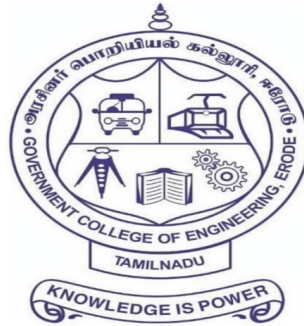


**GOVERNMENT COLLEGE OF ENGINEERING(Formerly IRTT)
ERODE-638 316**



BONAFIDE CERTIFICATE

Certified that this project titled "**CUSTOMER CARE REGISTRY**" is the bonafide work done by "**GUHAN S (731119104017), MUTHUKUMAR S (731119104030), NAVIEN V (731119104032), SURESHKUMAR R (731119104048)**" of the VII semester of **COMPUTER SCIENCE &ENGINEERING** branch during the academic year **2022-23** in the **HX8001-Professional Readiness for Innovation, Employability and Entrepreneurship**.

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.,
Professor
Department of CSE,
Government College of Engineering,
Erode - 638316

SIGNATURE OF HOD

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 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

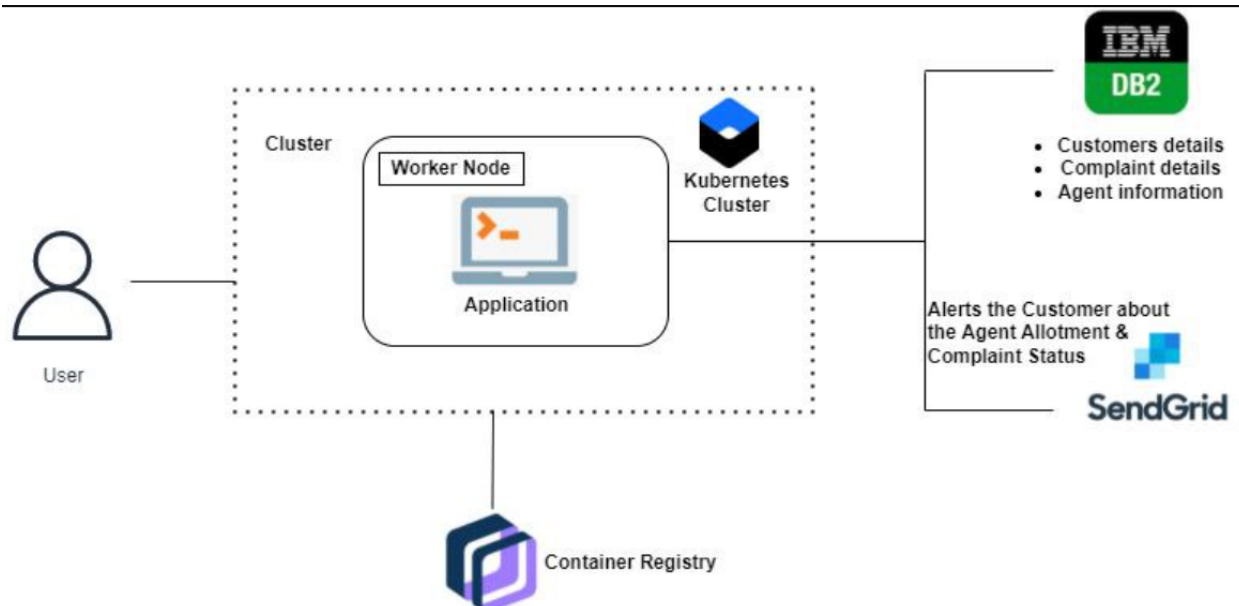
Customer care is a way of dealing with customers when they interact with your brand, products, or services. This Application has been developed to help the customer in processing their complaints.

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.



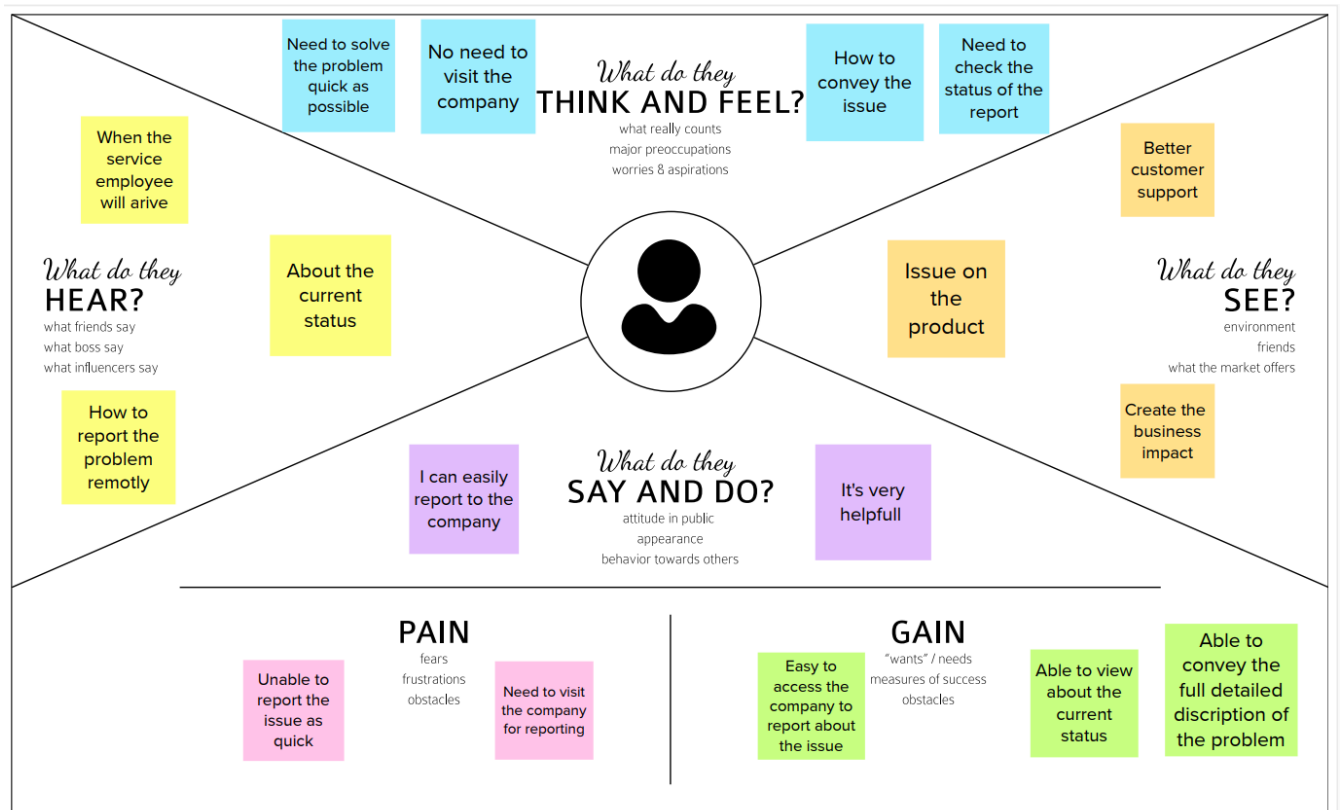
3. IDEATION & PROPOSED SOLUTION

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.

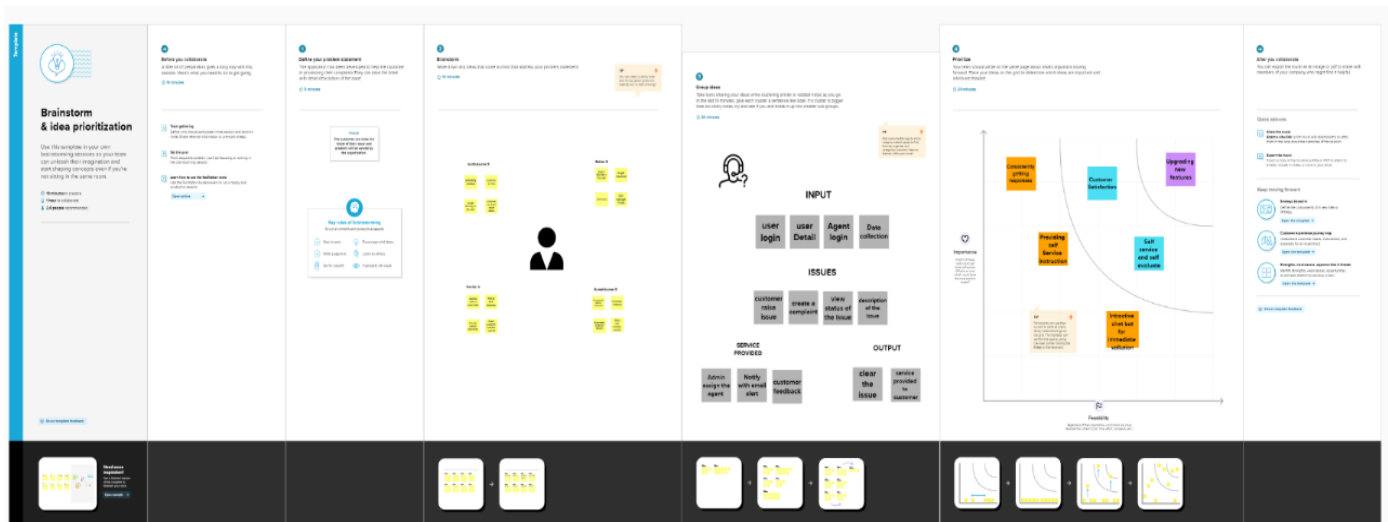


3.2 Ideation & Brainstorming

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	<p>At the Novelty Shop you will find unique products along with unique customer service. The shopping experience will be both enjoyable and exciting.</p> <p>We will offer many unusual items not found elsewhere. We will constantly strive to offer new novelties and different products to our customers.</p>
Social Impact / Customer Satisfaction	<p>Customer satisfaction goes beyond just providing good products or offering great customer service.</p> <p>It is the act of making customers feel good about their purchases. It is the idea of making customers feel valued.</p>
Business Model (Revenue Model)	<p>In the Business Model Canvas, the Customer Relationships building block describes the type of relationships a business creates with different customer segments.</p> <p>Customer relationships are designed around three major goals: customer acquisition, customer retention, and upselling.</p>

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.

Project Title: Customer Care Registry		Project Design Phase-I Problem Solution Fit		Team ID: PNT2022TMID44302	
Define CS, fit into CC	1. CUSTOMER SEGMENT(S) CS Who is your customer? i.e. working parents of 0-5 y.o. kids Our customers are usually above 16 years old. Ranging from college students to working adults to retired professionals. Also, reputed organizations too.	6. CUSTOMER CONSTRAINTS CC What constraints prevent your customers from taking action or limit their choices of solutions? i.e. spending power, budget, no cash, network connection, available devices. 1. Late replies for their queries 2. Complicated process to take over 3. High chance their queries may not be considered at all 4. Replies irrelevant to their queries 5. Advertisements shown	5. AVAILABLE SOLUTIONS AS Which solutions are available to the customers when they face the problem or need to get the job done? What have they tried in the past? What pros & cons do these solutions have? i.e. pen and paper is an alternative to digital notetaking Customers most probably use helpdesk . <u>Pros:</u> 1. Reasonably priced 2. Highly scalable for team of any size <u>Cons:</u> They do not understand the severity of all complaints and end up treating them all in the same way	Explore AS, differentiate	
	2. JOBS-TO-BE-DONE / PROBLEMS J&P Which jobs-to-be-done (or problems) do you address for your customers? There could be more than one; explore different sides. ✓ Simplifying the user account creation process ✓ Giving instant replies to the customers to their queries ✓ Providing expert solutions to the queries ✓ Assigning individual agents/experts to the customers queries ✓ Sending the status of the queries to the customer's mail	9. PROBLEM ROOT CAUSE RC What is the real reason that this problem exists? What is the back story behind the need to do this job? i.e. customers have to do it because of the change in regulations. 1. No proper registry 2. Lack of experts in a common place 3. Replies for queries from random persons 4. Communication lag 5. High-cost	7. BEHAVIOUR BE What does your customer do to address the problem and get the job done? i.e. Directly related: find the right solar panel installer, calculate usage and benefits; indirectly associated: customers spend free time on volunteering work (i.e. Greenpeace) 1. Asking their friend's opinions 2. Checking solutions in the online forums 3. Using helpdesk 4. Solve the issues themselves based on their own knowledge 5. Seeing reviews posted by the users in the website forums		
Focus on J&P, tap into BE, understand RC	3. TRIGGERS TR What triggers customers to act? i.e. seeing their neighbor installing solar panels, reading about a more efficient solution in the news. Overtime, they get disappointed with late and irrelevant replies and triggered to act	10. YOUR SOLUTION SL If you are working on an existing business, write down your current solution first, fill in the canvas, and check how much it fits reality. If you are working on a new business proposition, then keep it blank until you fill in the canvas and come up with a solution that fits within customer limitations, solves a problem and matches customer behavior. • Creating a Customer Care Registry • Simple User creation process • Customers can raise their queries to the experts • Individual agents will be assigned to each customer • Their queries will be answered earnestly • Customers can also check the status of their queries	8. CHANNELS of BEHAVIOUR CH #1 ONLINE What kind of actions do customers take online? Extract online channels from #7 #2 OFFLINE What kind of actions do customers take offline? Extract offline channels from #7 and use them for customer development. <u>ONLINE:</u> 1. https://www.helpdesk.com/ 2. https://www.google.com/ 3. https://www.quora.com/ <u>OFFLINE:</u> 1. Asking friends and colleagues 2. Take actions themselves	Identify strong TR & EM	
	4. EMOTIONS: BEFORE / AFTER EM How do customers feel when they face a problem or a job and afterwards? i.e. lost, insecure > confident, in control - use it in your communication strategy & design. × Disappointed - after they do not get instant replies for their queries × Dejected - when they get irrelevant replies even after waiting for a long time				

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 Registration through Gmail Registration through LinkedIN
FR-2	User Confirmation	Confirmation via Email Confirmation via OTP
FR-3	User Login	Login via Google Login with Email id and Password
FR-4	Admin Login	Login via Google Login with Email id and Password
FR-5	Query Form	Description of the issues 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

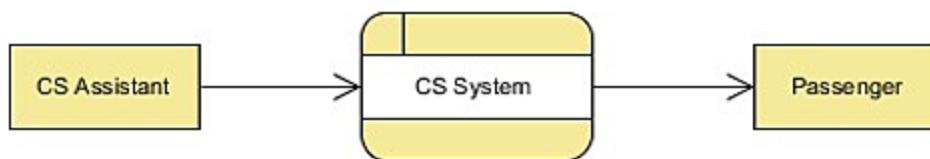
FR No:	Non-Functional Requirement	Description
NFR-1	Usability	To provide the solution to the problem
NFR-2	Security	Track of login authentication
NFR-3	Reliability	Tracking of decade status through email
NFR-4	Performance	Effective development of web application
NFR-5	Availability	24/7 service

5. PROJECT DESIGN

5.1 Data Flow Diagrams

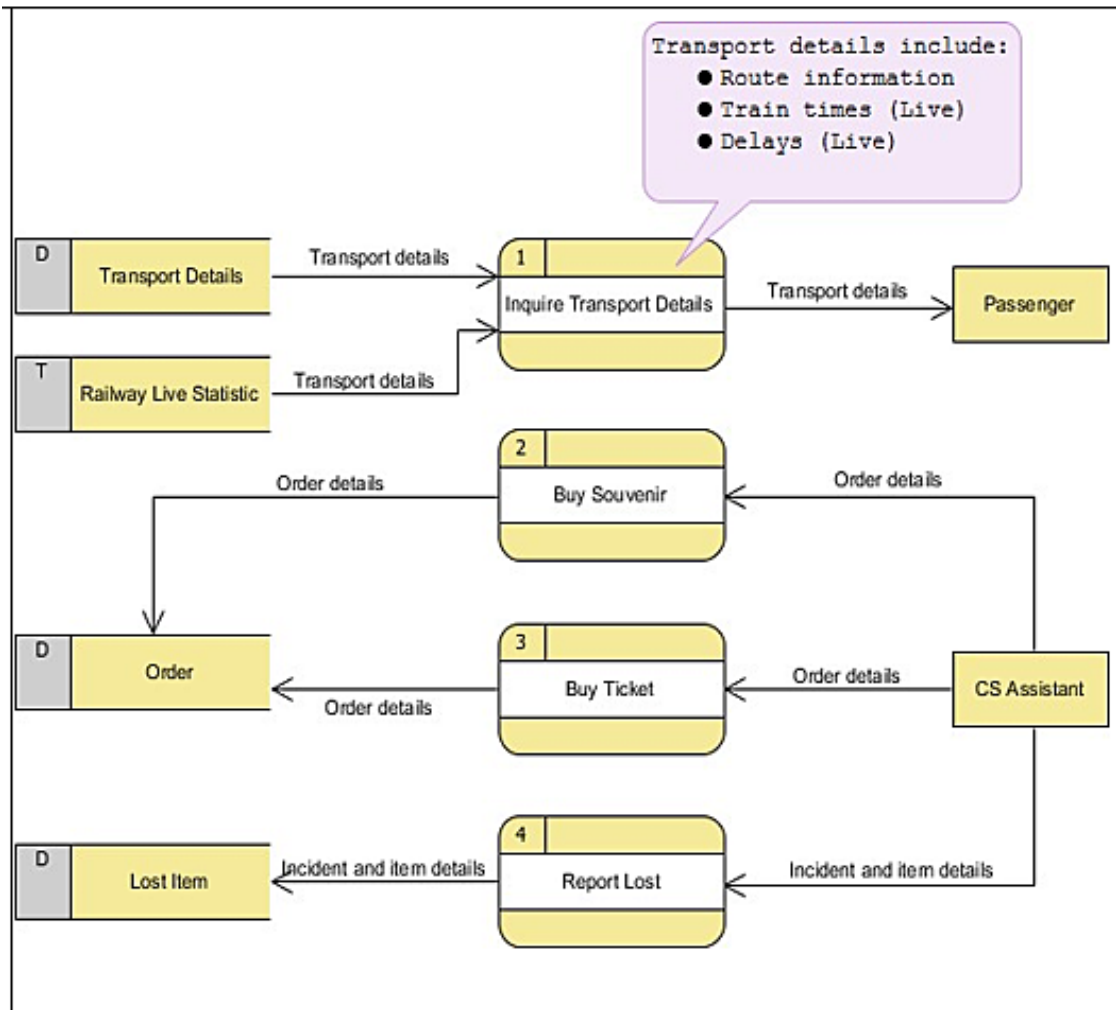
A Data Flow Diagram (DFD) is a traditional visual representation of the information flows within a system. A neat and clear DFD can depict the right amount of the system requirement graphically. It shows how data enters and leaves the system, what changes the information, and where data is stored

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.

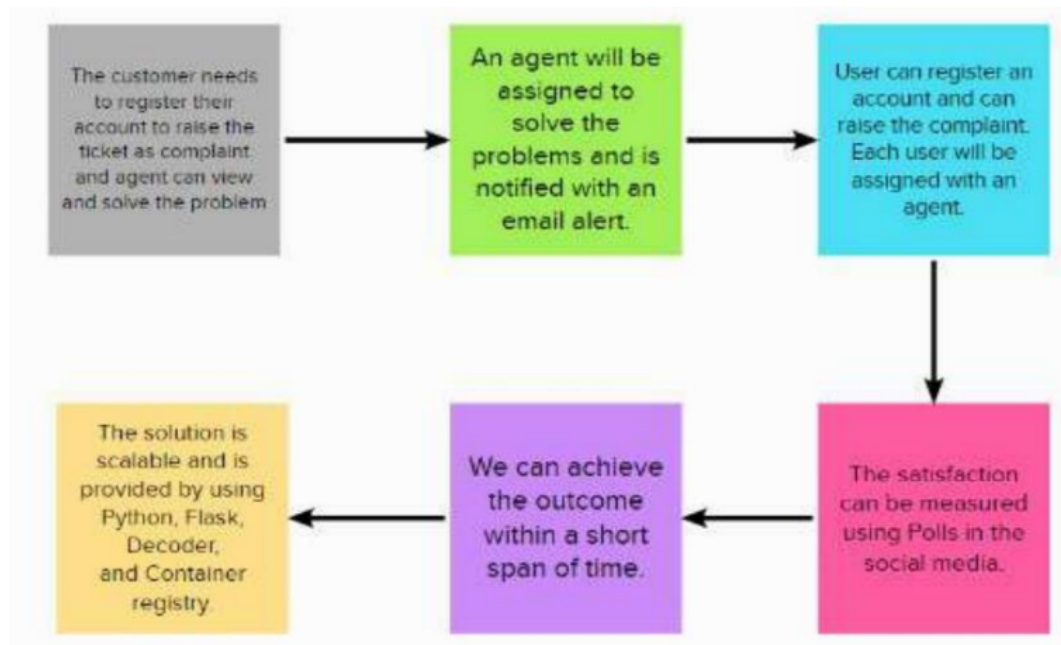
Physical diagrams show you the practical process of moving information through a system, like how your system's specific software, hardware, files, employees, and customers influences its flow of information.



5.2 Solution & Technical Architecture

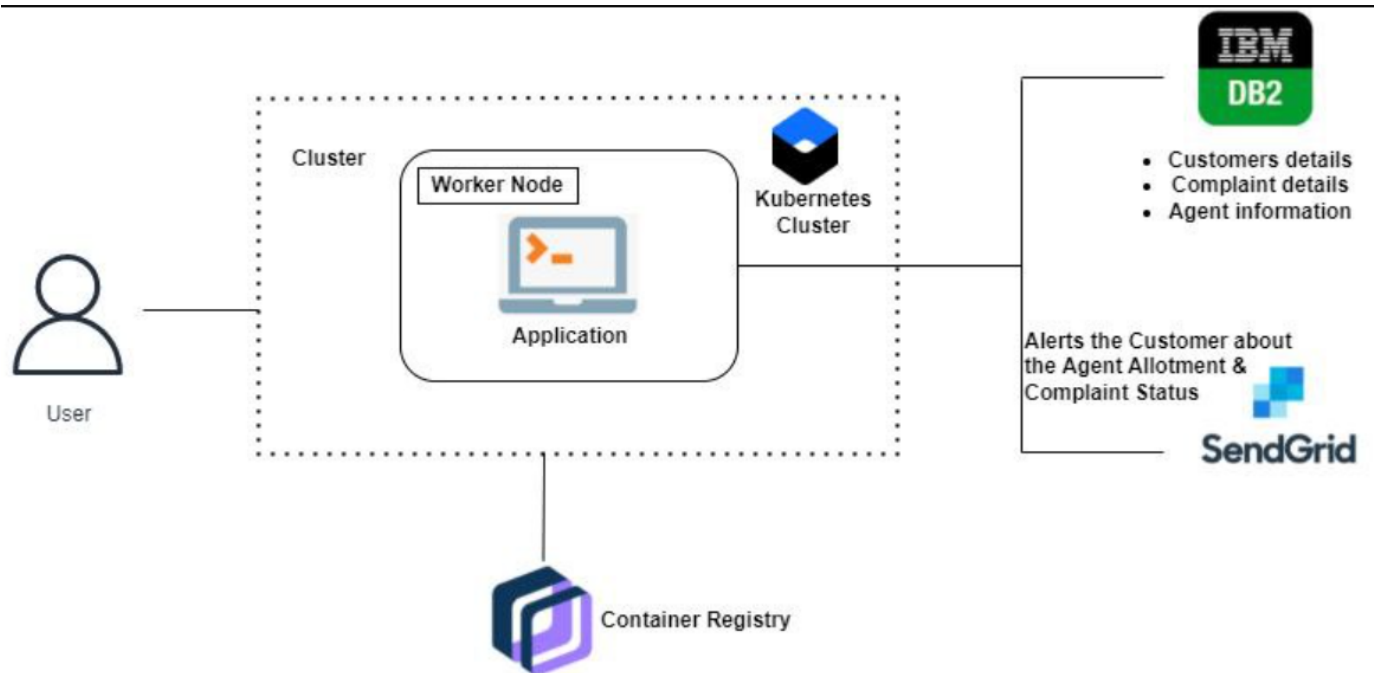
Solution architecture is a complex process with many sub-processes that bridges the gap between business problems and technology solutions. Its goals are to:

- 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 Gukan Navin 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 Navin 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	Sureshkumar Gukan 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 Navin Gukan
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	Muthukumar Gukan Navin 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).

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

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.

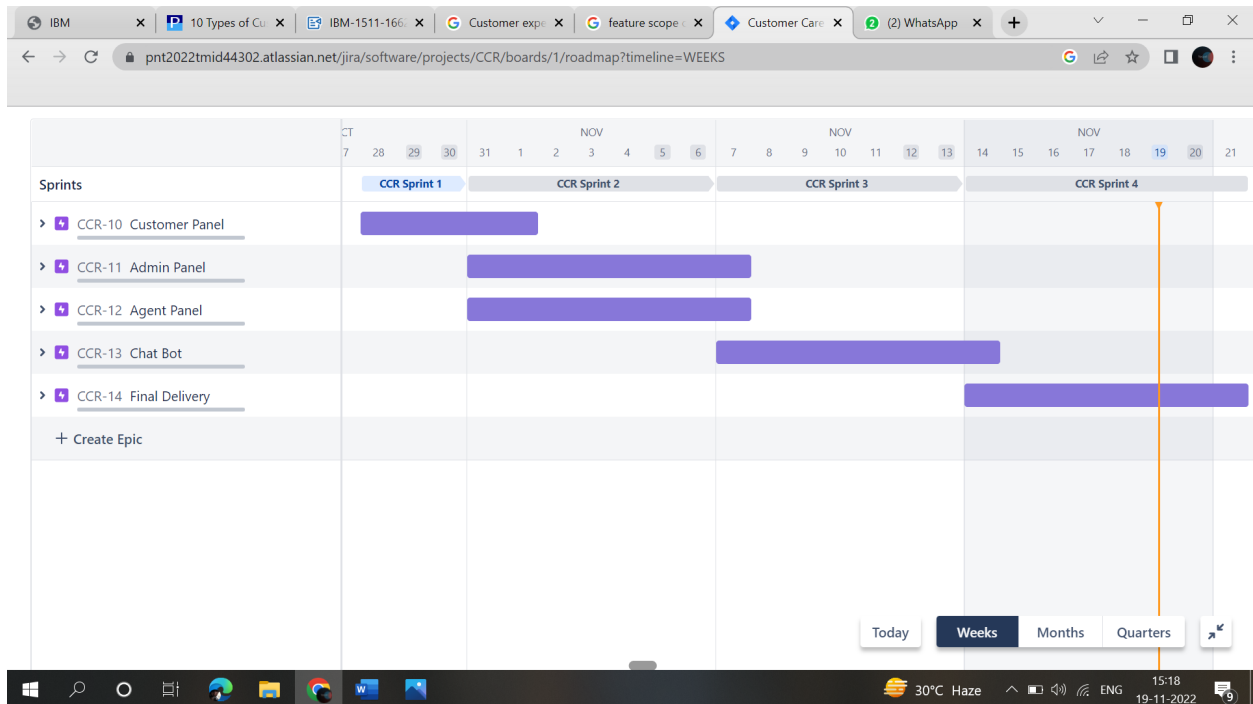
Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Sprint Release Date (Actual)
Sprint-1	20	4 Days	28 Oct 2022	01 Nov 2022	01 Nov 2022
Sprint-2	20	7 Days	31 Oct 2022	06 Nov 2022	06 Nov 2022
Sprint-3	20	8 Days	07 Nov 2022	14 Nov 2022	14 Nov 2022
Sprint-4	20	7 Days	14 Nov 2022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

6.3 Reports from JIRA

The screenshot shows the Jira Software interface for the 'Customer Care Registry' project. The left sidebar contains navigation options: PLANNING (Roadmap, Backlog, Board), DEVELOPMENT (Code), Project pages, and Add shortcut. The main area displays the 'Backlog' view for 'CCR Sprint 1' (Add dates, 5 issues). The backlog items are:

- CCR-15 As a Customer, I can register for the application by entering my email, password, an... (CUSTOMER PANEL, 2, TO DO, SR)
- CCR-2 As an admin, I can Login to the Application by entering correct login credentials an... (ADMIN PANEL, 2, TO DO, SR)
- CCR-3 As an agent, I can Login to the Application by entering correct login credentials and I will... (AGENT PANEL, 2, TO DO, SR)
- CCR-4 The Customer can directly interact to the Chatbot regarding the services offered by the Web... (CHAT BOT, 2, TO DO, SR)
- CCR-9 Container of applications using docker kubernetes and deployment the application.Cre... (FINAL DELIVERY, 2, TO DO, SR)

At the bottom, there is a '+ Create issue' button. The system tray at the bottom right shows the date and time: 15:10 19-11-2022.



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

7.1 Feature 1

cust

```
1  from flask import Blueprint, render_template, request, redirect,
    session, url_for
2  from flask_login import login_required, logout_user
3  import ibm_db
4  from .views import conn, customer
5  import uuid
6  from datetime import date, datetime
7  from .views import pass_regex
8  import re
9  import hashlib
10
11  QUERY_STATUS_OPEN = "OPEN"
12  QUERY_STATUS_ASSIGNED_AGENT = "AGENT ASSIGNED"
13  QUERY_STATUS_CLOSE = "CLOSE"
14
15  cust = Blueprint("customer", __name__)
16
17  @cust.route('/customer/')
18  @login_required
19  def profile():
20      '''
21          Custome can see his/her profile card
22      '''
23      from .views import customer
24      if hasattr(customer, 'uuid'):
25          return render_template('cust profile.html', customer = customer,
26                                  id = 0)
27      else:
28          return redirect(url_for('blue_print.logout'))
29
30  @cust.route('/customer/new', methods = ['GET', 'POST'])
31  @login_required
32  def new():
33      '''
34          Customer can create a new ticket
35      '''
```

```

36     from .views import customer
37
38     if(hasattr(customer, 'uuid')):
39         if request.method == 'POST':
40             # collecting the query entered by the customer in the
textarea
41             query = request.form.get('query-box')
42
43             msg = ""
44             to_show = False
45
46             if(len(query) == 0):
47                 msg = "Query cannot be empty!"
48                 to_show = True
49
50             else:
51                 # updating the query in the database
52                 update_query = '''
53                     INSERT INTO tickets
54                     (ticket_id, raised_by, raised_on, issue,
query_status)
55                     VALUES
56                     (?, ?, ?, ?, ?)
57                 '''
58
59             try:
60                 stmt = ibm_db.prepare(conn, update_query)
61
62                 # creating a uuid for the ticket_id
63                 ticket_id = str(uuid.uuid4())
64                 raised_by = customer.uuid
65                 raised_on = datetime.now()
66
67                 ibm_db.bind_param(stmt, 1, ticket_id)
68                 ibm_db.bind_param(stmt, 2, raised_by)
69                 ibm_db.bind_param(stmt, 3, raised_on)
70                 ibm_db.bind_param(stmt, 4, query)
71                 ibm_db.bind_param(stmt, 5, QUERY_STATUS_OPEN)
72
73                 ibm_db.execute(stmt)
74
75                 msg = "Ticket created!"
76                 to_show = True
77

```

```

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
111         DESC
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)
119         # and the agents on the side
120
121     from .views import customer
122     raised_by = customer.uuid
123
124     try:
125         stmt = ibm_db.prepare(conn, fetch_query)
126         ibm_db.bind_param(stmt, 1, raised_by)
127         ibm_db.execute(stmt)
128
129         tickets = ibm_db.fetch_assoc(stmt)
130         tickets_list = []
131
132         if tickets:
133             # means, the customer has raised some tickets before
134             while tickets != False:
135                 temp = []
136
137                 temp.append(tickets['TICKET_ID'])
138                 temp.append(str(tickets['RAISED_ON'])[0:10])
139                 temp.append(tickets['QUERY_STATUS'])
140                 temp.append(tickets['ISSUE'])
141                 temp.append(tickets['FIRST_NAME'])
142
143                 tickets_list.append(temp)
144
145                 tickets = ibm_db.fetch_assoc(stmt)
146
147         return render_template(
148             'cust tickets.html',
149             id = 2,
150             tickets_to_show = True,
151             tickets = tickets_list,
152             msg = "These are your tickets"
153         )

```

7.2 Feature 2

chat.py

```
1  from flask import render_template, Blueprint, request, session,
    redirect, url_for
2  import ibm_db
3  from datetime import datetime
4  import time
5
6  chat = Blueprint("chat_bp", __name__)
7
8  @chat.route('/chat/<ticket_id>/<receiver_name>/', methods = ['GET',
    'POST'])
9  def address(ticket_id, receiver_name):
10     '''
11         Address Column - Agent and Customer chats with one another
12
13         : param ticket_id ID of the ticket for which the chat is being
    opened
14         : param receiver_name Name of the one who receives the texts,
    may be Agent / Customer
15     '''
16     # common page for both the customer and the agent
17     # so cannot use login_required annotation
18     # so to know who signed in, we have to use the session
19     user = ""
20     sender_id = ""
21     value = ""
22     can_trust = False
23     post_url = f'/chat/{ticket_id}/{receiver_name}/'
24
25     if session['LOGGED_IN_AS'] is not None:
26         if session['LOGGED_IN_AS'] == "CUSTOMER":
27             # checking if the customer is really logged in
28             # by checking, if the customer has uuid attribute
29             from .views import customer
30
31             if(hasattr(customer, 'uuid')):
32                 user = "CUSTOMER"
33                 sender_id = customer.uuid
34                 can_trust = True
35
```



```

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 attribute
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, so 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

```

```
79             (?, ?, ?, ?)
80         '''
81     try:
82         stmt = ibm_db.prepare(conn,
message_insert_query)
83         ibm_db.bind_param(stmt, 1, ticket_id)
84         ibm_db.bind_param(stmt, 2, sender_id)
85         ibm_db.bind_param(stmt, 3, myMessage)
86         ibm_db.bind_param(stmt, 4, datetime.now())
87
88         ibm_db.execute(stmt)
89     except:
90         to_show = True
91         message = "Please send again!"
92
93     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.

PNT2022TMD44302											
Project - Customer Care Registry											
4 marks											
Test case ID	Feature Type	Component	Test Scenario	Steps To Execute	Test Data	Expected Result	Actual Result	Status	Comments	Executed By	
1	Functional	Registration Page	Customer is trying to register with the invalid data	1. Go to application 2. Click "Don't have an account? Register" 3. Enter first name, last name, select the role, password and confirm password 4. Click Register button	First Name = Bala Last Name = Abinash Role = Customer Email = suryathaya10@gmail.com Password = 12345678 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 2. Click "Don't have an account? Register" 3. Enter first name, last name, select the role, password and confirm password 4. Click Register button	First Name = Bala Last Name = Abinash Role = Customer Email = suryathaya10@gmail.com Password = 12345678 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 2. Click "Don't have an account? Register" 3. Enter first name, last name, select the role, password and confirm password 4. Click Register button	First Name = Bala Last Name = Abinash Role = Customer Email = suryathaya10@gmail.com Password = 12345678 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 2. Click "Don't have an account? Register" 3. Enter first name, last name, select the role, password and confirm password 4. Click Register button	First Name = Bala Last Name = Abinash Role = Customer Email = suryathaya10@gmail.com Password = 1234 Confirm Password = 1234	Customer should get an alert saying "Passwords must be at least 6 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 2. Click "Don't have an account? Register" 3. Enter first name, last name, select the role, password and confirm password 4. Click Register button	First Name = Bala Last Name = Abinash Role = Customer Email = suryathaya10@gmail.com Password = 12345678 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 2. Enter email, password 3. Click Login	Email = suryathaya10@gmail.com 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

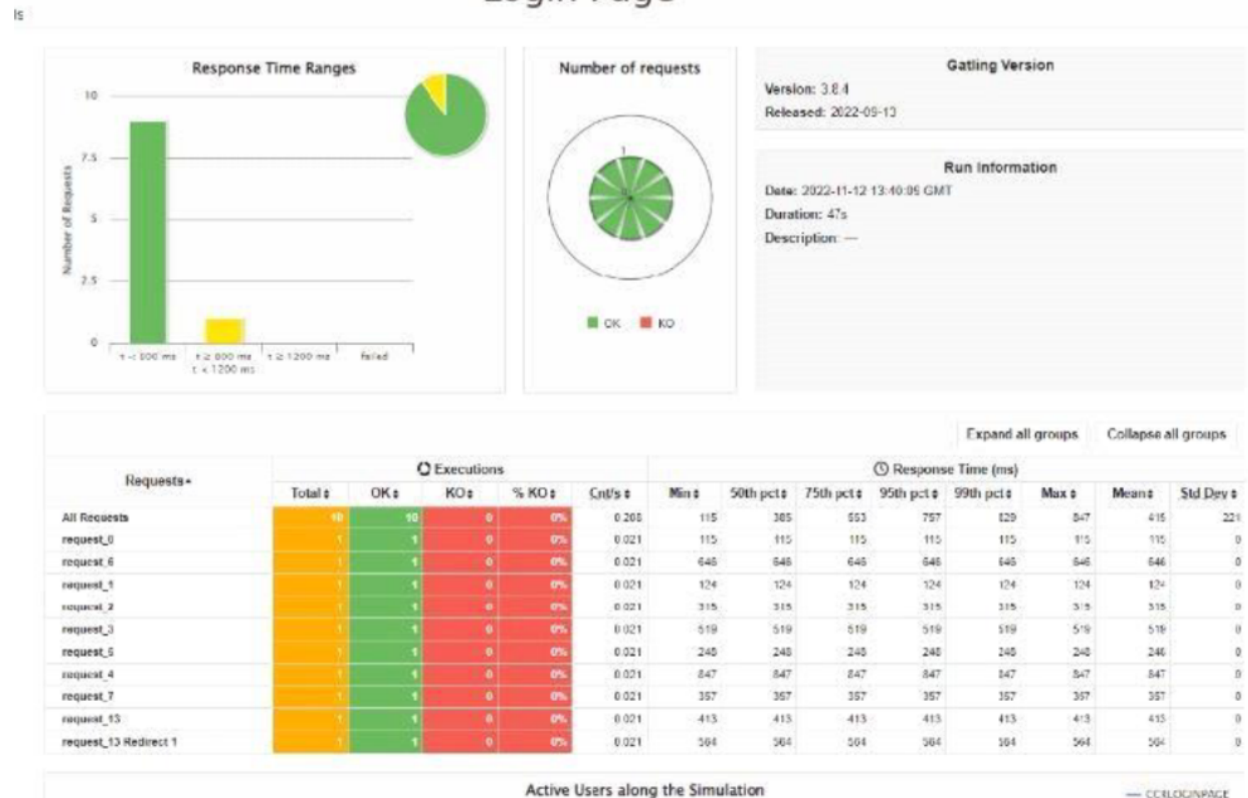
Test Case Analysis

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

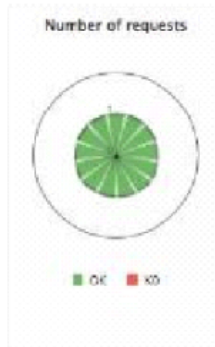
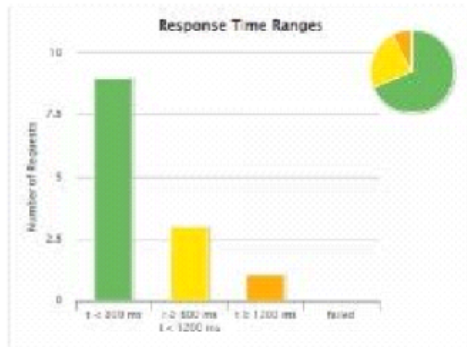
Final Report Output	4	0	0	4
---------------------	---	---	---	---

9.1 Performance Metrics

Login Page



Admin Page



Getling Version

Version: 1.0.4
Released: 2022-09-10

Run Information

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

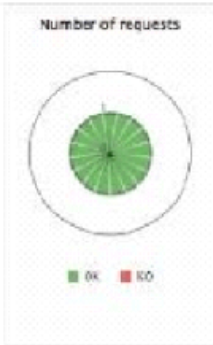
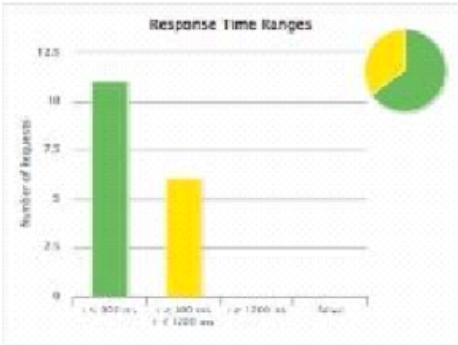
Expand all groups Collapse all groups

Requests*	Executions					Response Time (ms)									
	Total s	OK s	KO s	% KO s	COUP s	Min s	50th pct s	75th pct s	95th pct s	99th pct s	Max s	Mean s	StdDev s		
All Requests	10	9	1	10%	0.481	184	443	509	1311	1727	1670	314	438		
request_0	1	1	0	0%	0.037	104	104	104	104	104	104	104	0		
request_5	1	1	0	0%	0.037	883	883	883	883	883	883	883	0		
request_2	1	1	0	0%	0.037	300	300	300	300	300	300	300	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	208	208	208	208	208	208	208	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_11	1	1	0	0%	0.037	440	440	440	440	440	440	440	0		
request_12 Redirect 1	1	1	0	0%	0.037	1810	1810	1810	1810	1810	1810	1810	0		
request_18	1	1	0	0%	0.037	1075	1075	1075	1075	1075	1075	1075	0		
request_20	1	1	0	0%	0.037	789	789	789	789	789	789	789	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



Version: 2.0.4
Released: 2022-09-13

Date: 2022-11-13 05:14:30 GMT
Duration: 1m 15s
Description: —

Expand all groups													Collapse all groups	
Requests	Executions					Response Time (ms)								
	Total	OK	KO	% KO	Cmts	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev	
All Requests	17	17	0	0%	0.264	270	328	1092	1129	1169	1117	719	276	
request_0	1	1	0	0%	0.018	270	329	170	270	270	270	270	0	
request_1	1	1	0	0%	0.015	1090	1100	1100	1100	1100	1100	1100	0	
request_2	1	1	0	0%	0.015	897	897	897	897	897	897	897	0	
request_3	1	1	0	0%	0.015	828	828	828	828	828	828	828	0	
request_4	1	1	0	0%	0.015	899	899	899	899	899	899	899	0	
request_5	1	1	0	0%	0.015	891	891	891	891	891	891	891	0	
request_6	1	1	0	0%	0.015	844	844	844	844	844	844	844	0	
request_7	1	1	0	0%	0.015	421	421	421	421	421	421	421	0	
request_13	1	1	0	0%	0.015	499	499	499	499	499	499	499	0	
request_13 Redundant !	1	1	0	0%	0.015	1127	1127	1127	1127	1127	1127	1127	0	
request_18	1	1	0	0%	0.015	1100	1100	1100	1100	1100	1100	1100	0	
request_19	1	1	0	0%	0.015	816	816	816	816	816	816	816	0	
request_25	1	1	0	0%	0.015	890	890	890	890	890	890	890	0	
request_25 Redundant !	1	1	0	0%	0.015	1082	1082	1082	1082	1082	1082	1082	0	
request_33	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 who wants to have success and grow, needs to remember, that in order to do so, it must begin with establishing a code 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. 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 be a 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

```
1  from flask_login import UserMixin
2  import smtplib
3  from email.mime.text import MIMEText
4  from email.mime.base import MIMEBase
5  from email.mime.multipart import MIMEMultipart
6  from email import encoders
7  import os
8  class Customer(UserMixin):
9      def set(self, uuid, first_name, last_name, email, password, date):
10         '''
11             Method to initialise the Customer
12         '''
13         self.uuid = uuid
14         self.first_name = first_name
15         self.last_name = last_name
16         self.email = email
17         self.password = password
18         self.date = date
19     def get_id(self):
20         '''
21             Method to return the uuid of the Customer
22         '''
23         return (self.uuid)
24 class Agent(UserMixin):
25     def set(self, uuid, first_name, last_name, email, password, date,
26 confirm):
27         '''
28             Method to initialise the Agent
29         '''
30         self.uuid = uuid
31         self.first_name = first_name
32         self.last_name = last_name
33         self.email = email
34         self.password = password
35         self.date = date
36         self.confirm = confirm
37     def get_id(self):
38         '''
```

```

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

```

views.py

```
1  from flask import Blueprint, render_template, request, redirect,
    session, url_for
2  import hashlib
3  import re
4  from flask_login import login_required, login_user, logout_user
5  import ibm_db
6  import uuid
7  from datetime import date
8  import random
9  from registry.model import Customer, Agent, Admin, Mail
10 from ..secret import connection_string
11
12 views = Blueprint("blue_print", __name__)
13 email_regex = r"\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b"
14 pass_regex = r"^[A-Za-z0-9_-]*$"
15
16 customer = Customer()
17 agent = Agent()
18 admin = Admin()
19 mail = Mail()
20
21 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", '', '')
22
23 @views.route('/logout')
24 @login_required
25 def logout():
26     session.pop('LOGGED_IN_AS')
27     logout_user()
28
29     return redirect(url_for('blue_print.login'))
30
31 @views.route('/', methods = ['GET', 'POST'])
32 @views.route('/login', methods = ['GET', 'POST'])
33 def login():
34     # if method is POST
35     if request.method == 'POST':
36         # getting the data entered by the user
37         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 = "?"
81         stmt = ibm_db.prepare(conn, mail_check_query)
82         ibm_db.bind_param(stmt, 1, email)
83         ibm_db.execute(stmt)
84
85         account = ibm_db.fetch_assoc(stmt)
86
87         if account:
88             # valid customer
89             # i.e, mail is present in the database
90
91             # checking if the customer entered a valid
password now
92             # encrypting the entered password
93             passcode =
str(hashlib.sha256(password.encode()).hexdigest())
94
95             # now checking if the encrypted string is same
as that of the one in database
96             if (account['PASSCODE'] == passcode):
97                 msg = "Valid Login"
98                 to_show = True
99
100             # creating a customer object
101             customer.set(
102                 account['CUST_ID'],
103                 account['FIRST_NAME'],
104                 account['LAST_NAME'],
105                 account['EMAIL'],
106                 account['PASSCODE'],
107                 account['DATE_JOINED']
108             )
109
110             session.permanent = False
111             session['LOGGED_IN_AS'] = "CUSTOMER"
112             login_user(customer, remember=True)
113
114             return redirect(url_for('customer.profile'))
115
116         else:
117             # customer entered invalid password
118             msg = "Invalid password"
119             password = ""
120             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,

```

```

196         email = email,
197         password = password
198     )
199
200     return render_template('login.html')
201
202 @views.route('/register', methods = ['GET', 'POST'])
203 def register():
204     # if method is POST
205     if request.method == 'POST':
206         # getting all the data entered by the user
207         first_name = request.form.get('first_name')
208         last_name = request.form.get('last_name')
209         email = request.form.get('email')
210         password = request.form.get('password')
211         confirm_password = request.form.get('confirm_password')
212         role = request.form.get('role-check')
213
214         msg = ""
215         to_show = False
216
217         # validating the inputs
218         if len(first_name) < 3:
219             msg = "First Name must be atleast 3 characters long!"
220             to_show = True
221
222         elif len(last_name) < 1:
223             msg = "Last Name must be atleast 1 characters long!"
224             to_show = True
225
226         elif(not (re.fullmatch(email_regex, email))):
227             msg = "Please enter valid email"
228             to_show = True
229
230         elif((len(password) < 8) or (len(confirm_password) < 8)):
231             msg = "Password must be atleast 8 characters long!"
232             to_show = True
233
234         elif (password != confirm_password):
235             msg = "Passwords do not match"
236             to_show = True
237
238         elif (not (re.fullmatch(pass_regex, password))):
239             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 database
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,
325             passcode, date_joined, confirmed)
326             VALUES (?, ?, ?, ?, ?, ?, ?)
327         '''
328
329         # creating a unique id for the agent
330         agent_id = str(uuid.uuid4())
331         date_joined = date.today()
332         confirmed = False
333
334         # encrypting the agent's password with SHA-256
335         passcode =
336         str(hashlib.sha256(password.encode()).hexdigest())
337
338         try:
339             stmt = ibm_db.prepare(conn, agent_input_query)
340             ibm_db.bind_param(stmt, 1, agent_id)
341             ibm_db.bind_param(stmt, 2, first_name)
342             ibm_db.bind_param(stmt, 3, last_name)
343             ibm_db.bind_param(stmt, 4, email)
344             ibm_db.bind_param(stmt, 5, passcode)
345             ibm_db.bind_param(stmt, 6, date_joined)
346             ibm_db.bind_param(stmt, 7, confirmed)
347
348             ibm_db.execute(stmt)
349
350             msg = "Account created! Please login"
351             to_show = True
352
353             # re-directing the agent to the login page
354             return render_template('login.html', message =
355             msg, to_show = to_show)
356
357         except:
358             msg = "Something went wrong!"
359             to_show = True
360
361         return render_template(
362             'register.html',
363             to_show = to_show,
364             message = msg,
365             first_name = first_name,

```

```

363         last_name = last_name,
364         email = email,
365         password = password,
366         confirm_password = confirm_password,
367         role = role
368     )
369     return render_template('register.html')
370
371 @views.route('/forgot', methods = ['GET', 'POST'])
372 def forgot():
373     '''
374         Changing the password for the customer / agent
375     '''
376     msg = ""
377     to_show = False
378
379     if request.method == 'POST':
380         # getting the email and role entered by the user (Customer or
381         # Agent)
382         email = request.form.get('email')
383         role = request.form.get('role-check')
384
385         if len(email) == 0:
386             msg = "Email cannot be empty!"
387             to_show = True
388
389         elif(not (re.fullmatch(email_regex, email))):
390             msg = "Email valid email!"
391             to_show = True
392
393         else:
394             if role == "Customer":
395                 # the user is a customer
396                 # checking if the email entered by the customer is in
397                 # the database
398                 # query to check if the customer's mail exists in the
399                 # customer table
400                 mail_check_query = '''
401                 SELECT email FROM customer WHERE email = ?
402                 '''
403                 stmt = ibm_db.prepare(conn, mail_check_query)
404                 ibm_db.bind_param(stmt, 1, email)
405                 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":

```

```

441         # the user is an Agent
442         # checking if the email entered by the agent is in the
        database
443         # query to check if the agent's mail exists in the agent
        table
444         mail_check_query = '''
445             SELECT email FROM agent WHERE email = ?
446         '''
447
448         stmt = ibm_db.prepare(conn, mail_check_query)
449         ibm_db.bind_param(stmt, 1, email)
450         ibm_db.execute(stmt)
451         account = ibm_db.fetch_assoc(stmt)
452
453         if account:
454             # then the email is in the database
455             # the agent is a valid agent then
456             # generating a random 6-digit number to send to the
        customer
457             randomNumber = random.randint(11111111, 99999999)
458
459             # sending this number to the customer's email
460             values = mail.sendEmail(
461                 "Forgot Password?",
462                 f'Your verification code is
        <strong>{randomNumber}</strong>',
463                 [f'{email}']
464             )
465
466             # encrypting the random number sent to the customer
        using SHA
467             code =
        str(hashlib.sha256(str(randomNumber).encode()).hexdigest())
468
469             if (not len(values.keys())) == 0:
470                 # something happened fishy
471                 msg = "Please try again!"
472                 to_show = True
473
474             else:
475                 # the mail with the random number is sent
        successfully
476                 # redirecting the customer to the code entering
        page

```



```

477             return
478         redirect(f'/forgot/{role}/{email}/{code}/')
479     else:
480         # the email is not in the database
481         # just someone trying to do fishy
482         msg = "Agent does not exist!"
483         to_show = True
484
485     return render_template(
486         'forgot.html',
487         message = msg,
488         to_show = to_show
489     )
490
491 @views.route('/forgot/<role>/<email>/<code>/', methods = ['GET',
492     'POST'])
493 def code(role, email, code):
494     if request.method == 'POST':
495         # getting the code entered by the customer
496         myCode = str(request.form.get('code-input'))
497
498         if len(myCode) == 0:
499             msg = "Code cannot be empty!"
500             to_show = True
501
502         else:
503             return render_template(
504                 'change password.html',
505                 role = role, email = email

```

GitHub & Project Demo Link

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

<https://drive.google.com/file/d/1c40LL5wwwoXQZ-lmQ21iEzhfCOjBtj-TM/view?usp=drivesdk>