

# **A RELIABLE COMPLAINT MANAGEMENT SYSTEM**

A Project Report

submitted in partial fulfillment of the requirements

of

“Applied Cloud Computing for Software Development”

by

**Penyamine .R                      (822720104026)**

**Tamizhselvan .B                  (822720104044)**

**Vasanth .M                        (822720104048)**

**Manikandan M                    (822720104501)**

Under the Esteemed Guidance of

**Mrs. R. Umamaheshwari**

## ACKNOWLEDGEMENT

---

We would like to take this opportunity to express our deep sense of gratitude to all individuals who helped us directly or indirectly during this thesis work.

Firstly, we would like to thank my supervisor Mrs. R. Umamaheshwari ,for being a great mentor and the best adviser I could ever have. His advice, encouragement and critics are source of innovative ideas, inspiration and causes behind the successful completion of this dissertation. The confidence shown on me by him was the biggest source of inspiration for me. It has been a privilege working with him from last one year. He always helped me during my thesis and many other aspects related to academics. His talks and lessons not only help in thesis work and other activities of college but also make me a good and responsible professional.

# ABSTRACT

---

The project aims to create a web application called "The Issue Tracker" that serves as a liable complaint management system for improved customer service.

The main objective is to provide a smart and simple method for individuals to register complaints, track their progress, and resolve them online, eliminating the need for frequent visits to administrative offices.

The target audience for this web application is the general population, particularly residents of housing schemes and societies in India. By utilizing this application, users can save time and money by reporting and monitoring their complaints remotely.

Additionally, the project seeks to tackle corruption in government offices by providing a transparent and efficient complaint management system.

The application allows customers to raise tickets for their complaints. They can provide a detailed description of the issue, such as problems with streetlights, water pipe leakages, or road renovations. Users can also attach image files to better explain their complaints and specify the issue's location. Once a ticket is raised, an agent or service person will be assigned to handle the problem.

# TABLE OF CONTENTS

---

Abstract .....	3
<b>Chapter 1. Introduction .....</b>	<b>6</b>
1.1 Problem Statement .....	6
1.2 Problem Definition .....	6
1.3 Expected Outcomes.....	7
1.4. Organization of the Report .....	7
<b>Chapter 2. Literature Survey .....</b>	<b>8</b>
2.1 Paper-1 .....	9
2.1.1 Brief Introduction of Paper .....	9
<b>Chapter 3. Proposed Methodology .....</b>	<b>10</b>
3.1 System Design .....	11
3.2 Modules Used ... ..	15
3.3 Data Flow Diagram ... ..	16
3.4 Advantages... ..	17
3.5 Requirements Specification... ..	18
<b>Chapter 4. Implementation and Results .....</b>	<b>19</b>
4.1 Login Page .....	20
4.2 Registration Page .....	22
4.3 Post Complaints Page .....	23
4.4 View Complaints Page .....	24
4.5 Assign Agent .....	24
4.6 Assign Status of Problems Page.....	25
<b>Chapter 5.</b>	
Conclusion .....	27
References.....	29
<b>Chapter 6:</b>	
Appendix.....	30

# **CHAPTER 1**

## **INTRODUCTION**

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1. Problem Statement:**

Usually, The people have met problems to complain about corruption in government Offices and problems with street lights, water pipes leakages and road renovation. So people have to visit the admin to complain about these problems .It takes long time and money .

### **1.2. Problem Definition :**

Usually ,The people have met problems to complain about corruption in government Offices and problems with street lights, water pipes leakages and road renovation. So people have to visit the admin to complain about these problems . It takes long time and money .

### **1.3.Expected Outcomes:**

- Good Interface For Issuers, Admin and Agents.
- Authentication for users, Admin and Agents.
- Provide a Direct Contact to Admin.
- Checking a Current Status Of Problems
- Easy to Resolve the Problems.

### **1.4. Organization of the Report:**

The remaining report is organized as follows:

Chapter 2 - Literature Survey

Chapter 3 - Proposed Methodology

Chapter 4 - Implementation and Result

Chapter 5 - Conclusion

Chapter 6 – Appendix

# **CHAPTER 2**

## **LITERATURE SURVEY**



# **CHAPTER 2**

## **LITERATURE SURVEY**

### **2.1. Paper-1**

➤ **Design and Development of a Secure Compliance Management System using Python** by Smith, J. Johnson, .A, Brown, .M

#### **2.1.1. Brief Introduction of Paper**

The literature survey for Paper 1 delves into an exploration of existing research, scholarly articles, and publications relevant to the selected topic. The objective is to establish a comprehensive understanding of the current state of knowledge, identify gaps or areas for further investigation, and contextualize the forthcoming research within the broader academic landscape.

The feasibility of the project is analyzed in this phase, and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis, a feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the developer.

# **CHAPTER 3**

## **PROPOSED METHODOLOGY**

# CHAPTER 3

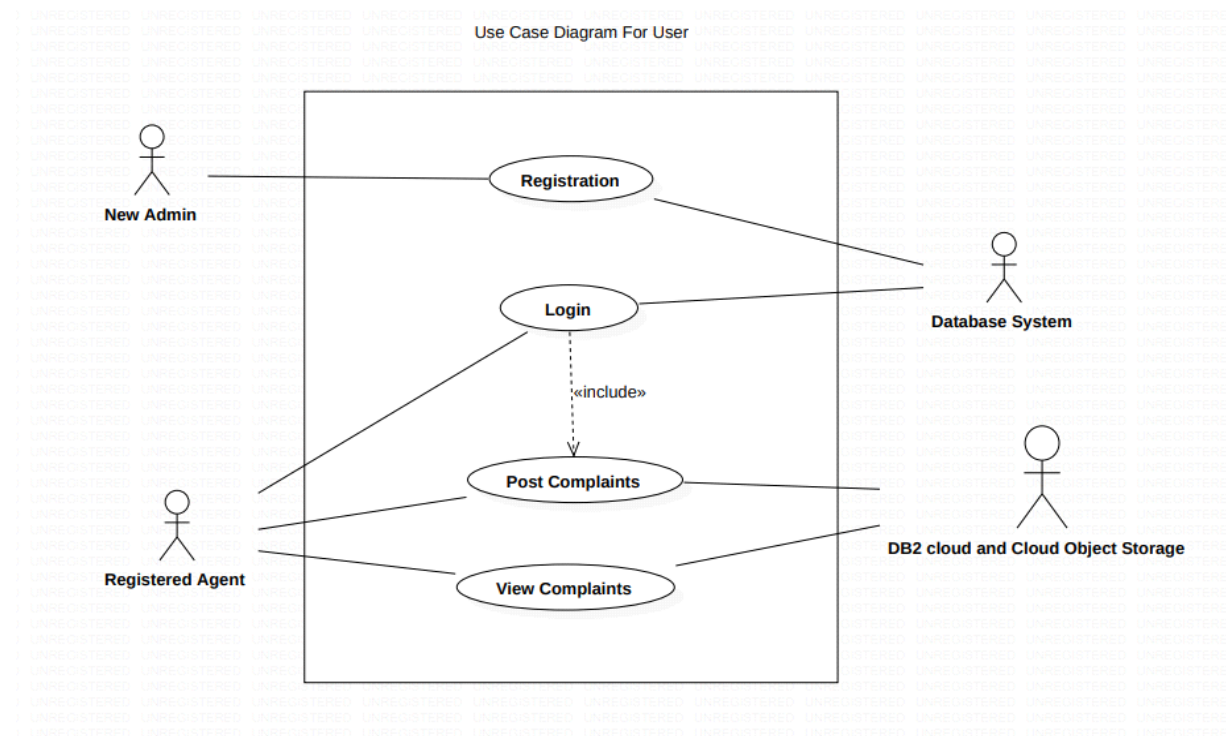
## PROPOSED METHODOLOGY

### 3.1 System Design:

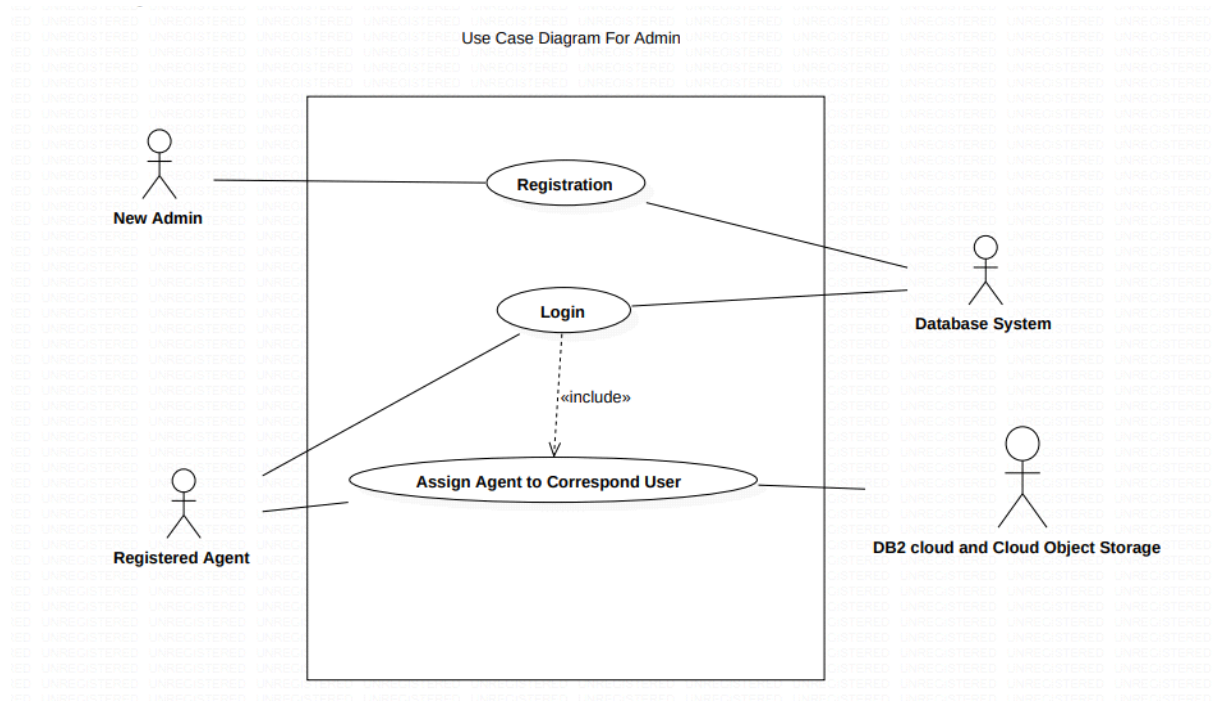
#### USE CASE DIAGRAM

A use case diagram is used to represent the dynamic behavior of the system. The use case diagram for our project is shown in figure 4.1.1.

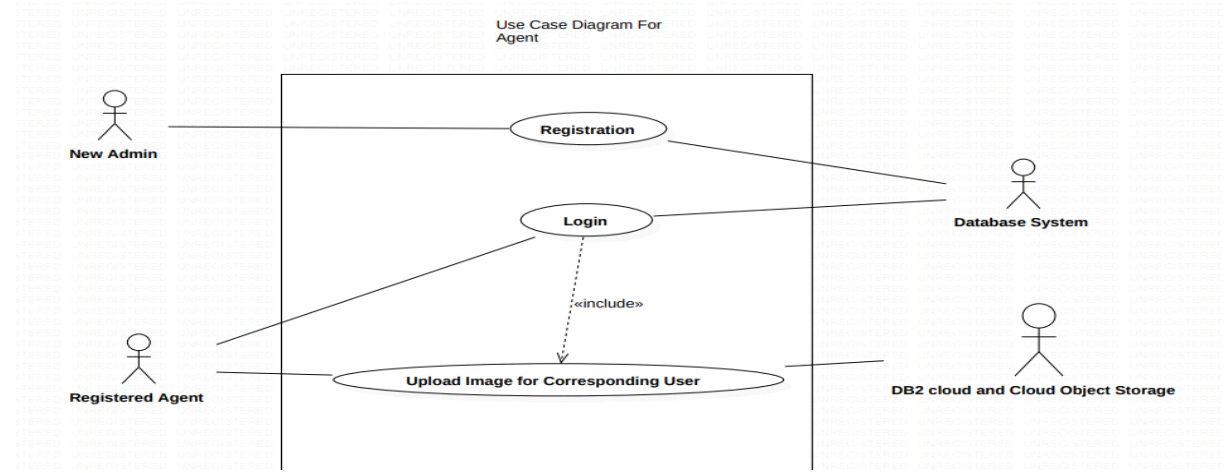
**User :**



## Admin :



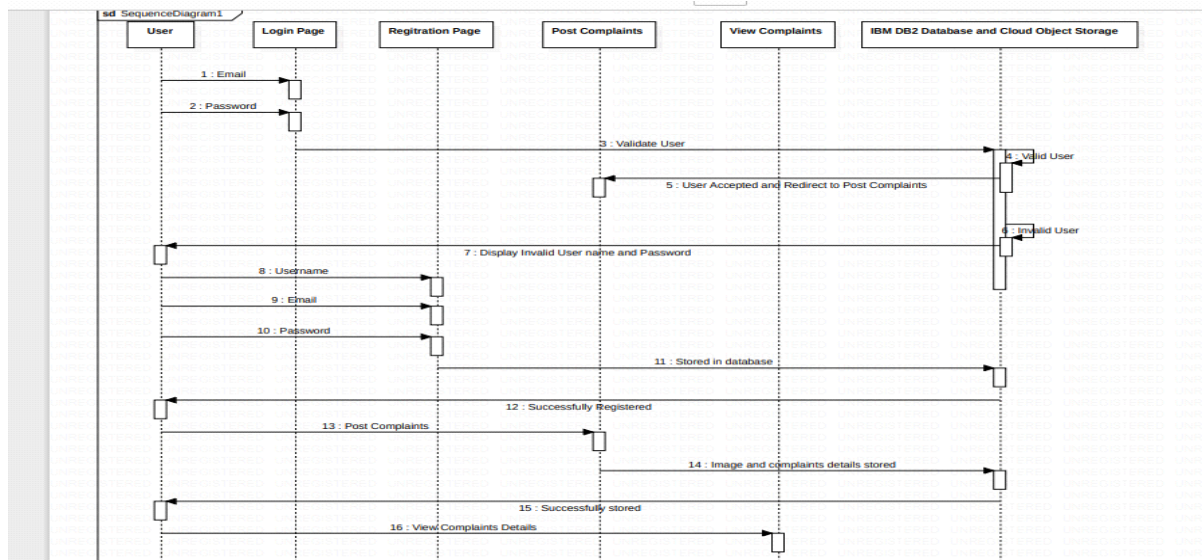
## Agent:



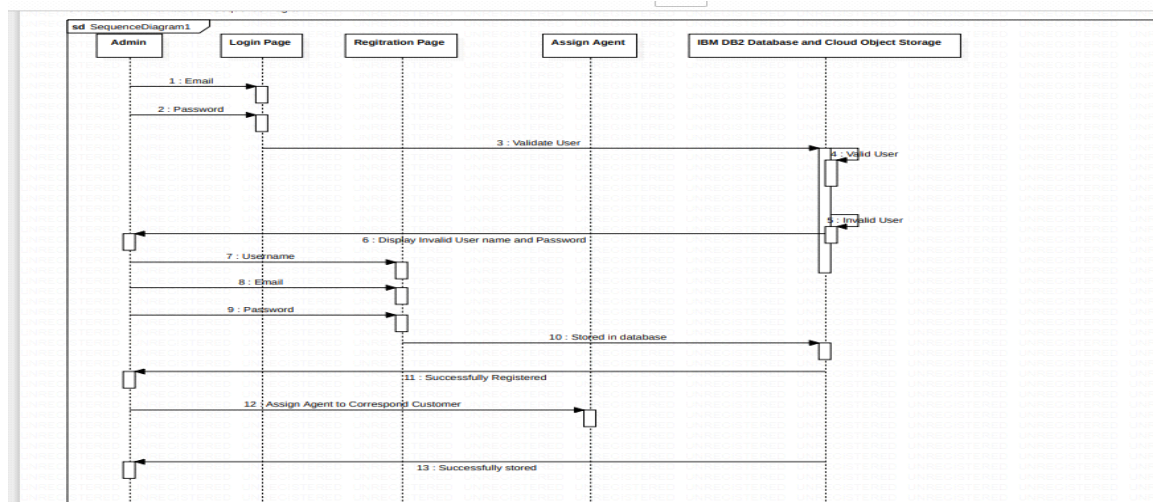
## Sequence Diagram:

### User:

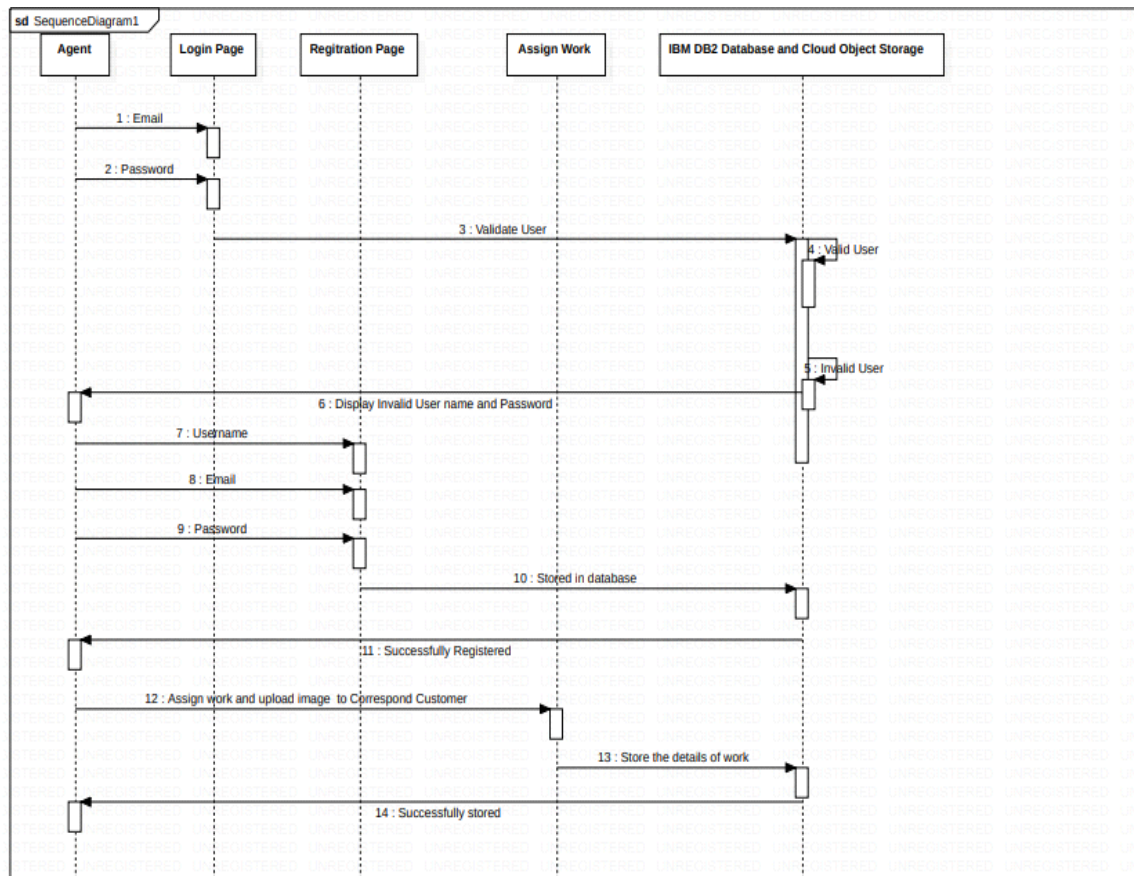
Sequence diagram represents the dynamic behavior of the system. The sequence diagram for the project .



### Admin:



## Agent:



## 3.2 Modules Used:

- User Registration and Authentication
- Post Complaints
- View Complaints
- Assign Agent
- Assign Current Status of a problem

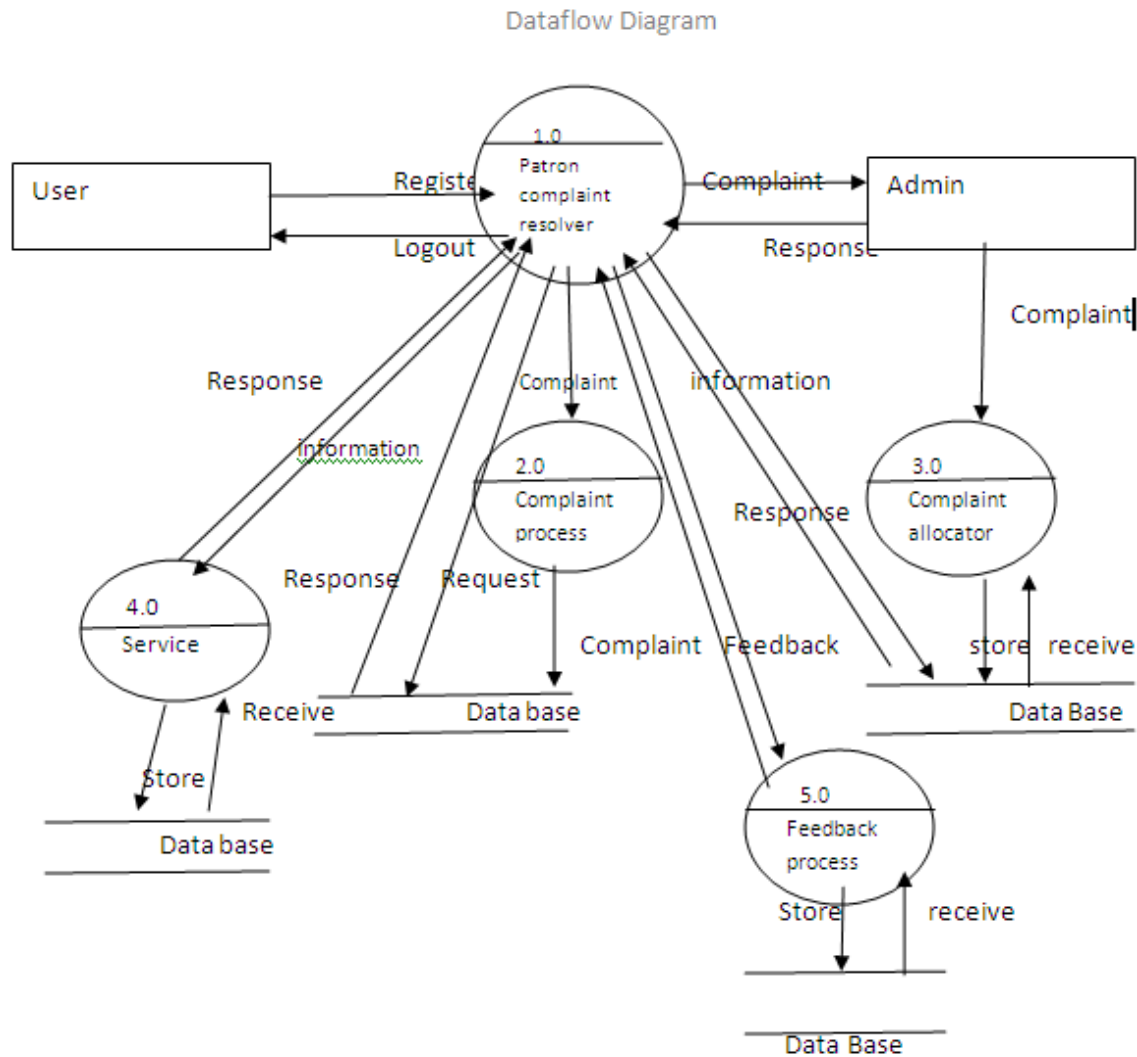
- Front End: HTML, CSS
- Back End: Python Flask
- Database: MySQL
- Development Environment: Visual Studio CODE

### **3.3 Data Flow Diagram:**

A Data Flow Diagram (DFD) is a graphical representation of the "flow" of data through an information system, modeling its process aspects.

A DFD is often used as a preliminary step to create an overview of the System , which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).





### 3.4 Advantages:

- 1.Improved Customer Service: The project provides a user-friendly and efficient platform for customers to register complaints and track their progress. This improves customer satisfaction by offering a convenient and transparent complaint management process.
- 2.Time and Cost Savings: By allowing users to report complaints online, the project saves time and money for both customers and administrative staff. Users can avoid the need to visit physical offices, reducing travel costs and time spent in queues.
3. Transparency and Accountability: The project promotes transparency by documenting and tracking all actions taken on each complaint. This ensures accountability and eliminates potential corruption or negligence in the complaint management process.
4. Automation and Efficiency: By automating the complaint management system, the project improves response times, and enhances overall efficiency.

5. Enhanced Communication: The project facilitates effective communication between users, agents, and administrators. Users can **provide** detailed descriptions of their complaints, while agents and administrators can respond, update, and address complaints promptly.

6. Data-Driven Decision Making: The project collects and stores complaint data, allowing for analysis and insights. Administrators can use this data to identify trends, allocate resources effectively, and make data-driven decisions for process improvements.

7. User Feedback and Continuous Improvement: The project includes mechanisms for users to provide feedback on the service received. This feedback helps identify areas for improvement, enabling the project team to continuously enhance the complaint management system.

8. Scalability and Flexibility: The web-based nature of the project allows for scalability and flexibility. It can accommodate a growing number of users and handle various types of complaints, adapting to changing needs and requirements.

9. Reduction in Paperwork: The project minimizes the need for physical paperwork by digitizing the complaint management process. This reduces administrative burden, saves storage space, and contributes to environmental sustainability.

### **3.5 Requirements Specification:**

#### **HARDWARE REQUIREMENTS:**

- A System with a minimum of 8GB RAM
- 512GB Hard Disk
- Good Internet Connection
- MYSQL Database Server

#### **SOFTWARE REQUIREMENTS:**

- Python flask
- SQL
- HTML , CSS
- Operating system minimum windows 10

# **CHAPTER 4**

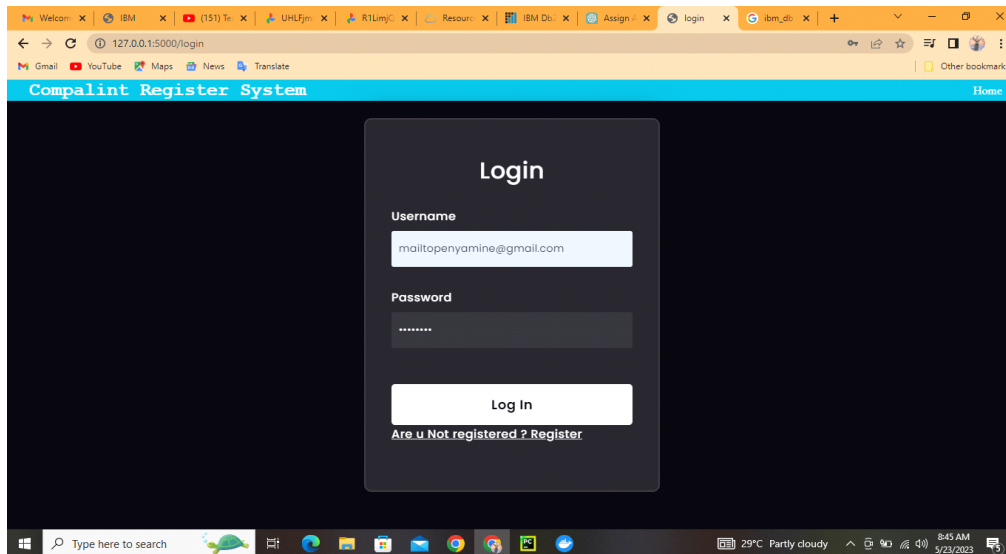
## **IMPLEMENTATION AND RESULTS**

# CHAPTER 4

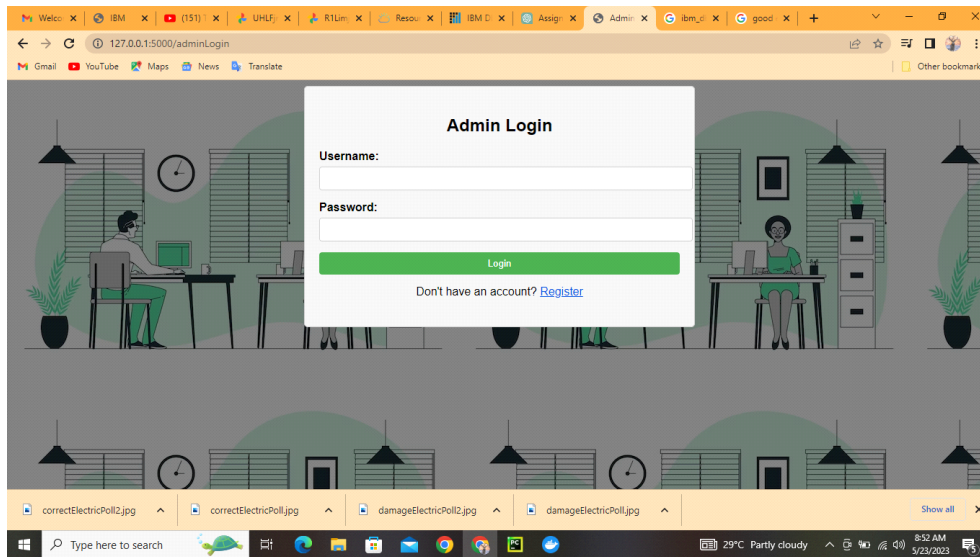
## IMPLEMENTATION AND RESULTS

### 4.1. Login Page :

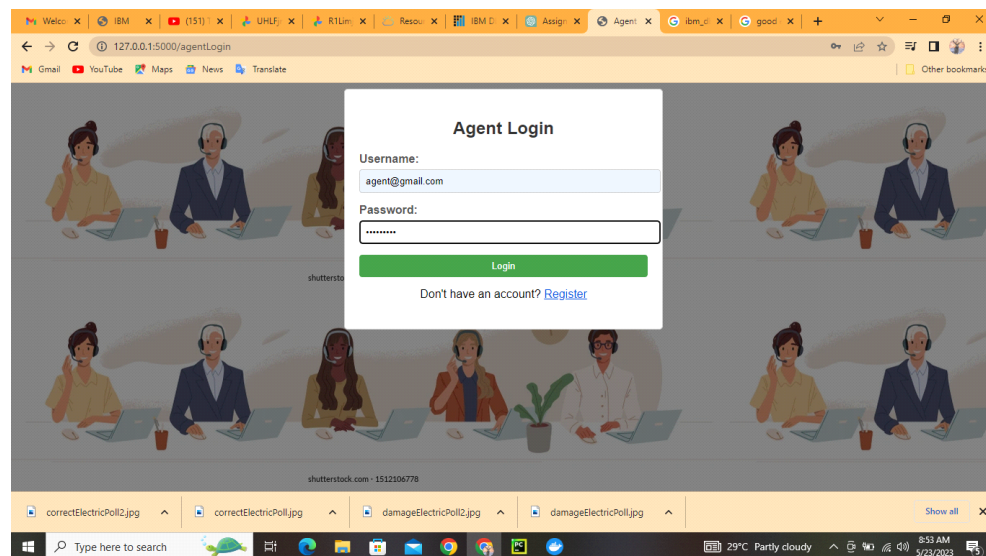
**User:**



**Admin:**



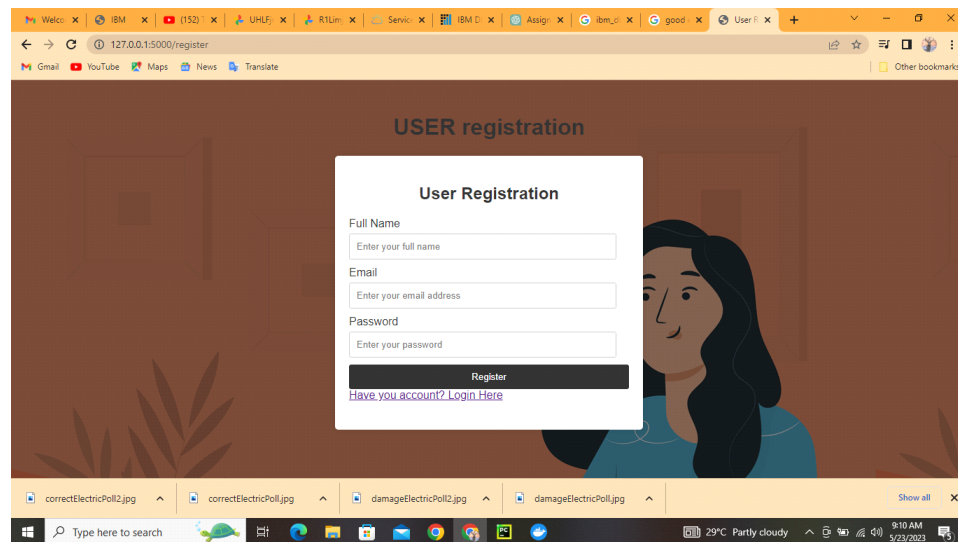
## Agent:



The Login Form validates the credentials provided by the user . If the credentials provided by the user is correct ,then it will redirect to the main page of our project .Otherwise ,it will display error message to the user to register first then login .

## 4.2. Registration Page:

User:

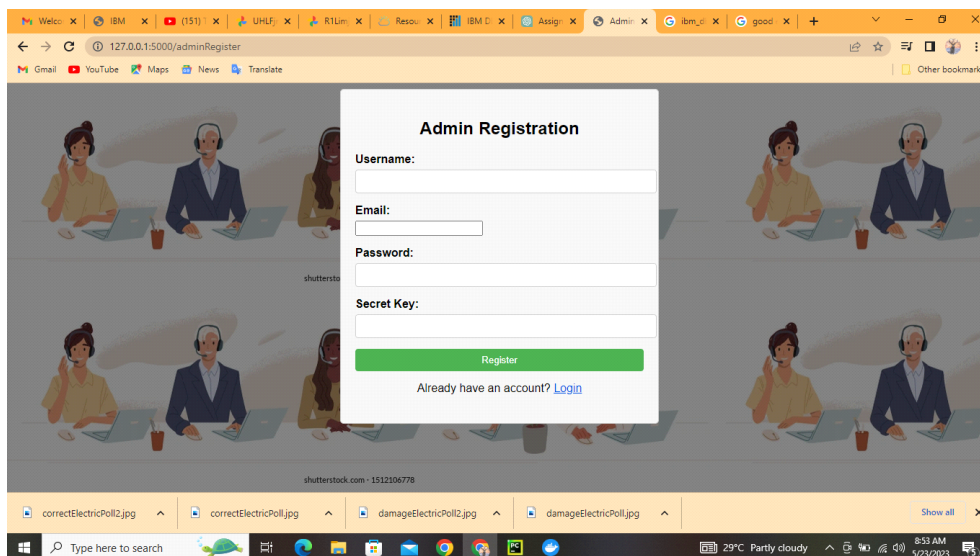


The screenshot shows a web browser window with the URL `127.0.0.1:5000/register`. The page has a dark brown background with a faint illustration of a woman's face. A white modal box titled "User Registration" is centered on the page. It contains the following fields and elements:

- Full Name:** A text input field with the placeholder "Enter your full name".
- Email:** A text input field with the placeholder "Enter your email address".
- Password:** A text input field with the placeholder "Enter your password".
- Register:** A black button with white text.
- Have you account? Login Here:** A blue link.

The browser's taskbar at the bottom shows several open files: `correctElectricPoll2.jpg`, `correctElectricPoll.jpg`, `damageElectricPoll2.jpg`, and `damageElectricPoll.jpg`. The system tray shows the date and time as 9:10 AM on 5/23/2023.

Admin:

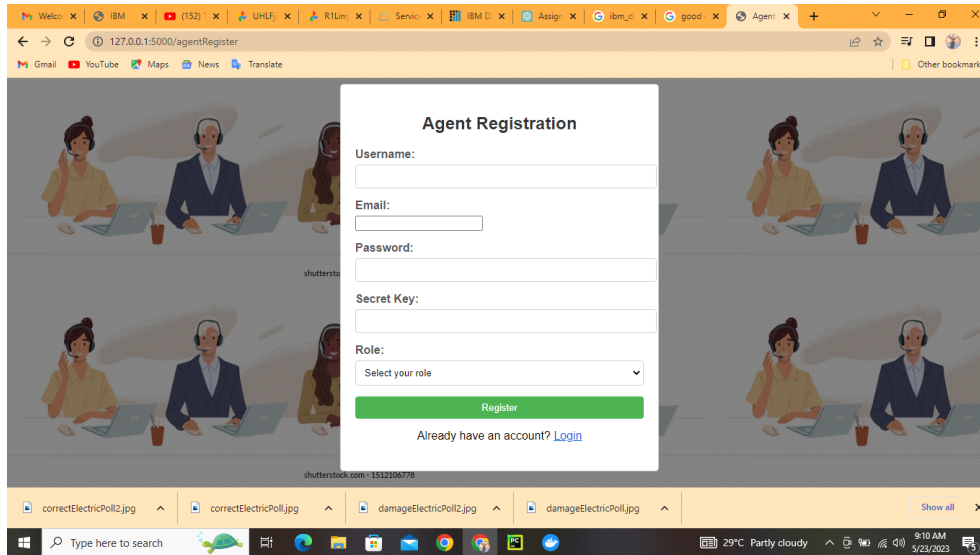


The screenshot shows a web browser window with the URL `127.0.0.1:5000/adminRegister`. The page has a light gray background with a faint illustration of several people working at computers. A white modal box titled "Admin Registration" is centered on the page. It contains the following fields and elements:

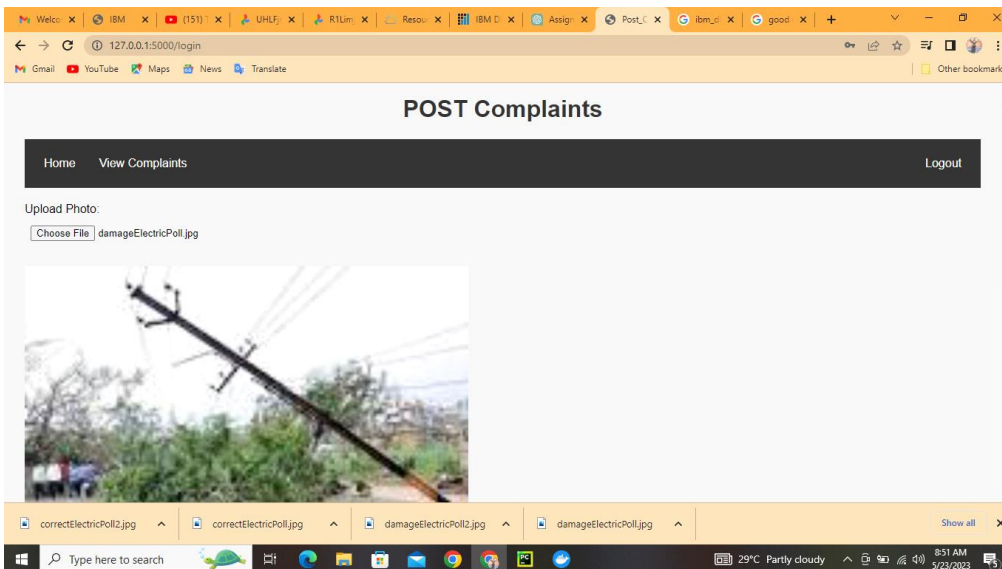
- Username:** A text input field.
- Email:** A text input field.
- Password:** A text input field.
- Secret Key:** A text input field.
- Register:** A green button with white text.
- Already have an account? Login:** A blue link.

The browser's taskbar at the bottom shows the same open files as the previous screenshot. The system tray shows the date and time as 8:53 AM on 5/23/2023.

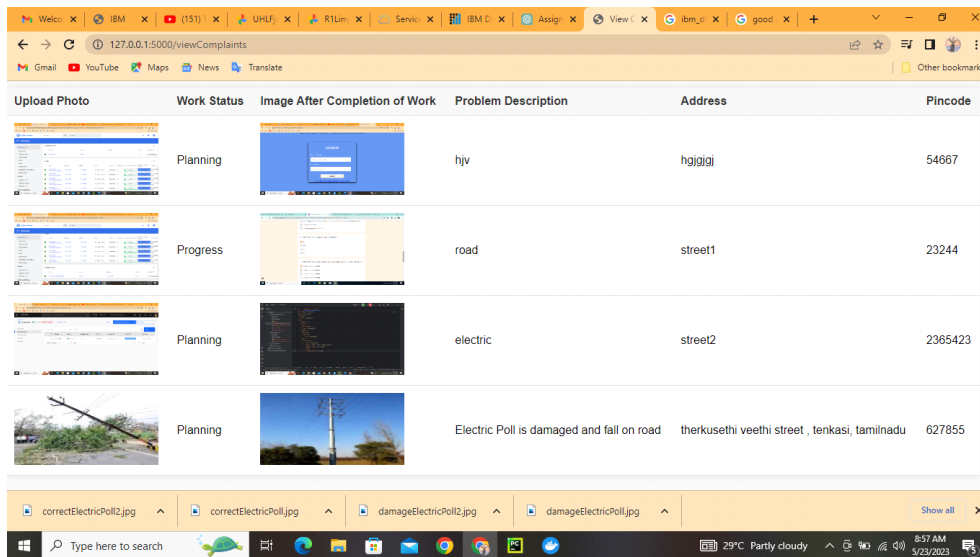
Agent:

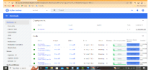




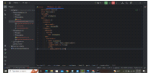




### 4.3. Post Complaints page:

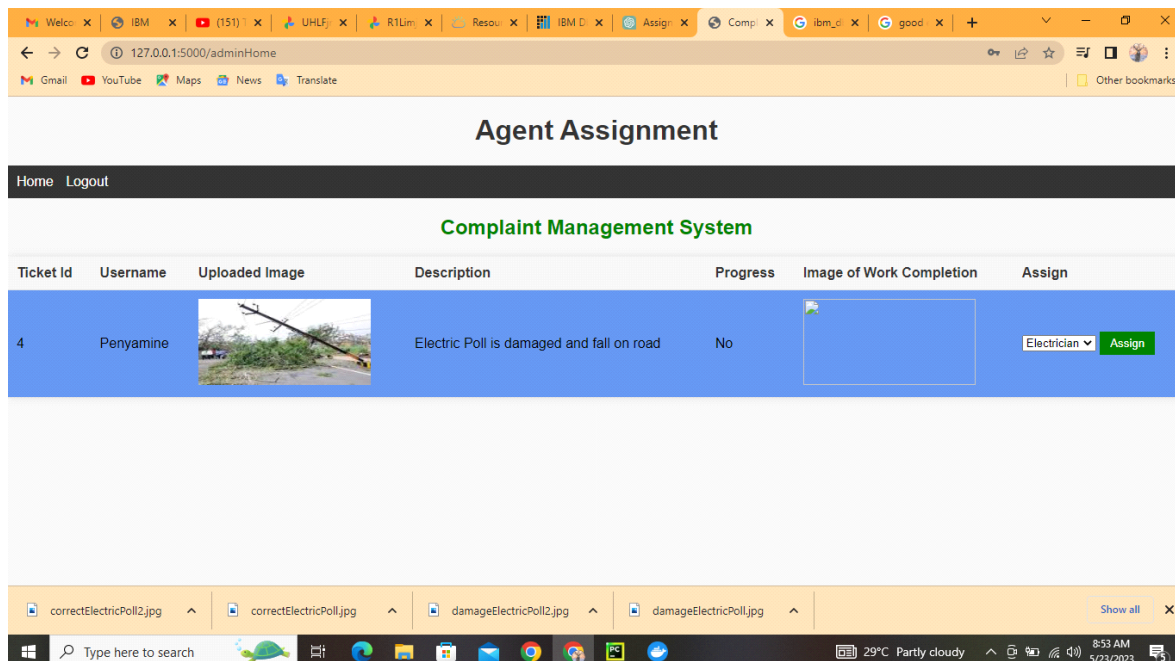


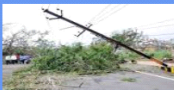
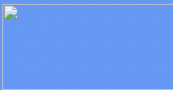
## 4.4. View Complaints Page:



Upload Photo	Work Status	Image After Completion of Work	Problem Description	Address	Pincode
	Planning		hvj	hgjgigj	54667
	Progress		road	street1	23244
	Planning		electric	street2	2365423
	Planning		Electric Poll is damaged and fall on road	therkusethi veethi street , tenkasi, tamilnadu	627855

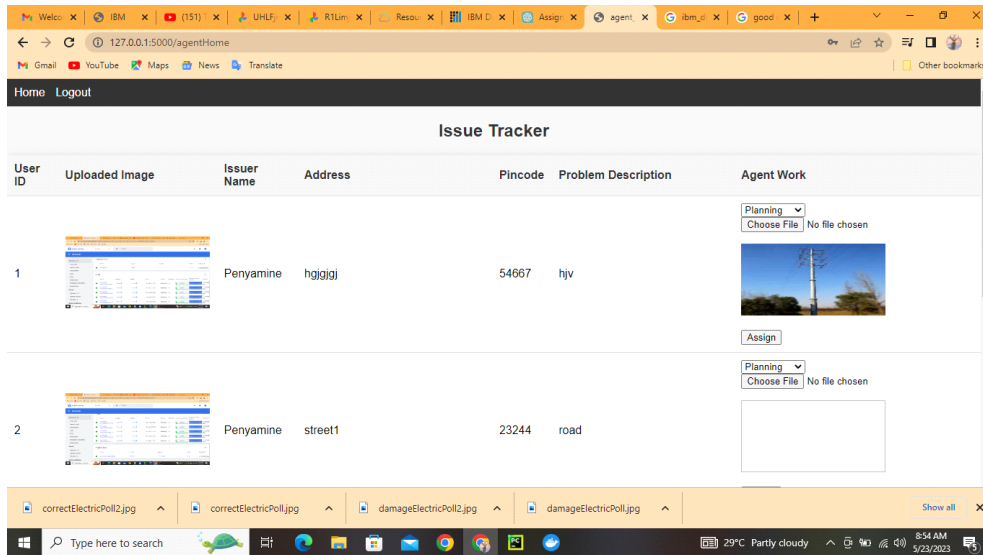
## 4.5. Assign Agent:



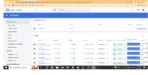


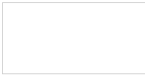
Agent Assignment						
Home Logout						
Complaint Management System						
Ticket Id	Username	Uploaded Image	Description	Progress	Image of Work Completion	Assign
4	Penyamine		Electric Poll is damaged and fall on road	No		Electrician <input type="button" value="Assign"/>



## 4.6. Assign status of problems Page:



The screenshot displays a web application titled "Issue Tracker" with a table containing two rows of problem data. The table columns are: User ID, Uploaded Image, Issuer Name, Address, Pincode, Problem Description, and Agent Work. The first row shows a problem with User ID 1, Issuer Name Penyamine, Address hgigigi, Pincode 54667, and Problem Description hvj. The second row shows a problem with User ID 2, Issuer Name Penyamine, Address street1, Pincode 23244, and Problem Description road. Each row has a "Planning" dropdown menu, a "Choose File" button, and a "No file chosen" status. The first row also has a "Assign" button. The application is running on a browser with the URL 127.0.0.1:5000/agentHome. The Windows taskbar at the bottom shows the date and time as 8:54 AM on 3/23/2023.

User ID	Uploaded Image	Issuer Name	Address	Pincode	Problem Description	Agent Work
1		Penyamine	hgigigi	54667	hvj	<div>Planning Choose File No file chosen</div>  <div>Assign</div>
2		Penyamine	street1	23244	road	<div>Planning Choose File No file chosen</div> 

# **CHAPTER 5**

## **CONCLUSION**

# CHAPTER 5

## CONCLUSION

- The project "Reliable Compliance management System" aims to empower vehicle owners by providing a web-based application that facilitates informed decision-making in this domain.
- Through comprehensive cost analysis and user-friendly interface, the application will enable users to make cost-effective choices, ultimately enhancing their overall Compliance experience.
- It aims to assist individuals in resolving the complaints of the Customers.
- Overall, the project aims to simplify the complex world of Compliance Management and provide individuals with the knowledge and tools necessary to make better decisions when it comes to resolving problems for their vehicles.
- It is mainly used to reduce the manual work i.e., loss of manpower, time, and cost.

## **FUTURE ENHANCEMENT :**

- This project is only for Complaints, so in future it need to be extended to Complaints also to provide efficient Complaint Management System.
- Based upon the customer Complaints, the Resolving report should be generated .
- Interactive Visualization and User-Friendly Interface: To improve user engagement and comprehension, the application can enhance its visualizations and user interface.
- Interactive charts, graphs, and maps can be incorporated to present data in a visually appealing manner. Additionally, intuitive user interfaces and streamlined workflows will make it easier for users to navigate the complex world of Compliance management and make well-informed decisions.

## REFERENCES:

[1]. Prof. Bharati Pursharthi a, Kanchan Ramesh Deshmukh, "Building a Modern Banking System: Implementation of a Advanced Java-Based Solution", ISSN 2582-7421.

[2]. Search engines by W. Bruce Croft, Donald Metzler, Trevor Strohman

[3]. Web search engine and International journal and magazine of engineering, technology, management and research.

[4]. <https://www.w3schools.com/>

[5]. <https://stackoverflow.com/>

[6]. <https://www.mysql.com/>

## APPENDIX

### SAMPLE SOURCE CODE

**APP.py:**

```
from flask import Flask, render_template, request, redirect, url_for, session, send_from_directory
import mysql.connector
import hashlib
import re
import base64
import os

app = Flask(__name__, template_folder='templates')
app.secret_key = 'your_secret_key_here'

conn=mysql.connector.connect(host='127.0.0.1',password='peny',user='root',database='IssueTracker')
if conn.is_connected():
    print("Connection is Established")
cursor=conn.cursor()

@app.route("/", methods=['POST', 'GET'])
def dashboard():
    return render_template('home.html')

@app.route("/login", methods=['POST', 'GET'])
def login():
    msg = "
```

```

if (request.method == "POST"):
    email = request.form.get("email")
    password = request.form.get("password")
    sql = "SELECT * FROM USERN WHERE EMAIL=%s and password=%s"
    cursor.execute(sql,(email,password))
    account=cursor.fetchone()

    if account:
        session['Loggedin'] = True
        session['ID'] = account[0]
        session['USERNAME'] = account[2]
        msg = "logged Successfully"
        return render_template('/postComplaints.html', msg=' ')
    else:
        msg = "Incorrect Email/Password"
        return render_template('/login.html', msg=msg)

return render_template('login.html', msg=msg)

```

```
@app.route("/adminLogin", methods=['POST', 'GET'])
```

```
def adminLogin():
```

```

    msg = ""
    if (request.method == "POST"):
        email = request.form.get("email")
        password = request.form.get("password")
        sql = "SELECT * FROM USERN WHERE EMAIL=%s and password=%s"

```

```

        cursor.execute(sql,(email,password))

        account=cursor.fetchone()


        if account:

            session['Loggedin'] = True

            session['ID'] = account[0]

            session['USERNAME'] = account[2]

            msg = "logged Successfully"

            return render_template('/adminHome.html', msg=msg)

        else:

            msg = "Incorrect Email/Password"

            return render_template('/adminLogin.html', msg=msg)

    return render_template('adminLogin.html', msg=msg)


@app.route("/agentLogin", methods=['POST', 'GET'])
def agentLogin():

    msg = ""

    if (request.method == "POST"):

        email = request.form.get("email")

        password = request.form.get("password")

        sql = "SELECT * FROM USERN WHERE EMAIL=? AND PASSWORD=?"

        cursor.execute(sql,(email,password))

        account=cursor.fetchone()


        if account:

            session['Loggedin'] = True

            session['ID'] = account[0]

```



```

        session['USERNAME'] = account[2]

        msg = "logged Successfully"

        return render_template('/agentHome', msg=msg)

    else:

        msg = "Incorrect Email/Password"

        return render_template('/agentLogin.html', msg=msg)

return render_template('agentLogin.html', msg=msg)

```

```
@app.route("/register", methods=['POST', 'GET'])
```

```
def register():
```

```

    msg = ""

    print("hello")

    if (request.method == "POST"):

        print("Hello")

        username = request.form.get('username')

        email = request.form.get('email')

        password = request.form.get('password')

        ROLE = 'USER'

        print(username+email+password)

        sql="select *from usern where email=%s and password=%s"

        cursor.execute(sql,(email,password))

        account=cursor.fetchone()

        print(account)

        if account:

            msg = "Your signup details  already exists please login"

```

```

elif not re.match(r'^@]+@^[^@]+\.[^@]+', email):

    msg = "Invalid Email Address"

else:

    sql = "SELECT COUNT(*) FROM USERN"

    cursor.execute(sql)

    result=cursor.fetchone()

    length=result[0] if result is not None else 0

    # length=cursor.rowcount

    print(length)


    insert_sql = "INSERT INTO USERN
(id,role,username,email,password)values(%s,%s,%s,%s,%s)"

    cursor.execute(insert_sql,(length+1,ROLE,username,email,password))

    conn.commit()

    msg = "You have Successfully Registered !!!"

print(msg)

return render_template("register.html", msg=msg)


@app.route("/adminRegister", methods=['POST', 'GET'])
def adminRegister():

    msg = ""

    print("hello")

    if (request.method == "POST"):

        print("Hello")

        username = request.form.get('username')

        email = request.form.get('email')

        password = request.form.get('password')

```

```

ROLE = 'ADMIN'

secret_key = request.form.get('secret')


sql = "SELECT *FROM  USERN WHERE EMAIL='"+email+"' and  PASSWORD='"+password+"'"
cursor.execute(sql)
account=cursor.fetchone()
print(account)
if account:
    secret_key == "12345"
    msg = "Your signup details  already exists please login"
    return render_template("adminRegister.html", msg=msg)
elif not re.match(r'^[@]+@[^@]+\.[^@]+', email):
    msg = "Invalid Email Address"
else:
    secret_key = "12345"
    sql = "SELECT COUNT(*) FROM USERN"
    cursor.execute(sql)
    record=cursor.fetchone()
    length=record[0] if record[0] is not None else 0
    print(length)

    insert_sql = "INSERT INTO USERN(id,role,username,email,password) VALUES
(%s,%s,%s,%s,%s)"

    cursor.execute(insert_sql,(length+1,ROLE,username,email,password))
    conn.commit()
    msg = "You have  Successfully Registered !!!"
    return render_template("adminRegister.html", msg=msg)

```

```

    print(msg)

    return render_template("adminRegister.html", msg=msg)

@app.route("/agentRegister", methods=['POST', 'GET'])
def agentRegister():
    msg = ""
    print("hello")

    if (request.method == "POST"):
        print("Hello")

        username = request.form.get('username')
        email = request.form.get('email')
        password = request.form.get('password')
        role = request.form.get('role')
        secret_key = request.form.get('secret')
        print(role)

        sql = "SELECT *FROM  USERN WHERE EMAIL='"+email+"' AND PASSWORD='"+password+"'"
        cursor.execute(sql)
        account = cursor.fetchone()
        print(account)

        if account:
            msg = "Your signup details  already exists please login"
            return render_template("agentRegister.html", msg=msg)

        elif not re.match(r'^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid Email Address"

        else:
            sql = "SELECT COUNT(*) FROM USERN"

```

```

        cursor.execute(sql)

        record=cursor.fetchone()

        length=record[0] if record[0] is not None else 0

        print(length)


        insert_sql = "INSERT INTO USERN(id,role,username,email,password) VALUES
(%s,%s,%s,%s,%s)"

        cursor.execute(insert_sql,(length+1,role,username,email,password))

        conn.commit()

        msg = "You have   Successfully Registered !!!"

        return render_template("agentRegister.html", msg=msg)

    print(msg)

    return render_template("agentRegister.html", msg=msg)


@app.route('/adminHome', methods=["POST", "GET"])
def adminHome():

    print("admin Home")

    sql = "SELECT USER_ID,USERNAME , IMAGE_ID,DESCRIPTION,PROGRESS,AFTER_IMAGE_ID FROM
TICKETS WHERE PROGRESS = 'No' "

    cursor.execute(sql)


    data = []


    record = list(cursor.fetchall())

    for row in record:

        if row:

            row2=list(row)

```

```

        row2[2] = '../static/Uploads/' + row2[2]

        if row2[5]:
            row2[5] = '../static/Completed/' + row[5]

        data.append(tuple(row2))

    else:
        break

    return render_template("adminHome.html", data=data)

```

```
@app.route('/agentHome', methods=["POST", "GET"])
```

```
def agentHome():
```

```
    print("agent Home")
```

```
    EMAIL=request.form.get('email')
```

```
    data = []
```

```
    if EMAIL is not None:
```

```
        role_sql = "SELECT ROLE FROM USERN WHERE EMAIL = '"+EMAIL+"'"
```

```
        cursor.execute(role_sql)
```

```
        Role=cursor.fetchone()
```

```
    if Role:
```

```
        Role=Role[0]
```

```
        print(Role)
```

```
        sql = "SELECT USER_ID, IMAGE_ID, USERNAME, ADDRESS, PINCODE, DESCRIPTION FROM
TICKETS WHERE ASSIGNED = '"+Role+"'"
```

```
        cursor.execute(sql)
```

```
        record=list(cursor.fetchall())
```

```

        for row in record:
            if row:
                row2=list(row)
                row2[1] = '../static/Uploads/' + row2[1]
                data.append(tuple(row2))
            else:
                break
        for row in record:
            print(row)
    return render_template("agentHome.html", data=data)

```

```
@app.route('/assign-agent', methods=['POST'])
```

```
def assign_agent():
```

```

    print("This is assign agent")
    userId = request.form.get('userId')
    roleId = request.form.get('roleId')
    print(roleId)
    print(userId)
    sql = "UPDATE TICKETS SET ASSIGNED ='"+roleId+" WHERE USER_ID= '" + userId+"'"
    cursor.execute(sql)
    conn.commit()
    return redirect(url_for("adminHome"))

```

```
@app.route('/assignWork', methods=["POST", "GET"])
```

```
def assignWork():
```

```

if request.method == "POST":

    userId = request.form.get("userId")

    print(userId)

    print(request.form)

    f = request.files['image']

    save_directory = os.path.join(app.root_path, 'static', 'Completed')

    f.save(os.path.join(save_directory, f.filename))

    progress = request.form['status']

    after_image_id = f.filename

    print(progress)

    print(after_image_id)

    print(userId)

    sql = "UPDATE TICKETS SET PROGRESS = '"+progress+"' WHERE  USER_ID= '" + userId+"'"

    cursor.execute(sql)

    # conn.commit()

    sql = "UPDATE TICKETS SET AFTER_IMAGE_ID ='"+after_image_id+"'  WHERE  USER_ID= '"+
userId+"'"

    cursor.execute(sql)

    conn.commit()

    return "Image Uploaded Successfully"

@app.route("/logout",methods=["POST","GET"])
def logout():

    session.pop('loggedin', None)

    session.pop('USERID', None)

```



```

        return render_template("home.html")

@app.route("/home",methods=["POST","GET"])
def home():
    return render_template("home.html")

@app.route('/postComplaints', methods=['POST', 'GET'])
def postComplaints():
    sql = "SELECT * FROM USERN WHERE ID=" + str(session['ID'])
    cursor.execute(sql)
    User=cursor.fetchone()

    if User[0] != '0':
        if request.method == "POST":
            f = request.files['image']
            print('File name is :',f.filename)
            if (len(f.filename)==0):
                print("No Image")
                return render_template('postComplaints.html', msg="Select the image")

            save_directory = os.path.join(app.root_path, 'static', 'Uploads')
            f.save(os.path.join(save_directory, f.filename))
            print(f.filename)
            IMAGE_ID = f.filename

            DESCRIPTION = request.form.get("description")
            ADDRESS = request.form.get("address")

```

```
PINCODE = request.form.get("pincode")
```

```
print(DESCRIPTION)
```

```
print(ADDRESS)
```

```
print(PINCODE)
```

```
sql = "SELECT * FROM USERN WHERE ID=" + str(session['ID'])
```

```
cursor.execute(sql)
```

```
record=cursor.fetchall()
```

```
for row in record:
```

```
    email=row[3]
```

```
    username=row[2]
```

```
    break
```

```
print("email"+email)
```

```
print("username"+username)
```

```
assigned="NO"
```

```
progress="NO"
```

```
after_image_id="NoImage"
```

```
sql="select count(*) from tickets"
```

```
cursor.execute(sql)
```

```
record=cursor.fetchone()
```

```
length=record[0]if record is not None else 0
```

```
print(length+1)
```

```
sql="insert into tickets
```

```
(user_id,username,email,assigned,description,progress,address,pincode,image_id,after_image_id)
```

```
values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
```

```
cursor.execute(sql,(str(length+1),username,email,assigned,DESCRIPTION,progress,ADDRESS,str(PINCODE),IMAGE_ID,after_image_id))
```

```
conn.commit()
```

```
return render_template('postComplaints.html', msg="Your Complaints are provided to Successfully")
```

```
@app.route("/viewComplaints",methods=["post","GET"])
```

```
def viewComplaints():
```

```
    sql = "SELECT * FROM USERN WHERE ID=" + str(session['ID'])
```

```
    cursor.execute(sql)
```

```
    record=cursor.fetchall()
```

```
    for row in record:
```

```
        email=row[3]
```

```
        username=row[2]
```

```
        break
```

```
    #,ADDRESS ,PINCODE
```

```
    sql = "SELECT IMAGE_ID,PROGRESS,AFTER_IMAGE_ID,DESCRIPTION,ADDRESS,PINCODE  
FROM TICKETS WHERE EMAIL = '"+email+"'"
```

```
    cursor.execute(sql)
```

```
    record=list(cursor.fetchone())
```

```
    data = []
```

```
    if record:
```

```
        record[0]= "../static/Uploads/" + str(record[0])
```

```
        record[2]="../static/Completed/"+str(record[2]) if record[2]!="NoImage" else "NoImage"
        data.append(record)
    return render_template("viewComplaints.html",data=data)

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