Project Report

Batch ID	B2-2M4E
Team ID	PNT2022TMID43316
Project Name	Customer Care Registry

Team members:

- NETHIRAN N T
- NITIN SUBRAMANIAN S
- REVANTH S
- SARATHI KANNAN K

CONTENT

1. INTRODUCTION

Introduction to the project:

- 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.
- Customer Service also known as Client Service is the provision of service to customers. Its significance varies by product, industry and domain. In many cases customer services is more important if the information relates to a service as opposed to a Customer.
- An effective complaints management system is integral to providing quality customer service. It helps to measure customer satisfaction and is a useful source of information and feedback for improving services. Often customers are the first to identify when things are not working properly.

Purpose of the Project:

- An online comprehensive Customer Care Solution is to manage customer in teraction
and complaints with the Service Providers over phone or through and email.
- The system shouldhave capability to integrate with any Service Provider fro m any domain or industry like Banking, telecom Insurance, etc.
- Customer Service also known asclient service is the provision ofservice to customers its significance varies by product, Industry and domain with any service Provider from any domain or industry like Banking, Telecom Insurance etc.
- Provide a common platform to the customers to clarify their queries.
- Customers and Agents can chat with one another for better understanding.
- Customers and Agents can also submit their feedbacks to the Admin, for the betterment of the platform.

2. LITERATURE SURVEY

Literature Survey on Customer Care Registry

S.NO	PAPER	AUTHOR	YEAR	METHOD AND ALGORITHM	ACCURACY
1	Theory and practice of customer-relat ed improvements	Daniel Gyllenhamma r, et al	2022	The study ensures the Customer satisfactions and reliable on customer improvements, it uses PRISMA Model for customer relations	92%
2	Improving Customer Service in Healthcare	Muhammad Anshari, et al	2021	The study involves the focus on individual relationship and limited view of the customer & his community preferences, habits, etc It uses CRM 2.O Model	89%
3	A machine learning approach to analyze customer satisfaction from airline tweets	Sachin Kumar and Mikhail Zymbler	2019	Features were extracted from the tweets. SVM and several ANN architectures were considered to develop classifcation model that maps the tweet into positive and negative category	92.3%
4	Cybercrime Case As Impact Development Of Communicatio n Technology That Troubling Society	M Chairul Basrun Umanailo, et al	2020	This analysis will be the process of selecting, comparing, combining and sorting various information and data analysis	90%

5	Customer Experience modelling from customer experience to service design	Jorge Teixeira, Lia Patrı´cio, et al	2019	It uses CEM method and models to synthesize and communicate knowledge between members of a multidisciplinary service design.	90%

Existing System:

- The existing system is a semi-automated at where the information is stored in the form of excel sheets in disk drives.
- Volunteers, Group members, etc. is through mailing feature only. The information storage and maintenance is more critical in this system. Tracking the member's activities and progress of the work is a tedious job here. This system cannot provide the information sharing by 24x7 days.

Reference:

- Simon Haykin, “Bird classification using CNN: a comprehensive foundation,” PrenticeHall PTR, 1994.
- Paul Viola, Michael Jones, “Classification and Grading of Image Using Texture BasedBlock-Wise Local Binary Patterns” CVPR (1) 1 (2001), 511–518, 2001.
- Gary Bradski and Adrian Kaehler. “Texture Classification from RandomFeatures”, 2008.
- Schmid Huber J, “Adapted approach for Species Classification: An OverviewNeural Networks” 61: 85-117, 2015.
- Haibing Wu and Xiaodong Gu, “Detection and Classification of images usingDetection Line” 71,1–10, 2015.

Problem Statement Definition:

- Previous research or relevant research is very important in a scientific research or article. Previous research or relevant research serves to strengthen the theory and influence of relationships or influences between variables.

- The existing solutions need more computing power, time and more information to provide accurate results.

- Reviews are not from the authentic individuals.

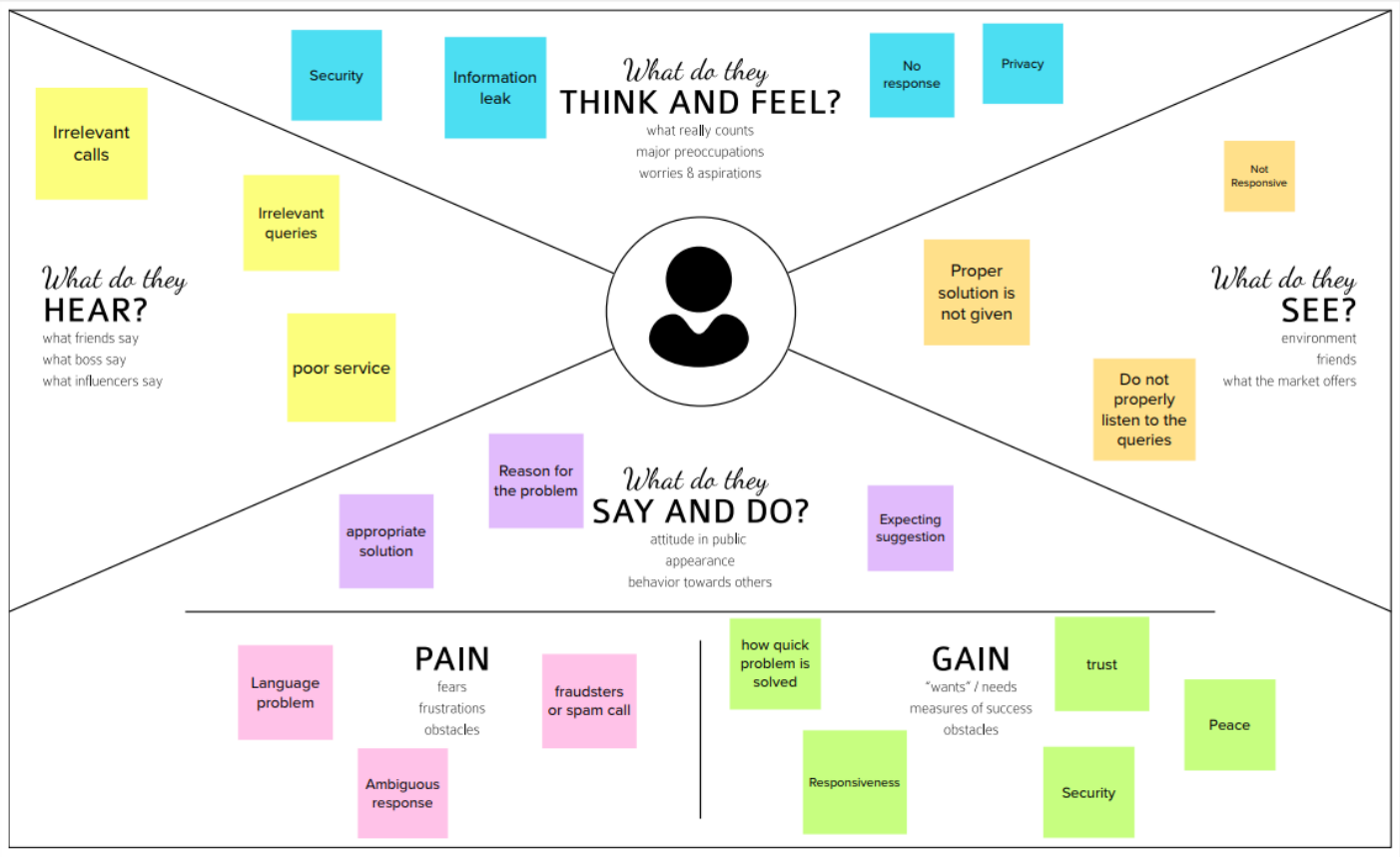
- After buying common platform available to us, the customer, to have our doubts cleared.

- If it is existing, we are not getting fast by the time the reply comes issues has been cleared or of not worth of being cleaned to the customer.

3. IDEATION & PROPOSED SOLUTION

Empathy Map Canvas:

- An empathy map is a collaborative tool teams can use to gain a deeper insight into their customers. Much like a user persona, an empathy map can represent a group of users, such as a customer segment. The empathy map was originally created by Dave Gray and has gained much popularity within the agile community.



Ideation & Brainstorming

Ideation and the practice of brainstorming, a particular method for coming up with fresh ideas, are frequently closely related. The main distinction between ideation and brainstorming is that whereas brainstorming is nearly often done in groups, ideation is typically seen as being more of a solitary endeavor. A group of people are frequently gathered for a brainstorming session to generate either fresh, general ideas or solutions to specific problems or circumstances.

On instance, a large firm that has discovered it is the target of a significant lawsuit might wish to consult with its top executives to come up with ideas for how to publicly respond to the case being filed.

In a brainstorming session, participants are encouraged to freely share any ideas that may come to mind. According to the theory, by coming up with a lot of ideas, the brainstorming group is more likely to find a workable solution to the problem they are trying to solve.

Template

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.

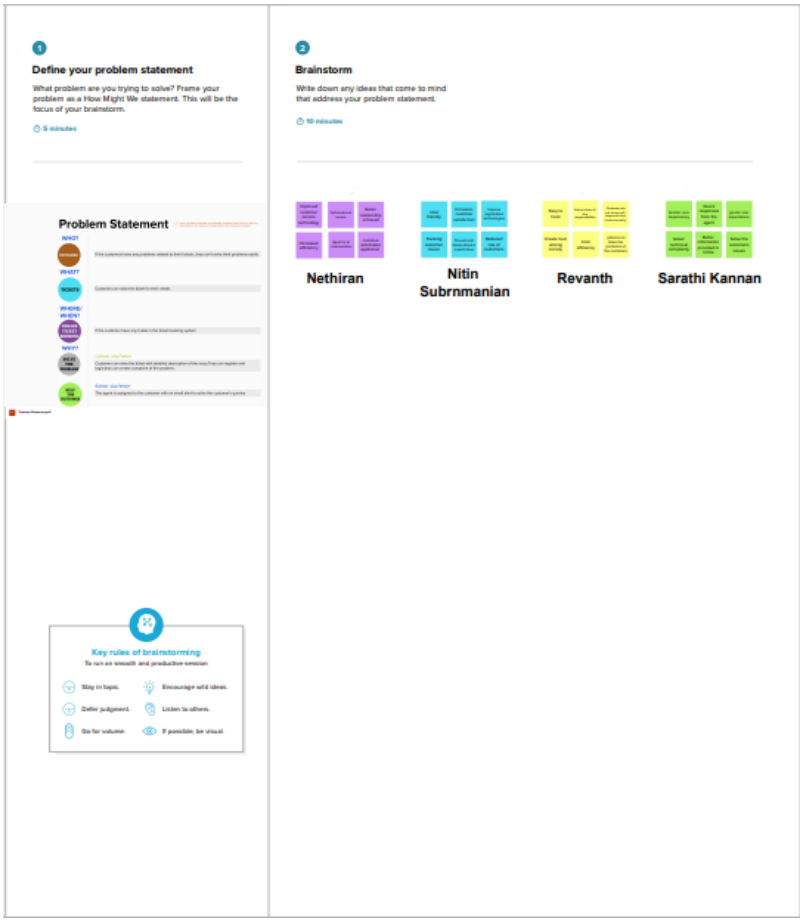
10 minutes

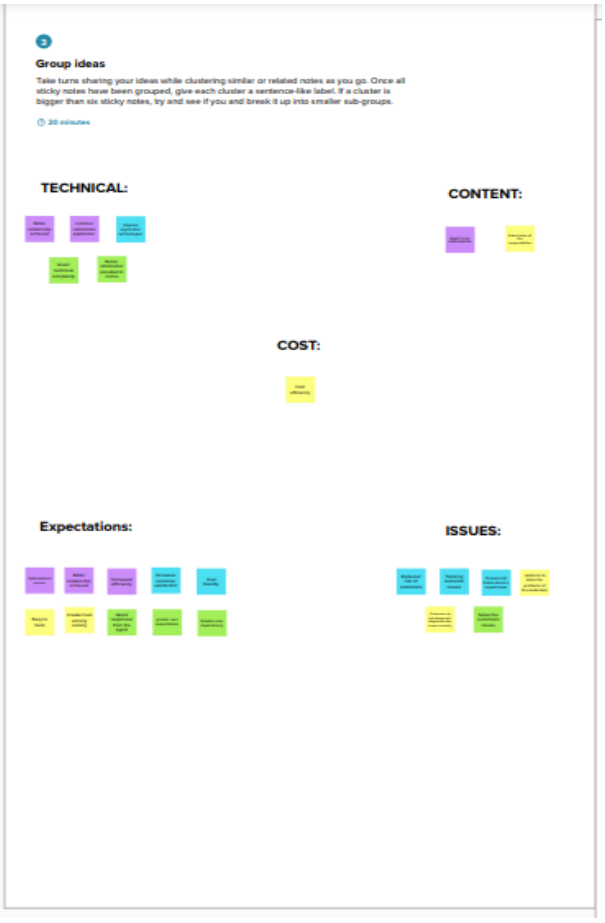
Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

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

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 Statement
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
Nethiran
Nitin Subrnmanian
Revanth
Sarathi Kannan


3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, 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
TECHNICAL:
CONTENT:
COST:
Expectations:
ISSUES:

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

After you collaborate

You can export the mural as an image or pdf to share with members of your company who might find it helpful.

Quick add-ons

Share the mural
Share a view link to the mural with stakeholders to keep them in the loop about the outcomes of the session.

Export the mural
Export a copy of the mural as a PNG or PDF to attach to emails, include in slides, or save in your drive.

Keep moving forward

Strategy blueprint
Define the components of a new idea or strategy.
Open the template →

Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

Strengths, weaknesses, opportunities & threats
Identify strengths, weaknesses, opportunities, and threats (SWOT) to develop a plan.
Open the template →

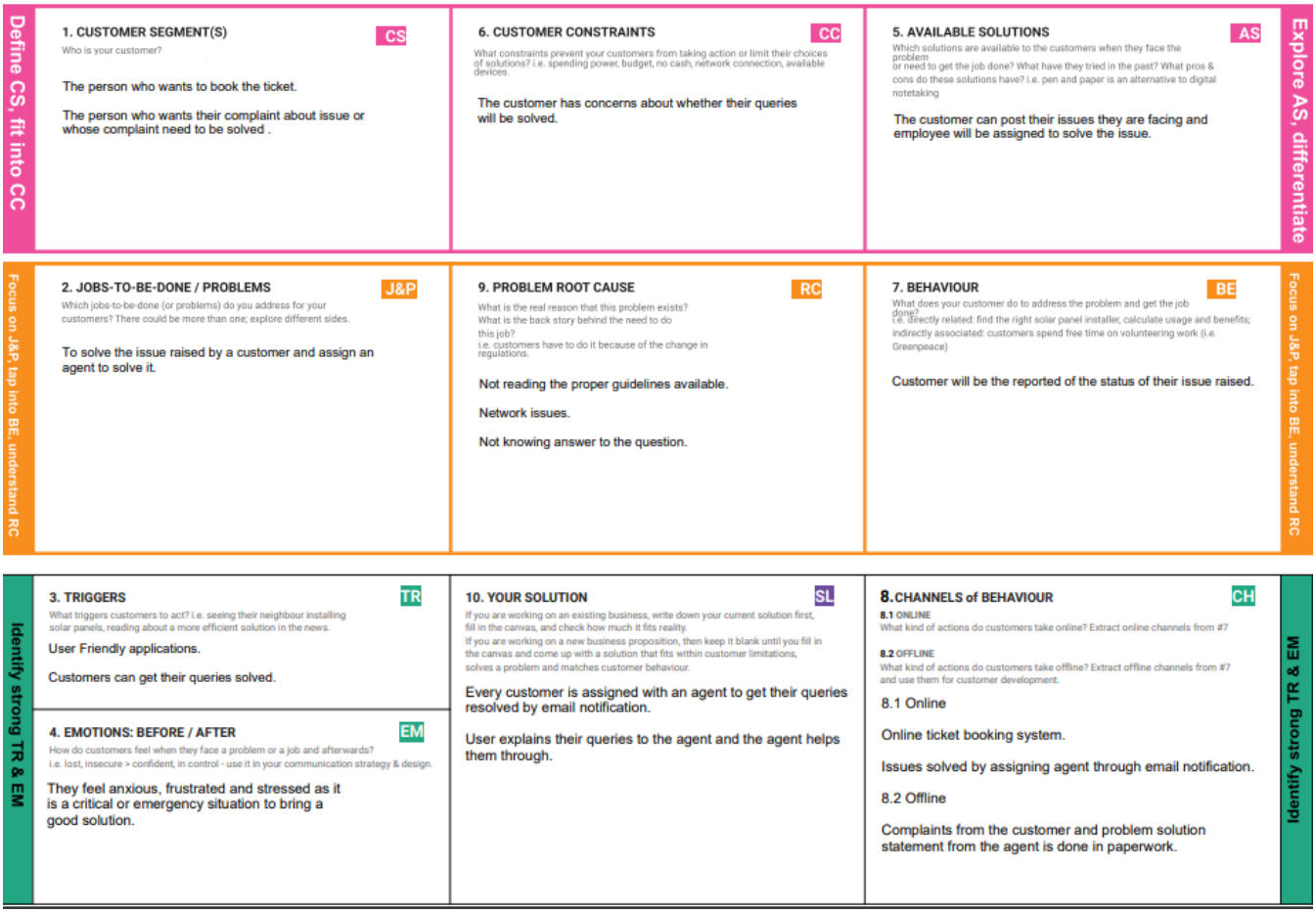
Share template feedback

Proposed Solution:

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To provide the services that are required by Customers and solve their problems Using cloud app development.
2.	Idea / Solution description	The help desk has to cover a wide range of information technology products and services. The customer can raise the ticket with a detailed description of the issues. User can register for an account. After the login they can create a complaint with a detailed description of the problem they are facing. Each user will be assigned with an agent. They can view the Status of the complaint.
3.	Novelty / Uniqueness	Chat bot for feedback and friendly service, automatic email and message service for customer notifications.
4.	Social Impact / Customer Satisfaction	Customer can get their problems solved easily with the help of agents solutions, trustable and reachable.
5.	Business Model (Revenue Model)	Key partners and third parties applications, agents and customers. Activities held as customer service, System maintenance.

6.	Scalability of the Solution	The real goal of scaling customer service is providing an environment that will allow your customer service specialist to be as efficient as possible. An environment where they will be able to spend less time on grunt work and more time on Resolving actual customer issues .

Problem Solution fit:



4. REQUIREMENT ANALYSIS

What is Requirement Analysis?

It is the process of determining user expectations for a system under consideration.

Functional requirement:

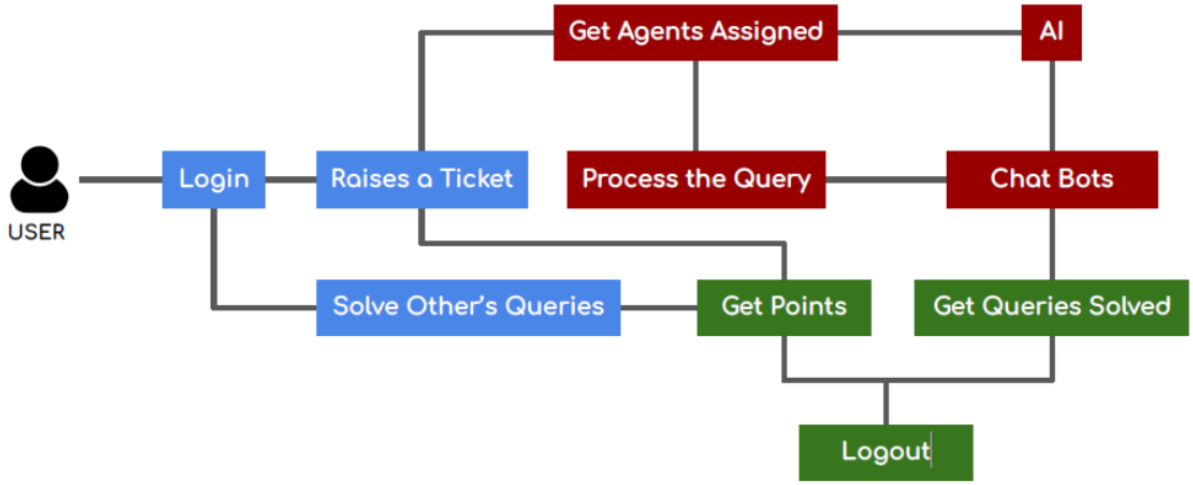
FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through Google
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User login	Login using Email and password
FR-4	Admin login	Login using Email and password
FR-5	Query Form	Description of the issue Contact details
FR-6	Feedback	Customer Feedback

Non-functional requirements :

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	User Friendly, providing solution to the posed problem, easy accessibility
NFR-2	**Security**	Login authentication
NFR-3	**Reliability**	Tracking the status of the query posted and providing quality solution
NFR-4	**Performance**	Easy and quick access, responsive
NFR-5	**Availability**	Available at all time
NFR-6	**Scalability**	Scalability depends on the number of customers

5. PROJECT DESIGN

Data Flow Diagrams:



Solution Architecture:

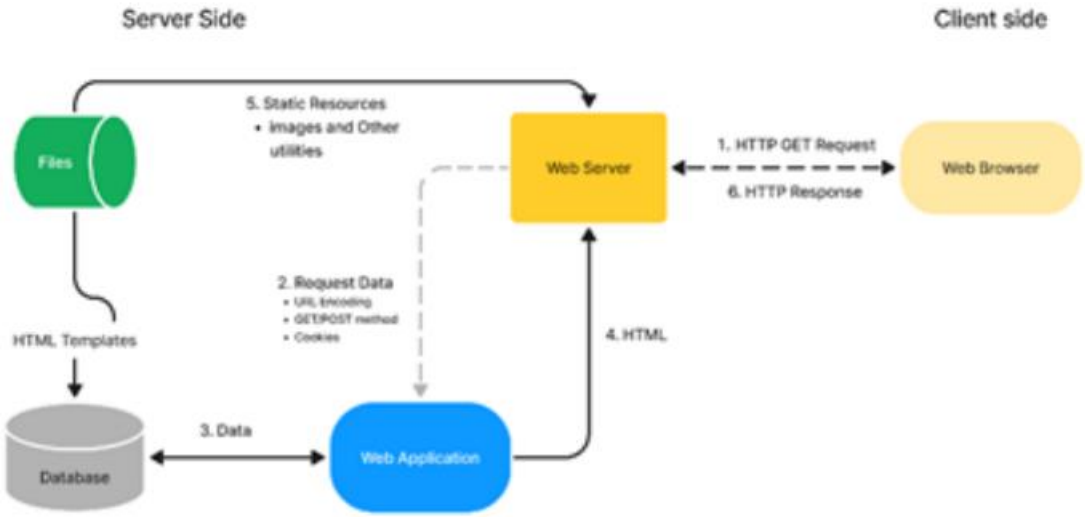


Table-1 : Components & Technologies:

S.No	Component	Description	Technology
1.	User Interface	How user interacts with application e.g. Web UI, Mobile App, Chatbot etc.	HTML, CSS, JavaScript / Angular Js / React Js etc.
2.	Application Logic-1	Logic for a process in the application	Java / Python
3.	Application Logic-2	Logic for a process in the application	IBM Watson STT service
4.	Chat bot Assistance	Conversational interface	IBM Watson Assistant
5.	Database	Data Type, Configurations etc.	MySQL, NoSQL, etc.
6.	Cloud Database	Database Service on Cloud	IBM DB2, IBM Cloudant etc.
7.	File Storage	File storage requirements	IBM Block Storage or Other Storage Service or Local Filesystem
8.	External API-1	Purpose of External API used in the application	IBM Weather API, etc.
9.	Machine Learning Model	Purpose of Machine Learning Model	Object Recognition Model, etc.
10.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration: Cloud Server Configuration :	Local, Cloud Foundry, Kubernetes, etc.

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	List the open-source frameworks used	Python Flask, Watson Assistant
2.	Scalable Architecture	Justify the scalability of architecture (3 – tier, Micro-services)	Python Flask
3.	Availability	Justify the availability of application (e.g. use of load balancers, distributed servers etc.)	Python Flask
4.	Performance	Design consideration for the performance of the application (number of requests per sec, use of Cache, use of CDN's) etc.	Python Flask

User Stories:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer	Registration	USN-1	As a user, I can register for the application by entering my email, password, and confirming my password.	I can access my account / dashboard	High	Sprint-1
	Login	USN-2	As a user, I can log the application by entering email & password	I can log into my account / dashboard	High	Sprint-1
	Raising Tickets	USN-3	As a user, I can raise a ticket regarding my query	I can raise a ticket	High	Sprint-1
	Rewarding points	USN-4	As a user, I can get points or be able to get a reward for resolved queries	I can resolve other's queries	Medium	Sprint-4
	Logout	USN-5	As a user, I can logout from my account	I can logout from my account	High	Sprint-2
Admin	Login	ASN-1	As a admin, I can login to my account / dashboard	I can access my dashboard / account	High	Sprint-2
	Track the status	ASN-2	As a admin, I can track the status of the user raised queries	I can track the status of the ticket	Medium	Sprint-2
	Assign the agents	ASN-3	As a admin, I can assign the agents respective to the customers	I can assign agents	High	Sprint-2
	Ban suspicious accounts	ASN-4	As a admin, I can ban the suspicious accounts or users	I can ban accounts	Medium	Sprint-2

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Agent	Resolve Queries	AGSN-1	As a Agent, I can resolve the customer queries	I can resolve user queries	High	Sprint-3
	Connecting with related problems	AGSN-2	As a Agent, I can connect to the related problems	I can connect related problems or queries	Medium	Sprint-3
	Chat with the customers	AGSN-3	As a Agent, I can chat with the customers or users	I can chat with the one raised ticket	Medium	Sprint-3
	Flag the tickets	AGSN-4	As a Agent, I can flag the status of the raised ticket	I can flag the ticket	Low	Sprint-4

6. PROJECT PLANNING & SCHEDULING

‘Project Planning and Scheduling’, though separate, are two sides of the same coin in project management. Fundamentally, ’Project planning’ is all about choosing and designing effective policies and methodologies to attain project objectives. While ‘Project scheduling’ is a procedure of assigning tasks to get them completed by allocating appropriate resources within an estimated budget and time-frame.

Project Planning:

The project planning phase refers to:

- Developing a project to make it ready for investment
- Determines the jobs/tasks required to attain project objectives

Sprint Planning & Estimation :

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Registration	USN-1	The user will be able to login to the website and can use the services available on the web page.	2	High	Nethiran Nitin Subramanian
Sprint-2	Admin Panel	USN-2	As an admin, with the help of correct credentials dashboard can be accessed to create agents and assign agents to solve a query.	2	High	Nethiran Revanth
Sprint-3	Agent Panel	USN-3	The agent using their credentials, login and access dashboard to check the posted queries.	2	High	Revanth Sarathi Kannan
Sprint-4	Chat bot	USN-4	The chat bot is provided so that user can directly access the services offered by the web portal.	2	Medium	Nitin Subramanian Sarathi Kannan
Sprint-5	Final Delivery	USN-5	Container of applications using docker kubernetes and deployment the application. Create the documentation and final submit the application	2	High	Nitin Subramanian Nethiran Revanth Sarathi Kannan

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022		29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022		09 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022		10 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022		19 Nov 2022

7. CODING & SOLUTIONING

Features:

- Users can log in with credentials or if they do not have an account and they signup with the all the details for creating an account.
- The overall UI of the web page is simple and user-friendly, Users can easily understand the site.
- The home page gives the overview of why is this website and with quick links at the header they can quickly navigate to another page.
- The users who have issues to complaint can give the details required and the agent who is assigned for the user will resolve the issue that is present.
- Admin can only create agents and can see the list of tickets by the user and the progress of the tickets.
- On the Admin dashboard, the admin can able to view all the agents that are present and all the tickets that are available.

Database Schema :

A database schema defines how data is organized within a relational database, this is inclusive of logical constraints such as, table names, fields, data types, and the relationships between these entities. Schemas commonly use visual representations to communicate the architecture of the database, becoming the foundation for an organization's data management discipline. This process of database schema design is also known as data modeling.

These data models serve a variety of roles, such as database users, database administrators, and programmers. For example, it can help database administrators manage normalization processes to avoid data duplication.
Alternatively, it can enable analysts to navigate these data structures to conduct reporting or other valuable business analyses. These diagrams act as valuable documentation within the database management system (DBMS), ensuring alignment across various stakeholders.

Types of database schemas:

Database schema types

- Although the term "schema" is used in a wide variety of contexts, it most frequently refers to three distinct types of schema: conceptual database schemas, logical database schemas, and physical database schemas.

- Conceptual schemas provide a broad overview of the system's contents, organisational structure, and business rules. Typically, conceptual models are developed as a part of obtaining the initial project requirements.

- Comparatively speaking, logical database schemas are less abstract than conceptual schemas. Table names, field names, entity relationships, and integrity constraints—i.e., any regulations governing the database—are all well defined schema objects with information. They normally don't have any technical requirements, though.

- The technical details that the logical database schema lacks are provided by physical database schemas.

8.TESTING

- **TEST CASES**

A test case has components that describe input, action and an expected response, in order to determine if a feature of an application is working correctly. A test case is a set of instructions on “HOW” to validate a particular test objective/target, which when followed will tell us if the expected behavior of the system is satisfied or not.

Characteristics of a good test case:

- Accurate: Exacts the purpose.
- Economical: No unnecessary steps or words.
- Traceable: Capable of being traced to requirements.
- Repeatable: Can be used to perform the test over and over.
- Reusable: Can be reused if necessary.

• S.no	Scenario	Input	Expected Output	Actual output
• 1	Admin Login Form	User name and password	Login	• Login success.
• 2	Employee Login Form	User name and password	Login	• Login success.
• 3	User Registration form	User basic details	Registed successfully	User basic details are stored in the database.
• 4	User Login Form	User name and password	Login	• Login success.

USER ACCEPTANCE TESTING

This is a type of testing done by users, customers, or other authorized entities to determine application/software needs and business processes. Acceptance testing is the most important phase of testing as this decides whether the client approves the application/software or not. It may involve functionality, usability, performance, and UI of the application. It is also known as user acceptance testing (UAT), operational acceptance testing (OAT), and end-user testing.

• 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	**10**	**3**	**1**	**2**	**17**
Duplicate	**1**	**0**	**3**	**0**	**4**
External	**2**	**3**	**0**	**1**	**6**
Fixed	**11**	**2**	**4**	**20**	**40**
Not Reproduced	**0**	**0**	**1**	**0**	**1**
Skipped	**0**	**0**	**1**	**1**	**2**
Won't Fix	**0**	**5**	**2**	**1**	**8**
Totals	**24**	**13**	**12**	**25**	**78**

TEST CASE ANALYSIS

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

Test cases :

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	10	0	0	10
Client Application	50	0	0	50
Security	1	0	0	1
Outsource Shipping	3	0	0	3
Exception Reporting	8	0	0	8
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9.RESULTS

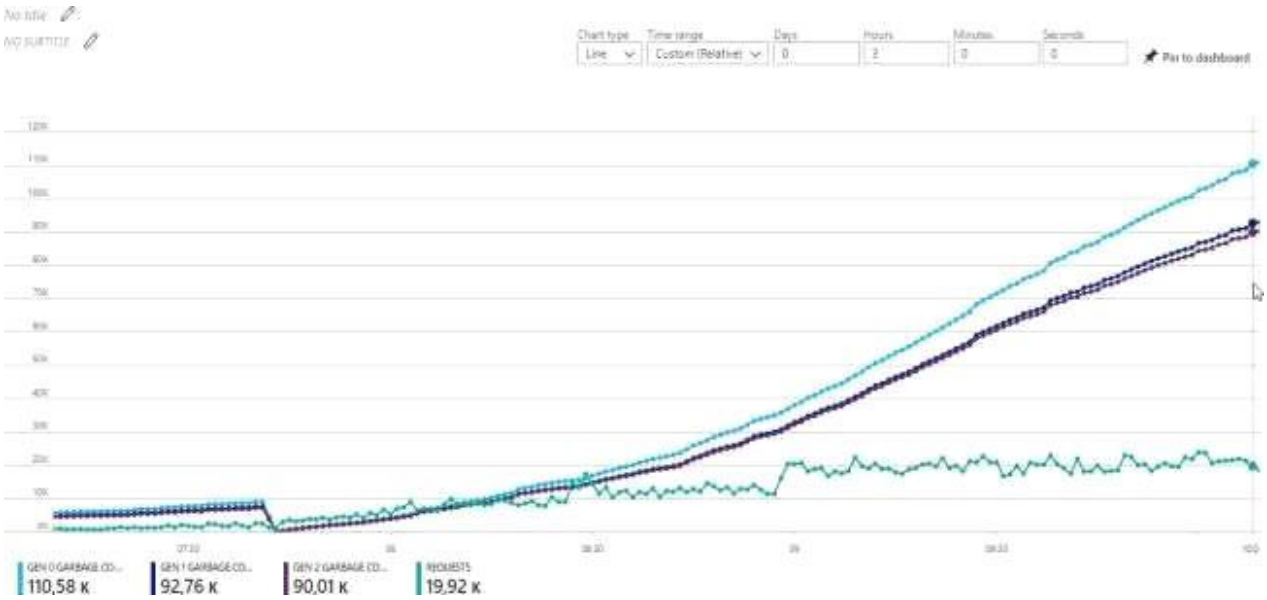


Figure 9.1.1 Performance Metrics

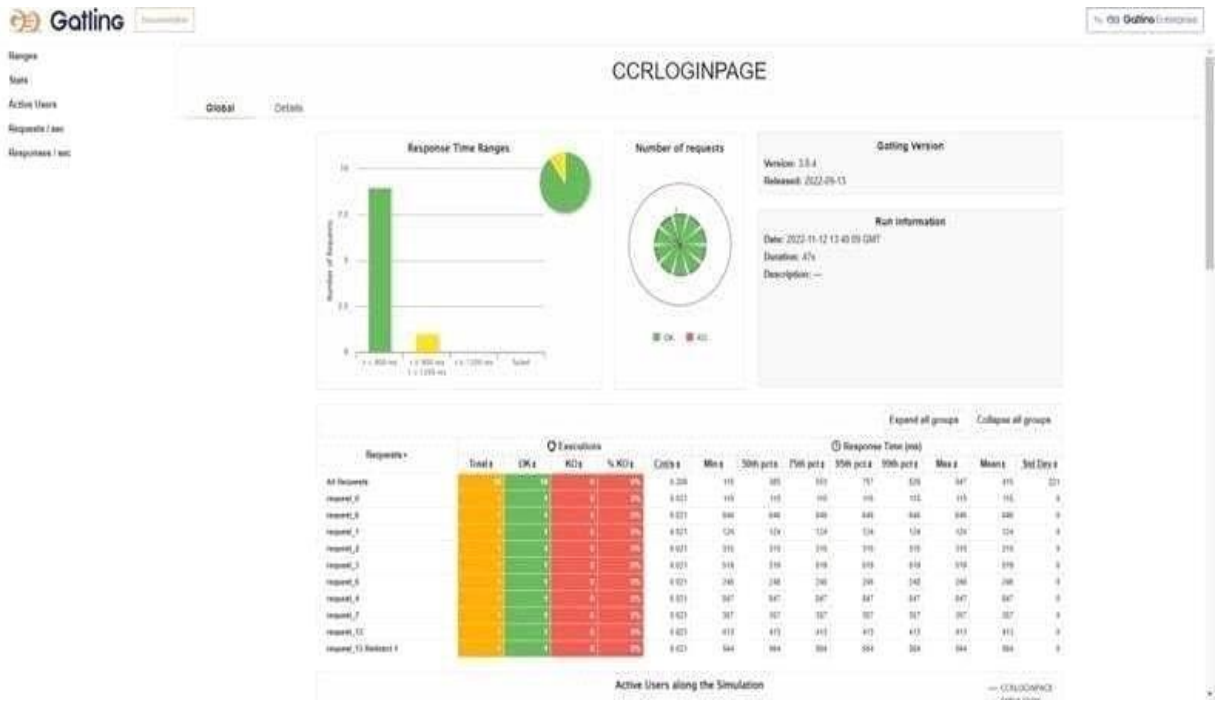
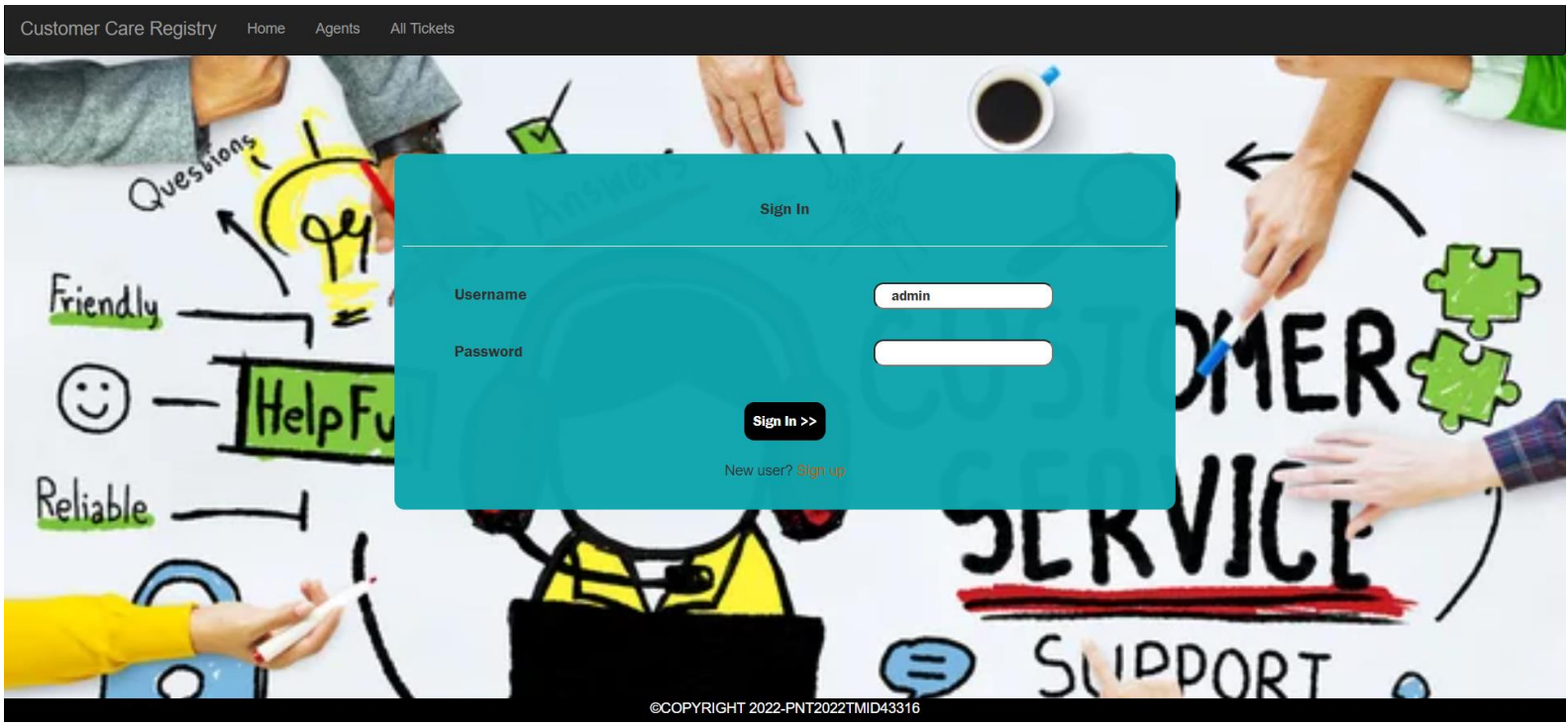
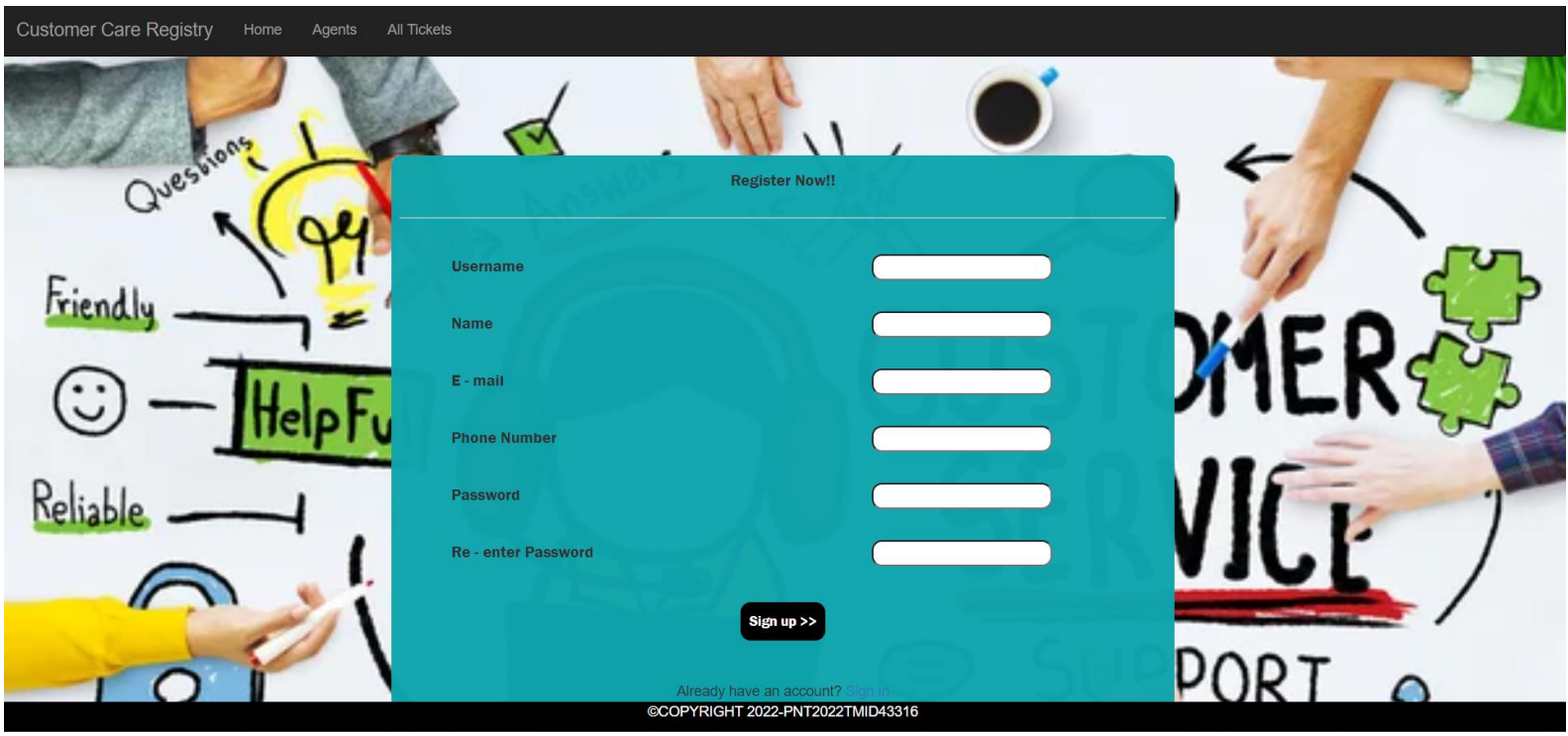
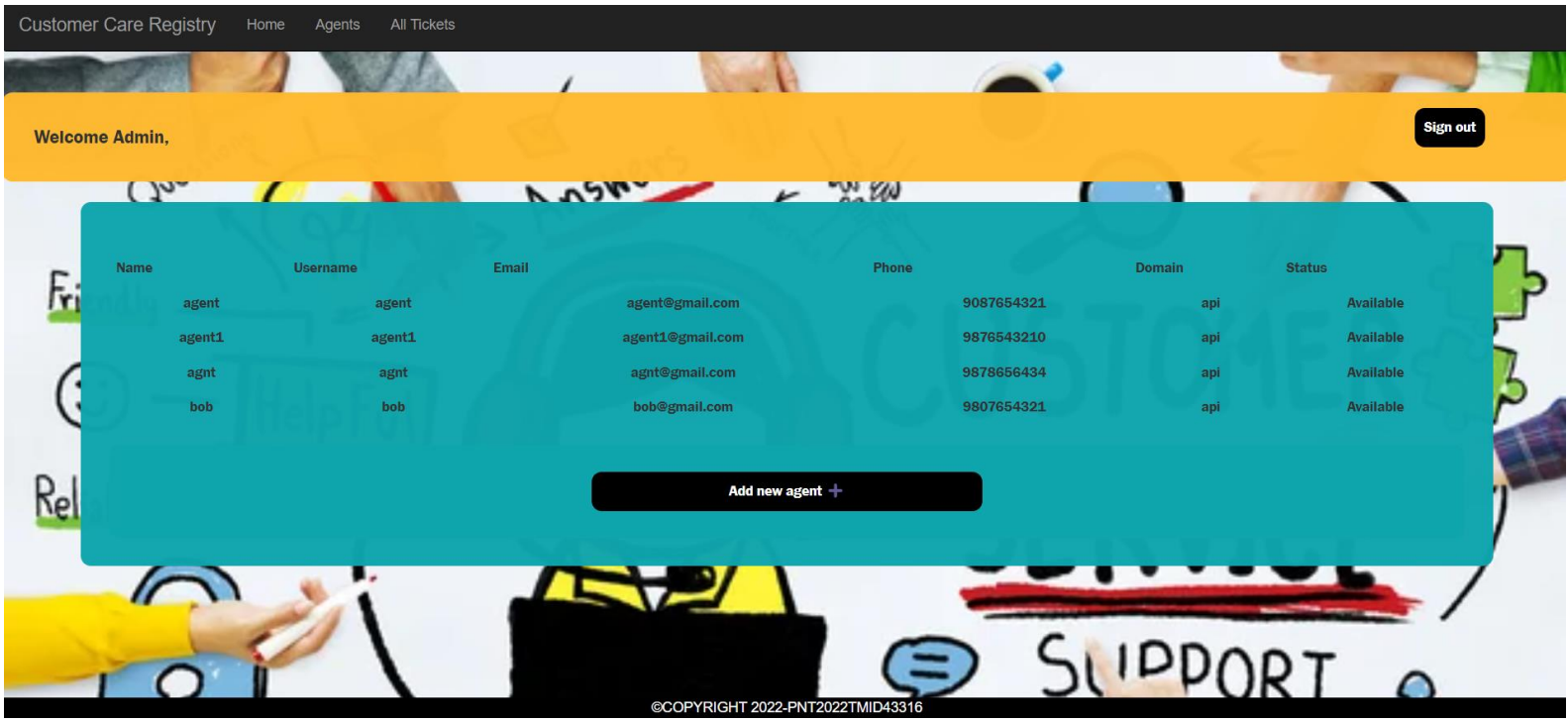


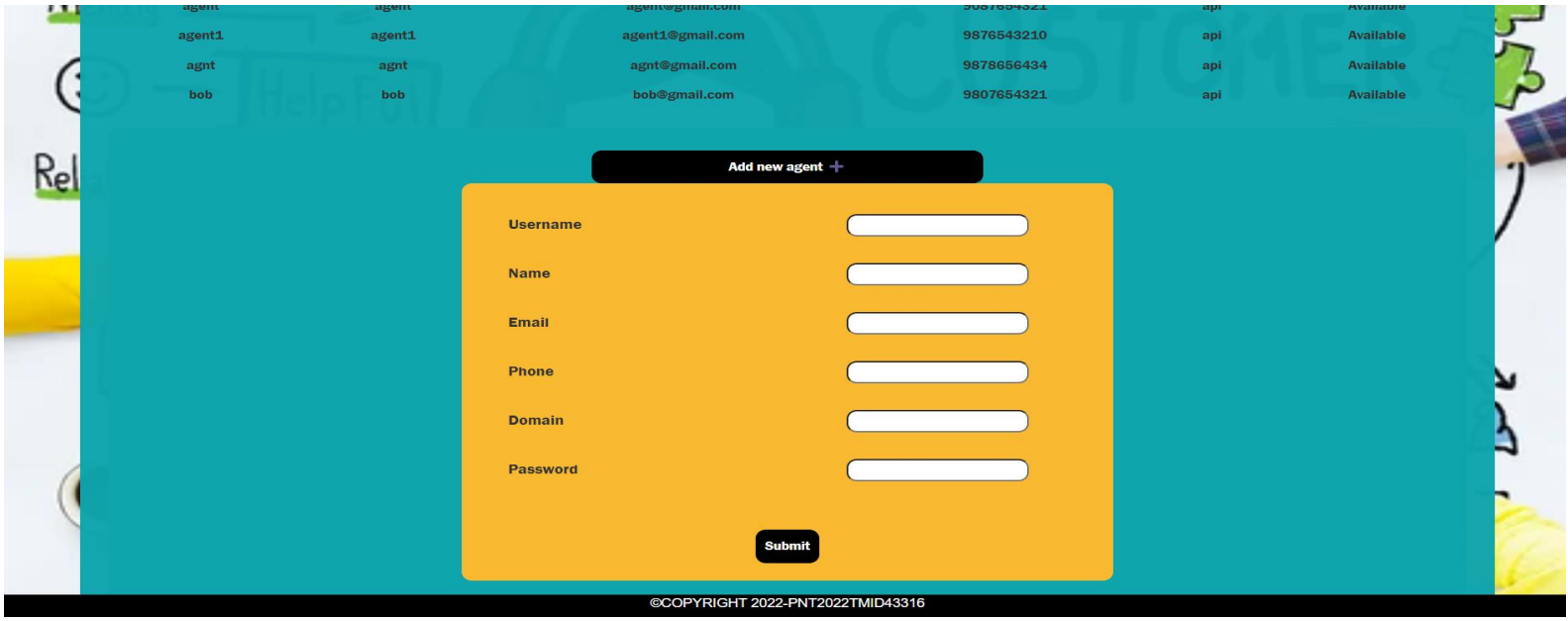
Figure 9.1.2 Performance Metric

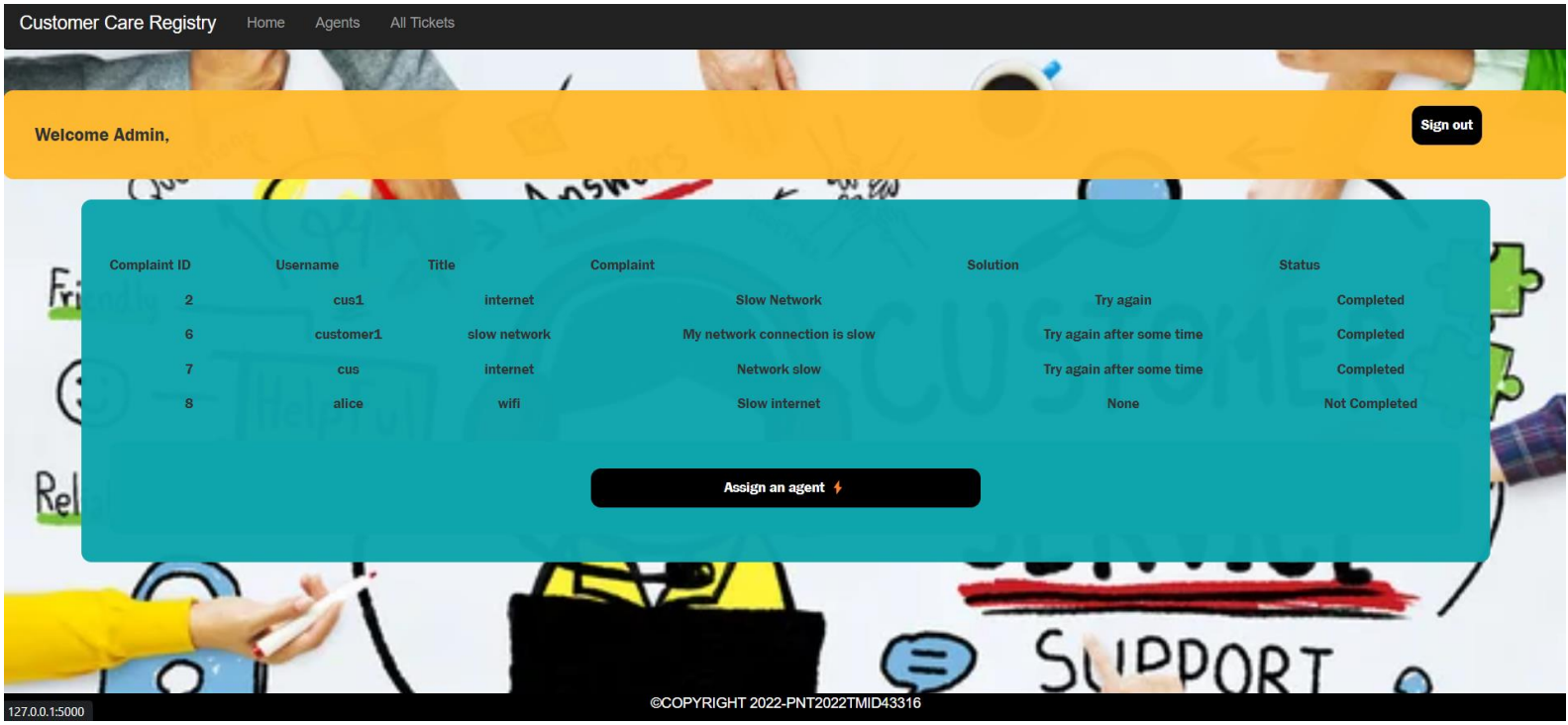
Output:

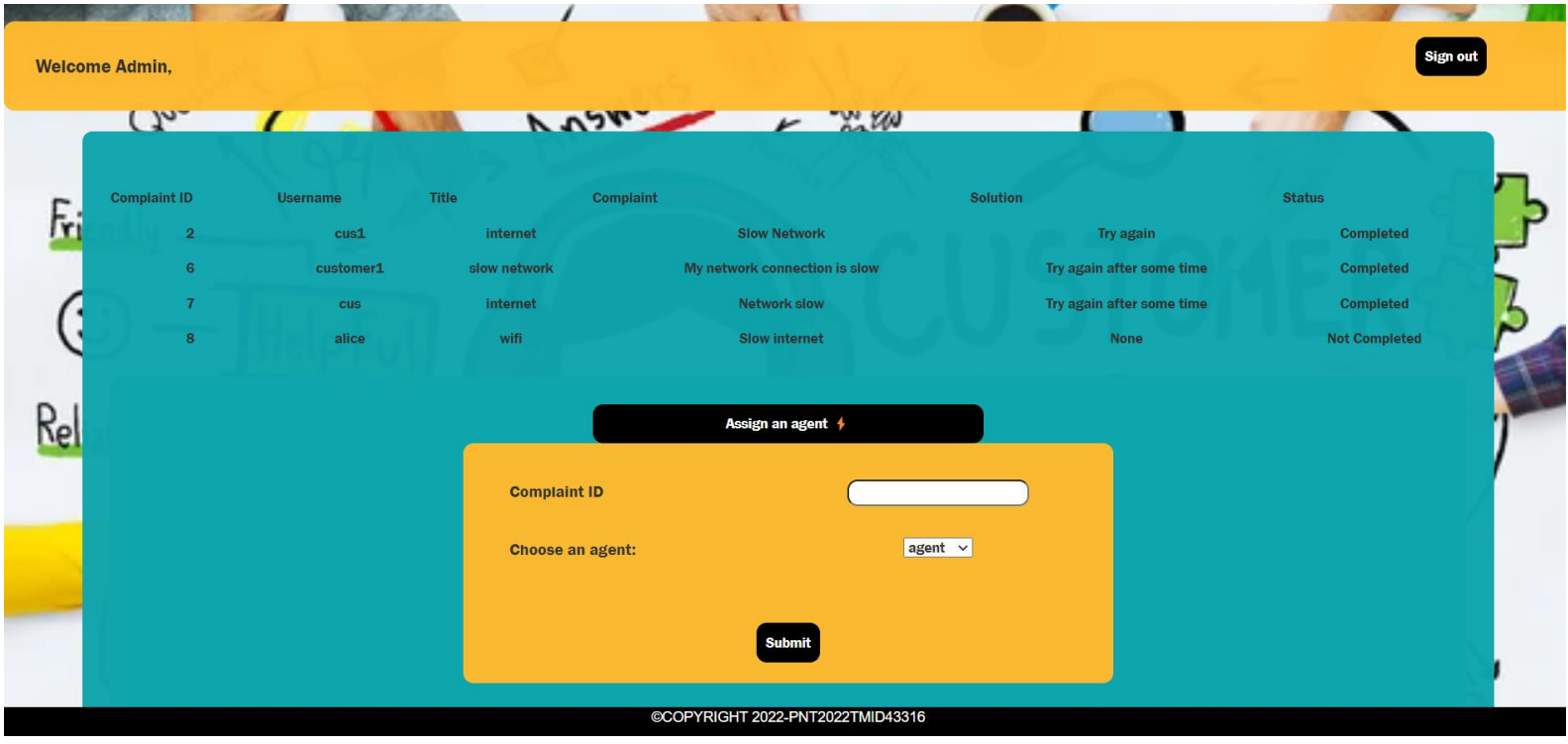

Customer Care Registry
Home
Agents
All Tickets
Sign In
Username
admin
Password
Sign In >>
New user? Sign up
Friendly
Reliable
©COPYRIGHT 2022-PNT2022TMID43316

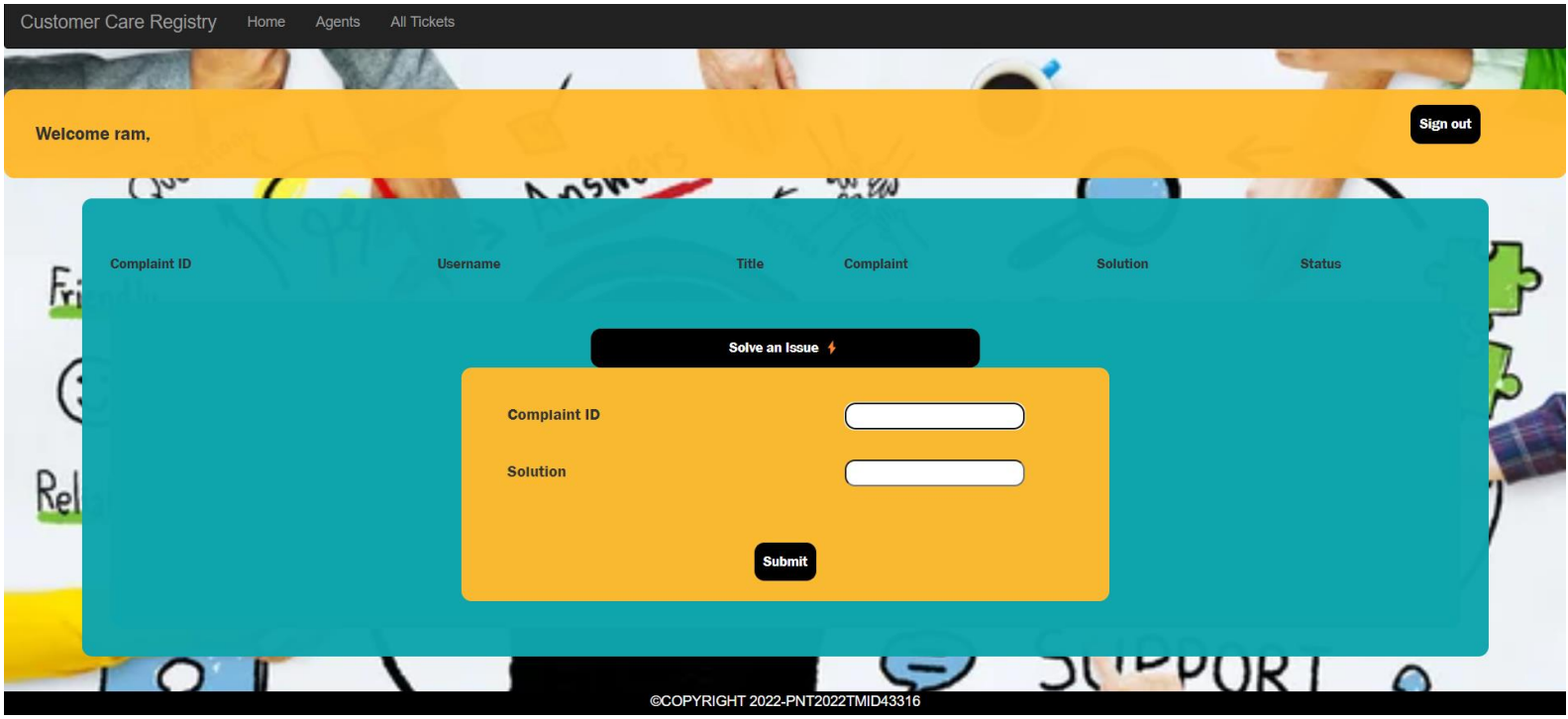

Customer Care Registry
Home
Agents
All Tickets
Register Now!!
Username
Name
E - mail
Phone Number
Password
Re - enter Password
Sign up >>
Already have an account? Sign in
Friendly
Reliable
©COPYRIGHT 2022-PNT2022TMID43316

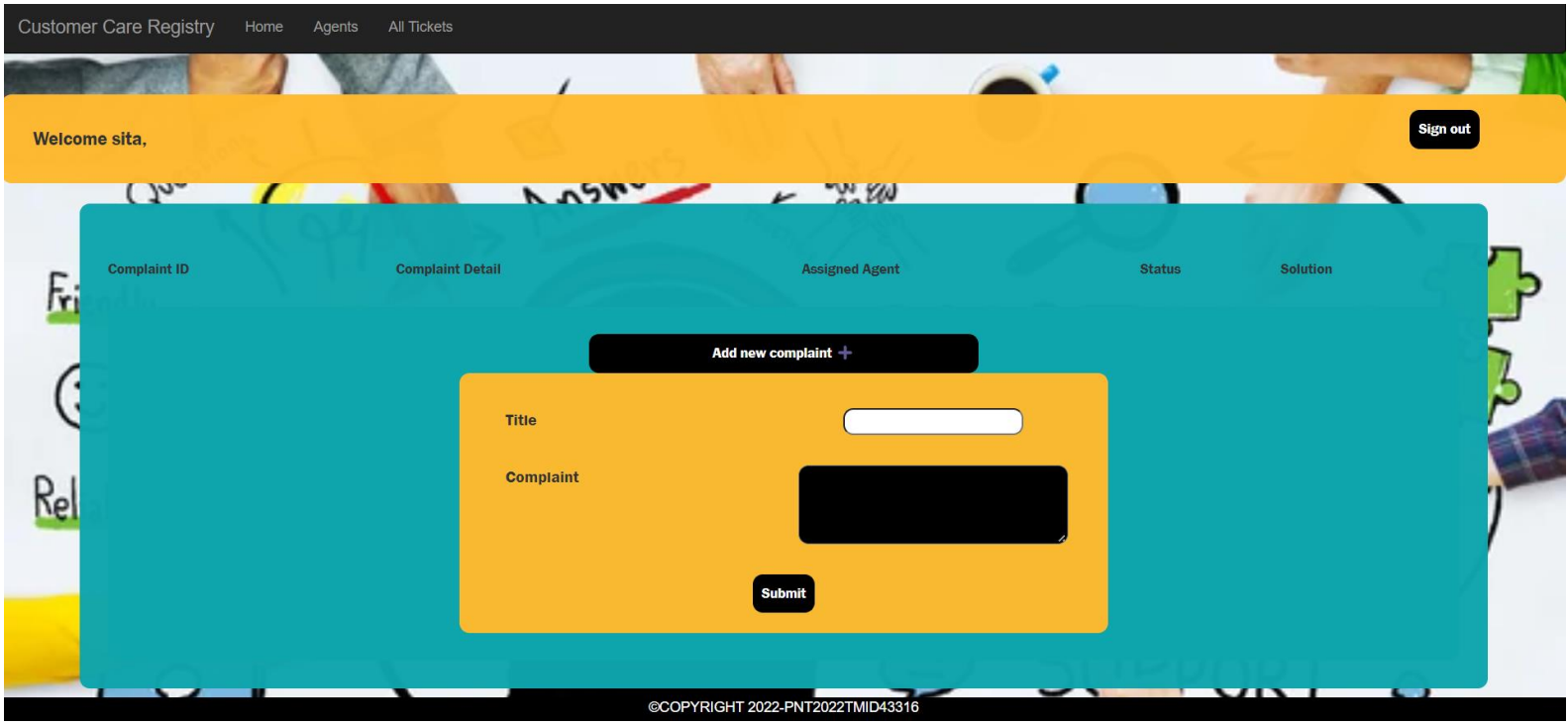

Customer Care Registry
Home
Agents
All Tickets
Welcome Admin,
Sign out
Name
Username
Email
Phone
Domain
Status
agent
agent
agent@gmail.com
9087654321
api
Available
agent1
agent1
agent1@gmail.com
9876543210
api
Available
agnt
agnt
agnt@gmail.com
9878656434
api
Available
bob
bob
bob@gmail.com
9807654321
api
Available
Add new agent +
SUPPORT
©COPYRIGHT 2022-PNT2022TMID43316


agent1
agent1
agent1@gmail.com
9876543210
api
Available
agnt
agnt
agnt@gmail.com
9878656434
api
Available
bob
bob
bob@gmail.com
9807654321
api
Available
Add new agent +
Username
Name
Email
Phone
Domain
Password
Submit
©COPYRIGHT 2022-PNT2022TMID43316


Customer Care Registry
Home
Agents
All Tickets
Welcome Admin,
Sign out
Complaint ID
Username
Title
Complaint
Solution
Status
2
cus1
internet
Slow Network
Try again
Completed
6
customer1
slow network
My network connection is slow
Try again after some time
Completed
7
cus
internet
Network slow
Try again after some time
Completed
8
alice
wifi
Slow internet
None
Not Completed
Assign an agent
SUPPORT
127.0.0.1:5000
©COPYRIGHT 2022-PNT2022TMID43316


Welcome Admin,
Sign out
Complaint ID
Username
Title
Complaint
Solution
Status
2
cus1
internet
Slow Network
Try again
Completed
6
customer1
slow network
My network connection is slow
Try again after some time
Completed
7
cus
internet
Network slow
Try again after some time
Completed
8
alice
wifi
Slow internet
None
Not Completed
Assign an agent
Complaint ID
Choose an agent:
agent
Submit
©COPYRIGHT 2022-PNT2022TMID43316


Customer Care Registry
Home
Agents
All Tickets
Welcome ram,
Sign out
Complaint ID
Username
Title
Complaint
Solution
Status
Solve an Issue
Complaint ID
Solution
Submit
©COPYRIGHT 2022-PNT2022TMID43316


Customer Care Registry
Home
Agents
All Tickets
Welcome sita,
Sign out
Complaint ID
Complaint Detail
Assigned Agent
Status
Solution
Add new complaint
Title
Complaint
Submit
©COPYRIGHT 2022-PNT2022TMID43316

10.ADVANTAGES & DISADVANTAGES

Advantages :

1. Customer loyalty

Loyal customers have many benefits for businesses. 91% of customers say a positive customer service experience makes them more likely to make a further purchase (source: Salesforce Research). Also, investing in new customers is five times more expensive than retaining existing ones (source: Invesp). Creating loyal customers through good customer service can therefore provide businesses with lucrative long-term relationships.

2. Increase profits

These long-term customer relationships established through customer service can help businesses become more profitable. Businesses can grow revenues between 4% and 8% above their market when they prioritise better customer service experiences (source: Bain & Company). Creating a better customer service experience than those offered by competitors can help businesses to standout in their market place, and in turn make more sales.

3. Customer recommendations

Providing good customer service can create satisfied customers, who are then more likely to recommend the business to others. 94% of customers will recommend a company whose service they rate as “very good” (source: Qualtrics XM Institute). This is useful, as 90% of customers are influenced by positive reviews when buying a product (source: Zendesk). Customers recommending a company through word of mouth or online reviews can improve the credibility of the business.

4. Increase conversion

Good customer service can help businesses turn leads into sales. 78% of customers say they have backed out of a purchase due to a poor customer experience (source: Glance). It is therefore safe to assume that providing good customer service will help to increase customer confidence and in turn increase conversion.

5. Improve public image

Customer service can help businesses to improve the public perception of the brand, which can then provide protection if there is a slip up. 78% of customers will forgive a company for a mistake after receiving excellent service (source: Salesforce Research). Meanwhile, almost 90% of customers report trusting a company whose service they rate as “very good.” On the other hand, only 16% of those who give a “very poor” rating trust companies to the same degree(source: Qualtrics XM Institute). Creating positive customer experiences is vital in gaining customer trust and creating a strong public image.

Disadvantage :

The Consumer Protection Act in India has numerous restrictions and drawbacks, which are listed in this article.

- Only services for which a particular payment has been made are covered under the consumer protection act. However, it does not protect medical professionals, or hospitals, and covers cases when this act does not apply to free medical care.

- This act does not apply to mandatory services, such as water supply, that are provided by state agencies.

- Only two clauses related to the supply of hazardous materials are covered by this act. Consumer redress is not given any power by the consumer
protection act.

- The consumer protection act focuses on the supply of ineffective products, but there are no strict regulations for those who produce it.

11.CONCLUSION

Application software has been computed successfully and was also tested successfully by taking “test cases”. It is user friendly, and has required option, which can be utilized by the user to perform the desired operations. Application meets the information requirements specified to a great extent. The system has been designed keeping in view the present and future requirements in mind and made very flexible. The goals that are achieved by the software are Instant access, improved productivity, Optimum utilization of resources, Efficient management of records, Simplifications of the operations, Less processing time and getting required information, User friendly, Portable and flexible for further enhancement. The system has the benefits of easy access because it is be developed as a platform independent web application, so the admin can maintain a proper contact with their users, which may be access anywhere. All communications between the police and administrator has done through the online, so this communication cost also is reduced.

12.FUTURE SCOPE

FUTURE IMPROVEMENT

- Developing this web application for more user and keep updatting the bugs.
- Planning on adding new features.
- Making the website more interactive to the user.

13.APPENDIX

SOURCE CODE

admin.html

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

{% block head %}

<title>
    Admin Dashboard
</title>

{% endblock %}

{% block body %}

<br>

<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 %}
```

agentdash.html

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



{% block head %}

<title>
    Agent Dashboard
</title>



{% endblock %}



{% block body %}

<br>

<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 %}
```

agents.html

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

{% block head %}

<title>
```

```
    Dashboard
</title>

{% endblock %}

{% block body %}

<br>

<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 %}
```

base.html

```
<!DOCTYPE html>

<head>

    <link rel="stylesheet" href="static/css/main.css"/>

    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></script>
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>


    {% block head %}
    {% endblock %}
```

```
</head>

<body>
    <nav class="navbar navbar-inverse">
        <div class="container-fluid">
          <div class="navbar-header">
            <a class="navbar-brand" href="/">Customer Care Registry</a>
          </div>
          <ul class="nav navbar-nav">
            <li class=""><a href="/">Home</a></li>
            <li><a href="/agents">Agents</a></li>
            <li><a href="/tickets">All Tickets</a></li>

          </ul>
        </div>
      </nav>

    {% 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>
    <div class="footer" style="position: fixed;
    left: 0;
    bottom: 0;
    width: 100%;
    background-color: rgb(0, 0, 0);
    color: white;
    text-align: center;">
        <p>©COPYRIGHT 2022-PNT2022TMID43316</p>
    </div>

</body>

</html>
```

dashboard.html

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

{% block head %}

<title>
    Dashboard
</title>

{% endblock %}

{% block body %}

<br>


<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 %}
```

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 class="fordashboardtopelements1">
            {{ msg }}
        </div>
        <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" style="    color: #ac5c00;">Sign up</a>
        </div>
        <br>
    </div>

</div>
</div>

{% endblock %}
```

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 %}
```

tickets.html

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

{% block head %}

<title>
    Agent Dashboard
</title>

{% endblock %}

{% block body %}

<br>

<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 %}
```

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=SSL;SSLServerCertifica
te=DigiCertGlobalRootCA.crt;UID=zyr46226;PWD=fIKQqRnXOVfcA0Ht;", '', '')
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,"here")
        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('login.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.run(debug=True,port=5000)
```

main.css

```
.sign {
    border-radius: 1rem;


    background-color: rgba(15, 165, 173, 0.976);
    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;
```

```css
}

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

body {
    background-image: url('https://st2.depositphotos.com/3591429/7169/i/450/depositphotos_71693845-stock-photo-diverse-people-and-customer-service.jpg');
    background-repeat: no-repeat;
    background-size: cover;
    /* 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: rgba(255, 185, 46, 0.976);

}

.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%;
}
```

```css
.fordashboarddetails {
    padding: 2%;
    border-radius: 1rem;
    background-color: rgba(15, 165, 173, 0.976);

}

.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: rgba(255, 185, 46, 0.976);
    width: 50%;
    /* overflow: hidden; */
    /* background-color: #f1f1f1; */
}
```

GITHUB :

GITHUB LINK : https://github.com/IBM-EPBL/IBM-Project-39364-1660408432