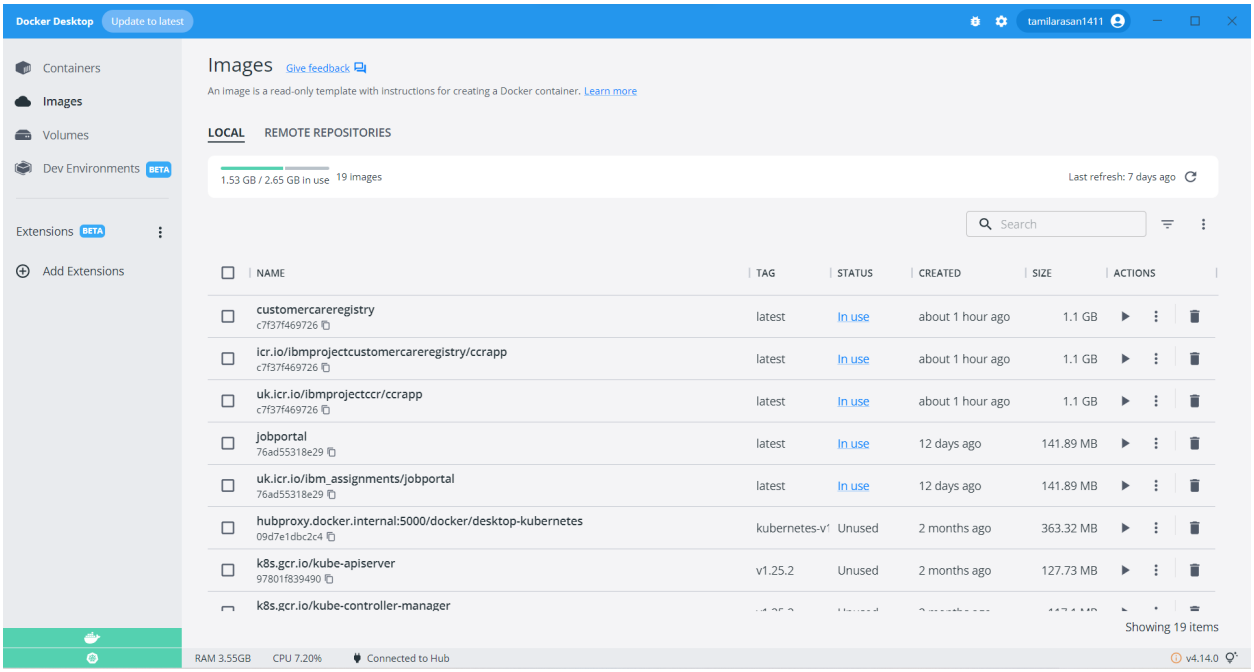
Dockerfile

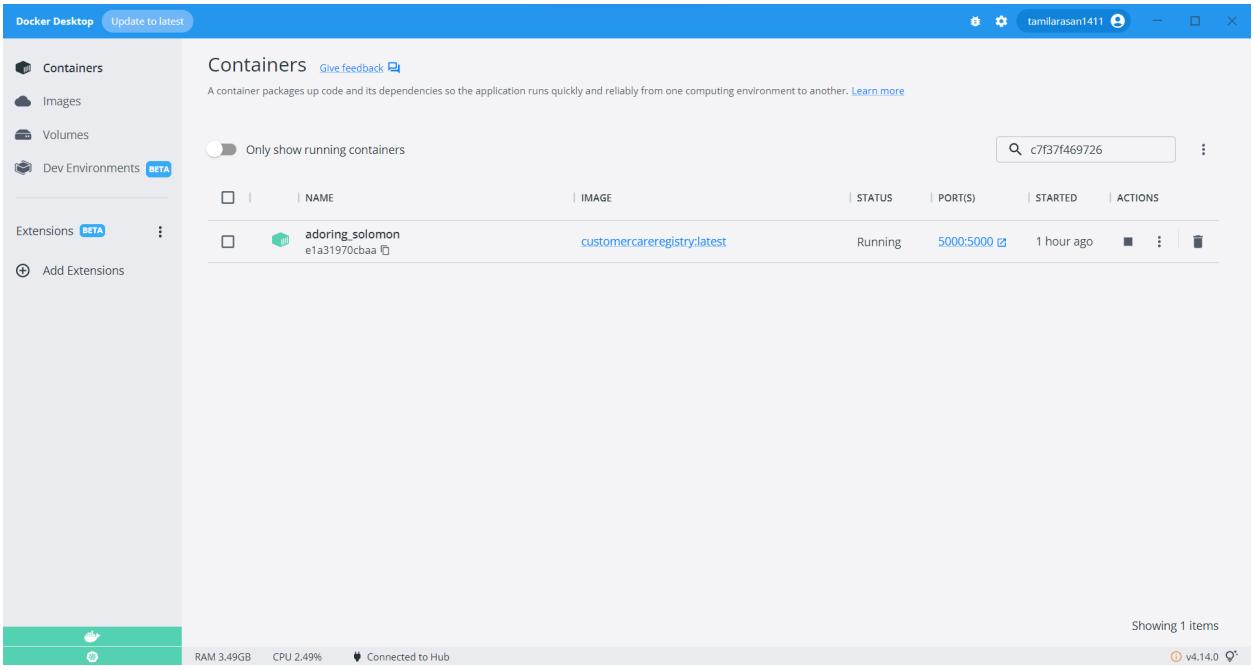
```
FROM python:3.10.6
WORKDIR /IBM Project
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY . .
EXPOSE 5000
CMD ["python","./app.py"]
```

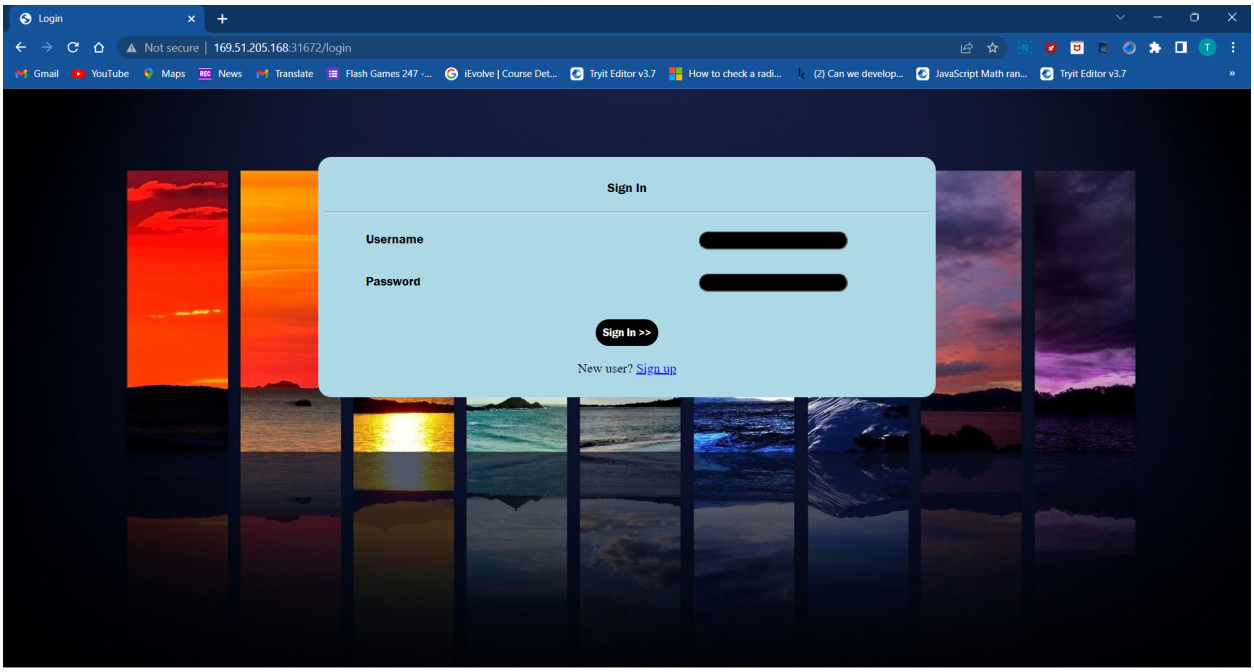
requirements

```
flask
ibm_db
```

Output


Docker Desktop
Update to latest
tamilarasan1411
Containers
Images
Volumes
Dev Environments
BETA
Extensions
BETA
Add Extensions
Images
Give feedback
An image is a read-only template with instructions for creating a Docker container. Learn more
LOCAL
REMOTE REPOSITORIES
1.53 GB / 2.65 GB in use
19 images
Last refresh: 7 days ago
Search
NAME
TAG
STATUS
CREATED
SIZE
ACTIONS
customercareregistry
c7f37f469726
latest
In use
about 1 hour ago
1.1 GB
icr.io/ibmprojectcustomercareregistry/ccrapp
c7f37f469726
latest
In use
about 1 hour ago
1.1 GB
uk.icr.io/ibmprojectccr/ccrapp
c7f37f469726
latest
In use
about 1 hour ago
1.1 GB
jobportal
76ad55318e29
latest
In use
12 days ago
141.89 MB
uk.icr.io/ibm_assignments/jobportal
76ad55318e29
latest
In use
12 days ago
141.89 MB
hubproxy.docker.internal:5000/docker/desktop-kubernetes
09d7e1dbc2c4
kubernetes-v1
Unused
2 months ago
363.32 MB
k8s.gcr.io/kube-apiserver
97801f839490
v1.25.2
Unused
2 months ago
127.73 MB
k8s.gcr.io/kube-controller-manager
Showing 19 items
RAM 3.55GB
CPU 7.20%
Connected to Hub
v4.14.0


Docker Desktop
Update to latest
tamilarasan1411
Containers
Images
Volumes
Dev Environments
BETA
Extensions
BETA
Add Extensions
Containers
Give feedback
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more
Only show running containers
c7f37f469726
NAME
IMAGE
STATUS
PORT(S)
STARTED
ACTIONS
adoring_solomon
e1a31970cbaa
customercareregistry:latest
Running
5000:5000
1 hour ago
Showing 1 items
RAM 3.49GB
CPU 2.49%
Connected to Hub
v4.14.0



GitHub & Project Demo Link

Website Link - http://169.51.205.168:31672/login
GitHub Link - https://github.com/IBM-EPBL/IBM-Project-18658-1659688073
Project Demo Link- https://drive.google.com/file/d/1B52ze4h7106NaB0wM-5kz1WUzTfR1Imn/view?usp=drivesdk