

11/19/2022

PROJECT REPORT

Customer Care Registry

PNT2022TMID50230

JAYARAJ ANNAPACKIAM CSI COLLEGE OF ENGINEERING

**HX8001 - PROFESSIONAL READINESS FOR INNOVATION, EMPLOYABILITY ANDENTREPRENEURSHIP**

**CUSTOMER CARE REGISTRY**

**URL’SUSING IBM WATSON STUDIO**

**TEAM ID:** PNT2022TMID50230

**FACULTY MENTOR NAME:** MRS. JENIFUS VINSTON

**TEAM LEADER:** R.ISWARYA

**TEAM MEMBERS:** S.SUDALAIVADIVU@GAYATHRI

V.GEETHAI

N.SELVA MARI

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No.** | **Title** | **Page No.** |
| 1 | **INTRODUCTION** |  |
| [1.1 Project Overview](#_bookmark0) | 3 |
| [1.2 Project Purpose](#_bookmark1) | 3 |
| 2 | **LITERATURE SURVEY** |  |
| [2.1 Existing Problem](#_bookmark2) | 4 |
| [2.2 References](#_bookmark3) | 4 |
| [2.3 Problem Statement Definition](#_bookmark4) | 4 |
| 3 | **IDEATION AND PROPOSED SOLUTION** |  |
| [3.1 Empathy Map Canvas](#_bookmark5) | 5 |
| [3.2 Ideation and Brainstorming](#_bookmark6) | 6 |
| [3.3 Proposed Solution](#_bookmark7) | 8 |
| [3.4 Problem Solution Fit](#_bookmark8) | 9 |
| 4 | **REQUIREMENT ANALYSIS** |  |
| 4.1 Functional Requirements | 10 |
| 4.2 Non-functional Requirements | 11 |
| 5 | **PROJECT DESIGN** |  |
| 5.1 Data Flow Diagram | 12 |
| 5.2 Solution and Technical Architecture | 13 |
| 5.3 User Stories | 14 |
| 6 | **PROJECT PLANNING AND SCHEDULING** |  |
| 6.1 Sprint Planning and Estimation | 16 |
| 6.2 Sprint Delivery Schedule | 17 |
| 6.3 Reports from JIRA | 18 |
| 7 | **CODING AND SOLUTIONING** |  |
| 7.1 Admin assigning an agent to a ticket | 20 |
| 7.2 Customer closing a ticket | 21 |
| 7.3 Database Schema | 22 |
| 8 | **TESTING** |  |
| 8.1 Test Cases | 23 |
| 8.2 User Acceptance Testing | 24 |

|  |  |  |
| --- | --- | --- |
| 9 | **RESULTS** |  |
| 9.1 Performance Metrics | 25 |
| 10 | **ADVANTAGES AND DISADVANTAGES** | 26 |
| 11 | **CONCLUSION** | 27 |
| 12 | **FUTURE SCOPE** | 27 |
| 13 | **APPENDIX** | 28 |
|  | Source Code | 29 |
|  | GitHub Link and Project Demo Link | 38 |

1. **INTRODUCTION**

## PROJECT OVERVIEW

**Short Description:**

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

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

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

## PURPOSE

The purpose of the whole project is to:

* Provide a common platform to the customers to clarify their queries
* Having expert agents in the platform for better answering
* Customer’s tickets (queries) are answered quickly by the agents
* Customers and Agents can chat with one another for better understanding
* While doing so, the former asks questions
* Later, answers those questions as quickly and as legitimately as possible
* Customers can raise as many tickets as they want
* Customers and Agents can also submit their feedbacks to the Admin, for the betterment of the platform

## LITERATURE SURVEY

* 1. **Existing Problem**
     + Reviews and rating in the e-commerce websites are not reliable
     + Even more so, they are often been given by the manufactures themselves
     + Reviews are not from the authentic individuals
     + After buying the products, I am left with no option to clear my doubts
     + There is no common platform available to us, the customers, to have our doubts cleared
     + If it is existing, we are not getting fast replies. By the time, the reply comes, the issue might have been cleared or of not worth of being cleared to the customers

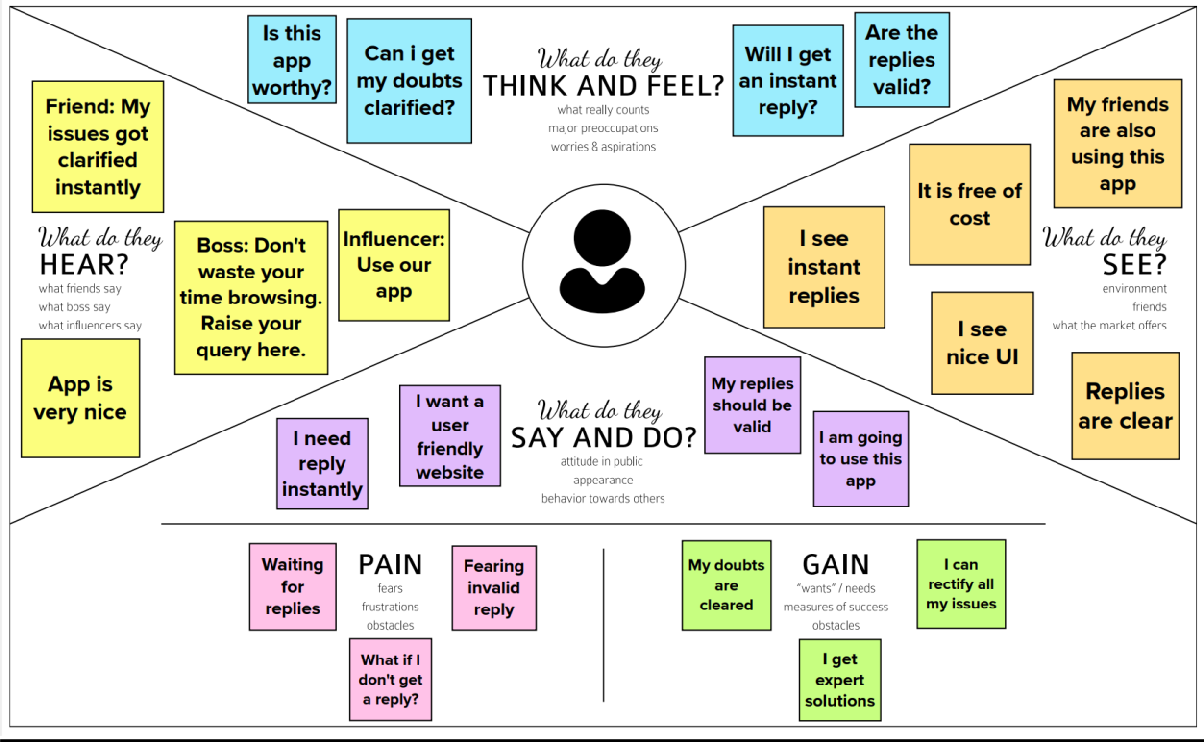
## Problem Statement Definition

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.

## IDEATION AND PROPOSED SOLUTION

* 1. **Empathy Map Canvas**
     + 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 help teams to better understand their users
     + Creating an effective solution requires understanding the true problem and the person who us 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



## 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

Team Gathering:

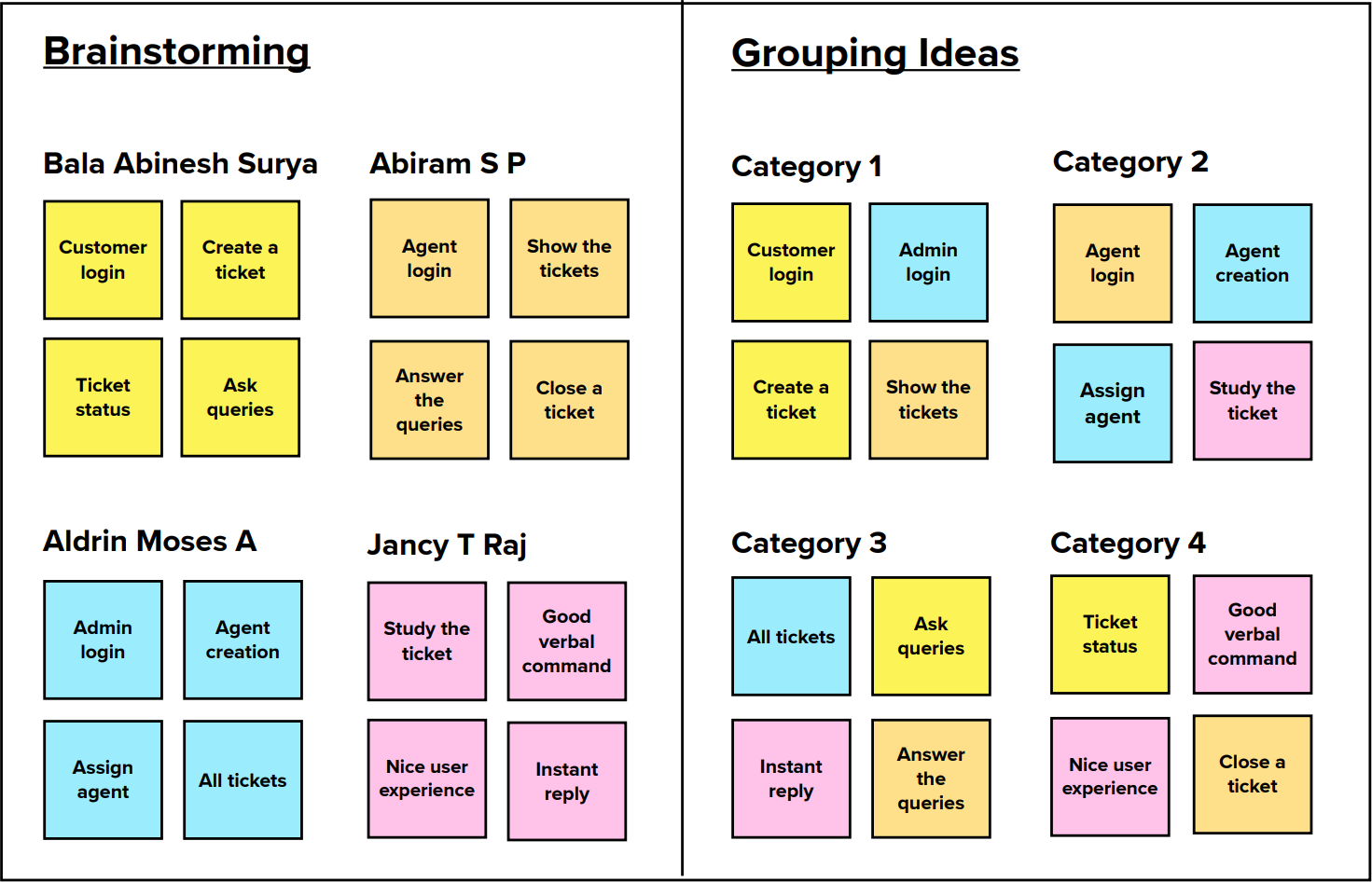
|  |  |
| --- | --- |
| **Team Members** | |
| Team Leader | Iswarya R |
| Team Members | Geethai V |
| Selva mari N |
| Sudalaivadivu@gayathri S |

Problem Statement:

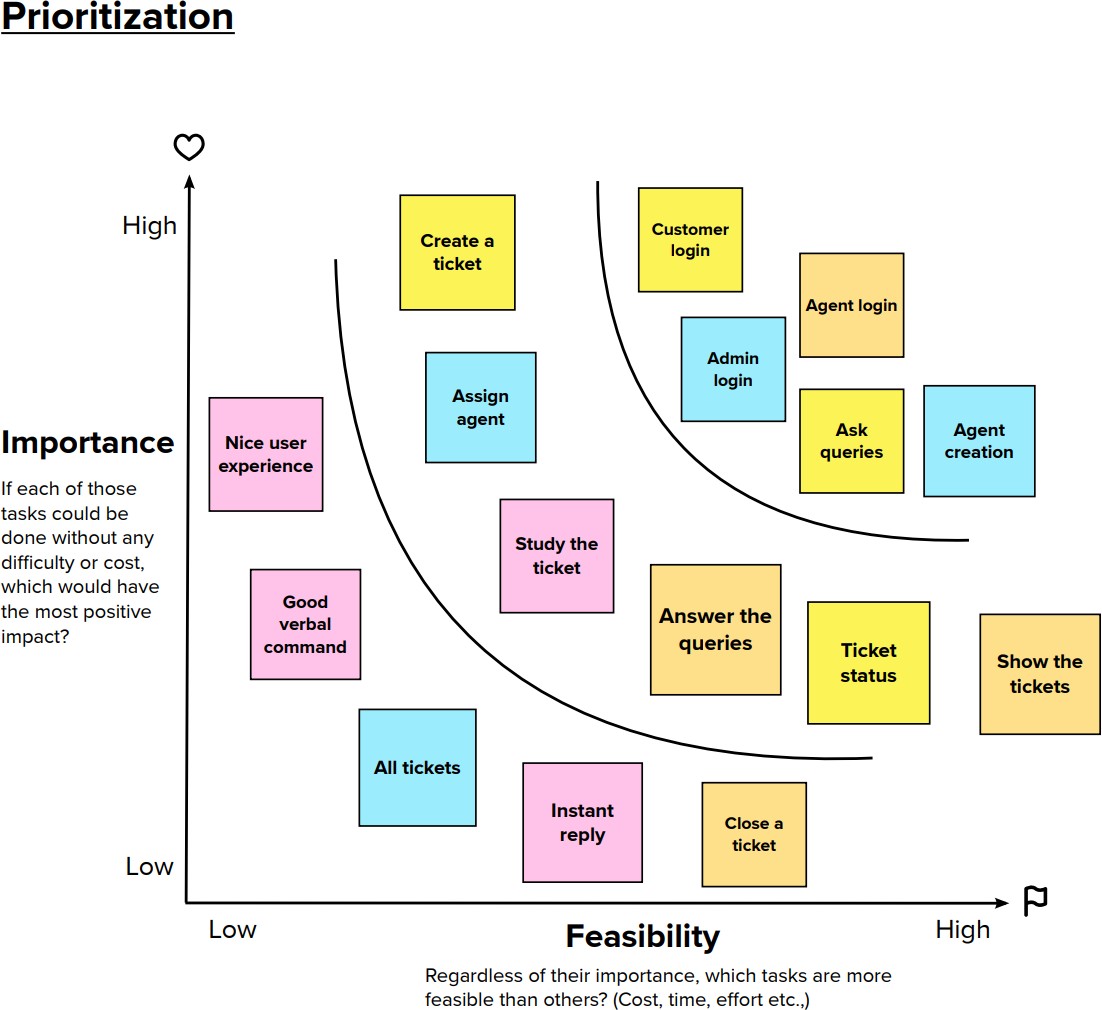
I am Iswarya and I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that in most times, I don’t have any reliable sources to clear my doubts in some of the products I buy.

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.

**Step-2:** Brainstorm, Idea Listing and Grouping



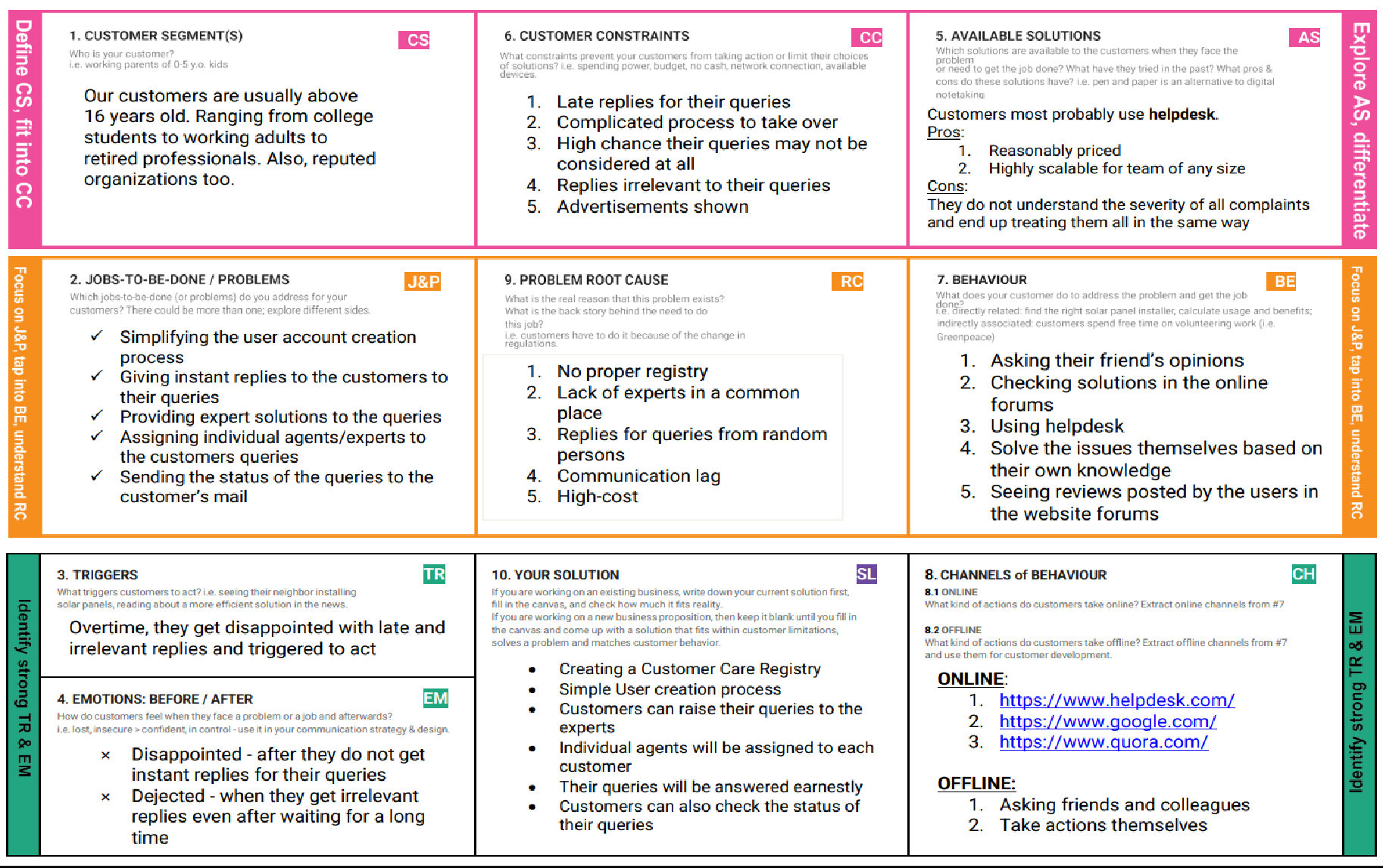
**Step-3:** Idea Prioritization



## 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 issue. |
|  | 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 | 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. |

* 1. **Problem Solution Fit**



## REQUIREMENT ANALYSIS

* 1. **Functional Requirements**
     + A functional requirement defines a function of a system or its component, where a function is described as a specification of behaviour between inputs and outputs.
     + It specifies “what should the software system do?”
     + Defined at a component level
     + Usually easy to define
     + Helps you verify the functionality of the software

|  |  |  |
| --- | --- | --- |
| **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 |
| FR-11 | Address Column | Agent clarifies the doubts of the customer |

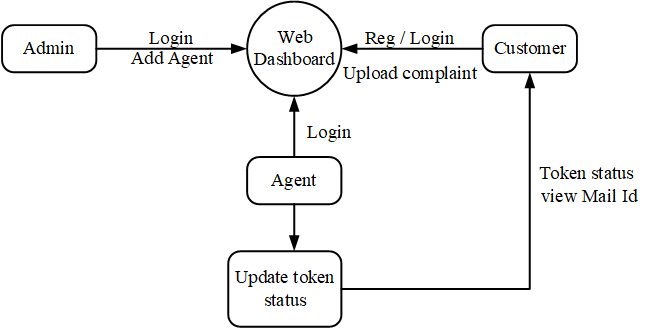
## Non-functional Requirements

* + - A non-functional requirement defines the quality attribute of a software system
    - It places constraint on “How should the software system fulfil the functional requirements?”
    - It is not mandatory
    - Applied to system as a whole
    - Usually more difficult to define
    - Helps you verify the performance of the software

|  |  |  |
| --- | --- | --- |
| **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. |

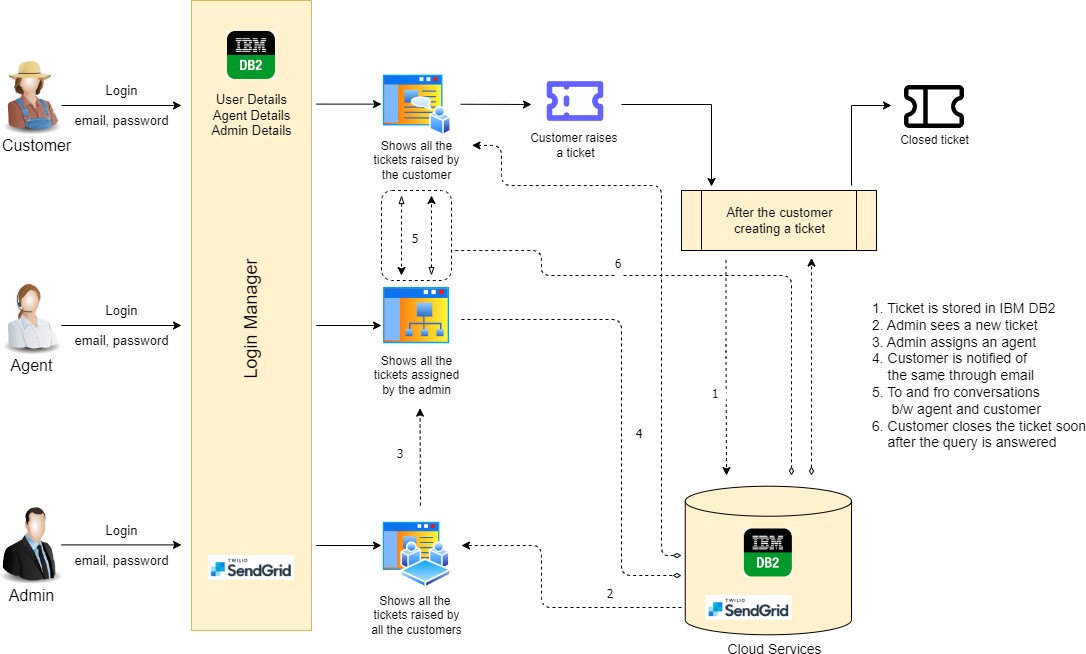
## PROJECT DESIGN

* 1. **Dataflow Diagram**

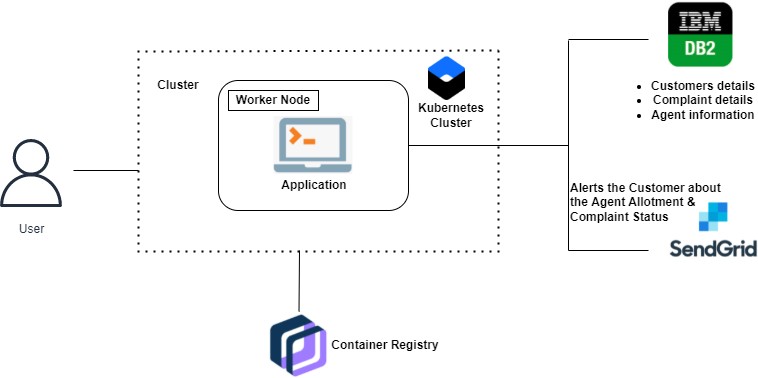


## Solution and Technical Architecture

Solution Architecture



# Technical Architecture



## 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 in case 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 option in case I forgot my old password | I get access to my account again | Medium | Sprint-4 |
| 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 |

1. **PROJECT DESIGN AND PLANNING**

## Sprint Planning and Estimation

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **User Type** | **Functional**  **Requirement (Epic)** | **User**  **Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| Sprint-1 | Customer (Web User) | Registration | USN-1 | As a customer, I can register for the application by entering my email, password, and confirming my password. | 2 | High | Iswarya,  geethai |
| Sprint-1 |  | Login | USN-2 | As a customer, I can login to the application by entering correct email and password | 1 | High | Geethai |
| Sprint-1 |  | Dashboard | USN-3 | As a customer, I can see all the tickets raised by me and lot more | 3 | High | Selva Mari Iswarya |
| Sprint-2 |  | Ticket creation | USN-4 | As a customer, I can create a new ticket with the detailed description of my query | 2 | High | Sudalaivadivu@gayathri |
| Sprint-3 |  | Address Column | USN-5 | As a customer, I can have conversations with the assigned agent and get my queries clarified | 3 | High | Iswarya  Geethai |
| Sprint-4 |  | Forgot password | USN-6 | As a customer, I can reset my password by this option in case I forgot my old password | 2 | Medium | Sudalaivadivu@garathri,Selava Mari |
| Sprint-4 |  | Ticket details | USN-7 | As a customer, I can see the current status of my tickets | 2 | Medium | Geethai  Sudalaivadivu |
| 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 | Selva mari |
| Sprint-3 |  | Dashboard | USN-2 | As an agent, I can see all the tickets assigned to me by the admin | 3 | High | Iswarya |
| Sprint-3 |  | Address Column | USN-3 | As an agent, I get to have conversations with the customer and clear his/her queries | 3 | High | Geethai  Sudalaivadivu |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Sprint** | **User Type** | **Functional Requirement (Epic)** | **User Story Number** | **User Story / Task** | **Story Points** | **Priority** | **Team Members** |
| 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 | Iswarya,  Geethi |
| 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 | Selva Mari |
| Sprint-1 |  | Dashboard | USN-2 | As an admin, I can see all the tickets raised in the entire system and lot more | 3 | High | SudalaiVadivu |
| Sprint-2 |  | Agent creation | USN-3 | As an admin, I can create an agent for clarifying the customer’s queries | 2 | High | Geethai |
| Sprint-2 |  | Assigning agent | USN-4 | As an admin, I can assign an agent for each ticket created by the customer | 3 | High | Selva Mari |
| 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 | Iswarya |

* 1. **Sprint Delivery Plan**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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 | 4 Days | 06 Nov 2022 | 11 Nov 2022 | 11 | 09 Nov 2022 |
| Sprint-4 | 8 | 4 Days | 10 Nov 2022 | 15 Nov 2022 | 8 | 13 Nov 2022 |

* 1. **Reports from JIRA**

# Sprint 1 – Burndown Chart



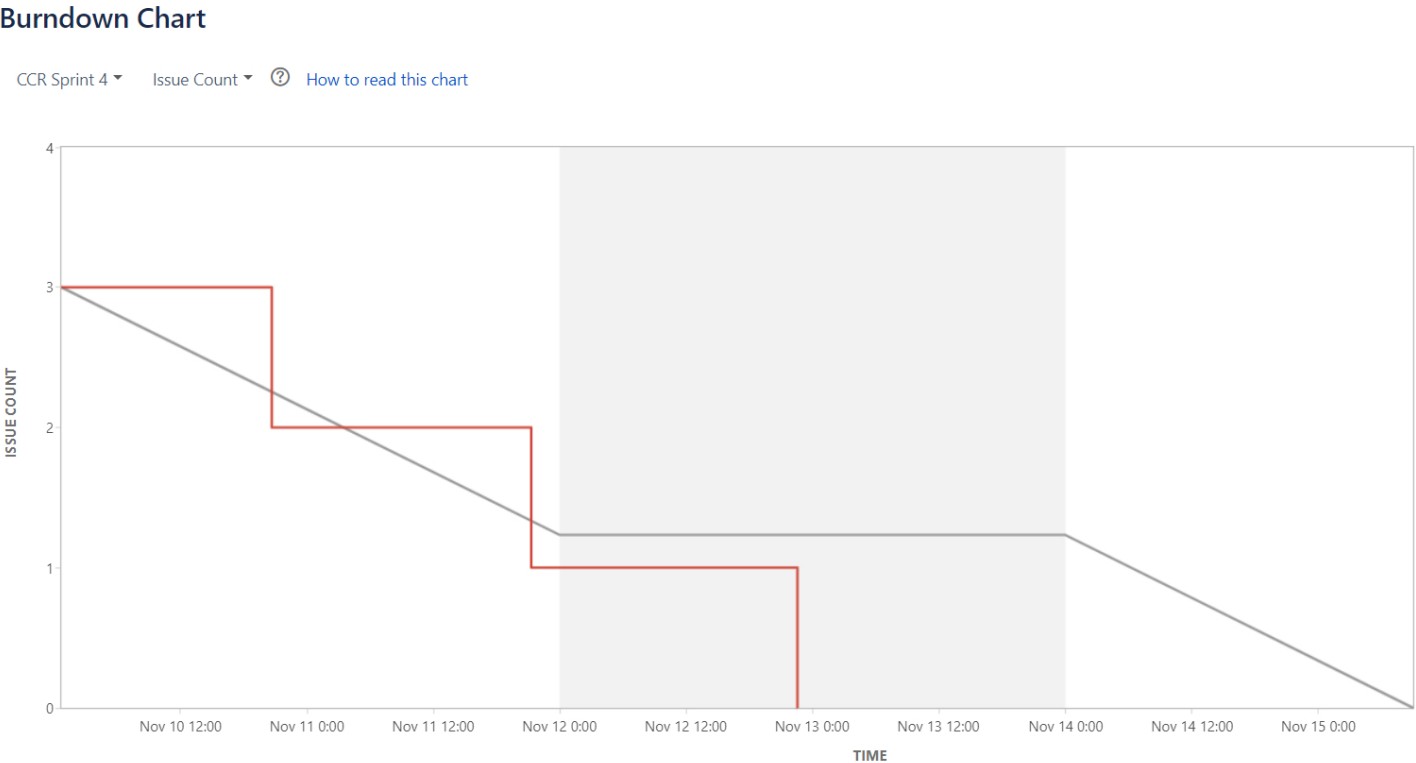
Sprint 2 – Burndown Chart



Sprint 3 – Burndown Chart



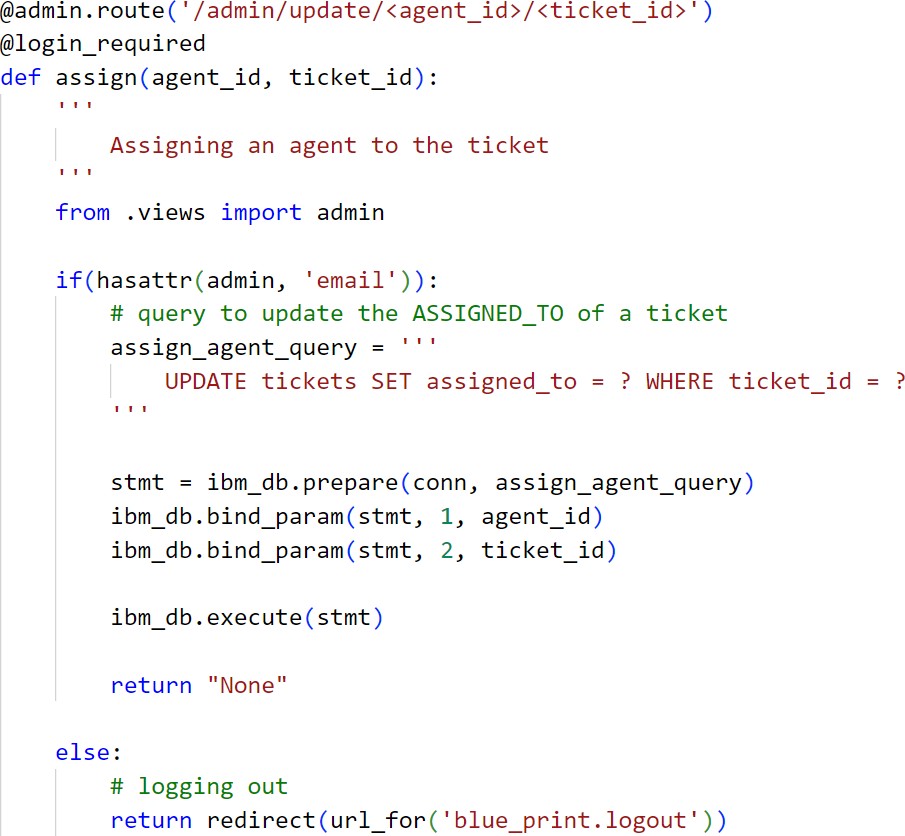
Sprint 4 – Burndown Chart



## CODING AND SOLUTIONING

* 1. **Admin assigning an agent to a ticket**

Code:

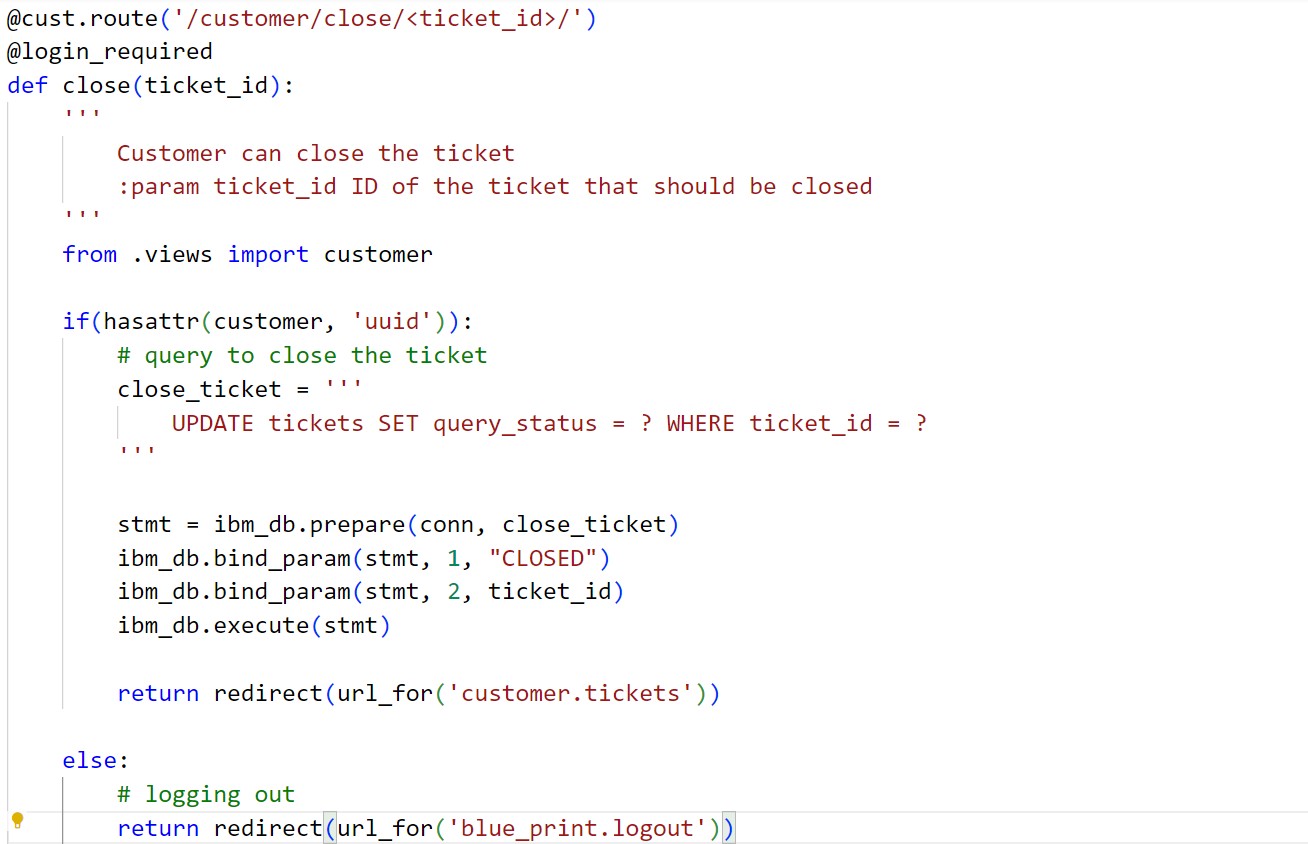


Explanation:

* + - 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

## Customer closing a ticket

Code:



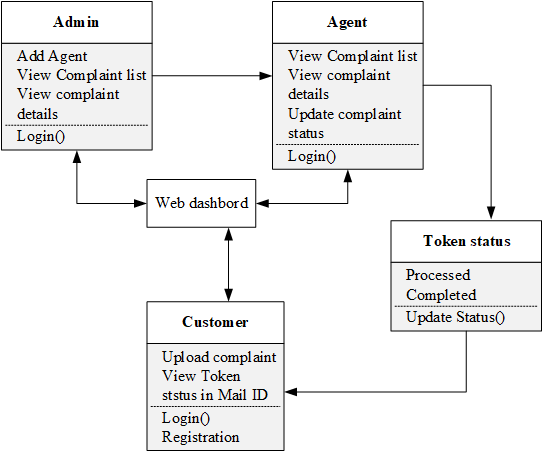
Explanation:

* + - 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

## Database Schema

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.



## TESTING

* 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.

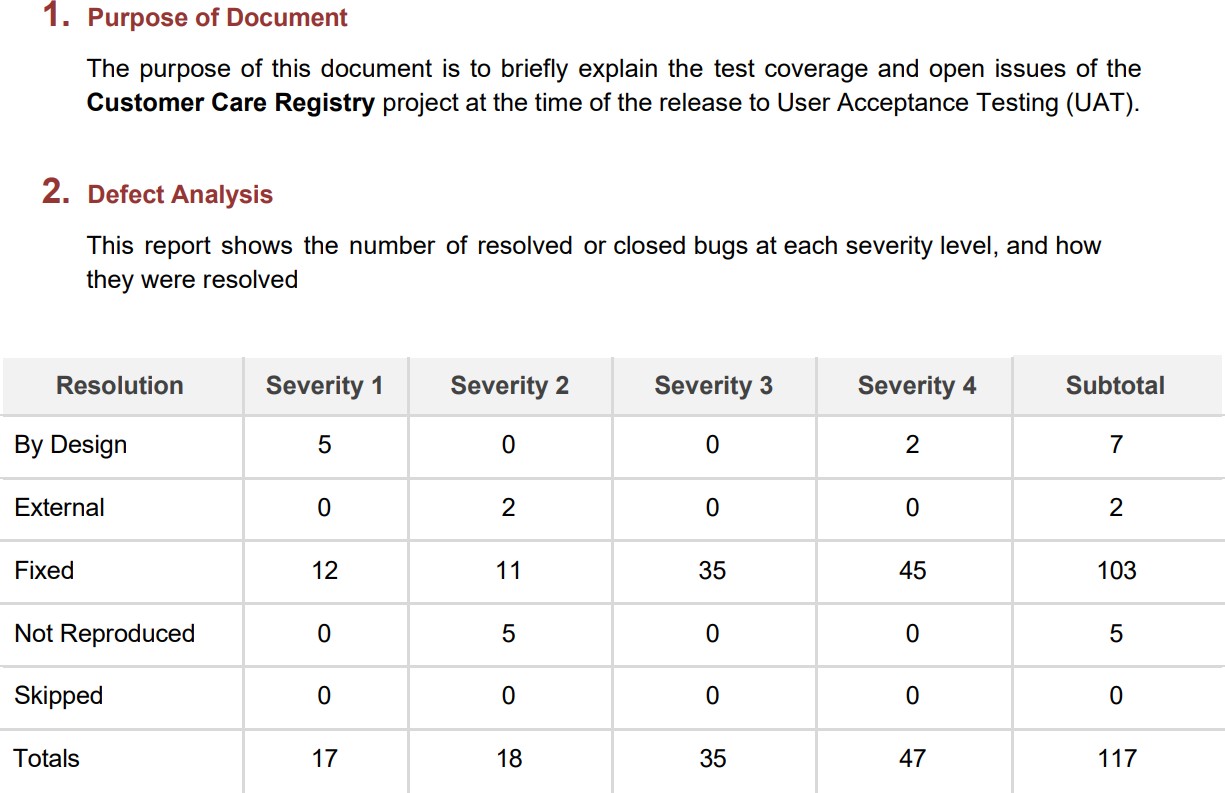
Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not.

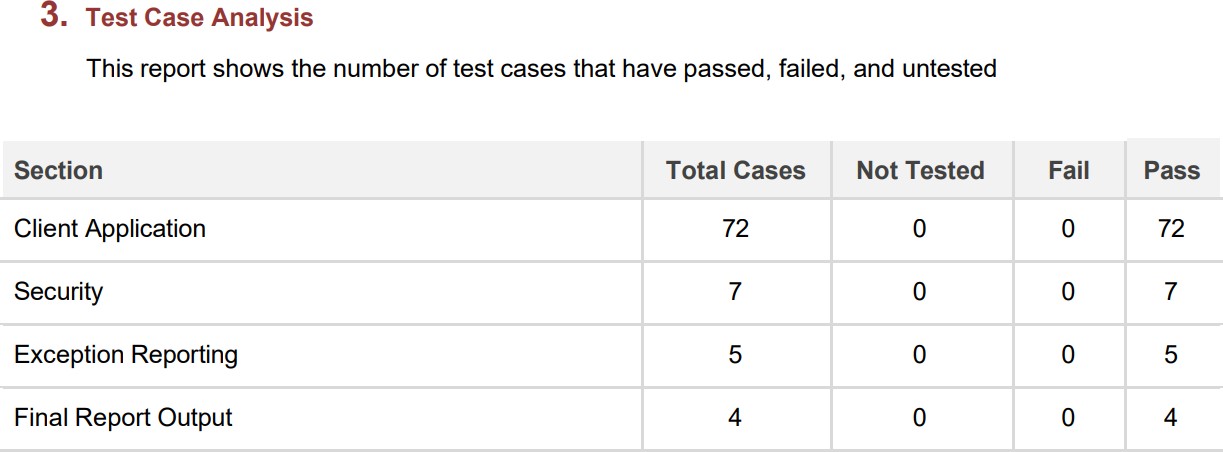
Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.

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.

Test Cases Performed:

* 1. **User Acceptance Testing**



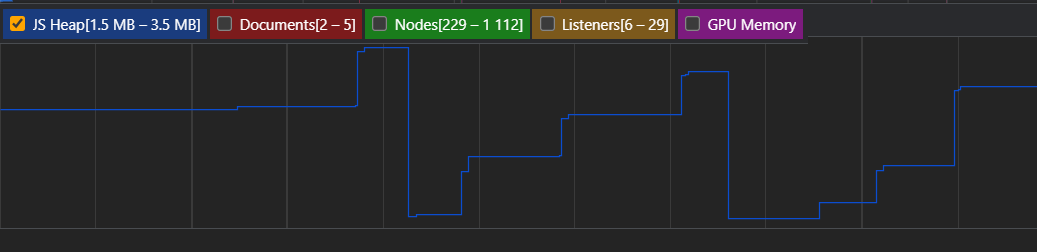


1. **RESULTS**

## Performance Metrics:

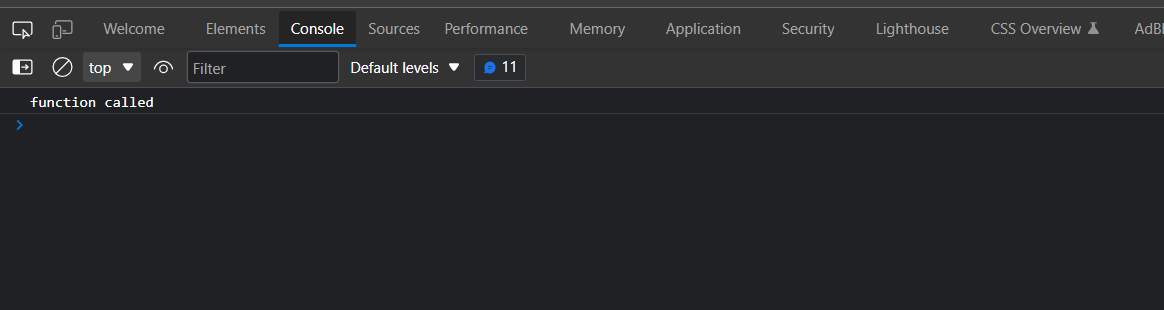
CPU usage:

* + - Since all the operations run using Flask is in server-side, the client (browser) need not worry about the CPU usage. Just rendering the page, static contents take place in the client-side.
    - Memory for client-side functions (Javascript) is allocated using heap. It can be either increased based upon the requirement or removed from the heap.



Errors:

* + - Since all the backend functions are done using flask, any exceptions / errors rising are well- handled. Though they appear, user’s interaction with the site is not affected in any way



## ADVANTAGES

**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

## 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.

## FUTURE SCOPE

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

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

## 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 (Only Samples)

**base,html**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>{% block title %}{% endblock %}</title>

<link rel="icon" type="image" href="{{ url\_for('static', filename='images/cart logo white-modified.png') }}">

<!-- Linking css, js, Google fonts -->

<link rel="preconnect" href="https://fonts.googleapis.com">

<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>

<link rel="stylesheet" href="{{ url\_for('static', filename='css/style.css') }}"/>

<link href[="ht](https://fonts.googleapis.com/css2?family=Roboto%3Aital%2Cwght%400%2C100%3B0%2C300%3B0%2C400%3B0%2C500%3B0%2C700%3B0%2C900%3B1%2C100%3B1%2C300%3B1%2C400)tp[s://fonts.googleapis.com/css2?family=Roboto:ital,wght@0,100;0,300;0,400;0,500;0,700;0,900;1,100;1,300;1,400](https://fonts.googleapis.com/css2?family=Roboto%3Aital%2Cwght%400%2C100%3B0%2C300%3B0%2C400%3B0%2C500%3B0%2C700%3B0%2C900%3B1%2C100%3B1%2C300%3B1%2C400)

;1,500;1,700;1,900&display=swap" rel="stylesheet">

<script src="{{ url\_for('static', filename='js/pass.js') }}"></script>

<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">

<!-- Linking Watson Assistant -->

{% block watson %}

{% endblock %}

</head>

<body>

{% block alert %}

{% if to\_show %}

<script>

alert('{{ message }}')

</script>

{% endif %}

{% endblock %}

{% block main %}

{% endblock %}

</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>

{% endblock %}

## 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 %}

</div>

{% endif %}

</div>

</div>

</div>

{% 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