Project Report

CUSTOMER CARE REGISTRY

CUSTOMER CARE REGISTERY

Project Name : Customer Care Registry

Project Domain : Cloud Application Development

College : SRM Valliammai Engineering College,Chengalpetu

College SPOC : Mr. S . Senthil Murugan

Team ID : PNT2022TMID21762

Team Size 4

Team Members : A - T. Gugashri

B - A. Charlet Priscilla

C - S.M. Durgapriya

D - I. Harini

Team Mentor : Mr. A . Pandian

Team Evaluator : Dr. N. Subhashini

Github Link : **https://github.com/IBM-EPBL/IBM-Project-18467-1668680564**

Project Demo Link : **https://youtu.be/O316pDOo5Tk**

CONTENT

INTRODUCTION

1.1 PROJECT OVERVIEW

Online customer care and service center is a web-based application Developed using Python programming language. With a platform of a typical “service center”, this system provides online technical services to its customers on a 24×7 basis. The whole process involves writing a large volume of data in registers and preparing several reports daily. The basic services include hardware and software of a computer. It also maintains a database of their customers, and many more. Online customer care and service center application is developed to automate all the office activities of a typical service center.The main objective of this Online Customer Care and Service Center software is to develop an information system to store, maintain, update and process data relating to the shop. It will prepare various reports to aid in smooth and speedy functioning of ‘Service Center’ activities.

1.2 PURPOSE

This includes supporting consumers with their enquiries and complaints, undertaking investigations on their behalf, educating consumers and businesses via our outreach work, and sharing information and resources via campaigns, media, and our website.

2. LITERATURE SURVEY

2.1 EXISTING PROBLEM

The present computer service centers generally keep the details of the customers and products in word documents, spreadsheets or paper registers, and the management of all records is illegal to some extent. There are problems relating to redundancy of data like customer name and address, details of their account and also allocation of duties to the employees.

When a customer takes some kind of service, the charge is calculated manually, and this process is time consuming. Also, regular and overtime duties are not maintained properly. This leads to improper calculation of employees becoming quite complicated for every employee. Another problem usually faced by the organization which has been solved in the proposed Online Customer Care and Service Center Project is the frequent complaints by the customers for not getting timely services.

2.2 REFERENCE

https://www.helpdesk.com/https://freshdesk.com/helpdesk-software

https://freshdesk.com/resources/case-study/hamleys

https://pulsedesk.com

https://www.redpoints.com/blog/amazon-fake-reviews

2.3 PROBLEM STATEMENT DEFINITION

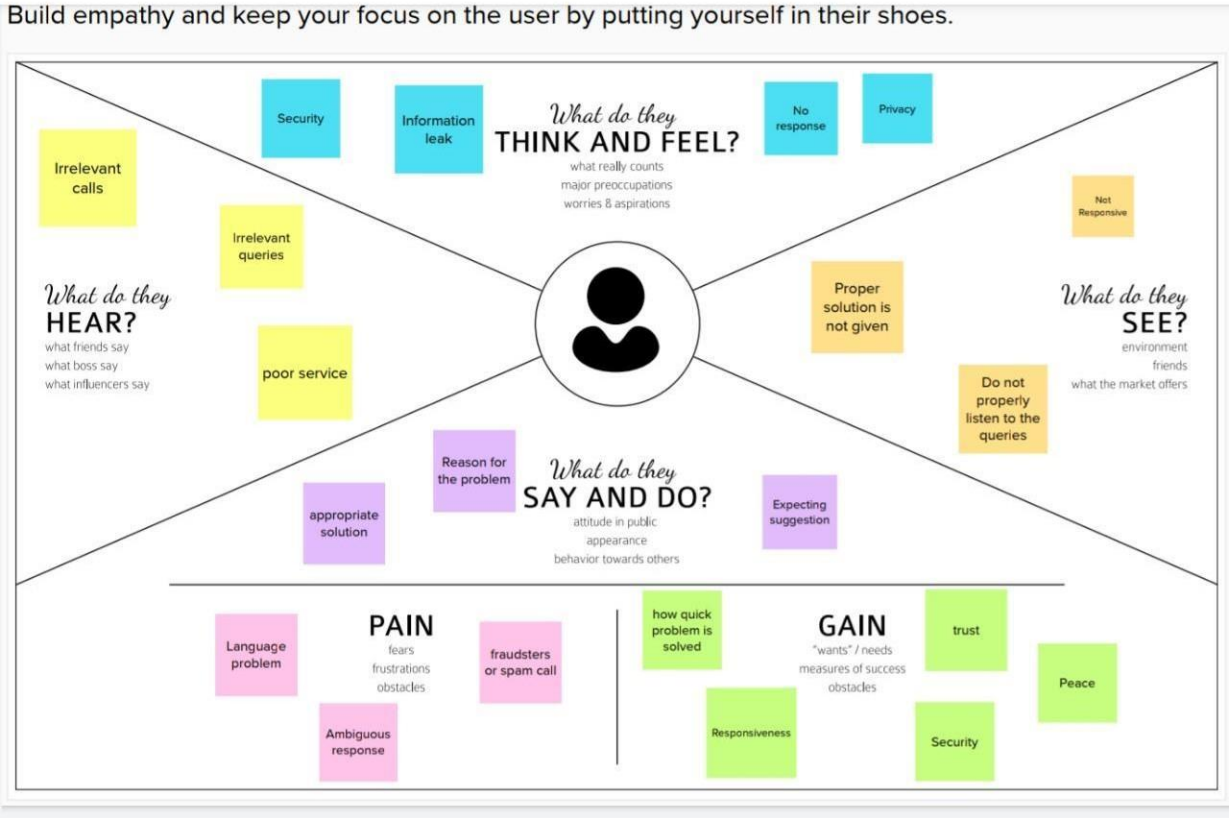
I am Shri and I am a regular customer in famous e-commerce websites like Amazon, Flipkart. I order regularly. The problem I have is that most times, I don't have any reliable sources to clear my doubts in some of the products I buy.There are reviews and customer ratings on those websites,

but somehow, I don’t feel they are authentic and real. It would make my world if those replies were from a real expert, and I could clarify all my doubts in a single platform. Of course, I would need instant replies from a real expert who knows about the products I am asking for.

3. IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

If you want to create a memorable experience for your customers you need to dig a little deeper into who your customer is and what they want. You need to get into your customers head and consider who they are influenced by, what their pain points are, and their goals and challenges. One of the most popular ways to extrapolate this information and begin improving your customers experience is by using a Customer Empathy Map.If you're ready to get started on your own, you can access our free and editable Customer Empathy Map template by clicking on the image below. For a more detailed explanation, keep reading this post !



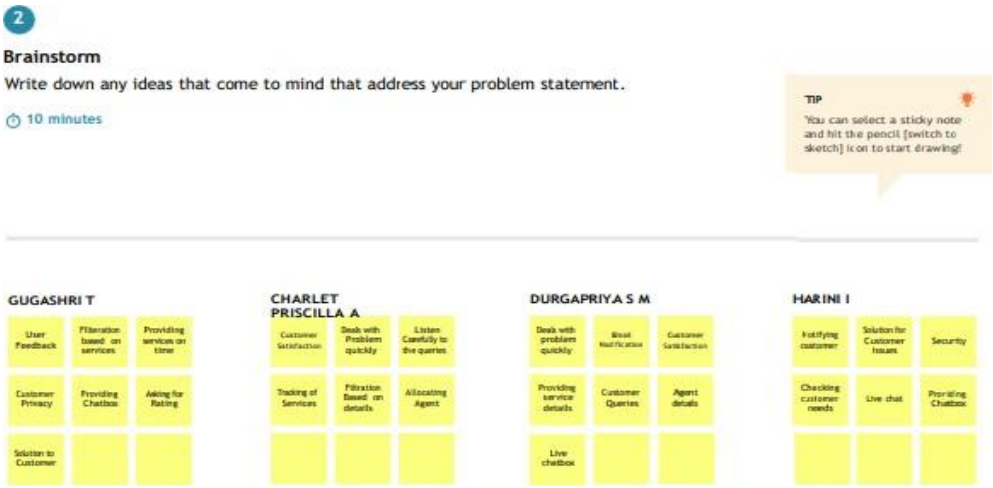
3.2 IDEATION AND BRAINSTORMING

Having a strong ideation framework is as important as the framework of any company department. This must be reflected in the company's culture, not only with words but also through actions. The model in Figure 1 below shows what should be the prominent pillars of ideation for firms embarking on this journey or trying to reorganize their innovation department. Each attribute under this model has a few important activities that can ensure a successful innovation pipeline for your business

STEP 1:


1
Define your problem statement
What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.
5 minutes
PROBLEM
How might we [your problem statement]?
Key rules of brainstorming
To run an smooth and productive session
Stay in topic.
Encourage wild ideas.
Defer judgment.
Listen to others.
Go for volume.
If possible, be visual.

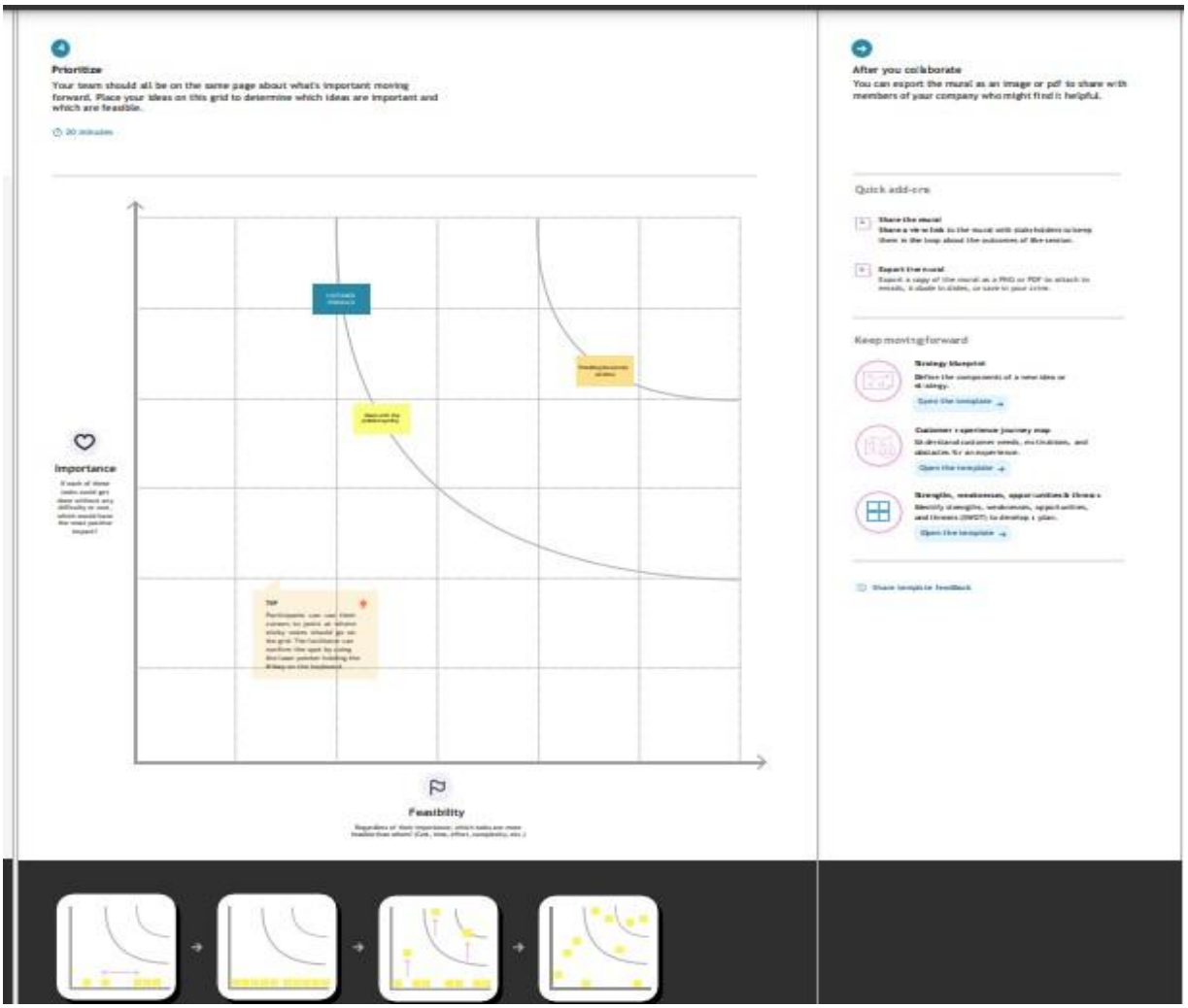
STEP 2:


2
Brainstorm
Write down any ideas that come to mind that address your problem statement.
10 minutes
TIP
You can select a sticky note and hit the pencil [switch to sketch] icon to start drawing!
GUGASHRI T
User Feedback
Filteration based on services
Providing services on time
Customer Privacy
Providing Chatbox
Asking for Rating
Solution to Customer
CHARLET PRISCILLA A
Customer Satisfaction
Deals with Problem quickly
Listen Carefully to the queries
Tracking of Services
Filtration Based on details
Allocating Agent
DURGAPRIYA S M
Deals with problem quickly
Email Notification
Customer Satisfaction
Providing service details
Customer Queries
Agent details
Live chatbox
HARINI I
Fortifying customer
Solution for Customer Issues
Security
Checking customer needs
Live chat
Providing Chatbox

STEP 3:


3
Group ideas
Take turns sharing your ideas while clustering similar or related notes as you go. In the last 10 minutes, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you and break it up into smaller sub-groups.
20 minutes
CUSTOMER
CHATBOX
FEEDBACKS
INFORMATION
SECURITY
SERVICES

STEP 4:



3.3 PROPOSED SOLUTION

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.

Admin: 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.

Project team shall fill the following information in the proposed solution template.

S.NO.	PARAMETER	DESCRIPTION
01	Problem Statement (Problem to be solved)	To solve customer issues using Cloud Application Development.
02	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.
03	Novelty / Uniqueness	Assigned Agent Routing, Automated Ticket Closure, Status Shown to the Customer, and Backup data in case of failures.

Project team shall fill the following information in the proposed solution template.

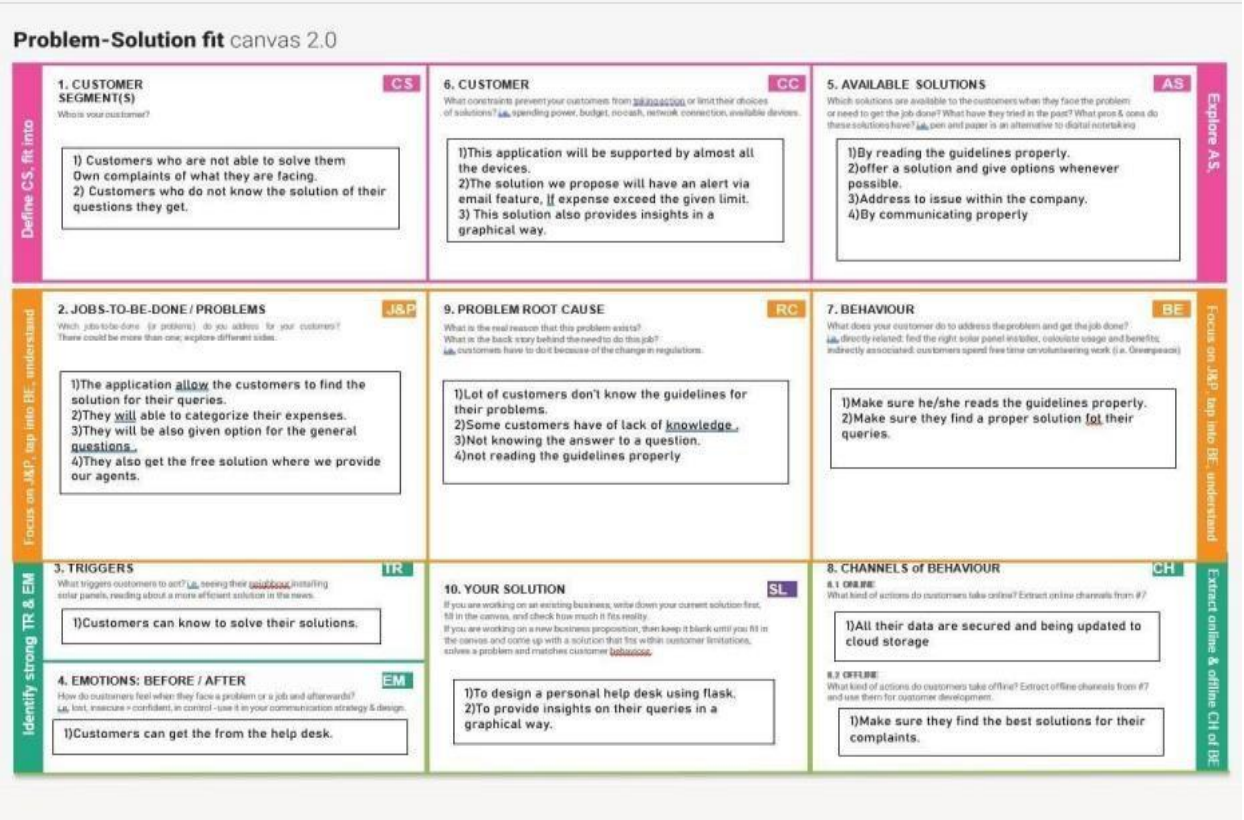
S.NO.	PARAMETER	DESCRIPTION
04	Social Impact / Customer Satisfaction	Customer Satisfaction, Customer can track their status and Easy agent communication.
05	Business Model (Revenue Model)	● Key Partners are Third-party applications, agents, and customers. ● Activities held as Customer Service, System Maintenance. ● Key Resources support Engineers, Multi-channel. ● Customer Relationship have 24/7 Email Support, Knowledge-based channel. ● Cost Structure expresses Cloud Platform, Offices

Project team shall fill the following information in the proposed solution template.

S.NO.	PARAMETER	DESCRIPTION
06	Scalability of the Solution	The real goal of scaling customer service is providing an environment that will allow your customer service specialists to be as efficient as possible. An environment where they will be able to spend less time on grunt work and more time on actually resolving critical customer issues .

3.4 PROBLEM SOLUTION FIT

This occurs when you have evidence that customers care about certain jobs, pains, and gains. At this stage you've proved the existence of a problem and have designed a value proposition that addresses your customers' jobs, pains and gains.



4. REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENT

Functional requirements may involve calculations, technical details, data manipulation and processing, and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describe all the cases where the system uses the functional requirements, these are captured in use cases.

Functional Requirements:

Following are the functional requirements of the proposed solution.

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

4.2 NON FUNCTIONAL REQUIREMENT

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.

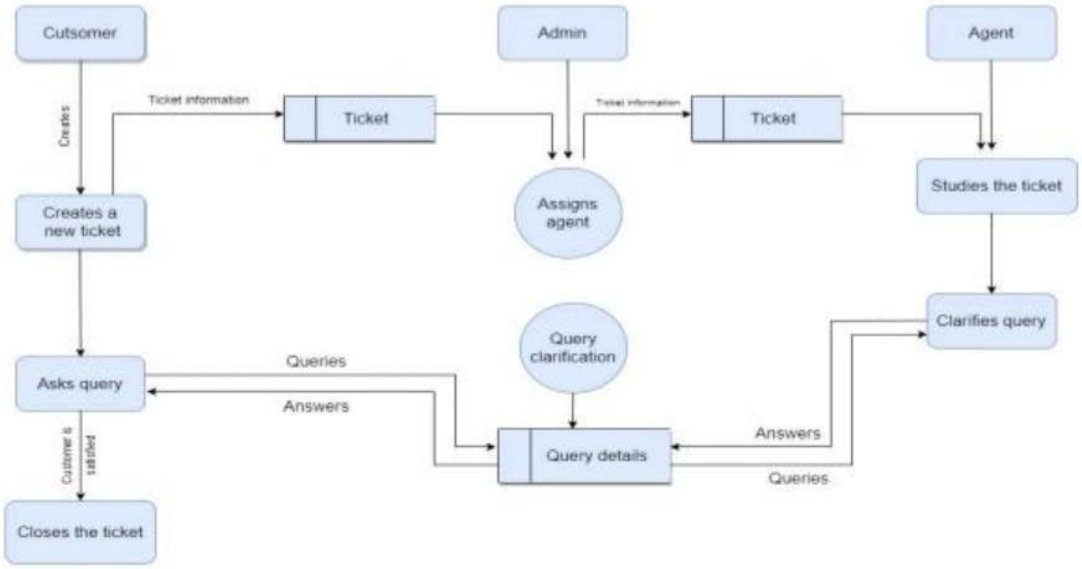
Non-functional Requirements:

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

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

5. PROJECT DESIGN

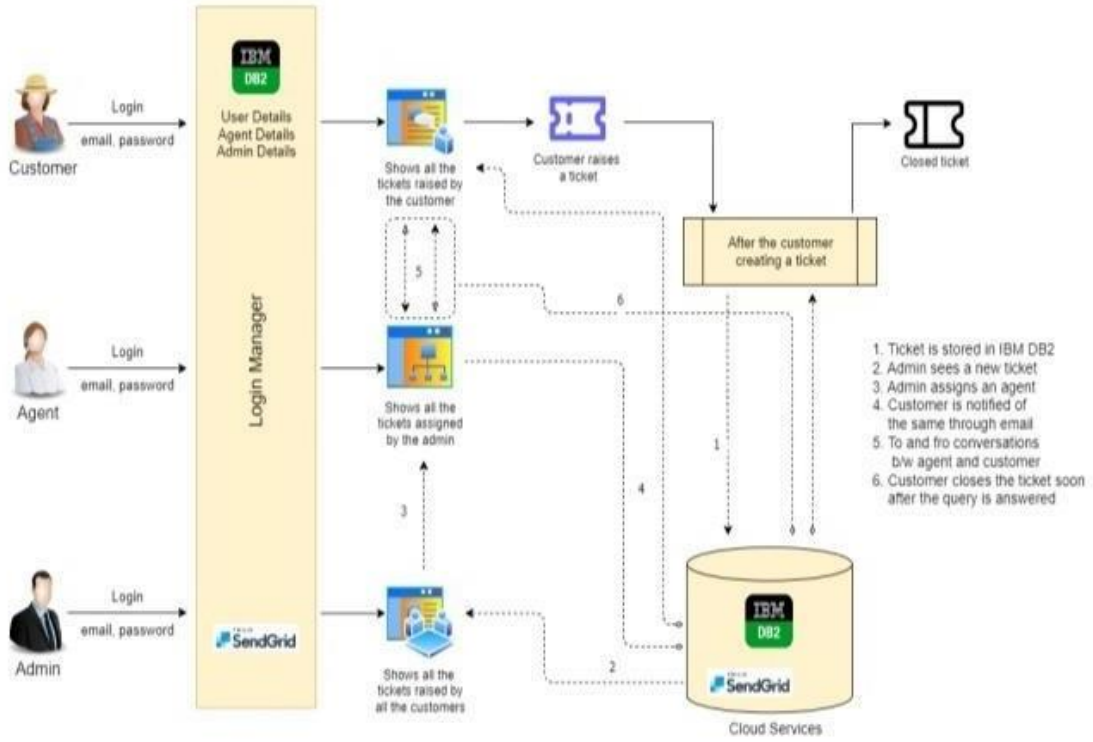
DATA FLOW DIAGRAM



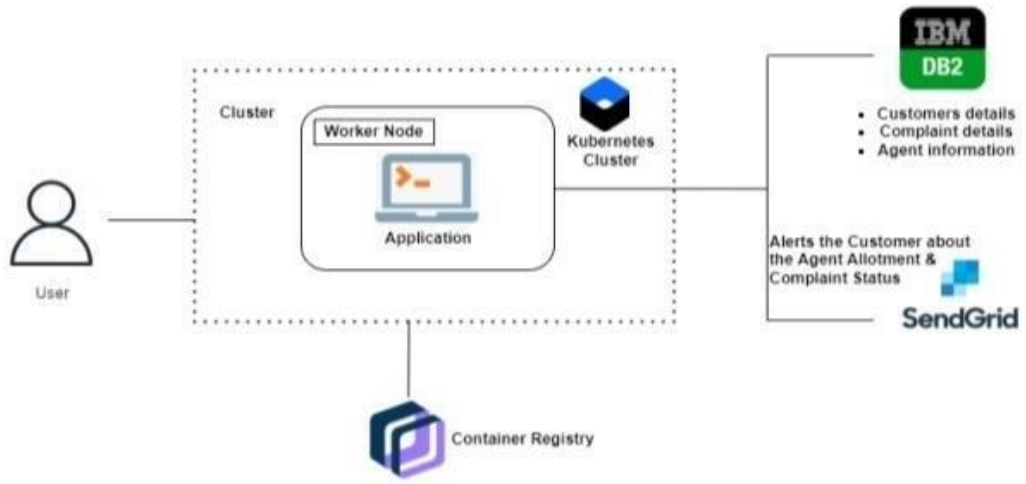
Data Flow Diagram (DFD) provides a visual representation of the flow of information (i.e. data) within a system. By creating a Data Flow Diagram, you can tell the information provided by and delivered to someone who takes part in system processes, the information needed in order to complete the processes and the information needed to be stored and accessed. Data Flow Diagram is widely-used in software engineering. You can use DFD in modeling information systems. This article describes and explains Data Flow Diagram (DFD) by using a customer service system as an example

5.2 SOLUTION AND TECHNICAL ARCHITECTURE

Solution Architecture



Technical Architecture



Solution Architects are most similar to project managers, ensuring that all parties, including stakeholders, are on the same page and moving in the right direction at all stages. Technical architects manage all activities leading to the successful implementation of a new application

5.3 USER STORIES

User stories, based on the estimated size, are taken for implementation in an iteration. User stories should be granular enough that they can be completed within an iteration and cannot be continued in the following iteration. If a story cannot be completed within an iteration, the same should be split logically. User stories are prioritized by the product owner based on business priority and are available at the top of the product backlog. The dev team pulls the stories into an iteration backlog and implements them. The Definition of Done(DOD) for user stories is decided by the team which includes acceptance criteria, and processes that need to be followed like unit testing, regression testing, code review, etc. The story is said to be “done” only when it meets the preset Definition of Done

6. PROJECT PLANNING AND SCHEDULING

6.1 SPRINT PLANNING AND ESTIMATION

In Scrum Projects, Estimation is done by the entire team during Sprint Planning Meeting. The objective of the Estimation would be to consider the User Stories for the Sprint by Priority and by the Ability of the team to deliver during the Time Box of the Sprint.

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	8 days	22/10/2022	31/10/2022	20	26/10/2022
Sprint 2	10	8 days	28/10/2022	05/11/2022	10	03/11/2022
Sprint 3	20	8 days	5/11/2022	12/11/2022	20	12/11/2022
Sprint 4	10	8 days	13/11/2022	19/11/2022	10	19/11/2022

In case you’re unfamiliar, a sprint schedule is a document that outlines sprint planning from end to end. It’s one of the first steps in the agile sprint planning process—and something that requires adequate research, planning, and communication. A scrum master or coach typically facilitates sprint planning in order to ensure that the discussion is effective and that there is agreement to the sprint goal and that the appropriate product backlog items are included in the sprint backlog.

6.2SPRINT DELIVERY SCHEDULE

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story points	Priority	Team Members
Sprint 1	Registration	USN-1	As a user, I can register for the web page by entering my email then password and confirming my password.	10	High	Charlet Priscilla Gugasgri
Sprint 1	Email Confirmation	USN-2	As a user,the web user will receive confirmation email once I have registered for the application.	10	High	Charlet Priscilla Harini
Sprint 2	Login	USN-3	As a user,I can login to the application by entering email and password.	10	High	Durgapriya Charlet Priscilla
Sprint 2	Details	USN-4	As a customer I can fill my details and personal information.	10	High	Durgapriya Gugashri
Sprint 3	Cloud Database	USN-5	As an administrator I can stored a details in the cloud database administrator can stored data into the database cloud.	10	High	Harini Durgapriya
Sprint 3	Details	USN-6	As a customer, I can send request to the website for booking or service issue and any other problem requires.	5	low	Harini CharletPriscilla
Sprint 3	Assign task	USN-7	As an administrator, can assign task to particular agent.	10	High	Durgapriya Harini
Sprint 4	Details	USN-8	As an agent take the customer details from the cloud database as a customer I can send the website for the booking or service issue.	5	Low	Gugashri Harini
Sprint 4	Email	USN-9	As a customer, I can receive the response.	10	High	Gugashri CharletPriscilla

7. CODING AND SOLUTION

7.1 ADMIN ASSIGNING AGENT TO A TICKET

CODE

```
@app.route('/addnewagent', methods=["GET", "POST"])
def addagent():
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        domain = request.form['domain']
        password = request.form['password']
        try:
            sql = "insert into agents values(?,?,?,?,?,?,?)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, username)
            ibm_db.bind_param(stmt, 2, name)
            ibm_db.bind_param(stmt, 3, email)
            ibm_db.bind_param(stmt, 4, phone)
            ibm_db.bind_param(stmt, 5, password)
            ibm_db.bind_param(stmt, 6, domain)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from agents"
        agents = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            agents.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)

        return render_template('agents.html', agents=agents)
```

```python
@app.route('/updatecomplaint', methods=["GET", "POST"])
def updatecomplaint():
    if request.method == 'POST':
        cid = request.form['cid']
        solution = request.form['solution']
        try:
            sql = "update complaints set solution =?,status=1 where c_id = ? and assigned_agent=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, solution)
            ibm_db.bind_param(stmt, 2, cid)
            ibm_db.bind_param(stmt, 3, userid)
            ibm_db.execute(stmt)
            sql = "update agents set status =3 where username=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1, userid)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from complaints where assigned_agent = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary = ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            complaints.append(dictionary)
            dictionary = ibm_db.fetch_assoc(stmt)
        # print(complaints)
        return render_template('agentdash.html', name=userid, complaints=complaints)
```

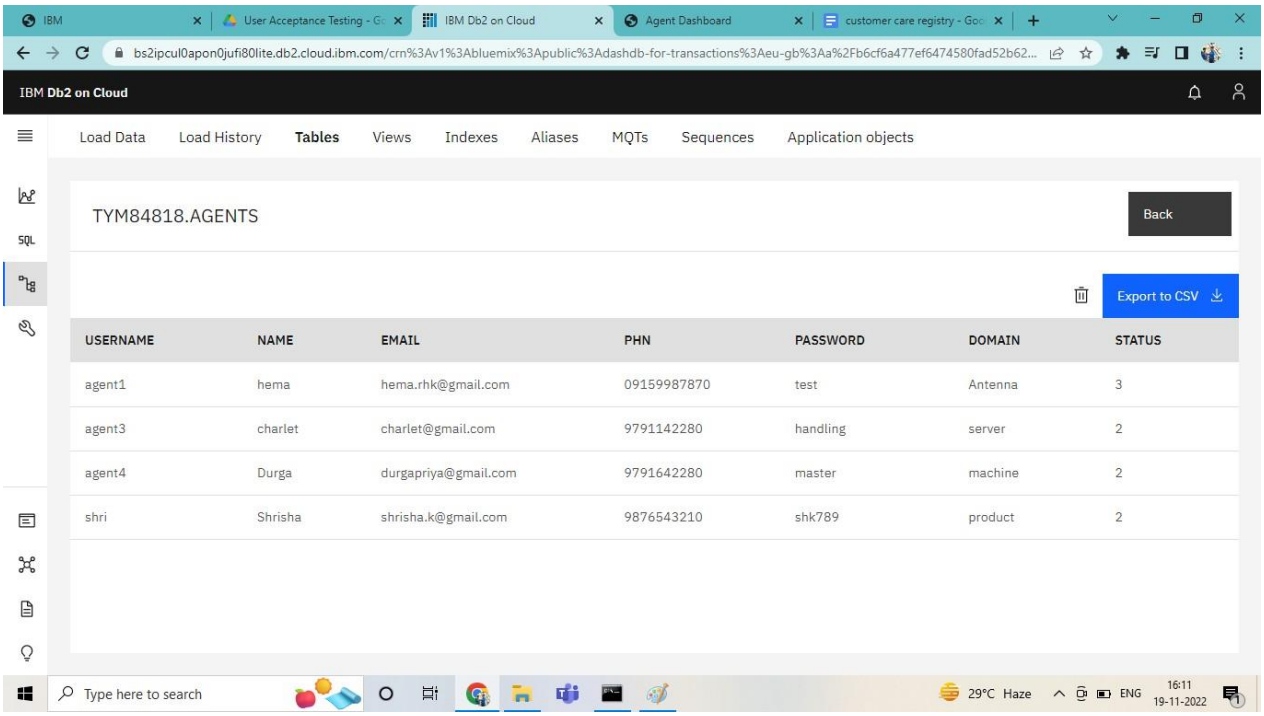
```python
@app.route('/tickets')
def tickets():
    sql = "select * from complaints"
    complaints = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        complaints.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)

    sql = "select username from agents where status <> 1"
    freeagents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        freeagents.append(dictionary)
        dictionary = ibm_db.fetch_assoc(stmt)
    print(freeagents)
    return render_template('tickets.html', complaints=complaints, freeagents=freeagents)
```

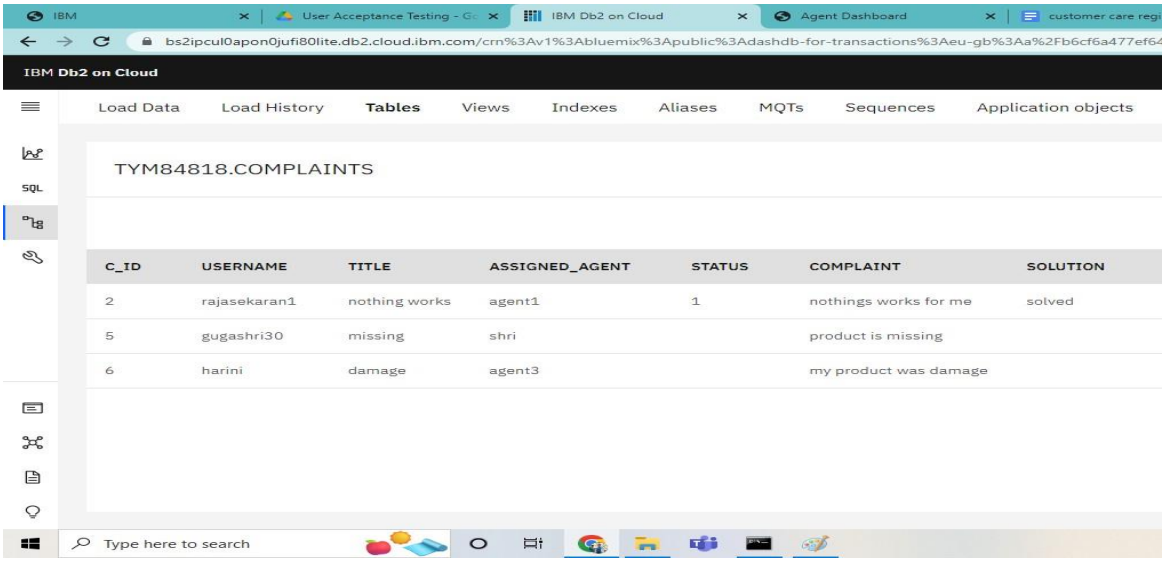
EXPLANATION:

- User creates a ticket by describing the query
- Admin views the newly created ticket in the dashboard• In the dropdown given, admin selects an agent
- Once selected, using fetch() the request is sent to the server
- The request URL contains both the Ticket ID and the selected Agent ID
- Using the shown SQL query, the assigned_to column of the tickets table is set to agent_idwhere the ticket_id column = ticket_id
- Then, the dashboard of the admin gets refreshed
- User creates a ticket by describing the query
- Admin assigns an agent to this ticket
- The customer and the agent, chat with each other, in the view of clearing the customer's doubts
- Once the customer is satisfied, the customer decides to close the ticket
- Using fetch() the request is sent to the server.
- The requested URL contains the Ticket ID
- Using the shown SQL query, the status of the ticket is set to "CLOSED"
- Thus the ticket is closed
- Then the customer gets redirected to the all-tickets page

Database schema



USERNAME	NAME	EMAIL	PHN	PASSWORD	DOMAIN	STATUS
agent1	hema	hema.rhk@gmail.com	09159987870	test	Antenna	3
agent3	charlet	charlet@gmail.com	9791142280	handling	server	2
agent4	Durga	durgapriya@gmail.com	9791642280	master	machine	2
shri	Shrisha	shrisha.k@gmail.com	9876543210	shk789	product	2



C_ID	USERNAME	TITLE	ASSIGNED_AGENT	STATUS	COMPLAINT	SOLUTION
2	rajasekaran1	nothing works	agent1	1	nothings works for me	solved
5	gugashri30	missing	shri		product is missing	
6	harini	damage	agent3		my product was damage	

IBM Db2 on Cloud

Load Data | Load History | **Tables** | Views | Indexes | Aliases | MQTs | Sequences | Application objects

TYM84818.USERS

Back

Export to CSV

USERNAME	NAME	EMAIL	PHN	PASSWORD
gugashri30	Gugashri T	gugashri.t@gmail.com	9791642280	kgr5678
harini	Harini	ilangoharini@gmail.com	7932985622	hari123
rajasekaran1	rajasekaran	g.rajasekaran@gmail.com	9894137134	Raja1

8. TESTING

8.1 TEST CASES TESTING

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios.Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not. Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases.To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

8.2 User Acceptance Testing

UAT Execution & Report Submission

Date	11 November 2022
Team ID	PNT2022TMID21762
Project Name	Project - Customer Care Registry
Maximum Marks	4 Marks

1. 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).

2. 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	4	2	3	20
Duplicate	1	0	3	0	4
External	2	3	0	1	6
Fixed	11	2	4	20	37
Not Reproduced	0	0	1	0	1
Skipped	0	0	1	1	2
Won't Fix	0	5	2	1	8
Totals	24	14	13	26	77

3. Test Case Analysis

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

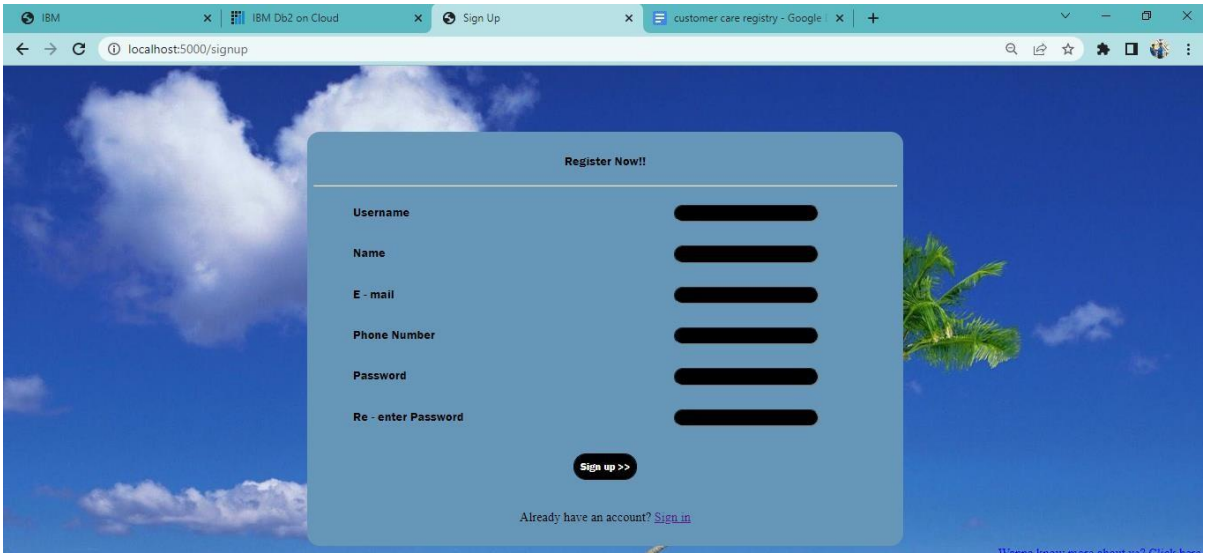
Section	Total Cases	Not Tested	Fail	Pass
Print Engine	7	0	0	7
Client Application	51	0	0	51
Security	2	0	0	2

Outsource Shipping	3	0	0	3
Exception Reporting	9	0	0	9
Final Report Output	4	0	0	4
Version Control	2	0	0	2

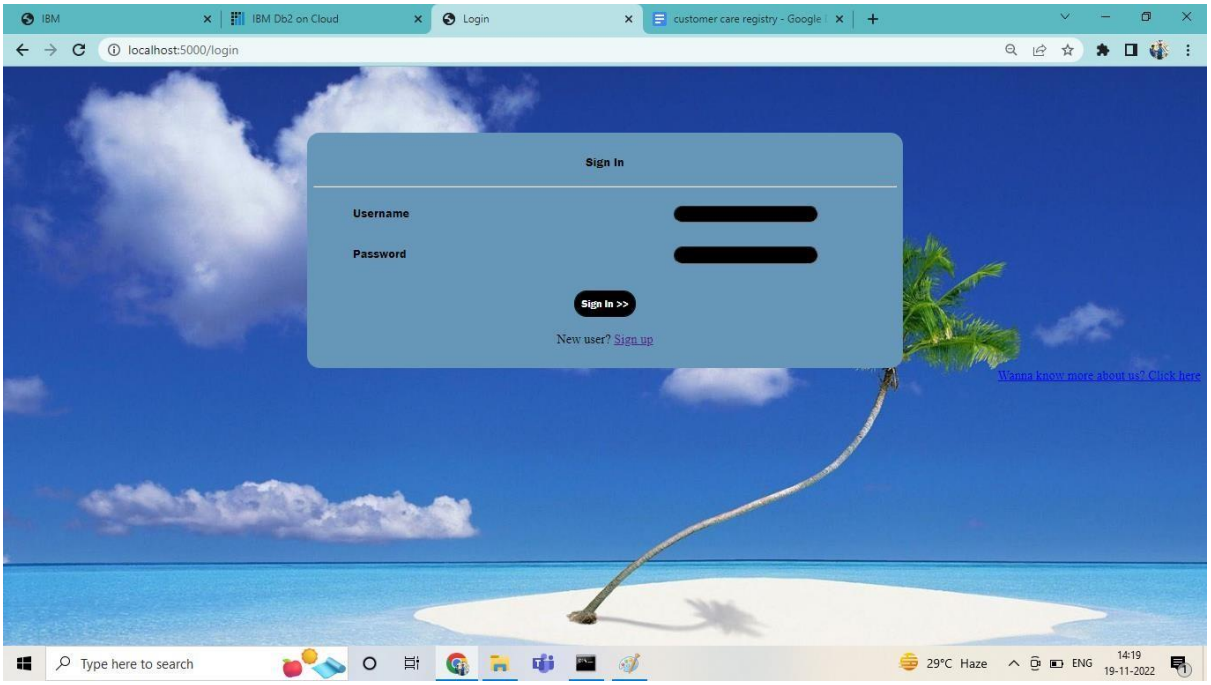
9. RESULT

9.1 Performance Metrics

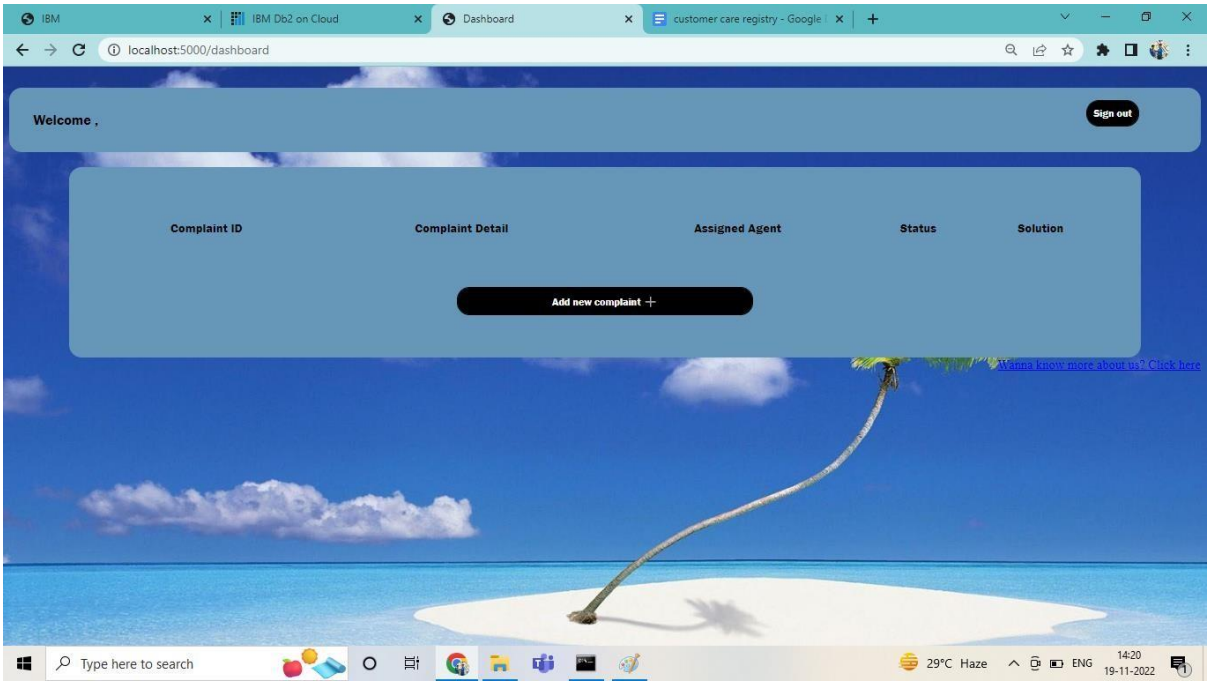
Register login or sign up



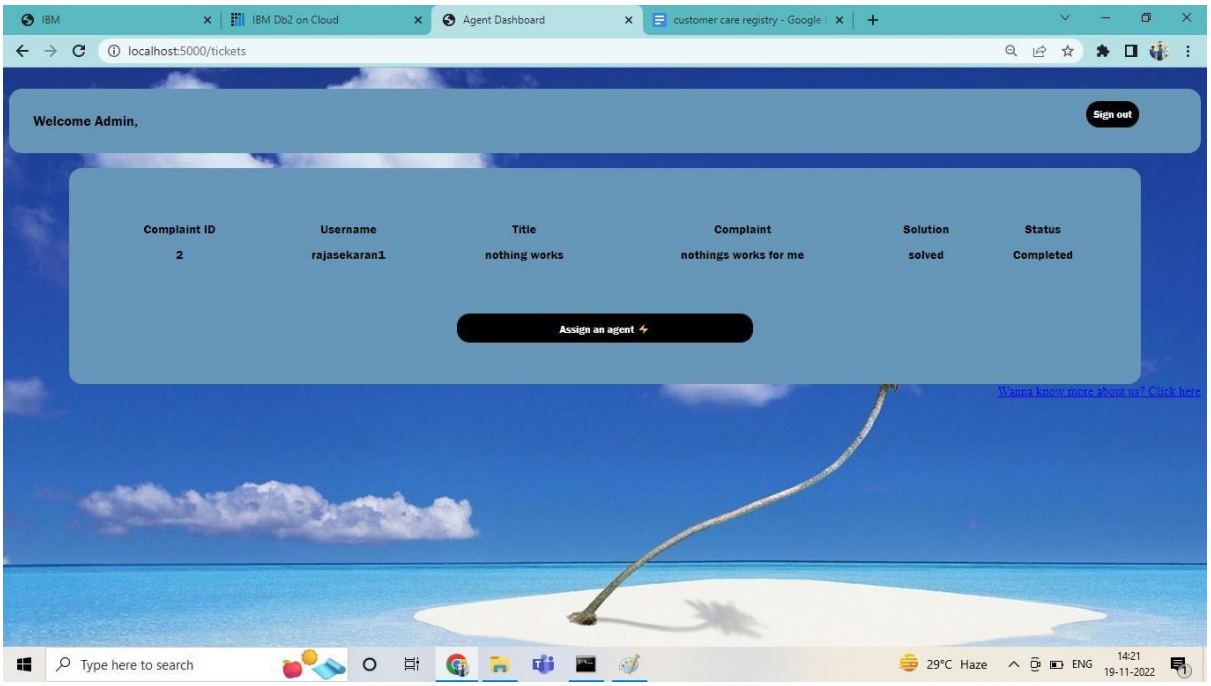
Signin up



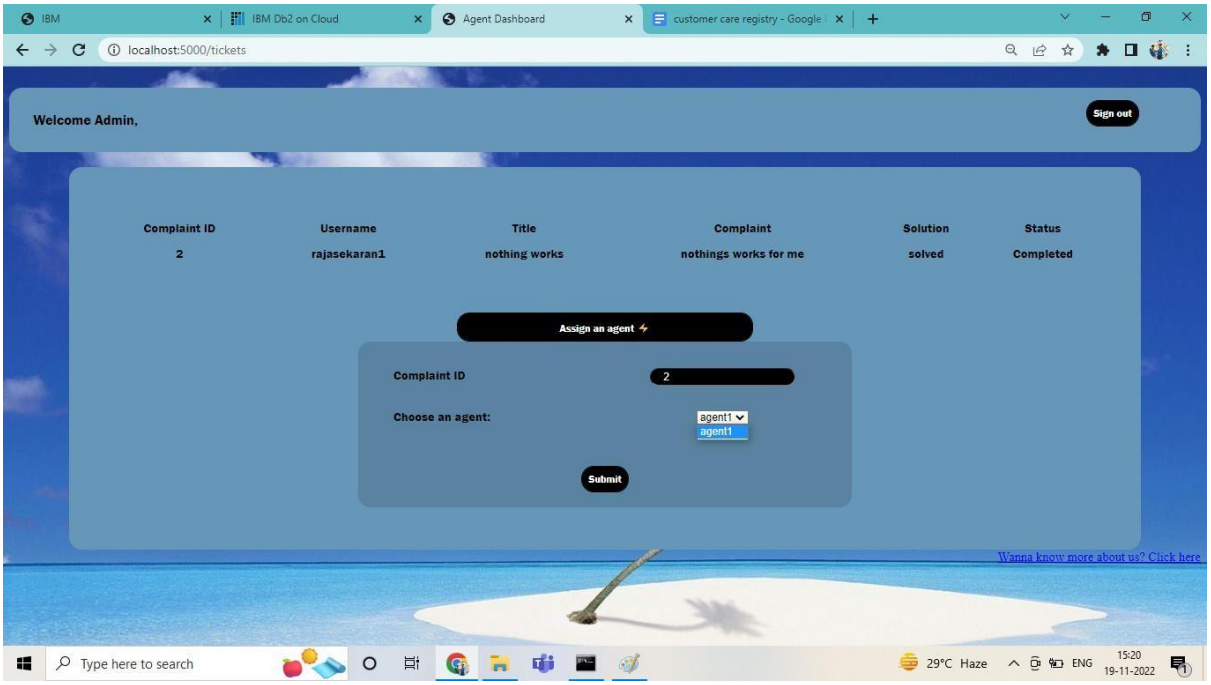
Dashboard



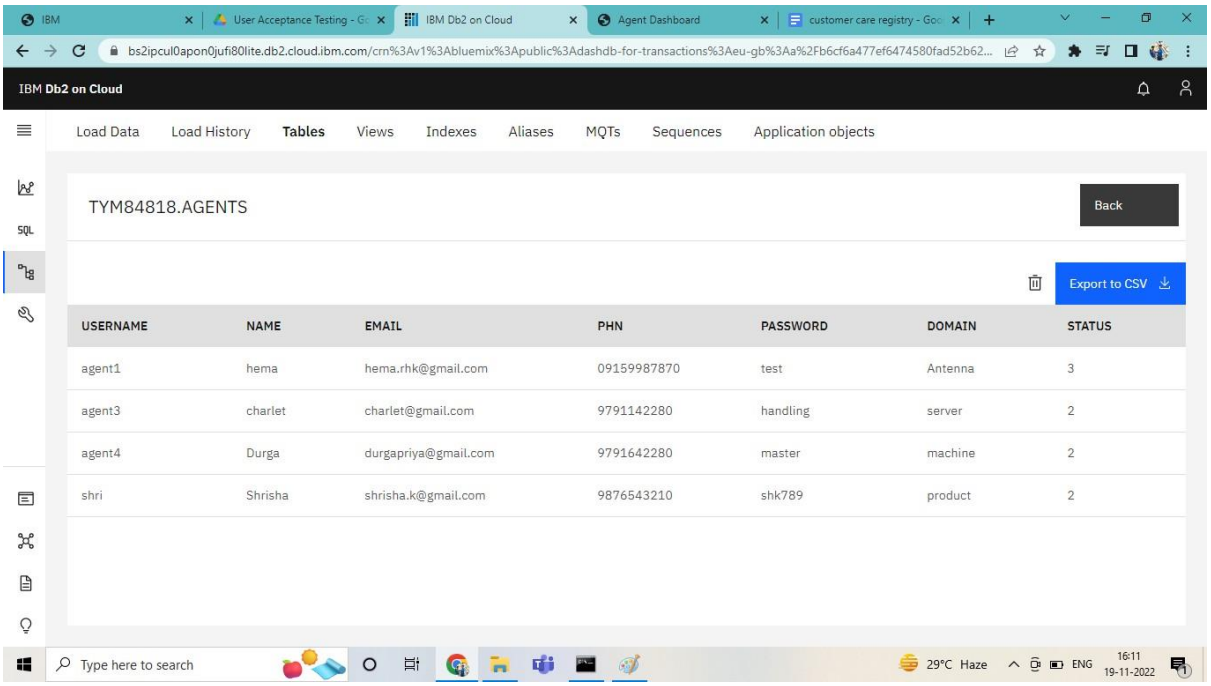
Agent dashboard



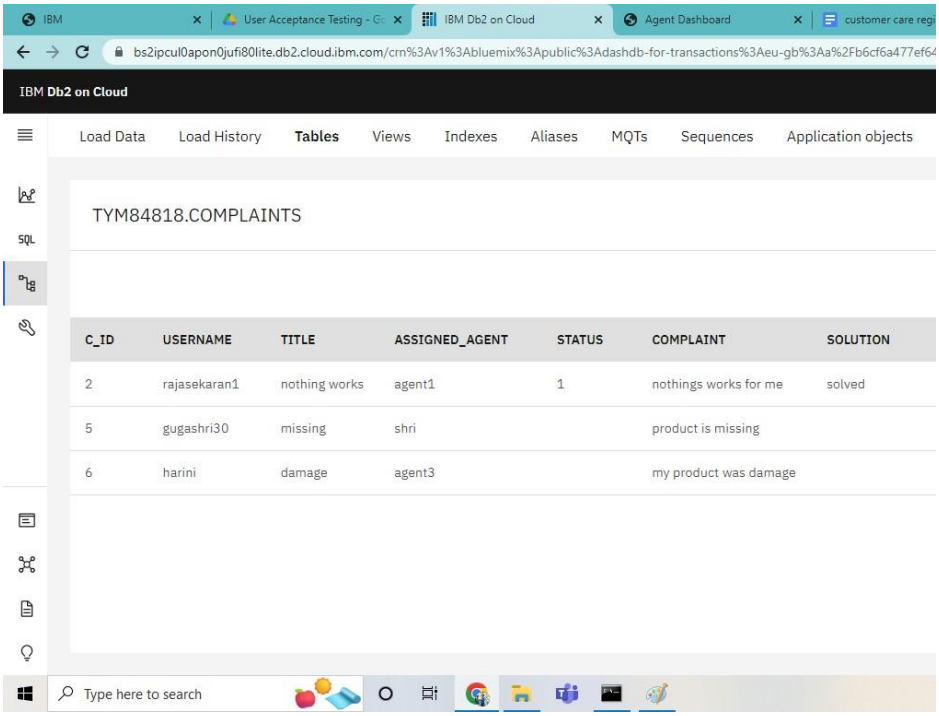
Agent assign



Agent Details

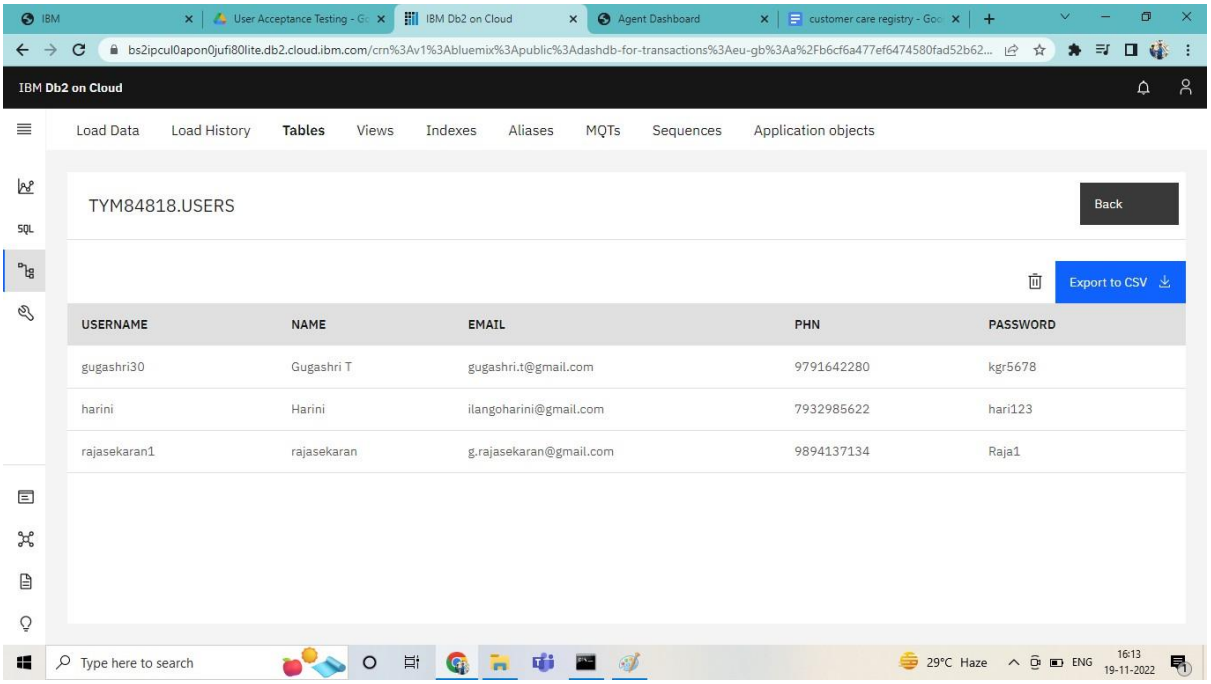


Complaint Details



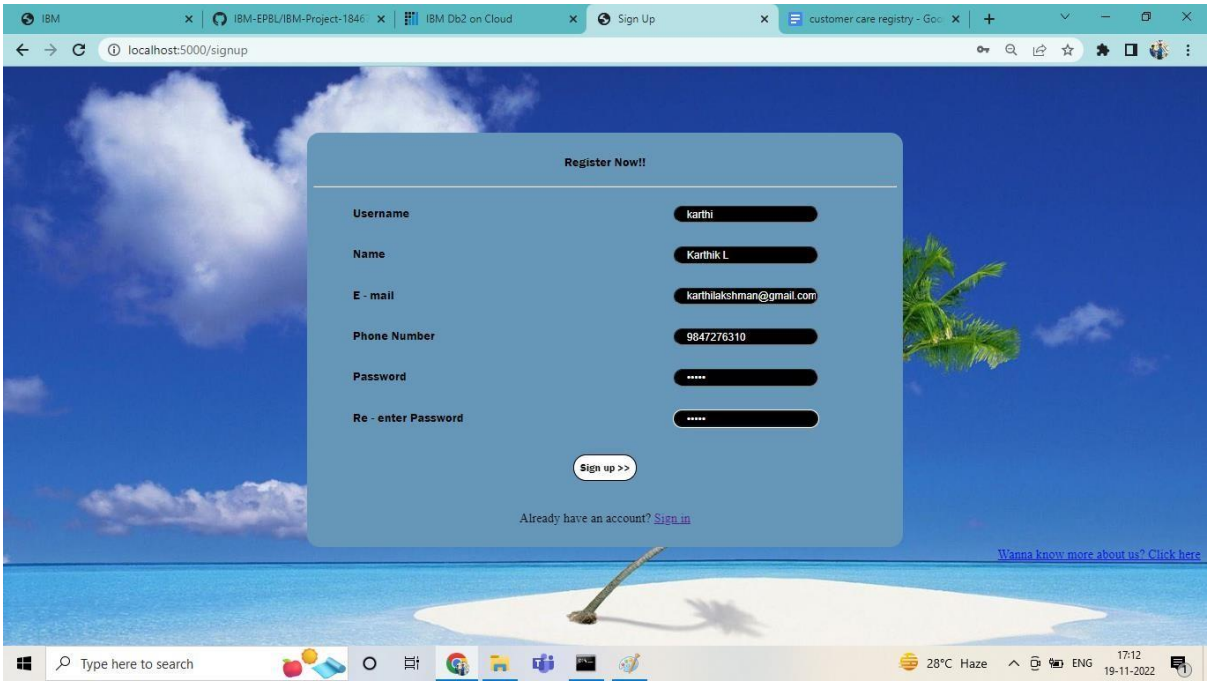
C_ID	USERNAME	TITLE	ASSIGNED_AGENT	STATUS	COMPLAINT	SOLUTION
2	rajasekaran1	nothing works	agent1	1	nothings works for me	solved
5	gugashri30	missing	shri		product is missing	
6	harini	damage	agent3		my product was damage	

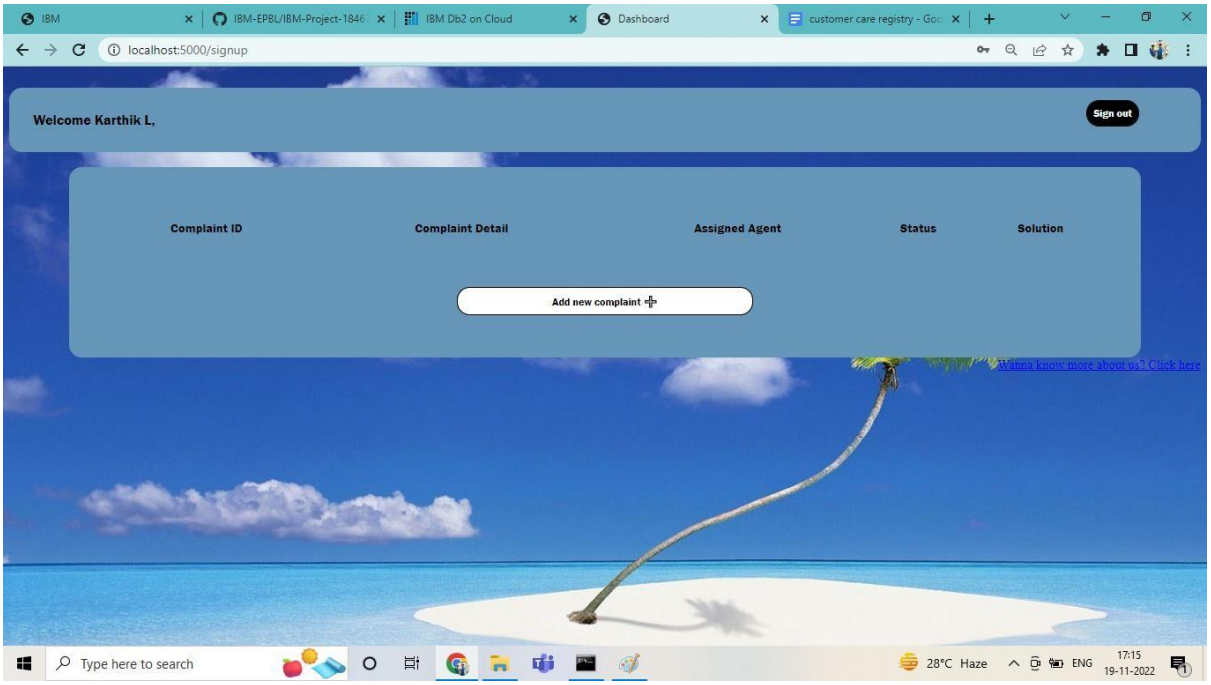
User Details

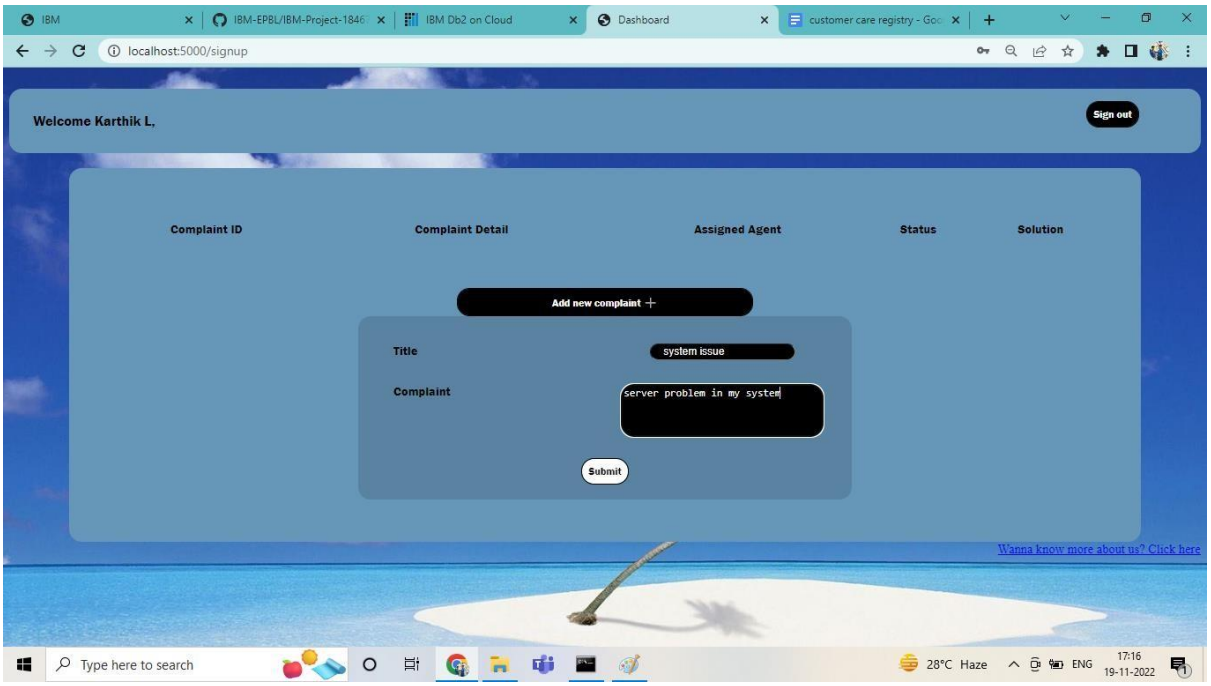


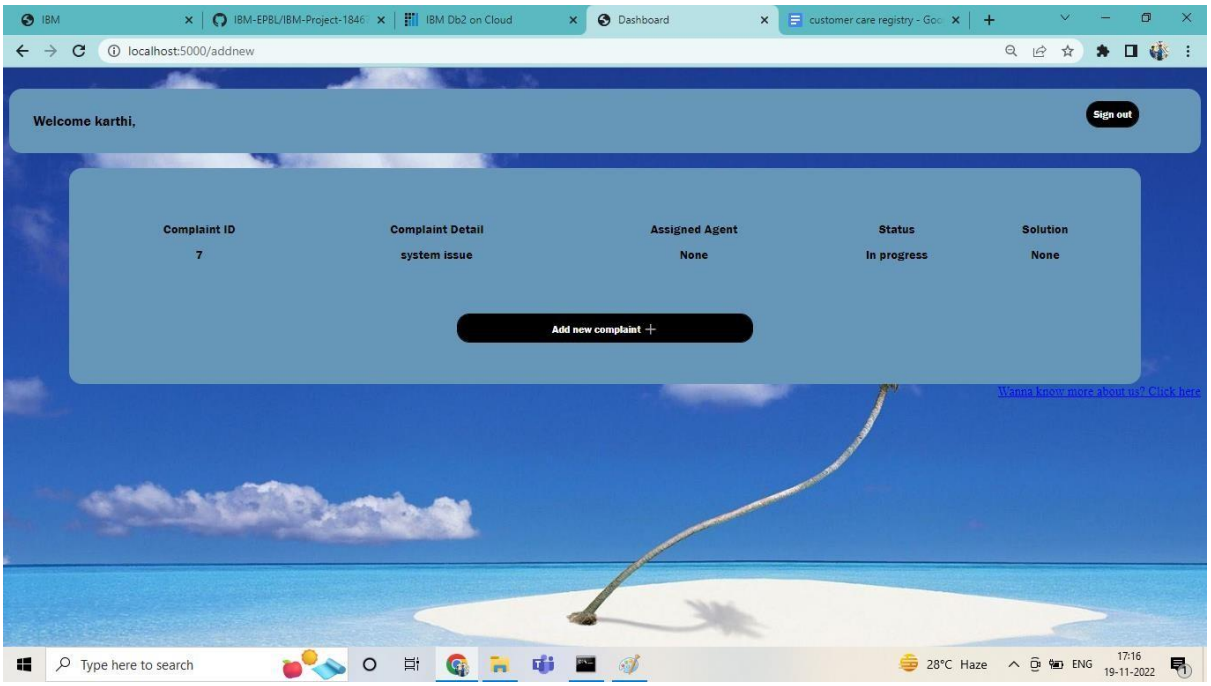
USERNAME	NAME	EMAIL	PHN	PASSWORD
gugashri30	Gugashri T	gugashri.t@gmail.com	9791642280	kgr5678
harini	Harini	ilangoharini@gmail.com	7932985622	hari123
rajasekaran1	rajasekaran	g.rajasekaran@gmail.com	9894137134	Raja1

Step by Step Process For Registration






IBM
IBM-EPBL/IBM-Project-1846
IBM Db2 on Cloud
Dashboard
customer care registry - Goo
localhost:5000/signup
Welcome Karthik L,
Sign out
Complaint ID
Complaint Detail
Assigned Agent
Status
Solution
Add new complaint +
Title
system issue
Complaint
server problem in my system
Submit
Wanna know more about us? Click here
Type here to search
28°C Haze
ENG
17:16
19-11-2022


IBM
IBM-EPBL/IBM-Project-1846
IBM Db2 on Cloud
Dashboard
customer care registry - Goo
localhost:5000/addnew
Welcome karthi,
Sign out
Complaint ID
Complaint Detail
Assigned Agent
Status
Solution
7
system issue
None
In progress
None
Add new complaint +
Wanna know more about us? Click here
Type here to search
28°C Haze
ENG
17:16
19-11-2022

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES

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

DISADVANTAGES

- Only web application is available right now (as of writing).
- UI is not so attractive, it's just simple looking.
- No automated replies.
- No SMS alerts .
- Supports only text messages while chatting with the Agent.
- No tap to reply feature.
- No login alerts.
- Cannot update the mobile number.
- Account cannot be deleted, once created.
- Customers cannot give feedback to the agent for clarifying the queries.

11. CONCLUSION

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

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

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

12. FUTURE SCOPE

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

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

13. APPENDIX

Flask:

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

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

JavaScript:

- JavaScript, often abbreviated as JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS.
- As of 2022, 98% of websites use JavaScript on the client side for webpage behavior, often incorporating third-party libraries.

IBM Cloud:

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

IBM Kubernetes:

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

Docker:

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

SOURCE CODE (Only Samples)

admin.html

```
{%
extends
'base.h
tml' %}



{% block head %}

<title>
    Admin Dashboard
</title>



{% endblock %}



{% block body %}


<!-- things
    div 1
welcome jetson,   sign out
    div 2
your complaints status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
    {{ j }}
{% endfor %}
```

```
<br>
{% endfor %} -->

<div class="fordashboardtop">
    <div class="fordashboardtopelements1">
        Welcome Admin,
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button class="forbutton">Sign
out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">

    <div class="fordashboarddetails">
        <br>
        <!-- table of customers complaints -->
        <table class="fortable">
            <thead>
            </thead>
            <tbody>
                <tr>
                    <td class="pad">
                        <a href="/agents">Agent
Details</a>
                    </td>
                    <td class="pad">
                        <a href="/tickets">Customer
Ticket Details</a>
                    </td>
                </tr>
            </tbody>

        </table>

        <br>

    </div>

</div>

{% endblock %}
```

Login.html

```
{%
extends
'base.h
tml' %}



          {% block head %}

          <title>
              Login
          </title>

          {% endblock %}



          {% block body %}


          <div class="forpadding">

              <!-- for box of the signup form -->
              <div class="sign">
                  <div>
                      <p class="fortitle">
                          Sign In
                      </p>
                      <hr>
                      <form action="/login"
          method="post">
                          <div class="forform">
                              <div
          class="textinformleft">
                                  Username
                              </div>
                              <div
          class="textinformright">
                                  <input type="name"
          name="username">
                              </div>
                          </div>

                          <div class="forform">
                              <div
          class="textinformleft">
```

```
                        Password
                    </div>
                    <div
class="textinformright">
                        <input
type="password" name="pass">
                    </div>
                </div>

                <br>
                <div>
                    <button
class="forbutton" type="submit"> Sign In
>></button>
                </div>
            </form>
            <br>

            <div>
                New user? <a
href="/signup">Sign up</a>
            </div>
            <br>
        </div>

    </div>
</div>

{% endblock %}
```

Signup.html

```
 {%
 extends
 'base.h
 tml' %}



           {% block head %}

           <title>
               Sign Up
           </title>
           {% endblock %}
```

```
{% block body %}

<div class="forpadding">

    <!-- for box of the signup
form -->
    <div class="sign">
        <div>
            <p class="fortitle">
                Register Now!!
            </p>
            <hr>
            <form
action="/signup" method="post">
                <div
class="forform">
                    <div
class="textinformleft">
                        Username
                    </div>
                    <div
class="textinformright">
                        <input
type="name" name="username">
                    </div>
                </div>
                <div
class="forform">
                    <div
class="textinformleft">
                        Name
                    </div>
                    <div
class="textinformright">
                        <input
type="name" name="name">
                    </div>
                </div>
                <div
class="forform">
                    <div
class="textinformleft">
                        E - mail
                    </div>
```

```
                    <div
class="textinformright">
                        <input
type="name" name="email">
                    </div>
                </div>
                <div
class="forform">
                    <div
class="textinformleft">
                        Phone
Number
                    </div>
                    <div
class="textinformright">
                        <input
type="name" name="phn">
                    </div>
                </div>
                <div
class="forform">
                    <div
class="textinformleft">
                        Password
                    </div>
                    <div
class="textinformright">
                        <input
type="password" name="pass">
                    </div>
                </div>
                <div
class="forform">
                    <div
class="textinformleft">
                        Re -
enter Password
                    </div>
                    <div
class="textinformright">
                        <input
type="password" name="repass">
                    </div>
                </div>
                <br>
                <div>
```

```
                    <button
class="forbutton" type="submit">
Sign up >></button>
                </div>
            </form>
            <br>
            <div>
                {{msg}}
            </div>
            <br>
            <div>
                Already have an
account? <a href="/login">Sign
in</a>
            </div>
            <br>

        </div>

    </div>
</div>

{% endblock %}
```

Agent.html

```
 {%
 extends
 'base.h
 tml' %}



           {% block head %}

           <title>
               Dashboard
           </title>



           {% endblock %}



           {% block body %}
```

```
<!-- things
    div 1
welcome jetson,
sign out
    div 2
your complaints
status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11)
%}
  {{ i }}
{% endfor %}
<br>
{% for i in
complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in
i.values() %}
    {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div
class="fordashboardto
p">
    <div
class="fordashboardto
pelements1">
        Welcome
Admin,
    </div>
    <div
class="fordashboardto
pelements2">
        <a
href="/login"><button
class="forbutton">Sig
n out</button></a>
    </div>

</div>
```

```
<br>
<div
class="outerofdashdet
ails">

    <div
class="fordashboardde
tails">
        <br>
        <!-- table of
customers complaints
-->
        <table
class="fortable">
            <thead>
                <th
class="pad">Name</th>

<th>Username</th>

<th>Email</th>

<th>Phone</th>

<th>Domain</th>

<th>Status</th>
            </thead>
            <tbody>
                {%
for i in agents %}
                <tr>

<td class="pad">

{{ i['NAME'] }}

</td>

<td class="pad">

{{ i['USERNAME'] }}

</td>

<td>
```

```
{{ i['EMAIL'] }}

</td>

<td>

{{ i['PHN'] }}

</td>

<td>

{{ i['DOMAIN'] }}

</td>

<td>

{% if i['STATUS'] ==
1 %}

Assigned to job

{% elif i['STATUS']
== 0 %}

not Available

{% else %}

Available

{% endif %}

</td>
                </tr>
                {%
endfor %}
            </tbody>

        </table>

        <br>
        <center>
```

```
            <div
class="fordashboardde
tails">


<button type="button"
class="collapsible">A
dd new agent
➕</button>
                <div
class="content">

<br>

<form
action="/addnewagent"
method="post">

<div class="forform">

<div
class="textinformleft
">

Username

</div>

<div
class="textinformrigh
t">

<input type="name"
name="username">

</div>

</div>

<div class="forform">

<div
class="textinformleft
">

Name
```

```
</div>

<div
class="textinformrigh
t">

<input type="name"
name="name">

</div>

</div>

<div class="forform">

<div
class="textinformleft
">

Email

</div>

<div
class="textinformrigh
t">

<input type="name"
name="email">

</div>

</div>

<div class="forform">

<div
class="textinformleft
">

Phone

</div>

<div
```

```
class="textinformrigh
t">

<input type="name"
name="phone">

</div>

</div>

<div class="forform">

<div
class="textinformleft
">

Domain

</div>

<div
class="textinformrigh
t">

<input type="name"
name="domain">

</div>

</div>

<div class="forform">

<div
class="textinformleft
">

Password

</div>

<div
class="textinformrigh
t">

<input
```

```
type="password"
name="password">

</div>

</div>


<br>

<br>

<div>

<button
class="forbutton"
type="submit"> Submit
</button>

</div>

</form>

<br>

</div>


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

</div>

{% endblock %}
```

Dashboard.html

```
{% extends
'base.html'
%}


                {% block head %}
```

```
<title>
    Dashboard
</title>



{% endblock %}



{% block body %}


<!-- things
    div 1
welcome jetson,   sign out
    div 2
your complaints status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
    {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
    <div class="fordashboardtopelements1">
        Welcome {{ name }},
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button
class="forbutton">Sign out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">
```

```
    <div class="fordashboarddetails">
        <br>
        <!-- table of customers complaints -->
        <table class="fortable">
            <thead>
                <th>Complaint ID</th>
                <th class="pad">Complaint
Detail</th>
                <th>Assigned Agent</th>
                <th>Status</th>
                <th>Solution</th>
            </thead>
            <tbody>
                {% for i in complaints %}
                <tr>
                    <td>
                        {{ i['C_ID'] }}
                    </td>
                    <td class="pad">
                        {{ i['TITLE'] }}
                    </td>
                    <td>
                        {{ i['ASSIGNED_AGENT'] }}
                    </td>
                    <td>
                        {% if i['STATUS'] == 1 %}
                        Completed
                        {% elif i['STATUS'] == 0 %}
                        Not completed
                        {% else %}
                        In progress
                        {% endif %}
                    </td>
                    <td>
                        {{ i['SOLUTION'] }}
                    </td>
                </tr>
                {% endfor %}
            </tbody>

        </table>

        <br>
        <center>
```

```
            <div class="fordashboarddetails">

                <button type="button"
class="collapsible">Add new complaint ➕</button>
                <div class="content">
                    <br>
                    <form action="/addnew"
method="post">
                        <div class="forform">
                            <div
class="textinformleft">
                                Title
                            </div>
                            <div
class="textinformright">
                                <input type="name"
name="title">
                            </div>
                        </div>

                        <div class="forform">
                            <div
class="textinformleft">
                                Complaint
                            </div>
                            <div
class="textinformright">
                                <textarea name="des"
                                    style="border-
radius: 1rem;width: 90%;height: 150%;background-
color: black;color: white;"></textarea>
                            </div>
                        </div>

                        <br>
                        <br>
                        <div>
                            <button
class="forbutton" type="submit"> Submit </button>
                        </div>
                    </form>
                    <br>
                </div>


            </div>
```

```
                    </center>
                </div>

            </div>

            {% endblock %}
```

Agentdash.html

```
{%
Ex
Te
Nd
S
'b
As
e.
Ht
Ml
'
%}



    {% block head %}

    <title>
        Agent Dashboard
    </title>



    {% endblock %}



    {% block body %}


    <!-- things
        div 1
    welcome jetson,   sign out
        div 2
    your complaints status
    add new complaint -->
    <br>
    <!-- <br>
    {% for i in range(11) %}
```

```
  {{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
    {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
    <div class="fordashboardtopelements1">
        Welcome {{ name }},
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button class="forbutton">Sign
out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">

    <div class="fordashboarddetails">
        <br>
        <!-- table of customers complaints -->
        <table class="fortable">
            <thead>
                <th>Complaint ID</th>
                <th class="pad">Username</th>
                <th>Title</th>
                <th>Complaint</th>
                <th>Solution</th>
                <th>Status</th>
            </thead>
            <tbody>
                {% for i in complaints %}
                <tr>
                    <td class="pad">
                        {{ i['C_ID'] }}
                    </td>
                    <td class="pad">
                        {{ i['USERNAME'] }}
                    </td>
```

```
                    <td>
                        {{ i['TITLE'] }}
                    </td>
                    <td>
                        {{ i['COMPLAINT'] }}
                    </td>
                    <td>
                        {{ i['SOLUTION'] }}
                    </td>
                    <td>
                        {% if i['STATUS'] == 1 %}
                        Completed
                        {% else %}
                        Not Completed
                        {% endif %}
                    </td>
                </tr>
                {% endfor %}
            </tbody>

        </table>

        <br>
        <center>

            <div class="fordashboarddetails">

                <button type="button"
class="collapsible">Solve an Issue ⚡ </button>
                <div class="content">
                    <br>
                    <form action="/updatecomplaint"
method="post">
                        <div class="forform">
                            <div class="textinformleft">
                                Complaint ID
                            </div>
                            <div class="textinformright">
                                <input type="name"
name="cid">
                            </div>
                        </div>
                        <div class="forform">
                            <div class="textinformleft">
                                Solution
                            </div>
```

```
                            <div class="textinformright">
                                <input type="text"
name="solution">
                            </div>
                        </div>

                        <br>
                        <br>
                        <div>
                            <button class="forbutton"
type="submit"> Submit </button>
                        </div>
                    </form>
                    <br>
                </div>


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

</div>

{% endblock %}
```

Ticket.html

```
{%
ex
te
nd
s
'b
as
e.
ht
ml
'
%}



      {% block head %}

      <title>
          Agent Dashboard
```

```
</title>



{% endblock %}



{% block body %}


<!-- things
    div 1
welcome jetson,   sign out
    div 2
your complaints status
add new complaint -->
<br>
<!-- <br>
{% for i in range(11) %}
  {{ i }}
{% endfor %}
<br>
{% for i in complaints %}
{{ i['USERNAME'] }}
<br>
{% for j in i.values() %}
    {{ j }}
{% endfor %}
<br>
{% endfor %} -->

<div class="fordashboardtop">
    <div class="fordashboardtopelements1">
        Welcome Admin,
    </div>
    <div class="fordashboardtopelements2">
        <a href="/login"><button
class="forbutton">Sign out</button></a>
    </div>

</div>
<br>
<div class="outerofdashdetails">

    <div class="fordashboarddetails">
```

```
<br>
<!-- table of customers complaints -->
<table class="fortable">
    <thead>
        <th>Complaint ID</th>
        <th class="pad">Username</th>
        <th>Title</th>
        <th>Complaint</th>
        <th>Solution</th>
        <th>Status</th>
    </thead>
    <tbody>
        {% for i in complaints %}
        <tr>
            <td>{{ i['C_ID'] }}</td>
            <td class="pad">
                {{ i['USERNAME'] }}
            </td>
            <td>
                {{ i['TITLE'] }}
            </td>
            <td>
                {{ i['COMPLAINT'] }}
            </td>
            <td>
                {{ i['SOLUTION'] }}
            </td>
            <td>
                {% if i['STATUS'] == 1 %}
                Completed
                {% else %}
                Not Completed
                {% endif %}
            </td>
        </tr>
        {% endfor %}
    </tbody>

</table>

<br>
<center>

    <div class="fordashboarddetails">
```

```
                <button type="button"
class="collapsible">Assign an agent  ⚡ </button>
                <div class="content">
                    <br>
                    <form action="/assignagent"
method="post">
                        <div class="forform">
                            <div
class="textinformleft">
                                Complaint ID
                            </div>
                            <div
class="textinformright">
                                <input type="name"
name="ccid">
                            </div>
                        </div>
                        <div class="forform">
                            <div
class="textinformleft">
                                <label
for="agent">Choose an agent:</label>
                            </div>
                            <div
class="textinformright">
                                <select name="agent"
id="agent">
                                    {% for i in
freeagents %}
                                    <option value={{
i['USERNAME'] }}>{{ i['USERNAME'] }}</option>
                                    {% endfor %}
                                </select>
                            </div>
                        </div>

                        <br>
                        <br>
                        <div>
                            <button class="forbutton"
type="submit"> Submit </button>
                        </div>
                    </form>
                    <br>
                </div>
```

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

</div>

{% endblock %}
```

App.py

```
from
flas
k
impo
rt
Flas
k,
rend
er_t
empl
ate,
requ
est,
redi
rect
,
sess
ion,
url_
for
      import ibm_db
      import logging
      import re

      app = Flask(_name_,
      template_folder='templates')
      app.config['EXPLAIN_TEMPLATE_LOADING'] =
      True
      app.logger.info('Info level log')
      #app = Flask(__name__)


      # for connection
      # conn= ""
```

```
app.secret_key = 'a'
print("Trying to connect...")
conn =
ibm_db.connect("DATABASE=bludb;HOSTNAME=21fe
cfd8-47b7-4937-840d-
d791d0218660.bs2io90l08kqb1od8lcg.databases.
appdomain.cloud;PORT=31864;SECURITY=SSL;SSLS
erverCertificate=DigiCertGlobalRootCA.crt;UI
D=tym84818;PWD=ewanJuj8SYcIupWC;", '', '')
print("connected..")


@app.route('/signup', methods=['GET',
'POST'])
def signup():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phn = request.form['phn']
        password = request.form['pass']
        repass = request.form['repass']
        print("inside checking")
        print(name)
        if len(username) == 0 or len(name)
== 0 or len(email) == 0 or len(phn) == 0 or
len(password) == 0 or len(repass) == 0:
            msg = "Form is not filled
completely!!"
            print(msg)
            return
render_template('signup.html', msg=msg)
        elif password != repass:
            msg = "Password is not matched"
            print(msg)
            return
render_template('signup.html', msg=msg)
        elif not re.match(r'[a-z]+',
username):
            msg = 'Username can contain only
small letters and numbers'
            print(msg)
            return
render_template('signup.html', msg=msg)
```

```
        elif not
re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = 'Invalid email'
            print(msg)
            return
render_template('signup.html', msg=msg)
        elif not re.match(r'[A-Za-z]+',
name):
            msg = "Enter valid name"
            print(msg)
            return
render_template('signup.html', msg=msg)
        elif not re.match(r'[0-9]+', phn):
            msg = "Enter valid phone number"
            print(msg)
            return
render_template('signup.html', msg=msg)

        sql = "select * from users where
username = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
            msg = 'Acccount already exists'
        else:
            userid = username
            insert_sql = "insert into users
values(?,?,?,?,?)"
            prep_stmt = ibm_db.prepare(conn,
insert_sql)
            ibm_db.bind_param(prep_stmt, 1,
username)
            ibm_db.bind_param(prep_stmt, 2,
name)
            ibm_db.bind_param(prep_stmt, 3,
email)
            ibm_db.bind_param(prep_stmt, 4,
phn)
            ibm_db.bind_param(prep_stmt, 5,
password)
            try:
                ibm_db.execute(prep_stmt)
                print("successs")
```

```
                msg = "succesfully signed
up"
                return
render_template('dashboard.html', msg=msg,
name=name)
            except Exception:
                            print("Procedure
failed with sqlstate
{}".format(ibm_db.stmt_error()))
                            print("Error
{}".format(ibm_db.stmt_errormsg()))
    else:
        return
render_template('signup.html')


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

@app.route('/')
def base():
    return redirect(url_for('login'))

@app.route('/login', methods=["GET",
"POST"])
def login():
    global userid
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        userid = username
        password = request.form['pass']
        if userid == 'admin' and password ==
'admin':
            print("its admin")
            return
render_template('admin.html')
        else:
            sql = "select * from agents
where username = ? and password = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
username)
            ibm_db.bind_param(stmt, 2,
password)
```

```
            ibm_db.execute(stmt)
            account =
ibm_db.fetch_assoc(stmt)
            print(account)
            if account:
                session['Loggedin'] = True
                session['id'] =
account['USERNAME']
                userid = account['USERNAME']
                session['username'] =
account['USERNAME']
                msg = 'logged in
successfully'

                # for getting complaints
details
                sql = "select * from
complaints where assigned_agent = ?"
                complaints = []
                stmt = ibm_db.prepare(conn,
sql)
                ibm_db.bind_param(stmt, 1,
username)
                ibm_db.execute(stmt)
                dictionary =
ibm_db.fetch_assoc(stmt)
                while dictionary != False:

complaints.append(dictionary)
                    dictionary =
ibm_db.fetch_assoc(stmt)
                print(complaints)
                return
render_template('agentdash.html',
name=account['USERNAME'],
complaints=complaints)

        sql = "select * from users where
username = ? and password = ?"
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, username)
        ibm_db.bind_param(stmt, 2, password)
        ibm_db.execute(stmt)
        account = ibm_db.fetch_assoc(stmt)
        print(account)
        if account:
```

```
            session['Loggedin'] = True
            session['id'] =
account['USERNAME']
            userid = account['USERNAME']
            session['username'] =
account['USERNAME']
            msg = 'logged in successfully'

            # for getting complaints details
            sql = "select * from complaints
where username = ?"
            complaints = []
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
username)
            ibm_db.execute(stmt)
            dictionary =
ibm_db.fetch_assoc(stmt)
            while dictionary != False:
                # print "The ID is : ",
dictionary["EMPNO"]
                # print "The Name is : ",
dictionary[1]

complaints.append(dictionary)
                dictionary =
ibm_db.fetch_assoc(stmt)

            print(complaints)
            return
render_template('dashboard.html',
name=account['USERNAME'],
complaints=complaints)
        else:
            msg = 'Incorrect user
credentials'
            return
render_template('dashboard.html', msg=msg)
    else:
        return render_template('login.html')


@app.route('/addnew', methods=["GET",
"POST"])
def add():
    if request.method == 'POST':
```

```
        title = request.form['title']
        des = request.form['des']
        try:
            sql = "insert into
complaints(username,title,complaint)
values(?,?,?)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
userid)
            ibm_db.bind_param(stmt, 2,
title)
            ibm_db.bind_param(stmt, 3, des)
            ibm_db.execute(stmt)
        except:
            print(userid)
            print(title)
            print(des)
            print("cant insert")
        sql = "select * from complaints
where username = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary =
ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            # print "The ID is : ",
dictionary["EMPNO"]
            # print "The Name is : ",
dictionary[1]
            complaints.append(dictionary)
            dictionary =
ibm_db.fetch_assoc(stmt)
        print(complaints)
        return
render_template('dashboard.html',
name=userid, complaints=complaints)


@app.route('/agents')
def agents():
    sql = "select * from agents"
    agents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
```

```
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        agents.append(dictionary)
        dictionary =
ibm_db.fetch_assoc(stmt)
    return render_template('agents.html',
agents=agents)


@app.route('/addnewagent', methods=["GET",
"POST"])
def addagent():
    if request.method == 'POST':
        username = request.form['username']
        name = request.form['name']
        email = request.form['email']
        phone = request.form['phone']
        domain = request.form['domain']
        password = request.form['password']
        try:
            sql = "insert into agents
values(?,?,?,?,?,?,2)"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
username)
            ibm_db.bind_param(stmt, 2, name)
            ibm_db.bind_param(stmt, 3,
email)
            ibm_db.bind_param(stmt, 4,
phone)
            ibm_db.bind_param(stmt, 5,
password)
            ibm_db.bind_param(stmt, 6,
domain)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from agents"
        agents = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.execute(stmt)
        dictionary =
ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            agents.append(dictionary)
```

```
            dictionary =
ibm_db.fetch_assoc(stmt)

        return
render_template('agents.html',
agents=agents)


@app.route('/updatecomplaint',
methods=["GET", "POST"])
def updatecomplaint():
    if request.method == 'POST':
        cid = request.form['cid']
        solution = request.form['solution']
        try:
            sql = "update complaints set
solution =?,status=1 where c_id = ? and
assigned_agent=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
solution)
            ibm_db.bind_param(stmt, 2, cid)
            ibm_db.bind_param(stmt, 3,
userid)
            ibm_db.execute(stmt)
            sql = "update agents set status
=3 where username=?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
userid)
            ibm_db.execute(stmt)
        except:
            print("cant insert")
        sql = "select * from complaints
where assigned_agent = ?"
        complaints = []
        stmt = ibm_db.prepare(conn, sql)
        ibm_db.bind_param(stmt, 1, userid)
        ibm_db.execute(stmt)
        dictionary =
ibm_db.fetch_assoc(stmt)
        while dictionary != False:
            complaints.append(dictionary)
            dictionary =
ibm_db.fetch_assoc(stmt)
        # print(complaints)
```

```
        return
render_template('agentdash.html',
name=userid, complaints=complaints)


@app.route('/tickets')
def tickets():
    sql = "select * from complaints"
    complaints = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        complaints.append(dictionary)
        dictionary =
ibm_db.fetch_assoc(stmt)

    sql = "select username from agents where
status <> 1"
    freeagents = []
    stmt = ibm_db.prepare(conn, sql)
    ibm_db.execute(stmt)
    dictionary = ibm_db.fetch_assoc(stmt)
    while dictionary != False:
        freeagents.append(dictionary)
        dictionary =
ibm_db.fetch_assoc(stmt)
    print(freeagents)
    return render_template('tickets.html',
complaints=complaints,
freeagents=freeagents)


@app.route('/assignagent', methods=['GET',
'POST'])
def assignagent():
    if request.method == "POST":
        ccid = request.form['ccid']
        agent = request.form['agent']
        print(ccid)
        print(agent)
        try:
            sql = "update complaints set
assigned_agent =? where c_id = ?"
            stmt = ibm_db.prepare(conn, sql)
```

```
            ibm_db.bind_param(stmt, 1,
agent)
            ibm_db.bind_param(stmt, 2, ccid)
            ibm_db.execute(stmt)
            sql = "update agents set status
=1 where username = ?"
            stmt = ibm_db.prepare(conn, sql)
            ibm_db.bind_param(stmt, 1,
userid)
            ibm_db.execute(stmt)
        except:
            print("cant update")
        return redirect(url_for('tickets'))

if __name___== "__main__":
    app.run(debug=True)
```

GITHUB AND PROJECT DEMO LINK

Github Rep Link

https://github.com/IBM-EPBL/IBM-Project-18467-1668680564

Project Demo Link

https://youtu.be/Q316pDOo5Tk