PROJECT REPORT ON CUSTOMER CARE REGISTRY

Date	07 November 2022
Team ID	PNT2022tmid31596
Project Name	Customer Care Registry

TEAM MEMBERS:

Team lead : Keerthana S

Team Member1 : Keerthana S

Team Member2 : Gokulkrishnan V

Team Member3 : Mukeswaran D

Team member4 : Jana shruthi M

CONTENTS

- **INTRODUCTION**
  - **Project Overview**
  - **Purpose**
- **LITERATURE SURVEY**
  - **Existing Problem**
  - **References**
  - **Problem Statement Definition**
- **IDEATION AND PROPOSED SOLUTION**
  - **Empathy Map Canvas**
  - **Ideation And Brainstorming**
  - **Proposed Solution**
  - **Problem Solution fit**
- **REQUIREMENT ANALYSIS**
  - **Functional requirement**
  - **Non-Functional requirements**

- **PROJECT DESIGN**
    - **Data Flow Diagrams**
    - **Solution & Technical Architecture**
    - **User Stories**
- **PROJECT PLANNING AND SCHEDULING**
    - **Sprint Planning And Estimation**
    - **Sprint Delivery Schedule**
    - **Reports from JIRA**
- **CODING AND SOLUTIONING**
    - **Feature 1**
    - **Feature 2**
    - **Feature 3**
    - **Database Schema (if Applicable)**
- **TESTING**
    - **Test Cases**
    - **User Acceptance Testing**
- **RESULTS**
    - **Performance Metrics**

10.ADVANTAGES AND

DISADVANTAGES

11.CONCLUSION

- **FUTURE SCOPE**
- **APPENDIX**
    - **Source Code**
    - **Screenshot**
    - **GitHub And Project Demo Link**

CUSTOMER CARE REGISTR

1.INTRODUCTION

1.1PROJECT OVERVIEW:

The application of Customer care registry has been developed to help the customer in processing their complaints. And also the customer 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. Customer can view the status of the ticket till the service provided. Companies benefits from investing in customer care for multiple reasons:

- Customers get the insights they need to make an informed purchase.
- Customer satisfaction can increase and customer loyalty can improve.
- The time spent by customer service agents on routine inquiries can also lead to increased productivity
- Customer receive faster and more efficient service.
- Customers are less likely to be frustrated with customer service agents.

Customer care and customer service together help create a positive customer experience, or the overall impression a person has when interacting with your company. Both are essential , but they are implemented in slightly difference ways. Customer care is proactive when it comes to providing high-quality services. Customers feel supported throughout the buyer's journey because their needs are

anticipated. Therefore , companies and customers are able to establish an emotional connection. Customer service is reactive. A self-service option or customer service is available for customers who need help solving problem or answering questions before purchasing.

1.2 PURPOSE:

The purpose of this proposed project work is : To help the customer in processing their complaints.To use a CNN approach in recognizing handwritten Tamil character in offline mode .The goal is to remove any potential causes of dis-satisfication. A potential source of dissatisfaction must be eliminated. The purpose of this work is to use a CNN approach in order to recognize handwritten Tamil characters in offline mode . A favourable client experience can lead to recurring business and referrals because of good customer service. Any business that provides excellent customer service thrives. To assist the client in resolving their concern

2. LITERATURE SURVEY

2.1 EXISTING PROCESS:

The expected outcome of this proposed project work is to: To gather the complaints about the customers. Customer service will be recover the issues. Customer issue are solved. User can easily access the web application**.** In order to compile client complaints. Customer care will fix the problems. Solved customer issues, The online application is easily accessible by the user. Customer can view the status of the ticket till the service provided.

2.2 REFRENCE:

S.No	Title	Publication Details	Methodology/ Algorithms	Merits	Demerits
1.	Customer care application	Zain Raza Syed M Raza , Hamid , Ahmed Faizan , Faisal , Malik	This proposed system helps to offer several applications in various fields includes Desktop based Admin Panel, etc. It can measure storage ,	Data storage consumption, Cost reduction	Very less application in down time

			entertainment , management , social networking, GPS.		
2.	A framework of customer Complaint handling system.	Amy J , C.Trappey , Ching-Hung Lee , Wen-Pin Chen , Charles V , Trappe	Customers have strong demands for quick responses to their complaints. In this study , a framework of complaint handling system is analysed and developed for a other restaurant chain	Customer satisfaction enhancement & Boost in customer communication. Positive impact an brand image	It can be a time consuming process to collate complaints and identify action.
3.	Automation to handle Customer complaints in Banks Using BPM Tool. .	Gnana Sunny Antony	This project was focused on developing a new customer centric application for automating Complaints mechanism throughout all platform. This project involved developing and testing the new application and focusing	Decrease the Maintenan e cost by 25%. Cloud server database. Reduced overhead costs.	The main drawback of automation is that it lacks human touch. Companies are dependent on technology .

			on being customer centric and to beat the growing demand of banking market .		
4.	Customer care registry What Affects Employee Performance thought work motivation	Nabilah Aliyyah, Indra Prasetyo, Rusdiyanto, Nawang Kalbuana	This type of research uses explanatory research with a quantitative approach. Quantitative method is a research method based on a specific population	Reduced processing time. Very information primitives.	Manual selection. Pre-processing is necessary (segmentation)
5	The Role of Customer Behavior and Relationship Management in Modern Enterprise Marketing	Jialuli	Customer behavior management is a new business model which takes customer as the center, information sharing and communicatio n as the main purpose. .	Reduce the stress Flexibility	It will be be harder to provide constructive feedback.
6.	CRM (customer relationship management)	Ni Made Nopita Wati,Erna Hendrawati,	The formulation of the problem in this research	It speed up the sales conversion process. It increase staff	CRM may not suit all businesses. Security and

		I Gede Juanamasta	how does the role of customer service through Customer Relationship Management (CRM) to improve customer loyalty and good image	productivity , lowering time-cost	data protection issues with centralised data.
7.	Chatbot for customer service	Gunasekaran A , Marri H.B , Chung Minjee. Joung Heerim	In this paper customer trust chatbots to provide the required support. Chatbots represent a potential means for automating customer service.	Cost efficient This provides automated customer service with the use of the cloud.	Less accuracy
8.	Integrated Information System for Customer Care	D Riananingrum, R R S Hari, F Nursaori and WA Astuti	The goal is to understand customers' needs and expectations to establish good relationships with customers. This study aims to determine Customer Facing and Ecosystem Facing services in the	The objectives and processes are essentially the same. Integation should reduce the possibility of resolving problems. Integation should lead to the avoidance of duplication.	Existing system may work well already

			digital transformation business.		
9.	E- Customer Care Service System for Benin Electricity Distribution Company	Olutayo Boyinbode , Akure	This paper aims to eliminate these weaknesses by automating the process through an e-Customer Care Service System by which customer makes an enquiries and complaints on the services	Time consuming Cost efficient	Work an irregular schedule .
10.	Customer care registry Effective Communicatio n, and Motivation in Customer Service	Mitra Madanchian, Hamed Taherdoost,J ay Ariken	Implementing feedback is one of the ways to achieve customer service motivation and have a positive effect on the overall performance of the team; One-on-one communication – Talking one-on-one to the customer service team is a great way to extract honest feedback and their experiences with customer interactions.	Lower cost Large sample sizes Enhanced generalisability	Ethical and legal issues .

- **PROBLEM STATEMENT DEFINITION:**

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

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

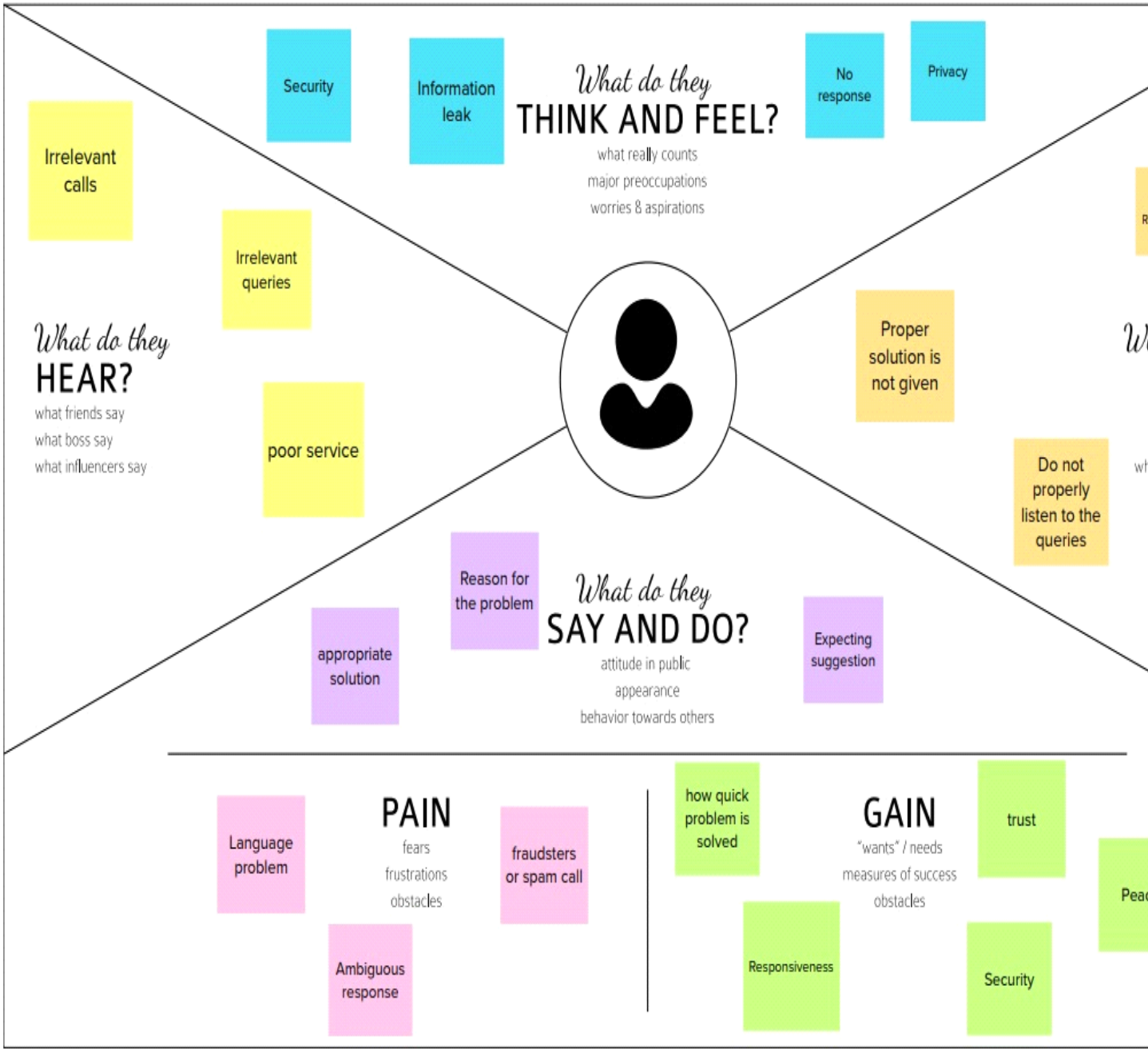
Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	Regular Customer	Purchase products online	I don't get proper reviews	They are not from real experts.	Frustrated
PS-2	Regular Customer	Bought a product	I cannot get my doubts clarified	There is no proper system	Disappointed
PS-3	Regular customer	Raise queries about a product	I am getting invalid answers / replies are	Replies are from unauthenticated persons	Stupid

			too late		

3.IDEATION AND PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS:

- An empathy map is a simple, easy-to-digest visual that captures knowledge about a user's behaviours and attitudes.
- It is a useful tool to helps teams better understand their users.
- Creating an effective solution requires understanding the true problem and the person who is experiencing it.
- The exercise of creating the map helps participants consider things from the user's perspective along with his or her goals and challenges.

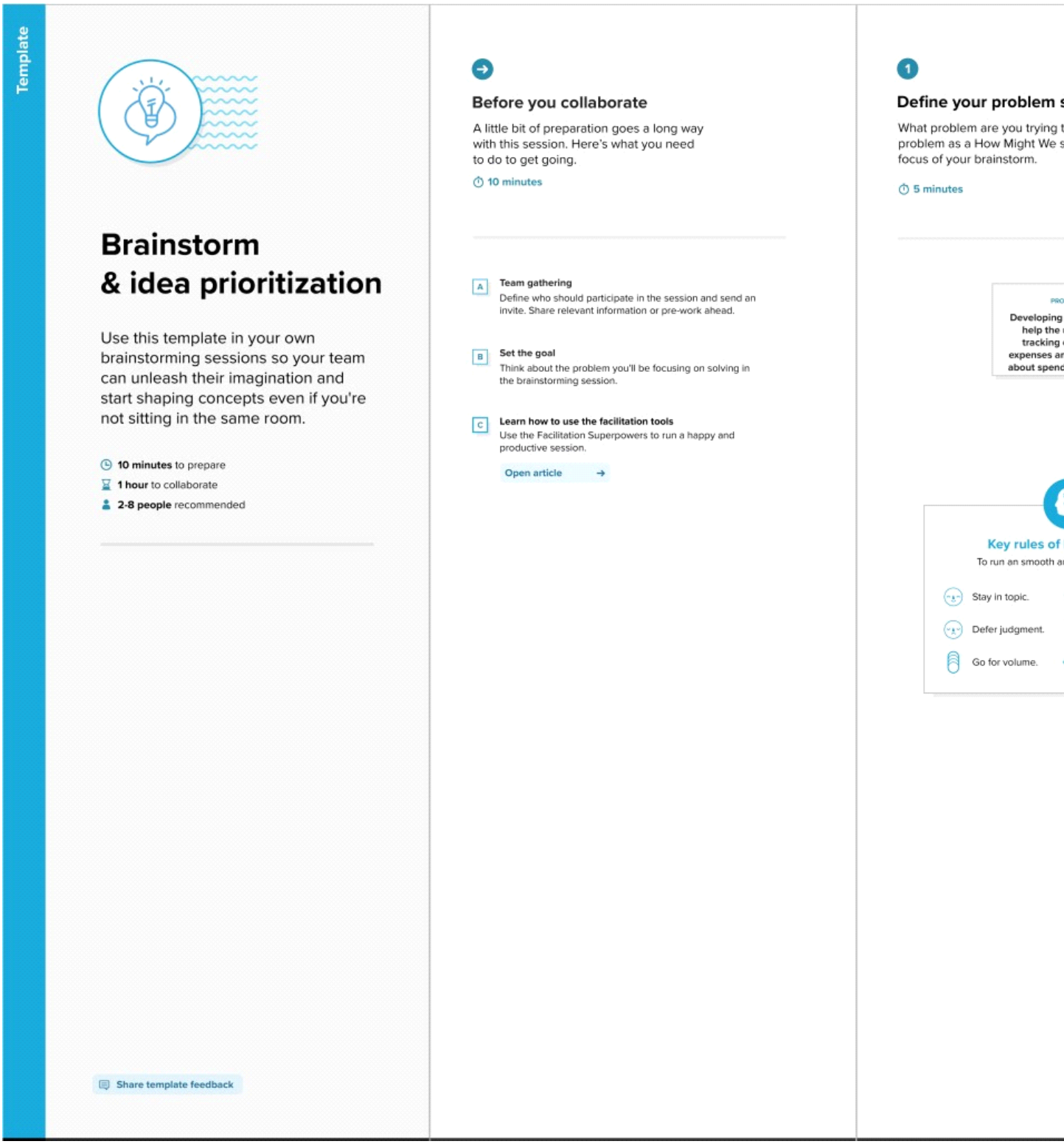


3.2 IDEATION AND BRAINSTORMING:

Brainstorming provides a free and open environment that encourages everyone within a team to participate in the creative thinking process that leads to problem solving. Prioritizing volume over value, out-of-the-box ideas are welcome

and built upon, and all participants are encouraged to collaborate, helping each other develop a rich number of creative solutions.

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



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!

Person 1

User Feedback	Filteration based on vendors	Providing services on time
Customer Privacy	Providing Chatbox	Asking for Rating
Solution to Customer		

Person 2

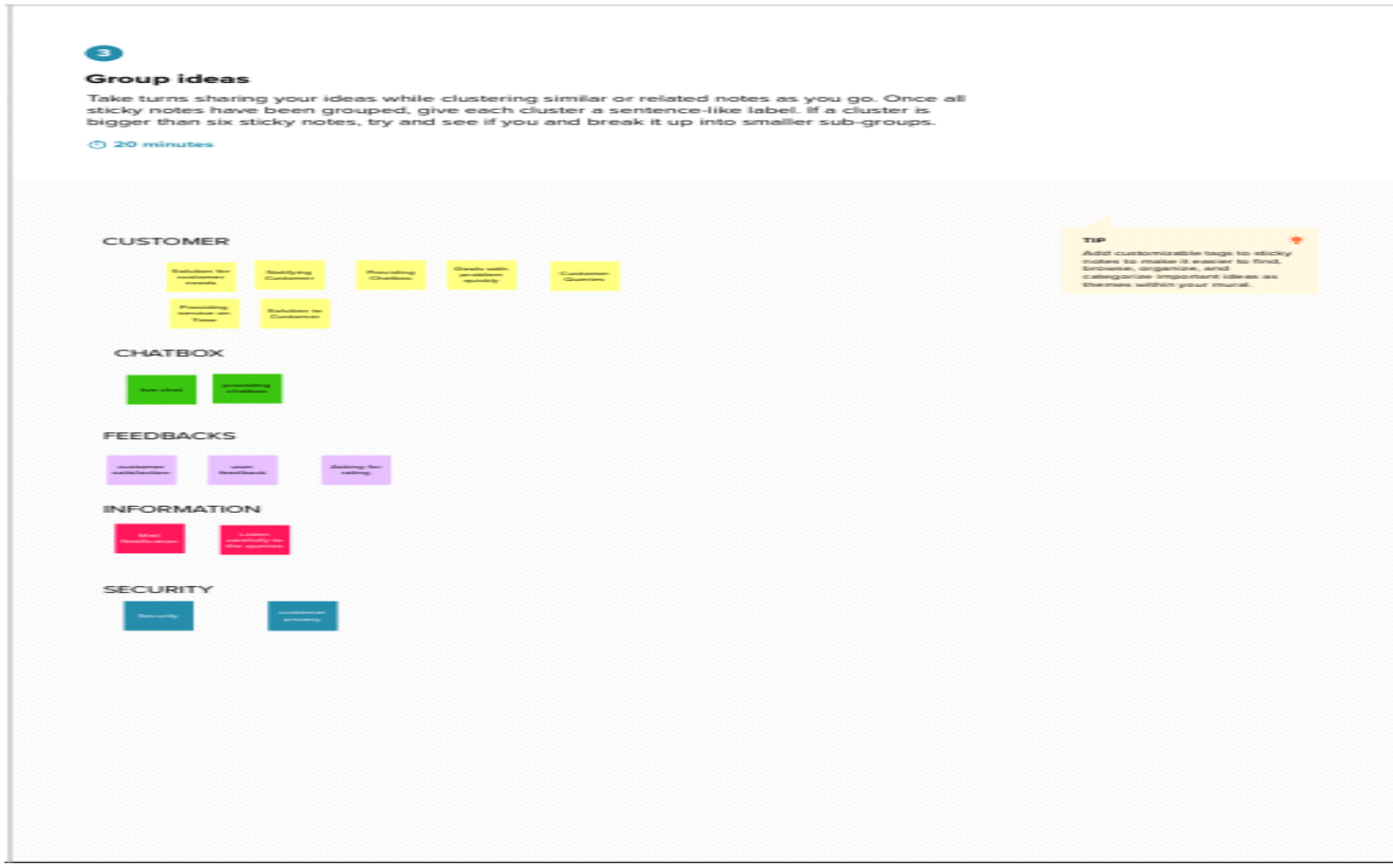
Customer Satisfaction	Deals with Problem quickly	Listen Carefully to the queries
Tracking of Services	Filteration Based on details	Allocating Agent

Person 3

Deals with problem quickly	Email Notification	Customer Satisfaction
Providing service details	Customer Query	Agent details
User chatbox		

Person 4

Verifying customer	Solution for Customer issues	Checking customer needs
Security	User chat	Providing Chatbox



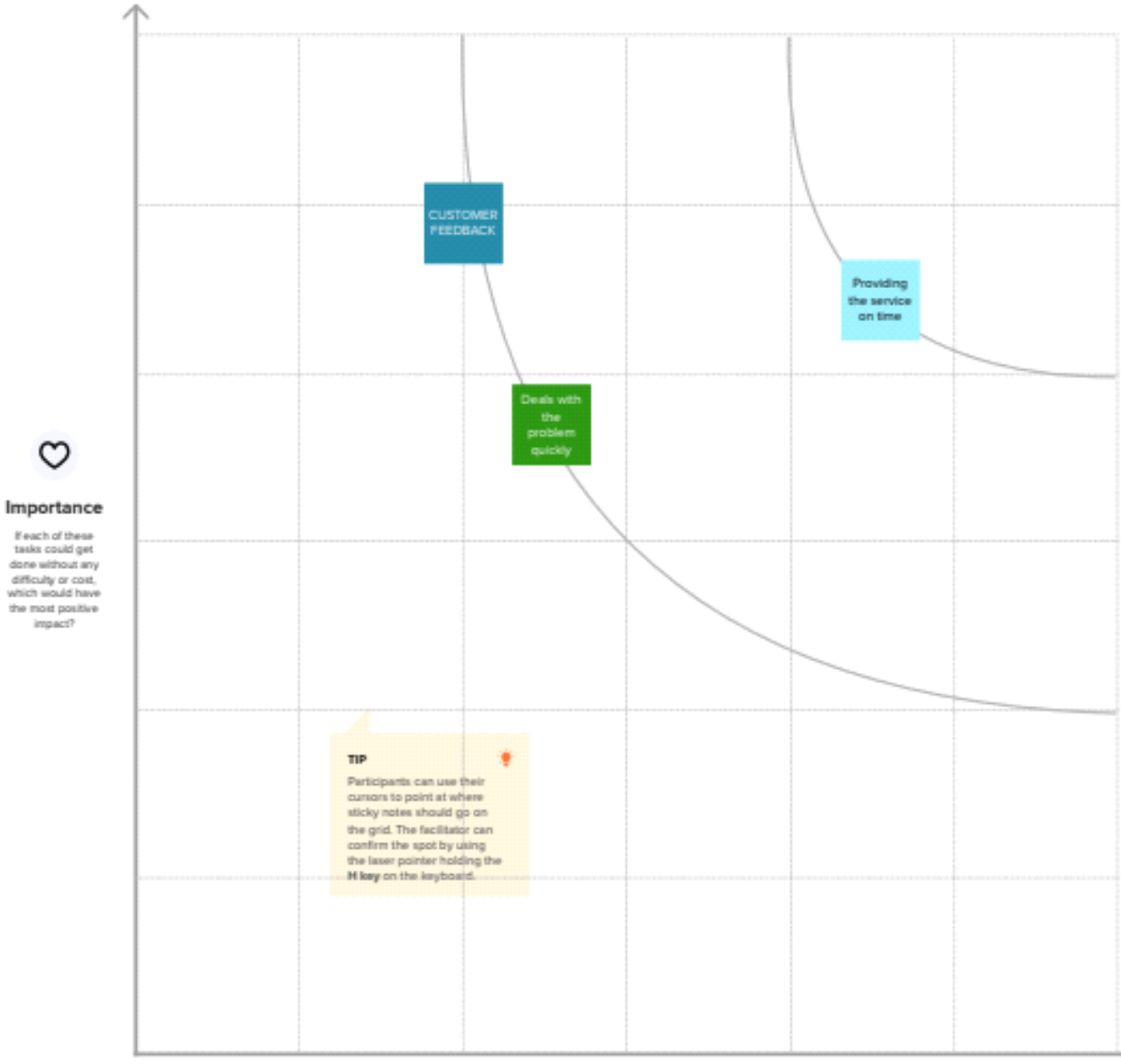
Step-3: Idea Prioritization

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

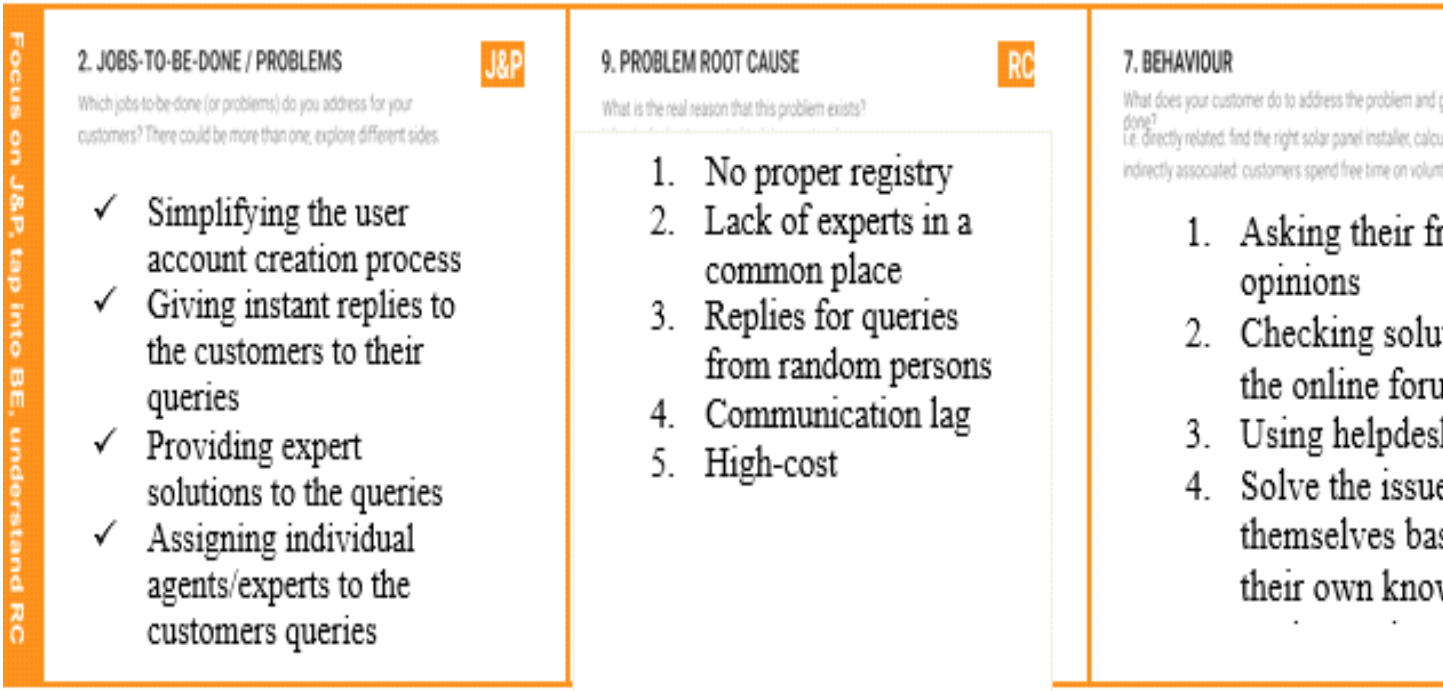


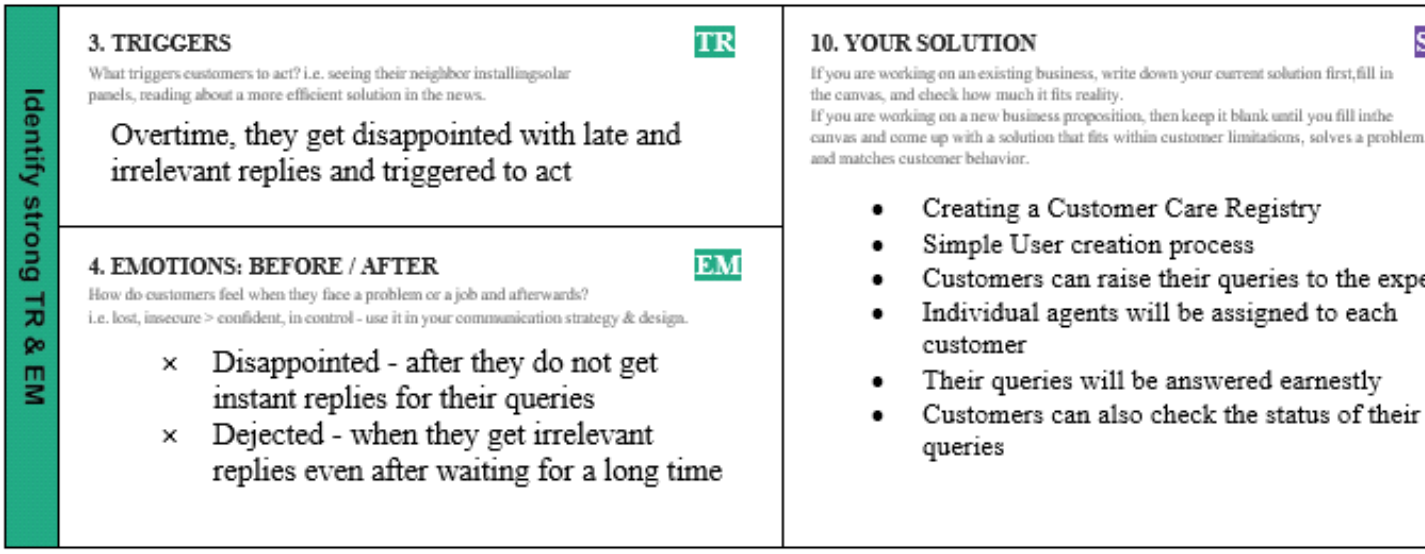
3.3 PROPOSED SOLUTION:

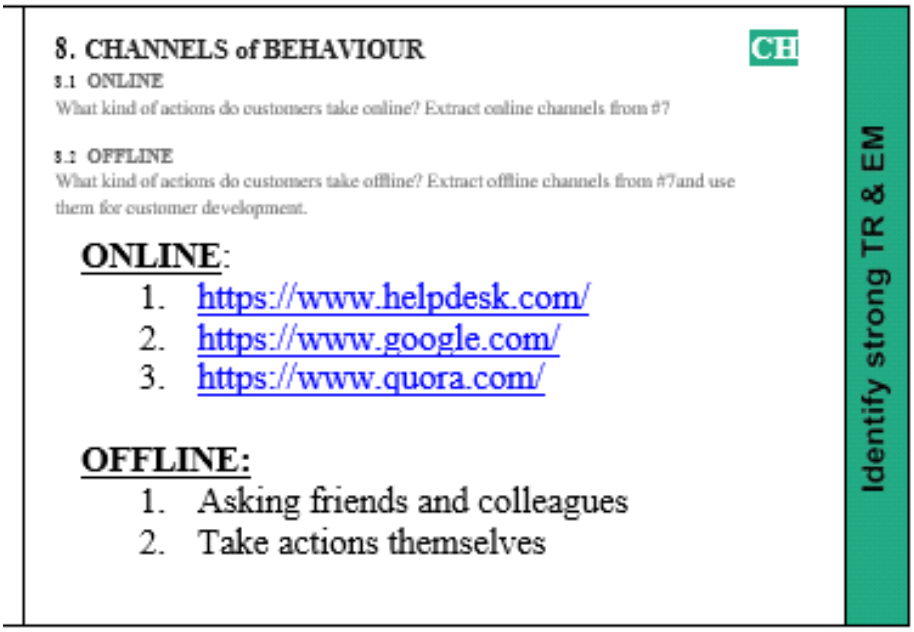
S.No.	Parameter	Description
•	Problem Statement (Problem to be solved)	I am Surya and I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don't have any reliable sources to clear my doubts in some of the products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

		this registry to their friends and colleagues at work. Naturally, the user base grows so does the number of queries answered. May be in the future, may be a cross-platform mobile application may be developed, making this customer care registry much more accessible to the users.

3.4 PROBLEM SOLUTION FIT:







4.REQUIREMENT ANALYSIS

- **FUNCTIONAL REQUIREMENT:**

Following are the functional requirements of the proposed solution:

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Signup form (customer)
FR-2	Forgot Password	Resetting the password by sending an OTP to

		user's mail (customer, agent, admin)
FR-3	User Login	Login through Login form (customer, agent, user)
FR-4	Agent creation (admin)	Create an agent profile with username, email and password
FR-5	Dashboard (customer)	Show all the tickets raised by the customer
FR-6	Dashboard (agent)	Show all the tickets assigned to the agent by admin
FR-7	Dashboard (Admin)	Show all the tickets raised in the entire system
FR-8	Ticket creation (customer)	Customer can raise a new ticket with the detailed description of his/her query
FR-9	Assign agent (admin)	Assigning an agent for the created ticket
FR-10	Ticket details (customer)	1. Showing the actual query, status, assigned agent details 2. Status of the ticket - OPEN, AGENT ASSIGNED, IN PROCESS, COMPLETE, CLOSED
FR-11	Address Column	Agent clarifies the doubts of the customer

- **NON-FUNCTIONAL REQUIREMENT:**

Following are the non-functional requirements of the proposed solution:

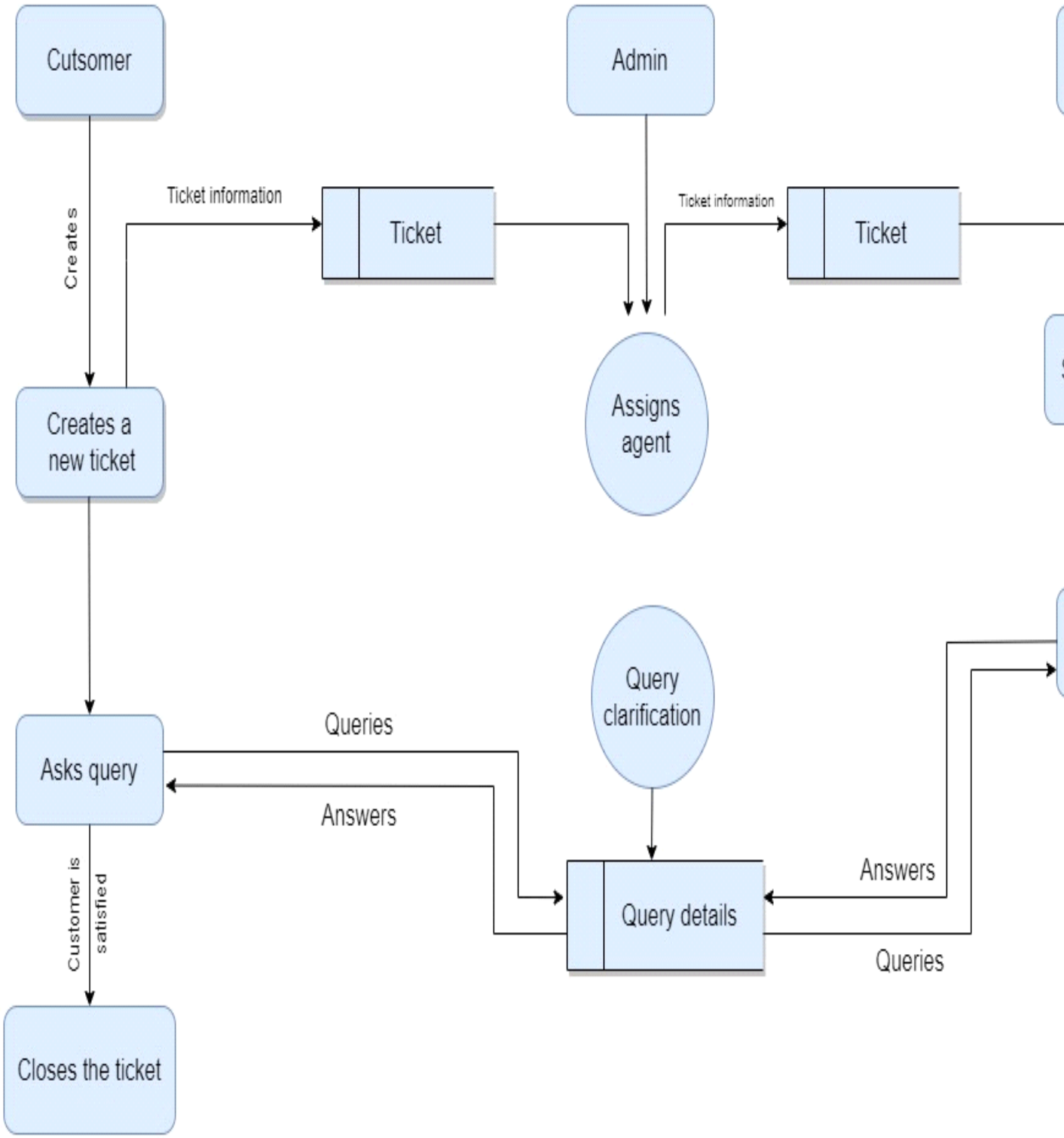
FR No.	**Non-Functional Requirement**	**Description**
NFR-1	**Usability**	Customers can use the application in almost all the web browsers. Application is with good looking and detailed UI, which makes it more friendly to use.
NFR-2	**Security**	Customers are asked to create an account for themselves using their email which is protected with an 8 character-long password, making it more secure.
NFR-3	**Reliability**	Customers can raise their queries and will be replied with a valid reply, as soon as possible,

		making the application even more reliable and trust-worthy.
NFR-4	**Performance**	Customers will have a smooth experience while using the application, as it is simple and is well optimised.
NFR-5	**Availability**	Application is available 24/7 as it is hosted on IBM Cloud
NFR-6	**Scalability**	In future, may be cross-platform mobile applications can be developed as the user base grows.

5.PROJECT DESIGN

5.1 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 depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored.

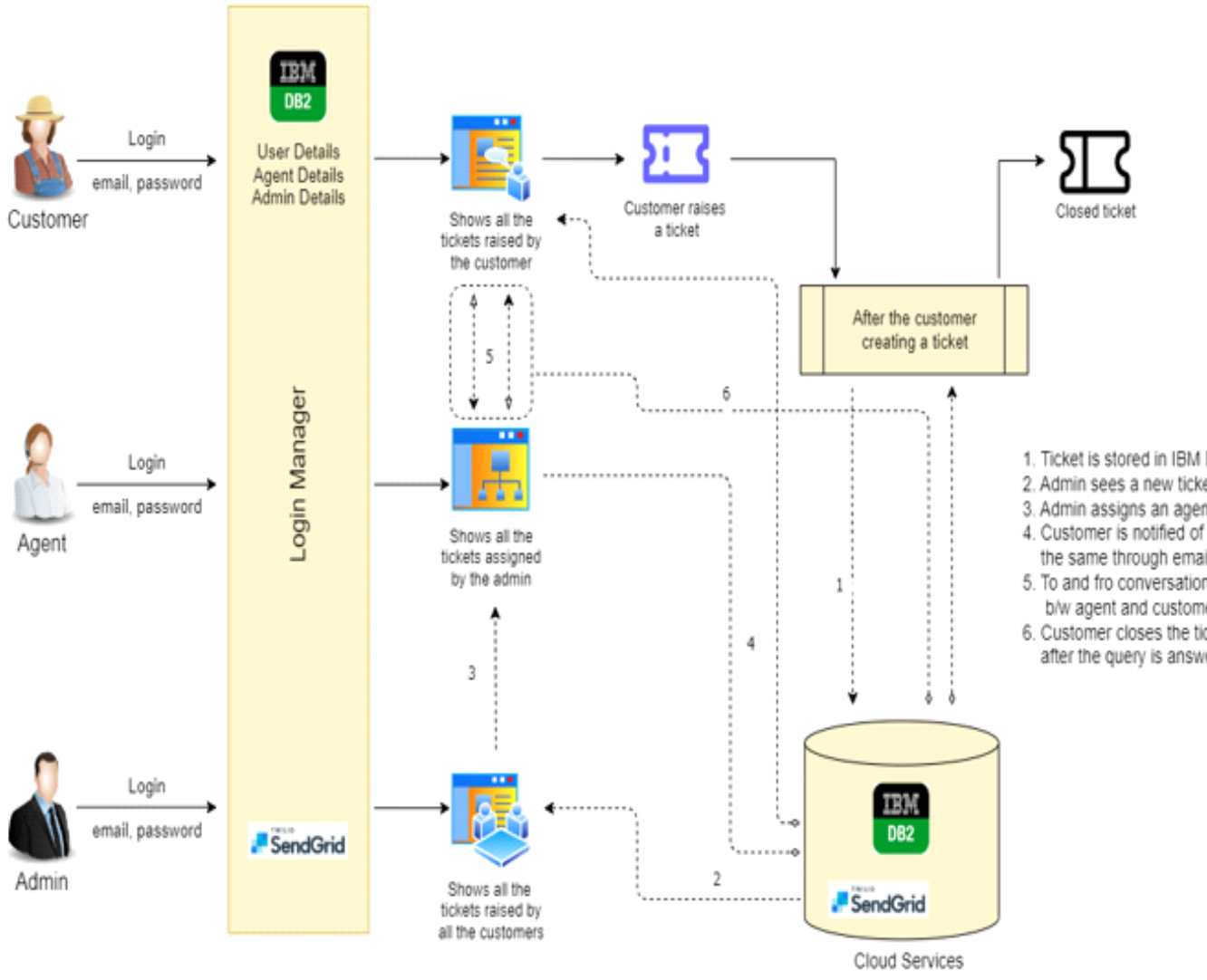

Cutsomer
Admin
Creates
Ticket information
Ticket
Ticket information
Ticket
Assigns agent
Creates a new ticket
Query clarification
Queries
Asks query
Answers
Answers
Query details
Customer is satisfied
Queries
Closes the ticket

5.2 SOLUTION AND TECHNICAL ARCHITECTURE:

a) Solution Architecture Diagram:

Solution architecture is a complex process – with many sub-processes – that bridges the gap between business problems and technology solutions. Its goals are to:

- Find the best tech solution to solve existing business problems.
- Describe the structure, characteristics, behaviour, and other aspects of the software to project stakeholders.
- Define features, development phases, and solution requirements.
- Provide specifications according to which the solution is defined, managed, and delivered.



b) TECHNICAL ARCHITECTURE:

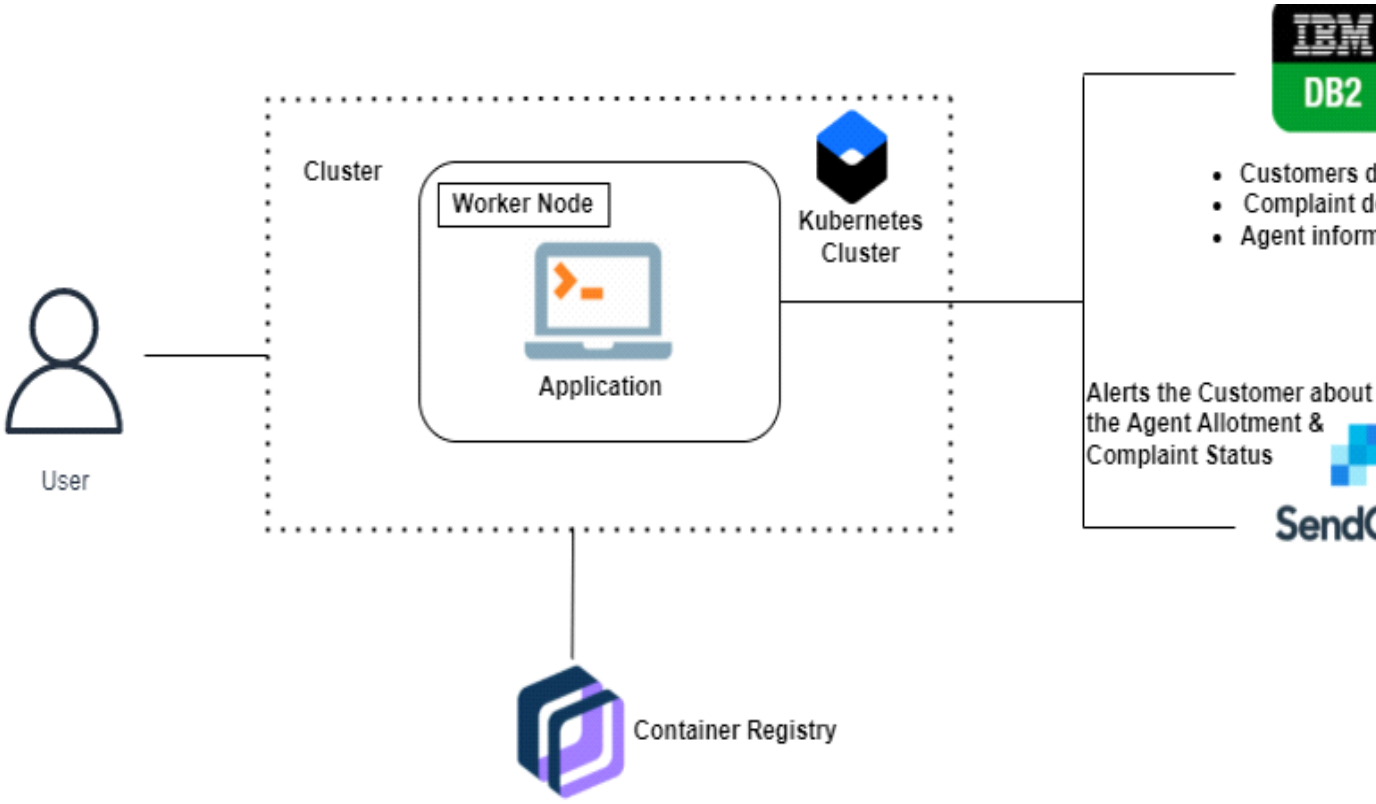


Table 1 : APPLICATION CHARACTERTICS

S No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask micro-web framework	Python, Jinja, WSGI
2.	Security Implementations	• All passwords are encrypted • Access control is implemented using Login Manager in Flask • Roles are defined in the SQL to prevent data manipulation and access	SHA-256 encryption, Flask, SQL

Table 2 : COMPONENTS AND TECHNOLOGIES

S No	Component	Description	Technology
•	User Interface	The user interacts with the Web UI (Login form, Signup form, Dashboard, Ticket status, Forget password page), chat bots	HTML, CSS, JavaScript

		(IBM Watson Assistant)	
•	Login Logic	The customer / agent enters their email and password, and their respective roles and click on the Login button. The data entered is collected and checked and verified for the corresponding entry in the IBM DB2 database. If everything is correspondence with the data in the IBM DB2, customer / agent logs in.	HTML forms, Python, SQL, IBM DB2
•	Register Logic	Customers registers in the application with their name, email, mobile number and password. The data entered is collected and stored in the IBM DB2 database. Once it is done, the customer is redirected to the Login page.	HTML forms, Python, SQL, IBM DB2
•	Agent Creation Logic	Admin creates an agent with the following credentials. Name, email, mobile, gender, username, password. The data is collected and stored in the database.	HTML forms, Python, SQL, IBM DB2
•	Ticket Creation Logic	Customer creates a new ticket in his dashboard, with the detailed description of his/her query (max of 150 characters). This ticket is then stored in the database with a unique ID and a foreign key as the customer ID.	HTML forms, Python, SQL, IBM DB2
•	Agent Assigning Logic	Agent sees all the newly created tickets in his/her dashboard. Agent then goes on to assign an agent for each ticket. The ticket status is updated in the IBM DB2 and then the customer who raised that ticket is notified through mail that as agent has been assigned.	HTML forms, Python, SQL, IBM DB2, SendGrid
•	Cloud Database	Stores all the details. Customer details, Agent details, Admin details, Ticket details.	IBM DB2 database
•	Object Storage	Stores some images in buckets. Used to display static images in the application.	IBM Cloud Object Storage
•	Chatbot (External API)	Used to guide customers, agents while logging in. Also, helps the customers while raising a ticket. Agents / Customers can interact with the chatbot and act right.	IBM Watson Assistant API
•	SendGrid (External	Used to notify the customers that an	SendGrid API, Python

	API)	agent has been assigned for their raised ticket. Also, for the agents and customers while resetting their passwords.	

5.3 USER STORIES:

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Web user)	Registration	USN-1	As a customer, I can register for the application byentering my email, password, and confirmingmy password.	I can access my account / dashboard	High	Sprint-1
	Login	USN-2	As a customer, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more	I get all the info needed in my dashboard	High	Sprint-1
	Ticket creation	USN-4	As a customer, I can create a new ticket with the detailed description of my query	I can ask my query	High	Sprint-2
	Address Column	USN-5	As a customer, I can have conversations with the assigned agent and get my queries clarified	My queries are clarified	High	Sprint-3
	Forgot password	USN-6	As a customer, I can reset my password by this option incase I forgot my old password	I get access to my account again	Medium	Sprint-4
	Ticket details	USN-7	As a customer, I can see the current status of my tickets	I get better understanding	Medium	Sprint-4
Agent (Web user)	Login	USN-1	As an agent, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-3
	Dashboard	USN-2	As an agent, I can see all the tickets assigned to me by the admin	I can see the tickets to which I could answer	High	Sprint-3
	Address Column	USN-3	As an agent, I get to have conversations with the customer and clear his/her queries	I can clarify the issues	High	Sprint-3
	Forgot password	USN-4	As an agent, I can reset my password by this	I get access to my account	Medium	Sprint-4

			option in case I forgot my old password	again		
Admin (Web user)	Login	USN-1	As an admin, I can login to the application by entering correct email and password	I can access my account / dashboard	High	Sprint-1
	Dashboard	USN-2	As an admin, I can see all the tickets raised in the entire system and lot more	I can assign agents by seeing those tickets	High	Sprint-1
	Agent creation	USN-3	As an admin, I can create an agent for clarifying the customer's queries	I can create agents	High	Sprint-2
	Assigning agent	USN-4	As an admin, I can assign an agent for each ticket created by the customer	Enables agent to clarify the queries	High	Sprint-2
	Forgot password	USN-4	As an admin, I can reset my password by this option in case I forgot my old password	I get access to my account again	Medium	Sprint-4

6.PROJECT PLANNING AND SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION:

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Priority	Team Members
Sprint-1	Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my password.	High	Keerthana S, Gokulkrishnan V
Sprint-1	Login	USN-2	As a customer, I can login to the application by entering correct email and password	High	Mukeswaran D , Jana shruthi M
Sprint-1	Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more	High	Keerthana S
Sprint-2	Ticket creation	USN-4	As a customer, I can create a new ticket with the detailed description of my query	High	Mukeswaran D
Sprint-3	Address Column	USN-5	As a customer, I can have conversations with the assigned agent and get my queries clarified	High	Jana shruthi M Gokulkrishnan V
Sprint-4	Forgot password	USN-6	As a customer, I can reset my password by this option in case I forgot	Medium	Jana shruthi M

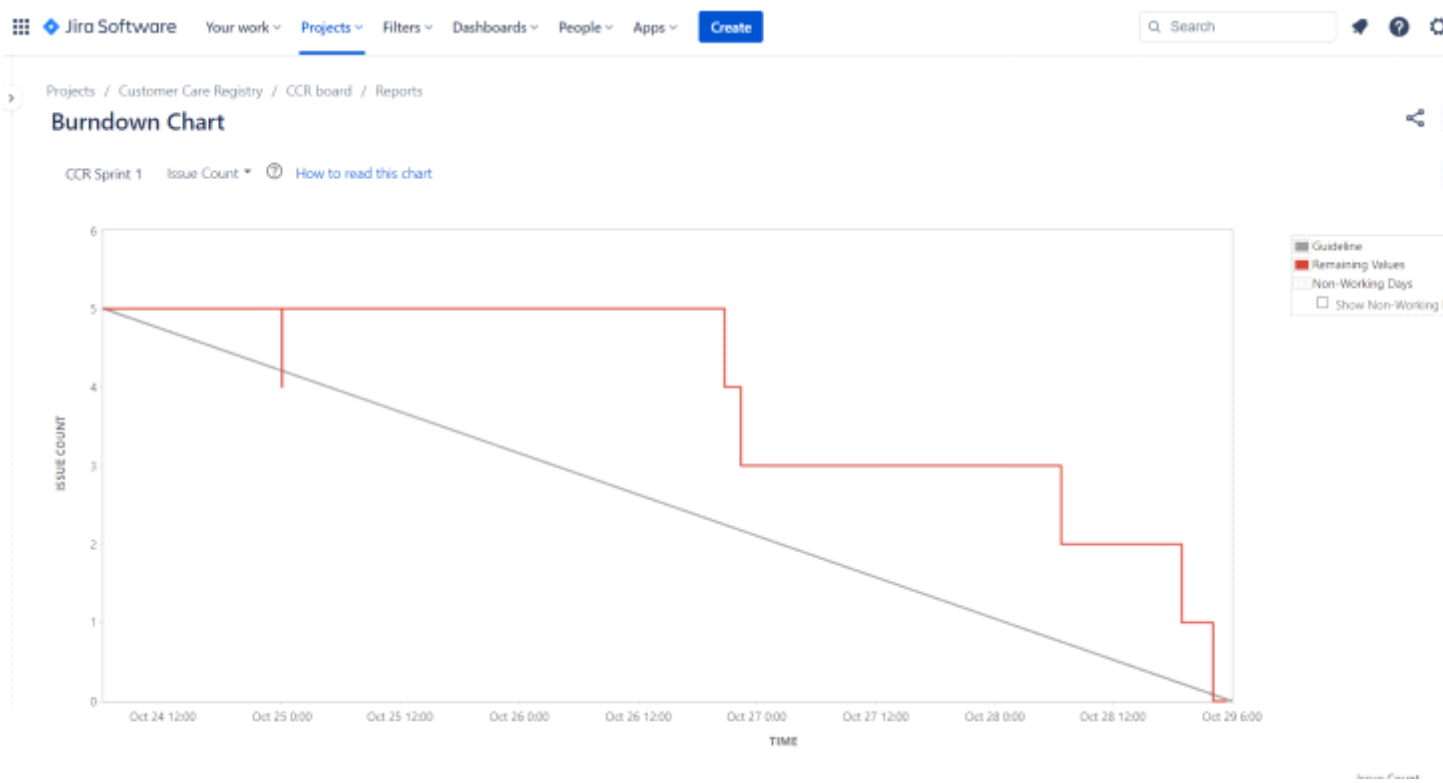
			my old password		
Sprint-4	Ticket details	USN-7	As a customer, I can see the current status of my tickets	Medium	Mukeswaran D Gokulkrishnan V
Sprint-3	Login	USN-1	As an agent, I can login to the application by entering correct email and password	High	Keerthana S
Sprint-3	Dashboard	USN-2	As an agent, I can see all the tickets assigned to me by the admin	High	Gokulkrishnan V
Sprint-3	Address Column	USN-3	As an agent, I get to have conversations with the customer and clear his/her queries	High	Keerthana S Jana shruthi M
Sprint-4	Forgot password	USN-4	As an agent, I can reset my password by this option in case I forgot my old password	Medium	Mukeswaran D Gokulkrishnan V
Sprint-1	Login	USN-1	As an admin, I can login to the application by entering correct email and password	High	Mukeswaran D

6.2 SPRINT DELIVERY SCHEDULE:

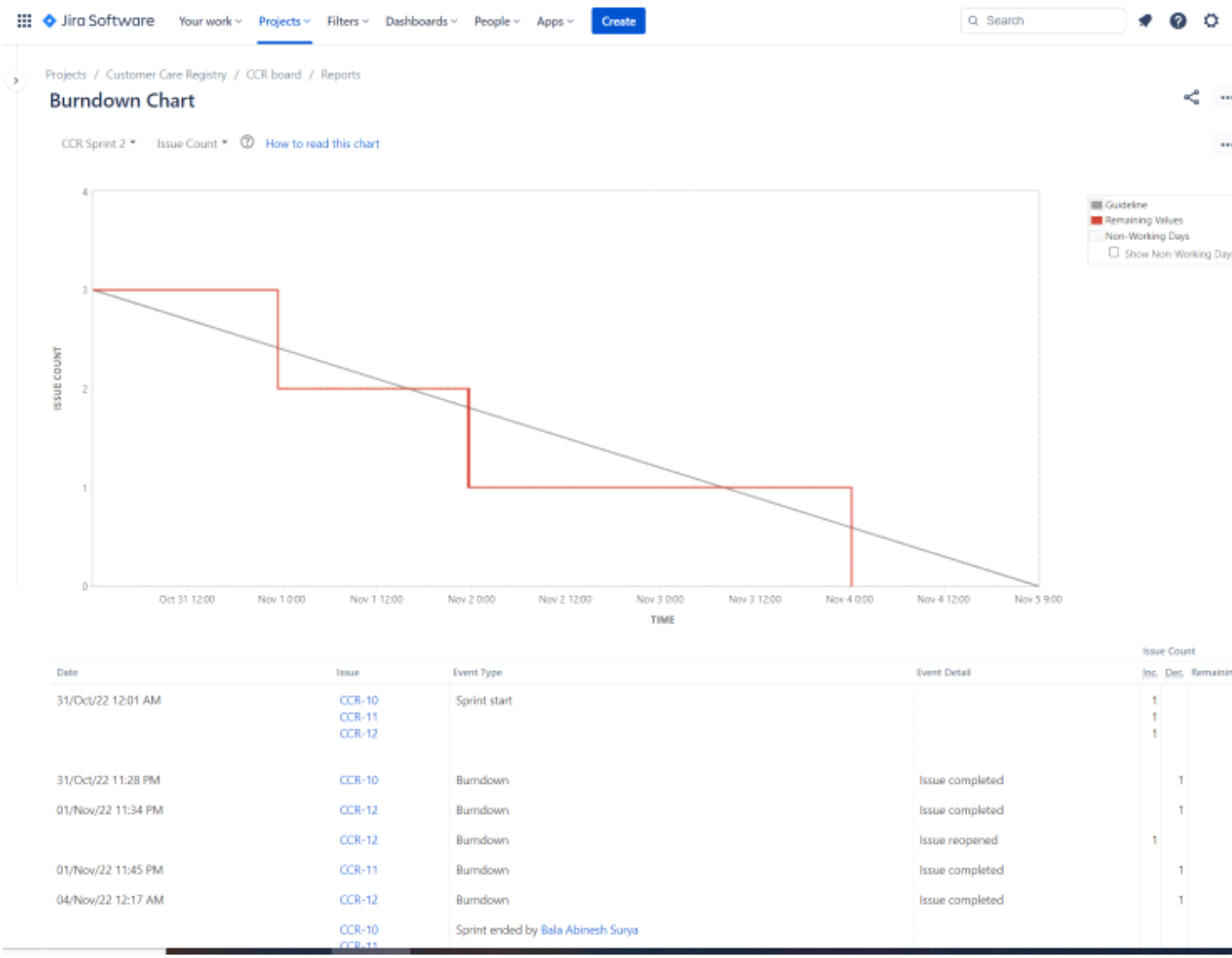
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	10	6 Days	24 Oct 2022	29 Oct 2022	10	29 Oct 2022
Sprint-2	7	6 Days	31 Oct 2022	05 Nov 2022	7	05 Nov 2022
Sprint-3	11	6 Days	07 Nov 2022	12 Nov 2022	11	12 Nov 2022
Sprint-4	8	6 Days	14 Nov 2022	19 Nov 2022	8	19 Nov 2022

6.3 REPORTS FROM JIRA:

Sprint 1:


Jira Software
Your work
Projects
Filters
Dashboards
People
Apps
Create
Search
Projects / Customer Care Registry / CCR board / Reports
Burndown Chart
CCR Sprint 1
Issue Count
How to read this chart
ISSUE COUNT
TIME
Guideline
Remaining Values
Non-Working Days
Show Non-Working

Sprint 2:



7.CODING AND SOLUTIONING:

7.1 FEATURE 1:

Python

- It is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significantindentation.
- Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural),object- oriented and functional programming.
- It is a high-level, general-purpose programming language. Its design

philosophy emphasizes code readability with the use of significantindentation.

- Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural),object- oriented and functional programming.

7.2 FEATURE 2:

FlASK

- Flask is a micro web framework written in Python. It is classified as a micro framework because it does not require particular tools or libraries.
- It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide commonfunctions.
- However, Flask supports extensions that can add application features as ifthey were implemented in Flask itself.
- Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several commonframework related tools.

7.3 FEATURE 3:

SENDGRID

- SendGrid is a cloud-based SMTP provider that allows you to send email without having to maintain email servers.
- SendGrid provides two ways to send email: through our SMTP relay or through our Web API. SendGrid provides client libraries in many languages.
- This is the preferred way to integrate with SendGrid. If you choose to use SendGrid without a client library, the Web API is recommended in most cases as it is faster, provides some benefit with encoding, and tends to be easier to use.
- SMTP provides many features by default, but is harder to setup.

7.4 DATABASE SCHEME

IBM DB2:

- A hybrid ANSI-compliant data virtualization tool for accessing, querying and summarizing data across the enterprise which:
- Provides a massively parallel processing (MPP) architecture Exploits Hive, H Base and Apache Spark concurrently for best-in-class analyticcapabilities
- Requires only a single database connection or query to connect disparate sources such as HDFS, RDMS, NoSQL databases, objectstores and Web HDFS
- Provides low latency support for ad-hoc and complex queries, high performance, and federation capabilities
- Understands dialects from other vendors and various products from Oracle, IBM® Db2® and IBM Netezza®
- Enables advanced row and column security

KUBERNATES-

- Kubernetes — also known as “k8s” or “kube” — is a container orchestration platform for scheduling and automating the deployment, management, and scaling of containerized applications.
- Kubernetes was first developed by engineers at Google before being open sourced in 2014. It is a descendant of Borg, a container orchestration platform used internally at Google. Kubernetes is Greek
- for *helmsman* or *pilot*, hence the helm in the Ku HYPERLINK "https://github.com/cncf/artwork/tree/master/projects/kubernetes" HYPERLINK "https://github.com/cncf/artwork/tree/master/projects/kubernetes" HYPERLINK "https://github.com/cncf/artwork/tree/master/projects/kubernetes"b HYPERLINK "https://github.com/cncf/artwork/tree/master/projects/kubernetes" HYPERLINK "https://github.com/cncf/artwork/tree/master/projects/kubernetes" HYPERLINK

"https://github.com/cncf/artwork/tree/master/projects/kubernetes"ernetes logo (linkresides outside IBM).

- Today, Kubernetes and the broader container ecosystem are maturing into a general-purpose computing platform and ecosystem that rivals — if not surpasses — virtual machines (VMs) as the basic building blocks of modern cloud infrastructure and applications.
- This ecosystem enables organizations to deliver a high- productivity Platform-as-a-Service (PaaS) that addresses multiple infrastructure-related and operations-related tasks and issues
- surrounding cloud-native HYPERLINK "https://www.ibm.com/in-en/cloud/learn/cloud-native" HYPERLINK "https://www.ibm.com/in-en/cloud/learn/cloud-native" HYPERLINK "https://www.ibm.com/in-en/cloud/learn/cloud-native" development so that development teams can focuss on coding and innovation.

8.TESTING

8.1 TEST CASES:

- The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product.
- It provides a way to check the functional it your components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectation and does not fail in an unacceptable manner.
- There are various types of tests. Each test type addresses a specific testing requirement

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	10	4	2	3	20
Duplicate	1	0	1	0	2
External	2	2	1	1	6
Fixed	4	1	1	10	16
Not Reproduced	0	0	0	0	0
Skipped	1	1	0	1	3
Won't Fix	0	2	2	0	4
Totals	24	14	13	26	51

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	9	0	0	9
Client Application	10	0	0	10
Security	1	0	0	1
Outsource Shipping	0	0	0	0
Exception Reporting	9	0	0	9
Final Report Output	9	0	0	9
Version Control	1	0	0	1

9.RESULTS

- **PERFORMANCE METRICES**

- Project metrics are used to track the progress and performance of a project.
- Monitoring parts of a project like productivity, scheduling, and scope makeit easier for team leaders to see what's on track.
- As a project evolves, managers need access to changing deadlines or budgetsto meet their client's expectations.

10.ADVANTAGES AND DISADVANTAGES:

a) ADVANTAGES:

- Feel good factor is high/enjoy going to work
- Improved reputation
- Fewer complaints
- Less stress for staff
- Greater jog security
- Improved team spirit, staff morale and motivation
- More enjoyable work atmosphere
- Greater staff loyalty and retention
- Improved communication within the business
- Greater chance of word of mouth advertising
- Strengthless reputation – enhances the image of the organization and helps to attract new customers
- Customers are satisfied and happy – they have confidence in what they are purchasing and feel highly valued by the organization

b) DISADVANTAGES:

- Long hold times can hamper the customer experience and lead to customer churn
- Customer service agents cannot handle multiple calls at the same time
- Phone support requires intensive agent training
- International or long-duration calls can prove to be expensive for our customers

Customers can share tweets or posts at any time of the day, even when your team is not available

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, JavaScript. It will be a ticket-based customer service registry. Customers can register into the application using their email, password, and a username. Then, they can login 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.

12.FUTURE SCOPE

- Upgrading the website that is more user-friendly which will help many users to access the website and also ensures that many customers complaints can be addedinto the community.
- Using agent balancer, it helps to handle multiple requests at the same time which will maintain the uptime of the website with negligible downtime.
- Ensures the faster and efficient communication between the agent and customers.
- The future of big data in healthcare will be determined by technologicalbreakthroughs from 2022 to 2030.

13.APPENDIX

- **SOURCE CODE:**

admin.html

```
{% extends 'base.html' %}
{% block head %}
<title>
Admin Dashboard
</title>
{% endblock %}
{% block body %}
<!-- things
div 1
welcome jetson, sign out
div 2
your complaints status
add new complaint -->
<br>
```

```
<!-- <br>
{% for i in range(11) %}
{{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
{{ j }}
{% endfor %}
<br>
{% endfor %} -->
<div class="fordashboardtop">
<div class="fordashboardtopelements1">
Welcome Admin,
</div>
<div class="fordashboardtopelements2">
<a href="/login"><button class="forbutton">Sign out</button></a>
</div>
</div>
<br>
<div class="outerofdashdetails">
<div class="fordashboarddetails">
<br>
<!-- table of customers complaints -->
<table class="fortable">
<thead>
</thead>
<tbody>
```

```
<tr>
<td class="pad">
<a href="/agents">Agent Details</a>
</td>
<td class="pad">
<a href="/tickets">Customer Ticket Details</a>
</td>
</tr>
</tbody>
</table>
<br>
</div>
</div>
{% endblock %}
```

base.html

```
<!DOCTYPE html>
<head>
<link rel="stylesheet" href="static/main.css"/>
{% block head %} {% endblock %}
</head>
<body>
{% block body %}
{% endblock %}
</body>
</html>
```

dashboard.html

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

```
<title>
Dashboard
</title>
{% endblock %}
{% block body %}
<!-- things
div 1
welcome jetson, sign out
div 2
your complaints status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
{{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
{{ j }}
{% endfor %}
<br>
{% endfor %} -->
<div class="fordashboardtop">
<div class="fordashboardtopelements1">
Welcome {{ name }},
</div>
<div class="fordashboardtopelements2">
```

```
<a href="/login"><button class="forbutton">Sign out</button></a>
</div>
</div>
<br>
<div class="outerofdashdetails">
<div class="fordashboarddetails">
<br>
<!-- table of customers complaints -->
<table class="fortable">
<thead>
<th>Complaint ID</th>
<th class="pad">Complaint Detail</th>
<th>Assigned Agent</th>
<th>Status</th>
<th>Solution</th>
</thead>
<tbody>
{% for i in complaints %}
<tr>
<td>
{{ i['C_ID'] }}
</td>
<td class="pad">
{{ i['TITLE'] }}
</td>
<td>
{{ i['ASSIGNED_AGENT'] }}
</td>
<td>
{% if i['STATUS'] == 1 %}
```

```
Completed
{% elif i['STATUS'] == 0 %}
Not completed
{% else %}
In progress
{% endif %}
</td>
<td>
{{ i['SOLUTION'] }}
</td>
</tr>
{% endfor %}
</tbody>
</table>
<br>
<center>
<div class="fordashboarddetails">
<button type="button" class="collapsible">Add new complaint
✚</button>
<div class="content">
<br>
<form action="/addnew" method="post">
<div class="forform">
<div class="textinformleft">
Title
</div>
<div class="textinformright">
<input type="name" name="title">
</div>
</div>
```

```
<div class="forform">
<div class="textinformleft">
Complaint
</div>
<div class="textinformright">
<textarea name="des"
style="border-radius: 1rem;width:
90%;height: 150%;background-color: black;color: white;"></textarea>
</div>
</div>
<br>
<br>
<div>
<button class="forbutton" type="submit"> Submit
</button>
</div>
</form>
<br>
</div>
</div>
</center>
</div>
</div>
{% endblock %}
```

Login.h

login.html

```
{% extends 'base.html' %}
{% block head %}
<link rel="stylesheet" href="main.css">
```

```
<title>
Login
</title>
{% endblock %}
{% block body %}
<div class="forpadding">
<!-- for box of the signup form -->
<div class="sign">
<div>
<p class="fortitle">
Sign In
</p>
<hr>
<form action="/login" method="post">
<div class="forform">
<div class="textinformleft">
Username
</div>
<div class="textinformright">
<input type="name" name="username">
</div>
</div>
<div class="forform">
<div class="textinformleft">
Password
</div>
<div class="textinformright">
<input type="password" name="pass">
</div>
</div>
```

```
<br>
<div>
<button class="forbutton" type="submit"> Sign In >></button>
</div>
</form>
<br>
<div>
New user? <a href="/signup">Sign up</a>
</div>
<br>
</div>
</div>
</div>
{% endblock %}
```

signup.html

```
<html>
  <head>
    <link rel="stylesheet" href="static/signup.css"/>
    <title>
      Sign Up
    </title>
  </head>
  <body>

    <div class="forpadding">
      <!-- for box of the signup form -->
```

```
<div class="sign">
   <div>
      <h1 class="fortitle">
         Register Now!!
      </h1>

      <form action="/signup" method="post">
         <div class="forform">
            <div class="textinformleft">
               Username
            </div>
            <div class="textinformright">
               <input type="name"name="username">
            </div>
         </div>
         <div class="forform">
            <div class="textinformleft">
               Name
            </div>
            <div class="textinformright">
               <input type="name" name="name">
            </div>
```

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

```
                    <input type="password"  name="pass">
                </div>
            </div>

            <div class="forform">
                <div class="textinformleft">
                    Re - enter Password
                </div>
                <div class="textinformright">
                    <input type="password" name="repass">
                </div>
            </div>
            <br>
            <div>
                <button class="forbutton" type="submit"> Sign up
>></button>
            </div>
        </form>
        <br>
        <div>
            {{msg}}
        </div>
```

```
                <br>
                <div>
                    <h2>Already have an account? <a href="/login">Sign
in</a></h2>
                </div>
                <br>
            </div>
        </div>
    </div>


  </body>
</html>
```

tickets.html

```
{% extends 'base.html' %}
{% block head %}
<title>
Agent Dashboard
</title>
{% endblock %}
{% block body %}
<!-- things
```

```
div 1
welcome jetson, sign out
div 2
your complaints status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
{{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
{{ j }}
{% endfor %}
<br>
{% endfor %} -->
<div class="fordashboardtop">
<div class="fordashboardtopelements1">
<h1>Welcome Admin</h1>
```

```
</div>
<div class="fordashboardtopelements2">
<a href="/login"><button class="forbutton">Sign out</button></a>
</div>
</div>
<br>
<div class="outerofdashdetails">
<div class="fordashboarddetails">
<br>
<!-- table of customers complaints -->
<table class="fortable">
<thead>
<th>Complaint ID</th>
<th class="pad">Username</th>
<th>Title</th>
<th>Complaint</th>
<th>Solution</th>
<th>Status</th>
</thead>
<tbody>
{% for i in complaints %}
<tr>
```

```
<td>{{ i['C_ID'] }}</td>
<td class="pad">
{{ i['USERNAME'] }}
</td>
<td>
{{ i['TITLE'] }}
</td>
<td>
{{ i['COMPLAINT'] }}
</td>
<td>
{{ i['SOLUTION'] }}
</td>
<td>
{% if i['STATUS'] == 1 %}
Completed
{% else %}
Not Completed
{% endif %}
</td>
</tr>
{% endfor %}
```

```
</tbody>
</table>
<br>
<center>
<div class="fordashboarddetails">
<button type="button" class="collapsible">Assign an agent
⚡</button>
<div class="content">
<br>
<form action="/assignagent" method="post">
<div class="forform">
<div class="textinformleft">
Complaint ID
</div>
<div class="textinformright">
<input type="name" name="ccid">
</div>
</div>
<div class="forform">
<div class="textinformleft">
<label for="agent">Choose an agent:</label>
```

```
</div>
<div class="textinformright">
<select name="agent" id="agent">
{% for i in freeagents %}
<option value={{ i['USERNAME'] }}>{{
i['USERNAME'] }}</option>
{% endfor %}
</select>
</div>
</div>
<br>
<br>
<div>
<button class="forbutton" type="submit"> Submit
</button>
</div>
</form>
<br>
</div>
</div>
</center>
```

```
</div>
</div>
{% endblock %}
```

main.css

```
body{
    margin: 0;
    padding: 0;
    font-family: montserrat;
    background-color: aquamarine;

    overflow: hidden;

}
h1{
  font-family: FontAwesome;
  color: rgb(0, 0, 0);

}
.textinformleft{
  font-size: 30px;
```

```
  font-family: FontAwesome;
  color: rgb(0, 0, 0);

}



table {
  border-collapse: collapse;
  border-spacing: 1;
  width: 100%;
  border: 2px solid rgb(255, 255, 255);
}

th, td {
  text-align: left;
  padding: 9px;
}
.forbutton{
  background-color: #2fdee4; /* Green */
  border: none;
  color: rgb(0, 0, 0);
```

```
  padding: 15px 32px;
  text-align: center;
  text-decoration: none;
  display: inline-block;
  font-size: 16px;
  margin: 4px 2px;
  cursor: pointer;
  -webkit-transition-duration: 0.4s; /* Safari */
  transition-duration: 0.4s;
  box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2), 0 6px 20px 0
rgba(0,0,0,0.19);
    }



    button{
      background-color: #2fdee4; /* Green */
      border: none;
      color: rgb(0, 0, 0);
      padding: 15px 32px;
      text-align: center;
      text-decoration: none;
```

```
    display: inline-block;
    font-size: 16px;
    margin: 4px 2px;
    cursor: pointer;
    -webkit-transition-duration: 0.4s; /* Safari */
    transition-duration: 0.4s;
    box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2), 0 6px 20px 0 rgba(0,0,0,0.19);
      }
```

appp.py

```
from flask import Flask, render_template, request, redirect, session, url_for
from flask import Flask, render_template, request, redirect, session
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=98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud;PORT=30875;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=wzl67403;PWD=GhnSbpHNQ0cNxoMe;,'', '')

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=824dfd4d-99de-440d-9991-629c01b3832d.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=30119;SECURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=qvk70423;PWD=saDlGasU4iQy1yvk;'', '', '')

print("connected..")

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

@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 = ? 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:
    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('/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, domain)
    ibm_db.bind_param(stmt, 6, password)
    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 =? 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")
```

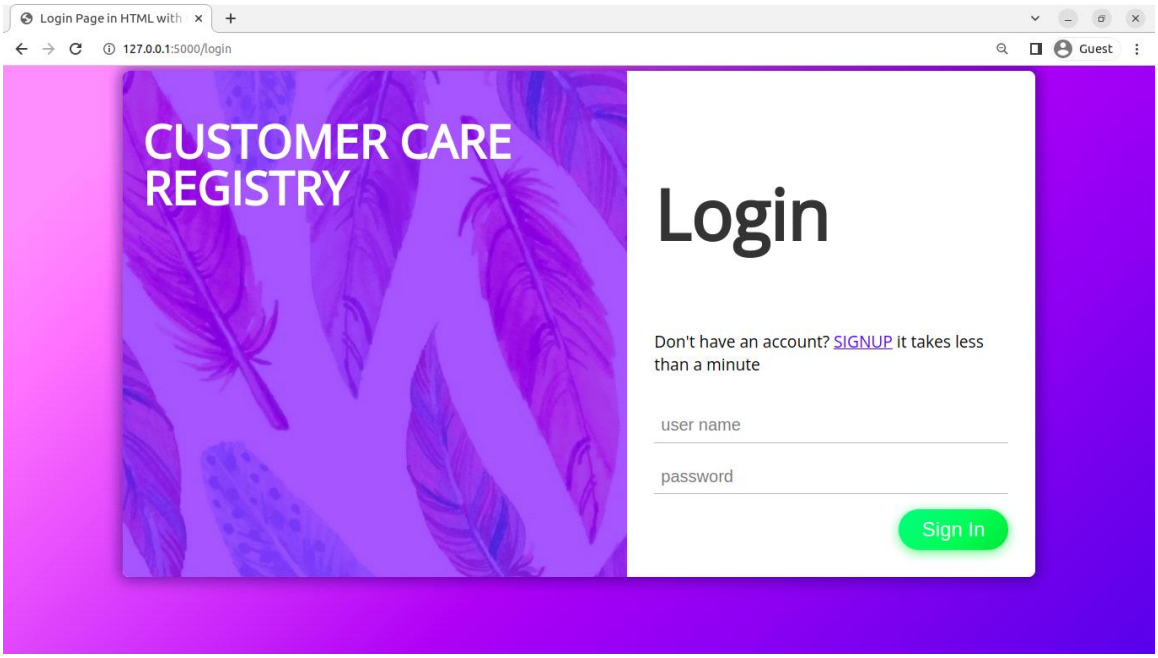
```
        sql = "select * from complaints where assigned_agent = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            complaints.append(dictionary)
            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)
```

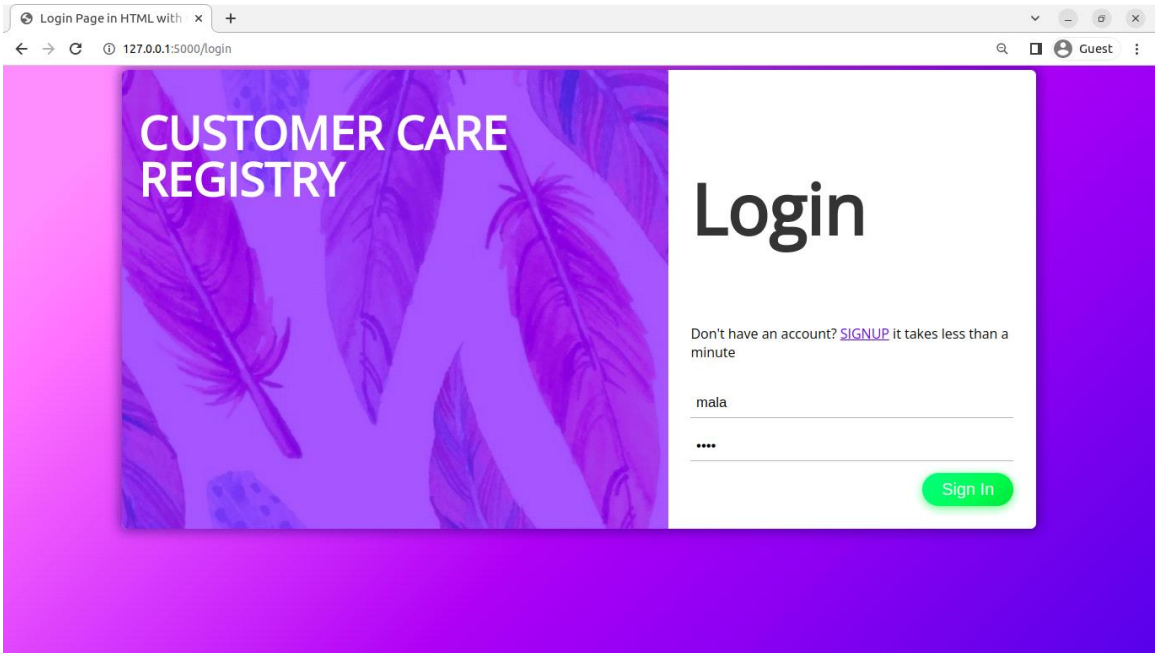
```
            ibm_db.bind_param(stmt, 2, ccid)
            ibm_db.execute(stmt)
            sql = "update agents set status =1 where username = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant update")
        return redirect(url_for('tickets'))
@app.route('/about')
def about():
    return render_template('about.html')
@app.route('/privacyterms')
def privacyterms():
    return render_template('privacyterms.html')
if __name__ == "__main__":
    app.run(debug=True)
```

SCREENSHOT:

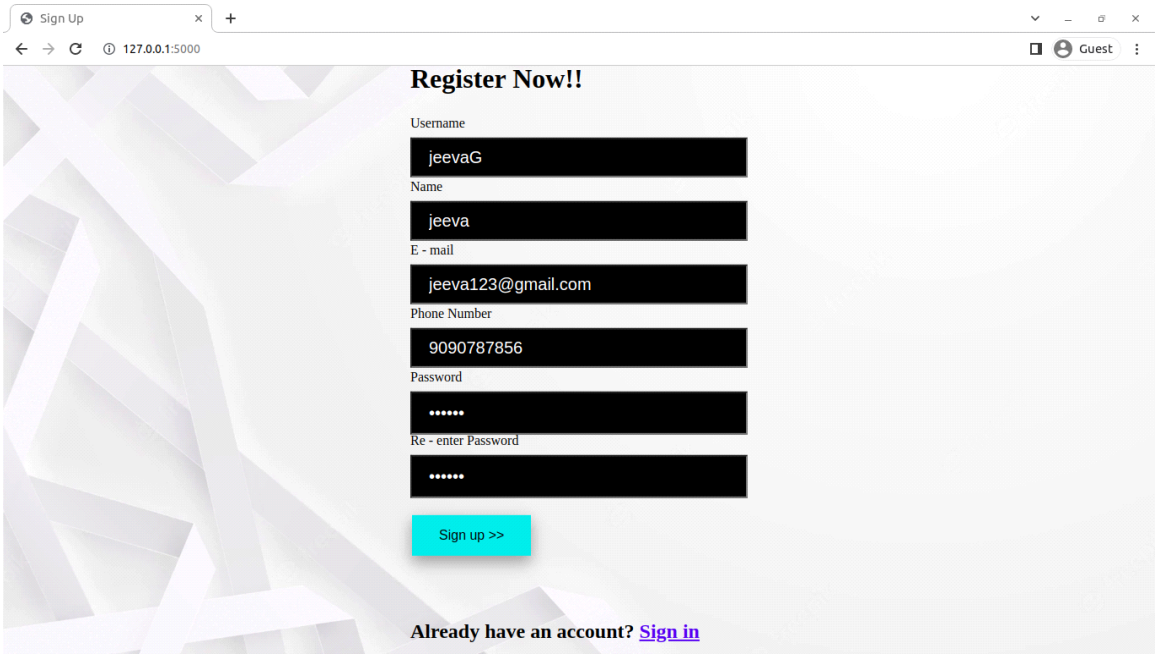
LOGIN PAGE



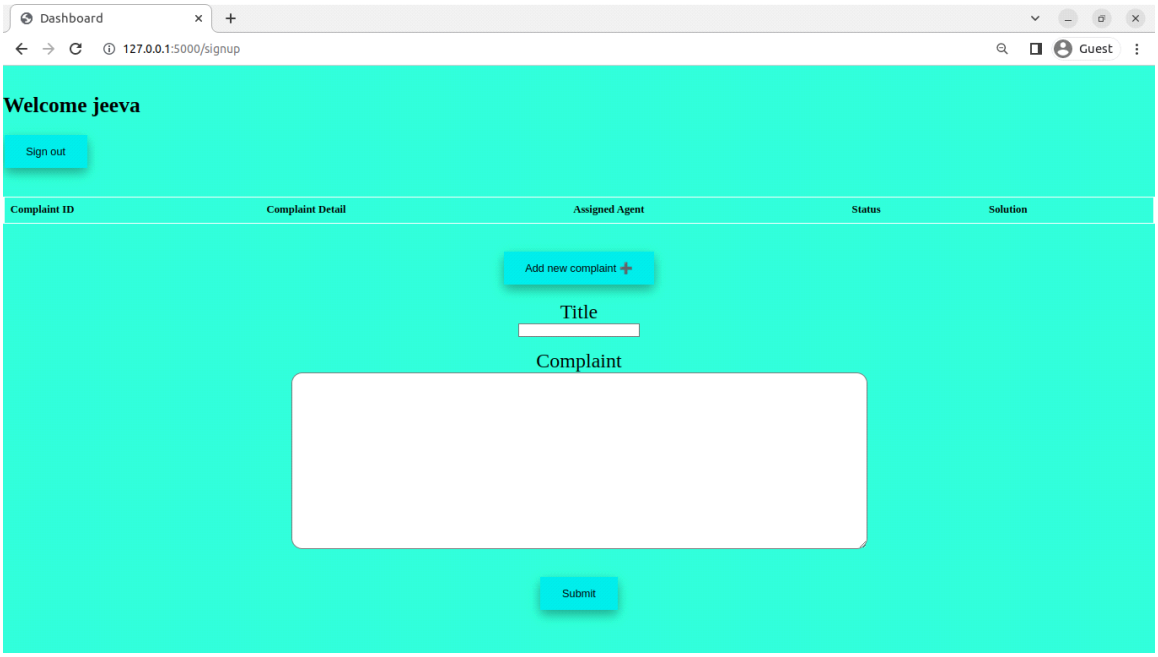
AGENT LOGIN



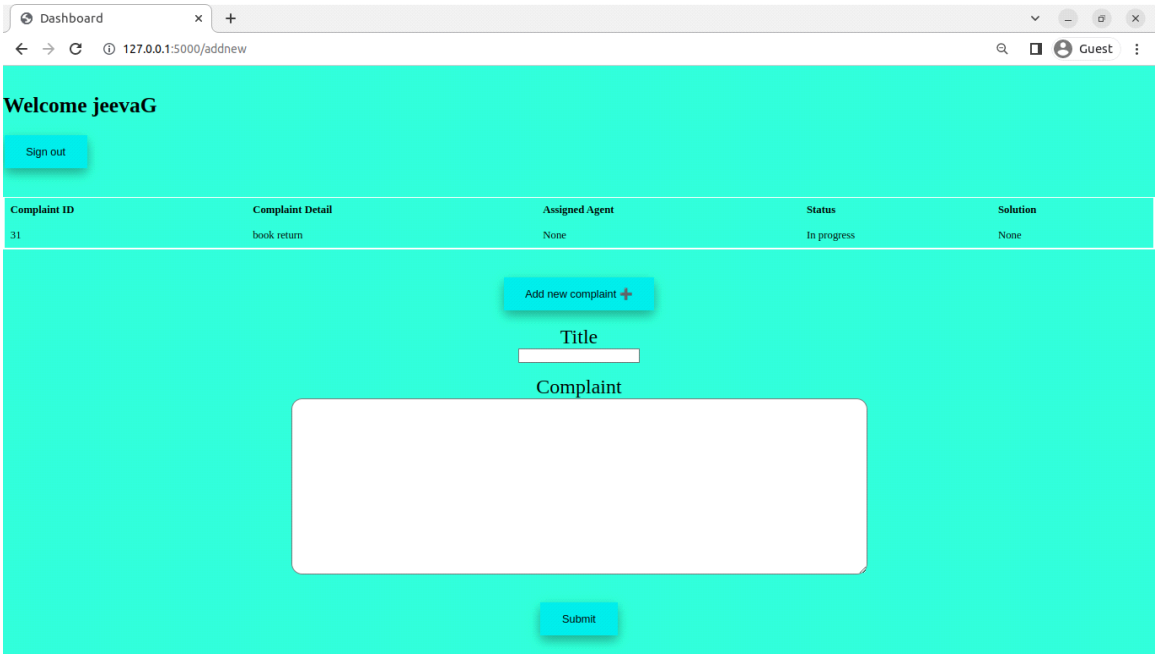
SIGNUP PAGE



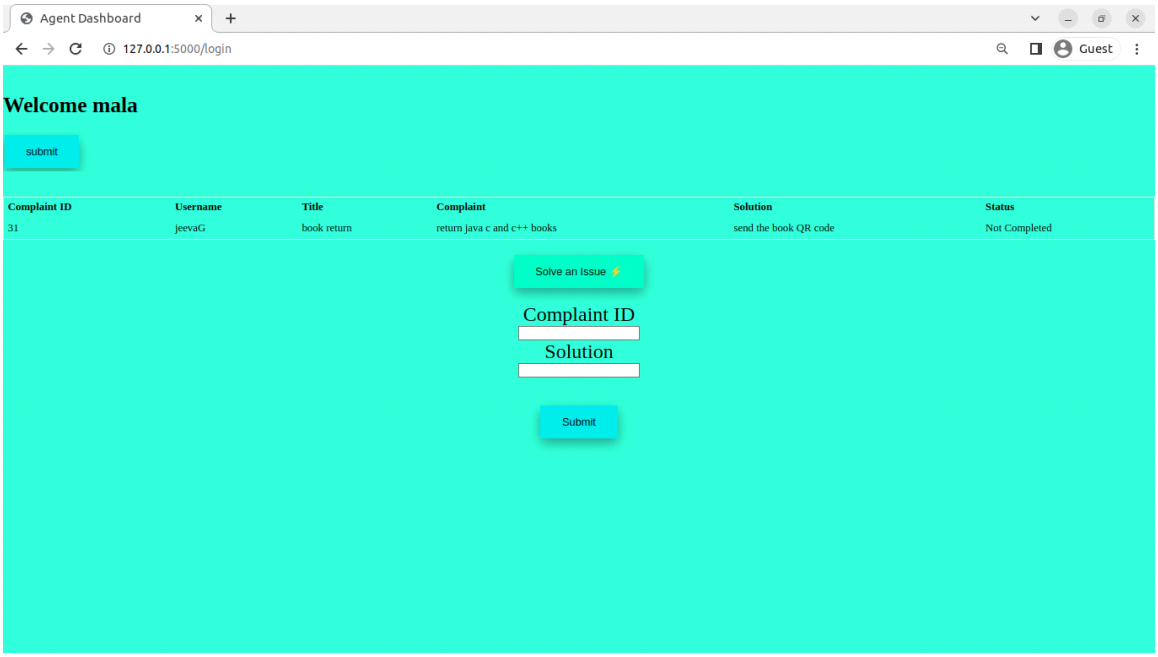
USER DASHBOARD



USER DASHBOARD WITH MAKE A COMPLAINT

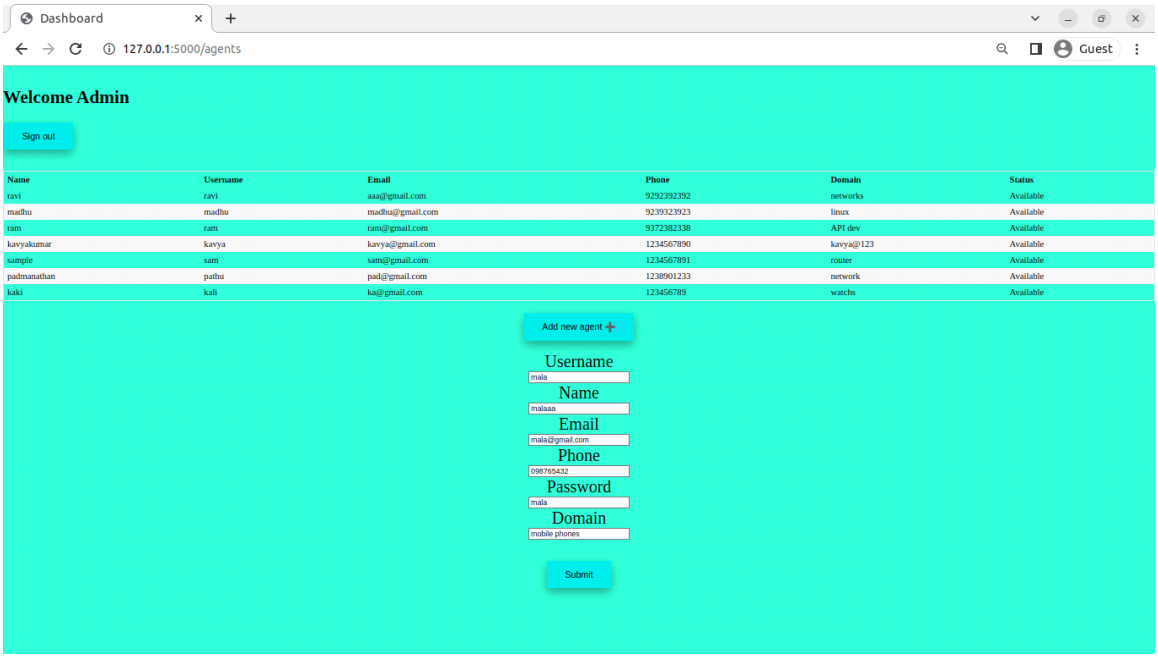


CUSTOMER VIEW THE SOLUTION



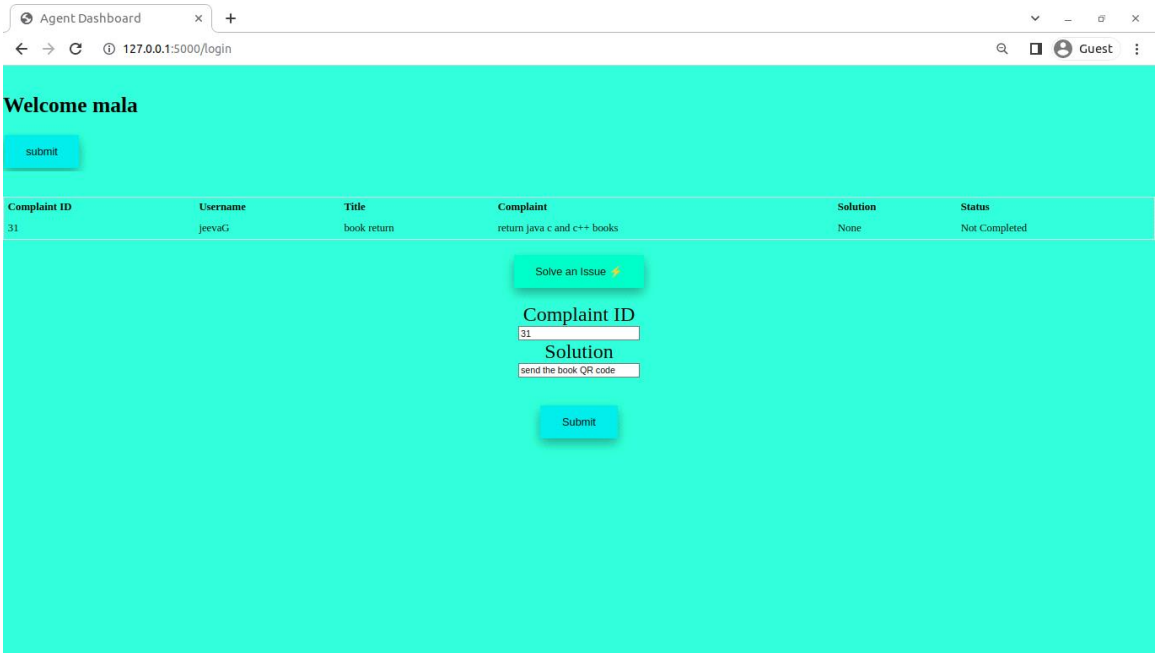
ADMIN DASHBOARD

ADMIN ADD NEW AGENT



ASSIGN A AGENT

AGENT PROVIDE THE SOLUTION



13.2 GitHub AND PROJECT DEMO LINK

a) GitHub

https://github.com/IBM-EPBL/IBM-Project-46335-1660745571

b) PROJECT DEMO LINK:

https://drive.google.com/file/d/1CA_6ocDxX5h_hyNeXTW8zR15NmCKpOL5/view?usp=drivesdk