CLOUD APPLICATION DEVELOPMENT

CUSTOMER CARE REGISTRY

A MINI-PROJECT REPORT

Submitted by

ARUN KUMAR S - AC19UCS008

BARATH KUMAR S - AC19UCS013

BALAJI S - AC19UCS012

DEEPAK K - AC19UCS021

In partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

ADHIYAMAAN COLLEGE OF ENGINEERING

DR.M.G.R NAGAR, HOSUR-635130

ANNA UNIVERSITY: CHENNAI 600025

NOVEMBER 2022

ANNA UNIVERSITY: CHENNAI 600 025

NOVEMBER 2022

BONAFIDE CERTIFICATE

Certified that this mini-project report **“CUSTOMER CARE REGISTRY”** is the bonafide work of **“ARUN KUMAR S (AC19UCS008), BALAJI S (AC19UCS012), BARATH KUMAR S (AC19UCS013), DEEPAK K (AC19UCS021)”** who carried out the project under my supervision.

SIGNATURE

Dr. G. FATHIMA, M.E., Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR,

Department of CSE,

Adhiyamaan College of Engineering,

(Autonomous)

Dr. M.G.R. Nagar,

Hosur - 635 130

SIGNATURE

Dr. B. GOPINATHAN, M.E., Ph.D.,

SUPERVISOR

PROFESSOR,

Department of CSE,

Adhiyamaan College of Engineering,

(Autonomous)

Dr. M.G.R. Nagar,

Hosur – 635 130

Submitted for the Mini project VIVA-VOICE Examination held on ……………………. at **Adhiyamaan College of Engineering (Autonomous), Hosur – 635 130**.

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

It is one of the most efficient tasks in life to choose the appropriate words to express one’s gratitude to the beneficiaries. We are very much grateful to God who helped us through the project and molded us into what we are today.

We are grateful to our beloved Principal **Dr. G. RANGANATH, M.E., Ph.D.,** Adhiyamaan College of Engineering (Autonomous), Hosur for providing the opportunity to do this work on the premises.

We acknowledge our heartful gratitude to **Dr. G. FATHIMA, M.E., Ph.D.,** Professor, and Head of the Department, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (Autonomous), Hosur, for her guidance and valuable suggestions and encouragement throughout this project.

We are highly indebted to **Dr. B. GOPINATHAN, M.E., Ph.D.,** Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (Autonomous), Hosur, whose immense support, encouragement, and valuable guidance made us complete this project successfully.

We also extend our thanks to the Project Coordinator and all Staff Members for their support to complete this project successfully.

Finally, we could like to thank our parents, without their motivation and support would not have been possible for us to complete this project successfully.

CUSTOMER CARE REGISTRY

CONTENTS

1.INTRODUCTION

1.1.PROJECT OVERVIEW:

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

Admin: The main role and responsibility of the admin are to take care of the whole process. Starting from Admin login followed by the agent creation and assigning the customer's complaints. Finally, He will be able to track the work assigned to the agent and a notification will be sent to the customer.

User: They can register for an account. After the login, they can create a complaint with a description of the problem they are facing. Each user will be assigned an agent. They can view the status of their complaint.

Customer care describes how people are treated when they interact with a brand. Customer service is an essential part of any service company. This includes all experiences with the company and its employees before, during, and after a purchase. Customer care is an important aspect of customer service because it fosters an emotional connection with the brand's community. Good customer service and support will assist the company in retaining and retaining customers. Customers are becoming accustomed to using applications as technology, the internet, and applications advance. Customer care isn't measured similarly to customer loyalty or success. That's because things like loyalty and success are a by-product of caring for your customers. Building a trustworthy, emotional connection with your customer base is impossible if you're too focused on measuring it. It is critical for service companies to participate in the development of a good customer care system. Good service will improve service quality, which will impact customer loyalty. Customer care goes a step further by ignoring the metrics and instead fully investing in your customers' goals and needs. Customer care is the process of building an emotional connection with your customers, whereas customer service is simply the advice or assistance your business provides them. Customer care is less quantifiable than customer service and is more concerned with one-to-one customer interactions. The Customer care registry supports the customer service system in terms of customer satisfaction, complaints, and self-solving problems. While both functions increase customer satisfaction, customer service does this by answering questions and providing support. Customer care, on the other hand, focuses on active listening and understanding the customer's emotional needs as much as the physical or business ones.

1.2.PROJECT PURPOSE

The customer service team is the face of the organization and the frontline when customers require assistance. Customer service agents help customers pay bills, review or make changes to accounts, handle returns, and answer frequently asked questions.

An online comprehensive customer care solution is to manage customer interaction and complaints with the service providers over the phone or through email. The system should have the capability to integrate with any service provider from any domain or industry like Banking, Telecom, and social media.

2.LITERATURE SURVEY

2.1.EXISTING PROBLEM:

Customers today expect communication with service departments to be instant. In fact, they want immediate resolution of their concerns too. This is, indisputably, the first common problem with customer service that needs to be addressed by businesses. You may not want to be in a position where you have to listen to customers complaining. Unless you give your full attention to what the customer is saying, it will be difficult to understand what they need or how to service their problem. When a customer keeps getting transferred from one agent or department to another, it ensures that a customer will never return to you or your business in the future. Neither will they recommend you to people they know. This brings us to the second most common customer service problem. No matter how frustrated or high-pitched a customer might be at the time of conversing with a service agent, it does not give the rep the license to be rude to the customer in any way. Generally, such situations are handled by an experienced manager. Customer service reps are only human and may not be able to offer a resolution of customer queries on the first contact. When customers have to chat or call the service department multiple times, it can be a hassle for them. Customers today want to talk to humans, not machines. This brings us to another key customer service issue that is quite common these days. Many times, customer service agents adopt a one size fits all kind of approach. This may result in them pushing a product or service to the customer, thus, adversely impacting their experience with the business. This brings us to the next customer service problem of reps not following through with the promise that they have made to the customer. It can be infuriating when the issue remains unsolved due to this. This brings us to the last problem with customer service, where businesses are not paying adequate attention to getting their customer service workflow in line with the customer’s lifecycle.

2.2.REFERENCES:

Leung, Nelson K. Y. & Lau, Sim. (2007). Information technology help desk survey: To identify the classification of simple and routine inquiries. Journal of Computer Information Systems. 47. 70-81.

Rachmawati, Eka, M. Kom, and M. Kom. "Web-Based Ticketing System Helpdesk Application Using CodeIgniter Framework (Case Study: PT Commonwealth Life)." *International Journal of Computer Science and Mobile Computing* 7.12 (2018):29-41.

Sadewa, Baghaskara, S. Suhendra, and M. Kom. "Complaint Handling Ticketing Application Web Based Using Codeigniter Framework (Case Study at PT Indosat Ooredoo Tbk Jakarta)." (2018).

C. Cassandra, S. Hartono and M. Karsen, "Online Helpdesk Support System for Handling Complaints and Service," *2019 International Conference on Information Management and Technology (ICIMTech)*, 2019, pp. 314-319, doi:10.1109/ICIMTech.2019.8843726.

Sorsa, Jani. "Helpdesk ticketing system for a small-sized company-selection and deployment.“ (2021).

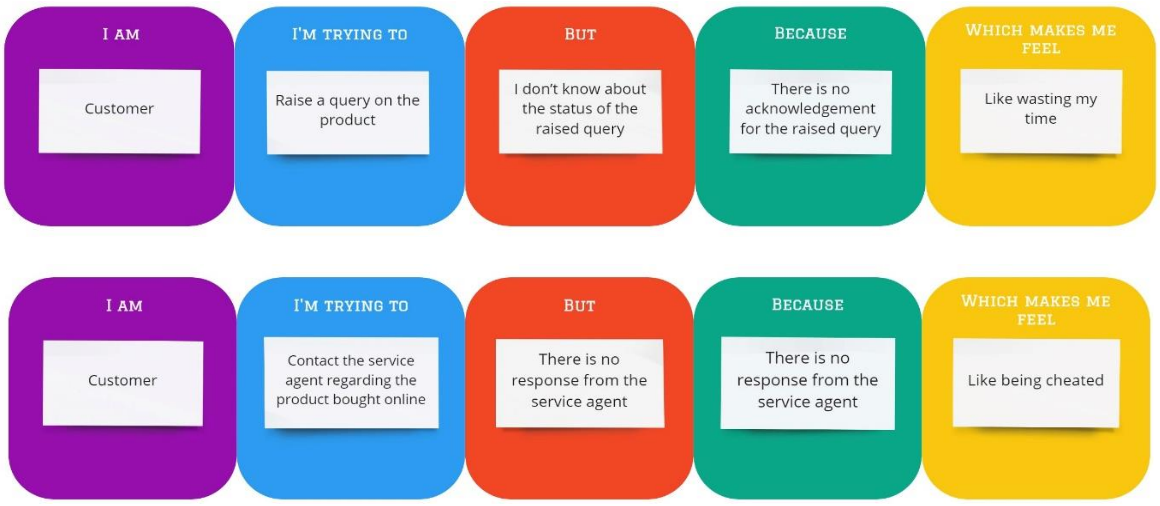
Marcella, R. and Middleton, I., "The Role of the Help Desk in the Strategic Management of Information Systems," OCLC Systems and Services, (12:4), 1996, pp.4-19.

2.3.PROBLEM STATEMENT:

The customer care registry provides efficient support to customers in solving their problems or queries. 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 the customer they will be notified with an email alert. Customers can view the status of the ticket till the service is provided. When a customer can view the status of the ticket till the service is provided. When a customer has a simple query and does not like to spend much time researching or contacting customer service for it, the customer care registry helps them in providing answers to the frequently asked questions. When a customer buys things online, the customer care registry makes the customer feel comfortable and provides satisfaction.

This Defines the Problem statement to understand the Customer's point of view. The Customer Problem Statement template helps us focus on what matters to create experiences people will love.

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



I AM	I'M TRYING TO	BUT	BECAUSE	WHICH MAKES ME FEEL
Customer	Raise a query on the product	I don't know about the status of the raised query	There is no acknowledgement for the raised query	Like wasting my time
Customer	Contact the service agent regarding the product bought online	There is no response from the service agent	There is no response from the service agent	Like being cheated

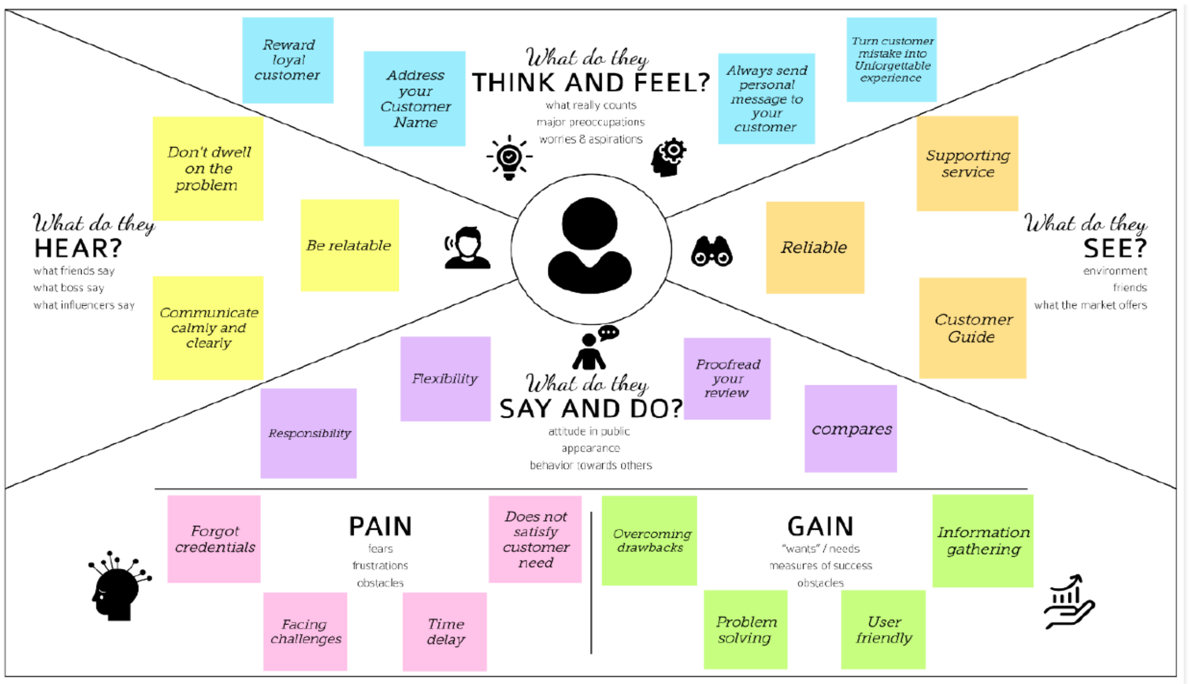
Problem Statement (PS)	I am	I'm trying to	But	Because	Which makes me feel
PS-1	Customer	Raise a query on the product	I don't know about the status of the raised query	There is no acknowledgment for the raised query	Like wasting my time
PS-2	Customer	Contact the service agent regarding the product bought online	There is no response from the service agent	There is no response from the service agent	Like being cheated

3.IDEATION & PROPOSED SOLUTION

3.1.Empathy Map Canvas:

An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviors and attitudes.

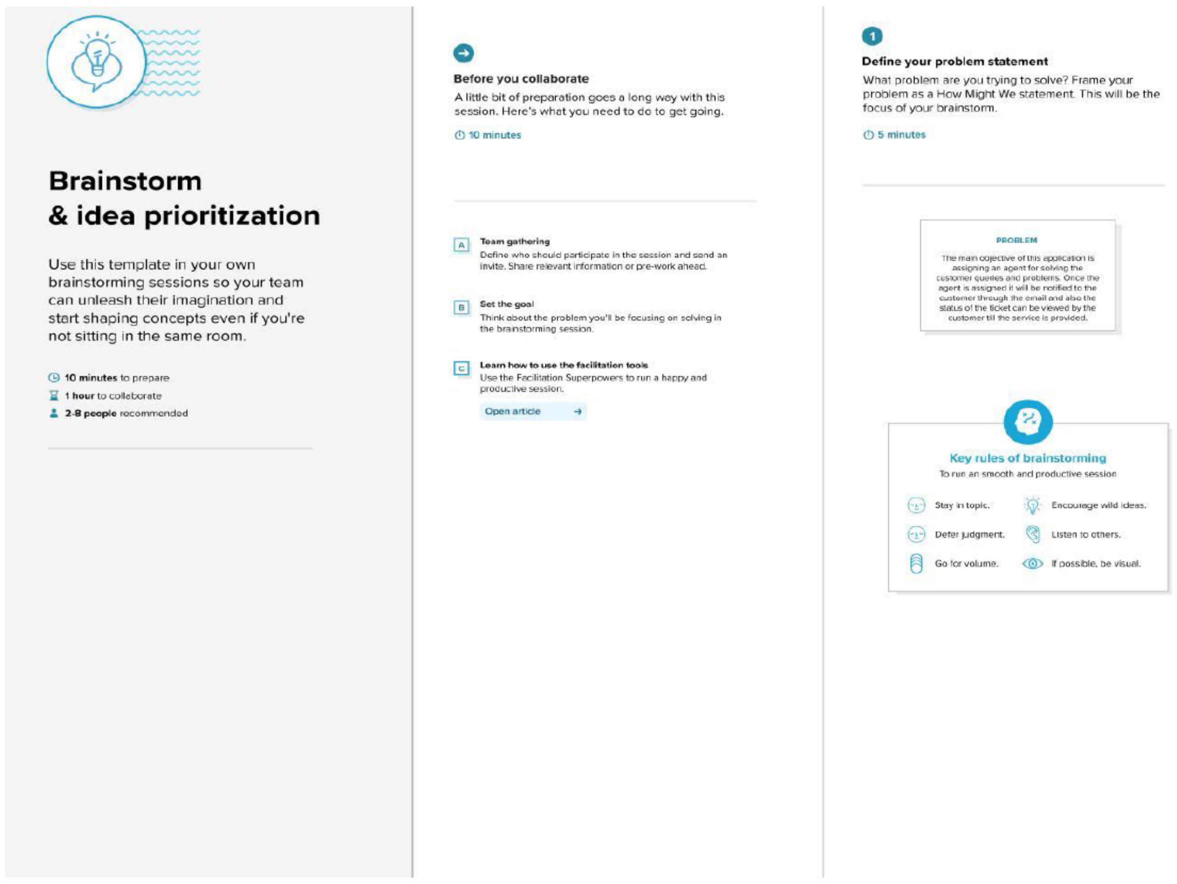
It is a useful tool to help teams better understand their users. Creating an effective solution requires understanding the true problem and the person who is experiencing it. The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.



3.2.Brainstorm & Idea Prioritization Template:

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

Step-1: Team Gathering, Collaboration and Select the Problem Statement



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

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

10 minutes

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

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

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

Open article →

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

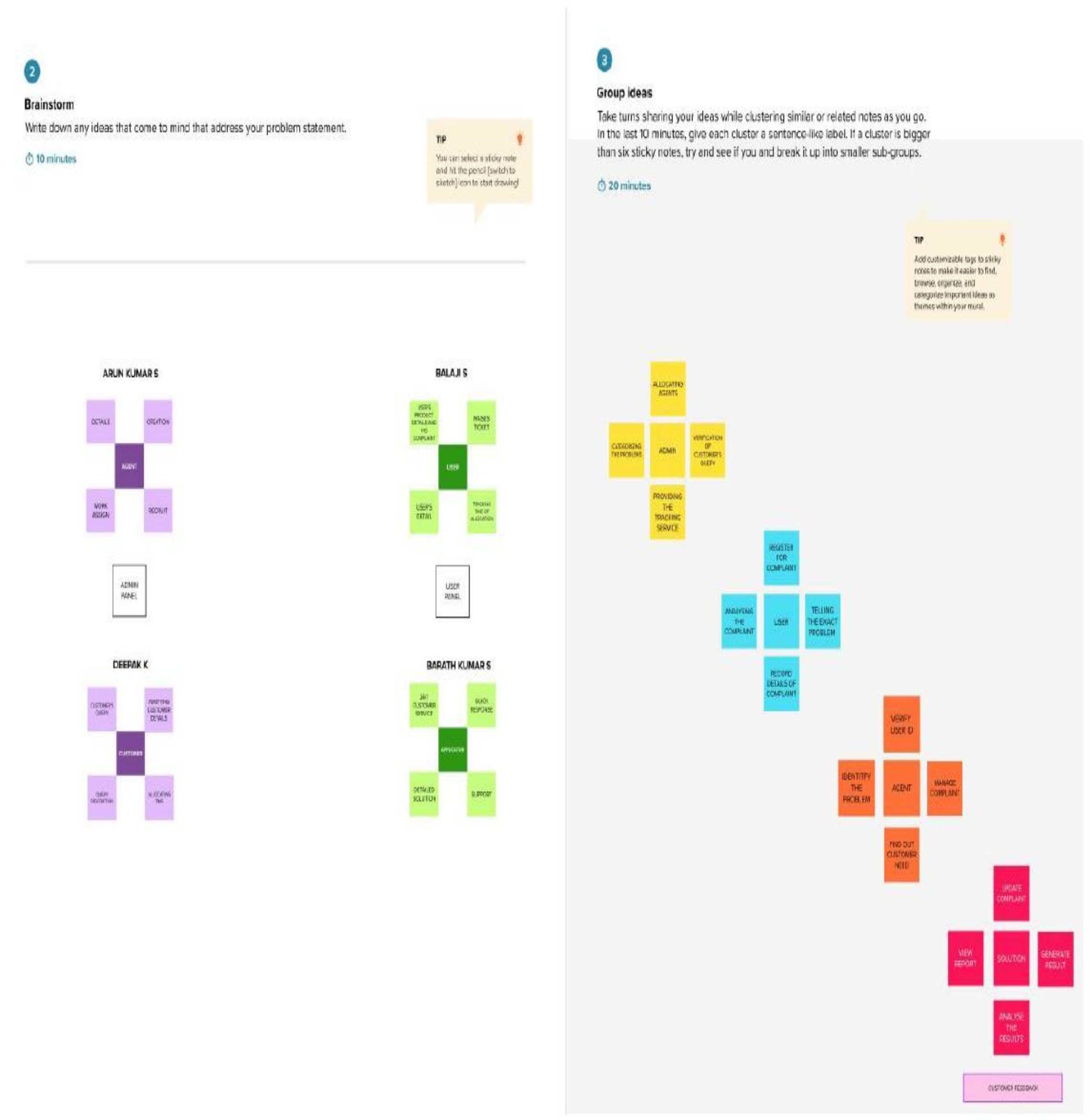
5 minutes

PROBLEM
The main objective of this application is assigning an agent for solving the customer queries and problems. Once the agent is assigned it will be notified to the customer through the email and also the status of the ticket can be viewed by the customer till the service is provided.

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.

Step 2: Brainstorm, Idea Listing, and Grouping



2

Brainstorm

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

10 minutes

TIP

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

3

Group ideas

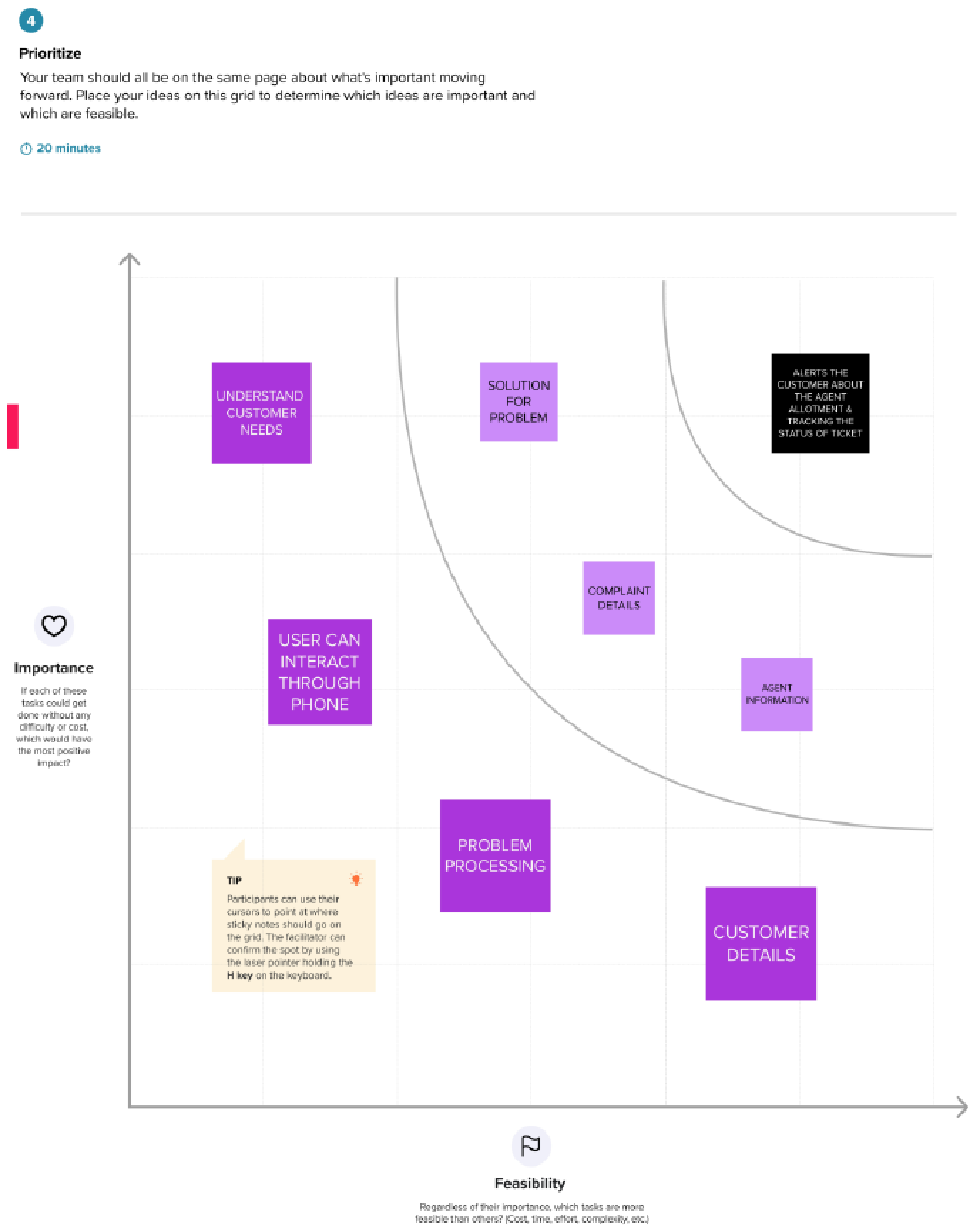
Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.

20 minutes

TIP

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

Step 3: Idea Prioritization



4

Prioritize

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

20 minutes

Importance

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

Feasibility

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

TIP

Participants can use their cursors to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H key** on the keyboard.

UNDERSTAND CUSTOMER NEEDS
SOLUTION FOR PROBLEM
ALERTS THE CUSTOMER ABOUT THE AGENT ALLOTMENT & TRACKING THE STATUS OF TICKET
COMPLAINT DETAILS
USER CAN INTERACT THROUGH PHONE
AGENT INFORMATION
PROBLEM PROCESSING
CUSTOMER DETAILS

3.3.Proposed Solution

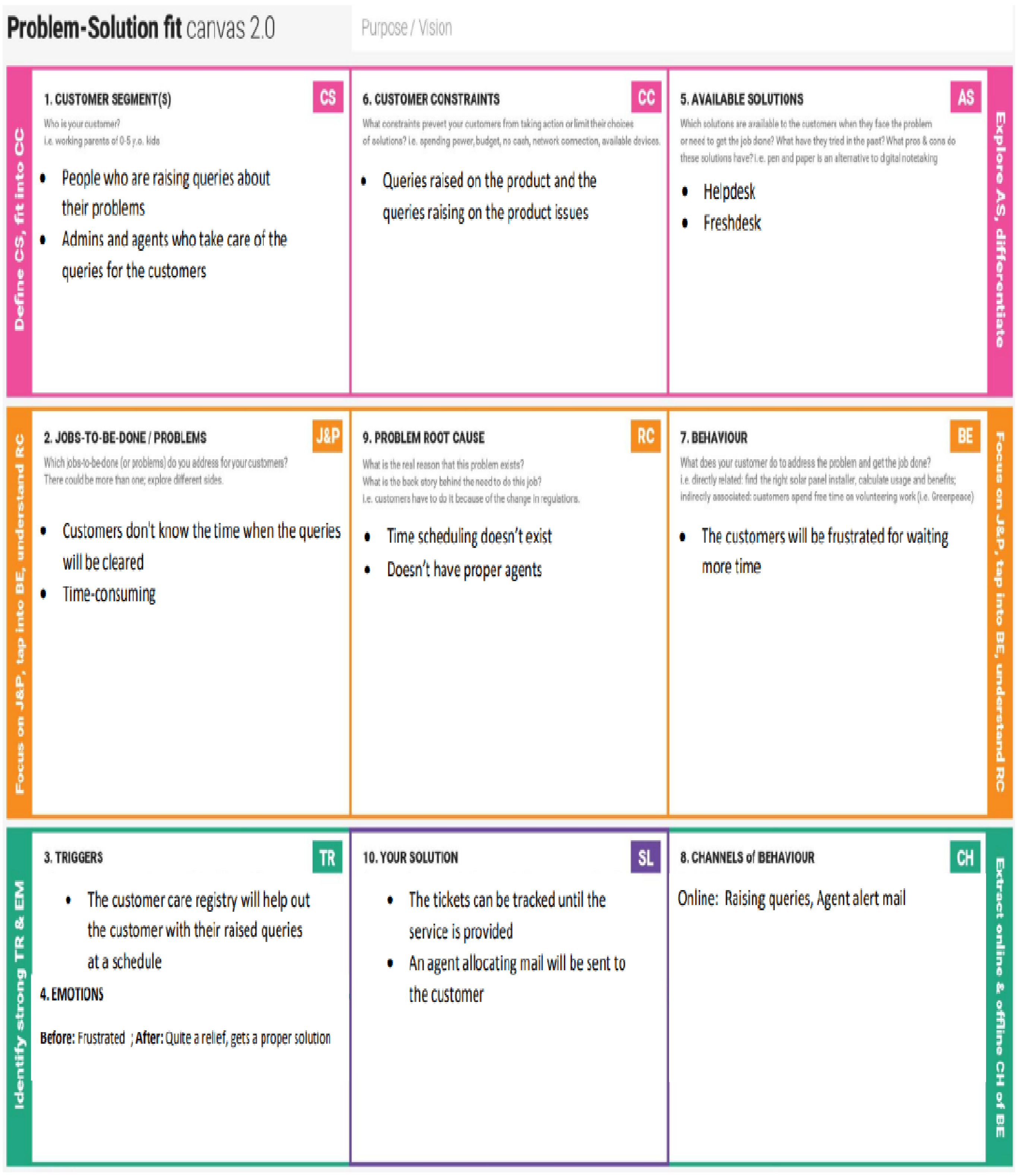
S.No	Parameter	Description
1	Problem statement (problem to be solved)	Instead of searching for different solutions on the internet, the customer can raise queries as tickets in this application.
2	Idea/Solution description	The customer needs to create a new account if they are a new user. The customer can raise the tickets to their problems with a detailed description of an issue. The customers can track their tickets and also an e-mail alert will be given to the customer once the agent is assigned.
3	Novelty/Uniqueness	The tracking method will keep updating you on the ticket processing and the agent details will be notified to the customers through an e-mail alert.
4	Social Impact / Customer Satisfaction	Our application can help the customer to track each step of their issue.
5	Business Model (financial benefit)	By providing this service to the companies for better customer support.
6	Scalability of Solution	We can integrate this application with business sites and e-commerce websites which solves issues faced by the customer.

3.4.Problem–Solution Fit Template:

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem. It helps entrepreneurs, marketers and corporate innovators identify behavioral patterns and recognize what would work and why.

Purpose:

- Solve complex problems in a way that fits the state of your customers.
- Succeed faster and increase your solution adoption by tapping into existing mediums and channels of behavior.
- Sharpen your communication and marketing strategy with the right triggers and messaging.
- Increase touch-points with your company by finding the right problem behavior fit and building trust by solving frequent annoyances, or urgent or costly problems.
- Understand the existing situation in order to improve it for your target group



Problem-Solution fit canvas 2.0

Purpose / Vision

Define CS, fit into CC

1. CUSTOMER SEGMENT(S) CS

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

- People who are raising queries about their problems
- Admins and agents who take care of the queries for the customers

6. CUSTOMER CONSTRAINTS CC

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

- Queries raised on the product and the queries raising on the product issues

5. AVAILABLE SOLUTIONS AS

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

- Helpdesk
- Freshdesk

Explore AS, differentiate

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

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

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

- Customers don't know the time when the queries will be cleared
- Time-consuming

9. PROBLEM ROOT CAUSE RC

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

- Time scheduling doesn't exist
- Doesn't have proper agents

7. BEHAVIOUR BE

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

- The customers will be frustrated for waiting more time

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

Identify strong TR & EM

3. TRIGGERS TR

- The customer care registry will help out the customer with their raised queries at a schedule

4. EMOTIONS

Before: Frustrated ; **After:** Quite a relief, gets a proper solution

10. YOUR SOLUTION SL

- The tickets can be tracked until the service is provided
- An agent allocating mail will be sent to the customer

8. CHANNELS of BEHAVIOUR CH

Online: Raising queries, Agent alert mail

Extract online & offline CH of BE

4.REQUIREMENT ANALYSIS

4.1.Functional requirement:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through a Google account
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Can log in through Google login with an E-mail ID and password
FR-4	Admin Login	Can log in through Google login with an E-mail ID and password
FR-5	Query table	A detailed description of an issue with the contact information
FR-6	Feedback	Customer’s thoughts about the service

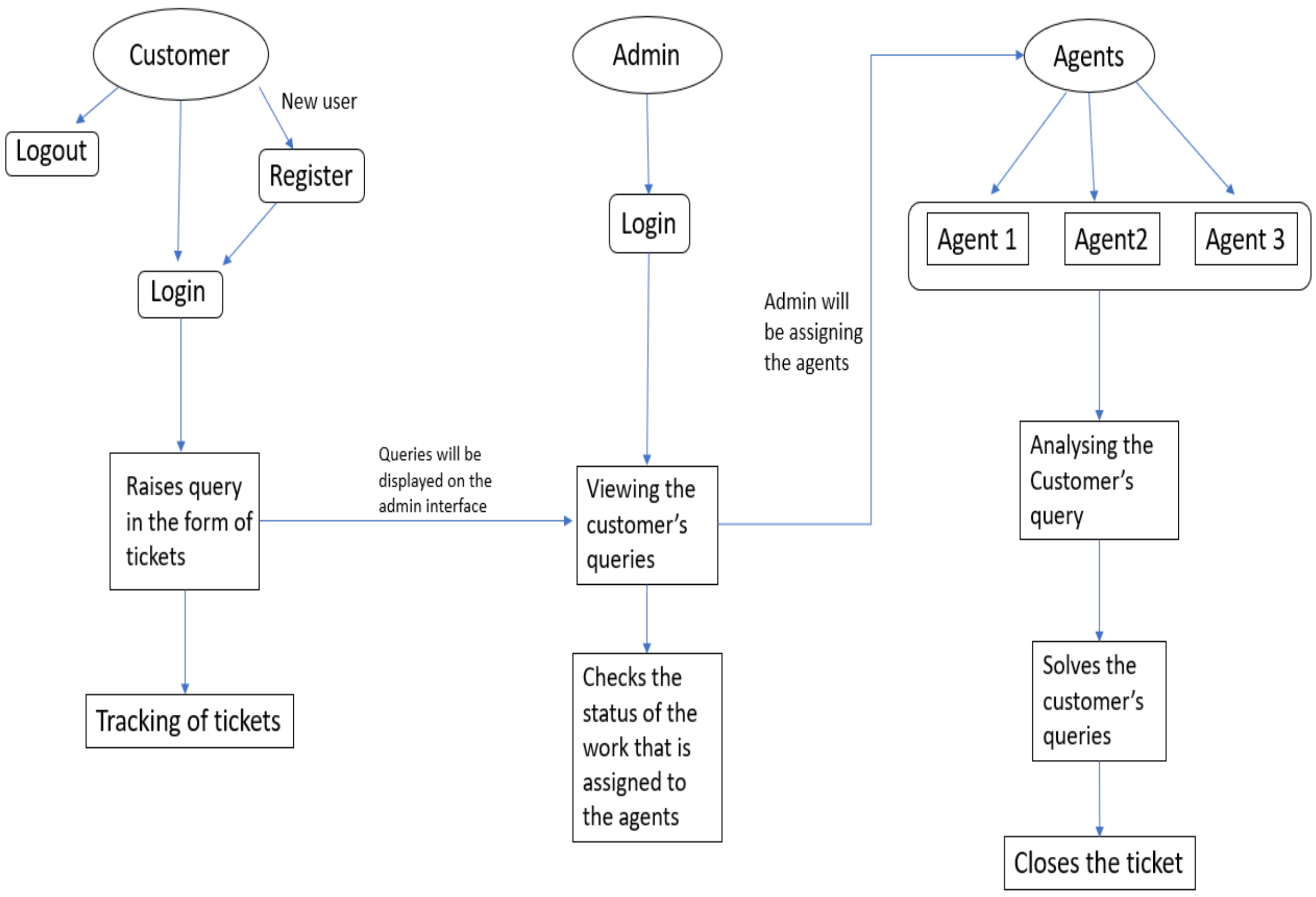
4.2.Non-functional Requirements:

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	Provides a solution to the problem
NFR-2	Security	E-mail alert while login to your account
NFR-3	Reliability	Tracking the status of the ticket
NFR-4	Performance	User-friendly and very responsive development of web application
NFR-5	Availability	Any-time service (24/7 service)
NFR-6	Scalability	Well knowledge and trained agents

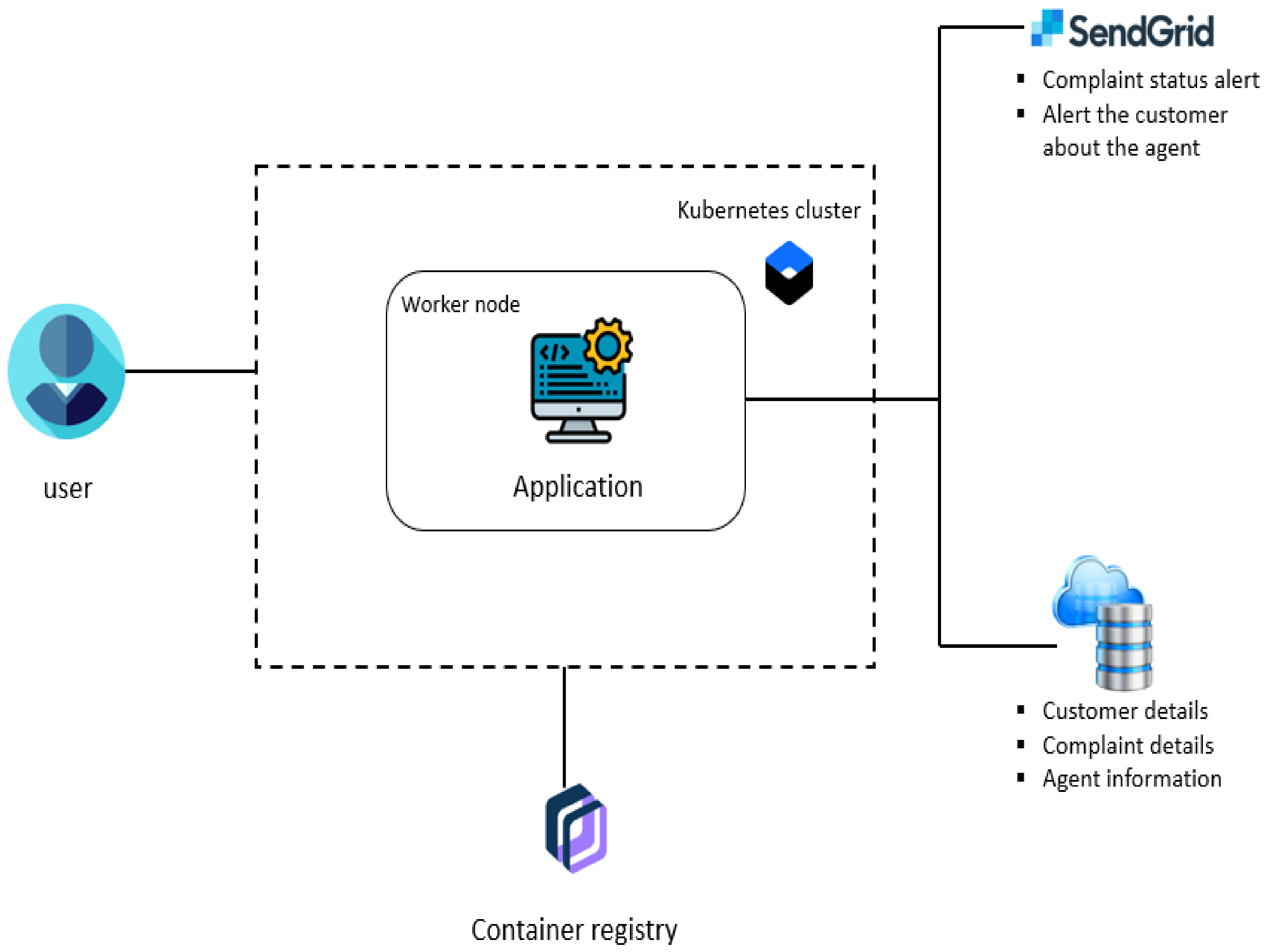
5.PROJECT DESIGN

5.1.Data Flow Diagram

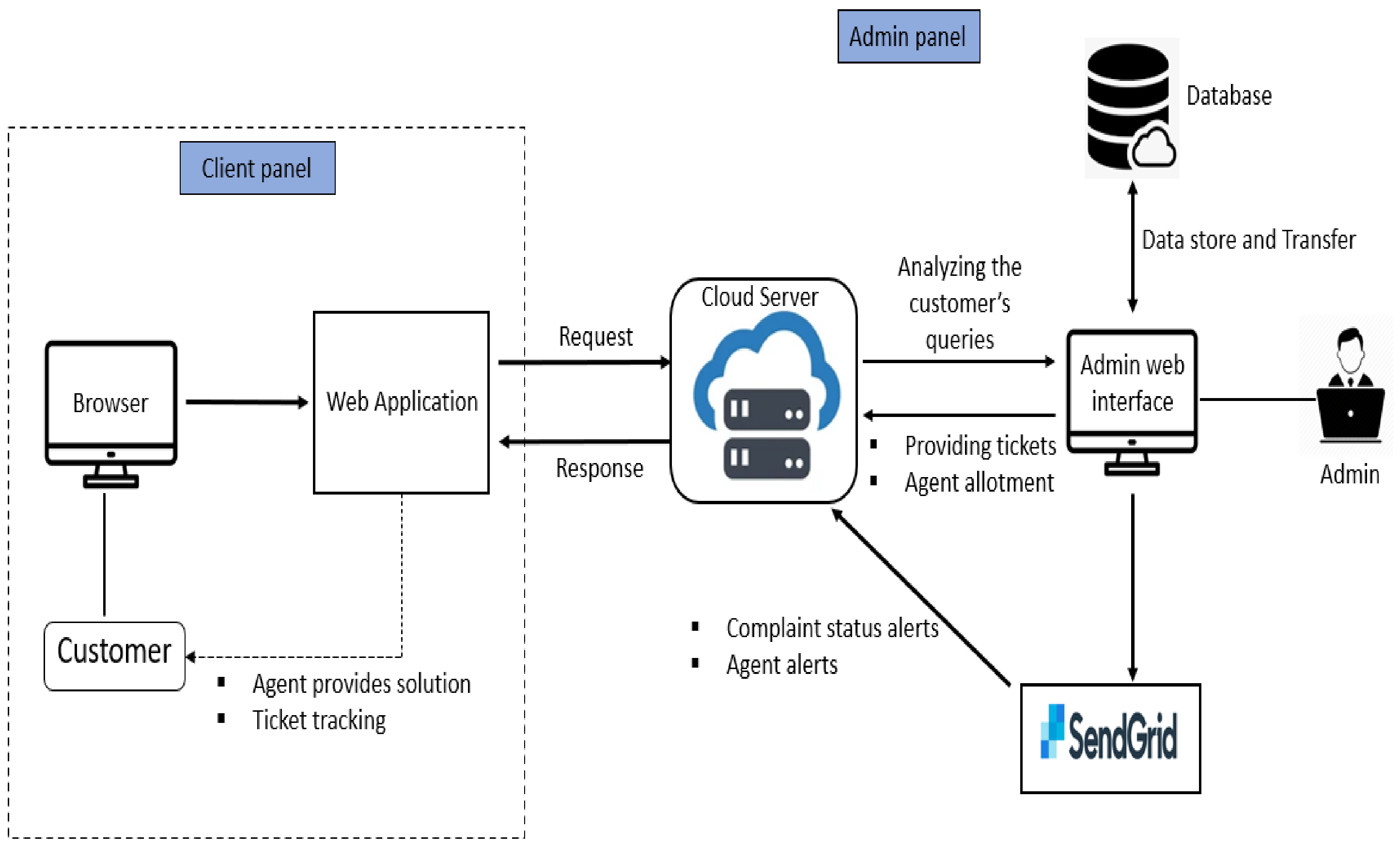
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can graphically depict the right amount of the system requirement. It shows how data enters and leaves the system, what changes the information, and where data is stored.


Customer
New user
Logout
Register
Login
Raises query in the form of tickets
Tracking of tickets
Queries will be displayed on the admin interface
Admin
Login
Viewing the customer's queries
Checks the status of the work that is assigned to the agents
Admin will be assigning the agents
Agents
Agent 1
Agent2
Agent 3
Analysing the Customer's query
Solves the customer's queries
Closes the ticket

5.2.SOLUTION ARCHITECTURE



TECHNICAL ARCHITECTURE



5.3.User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
User	Registration	USN-1	Users can register in the web application by entering their email, and password, and confirming their password.	Users can access their accounts and dashboard	High	Sprint-1
User	Email confirmation	USN-2	Users will receive a confirmation email once register the account	Users will receive a confirmation email & can click confirm	High	Sprint-1
User	Login	USN-3	Users can log into the application by entering their email & password	Users can log in to the website for further details	High	Sprint-1
User	Details	USN-4	Users can fill in their personal details.	Users can submit their personal details	High	Sprint-1
Agent	Login	USN-5	Agents can log into the application by entering their email & password	Agents can approach a particular customer	High	Sprint-2
Admin	Cloud Database	USN-6	Admin can store the user and agent details in the cloud database	Admins can store the data in the cloud database	High	Sprint-2
Admin	Cloud Database	USN-7	Stores the details	Admins can store the data in a cloud database	High	Sprint-3
Customer	Details	USN-8	Customers can send a request to the website for product/services issues	Customers can request product/services issues	High	Sprint-3
Admin	Assign task	USN-9	Admin can assign a task to the particular agent	He assigns the task to the agent	High	Sprint-3
Agent	Details	USN-10	The agent can take the customer details from the cloud database	He can take the data into a cloud database	High	Sprint-4
Agent	Details	USN-11	The agent will respond to the website for product/services issues	He can respond to the issues	High	Sprint-4
Customer	Email	USN-12	As a customer, I can receive the response	I can receive the response	High	Sprint-4

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	User Panel	USN-1	The user will log in to the website and go through the services available on the webpage	20	High	Barath Kumar S Balaji S Deepak K
Sprint 2	Admin Panel	USN-2	The role of the admin is to check out the database about the availability and have a track of all the things that the users are going to service	20	High	Arun Kumar S Barath Kumar S
Sprint-2	Agent panel	USN-2	As an agent, I can log in to the application by entering the correct login credentials and I will be able to access my dashboard to check the query Order and I can clarify the Issues	20	High	Deepak K Barath Kumar S
Sprint-3	Tracking System	USN-3	The user can track the process through the E-mail	20	High	Balaji S Deepak K
Sprint-4	Final delivery	USN-4	Container of applications using docker Kubernetes and deployment of the application. Create the documentation and finally submit the application	20	High	Balaji S Arun Kumar S

6.2.Sprint delivery schedule:

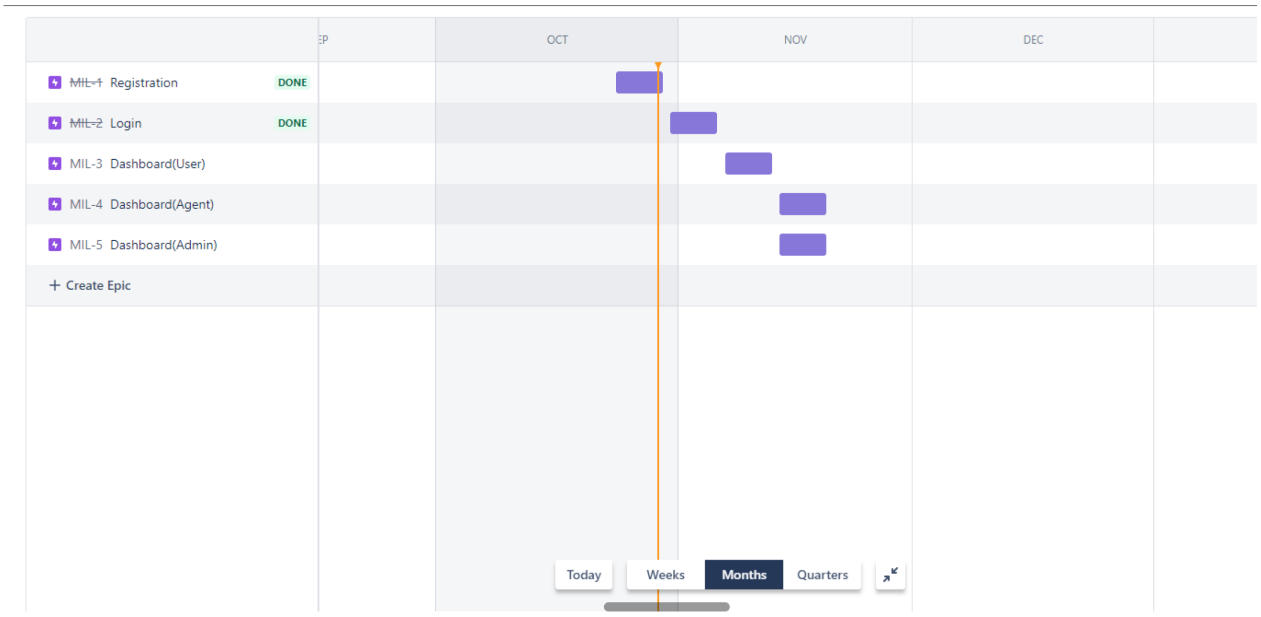
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022		29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022		5 Nov 2022
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022		12 Nov 2022
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022		19 Nov 2022

Velocity:

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)

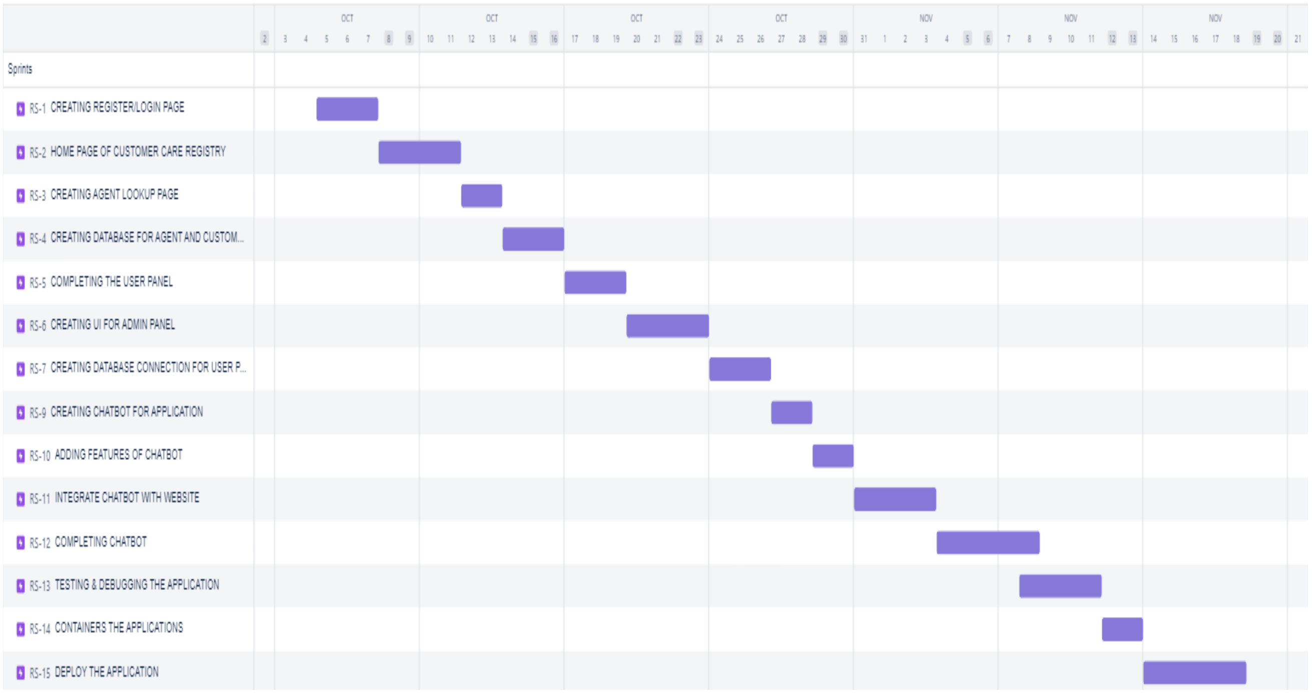
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

6.3.REPORTS FROM JIRA



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.


OCT
NOV
Sprints
RS-1 CREATING REGISTER/LOGIN PAGE
RS-2 HOME PAGE OF CUSTOMER CARE REGISTRY
RS-3 CREATING AGENT LOOKUP PAGE
RS-4 CREATING DATABASE FOR AGENT AND CUSTOM...
RS-5 COMPLETING THE USER PANEL
RS-6 CREATING UI FOR ADMIN PANEL
RS-7 CREATING DATABASE CONNECTION FOR USER P...
RS-9 CREATING CHATBOT FOR APPLICATION
RS-10 ADDING FEATURES OF CHATBOT
RS-11 INTEGRATE CHATBOT WITH WEBSITE
RS-12 COMPLETING CHATBOT
RS-13 TESTING & DEBUGGING THE APPLICATION
RS-14 CONTAINERS THE APPLICATIONS
RS-15 DEPLOY THE APPLICATION

7.CODING AND SOLUTIONING

7.1.ADMIN ASSIGNING AN AGENT TO A TICKET:

```
@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'))
```

Explanation:

• User creates a ticket by describing the query

• Admin views the newly created ticket in the dashboard

• In the dropdown given, admin selects an agent

• Once selected, using fetch() the request is sent to the server

• The request URL contains both the Ticket ID and the selected Agent ID

• Using the shown SQL query, the assigned_to column of the tickets table is set to agent_id where the ticket_id column = ticket_id

• Then, the dashboard of the admin gets refreshed

7.2.CUSTOMER RAISING A TICKET

```
@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)
```

Explanation:

• User creates a ticket by describing the query

• Admin assigns an agent to this ticket

• The customer and the agent, chat with each other, in the view of clearing the customer’s doubts

• Once the customer is satisfied, the customer decides to close the ticket

• Using fetch() the request is sent to the server. The requested URL contains the Ticket ID

• Using the shown SQL query, the status of the ticket is set to “CLOSED”

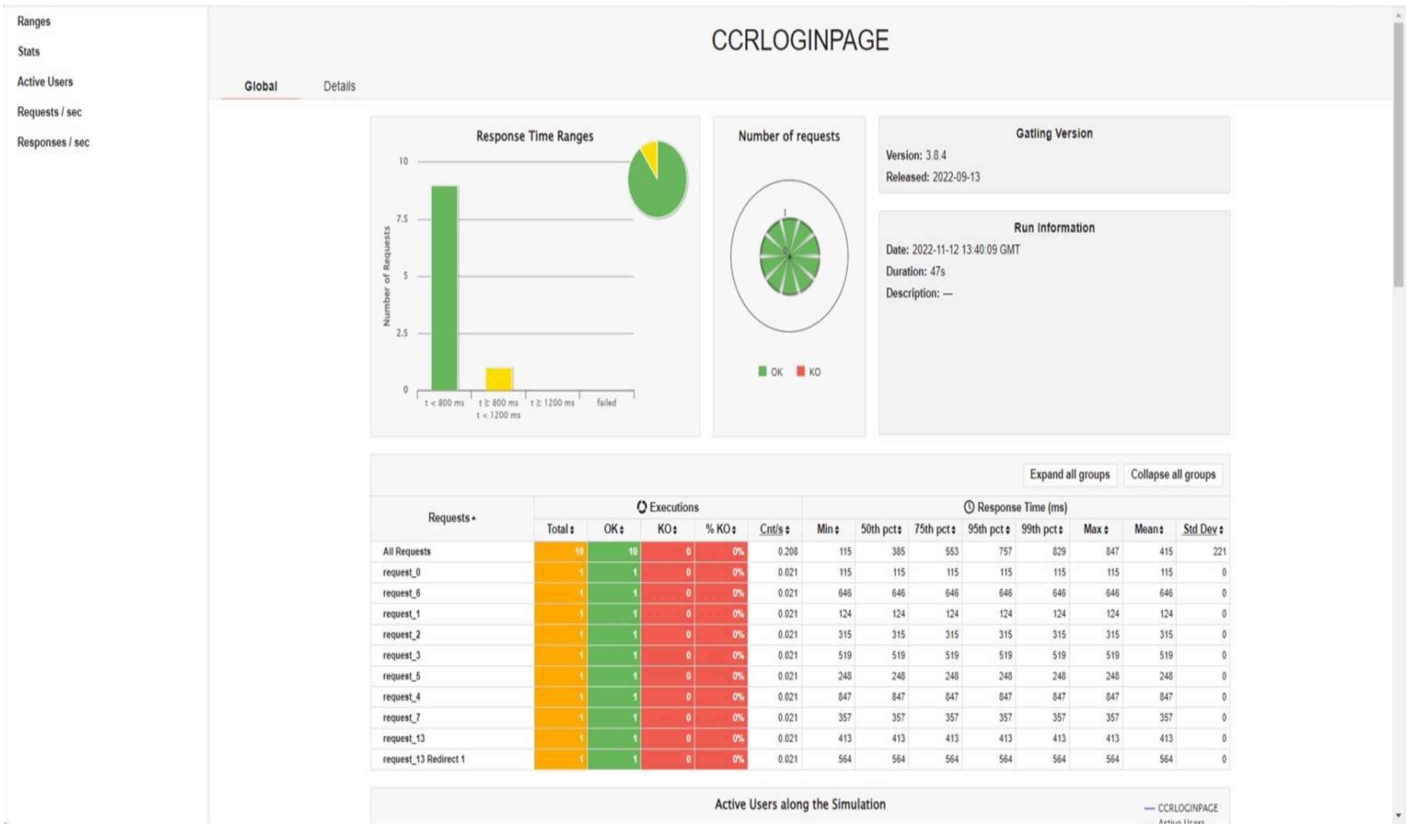
• Thus the ticket is closed

• Then the customer gets redirected to the all-tickets page

8.TESTING

8.1.TEST CASES

USER LOGIN PAGE

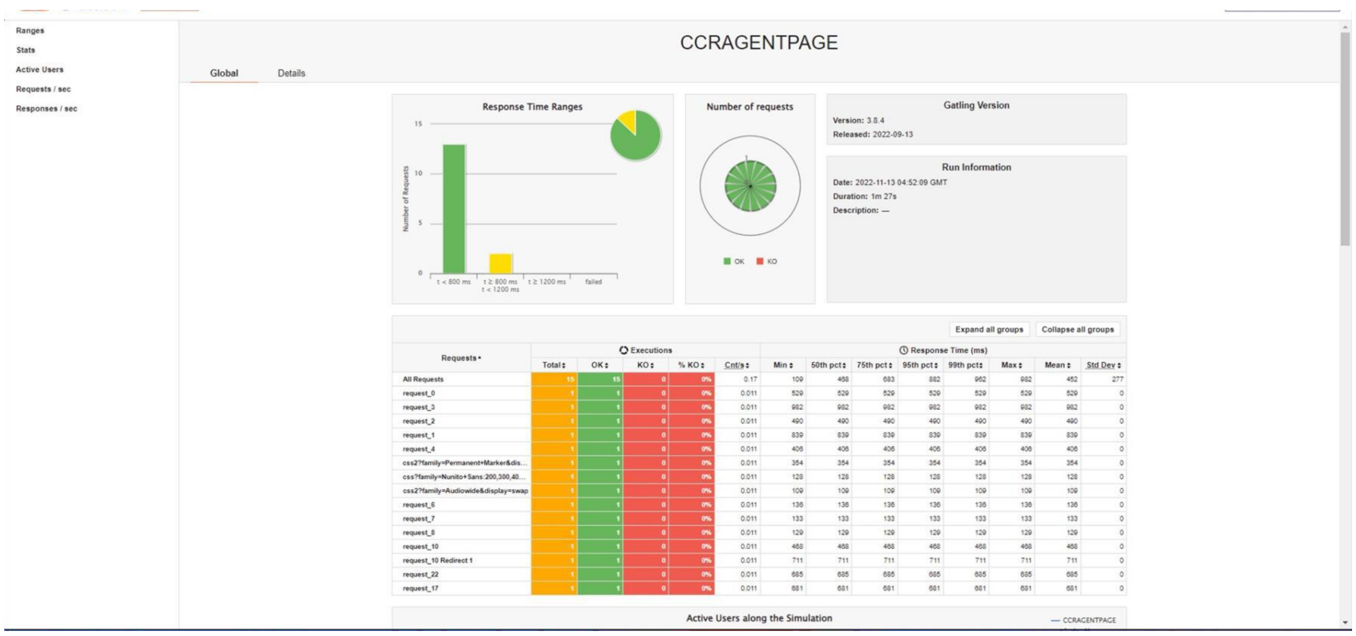


Gatling Version
Version: 3.8.4
Released: 2022-09-13

Run Information
Date: 2022-11-12 13:40:09 GMT
Duration: 47s
Description: —

Requests ▴	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	10	10	0	0%	0.208	115	385	553	757	829	847	415	221
request_0	1	1	0	0%	0.021	115	115	115	115	115	115	115	0
request_6	1	1	0	0%	0.021	646	646	646	646	646	646	646	0
request_1	1	1	0	0%	0.021	124	124	124	124	124	124	124	0
request_2	1	1	0	0%	0.021	315	315	315	315	315	315	315	0
request_3	1	1	0	0%	0.021	519	519	519	519	519	519	519	0
request_5	1	1	0	0%	0.021	248	248	248	248	248	248	248	0
request_4	1	1	0	0%	0.021	847	847	847	847	847	847	847	0
request_7	1	1	0	0%	0.021	357	357	357	357	357	357	357	0
request_13	1	1	0	0%	0.021	413	413	413	413	413	413	413	0
request_13 Redirect 1	1	1	0	0%	0.021	564	564	564	564	564	564	564	0

AGENT LOGIN PAGE

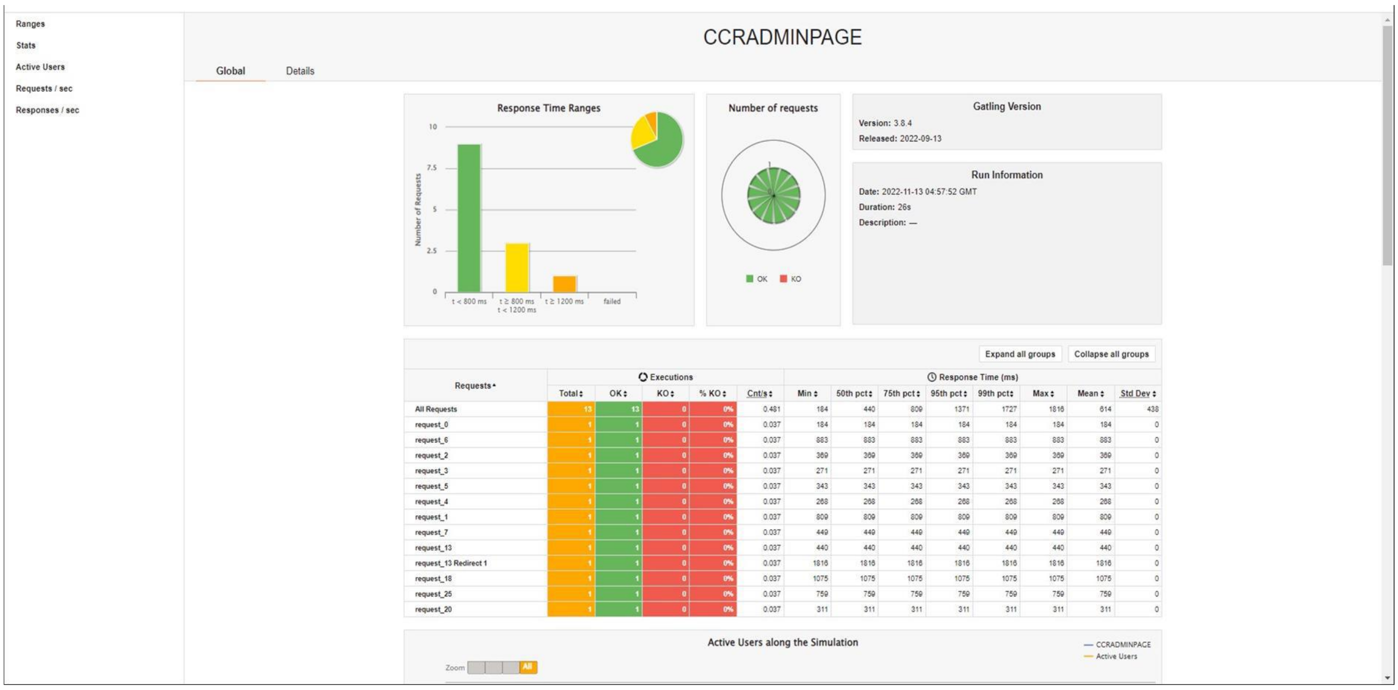


Gatling Version
Version: 3.8.4
Released: 2022-09-13

Run Information
Date: 2022-11-13 04:52:09 GMT
Duration: 1m 27s
Description: —

Requests ▴	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	15	15	0	0%	0.17	109	468	683	882	962	982	452	277
request_0	1	1	0	0%	0.011	529	529	529	529	529	529	529	0
request_3	1	1	0	0%	0.011	982	982	982	982	982	982	982	0
request_2	1	1	0	0%	0.011	490	490	490	490	490	490	490	0
request_1	1	1	0	0%	0.011	839	839	839	839	839	839	839	0
request_4	1	1	0	0%	0.011	406	406	406	406	406	406	406	0
css2?family=Permanent+Marker&dis...	1	1	0	0%	0.011	354	354	354	354	354	354	354	0
css?family=Nunito+Sans:200,300,40...	1	1	0	0%	0.011	128	128	128	128	128	128	128	0
css2?family=Audiowide&display=swap	1	1	0	0%	0.011	109	109	109	109	109	109	109	0
request_6	1	1	0	0%	0.011	136	136	136	136	136	136	136	0
request_7	1	1	0	0%	0.011	133	133	133	133	133	133	133	0
request_8	1	1	0	0%	0.011	129	129	129	129	129	129	129	0
request_10	1	1	0	0%	0.011	468	468	468	468	468	468	468	0
request_10 Redirect 1	1	1	0	0%	0.011	711	711	711	711	711	711	711	0
request_22	1	1	0	0%	0.011	685	685	685	685	685	685	685	0
request_17	1	1	0	0%	0.011	681	681	681	681	681	681	681	0

ADMIN LOGIN PAGE



CCRADMINPAGE

Gatling Version

Version: 3.8.4
Released: 2022-09-13

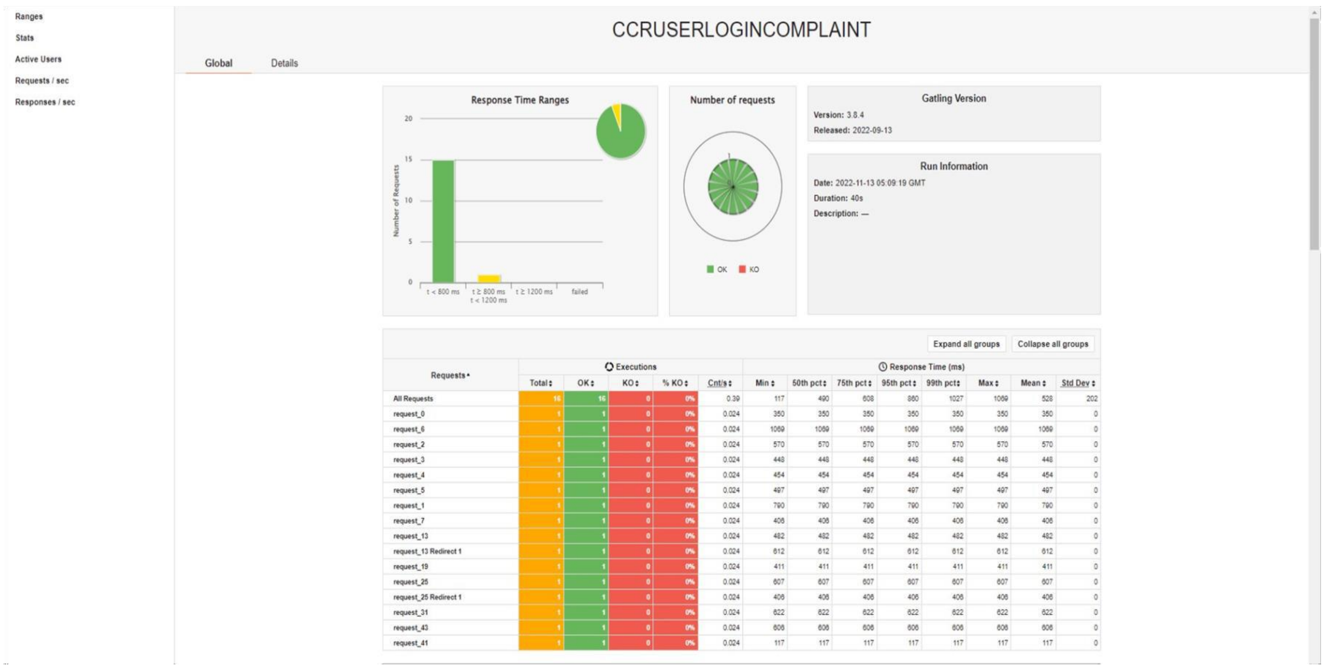
Run Information

Date: 2022-11-13 04:57:52 GMT
Duration: 26s
Description: —

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	13	13	0	0%	0.481	184	440	809	1371	1727	1816	614	438
request_0	1	1	0	0%	0.037	184	184	184	184	184	184	184	0
request_6	1	1	0	0%	0.037	883	883	883	883	883	883	883	0
request_2	1	1	0	0%	0.037	369	369	369	369	369	369	369	0
request_3	1	1	0	0%	0.037	271	271	271	271	271	271	271	0
request_5	1	1	0	0%	0.037	343	343	343	343	343	343	343	0
request_4	1	1	0	0%	0.037	268	268	268	268	268	268	268	0
request_1	1	1	0	0%	0.037	809	809	809	809	809	809	809	0
request_7	1	1	0	0%	0.037	449	449	449	449	449	449	449	0
request_13	1	1	0	0%	0.037	440	440	440	440	440	440	440	0
request_13 Redirect 1	1	1	0	0%	0.037	1816	1816	1816	1816	1816	1816	1816	0
request_18	1	1	0	0%	0.037	1075	1075	1075	1075	1075	1075	1075	0
request_25	1	1	0	0%	0.037	759	759	759	759	759	759	759	0
request_20	1	1	0	0%	0.037	311	311	311	311	311	311	311	0

USER COMPLAINT PAGE



CCRUSERLOGINCOMPLAINT

Gatling Version

Version: 3.8.4
Released: 2022-09-13

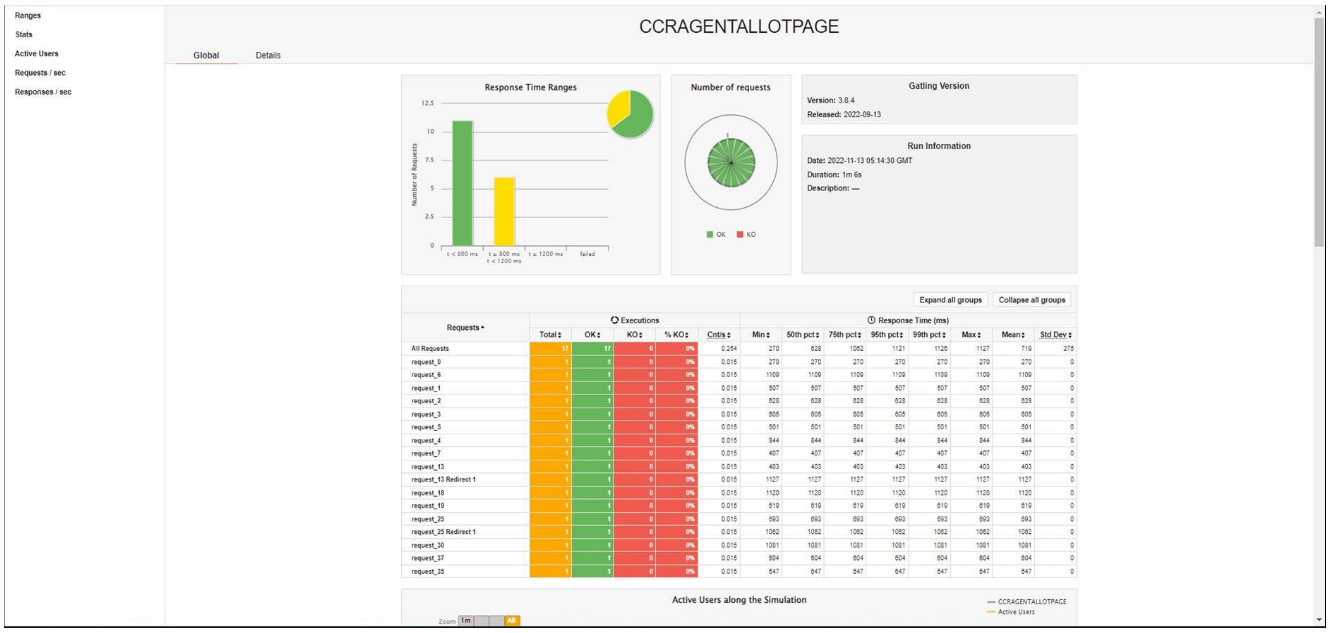
Run Information

Date: 2022-11-13 05:09:19 GMT
Duration: 40s
Description: —

Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	16	16	0	0%	0.39	117	490	608	860	1027	1069	528	202
request_0	1	1	0	0%	0.024	350	350	350	350	350	350	350	0
request_6	1	1	0	0%	0.024	1069	1069	1069	1069	1069	1069	1069	0
request_2	1	1	0	0%	0.024	570	570	570	570	570	570	570	0
request_3	1	1	0	0%	0.024	448	448	448	448	448	448	448	0
request_4	1	1	0	0%	0.024	454	454	454	454	454	454	454	0
request_5	1	1	0	0%	0.024	497	497	497	497	497	497	497	0
request_1	1	1	0	0%	0.024	790	790	790	790	790	790	790	0
request_7	1	1	0	0%	0.024	406	406	406	406	406	406	406	0
request_13	1	1	0	0%	0.024	482	482	482	482	482	482	482	0
request_13 Redirect 1	1	1	0	0%	0.024	612	612	612	612	612	612	612	0
request_19	1	1	0	0%	0.024	411	411	411	411	411	411	411	0
request_25	1	1	0	0%	0.024	607	607	607	607	607	607	607	0
request_25 Redirect 1	1	1	0	0%	0.024	406	406	406	406	406	406	406	0
request_31	1	1	0	0%	0.024	622	622	622	622	622	622	622	0
request_43	1	1	0	0%	0.024	606	606	606	606	606	606	606	0
request_41	1	1	0	0%	0.024	117	117	117	117	117	117	117	0

USER ALLOT PAGE



CCRAGENTALLOTPAGE

Gatling Version

Version: 3.8.4

Released: 2022-09-13

Run Information

Date: 2022-11-13 05:14:30 GMT

Duration: 1m 6s

Description: —

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
All Requests	17	17	0	0%	0.254	270	628	1062	1121	1126	1127	719	275
request_0	1	1	0	0%	0.015	270	270	270	270	270	270	270	0
request_6	1	1	0	0%	0.015	1109	1109	1109	1109	1109	1109	1109	0
request_1	1	1	0	0%	0.015	507	507	507	507	507	507	507	0
request_2	1	1	0	0%	0.015	628	628	628	628	628	628	628	0
request_3	1	1	0	0%	0.015	605	605	605	605	605	605	605	0
request_5	1	1	0	0%	0.015	501	501	501	501	501	501	501	0
request_4	1	1	0	0%	0.015	844	844	844	844	844	844	844	0
request_7	1	1	0	0%	0.015	407	407	407	407	407	407	407	0
request_13	1	1	0	0%	0.015	403	403	403	403	403	403	403	0
request_13 Redirect 1	1	1	0	0%	0.015	1127	1127	1127	1127	1127	1127	1127	0
request_18	1	1	0	0%	0.015	1120	1120	1120	1120	1120	1120	1120	0
request_19	1	1	0	0%	0.015	619	619	619	619	619	619	619	0
request_25	1	1	0	0%	0.015	693	693	693	693	693	693	693	0
request_25 Redirect 1	1	1	0	0%	0.015	1062	1062	1062	1062	1062	1062	1062	0
request_30	1	1	0	0%	0.015	1081	1081	1081	1081	1081	1081	1081	0
request_37	1	1	0	0%	0.015	604	604	604	604	604	604	604	0
request_35	1	1	0	0%	0.015	647	647	647	647	647	647	647	0

8.2.USER ACCEPTANCE TESTING:

Purpose of Document

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

Defect Analysis

This report shows the number of resolved or closed bugs at each severity level, and how they were resolved

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	5	0	0	2	7
External	0	2	0	0	2
Fixed	12	11	35	45	103
Not Reproduced	0	5	0	0	5
Skipped	0	0	0	0	0
Totals	17	18	35	47	117

Test case analysis

The report shows the number of tests that have passed, failed, and untested

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

10.ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

- Customers can clarify their doubts just by creating a new ticket
- The customer gets replies as soon as possible
- Not only the replies are faster, but the replies are also 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’ queries are always listened
- Free of cost

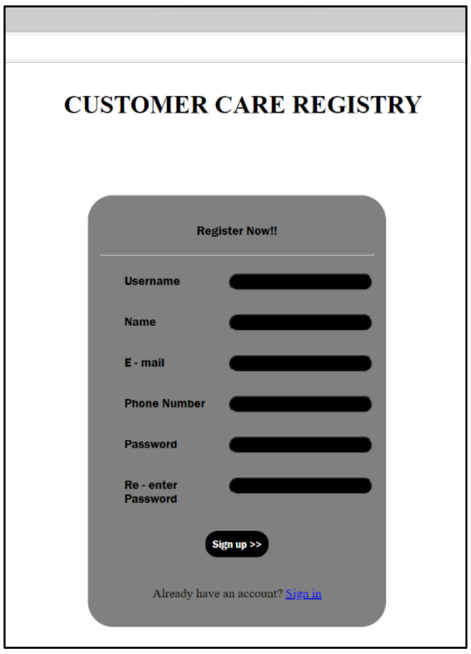
DISADVANTAGES:

- 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

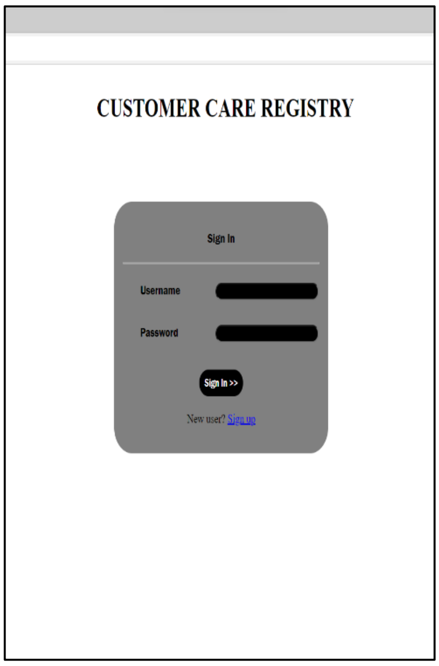
9.RESULTS

9.1.PERFORMANCE METRICS

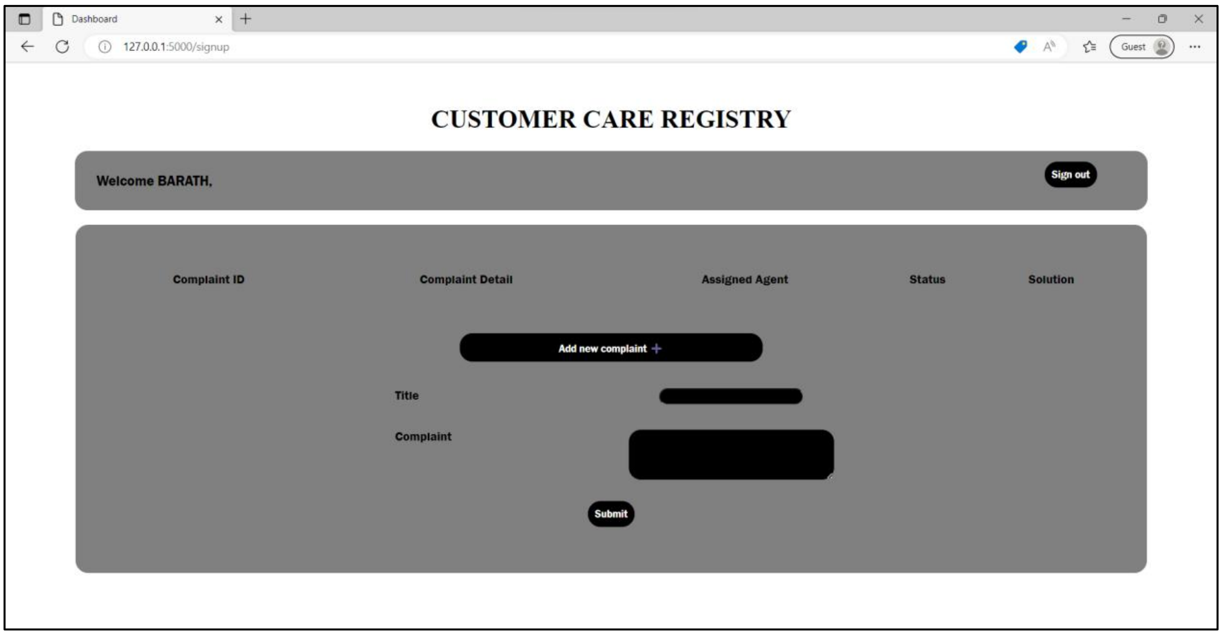
Register page



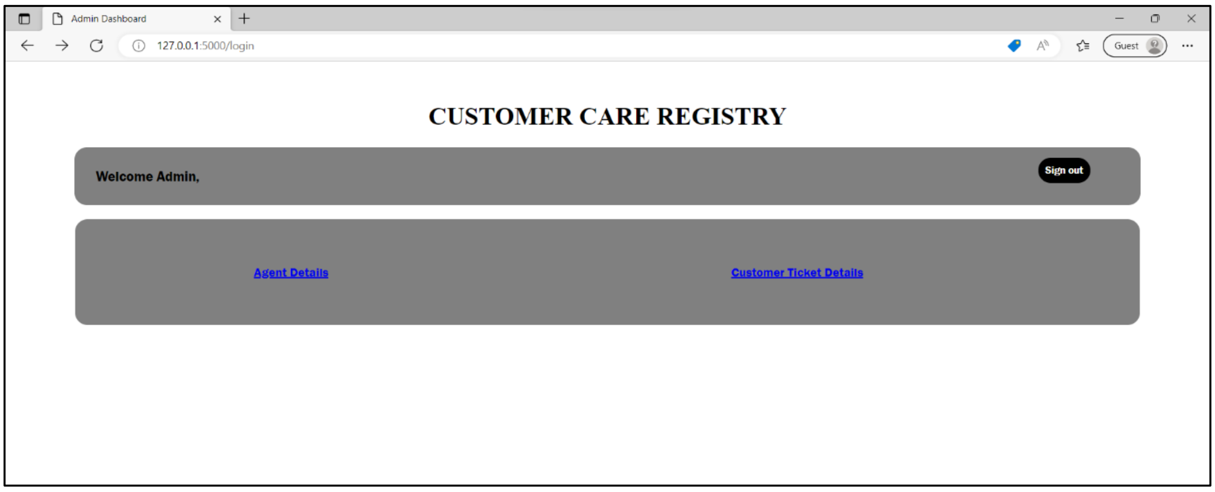
Sign in page



User dashboard



Admin dashboard



CUSTOMER CARE REGISTRY

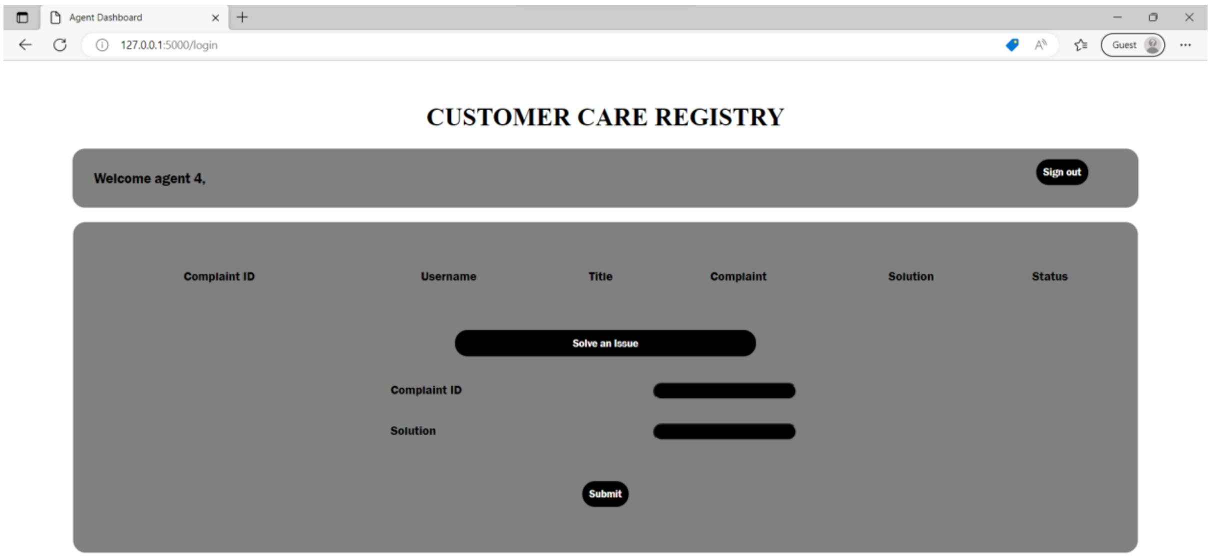
Welcome Admin,

Sign out

Agent Details

Customer Ticket Details

Agent dashboard



CUSTOMER CARE REGISTRY

Welcome agent 4,

Sign out

Complaint ID	Username	Title	Complaint	Solution	Status

Solve an Issue

Complaint ID

Solution

Submit

11.CONCLUSION

Thus, there are many customer service applications available on the internet. Noting down the structural components of those applications and building a customer care registry. It will be web application build with Flask (Python micro-web framework), HTML, CSS, IBMDB2. It will be a customer service registry. Customers can register into the application using their email, password, and username. Then, they can log in to the system, and raise as queries 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 and solved the queries.

12.FUTURE SCOPE

Over the years, customer expectations generally haven't changed. Customers want to be served quickly and completely on the first try. If they're speaking to a human agent, they want a friendly, knowledgeable interaction-- the goal being to resolve the customer's problem or answer their question quickly and easily.

Drilling down, however, customer expectations are influenced by the changes in technology. Just five years ago, for example, few customers would have expected to communicate with businesses over SMS or messaging services from their mobile phone. Now, it's common because consumers use those applications in other areas of their lives.

1. Hybrid workplaces

Planned by 71% of companies, hybrid workplaces will increase the availability of agents, allowing companies to better respond to an emergency or high-volume situations. If a weather issue is causing flight delays, for example, airlines can more easily call in agents off-hours because they can work from their home office instead of commuting to the contact center.

2. Chief customer officers

By the end of 2021, nearly 75% of companies had planned to have a chief customer officer on board. CCOS raises the visibility of customer experience to C-suite executives. They also conduct agent and voice-of-the-customer analyses to regularly evaluate and revise technology implementations.

3. Higher wages for agents, less turnover

In 2022, 44% of companies will increase agent pay. Chatbots and virtual assistants are replacing the functions of basic or entry-level agents. Agents are now required to be more experienced. Average pay ranges from $21.64 to $42.31 per hour. Higher pay, along with flexible work schedules, will help reduce agent turnover rates.

4. More virtual assistants and video

Virtual assistants and video are the two fastest-growing customer interaction channels. Virtual assistants help customers navigate websites and self-service portals, while video helps agents see what customers are doing and resolve their issues. As companies continue to add video, customer service will improve.

13.APPENDIX

SOURCE CODE:

```
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=55fbc997-9266-4331-
afd3-
888b05e734c0.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=3192
9;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=ynw48180;
PWD=I1EGQPDz745BoBjp;", '', '')

print("connected..")

@app.route('/signup', methods=['GET', 'POST'])

def signup():

    global userid

    msg = ''

    if request.method == 'POST':

        username = request.form['username']
```

```
name = request.form['name']

email = request.form['email']

phn = request.form['phn']

password = request.form['pass']

repass = request.form['repass']

print("inside checking")

print(name)

if len(username) == 0 or len(name) == 0 or len(email) == 0 or len(phn) ==
0 or len(password) == 0 or len(repass) == 0:

    msg = "Form is not filled completely!!"

    print(msg)

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

elif password != repass:

    msg = "Password is not matched"

    print(msg)

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

elif not re.match(r'[a-z]+', username):

    msg = 'Username can contain only small letters and numbers'

    print(msg)

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

elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):

    msg = 'Invalid email'

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


  elif not re.match(r'[A-Za-z]+', name):

    msg = "Enter valid name"

    print(msg)

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

 elif not re.match(r'[0-9]+', phn):

    msg = "Enter valid phone number"

    print(msg)

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


 sql = "select * from users where username = ?"

 stmt = ibm_db.prepare(conn, sql)

 ibm_db.bind_param(stmt, 1, username)

 ibm_db.execute(stmt)

 account = ibm_db.fetch_assoc(stmt)

 print(account)

 if account:

    msg = 'Acccount already exists'

 else:

    userid = username

    insert_sql = "insert into users values(?,?,?,?,?)"

    prep_stmt = ibm_db.prepare(conn, insert_sql)
```

```
            ibm_db.bind_param(prep_stmt, 1, username)


ibm_db.bind_param(prep_stmt, 2, name)

            ibm_db.bind_param(prep_stmt, 3, email)

            ibm_db.bind_param(prep_stmt, 4, phn)

            ibm_db.bind_param(prep_stmt, 5, password)

            ibm_db.execute(prep_stmt)

            print("successs")

            msg = "succesfully signed up"

        return render_template('dashboard.html', msg=msg, name=name)

    else:

        return render_template('signup.html')


@app.route('/dashboard')

def dashboard():

    return render_template('dashboard.html')


@app.route('/')

def base():

    return redirect(url_for('login'))


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

def login():
  global userid


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

```
            msg = 'logged in successfully'


            # for getting complaints details

            sql = "select * from complaints where assigned_agent = ?"

            complaints = []

            stmt = ibm_db.prepare(conn, sql)

            ibm_db.bind_param(stmt, 1, username)

            ibm_db.execute(stmt)

            dictionary = ibm_db.fetch_assoc(stmt)

            while dictionary != False:

                complaints.append(dictionary)

                dictionary = ibm_db.fetch_assoc(stmt)

            print(complaints)

            return render_template('agentdash.html',
name=account['USERNAME'], complaints=complaints)


        sql = "select * from users where username = ? and password = ?"

        stmt = ibm_db.prepare(conn, sql)

        ibm_db.bind_param(stmt, 1, username)

        ibm_db.bind_param(stmt, 2, password)

        ibm_db.execute(stmt)

        account = ibm_db.fetch_assoc(stmt)

        print(account)
```

```
if account:


    session['Loggedin'] = True

    session['id'] = account['USERNAME']

    userid = account['USERNAME']

    session['username'] = account['USERNAME']

    msg = 'logged in successfully'


    # for getting complaints details

    sql = "select * from complaints where username = ?"

    complaints = []

    stmt = ibm_db.prepare(conn, sql)

    ibm_db.bind_param(stmt, 1, username)

    ibm_db.execute(stmt)

    dictionary = ibm_db.fetch_assoc(stmt)

    while dictionary != False:

        # print "The ID is : ",  dictionary["EMPNO"]

        # print "The Name is : ", dictionary[1]

        complaints.append(dictionary)

        dictionary = ibm_db.fetch_assoc(stmt)


    print(complaints)
```

```
            return render_template('dashboard.html', name=account['USERNAME'],
complaints=complaints)


     else:

            msg = 'Incorrect user credentials'

            return render_template('dashboard.html', msg=msg)

    else:

        return render_template('login.html')




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

def add():

    if request.method == 'POST':

        title = request.form['title']

        des = request.form['des']

        try:

            sql = "insert into complaints(username,title,complaint) values(?,?,?)"

            stmt = ibm_db.prepare(conn, sql)

            ibm_db.bind_param(stmt, 1, userid)

            ibm_db.bind_param(stmt, 2, title)

            ibm_db.bind_param(stmt, 3, des)

            ibm_db.execute(stmt)

        except:
            print(userid)

            print(title)



          print(des)

            print("cant insert")

        sql = "select * from complaints where username = ?"

        complaints = []

        stmt = ibm_db.prepare(conn, sql)

        ibm_db.bind_param(stmt, 1, userid)

        ibm_db.execute(stmt)

        dictionary = ibm_db.fetch_assoc(stmt)

        while dictionary != False:

            # print "The ID is : ",  dictionary["EMPNO"]

            # print "The Name is : ", dictionary[1]

            complaints.append(dictionary)

            dictionary = ibm_db.fetch_assoc(stmt)

        print(complaints)

        return render_template('dashboard.html', name=userid,
complaints=complaints)



@app.route('/agents')

def agents():
    sql = "select * from agents"
    agents = []

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


@app.route('/addnewagent', methods=["GET", "POST"])
def addagent():
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        domain = request.form['domain']
        password = request.form['password']
        try:
            sql = "insert into agents values(?,?,?,?,?,?,2)"
            stmt = ibm_db.prepare(conn, sql)

            ibm_db.bind_param(stmt, 1, username)


          ibm_db.bind_param(stmt, 2, name)

            ibm_db.bind_param(stmt, 3, email)

            ibm_db.bind_param(stmt, 4, phone)

            ibm_db.bind_param(stmt, 5, password)

            ibm_db.bind_param(stmt, 6, domain)

            ibm_db.execute(stmt)

        except:

            print("cant insert")

        sql = "select * from agents"

        agents = []

        stmt = ibm_db.prepare(conn, sql)

        ibm_db.execute(stmt)

        dictionary = ibm_db.fetch_assoc(stmt)

        while dictionary != False:

            agents.append(dictionary)

            dictionary = ibm_db.fetch_assoc(stmt)


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

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

def updatecomplaint():


    if request.method == 'POST':

      cid = request.form['cid']

      solution = request.form['solution']

      try:

        sql = "update complaints set solution =?,status=1 where c_id = ? and
assigned_agent=?"

        stmt = ibm_db.prepare(conn, sql)

        ibm_db.bind_param(stmt, 1, solution)

        ibm_db.bind_param(stmt, 2, cid)

        ibm_db.bind_param(stmt, 3, userid)

        ibm_db.execute(stmt)

        sql = "update agents set status =3 where username=?"

        stmt = ibm_db.prepare(conn, sql)

        ibm_db.bind_param(stmt, 1, userid)

        ibm_db.execute(stmt)

      except:

        print("cant insert")

      sql = "select * from complaints where assigned_agent = ?"

      complaints = []

      stmt = ibm_db.prepare(conn, sql)
```

```
        ibm_db.bind_param(stmt, 1, userid)

        ibm_db.execute(stmt)


        dictionary = ibm_db.fetch_assoc(stmt)

        while dictionary != False:

            complaints.append(dictionary)

            dictionary = ibm_db.fetch_assoc(stmt)


   # print(complaints)

        return render_template('agentdash.html', name=userid,
complaints=complaints)




@app.route('/tickets')

def tickets():

    sql = "select * from complaints"

    complaints = []

    stmt = ibm_db.prepare(conn, sql)

    ibm_db.execute(stmt)

    dictionary = ibm_db.fetch_assoc(stmt)

    while dictionary != False:

        complaints.append(dictionary)

        dictionary = ibm_db.fetch_assoc(stmt)
    sql = "select username from agents where status <> 1"

    freeagents = []

    stmt = ibm_db.prepare(conn, sql)

    ibm_db.execute(stmt)

    dictionary = ibm_db.fetch_assoc(stmt)

    while dictionary != False:


    freeagents.append(dictionary)

        dictionary = ibm_db.fetch_assoc(stmt)

    print(freeagents)

    return render_template('tickets.html', complaints=complaints,
freeagents=freeagents)



@app.route('/assignagent', methods=['GET', 'POST'])

def assignagent():

    if request.method == "POST":

        ccid = request.form['ccid']

        agent = request.form['agent']

        print(ccid)

        print(agent)

        try:
            sql = "update complaints set assigned_agent =? where c_id = ?"
            stmt = ibm_db.prepare(conn, sql)


            ibm_db.bind_param(stmt, 1, agent)
            ibm_db.bind_param(stmt, 2, ccid)
            ibm_db.execute(stmt)
            sql = "update agents set status =1 where username = ?"
            stmt = ibm_db.prepare(conn, sql)


            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant update")
        return redirect(url_for('tickets'))
if __name__ == "__main__":
    app.run(debug=True)
```

GITHUB LINK:

https://github.com/IBM-EPBL/IBM-Project-8328-1658915584

DEMO LINK:

https://drive.google.com/file/d/1BBglOCJbuqSMO1eM1i2XvrKJnmHtElOi/view?usp=sharing