PROJECT REPORT

Project Name :	Customer Care Registry Project
Domain :	Cloud Application Development
College :	Bharathidasan Engineering College, Nattrampalli
Team ID :	PNT2022TMID39526
Team Size :	4
Team Members :	S. Gopika Soman
	S. Oviya Sree
	A. Seetha
	K. Shalini
Team Mentor :	Vasudeva Hanush
Github Link :	Click here

1. INTRODUCTION

To manage client interactions and solutions with the service provider over the phone or via email, a comprehensive online customer care registry is used. The system must be able to integrate with any service provider from any field or sector, including banking, telecommunications, insurance, etc.,

1.1. Project Overview

Customer Care Registry is crucial to the success of any business. However, there are several gaps in how this crucial service is provided, which hinders the organization's development. A study of pertinent recent research and literature highlights the legitimacy of this significant communication segment.

Customer Care is also known as Client Service is the provision of service to customers its significance varies by product, industry and domain.

1.2. Purpose

Customer Care Registry may be provided by a Person or Sales & Services Representatives Customer Care Registry is normally an integral part of a company's customer value proposition.

2. LITERATURE SURVEY

Reviewing prior relevant material that has already been investigated by other authors and can be a valuable resource for this study in terms of definitions and prior research findings is the topic of a literature review. The study will evaluate what constitutes customer service and the advantages of having a customer-focused organization.

The procedures and behaviours that make it simpler for customers to do business with a firm are referred to as customer service (Kotler, 2000). Depending on the situation, various people may have varying definitions of what customer service is. It is crucial that the business is clear about its goals while implementing "customer care" programmes and similar initiatives.

2.1 Existing problem

Customers will occasionally ask queries that catch your employees off guard or that they are unable to immediately respond to. But this doesn't imply they should ignore the situation and say, "I don't know."

Call transfers can be a pain for all parties involved. Customers frequently become irate after having to repeat information, and when several agents are working, it results in longer wait times for incoming calls or chats to be answered.

Customer satisfaction has become one of the key issues for companies in their efforts to improve quality in the competitive marketplace. It can be seen as either a goal of or a measurement tool in the development of construction quality.

Your clients are extremely busy. They don't have time to wait around; they expect you to meet them wherever they like to communicate. The support process slows down and can become frustrating when your customer service professionals lack the equipment needed to meet your customers wherever they are.

2.2 References

Marketing Management by Philip Kotler

Customer Satisfaction Theory by Helsen

Customer Relationship Management by V Kumar, Werner Reinartz

Mona N. Shah, Vineet Raitani, Aditya Oza and Kunal Gupta (2017) “Customer Satisfaction Study Of The Mumbai Metro Service”. NICMAR-Journal of construction management Vol. XXXII, No. 2, pp.30-42.

Pooria Rashvand and Muhd Zaimi Abd Majid (2014) “Critical Criteria on Client and Customer Satisfaction for the Issue of Performance Measurement”. Journal of Management in Engineering, Vol. 30, No. 1, January 1, 2014. ASCE, pp.10-18.

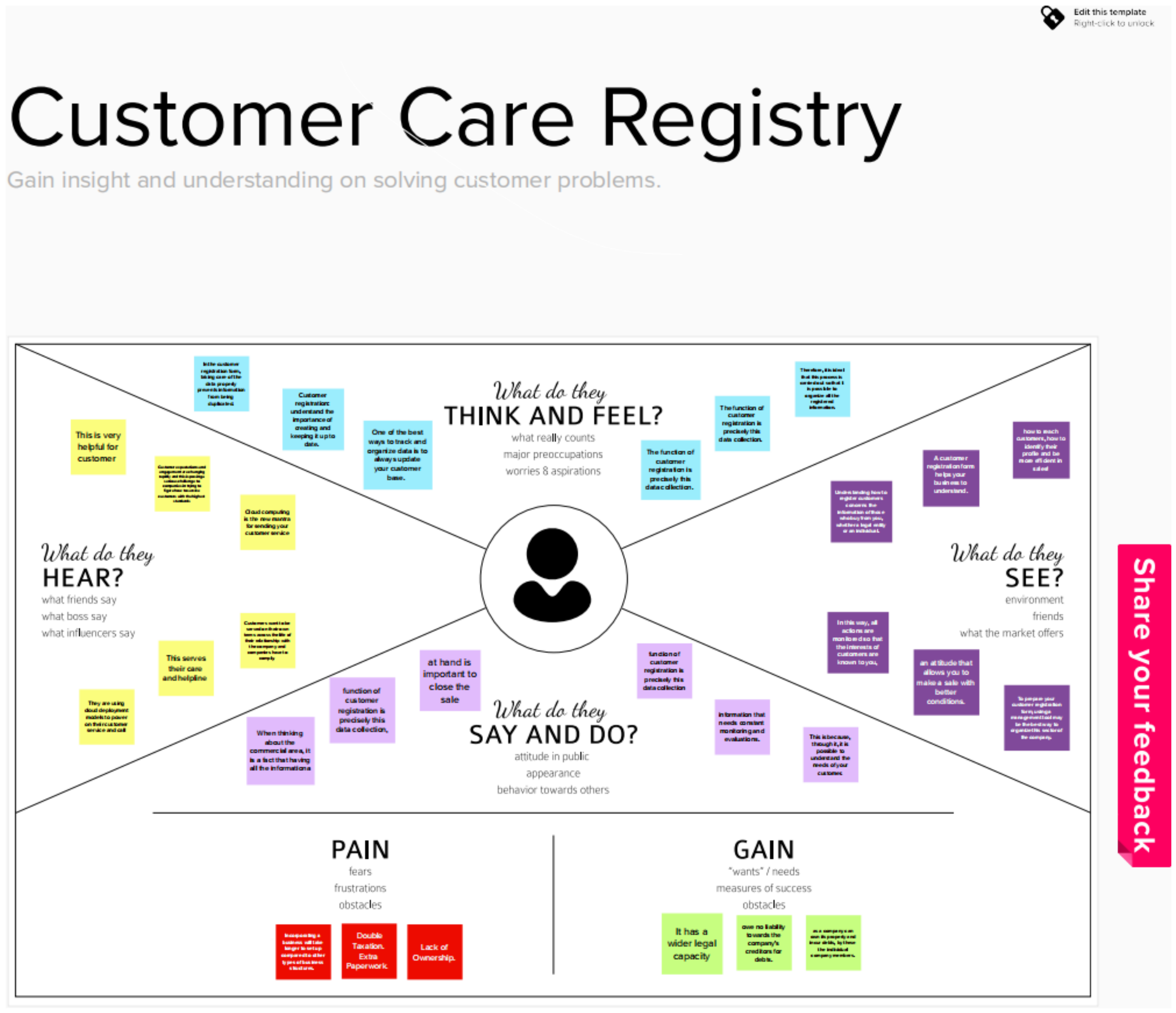
2.3 Problem Statement Definition

A problem statement is a brief explanation of the problem or problems that the project aims to solve. The current state, the anticipated future state, and any gaps between the two are all included in the issue statement. A problem statement is a crucial communication tool that may assist guarantee that everyone working on a project is aware of the issue they need to solve and the significance of the project.

A problem statement is crucial for a process improvement project since it clarifies the project's objectives and defines its parameters. It also aids in directing the actions and choices made by everyone involved in the project. A company or organisation might use the issue statement to win support and buy-in for a project to enhance a process.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



Customer Care Registry

Gain insight and understanding on solving customer problems.

What do they THINK AND FEEL?

what really counts
major preoccupations
worries 8 aspirations

- In the customer registration form, taking care of the data properly prevents information from being duplicated.
- Customer registration: understand the importance of creating and keeping it up to date.
- One of the best ways to track and organize data is to always update your customer base.
- The function of customer registration is precisely this data collection.
- The function of customer registration is precisely this data collection.
- Therefore, it is ideal that this process is centralized so that it is possible to organize all the registered information.

What do they HEAR?

what friends say
what boss say
what influencers say

- This is very helpful for customer
- [illegible]
- Cloud computing is the new mantra for sending your customer service
- Customers want to be served on their own terms across the life of their relationship with the company and companies have to comply.
- This serves their care and helpline
- They are using cloud deployment models to power on their customer service and call

What do they SEE?

environment
friends
what the market offers

- Understanding how to register customers concerns the information of those who buy from you, whether a legal entity or an individual.
- A customer registration form helps your business to understand.
- how to reach customers, how to identify their profile and be more efficient in sales!
- In this way, all actions are monitored so that the interests of customers are known to you,
- an attitude that allows you to make a sale with better conditions.
- To prepare your customer registration form, using a management tool may be the best way to organize this sector of the company.

What do they SAY AND DO?

attitude in public
appearance
behavior towards others

- When thinking about the commercial area, it is a fact that having all the information a
- function of customer registration is precisely this data collection,
- at hand is important to close the sale
- function of customer registration is precisely this data collection
- information that needs constant monitoring and evaluations.
- This is because, though it, it is possible to understand the needs of your customer.

PAIN

fears
frustrations
obstacles

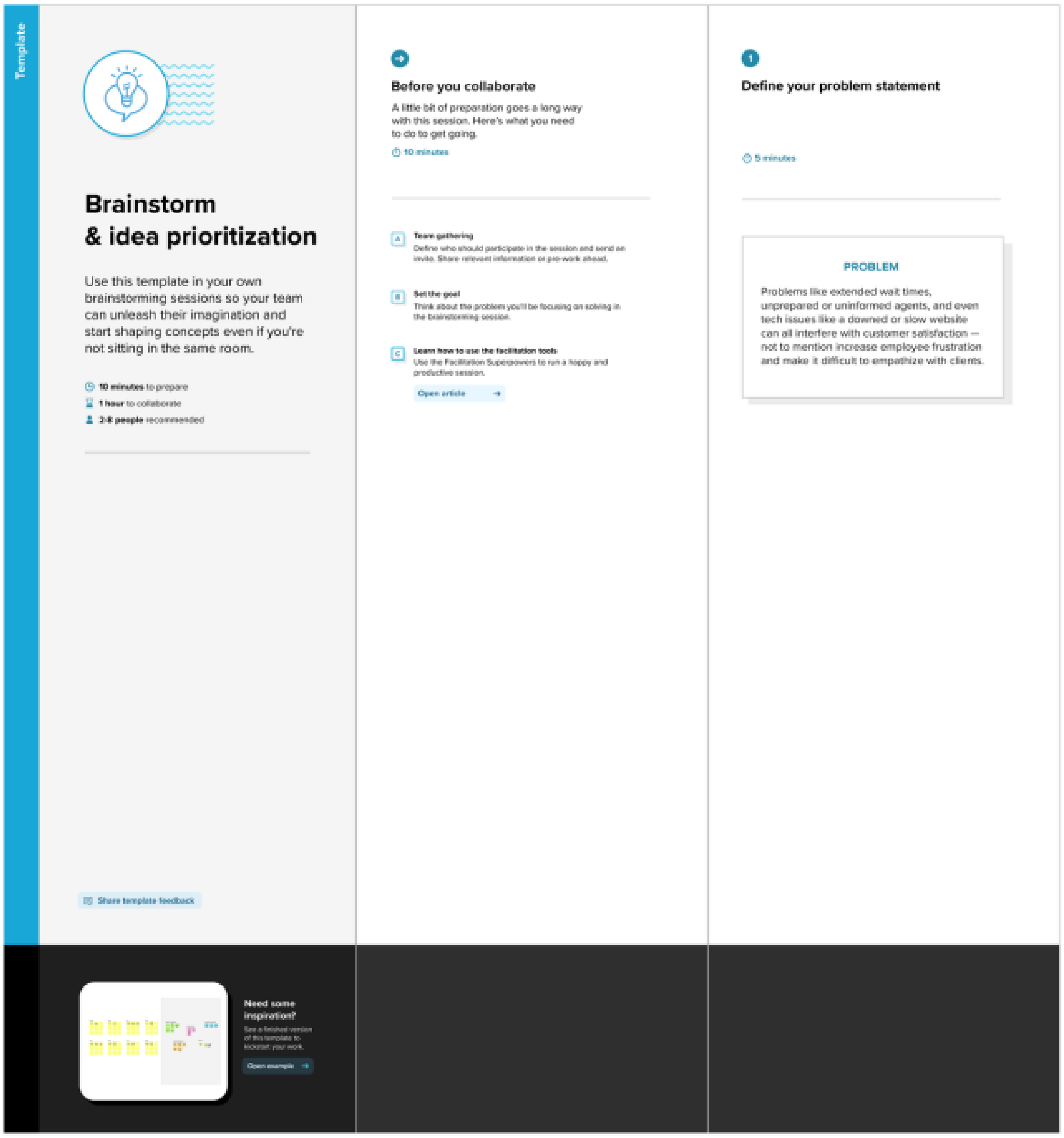
- Incorporating a business will take longer to set up compared to other types of business structures.
- Double Taxation. Extra Paperwork.
- Lack of Ownership.

GAIN

"wants" / needs
measures of success
obstacles

- It has a wider legal capacity
- owe no liability towards the company's creditors for debts.
- as a company can own its property and incur debts, by these the individual company members.

3.2 Ideation & Brainstorming



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

Need some inspiration?
See a finished version of this template to kickstart your work.
Open example

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

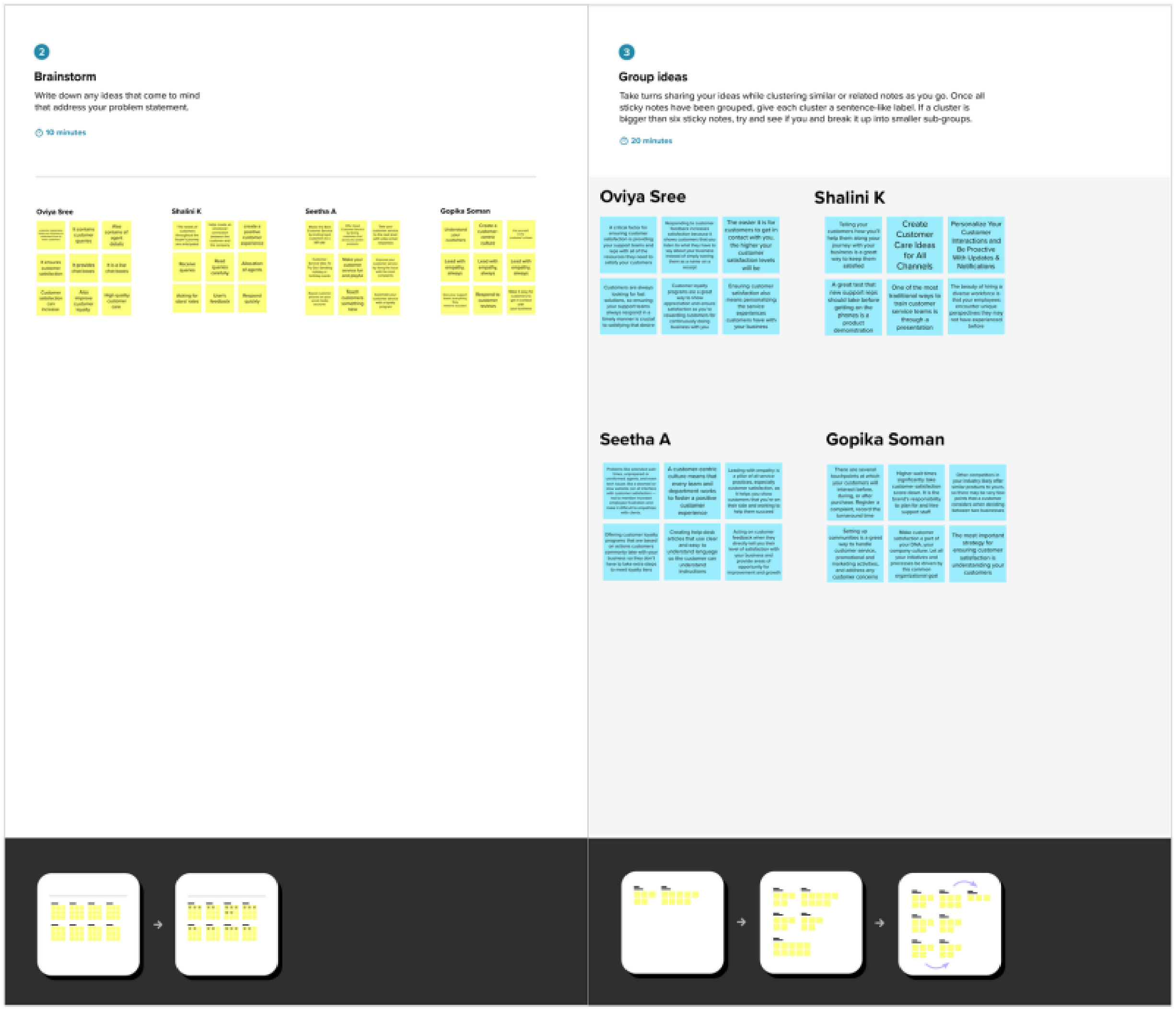
A **Team gathering**
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

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
5 minutes

PROBLEM
Problems like extended wait times, unprepared or uninformed agents, and even tech issues like a downed or slow website can all interfere with customer satisfaction — not to mention increase employee frustration and make it difficult to empathize with clients.



2

Brainstorm

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

10 minutes

Oviya Sree

- [illegible]
- It contains customer queries
- Also contains all agent details
- It ensures customer satisfaction
- It provides chat boxes
- It is a live chat boxes
- Customer satisfaction can increase
- Also improve customer loyalty
- High quality customer care

Shalini K

- [illegible]
- [illegible]
- Create a positive customer experience
- Receive queries
- Read queries carefully
- Allocation of agents
- Waiting for client roles
- User's feedback
- Respond quickly

Seetha A

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- Make your customer service fun and playful
- [illegible]
- [illegible]
- Teach customers something new
- [illegible]

Gopika Soman

- Understand your customers
- Create a customer-centric culture
- [illegible]
- Lead with empathy always
- Lead with empathy always
- Lead with empathy always
- [illegible]
- Respond to customer reviews
- Make it easy for customers to get in contact with your business

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

Oviya Sree

- A critical factor for ensuring customer satisfaction is providing your support teams and reps with all of the resources they need to satisfy your customers
- Responding to customer feedback increases satisfaction because it shows customers that you listen to what they have to say about your business instead of simply seeing them as a name on a receipt
- The easier it is for customers to get in contact with you, the higher your customer satisfaction levels will be
- Customers are always looking for fast solutions, so ensuring your support teams always respond in a timely manner is crucial to satisfying that desire
- Customer loyalty programs are a great way to show appreciation and ensure satisfaction as you're rewarding customers for continuously doing business with you
- Ensuring customer satisfaction also means personalizing the service experiences customers have with your business

Shalini K

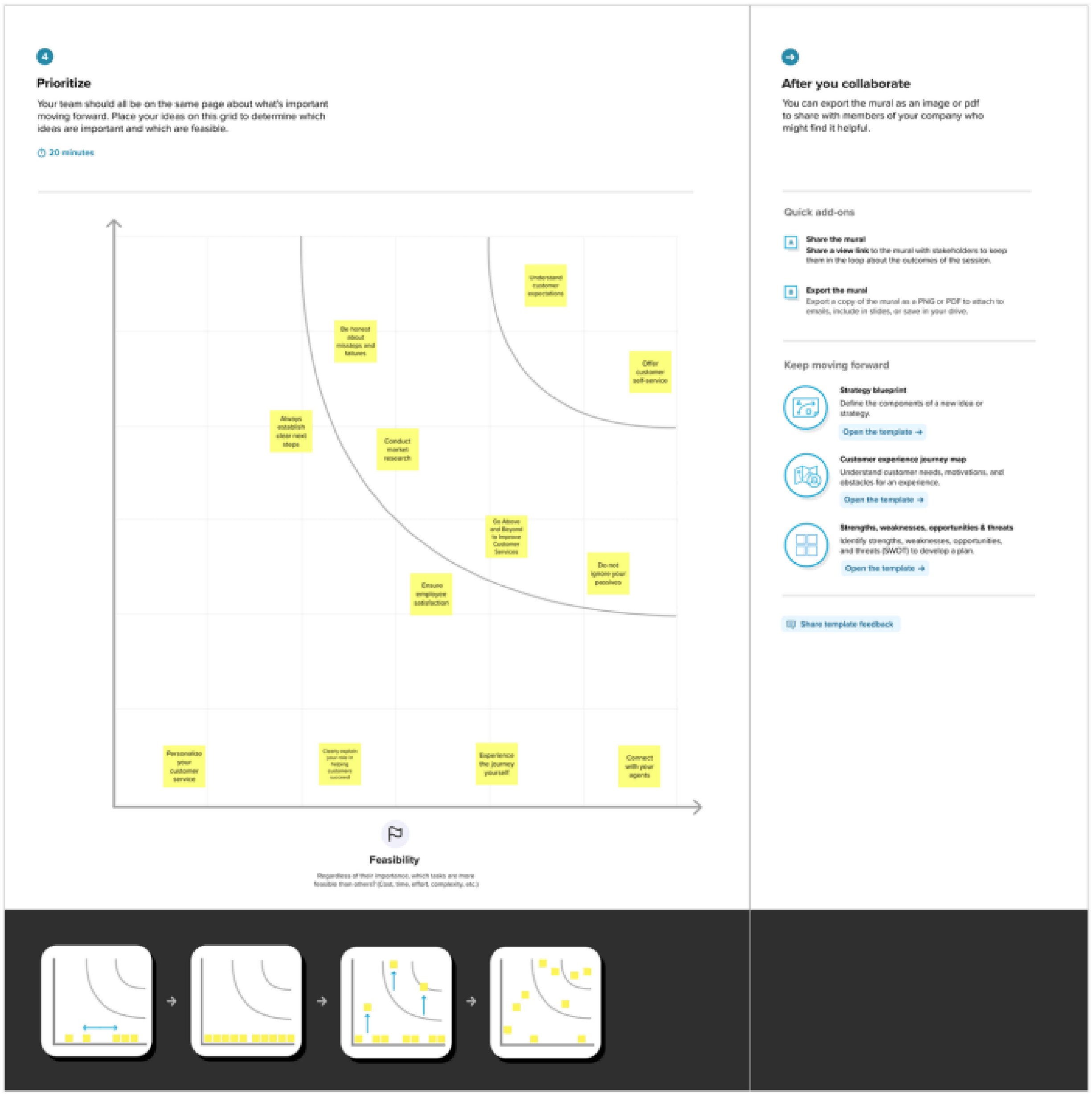
- Telling your customers how you'll help them along your journey with your business is a great way to keep them satisfied
- Create Customer Care Ideas for All Channels
- Personalize Your Customer Interactions and Be Proactive With Updates & Notifications
- A great tool that new support reps should take before getting on the phones is a product demonstration
- One of the most traditional ways to train customer service teams is through a presentation
- The beauty of hiring a diverse workforce is that your employees encounter unique perspectives they may not have experienced before

Seetha A

- [illegible]
- A customer-centric culture means that every team and department works to foster a positive customer experience
- Leading with empathy is a pillar of all service practices, especially customer satisfaction, as it helps you show customers that you're on their side and working to help them succeed
- Offering customer loyalty programs that are based on actions customers commonly take with your business so they don't have to take extra steps to meet loyalty tiers
- Creating help desk articles that use clear and easy to understand language so the customer can understand instructions
- Acting on customer feedback when they directly tell you their level of satisfaction with your business and provide areas of opportunity for improvement and growth

Gopika Soman

- There are several touchpoints at which your customers will interact before, during, or after purchase. Register a complaint, record the turnaround time
- Higher wait times significantly take customer satisfaction score down. It is the brand's responsibility to plan for and hire support staff
- Other competitors in your industry likely offer similar products to yours, so there may be very few points that a customer considers when deciding between two businesses
- Setting up communities is a great way to handle customer service, promotional and marketing activities, and address any customer concerns
- Make customer satisfaction a part of your DNA, your company culture. Let all your initiatives and processes be driven by this common organizational goal
- The most important strategy for ensuring customer satisfaction is understanding your customers


Understand customer expectations
Be honest about missteps and failures
Offer customer self-service
Always establish clear next steps
Conduct market research
Go Above and Beyond to Improve Customer Services
Do not ignore your passives
Ensure employee satisfaction
Personalize your customer service
Clearly explain your role in helping customers succeed
Experience the journey yourself
Connect with your agents
Feasibility

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

Feasibility

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

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

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

B 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 →

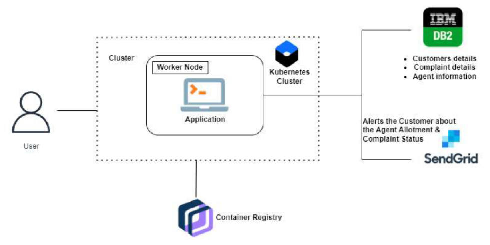
Customer experience journey map
Understand customer needs, motivations, and obstacles for an experience.
Open the template →

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

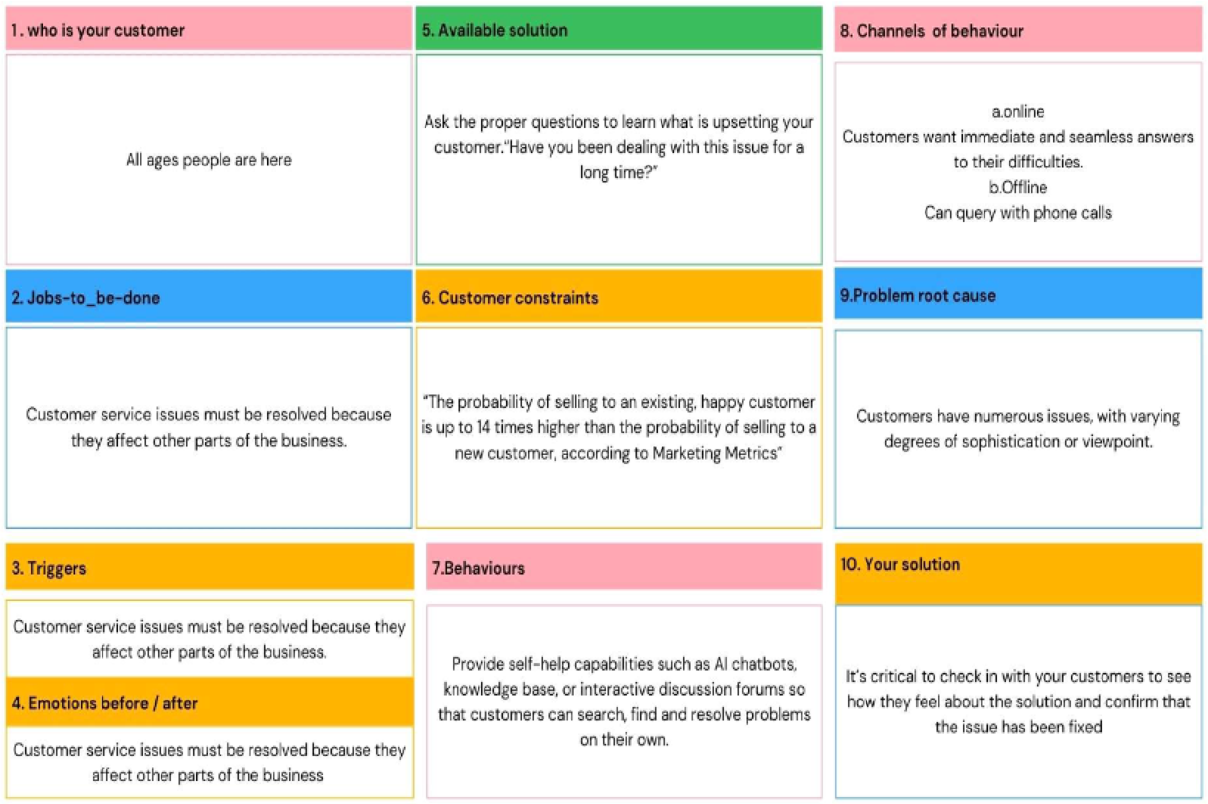
Share template feedback

3.3 Proposed Solution

S. No	Parameter	Description
1.	Problem Statement (Problem to be solved)	Customers in the present era expect instant communication with service departments. They, too, want immediate resolution to their concerns. This is without a doubt the first in a long list of common customer service issues that businesses must address.

2.	Idea / Solution description	Create a process that outlines the workflow of what an agent should do when he or she receives a customer query, with the goal of resolving it as quickly and efficiently as possible.
3.	Novelty / Uniqueness	1. Treat your employees as your first customer 2. Build an emotional connection with customers 3. Get real (time) about feedback 4. Focus furiously on individual customer needs 5. Practice Social Listening 6. Prove that you really, really appreciate your customers
4.	Social Impact / Customer Satisfaction	An organization's main focus must be to satisfy its customers. This applies to industrial firms, retail and wholesale businesses, government bodies, service companies, nonprofit organizations, and every subgroup within an organization.
5.	Business Model (Revenue Model)	 User Cluster Worker Node Application Kubernetes Cluster IBM DB2 • Customers details • Complaint details • Agent information Alerts the Customer about the Agent Allotment & Complaint Status SendGrid Container Registry
6.	Scalability of the Solution	Select the appropriate technology stack. Lay the groundwork for future expansion. Create a strong infrastructure. Simplify software deployment. Prepare for whatever may occur.

3.4 Problem Solution fit



1. who is your customer

All ages people are here

5. Available solution

Ask the proper questions to learn what is upsetting your customer."Have you been dealing with this issue for a long time?"

8. Channels of behaviour

a.online
Customers want immediate and seamless answers to their difficulties.
b.Offline
Can query with phone calls

2. Jobs-to_be-done

Customer service issues must be resolved because they affect other parts of the business.

6. Customer constraints

"The probability of selling to an existing, happy customer is up to 14 times higher than the probability of selling to a new customer, according to Marketing Metrics"

9.Problem root cause

Customers have numerous issues, with varying degrees of sophistication or viewpoint.

3. Triggers

Customer service issues must be resolved because they affect other parts of the business.

4. Emotions before / after

Customer service issues must be resolved because they affect other parts of the business

7.Behaviours

Provide self-help capabilities such as AI chatbots, knowledge base, or interactive discussion forums so that customers can search, find and resolve problems on their own.

10. Your solution

It's critical to check in with your customers to see how they feel about the solution and confirm that the issue has been fixed

4. REQUIREMENT ANALYSIS

Understanding how customers relate to and perceive an organization's products or services and brand involves doing a customer needs analysis.

An effective customer requirements analysis will pinpoint each of the reasons why customers choose to buy goods and services. Additionally, it can identify their broader expectations for the company, including its brand promise and position within the larger market, as well as how well the brand is performing in terms of expected levels of customer satisfaction and experience.

There are enormous chances to cut expenses and boost profits without having a detrimental effect on the business's consumer-facing operations when a company recognises who its consumers really are and the nature of the glue that holds them to its brand. A brand may create a product or service roadmap with better assurance after completing a consumer requirements analysis since they will know what to expect from upcoming releases.

4.1. Functional Requirements

"Any Requirement Which Specifies What The System Should Do."

In other words, a functional requirement will outline a specific action that the system should take when a set of criteria are satisfied, such as "Send email when a new client joins up" or "Open a new account."

Delivering a high-quality product that precisely matches the customer's request is the aim of every project. The main means through which a client conveys their expectations to the project team are functional requirements. Functional requirements aid in guiding the project team's progress.

Uncertain requirements result in a vague scope, which makes the project difficult to complete from the start. Poorly specified scope causes schedule extensions and expense increases. The consumer may not have the resources to devote the necessary time and money, so they just accept a subpar product.

The consumer often has both requirements and wants. They could request a reduction in the scope after seeing the cost estimate. The scope is often reduced by eliminating some of the non-functional needs. When there are too many non-functional needs, the cost might rise fast, and when there are not enough, the user experience may suffer.

- Business rules
- Transaction corrections, adjustments and cancellation
- Administrative functions
- Authentication
- Authorization levels

4.2 Non-Functional requirements

"*Any Requirement That Specifies How The System Performs A Certain Function.*"

In other words, a non-functional requirement will describe how a system should behave and what limits there are on its functionality.

All the remaining needs that are not addressed by the functional requirements are referred to as non-functional requirements. Instead of defining particular behaviours, they define criteria that assess a system's performance, such as "Modified data in a database should be updated for all users accessing it within 2 seconds."

Why are non-functional requirements significant if they are not necessary for the product to function? Usability is the solution. The user experience is impacted by non-functional requirements since they specify how a system behaves, what features it has, and how it functions in general.

When properly defined and implemented, non-functional requirements will contribute to the system's usability and performance.

Since user expectations are characteristics of the product, non-functional requirements concentrate on them.

5. PROJECT DESIGN

5.1. Data Flow Diagram

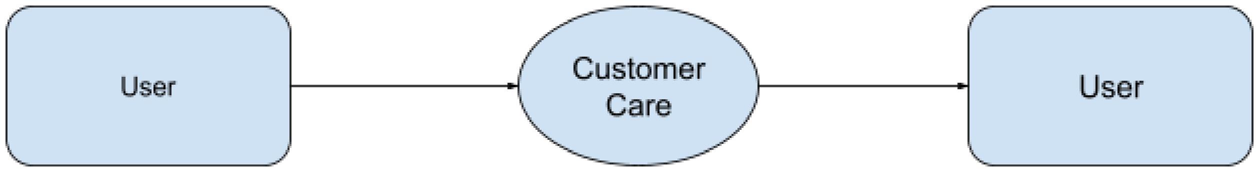


Diagram for admin

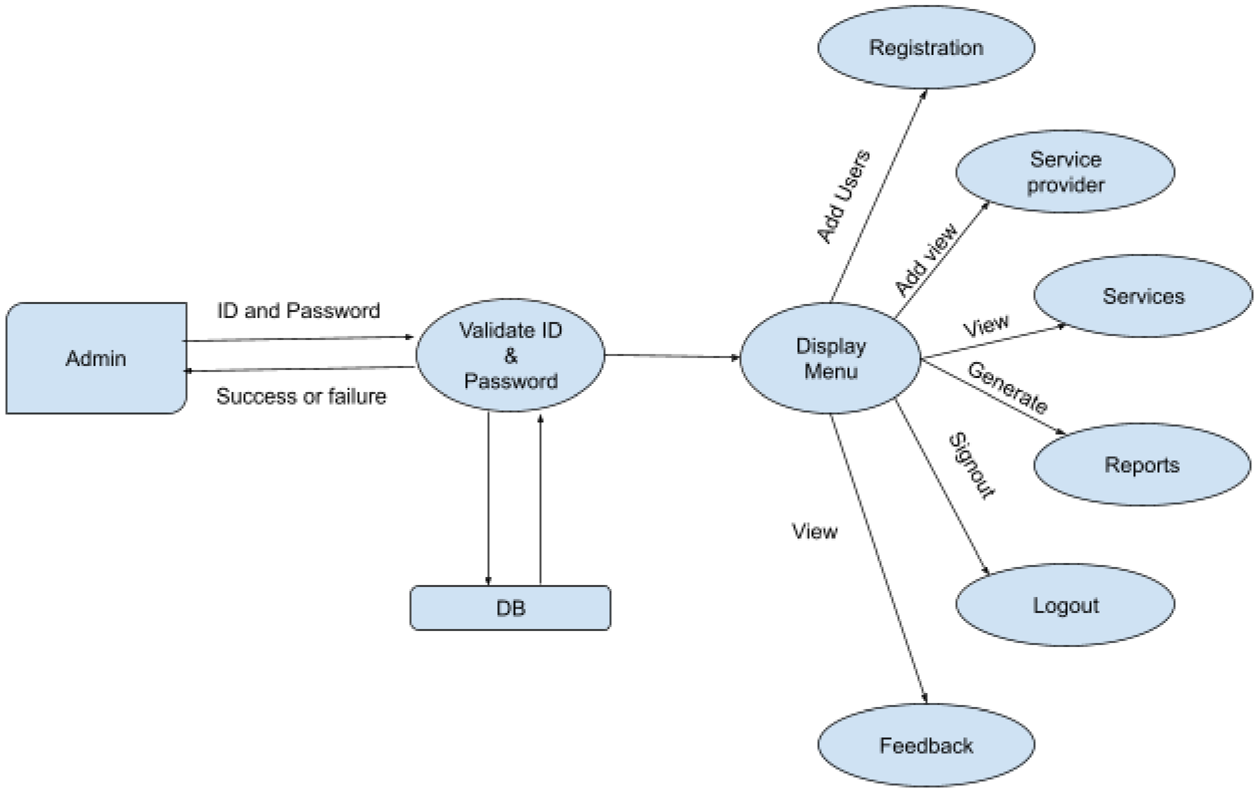


Diagram for customer

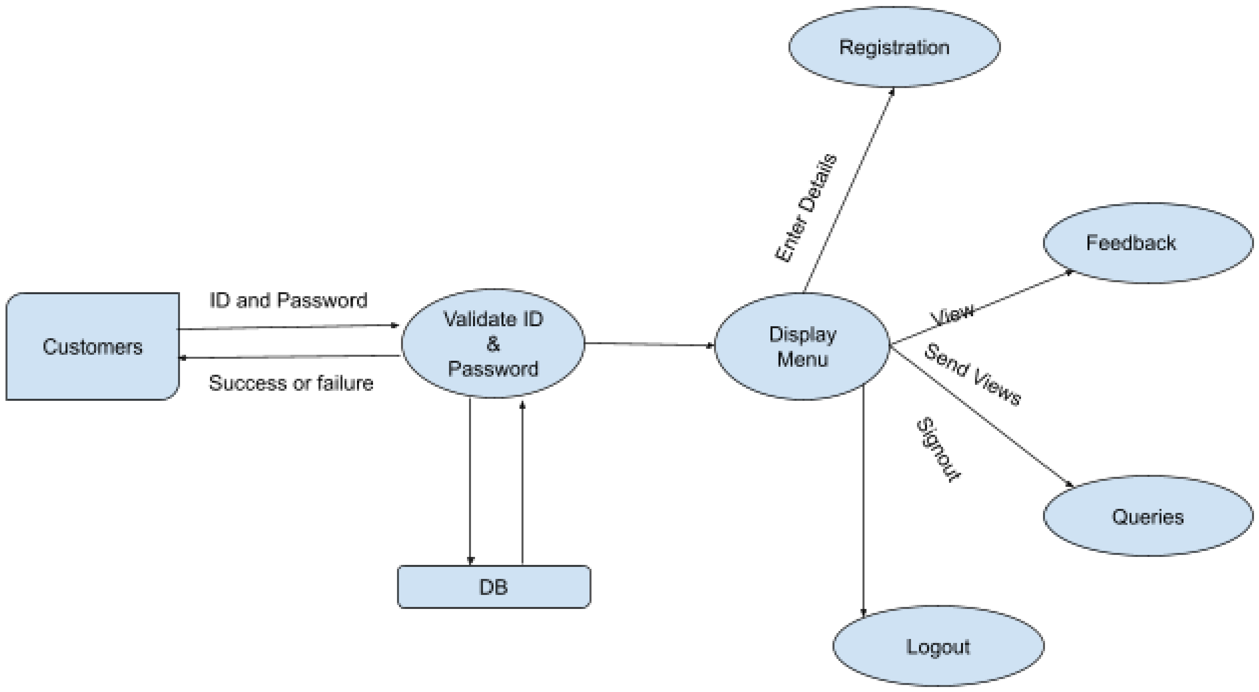
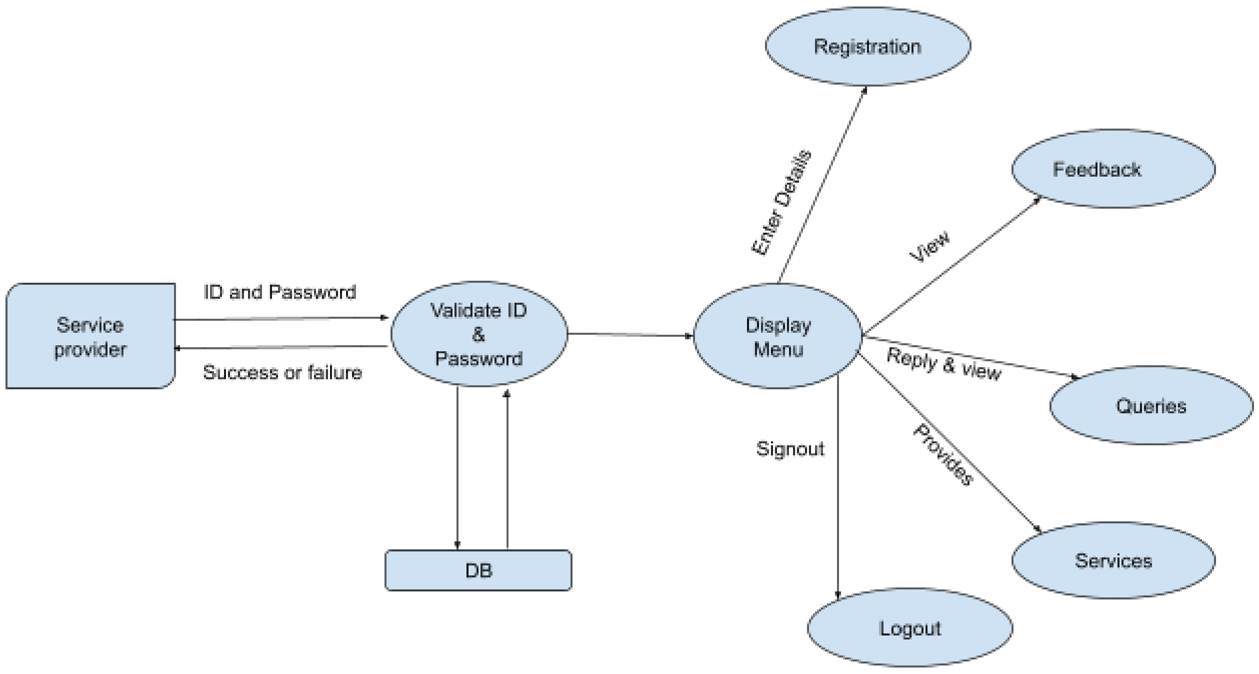
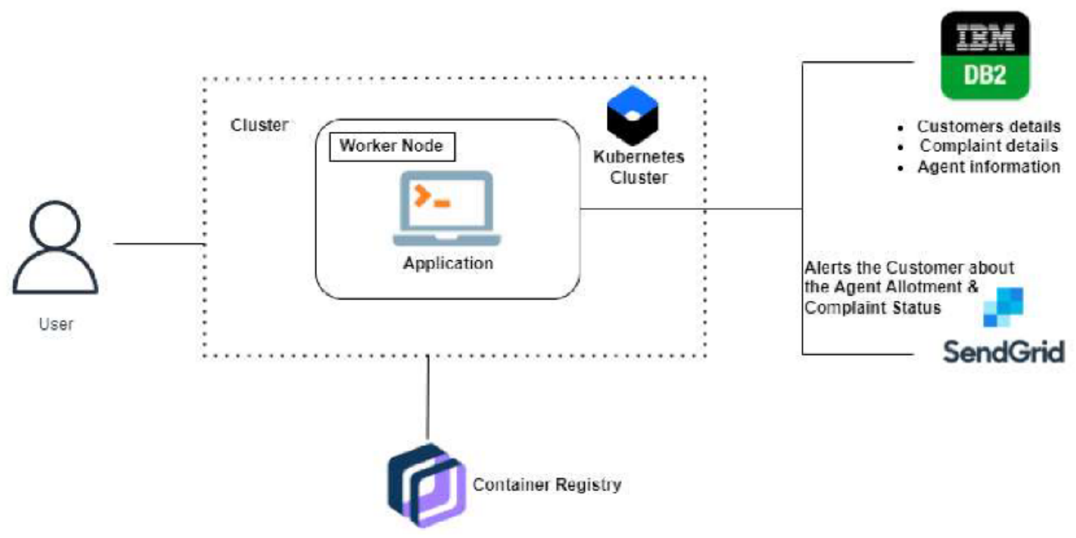


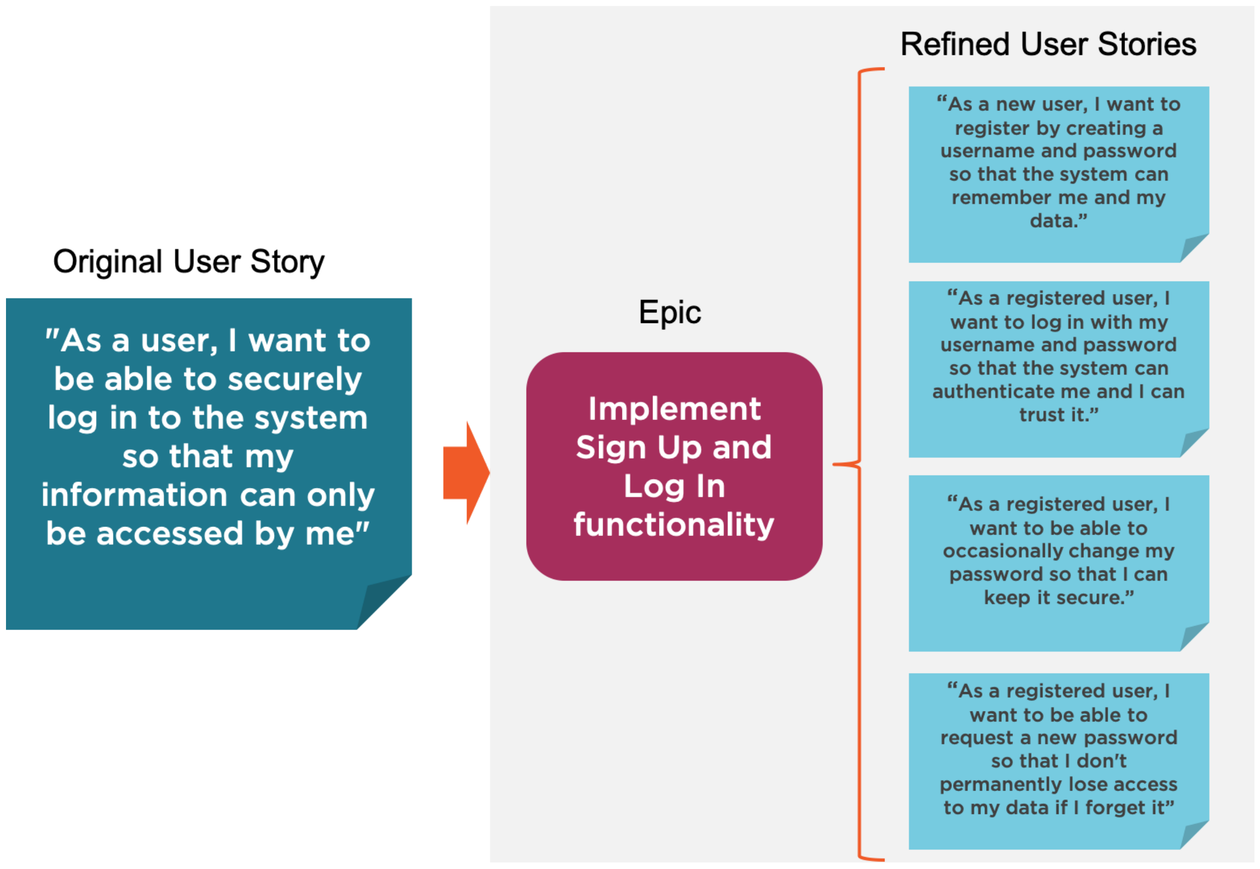
Diagram for service provider



5.2 Solution and Technical Architecture



5.3 User Stories


Original User Story
"As a user, I want to be able to securely log in to the system so that my information can only be accessed by me"
Epic
Implement Sign Up and Log In functionality
Refined User Stories
"As a new user, I want to register by creating a username and password so that the system can remember me and my data."
"As a registered user, I want to log in with my username and password so that the system can authenticate me and I can trust it."
"As a registered user, I want to be able to occasionally change my password so that I can keep it secure."
"As a registered user, I want to be able to request a new password so that I don't permanently lose access to my data if I forget it"

6. PROJECT PLANNING & SCHEDULING

6.1. Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Customer Panel	USN-1	As a Customer, I can register for the application by entering my email, password, and confirming my password and I will be able to Access my dashboard for creating a Query Order.	2	High	Gopika Soman Oviya Sree Shalini Seetha
Sprint-1	Admin Panel	USN-2	As an admin, I can Login to the Application by entering correct login credentials and I will be able to Access My dashboard to create Agents and Assign an Agent to a Query Order.	2	High	Seetha Oviya Sree
Sprint-2	Agent Panel	USN-3	As an agent, I can Login to the Application by entering correct login credentials and I will be able to Access my Dashboard to check the Query Order and I can Clarify the Issues.	2	High	Oviya Sree Gopika Soman Shalini
Sprint-3	Chat Bot	USN-4	The Customer can directly Interact to the Chatbot regarding the services offered by the Web Portal and get recommendations based on information provided by them.	2	Medium	Seetha Shalini Gopika Soman
Sprint-4	Final Delivery	USN-5	Container of applications using docker kubernetes and deployment the application.Create the documentation and final submit the application	2	High	Gopika Soman Oviya Sree Seetha Shalini

Project Tracker, Velocity & Burndown Chart: (4 Marks)

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	4 Days	28 Oct 2022	01 Nov 2022		01 Nov 2022
Sprint-2	20	7 Days	31 Oct 2022	06 Nov 2022		06 Nov 2022
Sprint-3	20	8 Days	07 Nov 2022	14 Nov 2022		14 Nov 2022
Sprint-4	20	7 Days	14 Nov 2022	21 Nov 2022		21 Nov 2022

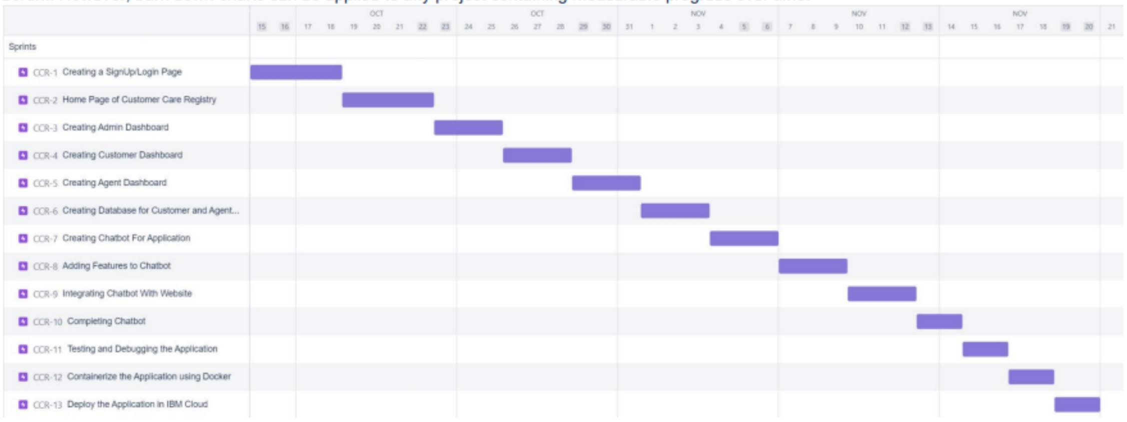
Velocity:

$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

Burndown Chart:

A burndown chart is a graphical representation of work left to do versus time. It is often used in agile software development methodologies such as Scrum. However, burn down charts can be applied to any project containing measurable progress over time.



6.2. Sprint Delivery Schedule

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Literature survey on the selected project & gathering information by referring the, technical papers,research publications etc.	20 OCTOBER 2022
Prepare Empathy Map	Prepare Empathy Map Canvas to capture the user Pains & Gains, Prepare list of problem statements	20 OCTOBER 2022
Ideation	List the by organizing the brainstorming session and prioritize the top 3 ideas based on the feasibility & importance.	20 OCTOBER 2022
Proposed Solution	Prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.	21 OCTOBER 2022
Problem Solution Fit	Prepare problem - solution fit document.	21 OCTOBER 2022
Solution Architecture	Prepare solution architecture document.	21 OCTOBER 2022

Customer Journey	Prepare the customer journey maps to understand the user interactions & experiences with the application (entry to exit).	22 OCTOBER 2022
Functional Requirement	Prepare the functional requirement document.	22 OCTOBER 2022
Data Flow Diagrams	Draw the data flow diagrams and submit for review.	22 OCTOBER 2022
Technology Architecture	Prepare the technology architecture diagram.	22 OCTOBER 2022
Prepare Milestone & Activity List	Prepare the milestones & activity list of the project.	31 OCTOBER 2022
Project Development - Delivery of Sprint-1, 2, 3 & 4	Develop & submit the developed code by testing it.	IN PROGRESS..

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

7.1 Feature 1

```
from flask import Blueprint, render_template, url_for, redirect
from flask_login import login_required, logout_user
from .views import conn, mail
import ibm_db
from .cust import QUERY_STATUS_OPEN

USER_ADMIN = "ADMIN"

admin = Blueprint("admin", __name__)

# query to get all the confirmed agents
get_confirmed_agents = '''
    SELECT first_name, agent_id FROM agent WHERE confirmed = ?
'''

@admin.route('/admin/tickets')
@login_required
def tickets():
    '''
        Loading all the OPEN tickets from the database
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # Query to get all the unassigned tickets raised by all the users
        get_unassigned_tickets = '''
            SELECT
                ticket_id,
                raised_on,
                customer.first_name,
                tickets.issue,
                customer.email
            FROM
                tickets
            JOIN
                customer ON tickets.raised_by = customer.cust_id
            AND
                tickets.assigned_to IS NULL
            ORDER BY
                raised_on ASC
        '''

        try:
            # getting the confirmed agents first
            stm = ibm_db.prepare(conn, get_confirmed_agents)
            ibm_db.bind_param(stm, 1, True)
            ibm_db.execute(stm)

            agents = ibm_db.fetch_assoc(stm)
            agents_list = []

            while(agents != False):
                temp = []

                temp.append(agents['FIRST_NAME'])
                temp.append(agents['AGENT_ID'])

                agents_list.append(temp)
                print(temp)

                agents = ibm_db.fetch_assoc(stm)

            # Getting the unassigned tickets
            stmt = ibm_db.prepare(conn, get_unassigned_tickets)
            ibm_db.execute(stmt)

            tickets = ibm_db.fetch_assoc(stmt)
            tickets_list = []

            if tickets:
                # means there are still some unassigned tickets
                while tickets != False:
                    temp = []

                    temp.append(tickets['TICKET_ID'])
                    temp.append(str(tickets['RAISED_ON'])[0:10])
                    temp.append(tickets['FIRST_NAME'])
                    temp.append(tickets['ISSUE'])
                    temp.append(tickets['EMAIL'])

                    print(temp)

                    tickets_list.append(temp)

                    tickets = ibm_db.fetch_assoc(stmt)

                return render_template(
                    'admin tickets.html',
                    id = 0,
                    tickets_to_show = True,
                    tickets = tickets_list,
                    msg = "These are the unassigned tickets",
                    agents = agents_list,
                    user = USER_ADMIN
                )

            else:
                # all the tickets may be assigned
                # may be, there are no tickets raised in the system at all
                return render_template(
                    'admin tickets.html',
                    id = 0,
                    tickets_to_show = False,
                    msg = "There is nothing to assign!",
                    user = USER_ADMIN
                )

        except:
            # something fishy happened while getting the tickets
            # so alerting the admin
            return render_template(
                'admin tickets.html',
                id = 0,
                to_show = True,
                message = "Something wrong! Please TrY Again",
                user = USER_ADMIN
            )
    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/agents')
@login_required
def agents():
    '''
        Returning all the confirmed agents from the database
    '''

    from .views import admin

    if(hasattr(admin, 'email')):
        # query to get all the confirmed agents
        get_confirmed = '''
            SELECT * FROM agent WHERE confirmed = ?
        '''

        try:
            stmt = ibm_db.prepare(conn, get_confirmed)
            ibm_db.bind_param(stmt, 1, True)
            ibm_db.execute(stmt)

            agents = ibm_db.fetch_assoc(stmt)
            agents_list = []

            if agents:
                # there are some confirmed agents
                while agents != False:
                    temp = []

                    temp.append(agents['AGENT_ID'])
                    temp.append(str(agents['DATE_JOINED'])[0:10])
                    temp.append(agents['FIRST_NAME'])
                    temp.append(agents['LAST_NAME'])
                    temp.append(agents['EMAIL'])

                    agents_list.append(temp)
                    agents = ibm_db.fetch_assoc(stmt)

                return render_template(
                    'admin agents.html',
                    id = 1,
                    msg = "List of confirmed agents",
                    agents_to_show = True,
                    agents = agents_list,
                    user = USER_ADMIN
                )

            else:
                # no confirmed agents present
                return render_template(
                    'admin agents.html',
                    id = 1,
                    msg = "No agents present",
                    agents_to_show = False,
                    user = USER_ADMIN
                )

        except:
            # something happened while fetching the agents
            return render_template(
                'admin agents.html',
                id = 1,
                mmessage = "Something happened! Please try again",
                to_show = True,
                user = USER_ADMIN
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/accept')
@login_required
def accept():
    '''
```

```
        Loading the agents info from the database who are not yet
confirmed
    '''

    from .views import admin

    if(hasattr(admin, 'email')):
        # query to get all the agents from the database who are all not
confirmed yet
        get_agents_query = '''
            SELECT * FROM agent WHERE confirmed = ?
        '''

        agents_to_show = False
        msg = ""

        try:
            stmt = ibm_db.prepare(conn, get_agents_query)
            ibm_db.bind_param(stmt, 1, False)
            ibm_db.execute(stmt)

            agents = ibm_db.fetch_assoc(stmt)

            agents_list = []

            while agents != False:
                temp = []

                temp.append(agents['AGENT_ID'])
                temp.append(agents['EMAIL'])
                temp.append(agents['FIRST_NAME'])
                temp.append(agents['DATE_JOINED'])

                agents_list.append(temp)

                agents = ibm_db.fetch_assoc(stmt)

            if len(agents_list) >= 1:
                # there are some agents who are not yet confirmed
                msg = "These are the pending requests"
                agents_to_show = True

            else:
                agents_to_show = False
                msg = "There are no pending requests"

            return render_template(
                'admin acc agent.html',
                id = 2,
                agents = agents_list,
                agents_to_show = agents_to_show,
                msg = msg,
                user = USER_ADMIN
            )

        except:
            # something happened while admin either accepts/denies the
agent
            return render_template(
                'admin acc agent.html',
                to_show = True,
                message = "Something went wrong!",
                id = 2,
                user = USER_ADMIN
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/about')
@login_required
def about():
    '''
        Showing the about of the application to the admin
    '''

    from .views import admin

    if(hasattr(admin, 'email')):
        return render_template(
            'admin about.html',
            id = 3,
            user = USER_ADMIN
        )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/support')
@login_required
def support():
    '''
        Showing all the feedbacks given by the agents and customers
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # query to retrieve all the feedbacks submitted
        get_feedbacks_query = '''
            SELECT * FROM feedback ORDER BY RAISED_ON DESC
        '''

        try:
            stmt = ibm_db.prepare(conn, get_feedbacks_query)
            ibm_db.execute(stmt)
            feedbacks = ibm_db.fetch_assoc(stmt)
            feedbacks_list = []

            feedbacks_to_show = False

            while feedbacks != False:
                temp = []

                temp.append(str(feedbacks['RAISED_ON'])[0:10])
                temp.append(feedbacks['RAISED_BY'])
                temp.append(feedbacks['RAISED_NAME'])
                temp.append(feedbacks['FEED'])
                feedbacks_list.append(temp)

                feedbacks = ibm_db.fetch_assoc(stmt)

            if len(feedbacks_list) > 0:
                # some feedbacks are submitted
                msg = 'All the feedbacks from the customers and agents'
                feedbacks_to_show = True

            else:
                # no feedbacks submitted yet
                msg = 'No feedbacks yet!'

            return render_template('admin support.html',
                id = 4,
                user = USER_ADMIN,
                feedbacks = feedbacks_list,
                msg = msg,
                true = feedbacks_to_show
            )

        except:
            # something happened while fetching the feedbacks
            return render_template('admin support.html',
                id = 4,
                user = USER_ADMIN,
                to_show = True,
                message = 'Something went wrong! Please try again'
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/<email>/<action>')
@login_required
def alter(email, action):
    '''
        Either accepting or denying the agent, as per the admin's decision
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        if action == "True":
            # admin chose to the accept the agent
            accept_query = '''
                UPDATE agent SET confirmed = ? WHERE email = ?
            '''

            stmt = ibm_db.prepare(conn, accept_query)
            ibm_db.bind_param(stmt, 1, True)
            ibm_db.bind_param(stmt, 2, email)

            ibm_db.execute(stmt)

        else:
            # admin must have chosen to delete the agent
            delete_query = '''
                DELETE FROM agent WHERE email = ?
            '''

            stmt = ibm_db.prepare(conn, delete_query)
            ibm_db.bind_param(stmt, 1, email)

            ibm_db.execute(stmt)

        return "None"

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/update/<agent_id>/<ticket_id>/<cust_email>/')
@login_required
def assign(agent_id, ticket_id, cust_email):
    '''
        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)

    # sending the acknowledgement mail to the customer
    mail.sendEmail(
        f'Customer Care Registry',
        f'''
            An agent has been assigned for your ticket <br/>
            <strong>Ticket ID : {ticket_id}</strong> <br>
            <br>
            Please login into CCR for more details! <br/>
            Thank you!!!
        ''',
        [f'{cust_email}']
    )

    return "None"

else:
    # logging out
    return redirect(url_for('blue_print.logout'))
```

8. TESTING

8.1. User Acceptance Testing

Acceptance Testing
UAT Execution & Report Submission

Date	03 November 2022
Team ID	PNT2022TMID39526
Project Name	Project - Customer Care Registry
Maximum Marks	4 Marks

- ### Purpose of Document

  The purpose of this document is to briefly explain the test coverage and open issues of the [ProductName] project at the time of the release to User Acceptance Testing (UAT).

- ### 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	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1

Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

- ## Test Case Analysis

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

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2
Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

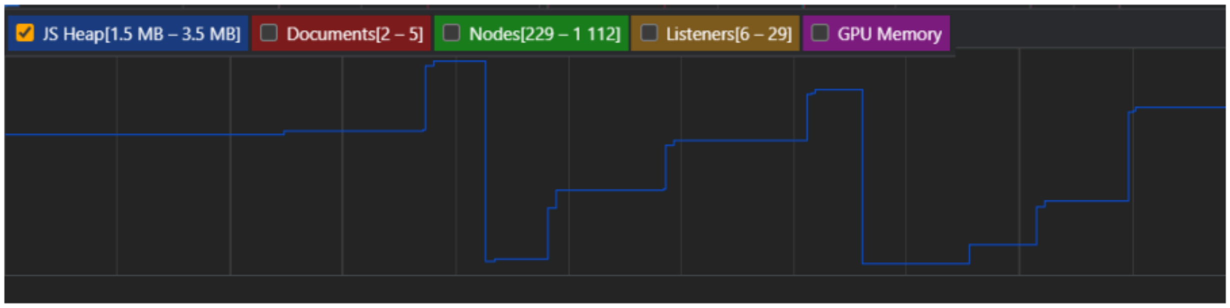
9. RESULTS

9.1 Performance Metrics:

CPU usage:

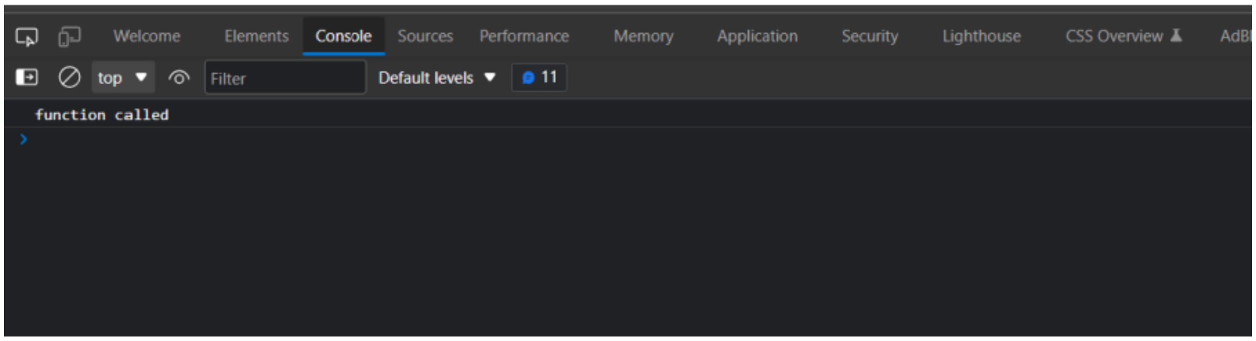
✓ Since all the operations run using Flask is in server-side, the client (browser) need not worry about the CPU usage. Just rendering the page, static contents take place in the client-side.

✓ 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.



```
function called
>
```

10. ADVANTAGES & DISADVANTAGES

Advantages:

✓ Customers can clarify their doubts just by creating a new ticket
✓ Customer gets replies as soon as possible
✓ Not only the replies are faster, the replies are more authentic and practical
✓ Customers are provided with a unique account, to which the latter can login at any time
✓ Very minimal account creation process
✓ Customers can raise as many tickets as they want
✓ Application is very simple to use, with well-known UI elements
✓ Customers are given clear notifications through email, of all the processes related lo login, ticket creation etc.,
✓ Customers' feedbacks are always listened
✓ Free of cost

Disadvantages:

× Only web application is available right now (as of writing)
× UI is not so attractive, it's just simple looking
× No automated replies
× No SMS alerts × Supports only text messages while chatting with the Agent
× No tap to reply feature
× No login alerts
× Cannot update the mobile number
× Account cannot be deleted, once created
× Customers cannot give feedback to the agent for clarifying the queries

11. CONCLUSION

Thus, there are many customer service applications available on the internet. Noting down the structural components of those applications and we built a customer care registry application. It will be a web application build with Flask (Python micro-web framework), HTML, JavaScript. It will be a ticket-based customer service registry.

Customers can register into the application using their email, password, first name and last name. Then, they can login to the system, and raise as tickets as they want in the form of their tickets.

These tickets will be sent to the admin, for which an agent is assigned. Then, the assigned agent will have a one-to-one chat with the customer and the latter's queries will be clarified. It is also the responsibility of the admin, to create an agent.

12. FUTURE SCOPE

Our application is not finished yet. There are many rooms for improvement. Some of them will be improved in the future versions

- ✓ Attracting and much more responsive UI throughout the application
- ✓ Releasing cross-platform mobile applications
- ✓ Incorporating automatic replies in the chat columns
- ✓ Deleting the account whenever customer wishes to
- ✓ Supporting multi-media in the chat columns
- ✓ Creating a community for our customers to interact with one another
- ✓ Call support
- ✓ Instant SMS alerts

13. APPENDIX
Source Code

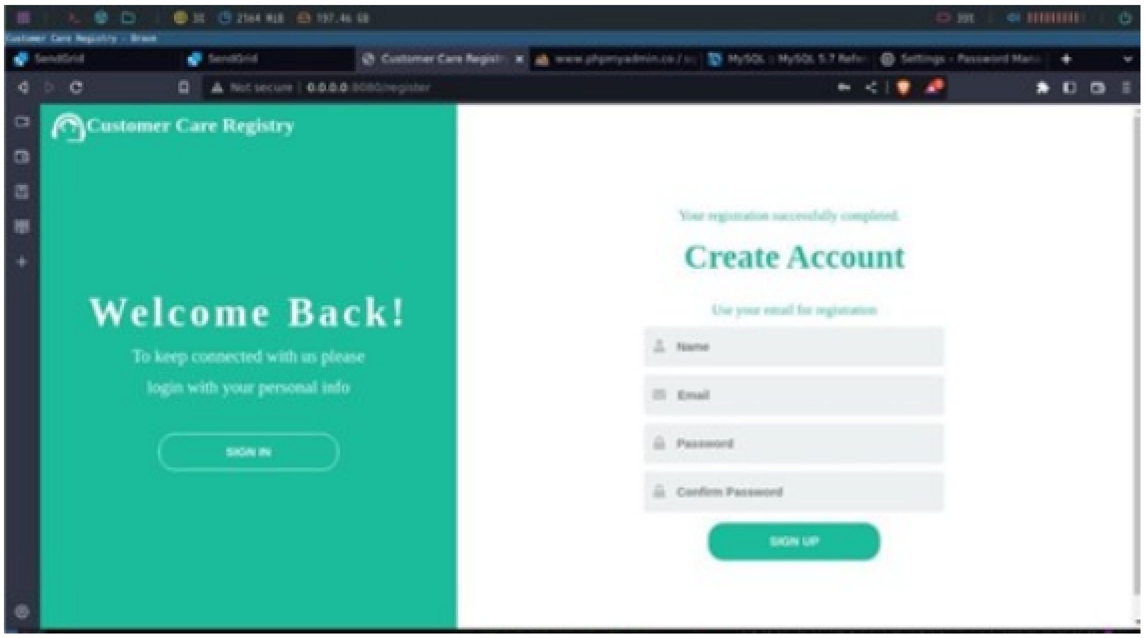
```
from flask import Blueprint, render_template, url_for, redirect
from flask_login import login_required, logout_user
from .views import conn, mail
import ibm_db
from .cust import QUERY_STATUS_OPEN

USER_ADMIN = "ADMIN"

admin = Blueprint("admin", __name__)

# query to get all the confirmed agents
get_confirmed_agents = '''
    SELECT first_name, agent_id FROM agent WHERE confirmed = ?
'''

@admin.route('/admin/tickets')
@login_required
def tickets():
    '''
        Loading all the OPEN tickets from the database
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # Query to get all the unassigned tickets raised by all the users
        get_unassigned_tickets = '''
            SELECT
                ticket_id,
                raised_on,
                customer.first_name,
                tickets.issue,
                customer.email
            FROM
                tickets
            JOIN
                customer ON tickets.raised_by = customer.cust_id
            AND
                tickets.assigned_to IS NULL
            ORDER BY
                raised_on ASC
        '''

        try:
            # getting the confirmed agents first
            stm = ibm_db.prepare(conn, get_confirmed_agents)
            ibm_db.bind_param(stm, 1, True)
            ibm_db.execute(stm)

            agents = ibm_db.fetch_assoc(stm)
            agents_list = []

            while(agents != False):
                temp = []

                temp.append(agents['FIRST_NAME'])
                temp.append(agents['AGENT_ID'])

                agents_list.append(temp)
                print(temp)

                agents = ibm_db.fetch_assoc(stm)

            # Getting the unassigned tickets
            stmt = ibm_db.prepare(conn, get_unassigned_tickets)
            ibm_db.execute(stmt)

            tickets = ibm_db.fetch_assoc(stmt)
            tickets_list = []

            if tickets:
                # means there are still some unassigned tickets
                while tickets != False:
                    temp = []

                    temp.append(tickets['TICKET_ID'])
                    temp.append(str(tickets['RAISED_ON'])[0:10])
```

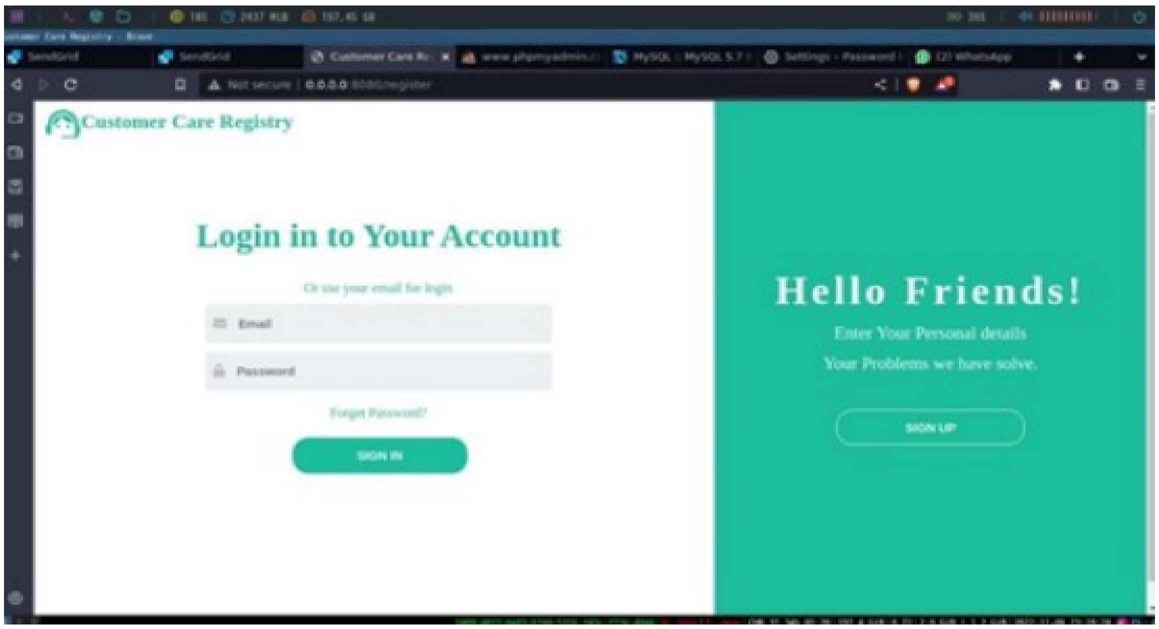
```
                temp.append(tickets['FIRST_NAME'])
                temp.append(tickets['ISSUE'])
                temp.append(tickets['EMAIL'])

                print(temp)

                tickets_list.append(temp)

                tickets = ibm_db.fetch_assoc(stmt)

            return render_template(
                'admin tickets.html',
                id = 0,
                tickets_to_show = True,
                tickets = tickets_list,
                msg = "These are the unassigned tickets",
                agents = agents_list,
                user = USER_ADMIN
            )

        else:
            # all the tickets may be assigned
            # may be, there are no tickets raised in the system at all
            return render_template(
                'admin tickets.html',
                id = 0,
                tickets_to_show = False,
                msg = "There is nothing to assign!",
                user = USER_ADMIN
            )

    except:
        # something fishy happened while getting the tickets
        # so alerting the admin
        return render_template(
            'admin tickets.html',
            id = 0,
            to_show = True,
            message = "Something wrong! Please TrY Again",
            user = USER_ADMIN
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/agents')
@login_required
def agents():
    '''
        Returning all the confirmed agents from the database
    '''

    from .views import admin

    if(hasattr(admin, 'email')):
        # query to get all the confirmed agents
        get_confirmed = '''
            SELECT * FROM agent WHERE confirmed = ?
        '''

        try:
            stmt = ibm_db.prepare(conn, get_confirmed)
            ibm_db.bind_param(stmt, 1, True)
            ibm_db.execute(stmt)

            agents = ibm_db.fetch_assoc(stmt)
            agents_list = []

            if agents:
                # there are some confirmed agents
                while agents != False:
                    temp = []

                    temp.append(agents['AGENT_ID'])
                    temp.append(str(agents['DATE_JOINED'])[0:10])
                    temp.append(agents['FIRST_NAME'])
                    temp.append(agents['LAST_NAME'])
                    temp.append(agents['EMAIL'])
                    agents_list.append(temp)

                    agents = ibm_db.fetch_assoc(stmt)

                return render_template(
                    'admin agents.html',
                    id = 1,
                    msg = "List of confirmed agents",
                    agents_to_show = True,
                    agents = agents_list,
                    user = USER_ADMIN
                )

            else:
                # no confirmed agents present
                return render_template(
                    'admin agents.html',
                    id = 1,
                    msg = "No agents present",
                    agents_to_show = False,
                    user = USER_ADMIN
                )

        except:
            # something happened while fetching the agents
            return render_template(
                'admin agents.html',
                id = 1,
                mmessage = "Something happened! Please try again",
                to_show = True,
                user = USER_ADMIN
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/accept')
@login_required
def accept():
    '''
        Loading the agents info from the database who are not yet
confirmed
    '''

    from .views import admin

    if(hasattr(admin, 'email')):
        # query to get all the agents from the database who are all not
confirmed yet
        get_agents_query = '''
            SELECT * FROM agent WHERE confirmed = ?
        '''

        agents_to_show = False
        msg = ""

        try:
            stmt = ibm_db.prepare(conn, get_agents_query)
            ibm_db.bind_param(stmt, 1, False)
            ibm_db.execute(stmt)

            agents = ibm_db.fetch_assoc(stmt)

            agents_list = []

            while agents != False:
                temp = []

                temp.append(agents['AGENT_ID'])
                temp.append(agents['EMAIL'])
                temp.append(agents['FIRST_NAME'])
                temp.append(agents['DATE_JOINED'])

                agents_list.append(temp)

                agents = ibm_db.fetch_assoc(stmt)

            if len(agents_list) >= 1:
                # there are some agents who are not yet confirmed
                msg = "These are the pending requests"
                agents_to_show = True

            else:
                agents_to_show = False
                msg = "There are no pending requests"

            return render_template(
                'admin acc agent.html',
                id = 2,
                agents = agents_list,
                agents_to_show = agents_to_show,
                msg = msg,
                user = USER_ADMIN
            )

        except:
            # something happened while admin either accepts/denies the
agent
            return render_template(
                'admin acc agent.html',
                to_show = True,
                message = "Something went wrong!",
                id = 2,
                user = USER_ADMIN
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/about')
@login_required
def about():
    '''
        Showing the about of the application to the admin
    '''

    from .views import admin
    if(hasattr(admin, 'email')):
        return render_template(
            'admin about.html',
            id = 3,
            user = USER_ADMIN
        )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/support')
@login_required
def support():
    '''
        Showing all the feedbacks given by the agents and customers
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        # query to retrieve all the feedbacks submitted
        get_feedbacks_query = '''
            SELECT * FROM feedback ORDER BY RAISED_ON DESC
        '''

        try:
            stmt = ibm_db.prepare(conn, get_feedbacks_query)
            ibm_db.execute(stmt)
            feedbacks = ibm_db.fetch_assoc(stmt)
            feedbacks_list = []

            feedbacks_to_show = False

            while feedbacks != False:
                temp = []

                temp.append(str(feedbacks['RAISED_ON'])[0:10])
                temp.append(feedbacks['RAISED_BY'])
                temp.append(feedbacks['RAISED_NAME'])
                temp.append(feedbacks['FEED'])
                feedbacks_list.append(temp)

                feedbacks = ibm_db.fetch_assoc(stmt)

            if len(feedbacks_list) > 0:
                # some feedbacks are submitted
                msg = 'All the feedbacks from the customers and agents'
                feedbacks_to_show = True

            else:
                # no feedbacks submitted yet
                msg = 'No feedbacks yet!'

            return render_template('admin support.html',
                id = 4,
                user = USER_ADMIN,
                feedbacks = feedbacks_list,
                msg = msg,
                true = feedbacks_to_show
            )

        except:
            # something happened while fetching the feedbacks
            return render_template('admin support.html',
                id = 4,
                user = USER_ADMIN,
                to_show = True,
                message = 'Something went wrong! Please try again'
            )

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

@admin.route('/admin/<email>/<action>')
@login_required
def alter(email, action):
    '''
        Either accepting or denying the agent, as per the admin's decision
    '''
    from .views import admin

    if(hasattr(admin, 'email')):
        if action == "True":
            # admin chose to the accept the agent
            accept_query = '''
                UPDATE agent SET confirmed = ? WHERE email = ?
            '''

            stmt = ibm_db.prepare(conn, accept_query)
            ibm_db.bind_param(stmt, 1, True)
            ibm_db.bind_param(stmt, 2, email)

            ibm_db.execute(stmt)

        else:
            # admin must have chosen to delete the agent
            delete_query = '''
                DELETE FROM agent WHERE email = ?
            '''

            stmt = ibm_db.prepare(conn, delete_query)
            ibm_db.bind_param(stmt, 1, email)

            ibm_db.execute(stmt)

        return "None"

    else:
        # logging out
        return redirect(url_for('blue_print.logout'))

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

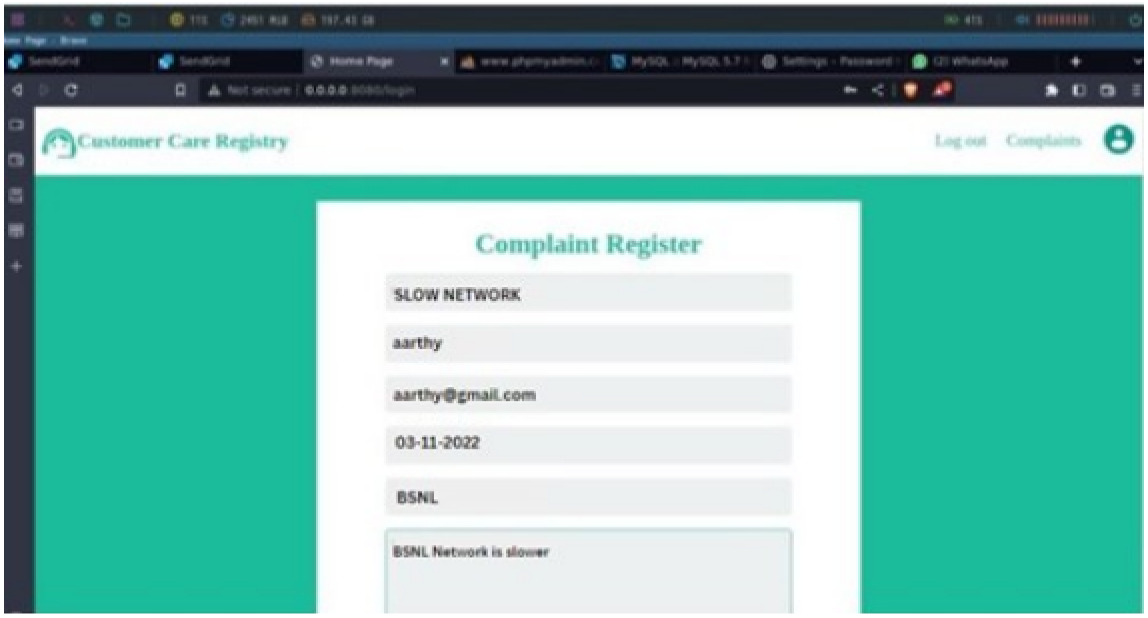
```
if(hasattr(admin, 'email')):
    # query to update the ASSIGNED_TO of a ticket
    assign_agent_query = '''
        UPDATE tickets SET assigned_to = ? WHERE ticket_id = ?
    '''

    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)

    # sending the acknowledgement mail to the customer
    mail.sendEmail(
        f'Customer Care Registry',
        f'''
            An agent has been assigned for your ticket <br/>
            <strong>Ticket ID : {ticket_id}</strong> <br>
            <br>
            Please login into CCR for more details! <br/>
            Thank you!!!
        ''',
        [f'{cust_email}']
    )

    return "None"

else:
    # logging out
    return redirect(url_for('blue_print.logout'))
```

OUTPUT:

FEATURE 1


Customer Care Registry
Welcome Back!
To keep connected with us please login with your personal info
SIGN IN
Your registration successfully completed.
Create Account
Use your email for registration
Name
Email
Password
Confirm Password
SIGN UP




Customer Care Registry

Log out Complaints

Complaint Register

SLOW NETWORK

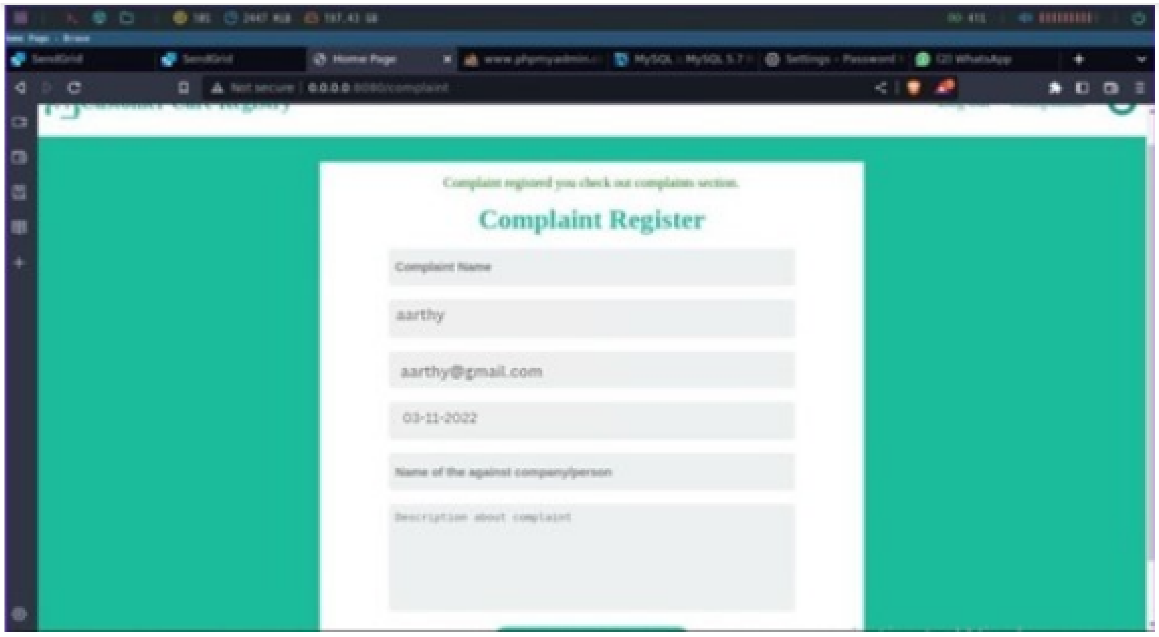
aarthy

aarthy@gmail.com

03-11-2022

BSNL

BSNL Network is slower


Complaint registerd you check out complaints section.

Complaint Register

Complaint Name

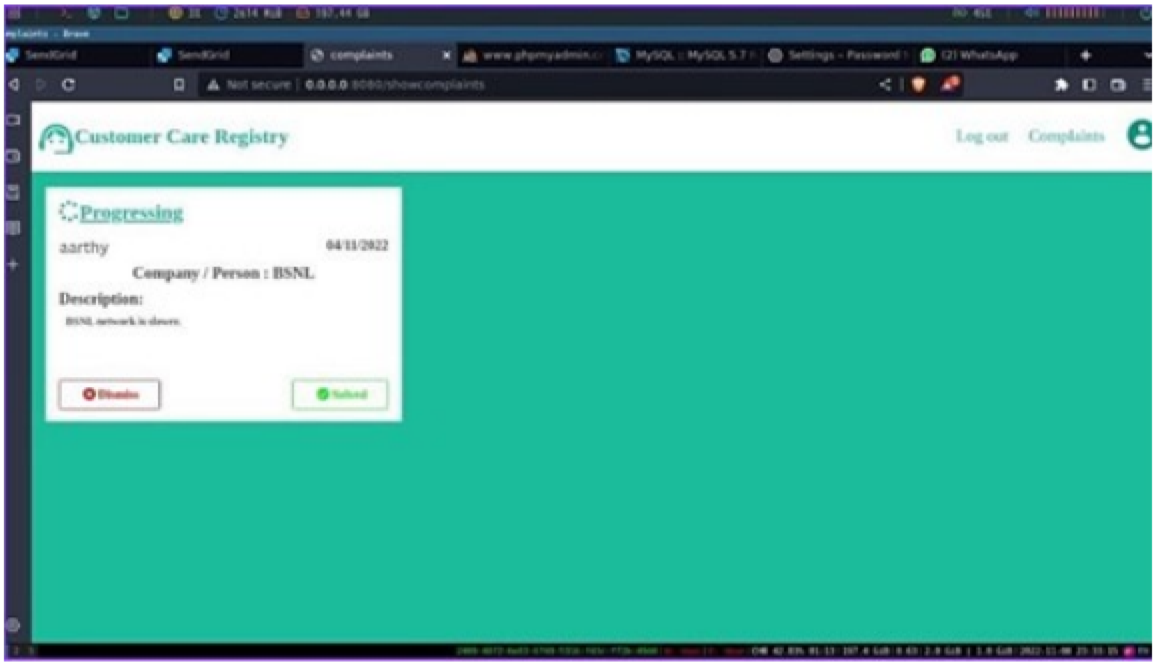
aarthy

aarthy@gmail.com

03-11-2022

Name of the against company/person

Description about complaint



Progressing

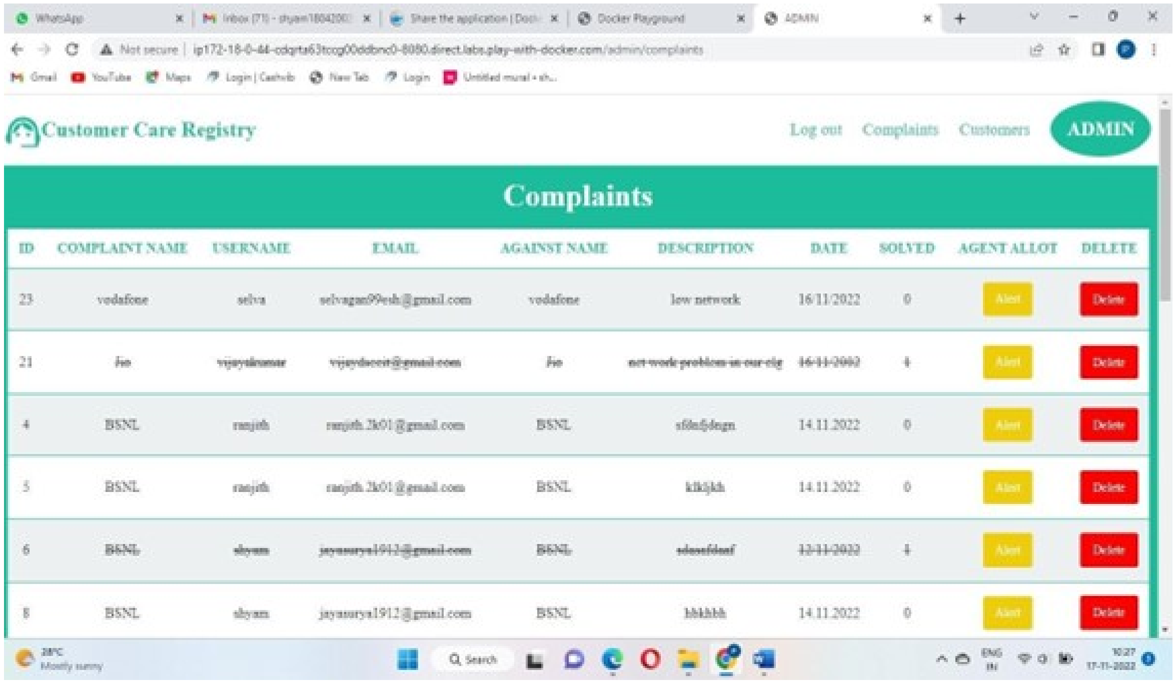
aarthy 04/11/2022

Company / Person : BSNL

Description:

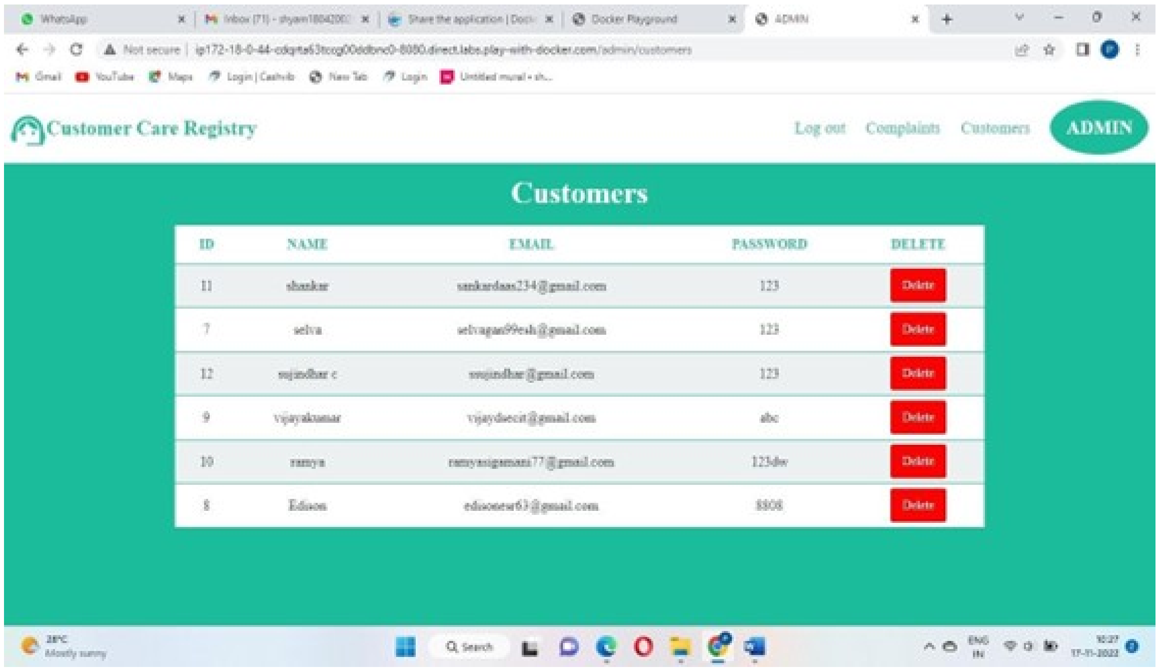
BSNL network is down.

FEATURE 2



Complaints

ID	COMPLAINT NAME	USERNAME	EMAIL	AGAINST NAME	DESCRIPTION	DATE	SOLVED	AGENT ALLOT	DELETE
23	vodafone	selva	selvagan99esh@gmail.com	vodafone	low network	16/11/2022	0	Alert	Delete
21	~~Jio~~	~~vijaykumar~~	~~vijaydsceit@gmail.com~~	~~Jio~~	~~net work problem in our clg~~	~~16-11-2002~~	~~1~~	Alert	Delete
4	BSNL	ranjith	ranjith.2k01@gmail.com	BSNL	sfdnfjdngn	14.11.2022	0	Alert	Delete
5	BSNL	ranjith	ranjith.2k01@gmail.com	BSNL	klkljkh	14.11.2022	0	Alert	Delete
6	~~BSNL~~	~~shyam~~	~~jayasurya1912@gmail.com~~	~~BSNL~~	~~sdasafdasf~~	~~12-11-2022~~	~~1~~	Alert	Delete
8	BSNL	shyam	jayasurya1912@gmail.com	BSNL	hhkhhh	14.11.2022	0	Alert	Delete

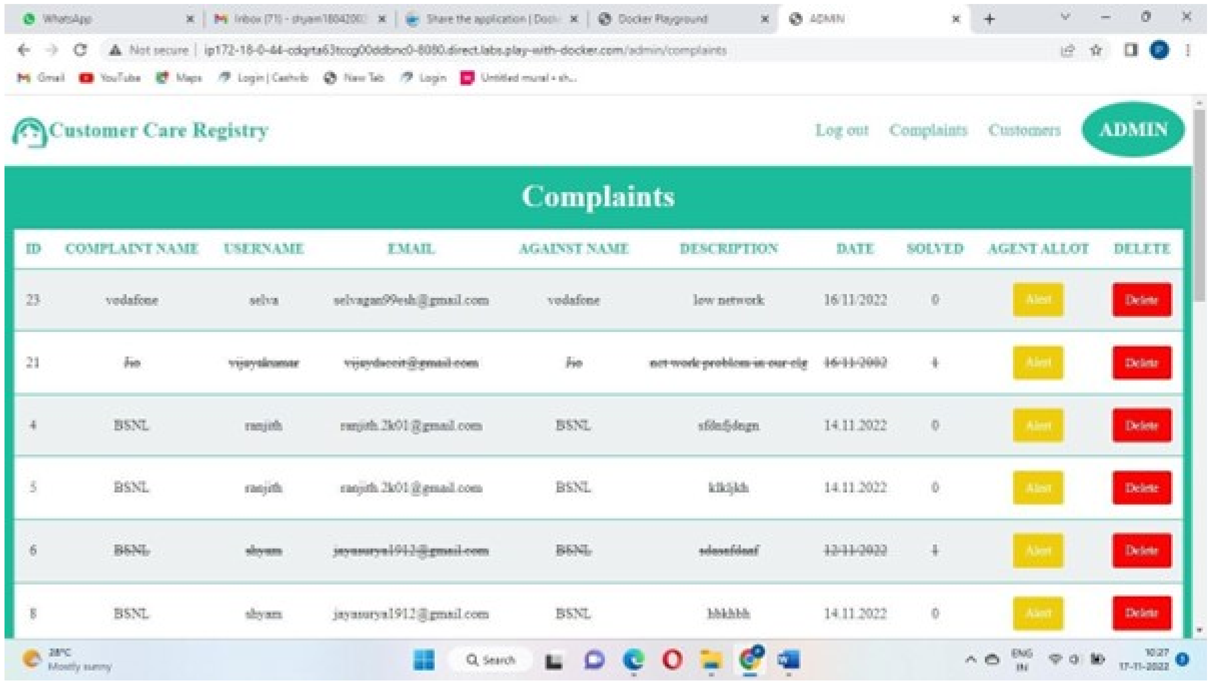


Customer Care Registry

Log out Complaints Customers ADMIN

Customers

ID	NAME	EMAIL	PASSWORD	DELETE
11	shankar	sankardaas234@gmail.com	123	Delete
7	selva	selvagan99esh@gmail.com	123	Delete
12	sujindhar c	ssujindhar@gmail.com	123	Delete
9	vijayakumar	vijaydsecit@gmail.com	abc	Delete
10	ramya	ramyasigamani77@gmail.com	123dw	Delete
8	Edison	edisonesr63@gmail.com	8808	Delete



Customer Care Registry

Log out Complaints Customers ADMIN

Complaints

ID	COMPLAINT NAME	USERNAME	EMAIL	AGAINST NAME	DESCRIPTION	DATE	SOLVED	AGENT ALLOT	DELETE
23	vodafone	selva	selvagan99esh@gmail.com	vodafone	low network	16/11/2022	0	Alert	Delete
21	~~Jio~~	~~vijayakumar~~	~~vijaydsecit@gmail.com~~	~~Jio~~	~~net work problem in our cig~~	~~16-11-2002~~	~~1~~	Alert	Delete
4	BSNL	ranjith	ranjith.2k01@gmail.com	BSNL	sfdnfjdngn	14.11.2022	0	Alert	Delete
5	BSNL	ranjith	ranjith.2k01@gmail.com	BSNL	klkljkh	14.11.2022	0	Alert	Delete
6	~~BSNL~~	~~shyam~~	~~jayasurya1912@gmail.com~~	~~BSNL~~	~~sdasafdasf~~	~~12-11-2022~~	~~1~~	Alert	Delete
8	BSNL	shyam	jayasurya1912@gmail.com	BSNL	hhkhhh	14.11.2022	0	Alert	Delete

Github Link:

https://github.com/IBM-EPBL/IBM-Project-31018-1660194538