Project Report

Team ID	**PNT2022TMID3443**
Project Name	**Customer Care Registry**

1. INTRODUCTION

1.1 Project Overview:

The Customer Service Desk is a web based project. Customer Service also known as Client Service is the provision of service to customers Its significance varies by product, industry and domain. In many cases customer services is more important if the information relates to a service as opposed to a Customer. Customer Service may be provided by a Service Representatives Customer Service is normally an integral part of a company's customer value proposition. Developing a cloud application not only for solving customer complaints but also gives satisfaction to the customer to use the respective business product. This Application helps a customer to raise complaints for the issue they are facing in the products. The Customer needs to give the detailed description and the priority level of the issues that they are facing. After the complaint reviewed by the admin, then the agents assigned to the complaints raised by the customer. The respective customer of the complaints gets the email notification of the process. And additionally, they can able to see the status of the complaints.

1.2 Purpose:

An online comprehensive Customer Care Solution is to manage customer interaction and complaints with the Service Providers over phone or through and e-mail. The system should have capability to integrate with any Service Provider from any domain or industry like Banking, Telecom Insurance etc. It is also known as Client Service is the provision of service to customers Its significance varies by product industry and domain. In many cases customer services is more important if the information relates to a service as opposed to as Customer. Customer Service may be provided by a Service Representatives Customer Service is normally an integral part of a company's customer value proposition. This Application mainly developed to help the customer in processing their complaints and issues. It is a process of examining customer tickets, which should be carried out in a systematic and orderly manner. This practice is primarily aimed at minimizing consumer dissatisfaction with the purchased products, increasing service satisfaction, and ensuring quality. It allows companies to respond to customer inquiries, provides support, and improves the handling of tickets at the appointed time.

2. LITERATURE SURVEY:

2.1 Existing problem:

The existing system is a semi-automated at where the information is stored in the form of excel sheets in disk drives. The information sharing to the Volunteers, Group members, etc. is through mailing feature only. The information storage and maintenance is more critical in this system. Tracking the member's activities and progress of the work is a tedious job here. This system cannot provide the information sharing by 24x7 days. When the company pushes the wrong product or service to customer this can severely impact to company's profit, growth and brand reputation. The customer cannot track the status of the Queries that are posted by them. Some queries will be left Unanswered. To overcome this issues a good customer care should be provided to solve the customer's queries.

2.2 References:

PAPER 1:

TITLE: Automated Ticket Routing System

AUTHOR NAME: Muhammad Zikri bin Zulkifli

PUBLICATION YEAR: 2011 **DESCRIPTION:**

In the existing helpdesk system, the tickets were created and assigned to the end user manually. When the ticket is created, it is assigned to the agent manually before they attend that specific ticket. This manual process of ticket creation needs more manpower and takes more time. Instead of putting the effort and time into this task, the ticket creation and assigning can be done automatically when we create an Automated Ticket Routing system. The automated ticket creation and assignment process reduce the time and then the manpower can be used for other purposes. Then, by using the manual ticket creation and assignment process, the distribution of good skill sets, and workload balancing will be missed out. Finding a good skill set and assigning the tickets to the specific skilled agent automatically is considered a good job distribution. Here, the wrong agent represents the sense that the agent doesn't know well about that particular problem or issue. If the tickets are mistakenly routed, then the resources may get wasted and a lot of time will be spent unnecessarily. Using the location, skill sets, work schedule, and workload balancing, the tickets can be routed automatically to that particular agent perfectly. We can execute the above process perfectly by categorizing the tickets based on the issues.

PAPER 2:

TITLE: Knowledge-Based Helpdesk System

AUTHOR NAME: Mohamad Safuan Bin Sulaiman , Abdul Muin Abdul Rahman , Norzalina Bt. Nasirudin

PUBLICATION YEAR: 2012 **DESCRIPTION:**

A knowledge-Based helpdesk system is a web-based system that is used to provide technical support to an organization or to management. Then, it acts as a Service Provider to that particular organization. The main objective of this Knowledge based system is to provide technical support to the end users of a particular organization. Using this Knowledge-based Helpdesk system, an organization can improve their end user's performance and make their end users technically well educated. Once the Knowledgebased helpdesk system is designed, it is tested on the Information Technology (IT) center, Engineering Division (BKJ), etc. To have a better support solution for management, the Knowledge-based system is introduced. Usually, the Knowledge-based system consists of questions that are frequently raised by the end users. All the frequent questions are combined into categories and then, it is provided as a solution. The end users can solve their problems manually by themselves just by reading and implementing the solution that is provided. Also, the solutions that are provided by the helpdesk team can be used on future problems too. Hence, it is called a continuity and contingency process.

PAPER 3:

TITLE: Smart Help Desk Automated Ticketing System

AUTHOR NAME: Dhiraj Temkar, Sheetal Singh, Leema Bari, Prof.Snigdha Banga

PUBLICATION YEAR: 2021

DESCRIPTION:

Automated technical queries help desk is proposed to possess instant real-time quick solutions and ticket categorization. Incorrect routing of tickets to the incorrect resolver group causes delays in resolving the matter. It also causes unnecessary resource utilization, and customer dissatisfaction and affects the business. To beat these problems, the proposed "Smart Automated Ticketing System" supports supervised machine learning techniques that automatically predict the category of the ticket using the natural language ticket description entered by the user through a chat interface. It also helps in faster resolution of customer issues and sends them an email about the status of the ticket. This process assures customer satisfaction and also keeps the customers within the loop.

PAPER 4:

TITLE: Theory and Practice of Customer-related Improvements

AUTHOR NAME: Daniel Gyllenhammar, et al

PUBLICATION YEAR: 2022 **DESCRIPTION:**

In an organization, the Information Technology (IT) support help desk operation is an important unit that handles the IT services of a business. Many large-scale organizations handle engagement and requests with employees on a 24×7 basis. As with any routine tasks, most processes of the support help desk unit are considered repetitive in nature repetitive tasks such as entering information into an application, resetting passwords, unlocking applications, and credentials errors. The industry has now come to realize that many repetitive business processes and tasks can be automated by using Robotic Process Automation (RPA) bots or robotic processes automotive software bots. The idea is to take the repetitive workload and hand it over to the RPA bots so that the employees could focus on more value-adding tasks and decision-making for the organization. The RPA bot would also help to reduce human errors and make processes more efficient, which would finally result in cost savings and productivity increase.

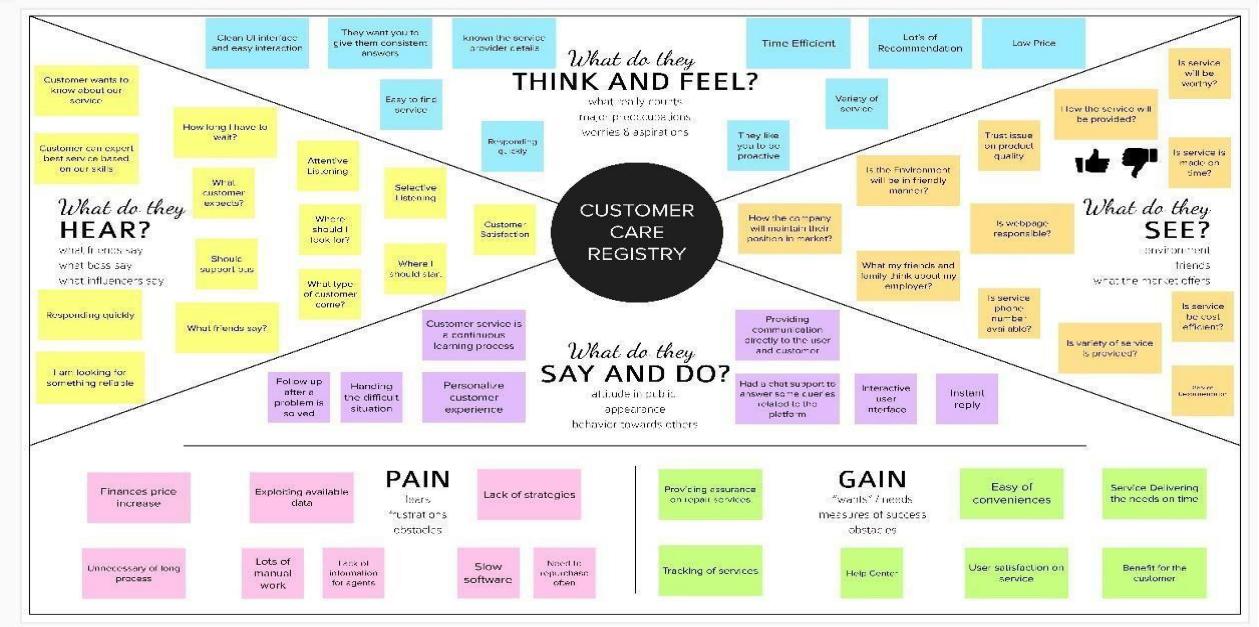
2.3 Problem Statement Definition:

Problem Statement (PS)	I am (Customer)	I'm trying to	But	Because	Which makes me feel
PS-1	User	Ticket Booking	Time Delay	Agent Not Responding	Sad

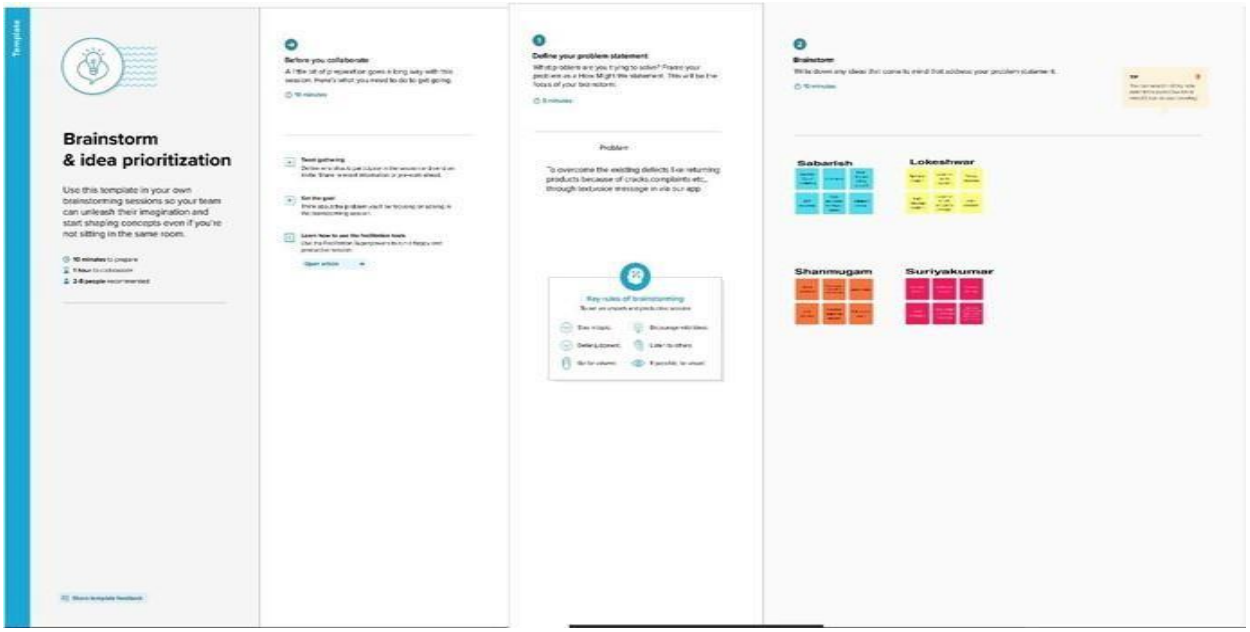
PS-2	User(Agent)	Solve Problem	Customer Not Responding	Customer Unavailable	Frustrated
PS-3	User(Admin)	Backup Data	Data Loss	System Failure	Anxiety
PS-4	User	Looking for Status	Status Unavailable	Agent Not Updated	Stressed

3.IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



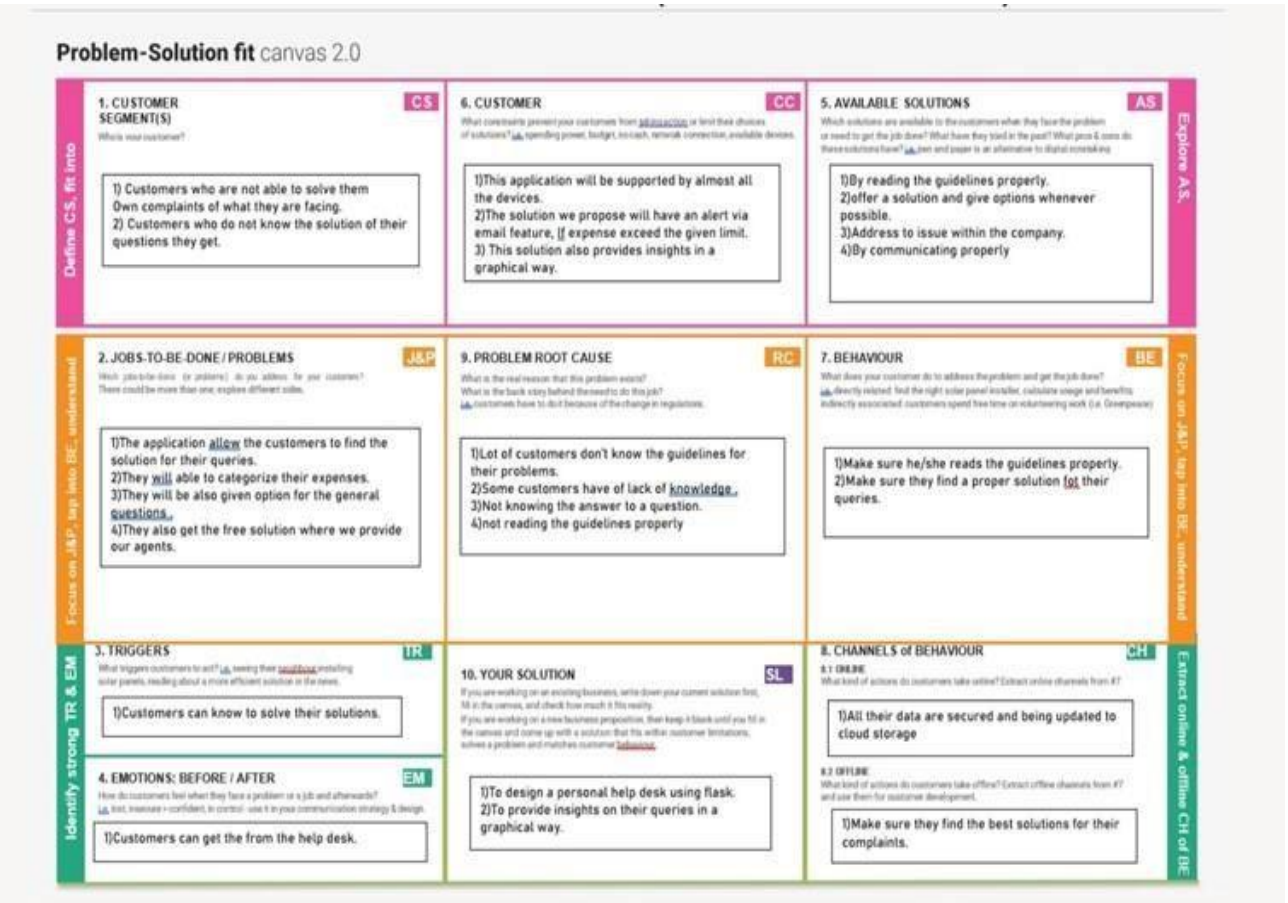
3.2 Ideation & Brainstorming



3.3 Proposed Solution

S.No.	Parameter	Description
1.	Problem Statement (Problem to be solved)	To solve customer issues using Cloud Application Development.
2.	Idea / Solution description	Assigned Agent routing can be solved by directly routing to the specific agent about the issue using the specific Email. Automated Ticket closure by using daily sync of the daily database. Status Shown to the Customer can display the status of the ticket to the customer. Regular data retrieval in the form of retrieving lost data.
3.	Novelty / Uniqueness	Assigned Agent Routing, Automated Ticket Closure, Status Shown to the Customer, and Backup data in case of failures.
4.	Social Impact / Customer Satisfaction	Customer Satisfaction, Customer can track their status and Easy agent communication.
5.	Business Model (Revenue Model)	• Key Partners - Third-party applications, agents, and customers. • Activities - Customer Service, System Maintenance. • Key Resources - Engineers, Multi-channel. • Customer Relationship - 24/7 Email Support, Knowledge-based channel. • Cost Structure - Cloud Platform, Offices.
6.	Scalability of the Solution	All customers are prioritized based on SLA(Service Level Agreement)Urgent,Moderate, Low.

3.4 Problem Solution fit



4.REQUIREMENT ANALYSIS

4.1 Functional requirement

Following are the functional requirements of the proposed solution.

FR No.	Functional Requirement (Epic)	Sub Requirement (Story / Sub-Task)
FR-1	Admin/Agent Registration	Registration through Gmail.
FR-2	Admin/Agent Confirmation	Confirmation via Email.
FR-3	Customer Query	Access through Email, Chatbot from targeted websites.
FR-4	Customer Confirmation	Confirmation through Ticket ID in Email.
FR-5	Database	Storing the object model.

4.2 Non-Functional requirements

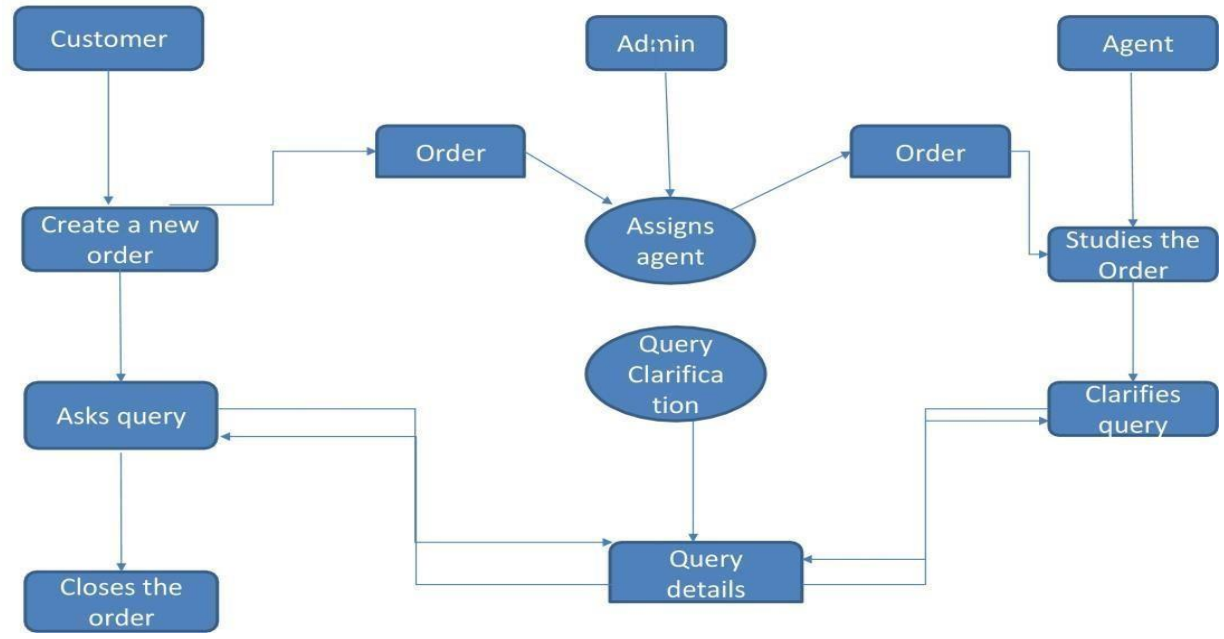
Following are the non-functional requirements of the proposed solution.

FR No.	Non-Functional Requirement	Description
NFR-1	**Usability**	User Friendly, Easily Accessible.
NFR-2	**Security**	IBM Digital Security Certificate(SSL) for Database.
NFR-3	**Reliability**	Providing Quality Content.
NFR-4	**Performance**	Quick Access, Flexible, and Responsive
NFR-5	**Availability**	24/7 Support
NFR-6	**Scalability**	Good performance for large Customers and workload

5.PROJECT DESIGN

5.1Data Flow Diagrams

Data flow diagram for Customer care Registry



5.2 Solution & Technical Architecture:

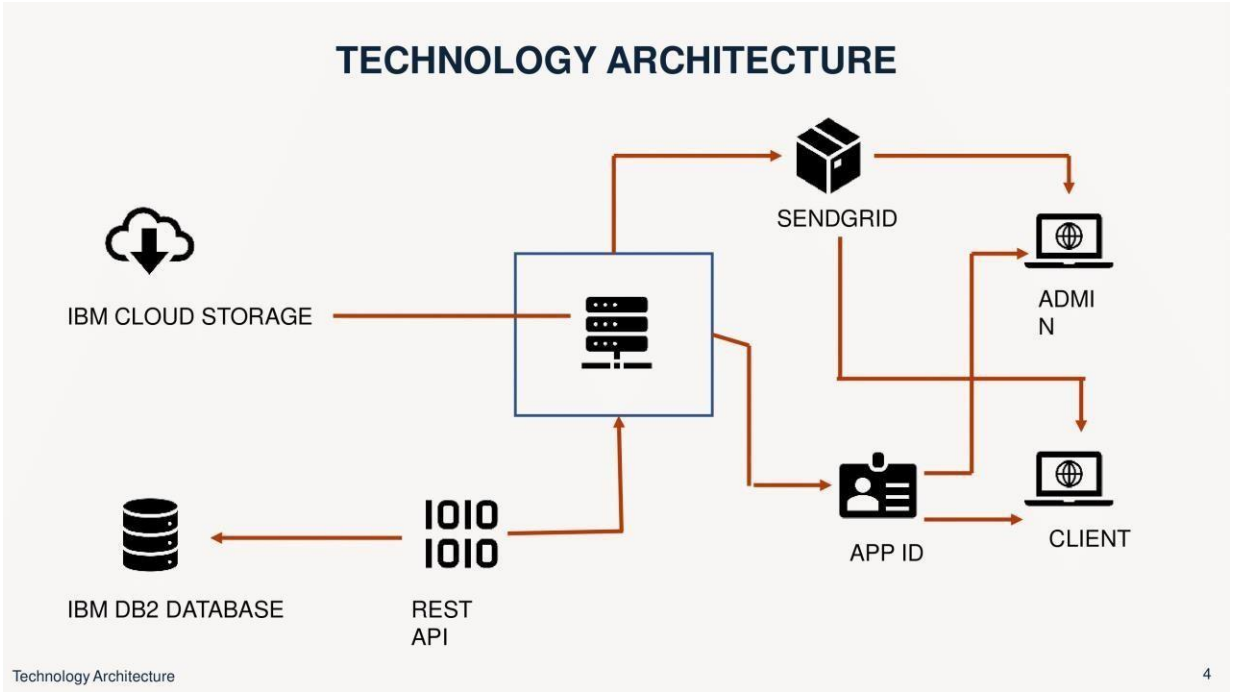


Table-1: Components & Technologies:

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
7.	Infrastructure (Server / Cloud)	Application Deployment on Local System / Cloud Local Server Configuration:Local Server Cloud Server Configuration : Online Server	Docker and Kubernetes

Table-2: Application Characteristics:

S.No	Characteristics	Description	Technology
1.	Open-Source Frameworks	Flask-Python Framework	Flask -Python
2.	Security Implementations	Digital Certificate SSL Secuirty	IBM Cloud and IBM DB2
3.	Scalable Architecture	Sendgrid API and Json Server	IBM Object Storage
4.	Availability	Large Number of Customer Utilize	IBM Kubernates
5.	Performance	Fast Recovering Data From IBM DB2 and flexible request and response from Cloud	IBM Cloud

5.3 User Stories

Use the below template to list all the user stories for the product.

User Stories

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Customer (Mobile user)	Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my 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 orders raised by me.	I get all the info needed in my dashboard.	Low	Sprint-2
	Order creation	USN-4	As a customer, I can place my order with the detailed description of my query	I can ask my query	Medium	Sprint-2
	Address Column	USN-5	As a customer, I can have conversations with the assigned agent and get my queries clarified	My queries are clarified.	High	Sprint-3
	Forgot password	USN-6	As a customer, I can reset my password by this option incase I forgot my old password.	I get access to my account again	Medium	Sprint-4
	Order details	USN-7	As a Customer ,I can see the current stats of order.	I get abetter understanding	Medium	Sprint-4
Agent (web user)	Login	USN-1	As an agent I can login to the application by entering Correct email and password.	I can access my account / dashboard.	High	Sprint-3
	Dashboard	USN-2	As an agent, I can see the order details assigned to me by admin.	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/er dobuts	I can clarify the issues.	High	Sprint-3
	Forgot password	USN-4	As an agent I can reset my password by this option in case I forgot my old password.	I get access to my account again.	Medium	Sprint-4
Admin (Mobile user)	Login	USN-1	As a admin, I can login to the appliaction 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 orders raised in the entire system and lot more	I can assign agents by seeing those order.	High	Sprint-1
	Agent creation	USN-3	As an admin I can create an agent for clarifying the customers queries	I can create agents.	High	Sprint-2
	Assignment agent	USN-4	As an admin I can assign an agent for each order created by the customer.	Enable agent to clarify the queries.	High	Sprint-1
	Forgot password	USN-5	As an admin I can reset my password by this option in case I forgot my old password.	I get access to my account.	High	Sprint-1

6. PROJECT PLANNING & SCHEDULING

6.1 Sprint Planning & Estimation

TITLE	DESCRIPTION	DATE
Literature Survey & Information Gathering	Literature survey on the selected project & gathering information by referring to technical papers,research publications etc.	09 SEPTEMBER 2022
Prepare Empathy Map	Prepare Empathy Map Canvas to capture the user Pains & Gains, Prepare list of problem statements	10 SEPTEMBER 2022
Ideation	List them by organizing the brainstorming session and prioritize the top 3 ideas based on feasibility & importance.	12 SEPTEMBER 2022
Proposed Solution	Prepare the proposed solution document, which includes the novelty, feasibility of idea, business model, social impact, scalability of solution, etc.	03 OCTOBER 2022
Problem Solution Fit	Prepare problem - solution fit document.	05 OCTOBER 2022
Solution Architecture	Prepare a solution architecture document.	07 OCTOBER 2022

Customer Journey	Prepare the customer journey maps to understand the user interactions & experiences with the application (entry to exit).	15 OCTOBER 2022
Functional Requirement	Prepare the functional requirement document.	15 OCTOBER 2022
Data Flow Diagrams	Draw the data flow diagrams and submit for review.	19 OCTOBER 2022
Technology Architecture	Prepare the technology architecture diagram.	14 OCTOBER 2022
Prepare Milestone & Activity List	Prepare the milestones & activity list of the project.	24 OCTOBER 2022
Project Development - Delivery of Sprint-1, 2, 3 & 4	Develop & submit the developed code by testing it.	20 NOVEMBER 2022 (PLANNED)

Product Backlog, Sprint Schedule, and 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	Jenish J,jebix shylin E

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	Aswin mon. M,Rajesh kumar.J
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	Jenish.J, Aswin Mon.M
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	Aswin Mon.M,Jenish.J, Jebix shylin
Sprint-4	Final Delivery	USN-5	Container of applications using docker kubernetes and deployment the application. Create the documentation and final submit the application	2	High	Aswin Mon.M,Jenish.J, Jebix shylin

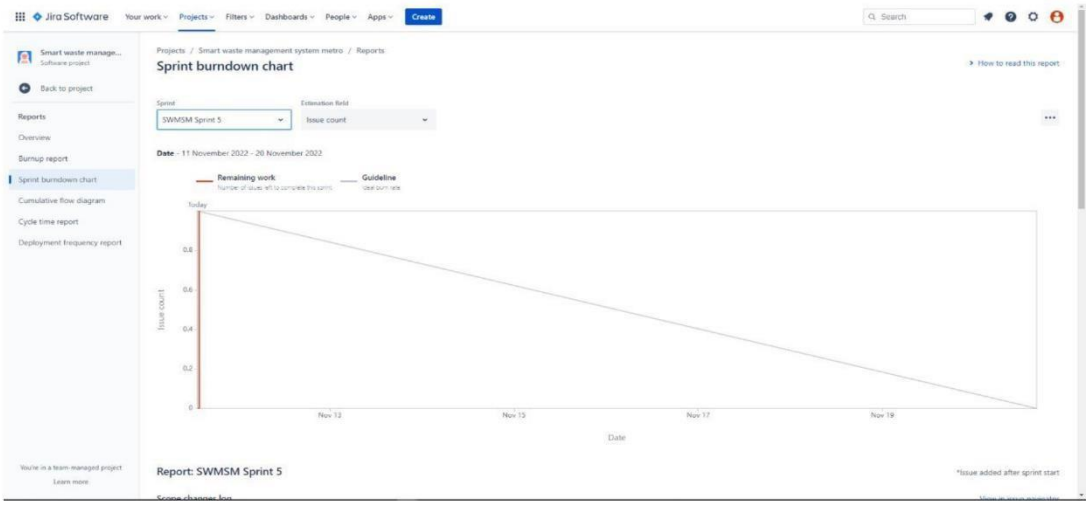
6.2. Sprint Delivery Schedule

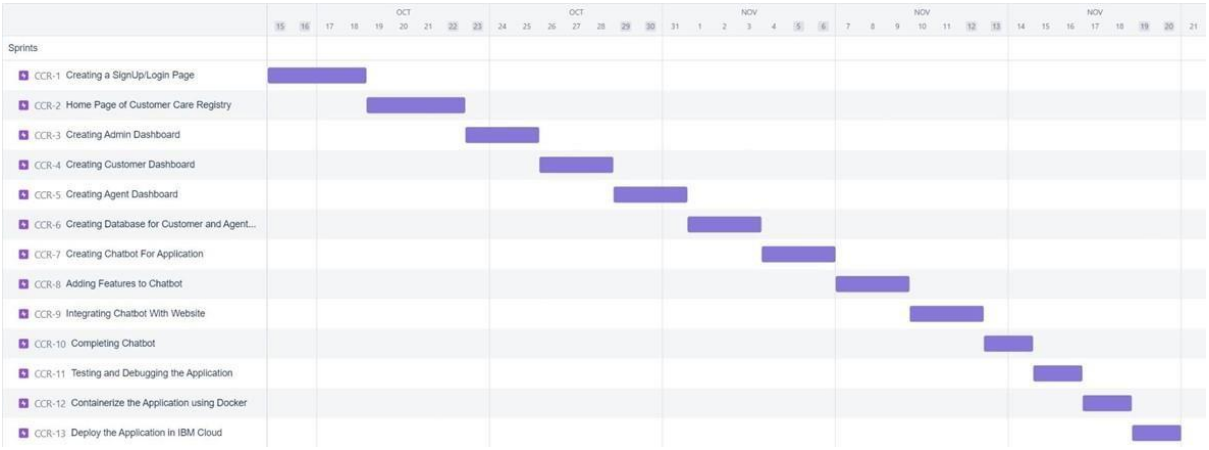
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	7 Days	24 Oct 2022	30 Oct 2022		30 Oct 2022
Sprint-2	20	7 Days	31 Oct 2022	06 Nov 2022		06 Nov 2022
Sprint-3	20	8 Days	07 Nov 2022	14 Nov 2022		14 Nov 2022

Sprint-4	20	7 Days	14 Nov 2022	21 Nov 2022		21 Nov 2022

6.3 Reports from JIRA

BURNDOWN CHART





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

College graduates with prior programming expertise or technical degrees are recruited and transitioned into professional positions with Alabama firms and organisations through the highly competitive Coding Solutions job accelerator and talent refinement programme at no cost to the graduates. We provide a pool of varied, well-trained, techs-savvy individuals that wants to launch and advance their career in Alabama.

The mission of veteran- and woman-owned Coding Solutions is to mobilise the next generation of IT talent and provide them the tools and resources they require to make your business successful. Innovative talent is necessary for innovative technologies. We wish to provide Coding Solutions prospects to assist you expand your Alabama team.

Our applicants are swiftly hired at the top of the list by growing businesses for lucrative, long-term positions.

7.1Feature 1

7 Main types of customer needs:

- Friendlines
- Empathy
- Fairness
- Control
- Alternatives
- Information

7.2 Feature

- Complaint Tracking
- Email Alert
- 24/7 Montoring

8.TESTING

8.1 User Acceptance Testing

Purpose of Document

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

Defect Analysis

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

Resolution	Severity 1	Severity 2	Severity 3	Severity 4	Subtotal
By Design	10	3	1	2	17
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	40
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	13	12	25	78

Test Case Analysis

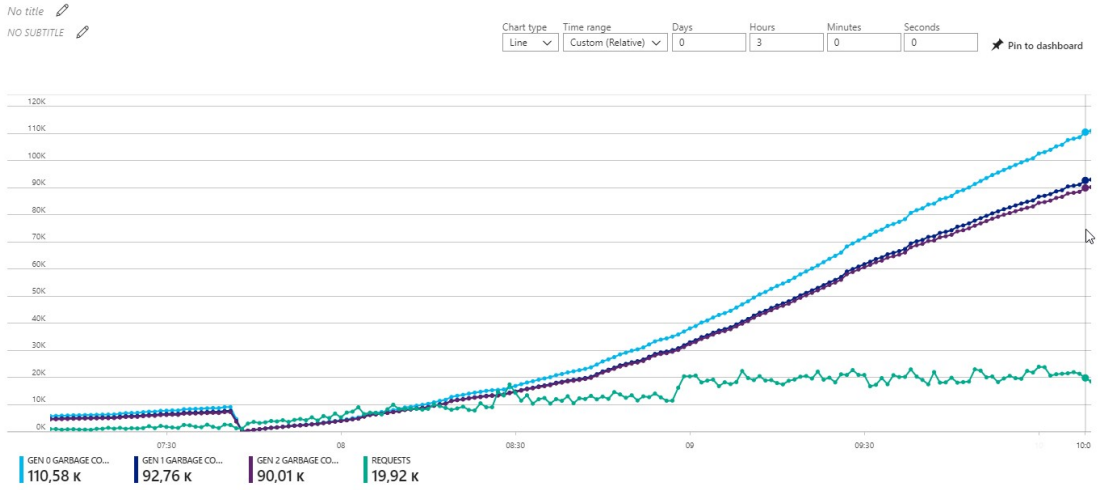
This report shows the number of test cases that have passed, failed, and untested

Section	Total Cases	Not Tested	Fail	Pass
Print Engine	10	0	0	10
Client Application	50	0	0	50
Security	1	0	0	1

Outsource Shipping	3	0	0	3
Exception Reporting	8	0	0	8
Final Report Output	4	0	0	4
Version Control	2	0	0	2

9.RESULTS

9.1 Performance Metrics



10. ADVANTAGES & DISADVANTAGES

ADVANTAGES:

- It retains the customer
- Gets you more references
- Increases profitability
- Gives you and your employees confidence
- Creates a holistic marketing scenario
- Competitive advantage
- Boost Customer Loyalty
- Enhance Brand Reputation
- Improve Products, Services, Procedures and Staff

DISADVANTAGES:

- Higher staff wages from hiring employees who are experts in customer service.
- Paying for staff training
- The extra services offered, such as refreshments
- Higher wage costs from the extra time staff take to provide post-sales service.
- It can be particularly difficult for small businesses to cope with these costs

11.CONCLUSION

In conclusion, customer care, involves the use of basic ethics and any company who wants to have success and grow, needs to remember, that in order to do so, it must begin with establishing a code of ethics 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 care involves, the treatment, care, loyalty, trust the employee should extend to the consumer, as well in life.

11.FUTURE SCOPE

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

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

12) APPENDIX

Source Code

```
# Project : Customer Care Registry #
Team ID : PNT2022TMID3443
```

index.py

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify from
flask_mysqldb import MySQL import MySQLdb.cursors import ibm_db import re, random,
smtplib, os, time, datetime from flask_mail import Mail, Message


app = Flask(__name__)

app.secret_key = '12345'

conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=21fecfd8-47b7-4937-
840dd791d0218660.bs2io90l08kqb1od8lcg.databases.appdomain.cloud;PORT=31864;SECURITY=SSL;
SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=rtp84701;PWD=DJ4gX1wChdTCGZPz",'','')

mail= Mail(app)

app.config['MAIL_SERVER']='smtp.gmail.com' app.config['MAIL_PORT'] =
465 app.config['MAIL_USERNAME']='customercareregistry22@gmail.com'
app.config['MAIL_PASSWORD'] = 'vxzttcjvdvrqeeve'
app.config['MAIL_USE_TLS'] = False app.config['MAIL_USE_SSL'] = True mail
= Mail(app)



@app.route('/', methods =['GET', 'POST']) def index():    if
request.method == 'POST' and 'email' in request.form:
    email = request.form['email']      cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)      cursor.execute('SELECT *
FROM subscriptions WHERE email = % s', (email, ))      subscriptions = cursor.fetchone()
if subscriptions:
       flash('This Email Is Already Subscribed')      else:
        ts = time.time()
       timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')        cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)          cursor.execute('INSERT INTO subscriptions
VALUES (%s, % s, % s)', (None, email, timestamp, ))        mysql.connection.commit()        flash('You have
successfully Subscribed')    return render_template('index.html')
```

```
@app.route('/customerlogin', methods =['GET', 'POST']) def
customerlogin():
  msgdecline = ''    if request.method == 'POST' and 'cemail' in request.form and 'cpassword' in
request.form:
    cemail = request.form['cemail']          cpassword = request.form['cpassword']          cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)          cursor.execute('SELECT * FROM
customers_details WHERE customer_email = % s AND customer_password = % s', (cemail, cpassword, ))
customers_details = cursor.fetchone()      if customers_details:
      session['loggedin'] = True        session['cemail'] =
customers_details['customer_email']        msgsuccess = 'Logged
in successfully !'        return redirect(url_for('welcome'))      else:
      msgdecline = 'Incorrect Email / Password !'    return
render_template('customerlogin.html', msgdecline = msgdecline)
```

```
@app.route('/agentlogin', methods =['GET', 'POST']) def agentlogin():    msgdecline = ''    if
request.method == 'POST' and 'aemail' in request.form and 'apassword' in request.form:
    aemail = request.form['aemail']      apassword = request.form['apassword']      cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)      cursor.execute('SELECT * FROM
agent_information WHERE agent_email = % s AND agent_password = %s', (aemail, apassword,))
agent_information = cursor.fetchone()      if agent_information:
      session['loggedin'] = True
      session['aemail']    =    agent_information['agent_email']
msgsuccess    =    'Logged    in    successfully    !'    return
redirect(url_for('agentdashboard'))
    else:
      msgdecline = 'Incorrect Email / Password !'    return
render_template('agentlogin.html', msgdecline = msgdecline)
```

```
@app.route('/adminlogin', methods =['GET', 'POST']) def adminlogin():    msgdecline = ''    if request.method
== 'POST' and 'adminusername' in request.form and 'adminpassword' in request.form:
    adminusername = request.form['adminusername']      adminpassword = request.form['adminpassword']
cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)          cursor.execute('SELECT * FROM
admin_details WHERE admin_username = % s AND admin_password = % s', (adminusername,
adminpassword, ))      admin = cursor.fetchone()      if admin:
      session['loggedin'] = True        session['adminusername'] =
admin['admin_username']        msgsuccess = 'Logged in
successfully !'        return redirect(url_for('admindashboard'))
else:
```

```
        msgdecline = 'Incorrect Username / Password !'    return
render_template('adminlogin.html', msgdecline = msgdecline)



@app.route('/customerregister', methods =['GET', 'POST']) def customerregister():     msgdecline = ''     if
request.method == 'POST' and 'cname' in request.form and 'cemail' in request.form and 'cpassword' in
request.form and 'cconfirmpassword' in request.form :
    cname = request.form['cname']      cemail = request.form['cemail']      cpassword =
request.form['cpassword']      cconfirmpassword = request.form['cconfirmpassword']      cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)      cursor.execute('SELECT * FROM
customers_details WHERE customer_email = % s', (cemail, ))      user_registration = cursor.fetchone()
if user_registration:
        msgdecline = 'Account already exists ! Try Login'      elif
cpassword != cconfirmpassword:
        msgdecline = 'Password did not match !'      else:
         ts = time.time()
        timestamp        =        datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d        %H:%M:%S')
cursor.execute('INSERT INTO customers_details VALUES (%s, % s, % s, % s, % s)', (None, cname, cemail,
cpassword, timestamp, ))        mysql.connection.commit()        flash('You have successfully registered ! Try
Login')        try:
          mailmsg = Message('Customer Care Registry', sender = 'Registration Successful', recipients
= ['{}', cemail])
          mailmsg.body = "Hello {},\nYou have successfully registered on Customer Care
Registry".format(cname)          mail.send(mailmsg)        except:          pass        return
redirect(url_for('customerlogin'))    elif request.method == 'POST':
    msgdecline = 'Please fill out the form !'    return
render_template('customerregister.html', msgdecline = msgdecline)



@app.route('/agentregister', methods =['GET', 'POST'])
def agentregister():    if not
session.get("adminusername"):
    return redirect("/adminlogin")    else:      msgdecline = ''      if request.method == 'POST' and 'aname' in
request.form and 'aemail' in request.form and 'ausername' in request.form and 'apassword' in request.form
and 'aconfirmpassword' in request.form :
        aname = request.form['aname']        aemail = request.form['aemail']        ausername =
request.form['ausername']        apassword = request.form['apassword']        aconfirmpassword =
request.form['aconfirmpassword']        cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)        cursor.execute('SELECT * FROM
agent_information WHERE agent_email = % s', (aemail, ))        agent_information = cursor.fetchone()
if agent_information:
          msgdecline = 'Account already exists ! Try Login'
else:          ts = time.time()
```

```
        timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
cursor.execute('INSERT INTO agent_information VALUES (%s, % s, % s, % s, % s, % s)', (None, aname,
aemail, ausername, apassword, timestamp,))          mysql.connection.commit()          flash('Agent Has
been successfully registered !')
        try:
          mailmsg = Message('Customer Care Registry', sender = 'Registration Successful', recipients = ['{}',
aemail])                    mailmsg.body = "Hello, You have been Successfully Registered as Agent"
mail.send(mailmsg)          except:          pass          return redirect(url_for('agentlogin'))       elif
request.method == 'POST':                  msg = 'Please fill out the form !'          return
render_template('agentregister.html', msgdecline = msgdecline)



@app.route('/welcome', methods =['GET', 'POST']) def
welcome():
  if not session.get("cemail"):
    return redirect("/customerlogin")    else:
    msgsuccess = ''          msgdecline = ''          cmail = session['cemail']          mycursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)          mycursor.execute('SELECT * FROM
complaint_details WHERE customer_email = %s ORDER BY timestamp DESC', (cmail,))          data =
mycursor.fetchall()          mycursor.execute('SELECT customer_name FROM customers_details WHERE
customer_email
= %s', (cmail,))      cname = mycursor.fetchone()      if request.method == 'POST' and 'name' in request.form
and 'email' in request.form and 'category' in request.form and 'subject' in request.form and 'description' in
request.form :
       name = request.form['name']        email =
request.form['email']        category =
request.form['category']        subject =
request.form['subject']        description =
request.form['description']        ticketno =
random.randint(100000, 999999)        ts =
time.time()
      timestamp = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')        cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)        cursor.execute('INSERT INTO
complaint_details VALUES (%s, % s, % s, % s, % s, % s, % s, % s, % s )', (ticketno, name, email, category,
subject, description, timestamp , "pending" , "pending" , ))        mysql.connection.commit()        try:
        mailmsg = Message('Customer Care Registry', sender = 'Request Received', recipients = ['{}', email])
        mailmsg.body = "Hello {},\n\nThanks for contacting Customer Care Registry\nWe have received
your complain\nYour Ticket Number: {}\nCategory: {}\nSubject: {}\nDescription: {}\n\nWe strive to provide
excellent service, and will respond to your request as soon as possible.".format(name, ticketno, category,
subject, description)          mail.send(mailmsg)        except:          pass
```

```
        flash ('Your complaint is successfully submitted !')
return redirect(url_for('welcome'))        elif request.method
== 'POST':
        msgdecline = 'Please fill out the form !'    return render_template('welcome.html', msgsuccess =
msgsuccess, data=data, cname=cname)



@app.route('/agentdashboard', methods =['GET', 'POST']) def
agentdashboard():    if not session.get("aemail"):      return
redirect("/agentlogin")    else:
    msg = ''                aemail = session['aemail']                mycursor1 =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)       mycursor1.execute('SELECT agent_name FROM
agent_information WHERE agent_email = %s',
(aemail, ))       agent =
mycursor1.fetchone()

    for x in agent:
      agent_name = agent[x]

    mycursor2 = mysql.connection.cursor(MySQLdb.cursors.DictCursor)       mycursor2.execute('SELECT *
FROM complaint_details WHERE agent_name = %s ORDER BY timestamp DESC', (agent_name, ))       data =
mycursor2.fetchall()       if request.method == 'POST' and 'status' in request.form :
        status = request.form['status']              ticketno = request.form['ticketno']              cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)       cursor.execute('UPDATE complaint_details SET
status = %s WHERE ticket_no = %s', (status, ticketno,) )       mysql.connection.commit()       msg = 'Your
complaint is successfully solved !'

        mailcursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)       mailcursor.execute('SELECT
customer_email FROM complaint_details WHERE ticket_no = %s', (ticketno,) )
        customer_mail = mailcursor.fetchone()

        for x in customer_mail:
          cemail = customer_mail[x]

try:
          mailmsg = Message('Customer Care Registry', sender = 'Your Ticket Status', recipients = ['{}', cemail])
          mailmsg.body = "Hello, \nYour complaint has been successfully solved\nYour Ticket Number:
{}".format(ticketno)              mail.send(mailmsg)              except:              pass              return
redirect(url_for('agentdashboard'))       elif request.method == 'POST':       msg = 'Please fill out the form !'
return render_template('agentdashboard.html', msg = msg, data=data, agent_name=agent_name)



@app.route('/admindashboard', methods =['GET', 'POST']) def
admindashboard():    if not session.get("adminusername"):
```

```
    return redirect("/adminlogin")    else:
    msg = ''
    mycursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)      mycursor.execute('SELECT *
FROM complaint_details ORDER BY timestamp DESC')      data = mycursor.fetchall()
mycursor.execute('SELECT * FROM agent_information')      agent = mycursor.fetchall()
mycursor.execute('SELECT COUNT(status) AS pending FROM complaint_details WHERE status
= %s', ("pending",))      pending = mycursor.fetchall()      mycursor.execute('SELECT COUNT(status) AS
assigned FROM complaint_details WHERE status = %s', ("Agent Assigned",))     assigned = mycursor.fetchall()
mycursor.execute('SELECT COUNT(status) AS completed FROM complaint_details WHERE status = %s',
("Closed",))      completed = mycursor.fetchall()      if request.method == 'POST' and 'agentassign' in
request.form :
      agentassign = request.form['agentassign']      adminusername =
request.form['adminusername']      ticketno = request.form['ticketno']
cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)
cursor.execute('UPDATE complaint_details SET agent_name = %s WHERE
ticket_no = %s',
(agentassign, ticketno,) )
      cursor.execute('UPDATE complaint_details SET status = %s WHERE ticket_no = %s', ("Agent Assigned",
ticketno,) )      mysql.connection.commit()      msg = 'Your complaint is Assigned to Agent !'

      mailcursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)      mailcursor.execute('SELECT
customer_email FROM complaint_details WHERE ticket_no = %s',
(ticketno,) )      customer_mail =
mailcursor.fetchone()

      for x in customer_mail:
        cemail = customer_mail[x]

try:
        mailmsg = Message('Customer Care Registry', sender = 'Agent Assigned', recipients = ['{}', cemail])
        mailmsg.body = "Hello,\nWe have received your complaint and agent {} has been Successfully
Assigned\nYour Ticket Number: {}\n\nYou will be notified when your complain will be
solved.".format(agentassign, ticketno)      mail.send(mailmsg)      except:      pass
      return redirect(url_for('admindashboard'))
elif request.method == 'POST':      msg = 'Please
fill out the form !'
  return      render_template('admindashboard.html',      msg      =      msg,      data=data,
       agent=agent, pending=pending, assigned=assigned, completed=completed)



@app.route('/admininanalytics') def
admininanalytics():
```

```
  mycursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor) mycursor.execute('SELECT COUNT(agent_name) AS JenTile FROM complaint_details WHERE agent_name = %s', ("Jen Tile",)) JenTile = mycursor.fetchall() mycursor.execute('SELECT COUNT(agent_name) AS AllieGrater FROM complaint_details WHERE agent_name = %s', ("Allie Grater",)) AllieGrater = mycursor.fetchall() mycursor.execute('SELECT COUNT(agent_name) AS RaySin FROM complaint_details WHERE agent_name = %s', ("Ray Sin",)) RaySin = mycursor.fetchall() mycursor.execute('SELECT COUNT(category) AS Category1 FROM complaint_details WHERE category = %s', ("Product Exchange or Return",))
  category1 = mycursor.fetchall() mycursor.execute('SELECT COUNT(category) AS Category2 FROM complaint_details WHERE category = %s', ("Product Out of Stock",)) category2 = mycursor.fetchall() mycursor.execute('SELECT COUNT(category) AS Category3 FROM complaint_details WHERE category = %s', ("Payments & Transactions",)) category3 = mycursor.fetchall() mycursor.execute('SELECT COUNT(category) AS Category4 FROM complaint_details WHERE category = %s', ("Product Delivery",)) category4 = mycursor.fetchall() mycursor.execute('SELECT COUNT(category) AS Category5 FROM complaint_details WHERE category = %s', ("Other",)) category5 = mycursor.fetchall() print(category1)
  return render_template('adminanalytics.html', JenTile=JenTile, AllieGrater=AllieGrater, RaySin=RaySin, category1=category1, category2=category2, category3=category3, category4=category4, category5=category5)



@app.route('/customerforgotpassword', methods =['GET', 'POST']) def customerforgotpassword(): msgdecline = '' if request.method == 'POST' and 'customerforgotemail' in request.form :
      forgotemail = request.form['customerforgotemail'] cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor) cursor.execute('SELECT * FROM customers_details WHERE customer_email = % s', (forgotemail, )) customers_details = cursor.fetchone() if customers_details:
        session['customerforgotemail'] = forgotemail
otp = random.randint(1000, 9999) session['otp'] = otp try:
          mailmsg = Message('Customer Care Registry', sender = 'Forgot Password', recipients = ['{}', forgotemail]) mailmsg.body = "Hello, \nYour OTP is: {}\nDo not share this OTP to anyone \nUse this OTP to reset your password.".format(otp) mailmsg.subject = 'Forgot Passowrd' mail.send(mailmsg) flash('OTP has been sent to your email') return redirect(url_for('enterotp')) except:
          msgdecline = 'Oops! Something went wrong! Email not sent' else:
        msgdecline = 'This email is not registered!'
    return render_template('customerforgotpassword.html', msgdecline = msgdecline)



@app.route('/agentforgotpassword', methods =['GET', 'POST']) def agentforgotpassword(): msgdecline = '' if request.method == 'POST' and 'agentforgotemail' in request.form : forgotemail = request.form['agentforgotemail'] cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor) cursor.execute('SELECT * FROM agent_information WHERE agent_email = % s', (forgotemail,
```

```
))
        agent_information = cursor.fetchone()
if agent_information:
            session['agentforgotemail'] = forgotemail
otp = random.randint(1000, 9999)
session['otp'] = otp            try:
                mailmsg = Message('Customer Care Registry', sender = 'Forgot Password', recipients = ['{}',
forgotemail])                mailmsg.body = "Hello, \nYour OTP is: {}\nDo not share this OTP to anyone \nUse
this OTP to reset your password.".format(otp)                mail.send(mailmsg)                flash('OTP has been
sent to your email')                return redirect(url_for('enterotp'))            except:
                msgdecline = 'Oops! Something went wrong! Email not sent'          else:
            msgdecline = 'This email is not registered!'       return
render_template('agentforgotpassword.html', msgdecline = msgdecline)



@app.route('/adminforgotpassword', methods =['GET', 'POST']) def adminforgotpassword():
msgdecline = ''       if request.method == 'POST' and 'adminforgotemail' in request.form :
forgotemail = request.form['adminforgotemail']          cursor =
mysql.connection.cursor(MySQLdb.cursors.DictCursor)          cursor.execute('SELECT * FROM
admin_details WHERE admin_email = % s', (forgotemail, ))          admin_details = cursor.fetchone()
if admin_details:
            session['adminforgotemail'] = forgotemail
otp = random.randint(1000, 9999)
session['otp'] = otp            try:
                mailmsg = Message('Customer Care Registry', sender = 'Forgot Password', recipients = ['{}',
forgotemail])                mailmsg.body = "Hello, \nYour OTP is: {}\nDo not share this OTP to anyone \nUse
this OTP to reset your password.".format(otp)                mail.send(mailmsg)                flash('OTP has been
sent to your email')                return redirect(url_for('enterotp'))            except:
                msgdecline = 'Oops! Something went wrong! Email not sent'          else:
            msgdecline = 'This email is not registered!'       return
render_template('adminforgotpassword.html', msgdecline = msgdecline)



@app.route('/enterotp', methods =['GET', 'POST']) def
enterotp():
    msgdecline = ''       if request.method == 'POST' and 'otp' in
request.form :
        otp = int(request.form['otp'])          if
int(session['otp']) == otp:            msgsuccess =
'success'            return
redirect(url_for('changepassword'))          else:
```

```
        msgdecline = 'You have entered wrong OTP'       elif
request.method == 'POST':       msg = 'Please fill out the form !'      return
render_template('enterotp.html', msgdecline = msgdecline)

@app.route('/changepassword', methods =['GET', 'POST']) def
changepassword():
    msgdecline = ''             if request.method == 'POST' and 'newpassword' in request.form and
'confirmnewpassword' in request.form:
      newpassword             =             request.form['newpassword']
confirmnewpassword = request.form['confirmnewpassword']           if
newpassword == confirmnewpassword:
        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)          if
session.get("customerforgotemail"):
          cursor.execute('UPDATE customers_details SET customer_password = %s WHERE customer_email
= %s', (newpassword, session['customerforgotemail'],) )            mysql.connection.commit()
flash('Your password changed Successful! Try Login')            return redirect(url_for('customerlogin'))
elif session.get("agentforgotemail"):
          cursor.execute('UPDATE        agent_information       SET      agent_password        =        %s
         WHERE agent_email = %s', (newpassword, session['agentforgotemail'],) )
mysql.connection.commit()            flash('Your password changed Successful! Try Login')            return
redirect(url_for('agentlogin'))          elif session.get("adminforgotemail"):
          cursor.execute('UPDATE admin_details SET admin_password = %s WHERE admin_email =
%s', (newpassword, 'admin@xyz',) )
mysql.connection.commit()            flash('Password changed
Successful! Try Login')            return
redirect(url_for('adminlogin'))          else:            msgdecline
= 'Incorrect details'       else:
        msgdecline = 'Password Did Not Match!'         elif
request.method == 'POST':
      msgdecline = 'Please fill out the form !'      return
render_template('changepassword.html', msgdecline = msgdecline)



@app.route('/logout') def
logout():
  session.pop('loggedin', None)
session.pop('cemail', None)
session.pop('aemail', None)
session.pop('adminusername', None)    return
redirect(url_for('index'))


@app.route('/offline.html') def
offline():
   return app.send_static_file('offline.html')
```

```
@app.route('/service-worker.js') def
sw():
    return app.send_static_file('service-worker.js')

@app.errorhandler(404) def
invalid_route(e):
  return render_template('404.html')



if __name__ == '__main__':
    app.run(host='0.0.0.0', debug = True,port = 8080)
```

GitHub Link:

https://github.com/IBM-EPBL/IBM-Project-46339-1660745653

Video Demo Link:

https://youtu.be/p1uPR20Qjqo

PNT2022TMID3443