CUSTOMER CARE REGISTRY

Submitted by

K.THANGA ESAKKIAMMAL (952319205047)

S.SAKTHESWARI (952319205037)

T.DHANUSIYA (952319205008)

G.MAHESHWARI (952319205021)

In partial fulfillment for the award of the degree

Of

BACHELOR OF ENGINEERING

In

INFORMATION TECHNOLOGY ENGINEERING

PSN ENGIREENING COLLEGE, MELATHIDIYOOR, THIRUNELVELI.

ANNA UNIVERSITY : : CHENNAI 600 025

NOVEMBER 2022

ANNA UNIVERSITY : : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this report titled "**CUSTOMER CARE RESGISTRY**" is the bonafide work of **" K.THANGA ESAKKIAMMAL (952319205047), ,S.SAKTHESWARI**

(952319205037), T.DHANUSIYA (952319205008), G.MAHESHWARI (952319205021), who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Mr.ROBIN JESUBALAN M.E.	**Mr.S.PITCHAIAHME.**
Head of the department,	**SPOC**
Department of IT,	Department of ECE,
Melathediyoor,	Melathediyoor,
Tirunelveli-627152	Tirunelveli-627152

Submitted for the Project work and Viva-Voce examination held on ……

MENTOR	**EVALUATOR**
Mrs.S. NANTHNI.CSE	Mrs.S.MARIYAMMAL M.E

Project Report Format

GitHub & Project Demo Link

1.INTRODUCTION

1.1Project Overview

The customer Service Dask is a web based project. customer service also known as client service is the provision of service to customers. Its significance varies by product, industry and domain. In many cases customer service is more important if the information relates to a service as opposed to a customer. Customer service may be provided by a service representatives customer service is normally an integral part of a company value proposition. An online comprehensive customer care solution is to manage customer interaction and complaints with the service providers over phone or through and e-mail. The system should have capability to integrate with any service provider from any domain or industry like banking, telecom, insurance,etc.

1.2 Purpose

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 vital, but there are subtle differences in how they are implemented. High-quality customer care is proactive. The needs of customers throughout the buyer's journey are anticipated, making customers feel supported. That, in turn, helps create an emotional connection between the customer and the company.

2.LITERATURE SURVEY

2.1 Existing problem

The existing system is a semi-automated at where the information is stored in the form of excel sheets in disk drives. The information sharing to the volunteers, group members, etc. is more critical in this system. Tracking the member's activities and progress of the work is a tedious job here. This system cannot provide the information sharing by 168 days.

2.2 References

Customer Approved Quote Deck

Customer Reference One Page Resource

Customer Insight Page (Case Study, Blog, Quote and Logo Process)

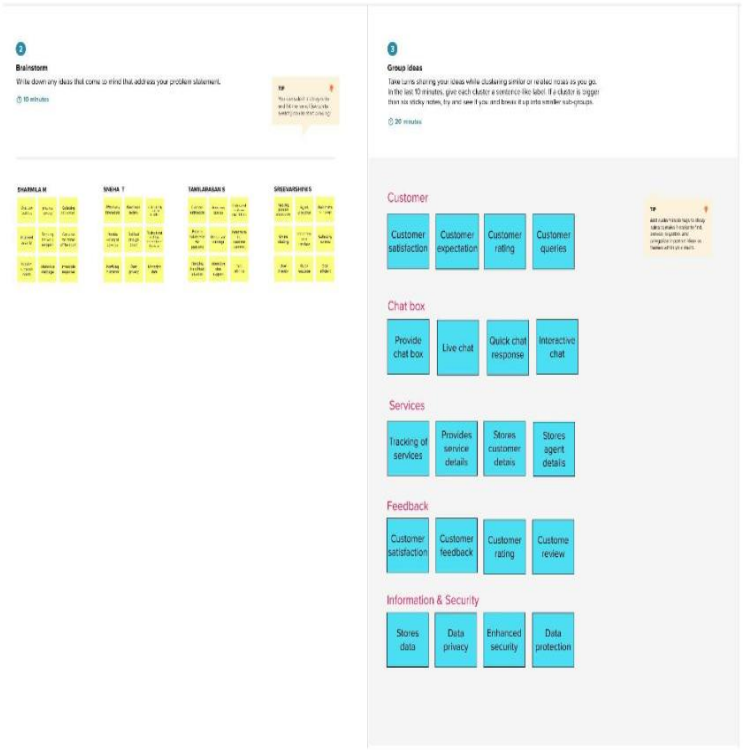
Customer Case Studies - Customers Page

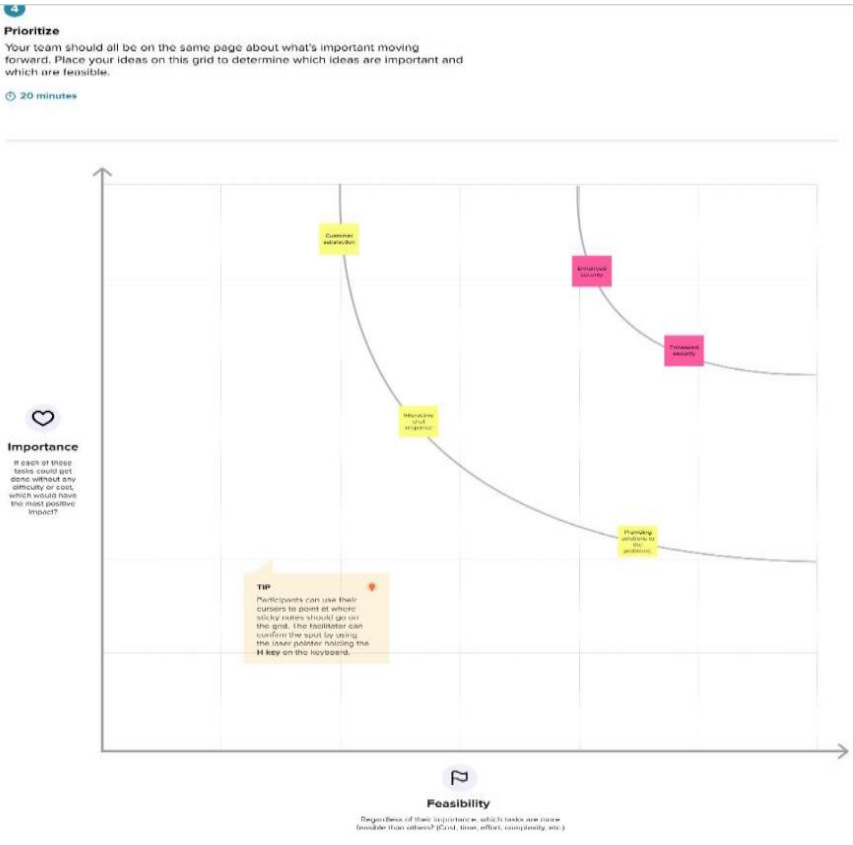
2.3 Problem Statement Definition

Customer care is more than just providing great customer service. It's a proactive approach to providing information, tools, and service to customers at each point they interact with a brand. For organizations, and for product and design teams, there can be a number of reasons why a product could fail. But not taking the time to consider a customer's conditions and their current situation could potentially harm your product's future. By working with a problem statement you can make sure you are defining a customer's experience and attempting to transform your product for the better.

3.IDEATION&PROPOSED SOLUTION

3.2 Ideation & Brainstorming

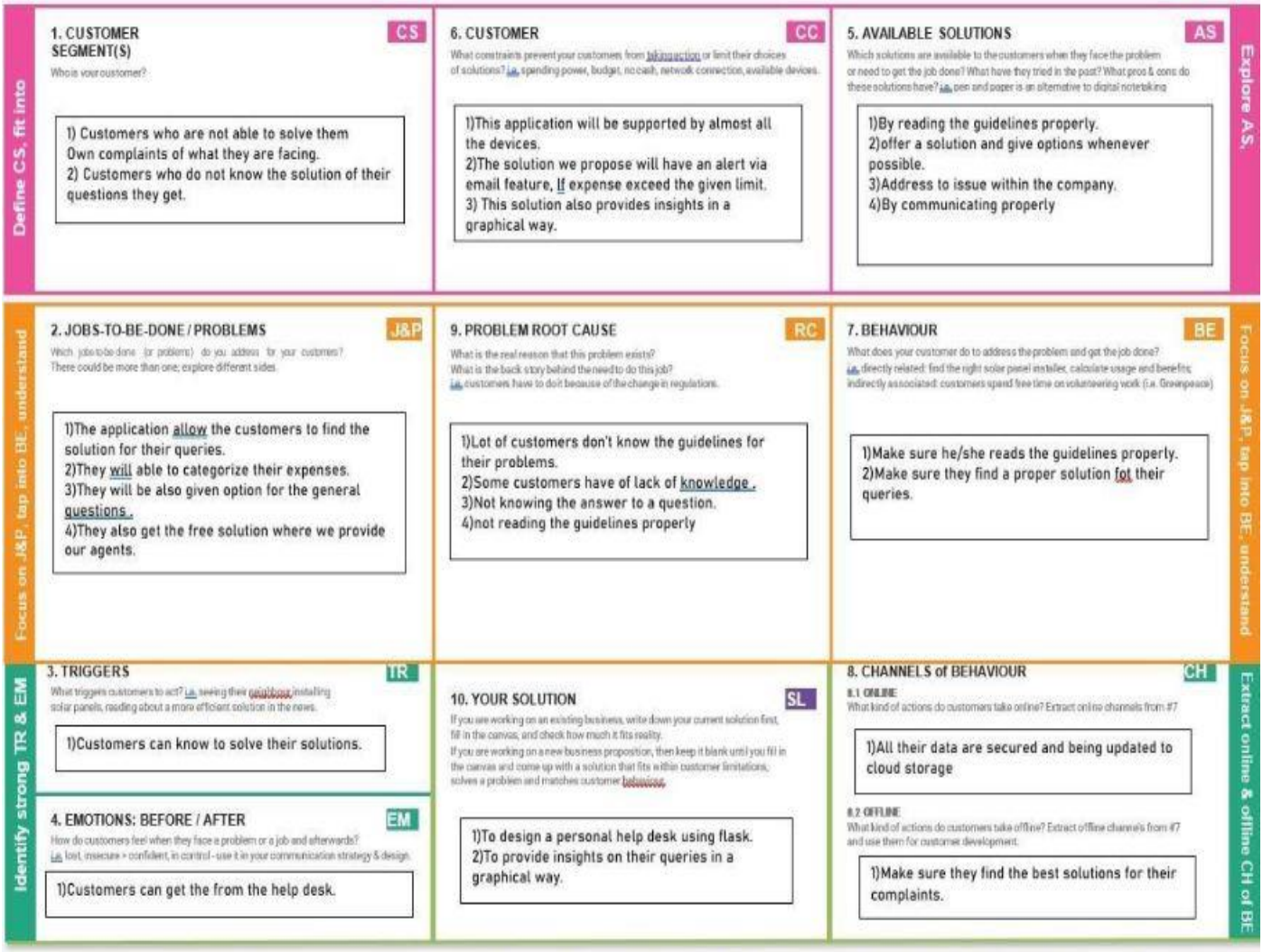

2
Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.
20 minutes
Customer
Customer satisfaction
Customer expectation
Customer rating
Customer queries
Chat box
Provide chat box
Live chat
Quick chat response
Interactive chat
Services
Tracking of services
Provides service details
Stores customer details
Stores agent details
Feedback
Customer satisfaction
Customer feedback
Customer rating
Custome review
Information & Security
Stores data
Data privacy
Enhanced security
Data protection


4
Prioritize
Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.
20 minutes
Importance
If each of these tasks could get done without any difficulty or cost, which would have the most positive impact?
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.
Feasibility

3.3 Proposed Solution

S.No	Parameter	Description
1.	Problem Statement (Problem to besolved)	Problem phase describes that the customer care is more than just providing great customer service. It's a proactive approach to providing information, tools, and services to customers ateach point they interact with a brand. For organizations, and for product and design teams, there can be a number of reasons why a product could fail. But not taking the time to consider a customer's conditions and their current situation could potentially harm your product's future. By working with a problem statement you can make sure you are defining a customer's experience and attempting totransform your product for the better. So the problem statement mainly defines to solve customer issues using Cloud Application Development.
2.	Idea / Solution description	Solution phase describes the web application that has been developed to help the customer in processing their complaints. The customers can raise the ticket with a detailed description of the issue. An Agent will be assigned to the Customer to solve the problem. Whenever the agent is assigned to a customer they will be notified withan email alert. Customers can view the status of the ticket till the service is provided.
3.	Novelty / Uniqueness	Customer care registry provides instant replyand the assigned work can be tracked at any time and provides tutorial for website.
4.	Social Impact / Customer Satisfaction	Customer care registry provides directcommunication between admin and user and provides variety of services.
5.	Business Model (Revenue Model)	Customer care registry can be linked with industrial organizations to provide customer care support.
6.	Scalability of the Solution	Customer care registry provides an environment which has both time and cost efficient.

3.4 Problem Solution fit



4. REQUIREMENT ANALYSIS

4.1Functional requirement

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	User Registration	Registration through Form Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email

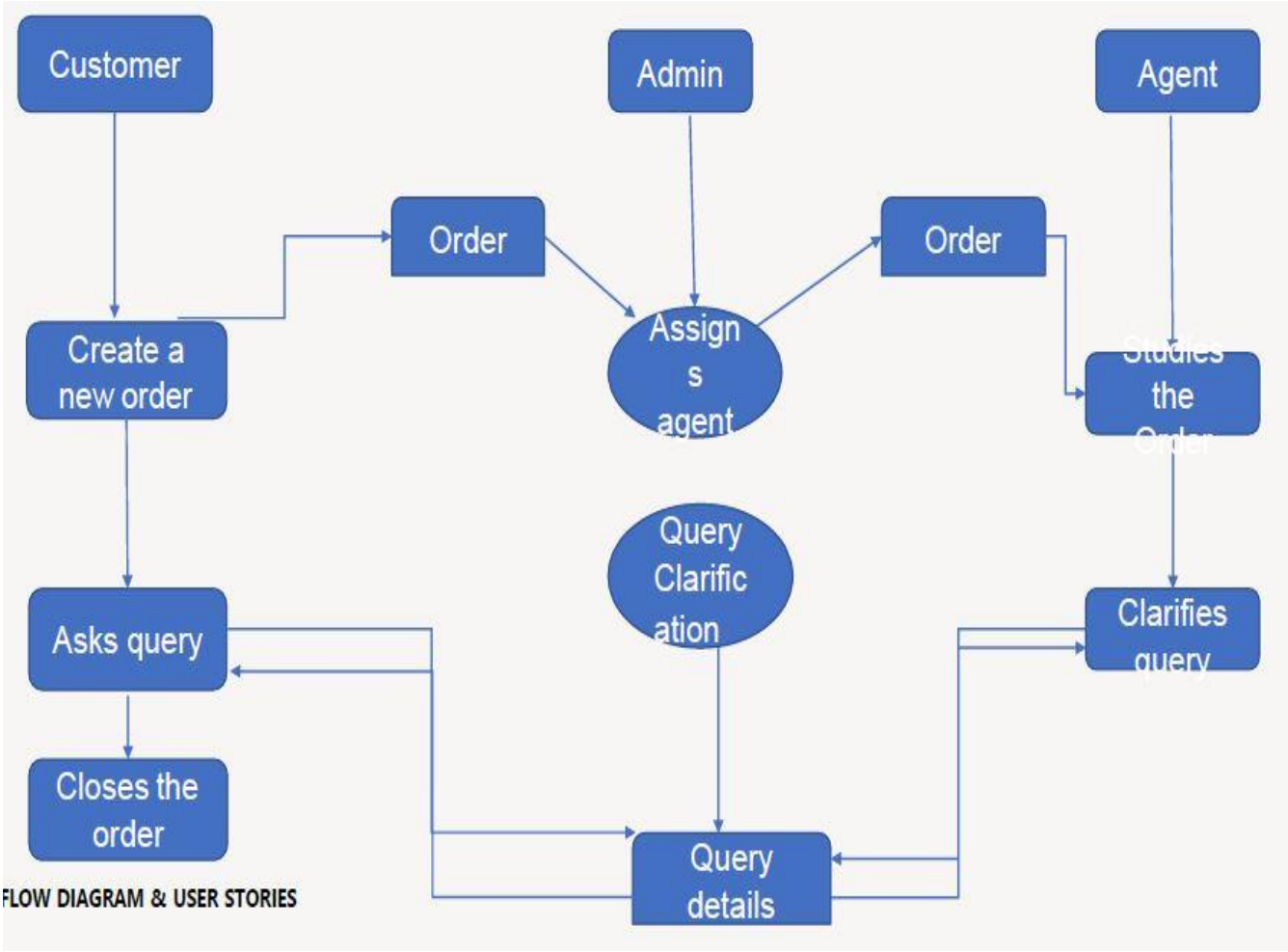
		Confirmation via OTP
FR-3	User Login	Confirmation of Gmail/username Confirmation of password
FR-4	Cyber attack	Immediate system shutdown in case of any malware Attack
FR-5	Backup and storage	Patient data can be backed up into cloud server. Backup can be scheduled for regularity.
FR-6	Prediction generation	The analysis and prediction can be downloaded in various formats.

4.2 Non Functional requirements

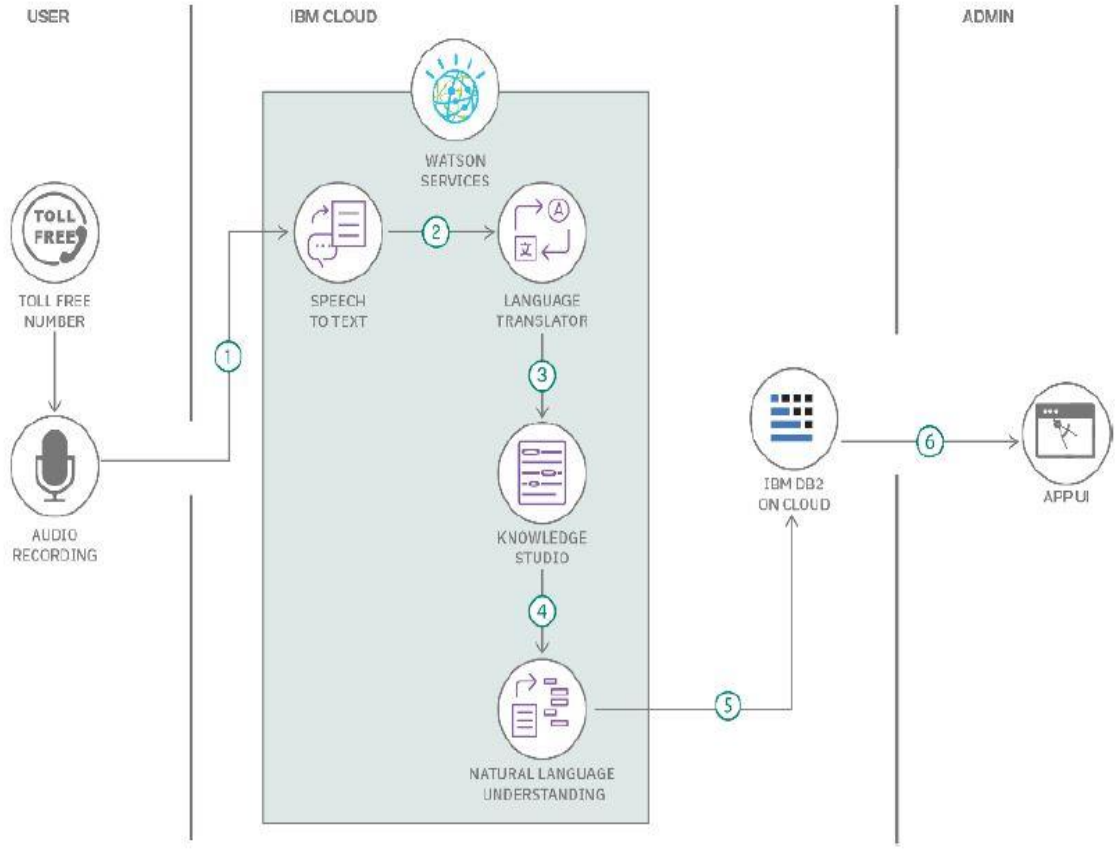
FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	Clear user interface for easy navigation which enables easy understanding of the contents of the product.
NFR-2	**Security**	Security of the patient data will be ensured by use of various measures like firewall, security questions and account locking in case of multiple incorrect passwords.
NFR-3	**Reliability**	Product will have same efficiency even after extensive use. Percentage of probability of failure will be less. Time between critical failures will be high.
NFR-4	**Performance**	Each page of the product will load within 2 secs. Prediction will take less than 5 secs and the generation of the analysis report will be less than 3 secs.
NFR-5	**Availability**	All product modules will be highly available and can be accessed with PC and internet.
NFR-6	**Scalability**	The scalability will be to improve 10% of annual patient in the hospital

5. PROJECT DESIGN

5.1Data Flow Diagrams



5.2 Solution & Technical Architecture



5.3 User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile Use)	Regostratin	USN -1	As a customer, I can register for the application by entering my email, password, and	can access my account / dashboard	High	Sprint -1

			confirming my password			
	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 orders raised by me.	I get all the info needed in my dashboard	Low	Sprint -2
	Order Creation	USN -4	As a customer, I can place my order with the detailed description of my query	I can ask my query	Medium	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 agian	Medium	Sprint -4
	Order Details	USN -7	As a Customer ,I can see the current stats of order.	I get abetter 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	can access my account / dashboard	High	Sprint -3
	Dashboard	USN -2	As an agent, I can see the order details assigned to me by admin.	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/er dobuts	I can clarify the issues	High	Sprint -3

	Forgot Password	USN -4	As an agent 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 & SCHEDULING

6.1 Sprint Planning &Estimation

Sprint	User Type	Functional Requirement (Epic)	User Story Number	User Story/Task	Story Points	Priority	Team Members
Sprint -3	Agent (Web User)	Login	USN -1	As an agent, I can login to the application by entering correct email and password	2	High	T.Dhanusiya
Sprint – 3		Dashboard	USN -2	As an agent, I can see all the tickets assigned to me by the admin	3	High	K.Thanga esakkiammal
Sprint -3		Address Column	USN -3	As an agent, I get to have conversations with the customer and clear his/her queries	3	High	S.Saktheswari
Sprint -4		Forgot password	USN -4	As an agent, I can reset my password by this option in case I forgot my old password	2	Medium	G.Mageshwari
Sprint -1	Admin (Web User)	Login	USN -1	As an admin, I can login to the application by entering correct email and password	1	High	S.Saktheswari

Sprint -1		Dashboard	USN -2	As an admin, I can see all the tickets raised in the entire system and lot more	3	High	K.Thanga esakkiammal
Sprint -2		Agent creation	USN -3	As an admin, I can create an agent for clarifying the customer's queries	2	High	S.Saktheswari
Sprint -2		Assigining agent	USN -4	As an admin, I can assign an agent for each ticket created by the customer	3	High	G.Mageshwari
Sprint -4		Forgot password	USN -4	As an admin, I can reset my password by this option in case I forgot my old password	2	Medium	T.Dhanusiya

6.2 Sprint Delivery Schedule

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

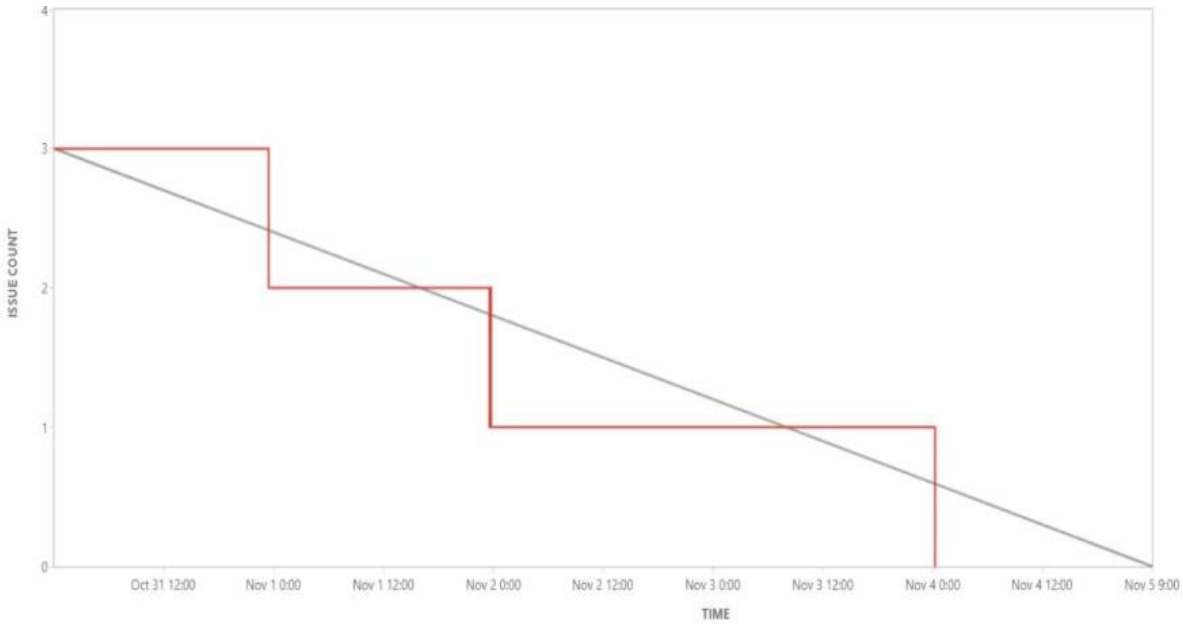
Velocity:

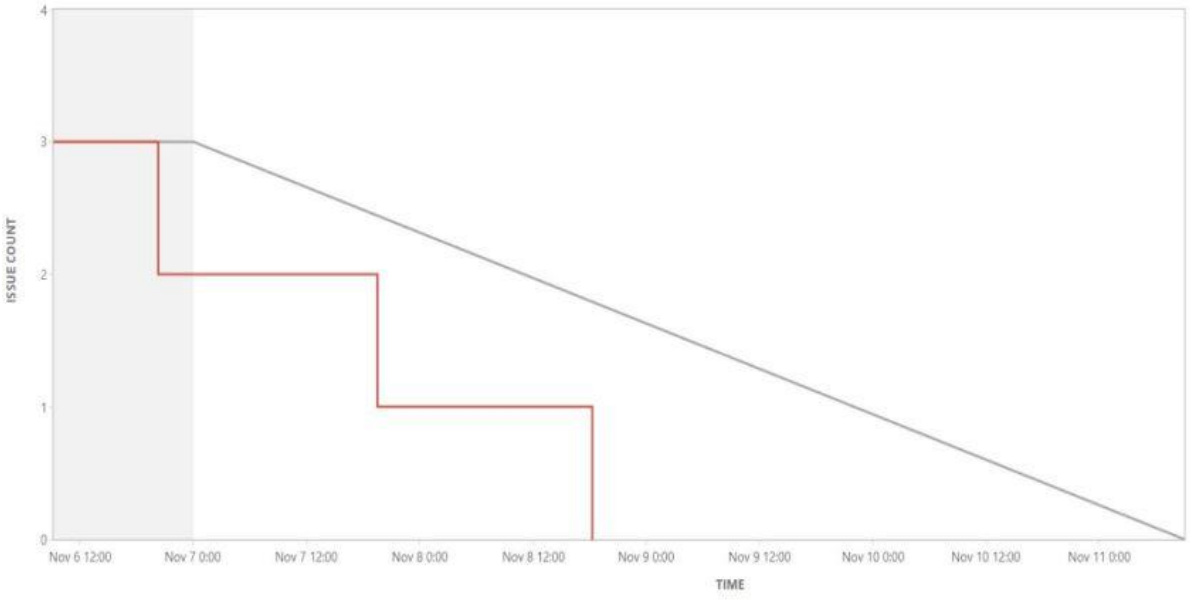
Imagine we have a 10-day sprint duration, and the velocity of the team is 20 (points per sprint). Let's calculate the team's average velocity (AV) per iteration unit (story points per day)

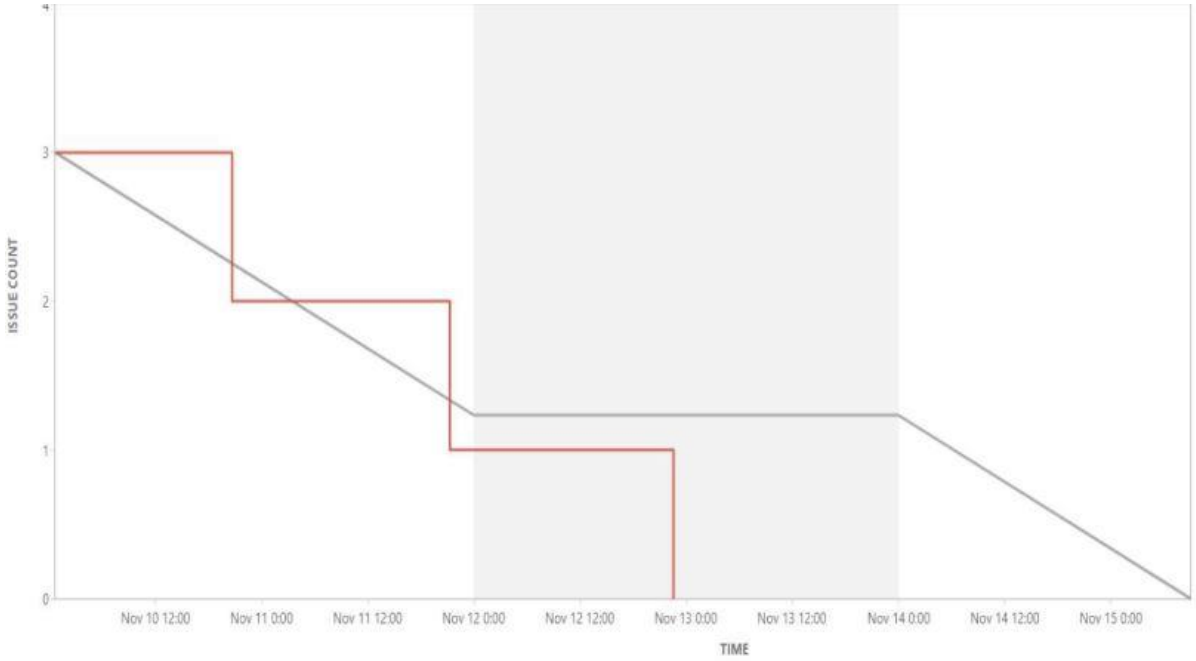
$$\mathbf{AV} = \text{sprint duration} / \text{velocity} = 20/10 = 2$$

6.3 Reports from JIRA

1.


ISSUE COUNT
4
3
2
1
0
Oct 31 12:00
Nov 1 0:00
Nov 1 12:00
Nov 2 0:00
Nov 2 12:00
Nov 3 0:00
Nov 3 12:00
Nov 4 0:00
Nov 4 12:00
Nov 5 9:00
TIME


ISSUE COUNT
4
3
2
1
0
Nov 6 12:00
Nov 7 0:00
Nov 7 12:00
Nov 8 0:00
Nov 8 12:00
Nov 9 0:00
Nov 9 12:00
Nov 10 0:00
Nov 10 12:00
Nov 11 0:00
TIME



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

7.1 Feature 1

Admin assigning an agent to a ticket Code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
      integrity="sha384-
wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN"
      crossorigin="anonymous"
    />
```

```
  <link rel="stylesheet" href="../static/styles.css" />
  <link href="static/icon.png" rel="icon">
<link href="static/icon.png" rel="apple-touch-icon">

  <title>Customer Care Registry</title>
</head>
<body id="body">
  <div class="container">
    <nav class="navbar">
      <div class="nav_icon" onclick="toggleSidebar()">
        <i class="fa fa-bars" aria-hidden="true"></i>
      </div>
      <div class="navbar__left">
        <a href="/admindashboard">Admin Dashboard</a>
        <a href="/viewCustomers">View Customers</a>
        <a href="/viewAgents">View agents</a>
        <a href="/deleteCustomers">Delete Customers</a>
        <a href="/deleteAgents">Delete agents</a>
        <a class="active_link" href="/assignTickets">Assign agents to tickets</a>
        <a href="/viewTickets">View tickets raised</a>
      </div>
      <div class="navbar__right">
        <a href="#">
          <i class="fa fa-clock-o" aria-hidden="true"></i>
        </a>
        <a href="/Profile">
          <img width="30" src="../static/avatar.svg" alt="" />
        </a>
      </div>
```

```
</nav>

<main>
  <div class="main__container">
    <div class="main__title">
      <img src="../static/hello.svg" alt="" />
      <div class="main__greeting">
        <h1>Hello {{username}}</h1>
        <p>View All Tickets Raised</p>
      </div>
    </div>
    <div class="chartsnew">


      <div class="charts__right">
        <div class="charts__right__title">
          <div>
            <h1>View All Tickets</h1>
            <p>All tickets rasied in the application</p>
          </div>
          <i class="fa fa-usd" aria-hidden="true"></i>
        </div>

        <div class="charts__right__cards">
          <div class="AllExpenses">
          <table border="1">
            <thead>
            <tr>
              <th>TICKET ID</th>
              <th>CUSTOMER USERNAME</th>
```

```
            <th>CUSTOMER EMAIL ADDRESS</th>
            <th>DESCRIPTION</th>
            <th>BILL NUMBER</th>
            <th>AGENT USERNAME</th>
            <th>AGENT EMAIL ADDRESS</th>
            <th>STATUS</th>
            <th>ASSIGN AGENT</th>
          </tr>
        </thead>
        <tbody>
            {% for row in tickets %}
            <tr>
              <td>{{row["TICKETID"]}}</td>
              <td>{{row["USERNAME"]}}</td>
              <td>{{row["EMAIL"]}}</td>
              <td> {{row["DESCRIPTION"]}}</td>
              <td>{{row["BILLNO"]}}</td>
              <td>{{row["AGENTUSERNAME"]}}</td>
              <td>{{row["AGENTEMAIL"]}}</td>
              <td>{{row["STATUS"]}}</td>
              <td><a href="/assignTickets/{{row['TICKETID']}}">Assign</a></td>
            </tr>
         {% endfor %}
        </tbody>
        </table>
      </div>

      </div>
    </div>
  </div>
```

```
  </div>
</main>

<div id="sidebar">
  <div class="sidebar__title">
    <div class="sidebar__img">
      <h1>Customer Care Registry</h1>
    </div>
    <i
      onclick="closeSidebar()"
      class="fa fa-times"
      id="sidebarIcon"
      aria-hidden="true"
    ></i>
  </div>

  <div class="sidebar__menu">
    <div class="sidebar__link">
      <i class="fa fa-home"></i>
      <a class="active_link" href="/admindashboard">Admin Dashboard</a>
    </div>
    <h2>Profile</h2>
    <div class="sidebar__link">
      <i class="fa fa-user" aria-hidden="true"></i>
      <a href="/adminprofile">Profile Information</a>
    </div>
    <h2>Manage Agents/Customers</h2>
    <div class="sidebar__link">
      <i class="fa fa-plus"></i>
```

```
      <a href="/viewCustomers">View Customers</a>
    </div>
    <div class="sidebar__link">
      <i class="fa fa-plus"></i>
      <a href="/viewAgents">View agents</a>
    </div>
    <div class="sidebar__link">
      <i class="fa fa-plus"></i>
      <a href="/deleteCustomers">Delete Customers</a>
    </div>
    <div class="sidebar__link">
      <i class="fa fa-plus"></i>
      <a href="/deleteAgents">Delete Agents</a>
    </div>
    <h2>Assign Tickets</h2>
    <div class="sidebar__link active_menu_link">
      <i class="fa fa-plus"></i>
      <a href="/assignTickets">Assign tickets to agents</a>
    </div>
    <div class="sidebar__link">
      <i class="fa fa-plus"></i>
      <a href="/viewTickets">View Tickets raised</a>
    </div>
    <div class="sidebar__logout">
      <i class="fa fa-power-off"></i>
      <a href="/adminlogout">Log out</a>
    </div>
  </div>
 </div>
</div>
```

```
    <script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>
    <script src="../static/script.js"></script>
  </body>
</html>
```

Explanation:

• User creates a ticket by describing the query

• Admin views the newly created ticket in the dashboard

• In the dropdown given, admin selects an agent

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

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

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

• Then, the dashboard of the admin gets refreshed

7.2 Feature 2

Customer closing a ticket Code:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link
      rel="stylesheet"
      href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css"
      integrity="sha384-
wvfXpqpZZVQGK6TAh5PVlGOfQNHSoD2xbE+QkPxCAFlNEevoEH3Sl0sibVcOQVnN"
      crossorigin="anonymous"
    />


    <link rel="stylesheet" href="../static/styles.css" />
    <link href="static/icon.png" rel="icon">
  <link href="static/icon.png" rel="apple-touch-icon">

    <title>Customer Care Registry</title>
  </head>
  <body id="body">
```

```
<style>
   td{
     width : 150px;
     text-align: center;
     border : 1px solid black;
     padding : 5px;
     color: black;
     font-size: 20px;
   }
 </style>
<div class="container">
 <nav class="navbar">
   <div class="nav_icon" onclick="toggleSidebar()">
     <i class="fa fa-bars" aria-hidden="true"></i>
   </div>
   <div class="navbar__left">
     <a href="/admindashboard">Admin Dashboard</a>
     <a class="active_link" href="/viewCustomers">View Customers</a>
     <a href="/viewAgents">View agents</a>
     <a href="/deleteCustomers">Delete Customers</a>
     <a href="/deleteAgents">Delete agents</a>
     <a href="/assignTickets">Assign agents to tickets</a>
     <a href="/viewTickets">View tickets raised</a>
   </div>
   <div class="navbar__right">
     <a href="#">
       <i class="fa fa-clock-o" aria-hidden="true"></i>
     </a>
     <a href="/Profile">
       <img width="30" src="../static/avatar.svg" alt="" />
     </a>
   </div>
 </nav>

 <main>
   <div class="main__container">
     <div class="main__title">
       <img src="../static/hello.svg" alt="" />
       <div class="main__greeting">
         <h1>Hello {{username}}</h1>
         <p>View All Customers</p>
       </div>
     </div>
     <div class="chartsnew">


       <div class="charts__right">
         <div class="charts__right__title">
           <div>
             <h1>Customers</h1>
```

```
          <p>All customers registered in the application</p>
        </div>
        <i class="fa fa-usd" aria-hidden="true"></i>
      </div>

      <div class="charts__right__cards">
        <div class="AllExpenses">
        <table border="1">
          <thead>
          <tr>
            <th>USERNAME</th>
            <th>EMAIL ADDRESS</th>
            <th>TICKETS RAISED</th>
            <th>TICKETS RESOLVED</th>
            <th>NOTIFICATIONS SENT</th>
          </tr>
        </thead>
        <tbody>
            {% for row in customers %}
            <tr>
              <td>{{row["USERNAME"]}}</td>
              <td> {{row["EMAILADDRESS"]}}</td>
              <td>{{row["TICKETS"]}}</td>
              <td>{{row["TICKETSRESOLVED"]}}</td>
              <td>{{row["NOTIFICATIONS"]}}</td>
            </tr>
         {% endfor %}
        </tbody>
        </table>
      </div>

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

 </div>
</main>

<div id="sidebar">
 <div class="sidebar__title">
   <div class="sidebar__img">
     <h1>Customer Care Registry</h1>
   </div>
   <i
     onclick="closeSidebar()"
     class="fa fa-times"
     id="sidebarIcon"
     aria-hidden="true"
   ></i>
 </div>
```

```
      <div class="sidebar__menu">
        <div class="sidebar__link">
          <i class="fa fa-home"></i>
          <a href="/admindashboard">Admin Dashboard</a>
        </div>
        <h2>Profile</h2>
        <div class="sidebar__link">
          <i class="fa fa-user" aria-hidden="true"></i>
          <a href="/adminprofile">Profile Information</a>
        </div>
        <h2>Manage Agents/Customers</h2>
        <div class="sidebar__link active_menu_link">
          <i class="fa fa-plus"></i>
          <a href="/viewCustomers">View Customers</a>
        </div>
        <div class="sidebar__link">
          <i class="fa fa-plus"></i>
          <a href="/viewAgents">View agents</a>
        </div>
        <div class="sidebar__link">
          <i class="fa fa-plus"></i>
          <a href="/deleteCustomers">Delete Customers</a>
        </div>
        <div class="sidebar__link">
          <i class="fa fa-plus"></i>
          <a href="/deleteAgents">Delete Agents</a>
        </div>
        <h2>Assign Tickets</h2>
        <div class="sidebar__link">
          <i class="fa fa-plus"></i>
          <a href="/assignTickets">Assign tickets to agents</a>
        </div>
        <div class="sidebar__link">
          <i class="fa fa-plus"></i>
          <a href="/viewTickets">View Tickets raised</a>
        </div>
        <div class="sidebar__logout">
          <i class="fa fa-power-off"></i>
          <a href="/adminlogout">Log out</a>
        </div>
      </div>
    </div>
  </div>
  <script src="https://cdn.jsdelivr.net/npm/apexcharts"></script>
  <script src="../static/script.js"></script>
 </body>
</html>
```

Explanation:

• User creates a ticket by describing the query

• Admin assigns an agent to this ticket

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

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

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

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

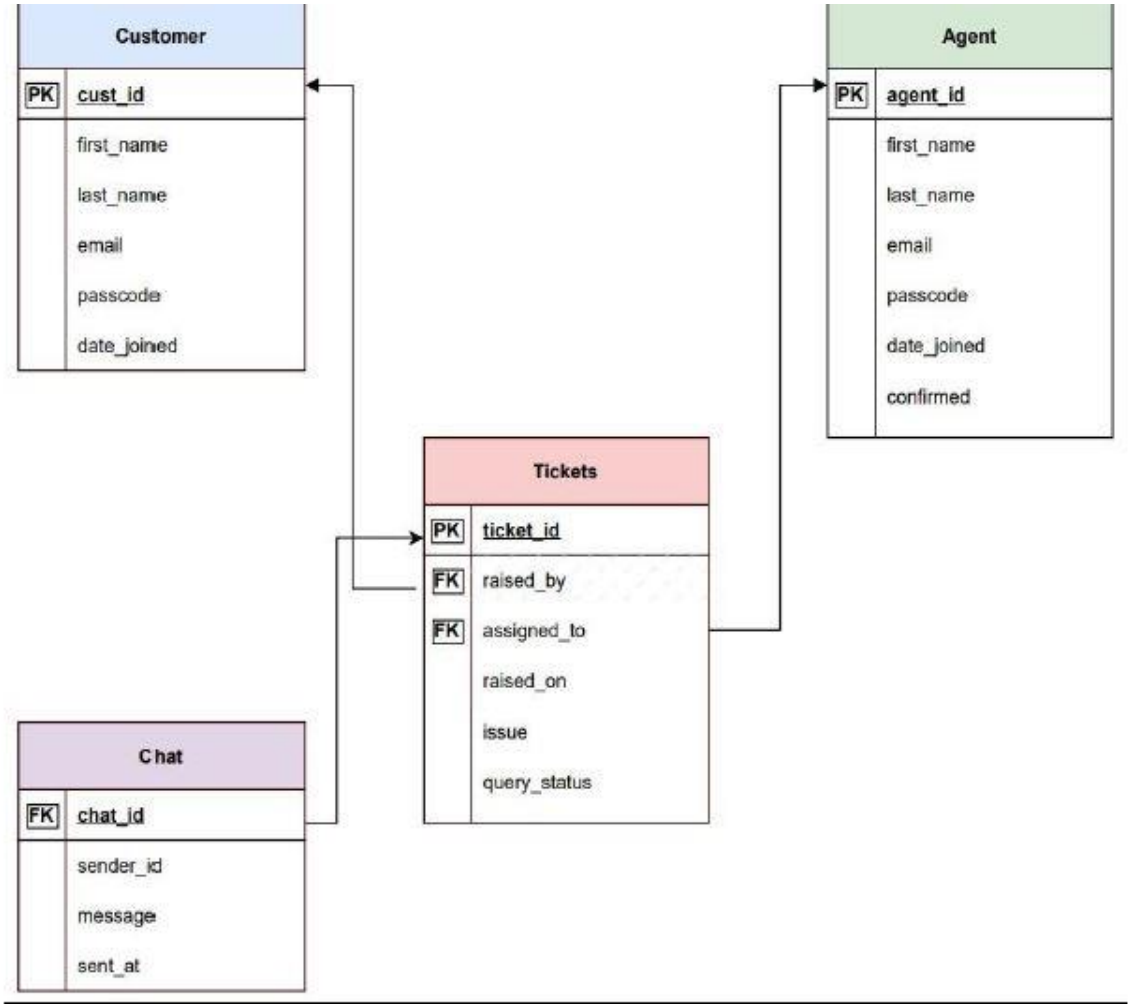
• Thus the ticket is closed

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

7.3 Database Schema (if Applicable)

A database schema is the skeleton structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data.

A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



8. TESTING

8.1 Test Cases

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.

Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because

if developer missed something, then it can be caught during execution of these full-proof test cases.

To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

8.2 User Acceptance Testing

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

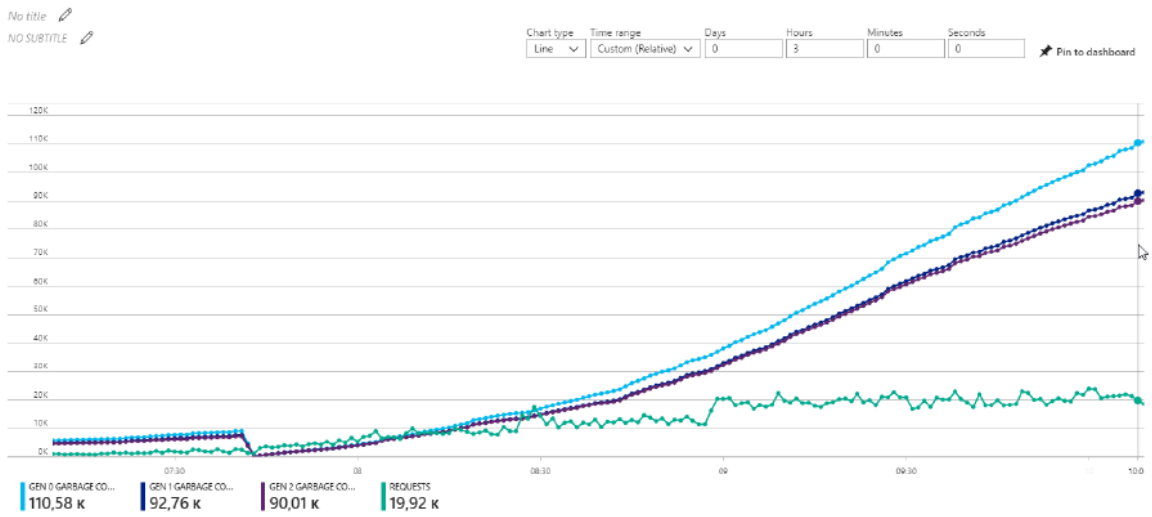
Test Case Analysis

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

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

9.RESULTS

9.1 Performance Metrics



10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

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

Disadvantages:

- Only web application is available right now (as of writing).
- UI is not so attractive, it's just simple looking.
- No automated replies.
- No SMS alerts.

- Supports only text messages while chatting with the Agent.
- No tap to reply feature.
- No login alerts.
- Cannot update the mobile number.
- Account cannot be deleted, once created.
- Customers cannot give feedback to the agent for clarifying the queries.

11. CONCLUSION

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

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

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

12.FUTURE SCOPE

Machine learning (ML), emerging customer service trends 2022 can help businesses in improving overall CX. Chat applications powered by AI are trending. Large companies, as well as startups, are leveraging this to reduce costs and improve service for customers.

Predictive analytics has particularly proved to be very useful. Through this, quarries that will result in a call for assistance can be predicted easily. Implementing ML in customer service trends will give you a significant difference in business growth.

13. APPENDIX

Flask:

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries.

It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions.

JavaScript:

JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS.

As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, often incorporating third-party libraries.

IBM Cloud:

IBM cloud computing is a set of cloud computing services for business offered by the information technology company IBM.

Kubernetes:

Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.

Docker:

Docker is a set of platforms as a service product that use OS-level virtualization to deliver software in packages called containers.

SOURCE CODE

base.html

```
<!DOCTYPE html>
<html>
<head>
```

```
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/css/bootstrap.min.css"
integrity="sha384-
xOolHFLEh07PJGoPkLv1IbcEPTNtaed2xpHsD9ESMhqIYd0nLMwNLD69Npy4HI+N"
crossorigin="anonymous">
<script src="https://cdn.jsdelivr.net/npm/bootstrap@4.6.2/dist/js/bootstrap.min.js"
integrity="sha384-
+sLIOodYLS7CIrQpBjl+C7nPvqq+FbNUBDunl/OZv93DB7Ln/533i8e/mZXLi/P+"
crossorigin="anonymous"></script>
<style>
 body{
background-color:"pink";

}
</style>
</head>
```

application.py

```
from flask import Flask, render_template
import sqlite3 as sql

app=Flask(__name__)

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

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

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

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

if __name__=='__main__':
   app.run(host='0.0.0.0' ,port=8000,debug=True)
```

```
<body>

<a href="/">HOME</a>
<a href="/about">ABOUT</a>
<a href="/signin">SIGN IN</a>
<a href="/signup">SIGN UP</a>
<hr><br>




</body>

</html>
```

login.html:

```
{% extends 'base.html' %}
{% block title %}
Login
{% endblock %}
{% block main %}
<div class="bg-main-div">
<section class="login-section">
<div class="login-div">
<div class="login-header">
<img src="{{ url_for('static', filename='images/cart logo white.png') }}" class="login-img"
alt="logo" /> <h2>Sign in</h2>
<p>Use your Registry Account</p>
```

```
</div>
<div class="login-remind">
<form action="{{ url_for('blue_print.login') }}" method="POST" class="login-form">
<label>Email</label>
<input type="email" required value="{{ email }}" name="email" placeholder="Enter your email"/>
<label>Password</label>
<input type="password" required value="{{ password }}" name="password" id="password-input" placeholder="Enter your password"/>
<div class="show-pass-div">
<input type="checkbox" onclick="showPassword()" style="height: 20px;"/>
<p>Show Password</p>
</div>
<div class="role-div">
<p>Role : </p>
<div>
<div>
<input type="radio" style="height: 20px;" value="Customer" checked name="role-check"/>
<p>Customer</p>
</div>
<div>
<input type="radio" style="height: 20px;" value="Agent" name="role-check"/>
<p>Agent</p>
</div>
```

```
</div>
</div>
<button class="submit-btn" type="submit">Login</button>
<div>
<!-- {{ url_for('blue_print.forgot') }} -->
<a href="{{ url_for('blue_print.forgot') }}" class="links">Forgot Password?</a> <br>
<div>
<a href="{{ url_for('blue_print.register') }}" class="links">Don't have an account yet?
Register</a>
</div>
</div>
</form>
</div>
</div>
</section>
</div>
```

address.html:

```
{% extends 'base.html' %}
{% block title %}
Address Column
{% endblock %}
{% block main %}
<div class="dashboard-div">
<nav>
<div class="dash-nav">
<div>
<div class="dash-img-text">
{% if user == "AGENT" %}
<a href="{{ url_for('agent.assigned') }}">
<i class="fa fa-arrow-left" aria-hidden="true"></i>
</a>
<img src="{{ url_for('static', filename='images/cust profile.png') }}" class="img-in-nav"
alt="logo"/> {% else %}
<a href="{{ url_for('customer.tickets') }}">
<i class="fa fa-arrow-left" aria-hidden="true"></i>
</a>
```

```
<img src="{{ url_for('static', filename='images/agent.png') }}" class="img-in-nav"
alt="logo"/>
{% endif %}
<h3>{{ name }}</h3>
</div>
</div>
<div>
<div style="align-items: center;">
{% if value == "True" %}
{% if user == "CUSTOMER" %}
<a href="/customer/close/{{ id }}"><button class="logout-btn">CLOSE
TICKET</button></a>
{% endif %}
{% endif %}
</div>
</div>
</div>
</nav>
<div class="chat-body">
<div class="chat-contents" id="content">
{% if msgs_to_show %}
{% for chat in chats %}
{% if chat['SENDER_ID'] == sender_id %}
<div class="message-sent">{{ chat['MESSAGE'] }}</div>
{% else %}
<div class="message-sent received">{{ chat['MESSAGE'] }}</div>
{% endif %}
{% endfor %}
{% endif %}
</div>
<div class="chat-input-div">
{% if value == "True" %}
<form method="POST" action="{{ post_url }}">
<input name="message-box" class="chat-input" type="text" placeholder="Type something"
required/>
<button type="submit" class="chat-send">
<i class="fa fa-paper-plane-o" aria-hidden="true"></i>
</button>
</form>
{% else %}
<div>
{% if user == "CUSTOMER" %}
<h4>You closed this ticket. Chats are disabled</h4>
```

{% else %}
<h4>{{ name }} closed this ticket. Chats are disabled</h4>
{% endif %}

{% endif %}

{% endblock %}

chat.py:

```
from flask import render_template, Blueprint, request, session, redirect, url_for import
ibm_db from datetime import datetime import time
chat = Blueprint("chat_bp", __name__)
@chat.route('/chat/<ticket_id>/<receiver_name>/', methods = ['GET', 'POST']) def
address(ticket_id, receiver_name):
'''
Address Column - Agent and Customer chats with one another
: param ticket_id ID of the ticket for which the chat is being opened
: param receiver_name Name of the one who receives the texts, may be Agent / Customer
'''
# common page for both the customer and the agent
# so cannot use login_required annotation
# so to know who signed in, we have to use the session user = "" sender_id = "" value = ""
can_trust = False
post_url = f'/chat/{ticket_id}/{receiver_name}/'
if session['LOGGED_IN_AS'] is not None: if session['LOGGED_IN_AS'] ==
"CUSTOMER":
# checking if the customer is really logged in
# by checking, if the customer has uuid attribute
from .views import customer
if(hasattr(customer, 'uuid')): user = "CUSTOMER" sender_id = customer.uuid can_trust =
True
```

else:
# logging out the so called customer return redirect(url_for('blue_print.logout'))
elif session['LOGGED_IN_AS'] == "AGENT":
# checking if the agent is really logged in # by checking, if the agent has uuid aatribute from .views import agent
if (hasattr(agent, 'uuid')): user = "AGENT" sender_id = agent.uuid can_trust = True
else:
# Admin is the one who logged in
# admin should not see the chats, sp directly logging the admin out return redirect(url_for('blue_print.logout'))
to_show = False message = ""
if can_trust:
# importing the connection string from .views import conn
if request.method == 'POST':
# chats are enabled, only if the ticket is OPEN # getting the data collected from the customer / agent myMessage = request.form.get('message-box')
if len(myMessage) == 0:
to_show = True message = "Type something!" else:
# inserting the message in the database
# query to insert the message in the database message_insert_query = '''
INSERT INTO chat
(chat_id, sender_id, message, sent_at)

VALUES
(?, ?, ?, ?)
''' try:
stmt = ibm_db.prepare(conn, message_insert_query) ibm_db.bind_param(stmt, 1, ticket_id)
ibm_db.bind_param(stmt, 2, sender_id) ibm_db.bind_param(stmt, 3, myMessage)
ibm_db.bind_param(stmt, 4, datetime.now())
ibm_db.execute(stmt)
except:
to_show = True message = "Please send again!"
return redirect(post_url)
else:
# method is GET
# retrieving all the messages, if exist from the database msgs_to_show = False
# query to get all the messages for this ticket get_messages_query = ''' SELECT * FROM
chat WHERE chat_id = ?
ORDER BY sent_at ASC
'''
query to check if the ticket is still OPEN query_status_check = '''
SELECT query_status FROM tickets WHERE ticket_id = ?
'''
try:
# first checking if the ticket is OPEN
check = ibm_db.prepare(conn, query_status_check) ibm_db.bind_param(check, 1, ticket_id)
ibm_db.execute(check)
value = "True" if ibm_db.fetch_assoc(check)['QUERY_STATUS'] == "OPEN" else "False"

```
# getting all the messages concerned with this ticket stmt = ibm_db.prepare(conn,
get_messages_query) ibm_db.bind_param(stmt, 1, ticket_id) ibm_db.execute(stmt)
messages = ibm_db.fetch_assoc(stmt) messages_list = []
while messages != False:
messages_list.append(messages) print(messages)
messages = ibm_db.fetch_assoc(stmt)
# then some messages exist in this chat if len(messages_list) > 0: msgs_to_show = True
elif len(messages_list) == 0 and value == "True":
# ticket is OPEN
# but no messages are sent b/w the customer and the agent msgs_to_show = False to_show =
True
message = f'Start the conversation with the {"Customer" if user == "AGENT" else "Agent"}'
except:
to_show = True
message = "Something happened! Try Again"
return render_template(
'address.html', to_show = to_show, message = message,
id = ticket_id, chats = messages_list, msgs_to_show = msgs_to_show, sender_id = sender_id,
name = receiver_name, user = user, post_url = post_url, value = value
)
else:
```

logging out whoever came inside the link return redirect(url_for('blue_print.logout'), user = user)

__init__.py:

from flask import Flask, session from flask_login import LoginManager
def create_app():
app = Flask(__name__) app.config['SECRET_KEY'] = "PHqtYfAN2v@CCR2022"
# registering the blue prints with the app from .routes.views import views
app.register_blueprint(views, appendix='/')
from .routes.cust import cust app.register_blueprint(cust, appendix='/customer/')
from .routes.admin import admin app.register_blueprint(admin, appendix='/admin/')
from .routes.agent import agent app.register_blueprint(agent, appendix='/agent/')
from .routes.chat import chat app.register_blueprint(chat, appendix='/chat/') # setting up the login manager login_manager = LoginManager() login_manager.login_view = "blue_print.login" login_manager.init_app(app)
@login_manager.user_loader def load_user(id): if session.get('LOGGED_IN_AS') is not None: if session['LOGGED_IN_AS'] == "CUSTOMER":
from .routes.views import customer
if hasattr(customer, 'first_name'):
return customer
elif session['LOGGED_IN_AS'] == "AGENT":
from .routes.views import agent

```
if hasattr(agent, 'first_name'):
return agent
elif session['LOGGED_IN_AS'] == "ADMIN":
from .routes.views import admin
if hasattr(admin, 'email'):
return admin
else:
return None
return app
```

GITHUB AND PROJECT DEMO LINK

Github Rep Link: