# GOVERNMENT COLLEGE OF ENGINEERING(Formerly IRTT)
# ERODE-638 316

## BONAFIDE CERTIFICATE

Certified that this project titled **"CUSTOMER CARE REGISTRY "** is the bonafide work of **"GUKAN S (731119104017), MUTHUKUMAR S (731119104030), NAVIEN V (731119104032), SURESHKUMAR R (731119104048)"** who carried out the project work under my supervision.

**SIGNATURE OF HOD**

Dr.A.SARADHA,M.E.,Ph.D.,
HEAD OF THE DEPARTMENT
DEPARTMENT OF CSE,
GOVERNMENT COLLEGE OF
ENGINEERING, ERODE – 638316

**SIGNATURE OF SPOC**

Dr.G.GOWRISON, M.E.,Ph.D.,
ASSISTANT PROFESSOR(SR)
DEPARTMENT OF ECE,
GOVERNMENT COLLEGE OF
ENGINEERING,ERODE - 638316

**SIGNATURE OF FACULTY MENTOR**

Dr.A.KAVIDHA,M.E.,Ph.D.,
ASSISTANT PROFESSOR(SR)
DEPARTMENT OF CSE,
GOVERNMENT COLLEGE OF
ENGINEERING, ERODE – 638316

**SIGNATURE OF FACULTY EVALUATOR**

Dr.A.SARADHA,M.E,Ph.D.,
HEAD OF THE DEPARTMENT
DEPARTMENT OF CSE,
GOVERNMENT COLLEGE OF
ENGINEERING, ERODE - 638316

# PROJECT REPORT FORMAT

**1. INTRODUCTION**

- 1.1 Project Overview
- 1.2 Purpose

**2. LITERATURE SURVEY**

- 2.1 Existing problem
- 2.2 References
- 2.3 Problem Statement Definition

**3. IDEATION & PROPOSED SOLUTION**

- 3.1 Empathy Map Canvas
- 3.2 Ideation & Brainstorming
- 3.3 Proposed Solution
- 3.4 Problem Solution fit

**4. REQUIREMENT ANALYSIS**

- 4.1 Functional requirement
- 4.2 Non-Functional requirements

**5. PROJECT DESIGN**

- 5.1 Data Flow Diagrams
- 5.2 Solution & Technical Architecture
- 5.3 User Stories

**6. PROJECT PLANNING & SCHEDULING**

- 6.1 Sprint Planning & Estimation
- 6.2 Sprint Delivery Schedule
- 6.3 Reports from JIRA

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

- 7.1 Feature 1
- 7.2 Feature 2

**8. TESTING**

8.1 Test Cases

8.2 User Acceptance Testing

**9. RESULTS**

9.1 Performance Metrics

**10. ADVANTAGES & DISADVANTAGES**

**11. CONCLUSION**

**12. FUTURE SCOPE**

**13. APPENDIX**

**Source Code GitHub & Project Demo Link**

# 1. INTRODUCTION

## 1.1 Project Overview

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

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

User: They can register for an account. After the login, they can create 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.

## 1.2 Purpose

Customer care is more than just providing great customer service. It’s a proactive approach to providing information, tools, and services to customers at each point they interact with a brand.

Companies benefit 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.

- Customer service agents spend less time on routine tasks and answering commonly asked questions, enabling agents to do more meaningful tasks.
- Using AI to optimize customer care can increase the bottom line and provide a positive return on investment.

Customer service is reactive. Here, the focus is on helping customers solve problems or answer questions before purchase, either in a self-serve fashion or via the customer support team.

# 2. LITERATURE SURVEY

## 2.1 Existing problem

This software has been developed for a cellular company Concerning all the details given by company. By this software anyone can handle customer complaint details without any difficulty. To maintain customer complaint details and to generate the complaint report to the clients they have to maintain the following information in various files:

1. In the first they record the client's personnel information, such as client code, client name, address, etc. this details are entered in this file when the new client comes into the organization.
2. Then second is used to record the product details of each individual product, this file, this file contain the detail like the product code and all other details concerning about products.
3. They records the complaints of the customers, which we received from the customers. Each complaint is assigned a separate a CCR No. I.e. Customer Complaint Number. This file records the detailed description of the complaint.

## 2.2 References

- Theory and practice of customer related improvements'' ***Daniel Gyllenham maretal*** '' 2022, 92%.It is proposed that future research should address howand when to involve the customer in improvements, and by this aid practitioners. Here, researchers can apply an action research approach to facilitate the enrichment acknowledgement, as those studies utilising action research .
- Improving customer Service in Healthcare''***Muhamma d Ansharietal***'' 2021 89%.The use of ICT in healthcare organizations has grown in the same pattern it is the growing within the larger industry landscape. The use of web technology, database management systems and network infrastructure are part of ICT initiative that willinfluence of healthcare practice and administration.
- Customer Experience modelling from customer experience to services they use design "***Jorge Teixeira,Lia Patrı cioetal***" 2019 90% This multimedia service provided a rich foundation for understanding the complexity of the customer experience and the systematic nature of CEM. New applications to other service contexts would enable further developmentsand refinements of the approach.

## 2.3 Problem Statement Definition

Customer care is a way of dealing with customers when they interact with your brand, products, or services. This Application has been developed to help the customer in processing their complaints.

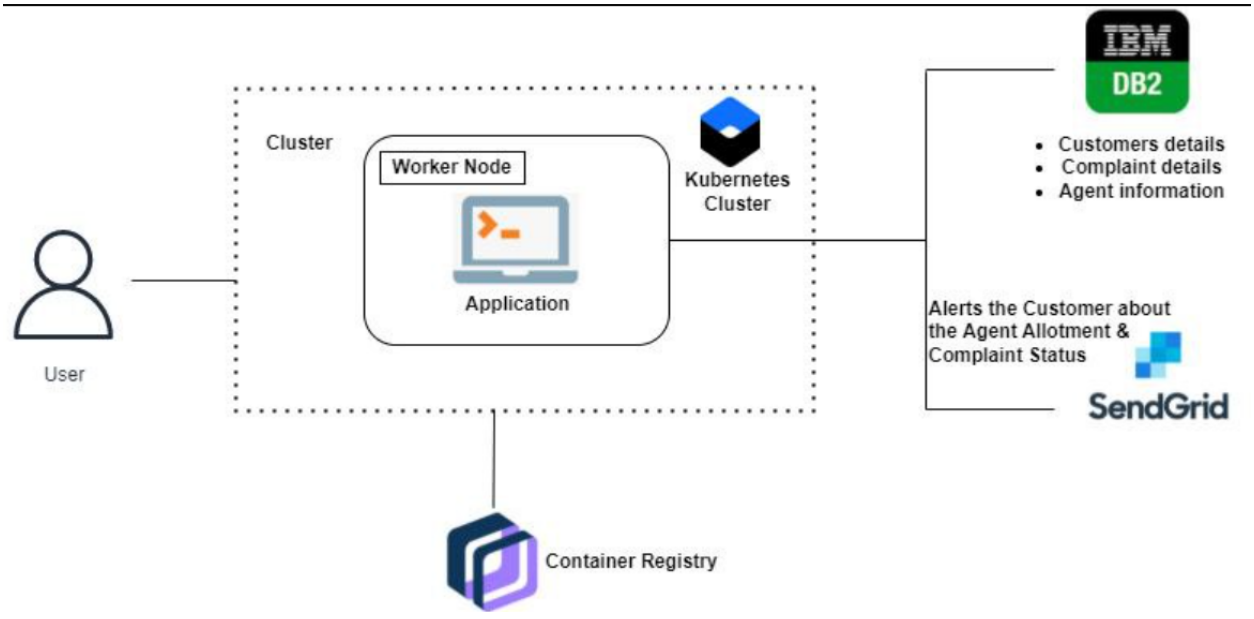
The customers can raise the ticket with a detailed description of the issue. An Agent will be assigned to the Customer to solve the problem. Whenever the agent is assigned to the customer, they will be notified with an email alert. Customers can view the status of the ticket till the service is provided.

Customer can register for an account. After the login, they can create a complaint with a description of the problem they are facing. Each user will be assigned an agent. They can view the status of their complaint. The main roles and responsibilities of the admin is to take care of the whole process.

Starting from Admin login followed by the agent creation and assigning the customers complaints. Finally, he will be able to track the work assigned to the agent and notification will be sent to the customer.

The main use of this project is to help the customer in processing their complaints. The customers can raise the ticket of their issues and the problem will be solved by the organization.

---



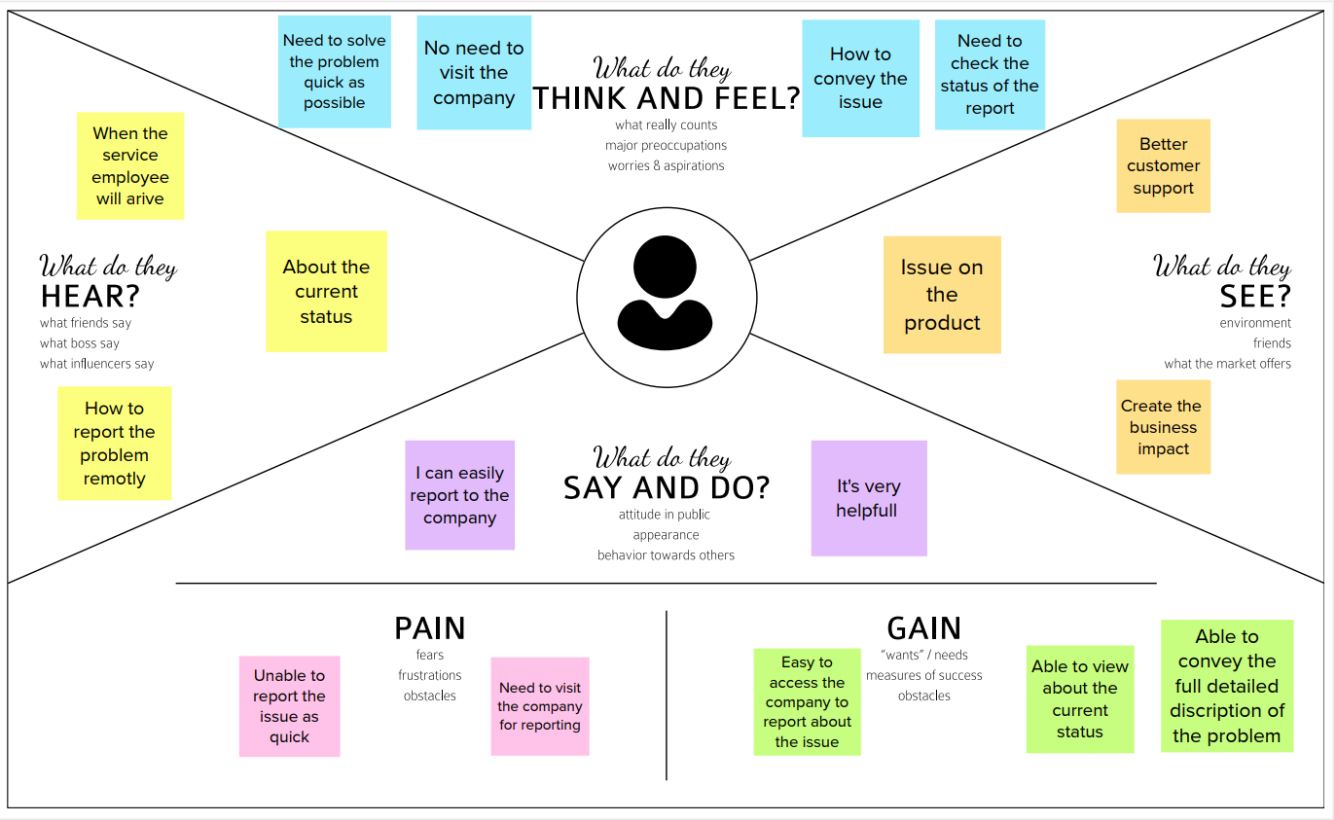
## 3. IDEATION & PROPOSED SOLUTION

### 3.1 Empathy Map Canvas

An empathy map is a collaborative tool teams can use to gain a deeper insight into their customers. Much like a user persona, an empathy map can represent a group of users, such as a customer segment. The empathy map was originally created by Dave Gray and has gained much popularity within the agile community.

Empathy maps should be used throughout any UX process to establish common ground among team members and to understand and prioritize user needs. In user-centered design, empathy maps are best used from the very **beginning of the design process**.

Traditional empathy maps are split into 4 quadrants (**Says, Thinks, Does, and Feels**), with the user or persona in the middle. Empathy maps provide a glance into who a user is as a whole and are not chronological or sequential.

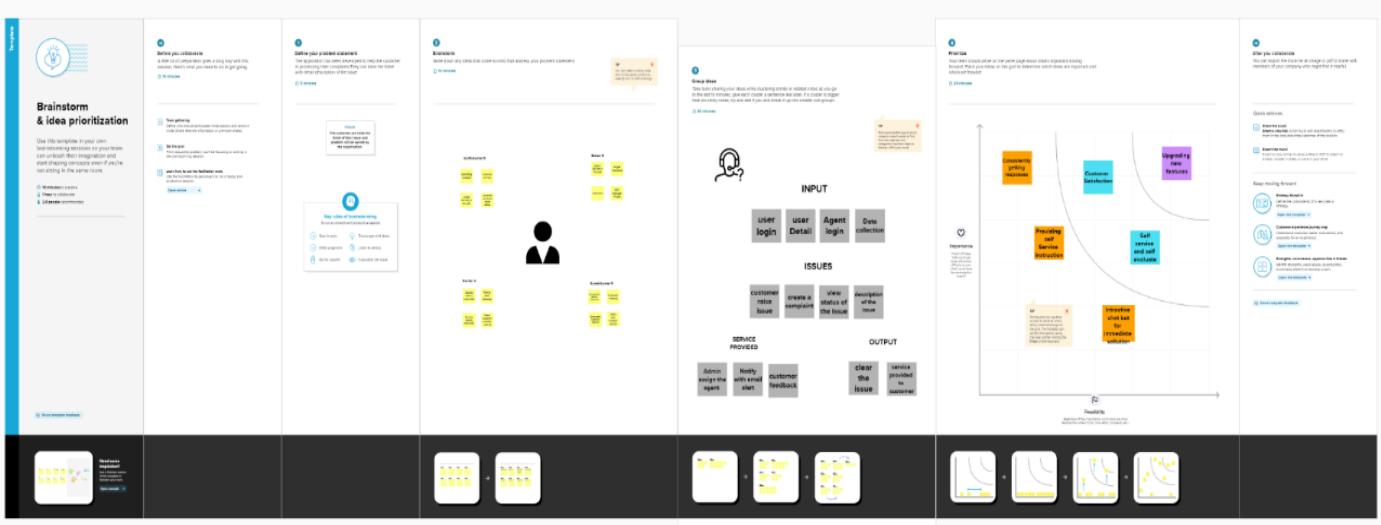


## 3.2 Ideation & Brainstorming

Ideation is the process of forming ideas from conception to implementation, most often in a business setting. Ideation is expressed via graphical, written, or verbal methods, and arises from past or present knowledge, influences, opinions, experiences, and personal conviction.

Ideation is often closely related to the practice of brainstorming, a specific technique that is utilized to generate new ideas. A principal difference between ideation and brainstorming is that ideation is commonly more thought of as being an individual pursuit, while brainstorming is almost always a group activity. Brainstorming is usually conducted by getting a group of people together to come up with either general new ideas or ideas for solving a specific problem or dealing with a specific situation.

Brainstorming is a method of generating ideas and sharing knowledge to solve a particular commercial or technical problem, in which participants are encouraged to think without interruption**.** Brainstorming is a group activity where each participant shares their ideas as soon as they come to mind.



## 3.3 Proposed Solution

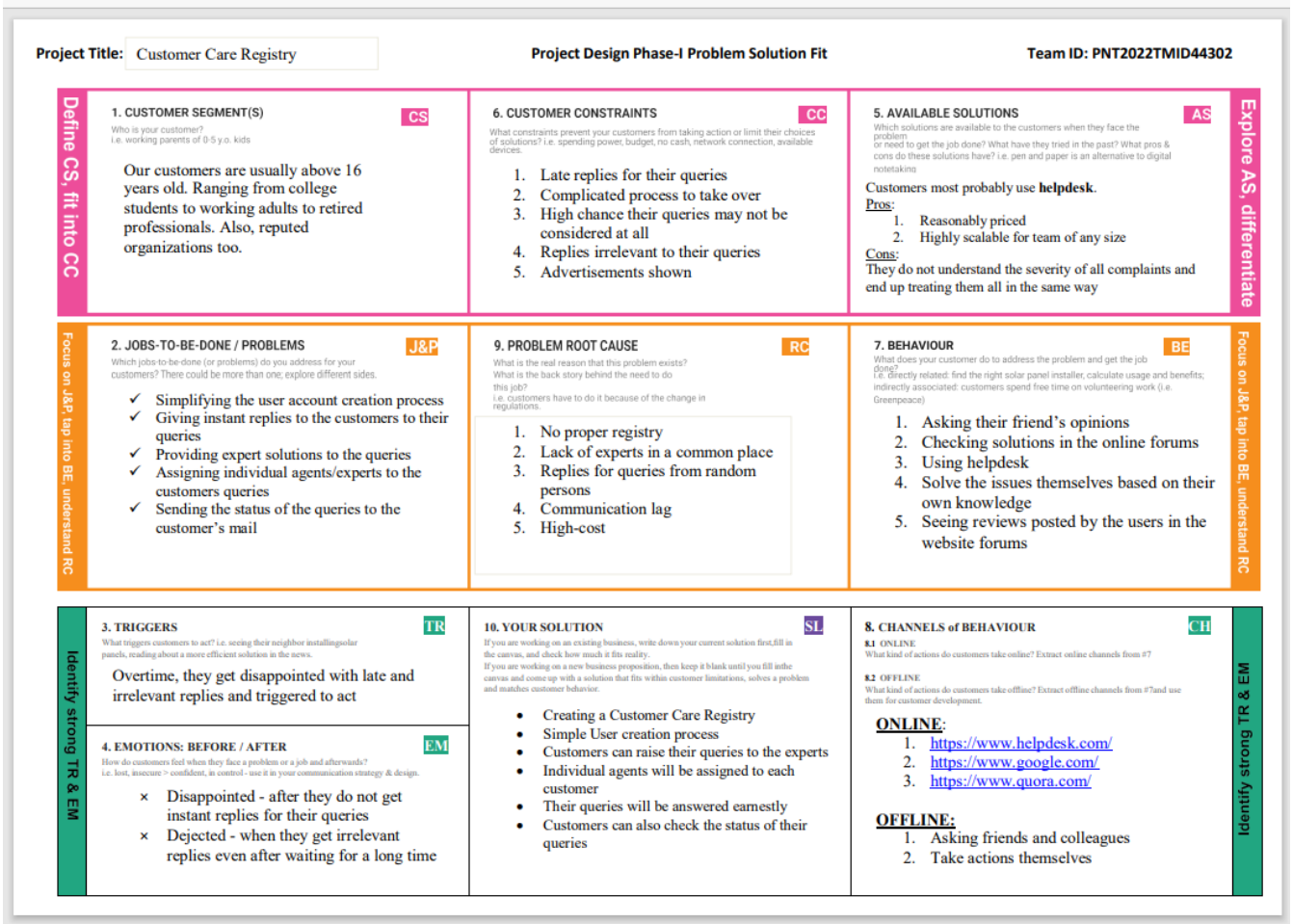
The proposed solution should relate the current situation to a desired result and describe the benefits that will accrue when the desired result is achieved. So, begin your proposed solution by briefly describing this desired result.

| | |
|---|---|
| Problem Statement (Problem to be solved) | Customer service exists to help customers with their needs and/or any problems that come up in doing business. It's the most important part of maintaining a good reputation as a business. |
| Idea / Solution description | Our growing retail business is looking for a skilled problem solver to join our team as a Customer Service Representative |
| Novelty / Uniqueness | At the Novelty Shop you will find unique products along with unique customer service. The shopping experience will be both enjoyable and exciting.<br><br>We will offer many unusual items not found elsewhere. We will constantly strive to offer new novelties and different products to our customers. |
| Social Impact / Customer Satisfaction | Customer satisfaction goes beyond just providing good products or offering great customer service.<br><br>It is the act of making customers feel good about their purchases. It is the idea of making customers feel valued. |
| Business Model (Revenue Model) | In the Business Model Canvas, the Customer Relationships building block describes the type of relationships a business creates with different customer segments.<br><br>Customer relationships are designed around three major goals: customer acquisition, customer retention, and upselling. |

| | |
|---|---|
| Scalability of the Solution | While attending to "customer - facing" processes is important, in the end – those are the internal processes of customer service that underlie and shape the overall customer experience you offer on the outside. |

## 3.4 Problem Solution fit

The Problem-Solution Fit simply means that you have found a problem with your customer and that the solution you have realized for it actually solves the customer's problem.



# 4. REQUIREMENT ANALYSIS

## 4.1 Functional requirement

Functional requirements are define what a product must do, what its features and functions.

| FR No: | Functional Requirement(Epic) | Sub Requirement(Story/ Sub-Task) |
|---|---|---|
| FR-1 | User Registration | Registration through Form<br>Registration through Gmail<br>Registration through LinkedIN |
| FR-2 | User Confirmation | Confirmation via Email<br>Confirmation via OTP |
| FR-3 | User Login | Login via Google<br>Login with Email id and Password |
| FR-4 | Admin Login | Login via Google<br>Login with Email id and Password |
| FR-5 | Query Form | Description of the issues<br>Contact information |
| FR-6 | E-mail | Login alertness |
| FR-7 | Feedback | Customer feedback |

## 4.2 Non-Functional requirements

Nonfunctional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs

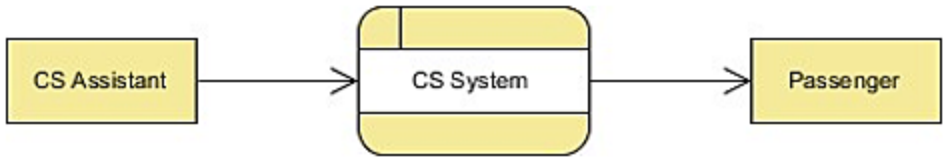
| FR No: | Non-Functional Requirement | Description |
|---|---|---|
| NFR-1 | Usability | To provide the solution to the problem |
| NFR-2 | Security | Track of login authentication |
| NFR-3 | Reliability | Tracking of decade status through email |
| NFR-4 | Performance | Effective development of web application |
| NFR-5 | Availability | 24/7 service |

# 5. PROJECT DESIGN

## 5.1 Data Flow Diagrams

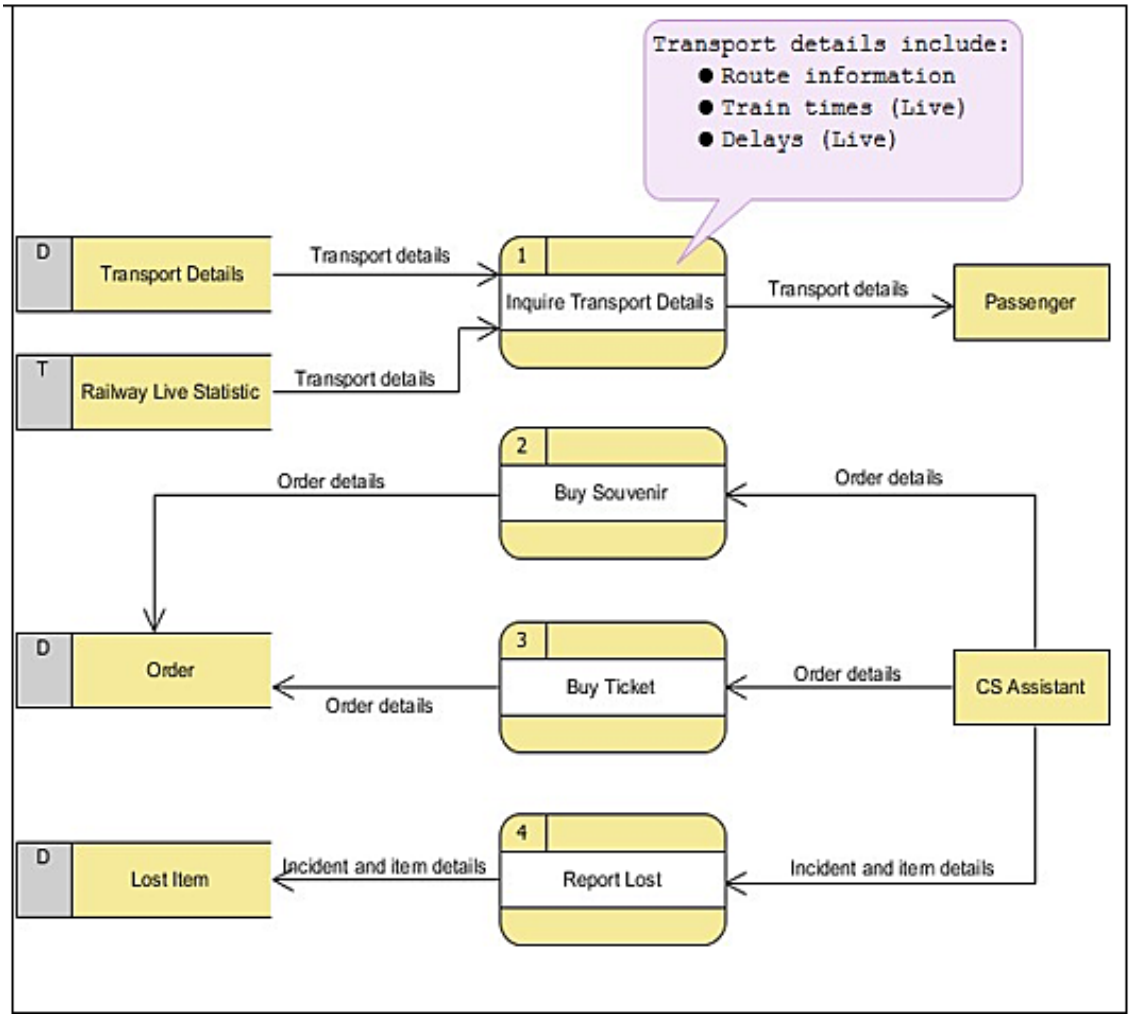
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored

**Example: (Simplified)**




There are two types of DFDs — logical and physical. Logical diagrams display the theoretical process of moving information through a system, like where the data comes from, where it goes, how it changes, and where it ends up.

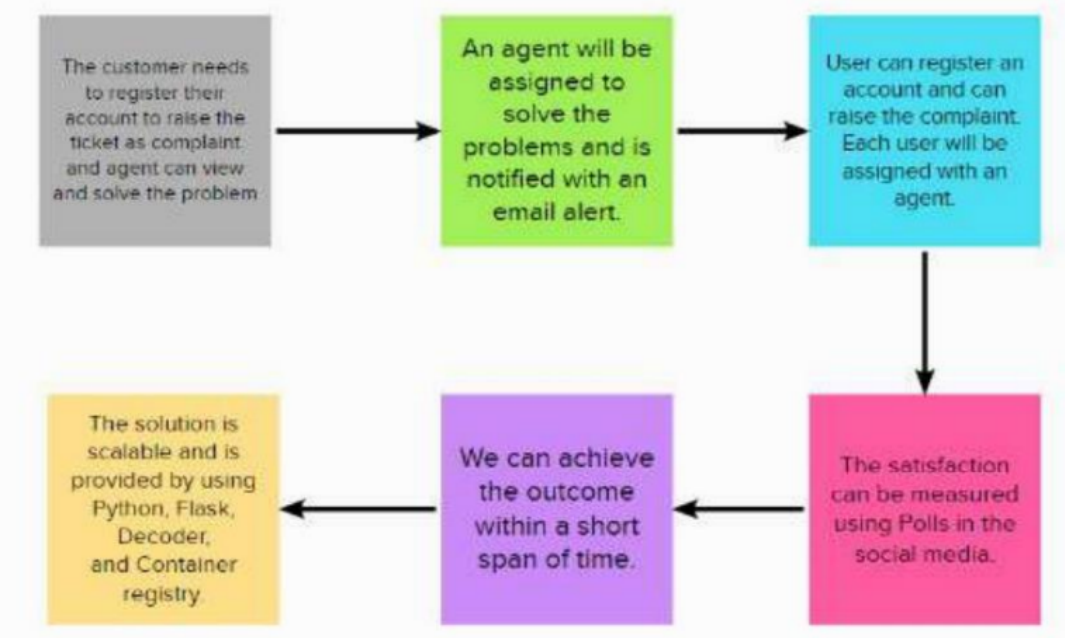
Physical diagrams show you the practical process of moving information through a system, like how your system's specific software, hardware, files, employees, and customers influences its flow of information.




## 5.2 Solution & Technical Architecture

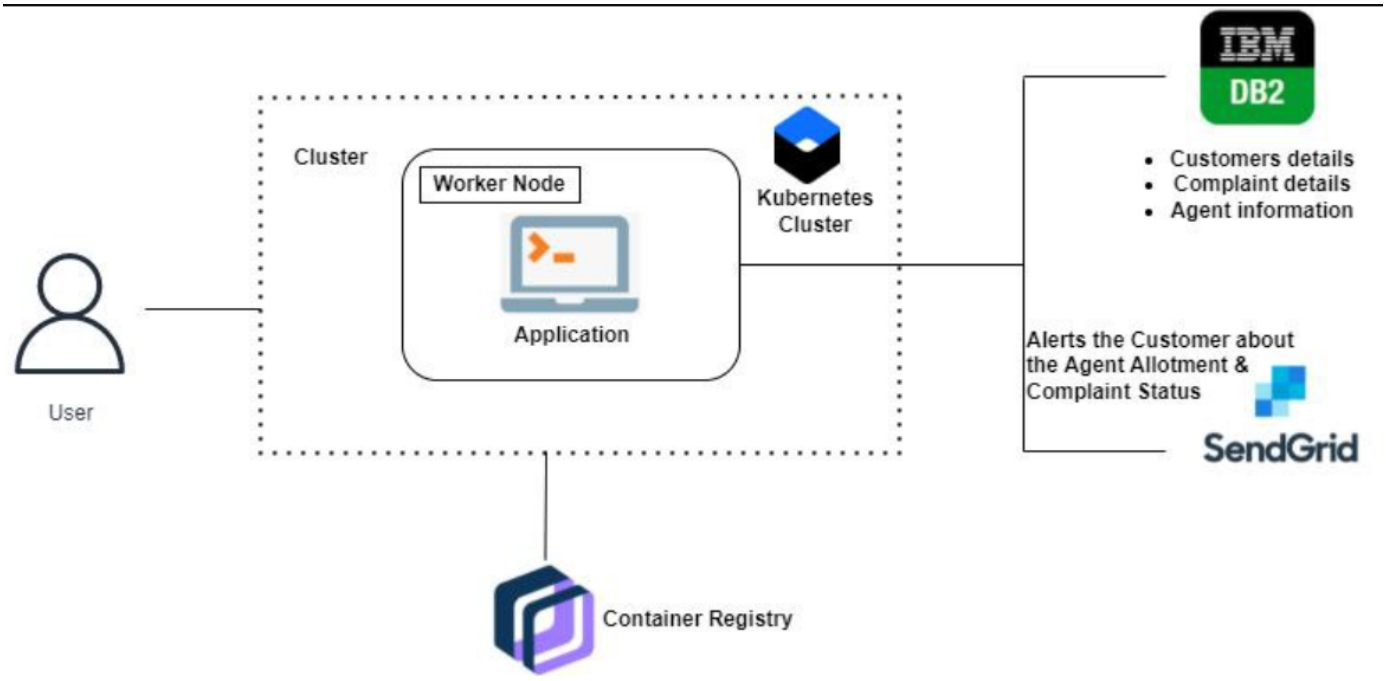
Solution architecture is a complex process with many sub-processes that bridges the gap between business problems and technology solutions. Its goals are to:

- 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




**Components & Technologies:**

| S.No | Component | Description | Technology |
|---|---|---|---|
| 1. | User Interface | Similarly Like Chatbot, Web UI and etc.… | HTML, CSS, JavaScript, Json, Jquery |
| 2. | Application Logic-1 | It helps to perform the Entire Functions and Tasks in the Application. | Python |
| 3. | Application Logic-2 | Providing the Virtual Assistant for Customer Queries | IBM Watson Assistant |
| 4. | Database | Data from config. json is used to configure virtual machine. After that JSON syntax is valid. | JSON |
| 5. | Cloud Database | Database Service on Cloud | IBM DB2 |
| 6. | File Storage | File storage requirements | IBM Block Storage and Object Storage |




## 5.3 User Stories

A user story is an informal, general explanation of a software feature written from the perspective of the end user or customer.

| User Type | Functional (Epic) | User Story Number | User Story / Task | Acceptance criteria | Priority | Release |
|---|---|---|---|---|---|---|
| Customer | User Registration | USN-1 | As a user, I can register for the application by my password. | I can access my account / dashboard | High | Sprint-1 |
| Customer | User Confirmation | USN-2 | As a user, I will receive confirmation email once I have registered for the application | I can receive confirmation email & click confirm | High | Sprint-1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Customer | User Login | USN-3 | I can log into the application by entering email & password | I can login and access my account | High | Sprint-2 |
| Administrator | Admin Login | USN-4 | As an admin, I can log into the application by entering email & password | I can login and access my the customers and agents | High | Sprint-1 |
| Customer | Query Form | USN-5 | As a user, I can raise tickets through the form | I can raise tickets | High | Sprint-1 |
| Agent | E-mail Alert | USN-6 | As a user, I can view the status of tickets for the application | I can see the tickets status | High | Sprint-2 |
| Customer | Feedback | USN-7 | As a user, I can give the customer feedback for the agent who communicated | I can give positive and negative feedback | Medium | Sprint-3 |

# 6. PROJECT PLANNING & SCHEDULING

## 6.1 Sprint Planning & Estimation

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

## Velocity

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

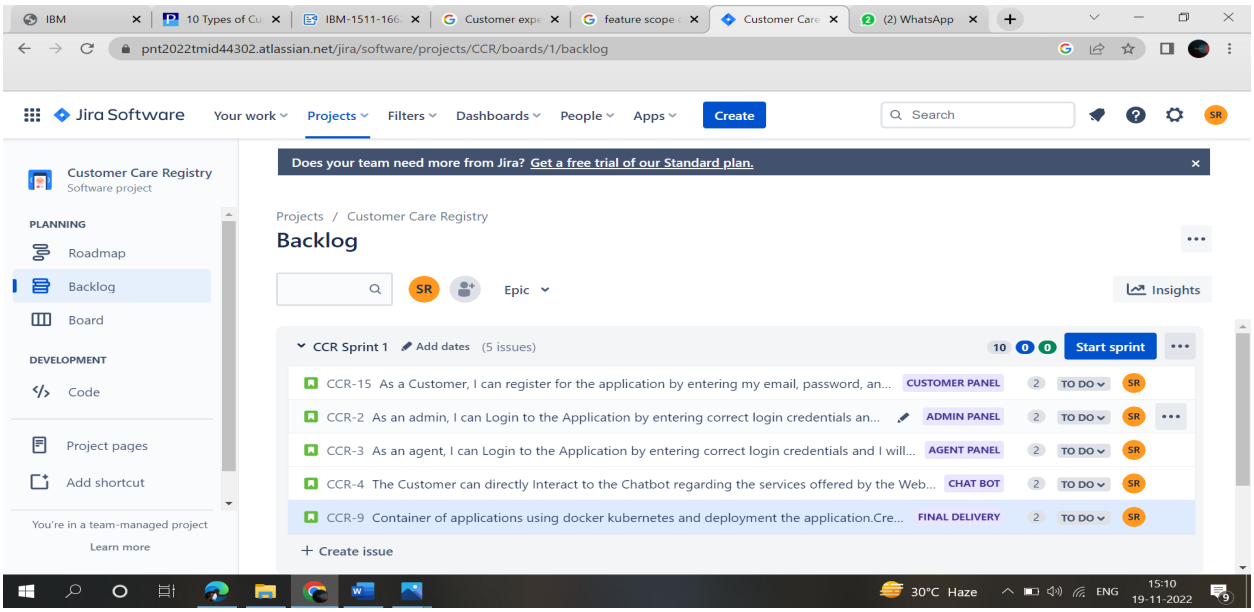
$$AV = \frac{sprint\ duration}{velocity} = \frac{20}{10} = 2$$

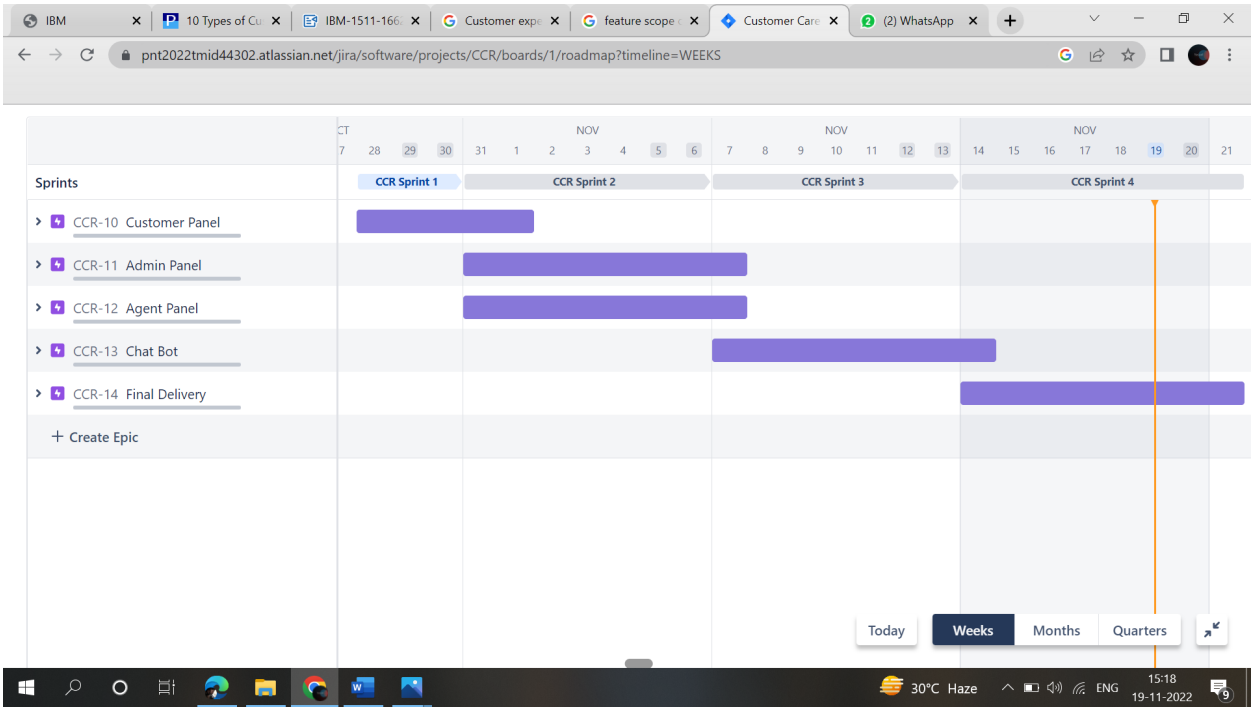
## 6.2 Sprint Delivery Schedule

sprint planning is to define what can be delivered in the sprint and how that work will be achieved. Sprint planning is done in collaboration with the whole scrum team.

| Sprint | Total Story Points | Duration | Sprint Start Date | Sprint End Date (Planned) | Sprint Release Date (Actual) |
|---|---|---|---|---|---|
| Sprint-1 | 20 | 4 Days | 28 Oct 2022 | 01 Nov 2022 | 01 Nov 2022 |
| Sprint-2 | 20 | 7 Days | 31 Oct 2022 | 06 Nov 2022 | 06 Nov 2022 |
| Sprint-3 | 20 | 8 Days | 07 Nov 2022 | 14 Nov 2022 | 14 Nov 2022 |
| Sprint-4 | 20 | 7 Days | 14 Nov 2022 | 21 Nov 2022 | 21 Nov 2022 |

# 6.3 Reports from JIRA



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

**7.1 Feature 1**

**cust**

```
from flask import Blueprint, render_template, request, redirect, session, url_for
from flask_login import login_required, logout_user
import ibm_db
from .views import conn, customer
import uuid
from datetime import date, datetime
from .views import pass_regex
import re
import hashlib

QUERY_STATUS_OPEN = "OPEN"
QUERY_STATUS_ASSIGNED_AGENT = "AGENT ASSIGNED"
QUERY_STATUS_CLOSE = "CLOSE"

cust = Blueprint("customer", __name__)

@cust.route('/customer/')
@login_required
def profile():
    '''
        Custome can see his/her profile card
    '''
    from .views import customer
    if hasattr(customer, 'uuid'):
        return render_template('cust profile.html', customer = customer, id = 0)

    else:
        return redirect(url_for('blue_print.logout'))

@cust.route('/customer/new', methods = ['GET', 'POST'])
@login_required
def new():
    '''
        Customer can create a new ticket
    '''
```

```
    from .views import customer

    if(hasattr(customer, 'uuid')):
        if request.method == 'POST':
            # collecting the query entered by the customer in the
textarea
            query = request.form.get('query-box')

            msg = ""
            to_show = False

            if(len(query) == 0):
                msg = "Query cannot be empty!"
                to_show = True

            else:
                # updating the query in the database
                update_query = '''
                    INSERT INTO tickets
                        (ticket_id, raised_by, raised_on, issue,
query_status)
                    VALUES
                        (?, ?, ?, ?, ?)
                '''

                try:
                    stmt = ibm_db.prepare(conn, update_query)

                    # creating a uuid for the ticket_id
                    ticket_id = str(uuid.uuid4())
                    raised_by = customer.uuid
                    raied_on = datetime.now()

                    ibm_db.bind_param(stmt, 1, ticket_id)
                    ibm_db.bind_param(stmt, 2, raised_by)
                    ibm_db.bind_param(stmt, 3, raied_on)
                    ibm_db.bind_param(stmt, 4, query)
                    ibm_db.bind_param(stmt, 5, QUERY_STATUS_OPEN)

                    ibm_db.execute(stmt)

                    msg = "Ticket created!"
                    to_show = True

```

```
78                  except:
79                      msg = "Something went wrong!"
80                      to_show = True
81
82              return render_template('cust new ticket.html', id = 1,
   to_show = to_show, message = msg)
83
84          return render_template('cust new ticket.html', id = 1)
85
86      else:
87          # logging out
88          return redirect(url_for('blue_print.logout'))
89
90 @cust.route('/customer/tickets')
91 @login_required
92 def tickets():
93      '''
94          Fetching all the tickets raised by the customer
95      '''
96      from .views import customer
97
98      if(hasattr(customer, 'uuid')):
99          fetch_query = '''
100             SELECT
101                 tickets.ticket_id,
102                 tickets.raised_on,
103                 tickets.query_status,
104                 agent.first_name,
105                 tickets.issue
106             FROM
107                 tickets
108             LEFT JOIN
109                 agent ON agent.agent_id = tickets.assigned_to AND
110                     tickets.raised_by = ? ORDER BY tickets.raised_on
   DESC
111         '''
112
113         # I am using the LEFT JOIN because
114             # the customer should see both the assigned and unassigned
   tickets
115         # Left side table - Tickets
116         # Right side table - Agent
117
118         # So all the tickets of the customer gets loaded (all the things
```

```
   in left table)
            # and the agents on the side

        from .views import customer
        raised_by = customer.uuid

        try:
            stmt = ibm_db.prepare(conn, fetch_query)
            ibm_db.bind_param(stmt, 1, raised_by)
            ibm_db.execute(stmt)

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

            if tickets:
                # means, the customer has raised some tickets before
                while tickets != False:
                    temp = []

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

                    tickets_list.append(temp)

                    tickets = ibm_db.fetch_assoc(stmt)

                return render_template(
                    'cust tickets.html',
                    id = 2,
                    tickets_to_show = True,
                    tickets = tickets_list,
                    msg = "These are your tickets"
                )
```

### 7.2 Feature 2

## 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

```

```
36              else:
37                  # logging out the so called customer
38                  return redirect(url_for('blue_print.logout'))
39
40          elif session['LOGGED_IN_AS'] == "AGENT":
41              # checking if the agent is really logged in
42              # by checking, if the agent has uuid aatribute
43              from .views import agent
44
45              if (hasattr(agent, 'uuid')):
46                  user = "AGENT"
47                  sender_id = agent.uuid
48                  can_trust = True
49
50          else:
51              # Admin is the one who logged in
52              # admin should not see the chats, sp directly logging the
    admin out
53              return redirect(url_for('blue_print.logout'))
54
55          to_show = False
56          message = ""
57
58          if can_trust:
59              # importing the connection string
60              from .views import conn
61
62              if request.method == 'POST':
63                  # chats are enabled, only if the ticket is OPEN
64                  # getting the data collected from the customer / agent
65                  myMessage = request.form.get('message-box')
66
67                  if len(myMessage) == 0:
68                      to_show = True
69                      message = "Type something!"
70
71                  else:
72                      # inserting the message in the database
73
74                      # query to insert the message in the database
75                      message_insert_query = '''
76                          INSERT INTO chat
77                              (chat_id, sender_id, message, sent_at)
78                          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)
```

# 8.TESTING

## 8.1 Test Cases

A test case is a set of actions performed on a system to determine if it satisfies software requirements and functions correctly.

| | | | | PNT2022TMID44302 | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Project - Customer Care Registr | | | | | | |
| | | | | 4 marks | | | | | | |
| Test case ID | Feature Type | Component | Test Scenario | Steps To Execute | Test Data | Expected Result | Actual Result | Status | Commenets | Executed By |
| 1 | Functional | Registration Page | Customer is trying to register with the invalid data | 1. Go to application<br>2. Click "Don't have an account? Register"<br>3. Enter first name, last name, select the role, password and confirm password<br>4. Click Register button | First Name = Bala<br>Last Name = Abinesh<br>Role = Customer<br>Email = suryathaya10@gmail.com<br>Password = 12345678<br>Confirm Password = 123456789 | Customer should get an alert saying "Passwords do not match" | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |
| 2 | Functional | Registration Page | Customer is trying to register with the invalid data | 1. Go to application<br>2. Click "Don't have an account? Register"<br>3. Enter first name, last name, select the role, password and confirm password<br>4. Click Register button | First Name = Bala<br>Last Name = Abinesh<br>Role = Customer<br>Email = suryathaya10gmail.com<br>Password = 12345678<br>Confirm Password = 12345678 | Customer should get an alert saying "Invalid email" | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |
| 3 | Functional | Registration Page | Customer is trying to register with the invalid data | 1. Go to application<br>2. Click "Don't have an account? Register"<br>3. Enter first name, last name, select the role, password and confirm password<br>4. Click Register button | First Name = Ba<br>Last Name = Abinesh<br>Role = Customer<br>Email = suryathaya10@gmail.com<br>Password = 12345678<br>Confirm Password = 12345678 | Customer should get an alert saying "Firstname should be atleast 6 characters long!" | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |
| 4 | Functional | Login page | Customer is trying to register with the invalid data | 1. Go to application<br>2. Click "Don't have an account? Register"<br>3. Enter first name, last name, select the role, password and confirm password<br>4. Click Register button | First Name = Bala<br>Last Name = Abinesh<br>Role = Customer<br>Email = suryathaya10@gmail.com<br>Password = 1234<br>Confirm Password = 1234 | Customer should get an alert saying "Passwords must be at least 8 characters long!" | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |
| 5 | Functional | Registration Page | Customer is trying to register with the valid data | 1. Go to application<br>2. Click "Don't have an account? Register"<br>3. Enter first name, last name, select the role, password and confirm password<br>4. Click Register button | First Name = Bala<br>Last Name = Abinesh<br>Role = Customer<br>Email = suryathaya@10gmail.com<br>Password = 12345678<br>Confirm Password = 12345678 | Customer's profile is added in the database and the customer is registered. Then, the customer is re-directed to the Login page to login | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |
| 6 | Functional | Login Page | Customer is trying to login using the invalid credentials | 1. Go to application<br>2. Enter email, password<br>3. Click Login | Email = suryathaya10gmail.com<br>Password = 12345678 | Customer should get an alert saying "Invalid email" | Working as expected | Pass | Everything is working properly | MUTHUKUMAR |

## 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

| Section | Total Cases | Not Tested | Fail | Pass |
|---|---|---|---|---|
| Client Application | 72 | 0 | 0 | 72 |
| Security | 7 | 0 | 0 | 7 |
| Exception Reporting | 5 | 0 | 0 | 5 |

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

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

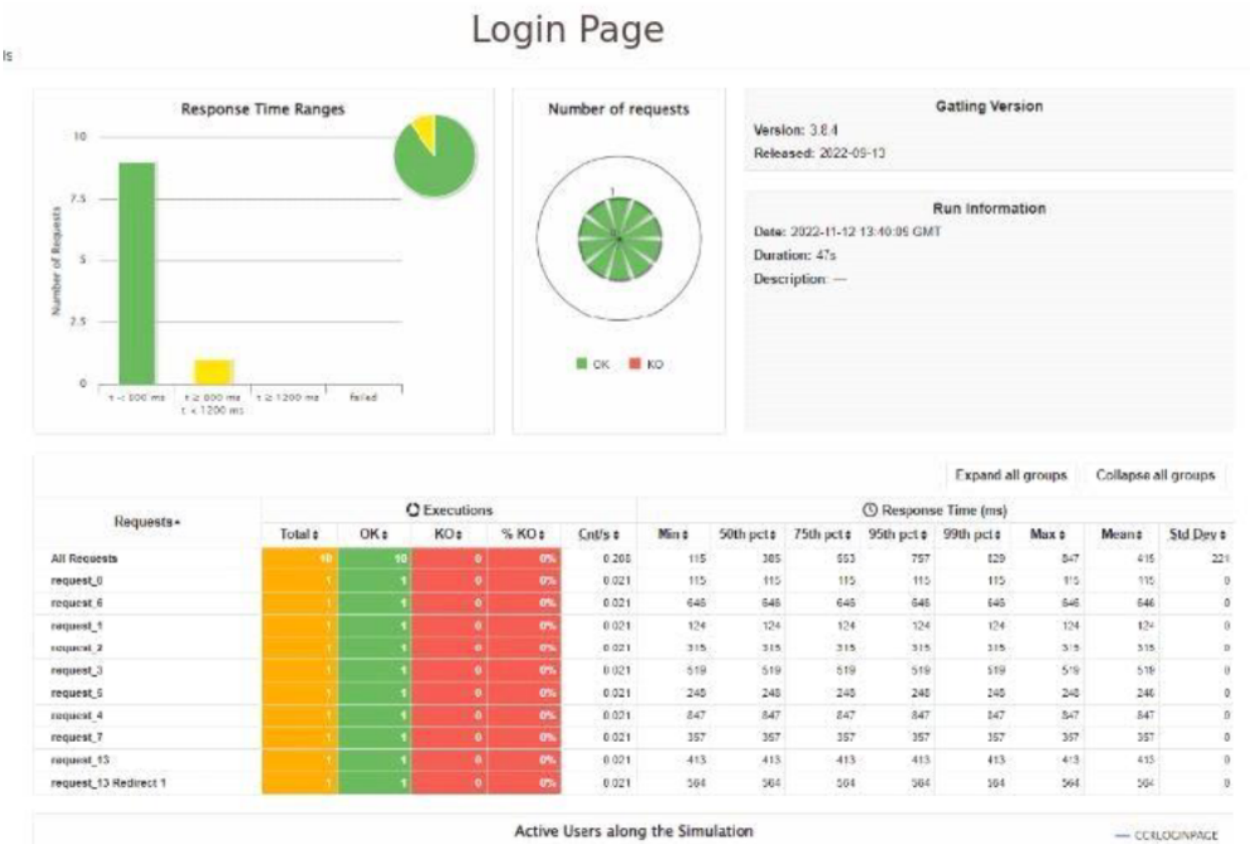
## Test Case Analysis

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

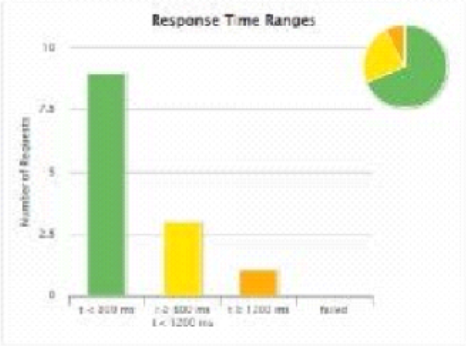
| | | | | |
|---|---|---|---|---|
| Final Report Output | 4 | 0 | 0 | 4 |

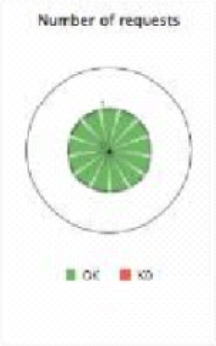
# 9. RESULTS

## 9.1 Performance Metrics



# Admin Page




**Gatling Version**

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

**Run Information**

**Date:** 2023-11-13 04:57:52 GMT
**Duration:** 26s
**Description:** —

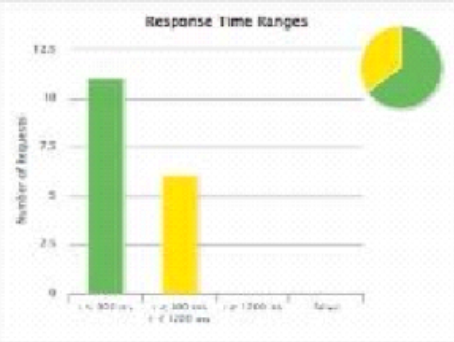
Expand all groups | Collapse all groups

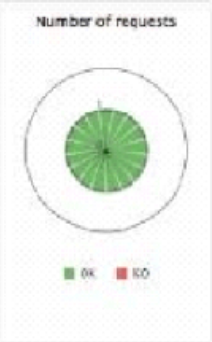
| Requests* | Executions | | | | | Response Time (ms) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Total ⇕ | OK ⇕ | KO ⇕ | % KO ⇕ | Cnt/s ⇕ | Min ⇕ | 50th pct ⇕ | 75th pct ⇕ | 95th pct ⇕ | 99th pct ⇕ | Max ⇕ | Mean ⇕ | Std Dev ⇕ |
| All Requests | 13 | 13 | 0 | 0% | 0.481 | 184 | 440 | 889 | 1371 | 1727 | 1810 | 814 | 488 |
| request_0 | 1 | 1 | 0 | 0% | 0.037 | 184 | 184 | 184 | 184 | 184 | 184 | 184 | 0 |
| request_6 | 1 | 1 | 0 | 0% | 0.037 | 883 | 883 | 883 | 883 | 883 | 883 | 883 | 0 |
| request_2 | 1 | 1 | 0 | 0% | 0.037 | 889 | 889 | 889 | 889 | 889 | 889 | 889 | 0 |
| request_3 | 1 | 1 | 0 | 0% | 0.037 | 271 | 271 | 271 | 271 | 271 | 271 | 271 | 0 |
| request_5 | 1 | 1 | 0 | 0% | 0.037 | 343 | 343 | 343 | 343 | 343 | 343 | 343 | 0 |
| request_4 | 1 | 1 | 0 | 0% | 0.037 | 268 | 268 | 268 | 268 | 268 | 268 | 268 | 0 |
| request_1 | 1 | 1 | 0 | 0% | 0.037 | 800 | 800 | 800 | 800 | 800 | 800 | 800 | 0 |
| request_7 | 1 | 1 | 0 | 0% | 0.037 | 440 | 440 | 440 | 440 | 440 | 440 | 440 | 0 |
| request_17 | 1 | 1 | 0 | 0% | 0.037 | 440 | 440 | 440 | 440 | 440 | 440 | 440 | 0 |
| request_13 Redirect 1 | 1 | 1 | 0 | 0% | 0.037 | 1810 | 1810 | 1810 | 1810 | 1810 | 1810 | 1810 | 0 |
| request_18 | 1 | 1 | 0 | 0% | 0.037 | 1005 | 1005 | 1005 | 1005 | 1005 | 1005 | 1005 | 0 |
| request_21 | 1 | 1 | 0 | 0% | 0.037 | 759 | 759 | 759 | 759 | 759 | 759 | 759 | 0 |
| request_20 | 1 | 1 | 0 | 0% | 0.037 | 311 | 311 | 311 | 311 | 311 | 311 | 311 | 0 |

**Active Users along the Simulation**

Zoom: All

# Complaint form




Gatling Version

Version: 3.8.4

Released: 2022-09-15

Run Information

Date: 2022-11-13 06:14:38 GMT

Duration: 1m 6s

Description: —

Expand all groups | Collapse all groups

| Requests | Executions | | | | | Response Time (ms) | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Total | OK | KO | % KO | Cnt/s | Min | 50th pct | 75th pct | 95th pct | 99th pct | Max | Mean | Std Dev |
| All Requests | 12 | 12 | 0 | 0% | 0.264 | 270 | 928 | 1092 | 1121 | 1126 | 1127 | 712 | 276 |
| request_0 | 1 | 1 | 0 | 0% | 0.015 | 270 | 270 | 270 | 270 | 270 | 270 | 270 | 0 |
| request_6 | 1 | 1 | 0 | 0% | 0.015 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 1130 | 0 |
| request_1 | 1 | 1 | 0 | 0% | 0.015 | 557 | 557 | 557 | 557 | 557 | 557 | 557 | 0 |
| request_2 | 1 | 1 | 0 | 0% | 0.015 | 628 | 628 | 628 | 628 | 628 | 628 | 628 | 0 |
| request_3 | 1 | 1 | 0 | 0% | 0.015 | 929 | 929 | 929 | 929 | 929 | 929 | 929 | 0 |
| request_5 | 1 | 1 | 0 | 0% | 0.015 | 851 | 851 | 851 | 851 | 851 | 851 | 851 | 0 |
| request_4 | 1 | 1 | 0 | 0% | 0.015 | 944 | 944 | 944 | 944 | 944 | 944 | 944 | 0 |
| request_7 | 1 | 1 | 0 | 0% | 0.015 | 427 | 427 | 427 | 427 | 427 | 427 | 427 | 0 |
| request_10 | 1 | 1 | 0 | 0% | 0.015 | 499 | 499 | 499 | 499 | 499 | 499 | 499 | 0 |
| request_10 Redirect 1 | 1 | 1 | 0 | 0% | 0.015 | 1127 | 1127 | 1127 | 1127 | 1127 | 1127 | 1127 | 0 |
| request_18 | 1 | 1 | 0 | 0% | 0.015 | 1126 | 1126 | 1126 | 1126 | 1126 | 1126 | 1126 | 0 |
| request_19 | 1 | 1 | 0 | 0% | 0.015 | 516 | 516 | 516 | 516 | 516 | 516 | 516 | 0 |
| request_20 | 1 | 1 | 0 | 0% | 0.015 | 893 | 893 | 893 | 893 | 893 | 893 | 893 | 0 |
| request_20 Redirect 1 | 1 | 1 | 0 | 0% | 0.015 | 1092 | 1092 | 1092 | 1092 | 1092 | 1092 | 1092 | 0 |
| request_36 | 1 | 1 | 0 | 0% | 0.015 | 1081 | 1081 | 1081 | 1081 | 1081 | 1081 | 1081 | 0 |
| request_37 | 1 | 1 | 0 | 0% | 0.015 | 854 | 854 | 854 | 854 | 854 | 854 | 854 | 0 |
| request_38 | 1 | 1 | 0 | 0% | 0.015 | 847 | 847 | 847 | 847 | 847 | 847 | 847 | 0 |

# 10. ADVANTAGES & DISADVANTAGES

## 10.1 ADVANTAGES

Customer service is the support you offer your customers both before and after they buy and use your products or services that helps them have an easy and enjoyable experience with you.

- Improves team-based care.
- Maintain a positive attitude.
- Creatively problem-solve.
- Respond quickly.
- Personalize your service.
- Help customers help themselves.
- Focus support on the customer.
- Customer service is reactive.
- Provides a dashboard of who needs what.
- Reduce your support ticket volume significantly

## 10.2 DISADVANTAGES

- Delayed email responses can make customers feel frustrated.
- Typing long replies can be time-consuming.
- Lack of real-time human-to-human interaction.
- It becomes difficult for agents to read customer emotions.
- Robotic responses can frustrate customers and force them to speak with an agent.
- Customer service agents cannot handle multiple calls at the same time.
- Impossible to cover solutions for all customer problems or requests.
- Negative customer reviews or comments can impact brand reputation.
- You will have to update your existing help content or add new pages constantly.
- Chat replies can often feel scripted and robotic to customers.

## 11. CONCLUSION

customer In conclusion,care, involves the use of basics and any company whowants to have success and grow, needs to remember, that in order to do so, it must begin with establishing a codes in regards to how each employee is to handle the dealing with customers. Customers are at the heart of the company and its growth or decline. Customer careinvolves, the treatment, care, loyalty, trust the employee should extend to the consumer, as well in life. This concept can be applied to so much more than just customer care. People need to treat others with respect and kindness, people should try to take others into consideration when making any decision. If more people were to practice this policy, chances are the world would bea better, more understanding place for all to exist.

A manager should be mindful of every interaction with a customer, including product delivery, customer complaints, billing and any problem resolution. Each of these interactions needs to be polite, efficient and convenient to the customer. And for the customer to notice, they need to be consistently pleasing.Customer satisfaction is addressed as a strategic business development tool. Customer satisfaction does have a positive effect on an organization's profitability,

Customer care systems allow enterprises to manage multiple customer conversations easily. They also help drive a culture of innovation within customer relationship management, through team empowerment and simplified access to caller insights. Therefore, firms must ensure that these 10 features are available as core offerings when reviewing scalable customer support systems.

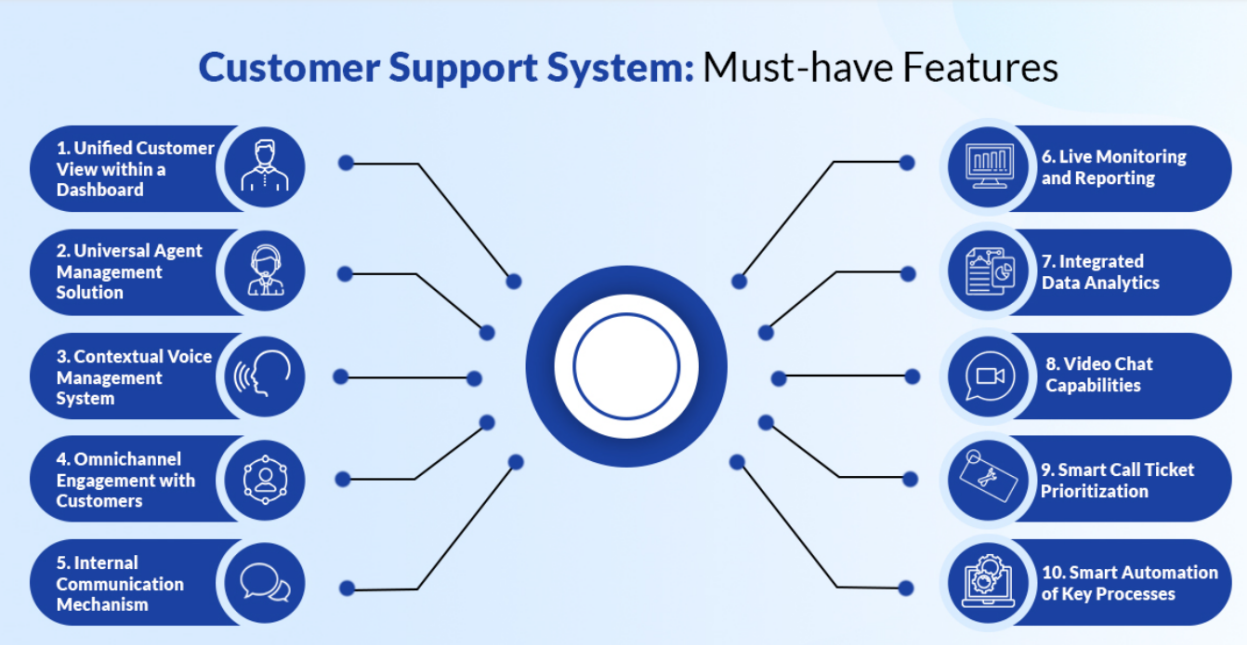
While having a ticket management system is the first step towards exceptional customer support, the story does not end here. Let us look at some ways to be smart about managing the customer support ticket to enhance the agent and customer experience.

## 12. FUTURE SCOPE

The future of customer service increasingly will be driven by technology innovations. Ideally, these new technologies will improve customer and agent experiences, along with business metrics like revenue, operational costs and customer ratings.Over the years, customer expectations generally haven't changed. Customers want to be served quickly and completely on the first try. If they're speaking to a human agent, they want a friendly, knowledgeable interaction -- the goal being to resolve the customer's problem or answer their question quickly and easily.

As we move towards a more remote and digitized world, organizations need greater transparency when engaging with customers at scale. The pandemic has demonstrated the need for managers to have real-time access to any interaction at any time.The dedicated interaction tools made available to customer service agents empowers them to make the right decisions at the right time. This significantly improves their productivity, enabling customer engagement to emerge as a profit center.

A customer support ticketing system is adept at managing large volumes of customer conversations with ease. Automated assignment and optimized resolution workflows also significantly improve customer satisfaction with service levels.



## 13. APPENDIX

### model.py

```
from flask_login import UserMixin
import smtplib
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email.mime.multipart import MIMEMultipart
from email import encoders
import os
class Customer(UserMixin):
    def set(self, uuid, first_name, last_name, email, password, date):
        '''
            Method to initialise the Customer
        '''
        self.uuid = uuid
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.password = password
        self.date = date
    def get_id(self):
        '''
            Method to return the uuid of the Customer
        '''
        return (self.uuid)
class Agent(UserMixin):
    def set(self, uuid, first_name, last_name, email, password, date,
confirm):
        '''
            Method to initialise the Agent
        '''
        self.uuid = uuid
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.password = password
        self.date = date
        self.confirm = confirm
    def get_id(self):
        '''
```

```
38              Method to return the uuid of the Agent
39          '''
40          return (self.uuid)
41  class Admin(UserMixin):
42      def set(self, email, password):
43          self.email = email
44          self.password = password
45      def get_id(self):
46          '''
47              Method to return the email of the Admin
48          '''
49          return (self.email)
50  class Mail():
51      # mail server essentials
52      from .secret import email, password
53      smptpHost = "smtp.gmail.com"
54      smtpPort = 587
55      mailUName = email
56      mailPwd = password
57      fromMail = email
58      # mail body, subject
59      mailSubject = ""
60      mailContent = ''
61      recipient = []
62
63      def sendEmail(self, subject, content, receivers):
64          msg = MIMEMultipart()
65
66          msg['From'] = self.fromMail
67          msg['To'] = ','.join(receivers)
68          msg['Subject'] = subject
69          msg.attach(MIMEText(content, 'html'))
70
71          # sending the message object
72          s = smtplib.SMTP(self.smptpHost, self.smtpPort)
73          s.starttls()
74          s.login(self.mailUName, self.mailPwd)
75          msgText = msg.as_string()
76          sendErrs = s.sendmail(self.fromMail, receivers, msgText)
77
78          s.quit()
79
80          return sendErrs
```

## views.py

```
from flask import Blueprint, render_template, request, redirect, session, url_for
import hashlib
import re
from flask_login import login_required, login_user, logout_user
import ibm_db
import uuid
from datetime import date
import random
from registry.model import Customer, Agent, Admin, Mail
from ..secret import connection_string

views = Blueprint("blue_print", __name__)
email_regex = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
pass_regex = r"^[A-Za-z0-9_-]*$"

customer = Customer()
agent = Agent()
admin = Admin()
mail = Mail()

conn = ibm_db.connect("DATABASE=bludb; HOSTNAME=98538591-7217-4024-b027-8baa776ffad1.c3n41cmd0nqnrk39u98g.databases.appdomain.cloud; PORT=30875; SECURITY=SSL; SSLServerCertificate=DigiCertGlobalRootCA.crt; UID=fwy33330; PWD=rJzjTsde9xhr3VYS", '', '')

@views.route('/logout')
@login_required
def logout():
    session.pop('LOGGED_IN_AS')
    logout_user()

    return redirect(url_for('blue_print.login'))

@views.route('/', methods = ['GET', 'POST'])
@views.route('/login', methods = ['GET', 'POST'])
def login():
    # if method is POST
    if request.method == 'POST':
        # getting the data entered by the user
        email = request.form.get('email')
```

```
38          password = request.form.get('password')
39          role = request.form.get('role-check')
40
41          msg = ""
42          to_show = False
43
44          # validating the inputs entered by the user
45          if(not (re.fullmatch(email_regex, email))):
46              msg = "Enter a valid email"
47              to_show = True
48
49          elif (len(password) < 8):
50              msg = "Password must be atleast 8 characters long!"
51              to_show = True
52
53          # Admin login
54          if email == "admin.ccr@gmail.com":
55              if password == "admin.ccr@2022":
56                  # initialising admin object
57                  admin.set(email, password)
58
59                  session.permanent = False
60                  session['LOGGED_IN_AS'] = "ADMIN"
61                  login_user(admin, remember=True)
62                  return redirect(url_for('admin.tickets'))
63
64              else:
65                  to_show = True
66                  password = ""
67                  msg = "Invalid password!"
68
69          # Customer or Agent
70          else:
71              if to_show:
72                  # there is something fishy with the user's inputs
73                  password = ""
74
75              elif (not to_show):
76                  # the user's inputs are valid
77                  # checking if the login credentials are valid
78                  if role == "Customer":
79                      # checking if the entry of the mail entered is
    present in the database
80                      mail_check_query = "SELECT * FROM customer WHERE
```

```
    email = ?"
                    stmt = ibm_db.prepare(conn, mail_check_query)
                    ibm_db.bind_param(stmt, 1, email)
                    ibm_db.execute(stmt)

                    account = ibm_db.fetch_assoc(stmt)

                    if account:
                        # valid customer
                        # i.e, mail is present in the database

                        # checking if the customer entered a valid
    password now
                        # encrypting the entered password
                        passcode =
    str(hashlib.sha256(password.encode()).hexdigest())

                        # now checking if the encrypted string is same
    as that of the one in database
                        if (account['PASSCODE'] == passcode):
                            msg = "Valid Login"
                            to_show = True

                            # creating a customer object
                            customer.set(
                                account['CUST_ID'],
                                account['FIRST_NAME'],
                                account['LAST_NAME'],
                                account['EMAIL'],
                                account['PASSCODE'],
                                account['DATE_JOINED']
                            )

                            session.permanent = False
                            session['LOGGED_IN_AS'] = "CUSTOMER"
                            login_user(customer, remember=True)

                            return redirect(url_for('customer.profile'))

                        else:
                            # customer entered invalid password
                            msg = "Invalid password"
                            password = ""
                            to_show = True
```

```
121
122                     else:
123                         # invalid customer
124                         # i.e, entered mail is not present in the
   database
125                         msg = "User does not exist"
126                         email = ""
127                         password = ""
128                         to_show = True
129
130                 else:
131                     # user is an Agent
132                     # checking if the entry of the mail entered is
   present in the agent's table
133                     mail_check_query = "SELECT * FROM agent WHERE email
   = ?"
134                     stmt = ibm_db.prepare(conn, mail_check_query)
135                     ibm_db.bind_param(stmt, 1, email)
136                     ibm_db.execute(stmt)
137
138                     account = ibm_db.fetch_assoc(stmt)
139
140                     if account:
141                         # the mail entered by the agent is in the
   database
142
143                         # checking if the customer entered a valid
   password now
144                         # encrypting the entered password
145                         passcode =
   str(hashlib.sha256(password.encode()).hexdigest())
146
147                         # now checking if this passcode is equal to that
   of the password in database
148                         if(account['PASSCODE'] == passcode):
149                             # valid password
150                             msg = "Valid Login"
151                             to_show = True
152
153                             # initialising the agent object
154                             agent.set(
155                                 account['AGENT_ID'],
156                                 account['FIRST_NAME'],
157                                 account['LAST_NAME'],
```

```
158                                 account['EMAIL'],
159                                 account['PASSCODE'],
160                                 account['DATE_JOINED'],
161                                 account['CONFIRMED']
162                             )
163 
164                             session.permanent = False
165                             session['LOGGED_IN_AS'] = "AGENT"
166                             login_user(agent, remember=True)
167 
168                             if agent.confirm:
169                                 # the agent is confirmed by the admin
170                                 # so, re-directing the agent to his/her
    profile page
171                                 return
    redirect(url_for('agent.profile'))
172 
173                             else:
174                                 # the agent is not yet verified by the
    admin
175                                 # re-directing the agent to the agent no
    show page
176                                 return
    redirect(url_for('agent.no_show'))
177 
178                         else:
179                             # invalid password
180                             msg = "Invalid password"
181                             password = ""
182                             to_show = True
183 
184                     else:
185                         # invalid agent
186                         # i.e, entered mail is not present in the
    database
187                         msg = "Agent does not exist"
188                         email = ""
189                         password = ""
190                         to_show = True
191 
192         return render_template(
193             'login.html',
194             to_show = to_show,
195             message = msg,
```

```
            email = email,
            password = password
        )

    return render_template('login.html')

@views.route('/register', methods = ['GET', 'POST'])
def register():
    # if method is POST
    if request.method == 'POST':
        # getting all the data entered by the user
        first_name = request.form.get('first_name')
        last_name = request.form.get('last_name')
        email = request.form.get('email')
        password = request.form.get('password')
        confirm_password = request.form.get('confirm_password')
        role = request.form.get('role-check')

        msg = ""
        to_show = False

        # validating the inputs
        if len(first_name) < 3:
            msg = "First Name must be atleast 3 characters long!"
            to_show = True

        elif len(last_name) < 1:
            msg = "Last Name must be atleast 1 characters long!"
            to_show = True

        elif(not (re.fullmatch(email_regex, email))):
            msg = "Please enter valid email"
            to_show = True

        elif((len(password) < 8) or (len(confirm_password) < 8)):
            msg = "Password must be atleast 8 characters long!"
            to_show = True

        elif (password != confirm_password):
            msg = "Passwords do not match"
            to_show = True

        elif (not (re.fullmatch(pass_regex, password))):
            msg = "Enter valid password"
```

```
240             to_show = True
241 
242         if to_show:
243             # there is something fishy with the inputs
244             password = confirm_password = ""
245 
246         # by here the inputs are validated, because to_show is False
247         # registering the user / agent with the database
248         elif (not to_show):
249             if role == "Customer":
250                 # the user is a Customer
251                 # checking whether the user with the same email already
    there
252                 check_mail_query = "SELECT * FROM customer WHERE email =
    ?"
253                 stmt = ibm_db.prepare(conn, check_mail_query)
254                 ibm_db.bind_param(stmt, 1, email)
255                 ibm_db.execute(stmt)
256 
257                 account = ibm_db.fetch_assoc(stmt)
258 
259                 if account:
260                     # user already exists
261                     msg = "Email already exists!"
262                     email = ""
263                     password = ""
264                     confirm_password = ""
265                     to_show = True
266 
267                 else:
268                     # new customer
269                     # adding the customer details to the detabase
270                     user_insert_query ='''INSERT INTO customer
271                             (cust_id, first_name, last_name, email,
    passcode, date_joined)
272                             VALUES (?, ?, ?, ?, ?, ?)'''
273 
274                     # creating a UUID for the customer
275                     user_uuid = str(uuid.uuid4())
276 
277                     # encrypting the customer's password using SHA-256
278                     passcode =
    str(hashlib.sha256(password.encode()).hexdigest())
279                     date_joined = date.today()
```

```
280
281                     try:
282                         stmt = ibm_db.prepare(conn, user_insert_query)
283                         ibm_db.bind_param(stmt, 1, user_uuid)
284                         ibm_db.bind_param(stmt, 2, first_name)
285                         ibm_db.bind_param(stmt, 3, last_name)
286                         ibm_db.bind_param(stmt, 4, email)
287                         ibm_db.bind_param(stmt, 5, passcode)
288                         ibm_db.bind_param(stmt, 6, date_joined)
289
290                         ibm_db.execute(stmt)
291
292                         # redirecting the customer to the login page
293                         msg = "Account created. Please Login!"
294                         to_show = True
295
296                         return render_template('login.html', message =
   msg, to_show = to_show)
297
298                     except:
299                         msg = "Something went wrong!"
300                         to_show = True
301             else:
302                 # the role is Agent
303                 # checking whether the user with the same email already
   there
304                 check_mail_query = "SELECT * FROM agent WHERE email = ?"
305                 stmt = ibm_db.prepare(conn, check_mail_query)
306                 ibm_db.bind_param(stmt, 1, email)
307                 ibm_db.execute(stmt)
308
309                 account = ibm_db.fetch_assoc(stmt)
310
311                 if account:
312                     # means an agent with the email exists already!
313                     msg = "Email already exists!"
314                     email = ""
315                     password = ""
316                     confirm_password = ""
317                     to_show = True
318
319                 else:
320                     # new Agent
321                     # adding the customer details to the database
```

```
322                     agent_input_query = '''
323                         INSERT INTO agent
324                         (agent_id, first_name, last_name, email,
   passcode, date_joined, confirmed)
325                         VALUES (?, ?, ?, ?, ?, ?, ?)
326                     '''
327 
328                     # creating a unique id for the agent
329                     agent_id = str(uuid.uuid4())
330                     date_joined = date.today()
331                     confirmed = False
332 
333                     # encrypting the agent's password with SHA-256
334                     passcode =
   str(hashlib.sha256(password.encode()).hexdigest())
335 
336                     try:
337                         stmt = ibm_db.prepare(conn, agent_input_query)
338                         ibm_db.bind_param(stmt, 1, agent_id)
339                         ibm_db.bind_param(stmt, 2, first_name)
340                         ibm_db.bind_param(stmt, 3, last_name)
341                         ibm_db.bind_param(stmt, 4, email)
342                         ibm_db.bind_param(stmt, 5, passcode)
343                         ibm_db.bind_param(stmt, 6, date_joined)
344                         ibm_db.bind_param(stmt, 7, confirmed)
345 
346                         ibm_db.execute(stmt)
347 
348                         msg = "Account created! Please login"
349                         to_show = True
350 
351                         # re-directing the agent to the login page
352                         return render_template('login.html', message =
   msg, to_show = to_show)
353 
354                     except:
355                         msg = "Something went wrong!"
356                         to_show = True
357 
358         return render_template(
359             'register.html',
360             to_show = to_show,
361             message = msg,
362             first_name = first_name,
```

```
            last_name = last_name,
            email = email,
            password = password,
            confirm_password = confirm_password,
            role = role
        )
    return render_template('register.html')

@views.route('/forgot', methods = ['GET', 'POST'])
def forgot():
    '''
        Changing the password for the customer / agent
    '''
    msg = ""
    to_show = False

    if request.method == 'POST':
        # getting the email and role entered by the user (Customer or Agent)
        email = request.form.get('email')
        role = request.form.get('role-check')

        if len(email) == 0:
            msg = "Email cannot be empty!"
            to_show = True

        elif(not (re.fullmatch(email_regex, email))):
            msg = "Email valid email!"
            to_show = True

        else:
            if role == "Customer":
                # the user is a customer
                # checking if the email entered by the customer is in the database
                # query to check if the customer's mail exists in the customer table
                mail_check_query = '''
                    SELECT email FROM customer WHERE email = ?
                '''

                stmt = ibm_db.prepare(conn, mail_check_query)
                ibm_db.bind_param(stmt, 1, email)
                ibm_db.execute(stmt)
```

```
404                 account = ibm_db.fetch_assoc(stmt)
405
406                 if account:
407                     # then the email is in the database
408                     # the customer is a valid customer then
409                     msg = "Valid customer"
410                     to_show = True
411                     # generating a random 6-digit number to send to the
    customer
412                     randomNumber = random.randint(11111111, 99999999)
413
414                     # sending this number to the customer's email
415                     values = mail.sendEmail(
416                         "Forgot Password?",
417                         f'Your verification code is
    <strong>{randomNumber}</strong>',
418                         [f'{email}']
419                     )
420
421                     # encrypting the random number sent to the customer
    using SHA
422                     code =
    str(hashlib.sha256(str(randomNumber).encode()).hexdigest())
423
424                     if (not len(values.keys())) == 0:
425                         # something happened fishy
426                         msg = "Please try again!"
427                         to_show = True
428
429                     else:
430                         # the mail with the random number is sent
    successfully
431                         # redirecting the customer to the code entering
    page
432                         return
    redirect(f'/forgot/{role}/{email}/{code}/')
433
434                 else:
435                     # the email is not in the database
436                     # just someone trying to do fishy
437                     msg = "Customer does not exist!"
438                     to_show = True
439
440             elif role == "Agent":
```

```
                # the user is an Agent
                # checking if the email entered by the agent is in the database
                # query to check if the agent's mail exists in the agent table
                mail_check_query = '''
                    SELECT email FROM agent WHERE email = ?
                '''

                stmt = ibm_db.prepare(conn, mail_check_query)
                ibm_db.bind_param(stmt, 1, email)
                ibm_db.execute(stmt)
                account = ibm_db.fetch_assoc(stmt)

                if account:
                    # then the email is in the database
                    # the agent is a valid agent then
                    # generating a random 6-digit number to send to the customer
                    randomNumber = random.randint(11111111, 99999999)

                    # sending this number to the customer's email
                    values = mail.sendEmail(
                        "Forgot Password?",
                        f'Your verification code is <strong>{randomNumber}</strong>',
                        [f'{email}']
                    )

                    # encrypting the random number sent to the customer using SHA
                    code = str(hashlib.sha256(str(randomNumber).encode()).hexdigest())

                    if (not len(values.keys())) == 0:
                        # something happened fishy
                        msg = "Please try again!"
                        to_show = True

                    else:
                        # the mail with the random number is sent successfully
                        # redirecting the customer to the code entering page
```

```
                        return
    redirect(f'/forgot/{role}/{email}/{code}/')

                else:
                    # the email is not in the database
                    # just someone trying to do fishy
                    msg = "Agent does not exist!"
                    to_show = True

    return render_template(
        'forgot.html',
        message = msg,
        to_show = to_show
    )

@views.route('/forgot/<role>/<email>/<code>/', methods = ['GET',
    'POST'])
def code(role, email, code):
    if request.method == 'POST':
        # getting the code entered by the customer
        myCode = str(request.form.get('code-input'))

        if len(myCode) == 0:
            msg = "Code cannot be empty!"
            to_show = True

        else:
 return render_template(
                 'change password.html',
                  role = role, email = email
 )
```

## GitHub & Project Demo Link

**https://github.com/IBM-EPBL/IBM-Project-1511-1658393002**

**https://drive.google.com/file/d/1c40LL5wwoXQZ-lmQ21iEzhfCOjBtj-TM/view?usp=drivesdk**