IBM NALAIYATHIRAN PROJECT REPORT

Title: Customer Care Registry

Submitted by

Team ID: PNT2022TMID37124

Team Members:

TEAM LEADER :HARIPRASAD
TEAM MEMBER:HARISHVAREN
TEAM MEMBER:GURUPRADEEP
TEAM MEMBER:BHUVANESH KUMAR
TEAM MEMBER:GNANESH

Industry Mentor(s) Name : Vasudeva Hanush
Faculty Mentor(s) Name : A.S. Balaji

Project Overview

1. INTRODUCTION

A comprehensive online Client Care Solution is used to managecustomer interactions and complaints with Service Providers via phone or e- mail. The system must be able to integrate with any service provider from anyfield or sector, including banking, telecommunications, insurance, etc. Theprovision of service to consumers, commonly referred to as client service, has varying significance depending on the product, business, and domain. When making a purchase of a service rather than a product, customer service is frequently of greater importance. People or Sales & Service Representatives may offer customer service. The customer value proposition of a business typically includes excellent customer service.

1.1 PROJECT OVERVIEW

The Customer Service Desk project is online. The provision of service to customers is known as customer service or client service. The importance varies depending on the product, industry, and domain. When information relates to a service rather than a customer, customer service is frequently more crucial. A service representative might offer customer service. The customer value proposition of a business typically includes excellent customerservice. Software like Flask, Docker, SendGrid, and IBM Watson are used to implement this.

1.2 PURPOSE

The project's goal is to create customer engagement, address customer issues, and offer a helpful service. It is an essential component of all businesses.

2.LITERATURE SURVEY

2.1 EXISTING PROBLEM

The information is now saved in disc drives in the form of excel sheets in a semi-automated approach. Only the mailing feature is used to share information with Volunteers, Group members, etc. In this system, information maintenance and storage are increasingly crucial. It is a hard task to monitor the members' activities and the development of the work here. This system is unable to offer information exchange every day of the week.

2.2 REFERENCES

a) In this article, a Chabot from the AWS cloud is deployed for customer care. Real world smart Chabot for customer care using SaaS architecture. This is done to offer cognitive and LUIS services to humans.

b) Chat bots for customer service are used in this paper in place of human customer service representatives. It makes decisions and offers services using AI.

c) Client service chatbot - In this paper, the customer gives the chatbot the information it needs based on the information it gives the customer service.

d) A clever cloud-based customer relationship management system that uses adjustable pricing to maintain customers. This essay analyses historical patterns to suggest consumer behavior that might be used for marketing.

2.3 PROBLEM STATEMENT DEFINITION

A problem statement is a concise description of the problem or issues a project seeks to address. The problem statement identifies the current state, the desired future state and any gaps between the two. A problem statement is an important communication tool that can help ensure everyone working on a project knows what the problem they need to address is and why the project is important.

3.IDEATION & PROPOSED SOLUTION

3.1 EMPATHY MAP CANVAS

An empathy map is a straightforward, simple-to-understand picture that summarizes information about a user's actions and views. It is a helpful tool that enables teams to comprehend their users more fully. It's important to comprehend both the actual issue and the person who is experiencing it in order to develop a workable solution. Participants learn to think about situations from the user's perspective, including goals and challenges, through the exercise of creating the map.

Fig 3.1: Empathy Map Canvas

3.2 IDEATION AND BRAINSTORMING

1. What issues are you attempting to address? Describe the issues.

2. How might we address the issue? Which does the customer raise?

Brainstorming:

Group Ideas:


Respond on Social Media
Improve the strategic value of the contact centre
Increase customer satisfication
Automated ticket tagging
Provide Fast, Convenient Customer Support
Provide Self-Help Options
Offer 24/7 Customer Support
Use Non-Generic Auto-Replies
Implement self-serve
Offer Omnichannel Support
Provide a Trial Period
Boost agent experience
Make Feedback Part of Your Brand
Channel Expansion
Make an Irresistible Offer
Contribute to business growth
Drive revenue from support
Reward Loyal Customers
Address Your Customers by Name
Offer a Lenient, Straightforward Refund Policy

3.3 PROPOSED SOLUTION

Allotted By sending the email directly to a specific agent about the problem, agent routing can be fixed. automated ticket closure by cloud database sync. The customer's status display may include their tickets. The platform that will enable the customer specialist to be effective is what the customer care service aims to give. And it takes less time to find the answer.

4.REQUIREMENT ANALYSIS

4.1 FUNCTIONAL REQUIREMENTS

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	Customer Query	Access through email and chatbot from the chosen website
FR-4	Database	preserving the modelled item
FR-5	Feedback	Customer's Feedback
FR-6	E-Mail	Login alertness

NON-FUNCTIONAL REQUIREMENTS

FR No.	Non-Functional Requirement	Description
NFR-1	Usability	To offer a remedy for the issue, user friendly.
NFR-2	Security	Logging and authentication history
NFR-3	Reliability	Tracking the status of the decade via email
NFR-4	Performance	responsive and adaptable
NFR-5	Availability	24/7 Support
NFR-6	Scalability	Scalability of agents according to consumer volume

5.PROJECT DESIGN

5.1DATA FLOW DIAGRAM

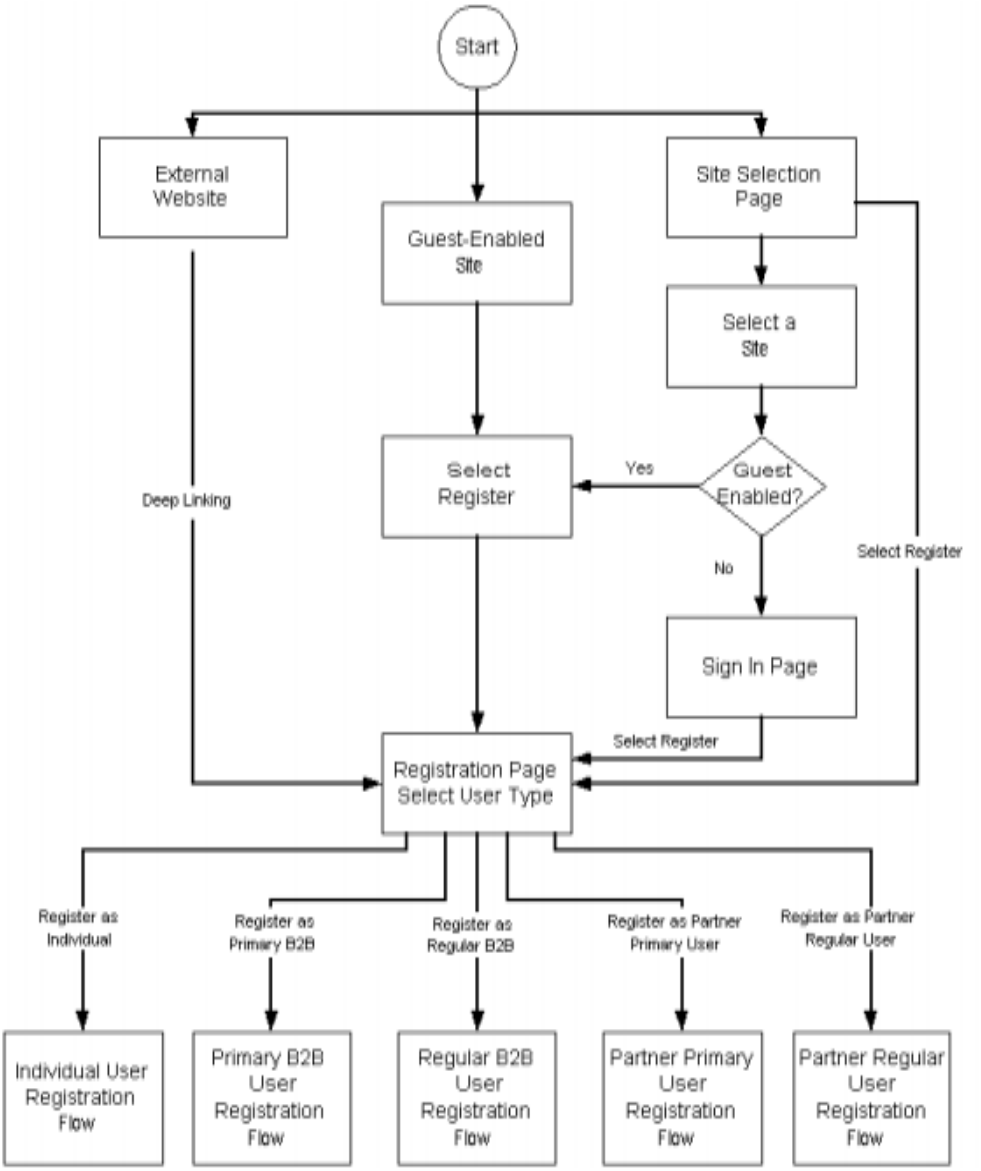


Fig 5.1: Data Flow Diagram

5.2SOLUTION AND TECHNICAL ARCHITECTURE

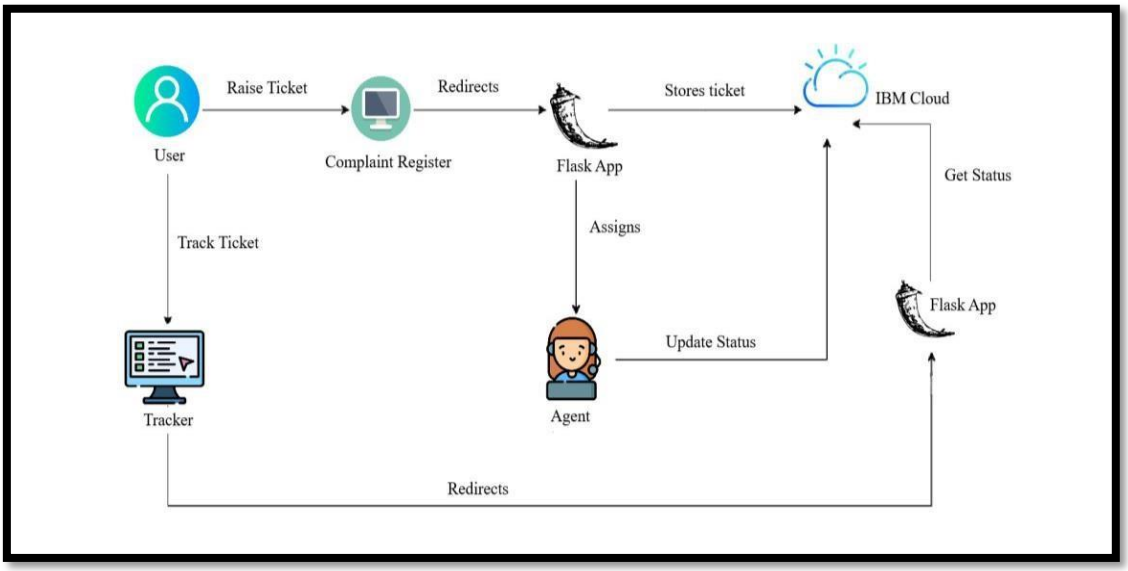


Fig 5.2: Solution and Technical Architecture

5.3USER STORIES

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

Admin (Mobile user)	Login	USN-1	As a admin, I can login to the appliaction by entering Correct email and password	I can access my account/dashboard	High	Sprint-1
	Dashboard	USN-2	As an admin I can see all the orders raised in the entire system and lot more	I can assign agentsby seeing those order.	High	Sprint-1
	Agent creation	USN-3	As an admin I can createan agent for clarifying the customers queries	I can create agents.	High	Sprint-2
	Assignment agent	USN-4	As an admin I can assignan agent for each order created by the customer.	Enable agent to clarify the queries.	High	Sprint-1
	Forgot password	USN-5	As an admin I can reset my password by this option in case I forgot my old password.	I get access to my account.	High	Sprint-1

6.PROJECT PLANNING & SCHEDULING

6.1SPRINT PLANNING & ESTIMATION

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	User Registration	USN-1	As a customer, I can register for the application by entering my email, password, and confirming my password.	5	High	Bhuvaneshkumar
	User Login	USN-2	As a customer, I can login to the application by entering correct email and password .	5	Medium	N.Hariprasad
	Admin login	USN-4	As an admin, I can login to the application by entering correct email and password .	10	High	Harishvaran M
Sprint-2	User Dashboard	USN-3	As a customer, I can see all the tickets raised by me and lot more .	10	Medium	Gurupradeep

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-2	Admin Dashboard	USN-5	As an admin, I can see all the tickets raised in the entire system and lot more	10	High	Gnanesh G
Sprint-3	User Ticket creation	USN-6	As a customer, I can create a new ticket with the detailed description of my query	10	High	Bhuvaneshkumar
	Assigning agent	USN-7	As an admin, I can assign an agent for each ticket created by the customer.	10	HIgh	Gurupradeep
Sprint-4	User Forgot password	USN-10	As a customer, I can reset my password by this option in case I forgot my old password provided by the user.	5	High	Harishvaran M
	Ticket details	USN-12	As a customer, I can see the current status of my tickets .	5	Medium	Gnanesh G
	Final deliverables	USN-13	Screen short of the project	10	High	N.Hariprasad

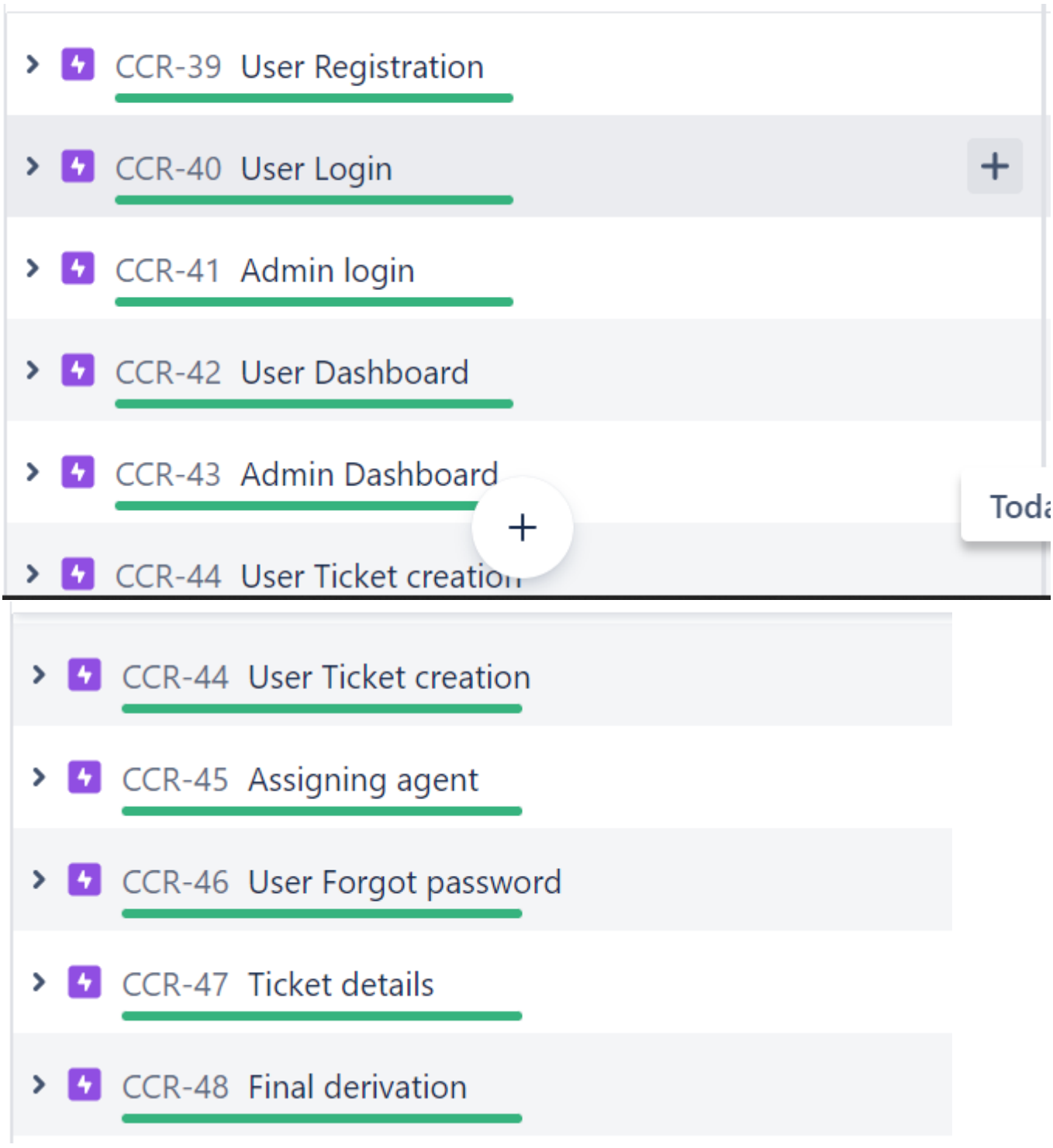
Fig 6.1: Sprint Planning& Estimation

6.2SPRINT DELIVERY SCHEDULE

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	6 Days	24 Oct 2022	29 Oct 2022	20	29 Oct 2022
Sprint-2	20	6 Days	31 Oct 2022	05 Nov 2022		
Sprint-3	20	6 Days	07 Nov 2022	12 Nov 2022		
Sprint-4	20	6 Days	14 Nov 2022	19 Nov 2022		

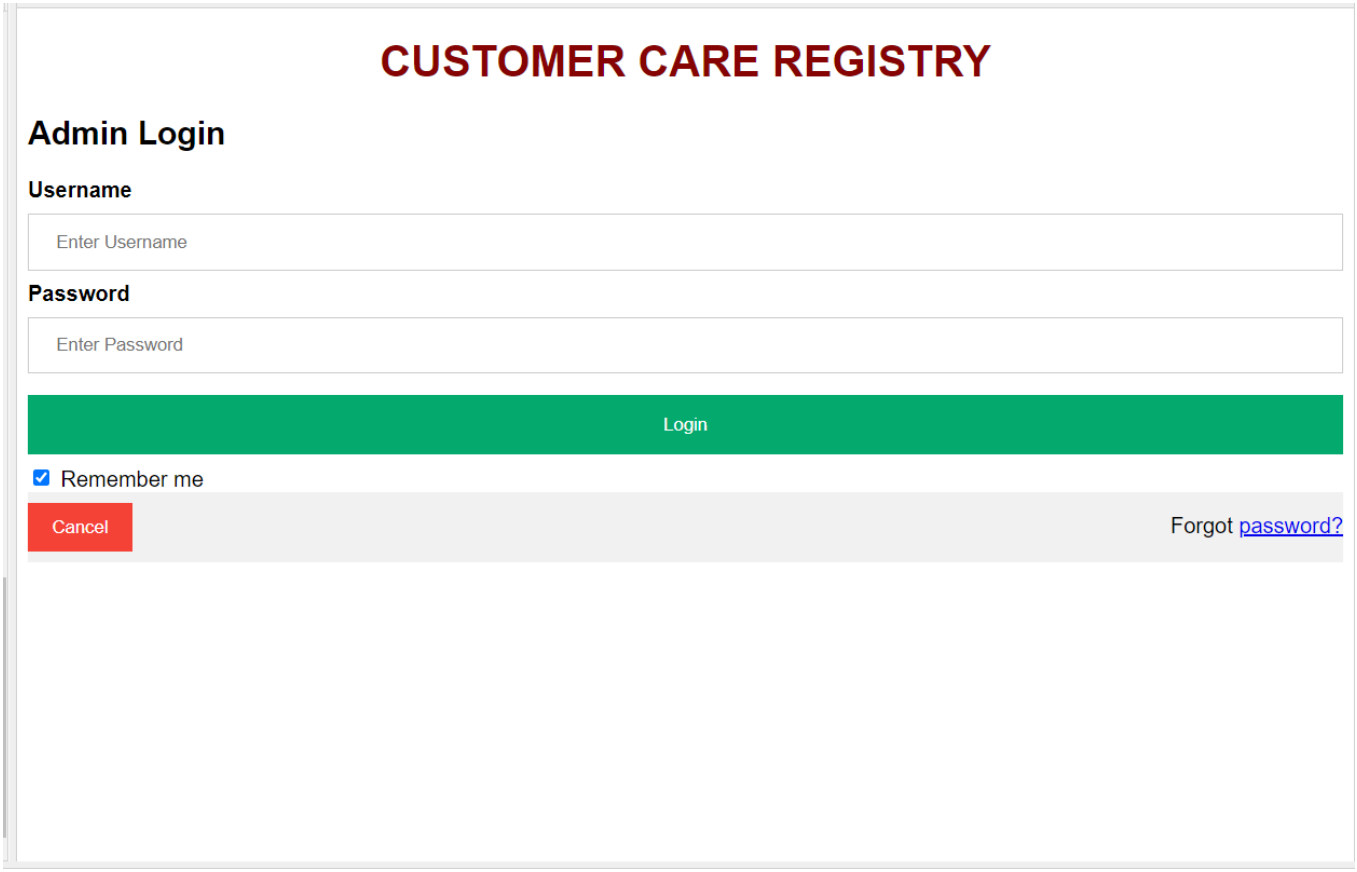
Fig 6.2: Sprint Delivery Schedule

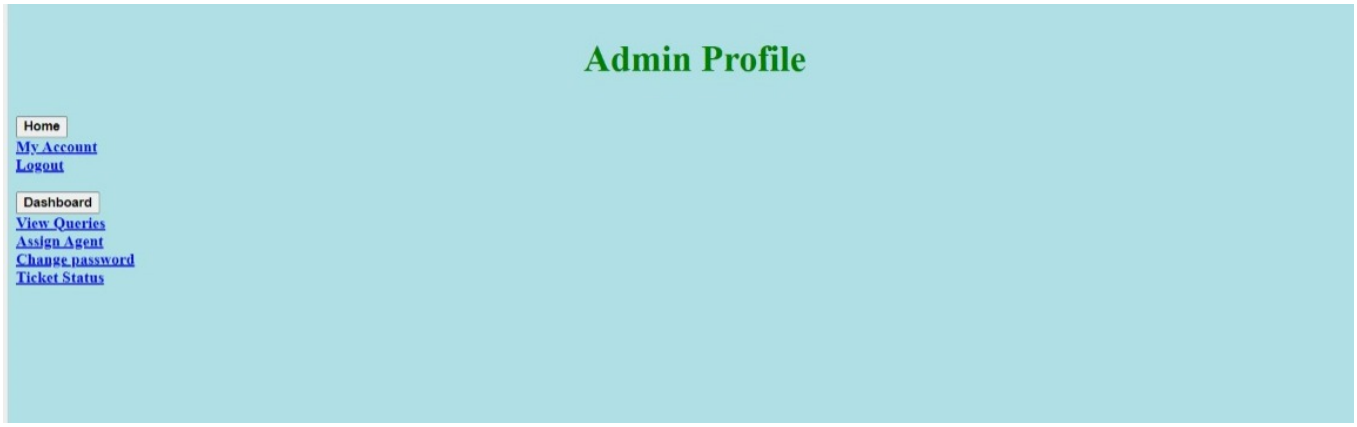
6.3REPORTS FROM JIRA



7.CODING & SOLUTIONING

7.1FEATURE 1





Admin Page

Assigning Agent

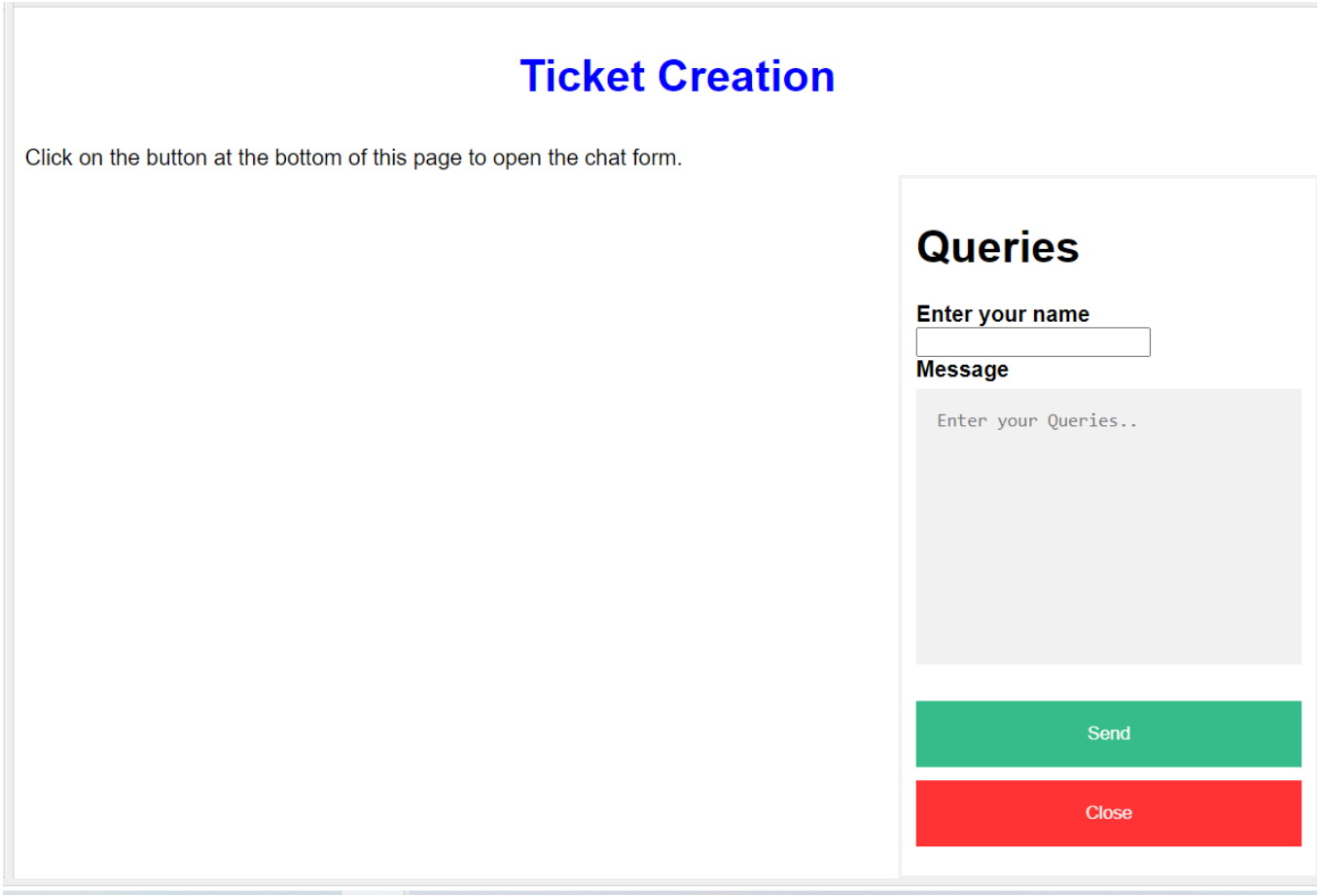
Agent Name	
Bhuvanesh	Assign Query
Harish	Assign Query
Guru Pratheep	Assign Query
Gnanesh	Assign Query
Hari Prasad	Assign Query

Change password

New Password:
Confirm Password:

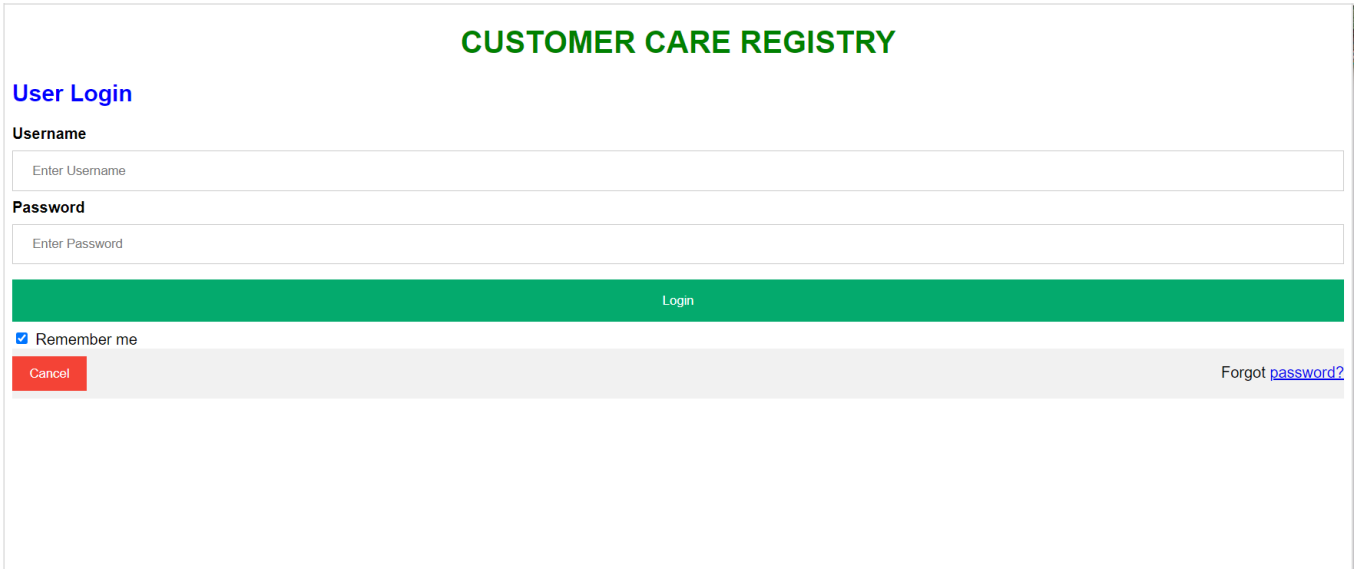
Change Password

7.2FEATURE 2



User Page

Ticket Status
Received Query
In Progress
Issues Solved


CUSTOMER CARE REGISTRY
User Login
Username
Enter Username
Password
Enter Password
Login
Remember me
Cancel
Forgot password?

User profile

Home
My Queries
My Account
Chatbot

Dashboard
Register
Login
Create Ticket
Change password
Ticket Status
Logout

Job Calendar:

When a user chooses a day from the job calendar, a list of jobs that are open on that day is displayed on the same page. Ajax features were used to construct this functionality from scratch.

8.TESTING

8.1TEST CASES

Test Case ID	Feature Type	Component	Test Scenario	Expected Result	Result Status
01	Functional	user page	**Verify user is able to see show complain popup**	**Show complain popup displayed**	**Pass**
02	**UI**	user page	**Verify user has no complain**	**No complain displayed**	**Pass**
03	**UI**	**AGENT login**	**Text field visible to enter email**	**Text field visible**	**Pass**
04	**UI**	**User login**	**Text field visible to enter email**	**Text field visible**	**Pass**
05	**UI**	**Agent login**	**Text field visible to enter password**	**Text field visible**	**Pass**

8.2 USER ACCEPTANCE TESTING

Test Case ID	Feature Type	Component	Test Scenario	Expected Result	Result Status
01	**Functional**	**Home page**	**Verify user is able to see the login/signup popup when clicked on my account**	**Login/Signup displayed**	**pass**
02	**Ui**	**Home page**	**Verify the UI elements in the Login/Signup**	**The UI elements Working accordingly**	**pass**
03	**Functional**	**Home page**	**Verify user is able to login to the application with valid credentials**	**Login successful**	**pass**
04	**Functional**	**login page**	**Verify user cannot login to the application without valid credentials**	**Login unsuccessful**	**pass**

9.RESULT

This project is designed to solve the customer queries and achieve customer satisfaction. It is a web-enabled project. With this project the details about the product will be given to the customers in detail within a short span of time. Queries regarding the product or the services will also be clarified. It provides more knowledge about the various technologies.

10. ADVANTAGES AND DISADVANTAGES

Advantage

- Flow sheet is a powerful tool to monitor clinical data and track trends.
- Provides a dashboard of who needs what.
- Provides total population data reporting with no chart abstraction.
- Generates revenue (it shows when services are needed).
- Provides outreach information at fingertips.
- Improves team-based care.
- Smaller software package than EHRs.
- Creating loyal customers through good customer service can provide businesses with lucrative long-term relationships.
- Customer loyalty. Loyal customers have many benefits for businesses.

Disadvantage

- Disease-specific, not longitudinal.
- Does not include information necessary for billing.
- Requires hardware, software and maintenance.
- Requires data entry and data maintenance.
- Parallel documentation system (i.e., some information has to be entered in two systems).
- Can't stand alone, must have an additional documentation system.
- Experience burnout and stress. Working as a customer service representative requires you to maintain a friendly demeanour at all times, regardless of how customers act or how you personally feel.

11.CONCLUSION

The goal of this project is to satisfy customers by providing answers to their questions. The project is web-enabled. With the help of this project, clients will quickly receive detailed information about the product. Questions about the goods or services will also be answered. More information about the various technologies is provided.

12.FUTURE SCOPE

1. Answering each customer's question individually.

2. It is a pivotal moment for marketing.

3. It will bring about a major revolution.

13.APPENDIX

Source Code:

```
from flask import Flask, render_template, request, redirect, session, url_for
import ibm_db
import re
app = Flask(__name__)


# for connection
# conn= ""

app.secret_key = 'a'
print("Trying to connect...")
conn = ibm_db.connect("DATABASE=bludb;HOSTNAME=b1bc1829-6f45-
4cd4-bef4-
10cf081900bf.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=32304;SE
CURITY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=yjs69202;P
WD=nCHXXVA1056i46ky;", '', '')
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)
```

```
            ibm_db.execute(prep_stmt)
            print("successs")
            msg = "succesfully signed up"
        return render_template('dashboard.html', msg=msg, name=name)
    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 Repository Link:

https://github.com/IBM-EPBL/IBM-Project-15960-1659606401

Project Demo Link:

YOUTUBE LINK:

https://youtu.be/RLlD1jZAqq8

DRIVE LINK:

https://drive.google.com/file/d/1eGbeX5hDP51SnSSctr-kOsUNsXmJJiLq/view?usp=drivesdk