ANUDIP FOUNDATION

**Project Title  
“Complaint Management System”**

|  |  |
| --- | --- |
| Name | Enrollment No |
| Kavita sonawane | **AF0481784** |

Under Guidance of

**Rajshri Thete**

**ABSTRACT**

The **Complaint Management System** is an innovative and efficient web-based application developed using the Django framework that addresses the common administrative challenge of handling complaints in a structured and transparent manner. Traditional methods of complaint lodging, often paper-based or spread across unorganized digital platforms, are prone to delays, lack of accountability, and loss of records. This system aims to provide a centralized platform where complaints can be registered, tracked, and resolved in a systematic and time-bound manner.

The system supports two main types of users: **regular users** and **administrators**. Users can register, log in securely, and file complaints specifying the subject, details, and category of the issue. Once a complaint is lodged, it is stored in the database and is visible in the user dashboard along with its current status (e.g., Pending, In Progress, Resolved). The **administrator**, who has backend access, can view all complaints submitted across the platform, assign them for resolution, change their statuses, and optionally add comments or updates visible to the user.

From a technical standpoint, the project utilizes the **Model-View-Template (MVT)** architectural pattern provided by Django, making it modular and scalable. The backend is powered by Django’s ORM, ensuring seamless data management with SQLite or MySQL. The frontend is designed to be user-friendly using HTML, CSS, and Bootstrap, ensuring responsiveness across devices.

This project not only simplifies the grievance redressal mechanism but also improves institutional transparency and user satisfaction by offering timely updates and consistent feedback. It can be implemented in colleges, municipalities, housing societies, or any organization that handles complaints regularly. In the future, the system can be enhanced with features such as SMS/email notifications, complaint categorization by urgency or department, and detailed analytics for tracking performance and bottlenecks in the resolution process.

In conclusion, the **Complaint Management System** is a practical and scalable solution that aligns with the increasing demand for digital transformation in administrative processes and significantly improves communication between users and the service providers responsible for addressing their issues.

**ACKNOWLEDGEMENT**

The project **“Compliant management System”** is the Project work carried out by

|  |  |
| --- | --- |
| **Name** | **Enrollment No** |
| **Rajshri Thete** | **AF0481784** |

Under the Guidance.

We are thankful to my project guide for guiding me to complete the Project.

His suggestions and valuable information regarding the formation of the Project Report have provided me a lot of help in completing the Project and its related topics.

We are also thankful to my family member and friends who were always there to provide support and moral boost up.

|  |  |  |
| --- | --- | --- |
| **Sr.no** | **Topic** | **Page no** |
| 1 | Title of Research | 3 |
| 2 | Certificate | 4 |
| 3 | Abstract | 2 |
| 4 | Introduction | 5 |
| 5 | Objective | 6 |
| 6 | System Anlaysis | 7-11 |
| 7 | System design /UML Diagrams | 12-14 |
| 8 | MODULES AND THEIR DESCRIPTION | 15 |
| 9 | Input Screen | 16-19 |
| 11 | Coding | 20-40 |
| 12 | Validation | 41-42 |
| 13 | Databases | 43 |
| 14 | Testing | 44-45 |
| 15 | conclusion | 46 |
| 16 | Future scope | 47 |
| 17 | Biography | 48 |



### ****Introduction****

In today’s fast-paced digital environment, effective communication between users and service providers is essential. One of the key components of this communication is a reliable and transparent complaint management system. The Complaint Management System is a web-based application designed to streamline the process of lodging, tracking, and resolving complaints. It provides a structured platform where users can submit grievances and receive timely updates regarding their resolution.

This system is particularly useful for organizations, institutions, or service providers who aim to maintain high customer satisfaction by efficiently handling user feedback and complaints. By digitizing the complaint handling process, the system reduces paperwork, enhances transparency, and improves response time.

The Complaint Management System is built using the Django framework, offering a secure, scalable, and user-friendly interface for both administrators and users. Admins can monitor, assign, and resolve complaints, while users can track the status of their submitted issues in real-time.

### ****Objectives****

The primary objectives of the Complaint Management System are:

1. **To provide an easy and accessible platform** for users to lodge complaints from any location at any time.
2. **To automate the complaint handling process** and minimize human intervention, ensuring faster and more accurate responses.
3. **To enable administrators to efficiently manage complaints**, track their status, and generate reports for better decision-making.
4. **To ensure transparency** by allowing users to monitor the progress and status of their complaints.
5. **To maintain a database of complaints** for future reference, auditing, and analysis to help improve service quality.
6. **To enhance accountability** by maintaining records of complaint responses and resolutions.
7. **To improve communication** between the service provider and the user, ensuring grievances are addressed promptly.

**3. System analysis**

### ****3.1 Problem Definition****

In many institutions, users struggle with reporting complaints due to inefficient manual systems or lack of structured platforms. Verbal reporting or written logs often result in untracked issues, miscommunication, and slow resolutions. This project addresses the problem by creating an online Complaint Management System that allows users to submit complaints digitally while enabling administrators to monitor, manage, and resolve them efficiently.

### ****3.2 Preliminary Investigation****

An initial study was conducted to understand the challenges faced by both users and administrators in the complaint process. It revealed the need for a centralized platform with user authentication, complaint submission, admin response, and status tracking. Existing systems were either too complex or lacked essential features like real-time updates and communication. The investigation concluded that a customized Django-based solution would best serve the needs of the users.

### ****3.3 Feasibility Study****

* **Technical Feasibility**: The system is feasible using Python and Django, which support rapid web development. Existing frameworks and libraries make implementation practical.
* **Operational Feasibility**: The system is user-friendly and easy to integrate into institutions, requiring minimal training for both users and administrators.
* **Economic Feasibility**: As it is built using open-source tools and technologies, the development cost is minimal, making it economically viable.

### ****3.4 Project Planning****

The project is planned in multiple phases:

1. Requirement gathering and analysis
2. System design (UI/UX and backend architecture)
3. Implementation of user and admin modules
4. Integration and testing
5. Deployment and maintenance

### ****3.5 Project Scheduling****

* Week 1–2: Requirement Analysis and Design
* Week 3–4: User Module Implementation
* Week 5–6: Admin Module Implementation
* Week 7: Integration and Testing
* Week 8: Final Testing and Documentation

### ****3.6 Software Requirement Specification****

* **Frontend**: HTML, CSS, Bootstrap (for responsive UI)
* **Backend**: Python, Django Framework
* **Database**: SQLite (can be upgraded to PostgreSQL/MySQL)
* **Browser Support**: Chrome, Firefox, Edge
* **Platform**: Web-based, works on desktop and mobile browsers

### ****3.9 Functional Requirements****

* User Registration and Login
* Complaint Submission with category and description
* Complaint Status Tracking
* Admin Login and Dashboard
* View, Update, and Resolve Complaints
* Notification or status update mechanism

### ****3.10 Software Engineering Paradigm****

The project follows the **Incremental Model** of software development. This approach allows the core features to be developed first, followed by iterative additions and enhancements. Each increment builds on the previous one and adds functionality until the final system is complete.

### ****3.11 Data Model Description****

The data model includes the following key entities:

* **User**: Stores user credentials and profile information.
* **Complaint**: Stores complaint details like title, category, description,area, and status.
* **Admin**: Stores admin credentials and tracks actions on complaints.

**Water–fall Model with feedback mechanism for the proposed application as Software Engineering paradigm:**

**System**

**Analysis**

**System**

**Design**

**Testing**

**Implementation**

**& Maintenance**

**Cod**

**ing**

## ANALYSIS DIAGRAMS

**DFD (Data Flow Diagram):**

In the late 1970s *data-flow diagrams (DFDs)* were introduced and popularized for structured analysis and design (Game and Sarson 1979). DFDs show the flow of data from external entities into the system, showed how the data moved from one process to another, as well as its logical storage.

A **data flow diagram** (DFD) is a significant modeling technique for analyzing and constructing information processes. DFD literally means an illustration that explains the course or movement of information in a process. DFD illustrates this flow of information in a process based on the inputs and outputs. A DFD can be referred to as a Process Model.

Data flow diagrams can be used to provide a clear representation of any business function. The technique starts with an overall picture of the business and continues by analyzing each of the functional areas of interest. This analysis can be carried out to precisely the level of detail required. The technique exploits a method called top-down expansion to conduct the analysis in a targeted way.

As the name suggests, Data Flow Diagram (DFD) is an illustration that explicates the passage of information in a process. A DFD can be easily drawn using simple symbols. Additionally, complicated processes can be easily automated by creating DFDs using easy-to-use, free downloadable diagramming tools. A DFD is a model for constructing and analyzing information processes. DFD illustrates the flow of information in a process depending upon the inputs and outputs. A DFD can also be referred to as a Process Model. A DFD demonstrates business or technical process with the support of the outside data saved, plus the data flowing from the process to another and the end results.

Additionally, a **DFD** can be utilized to visualize data processing or a structured design. A DFD illustrates technical or business processes with the help of the external data stored, the data flowing from a process to another, and the results.

A designer usually draws a context-level DFD showing the relationship between the entities inside and outside of a system as one single step. This basic DFD can be then disintegrated to a lower level diagram demonstrating smaller steps exhibiting details of the system that is being modeled. Numerous levels may be required to explain a complicated system.

### Process

The process shape represents a task that handles data within the application. The task may process the data or perform an action based on the data.

### Multiple Processes

The multiple process shape is used to present a collection of sub processes. The multiple processes can be broken down into its sub processes in another DFD.

### External Entity

The external entity shape is used to represent any entity outside the application that interacts with the application via an entry point.

### Data Flow

The data flow shape represents data movement within the application. The direction of the data movement is represented by the arrow.

### Data Store

The data store shape is used to represent locations where data is stored. Data stores do not modify the data, they only store data.

### Privilege Boundary

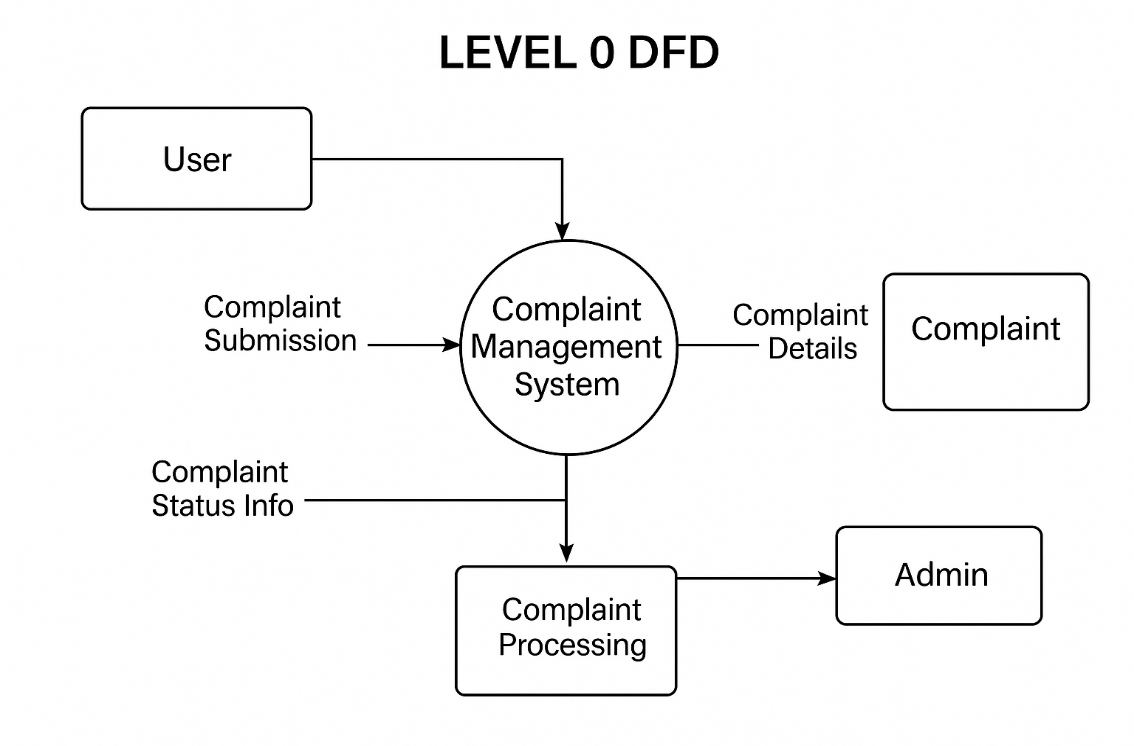
The privilege boundary shape is used to represent the change of privilege levels as the data flows through the application.

**Examples of Data Flow Diagrams -** These examples demonstrate how to draw data flow diagram.

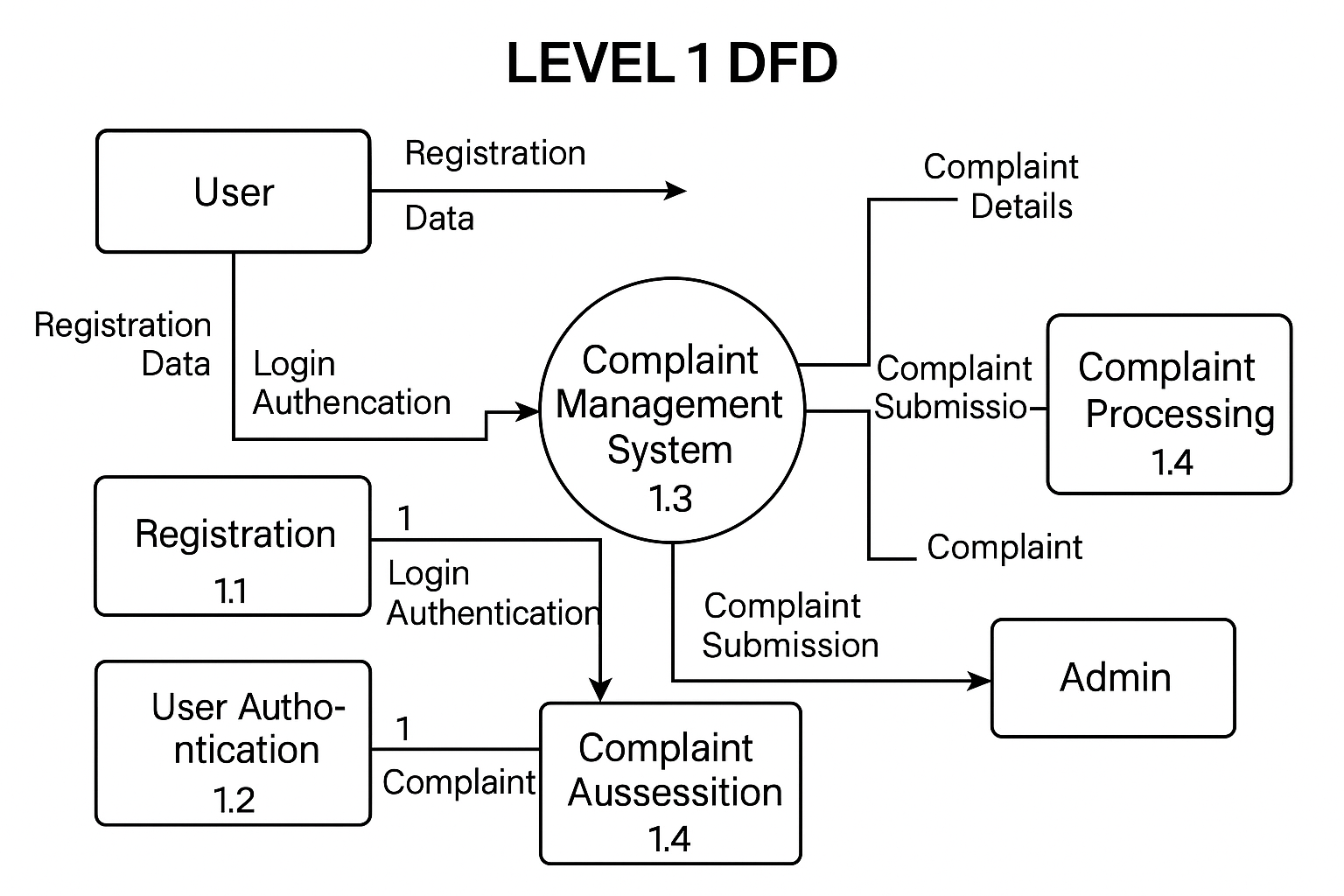
Before it was eventually replaced, a copy machine suffered frequent paper jams and became a notorious troublemaker. Often, a problem could be cleared by simply opening and closing the access panel. Someone observed the situation and flowcharted the troubleshooting procedure used by most people.

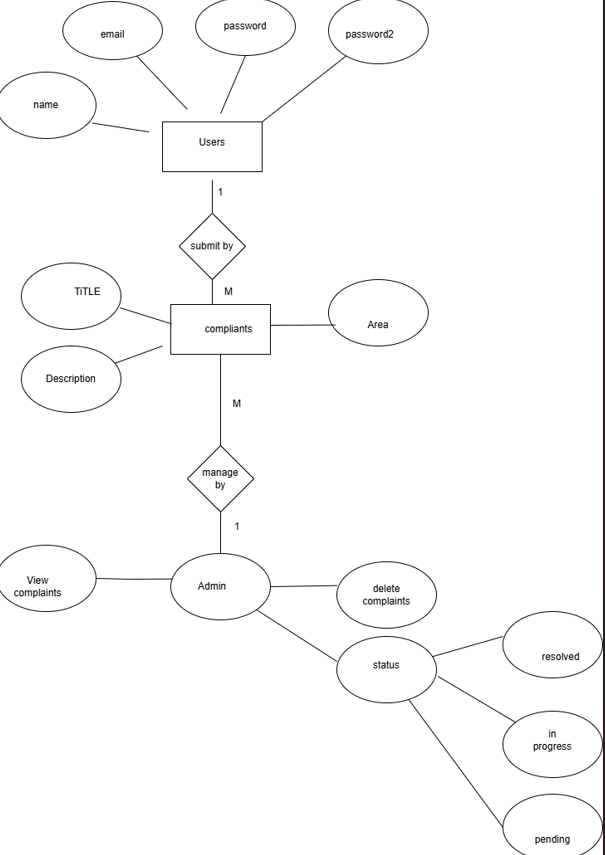
**System design /UML Diagrams**

## CONTEXT LEVEL DIAGRAM

0st Level Data Flow Diagram 

1st Level Data Flow Diagram





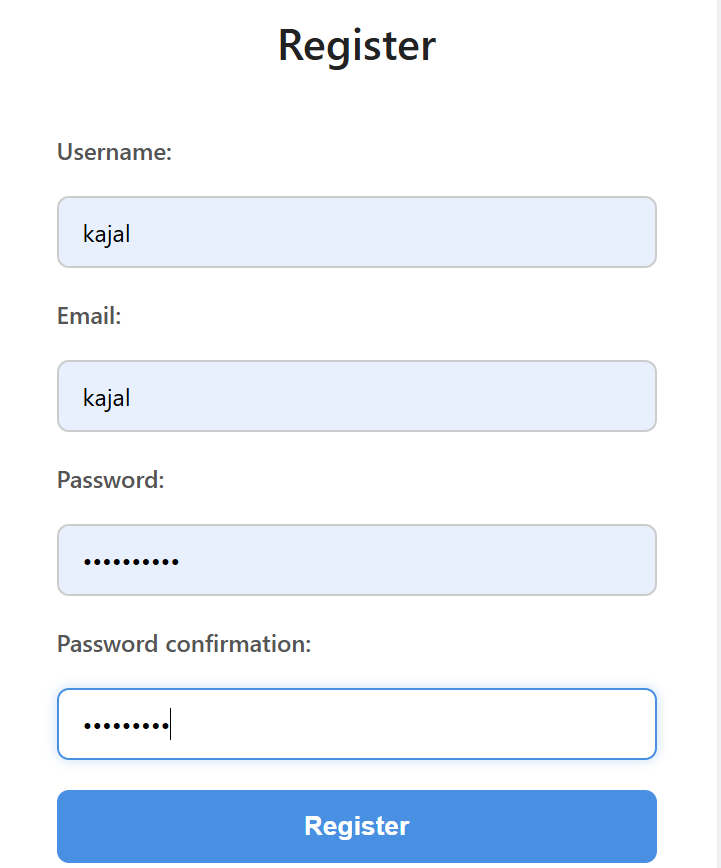
**MODULES AND THEIR DESCRIPTION**

**MODULES AND THEIR DESCRIPTION**

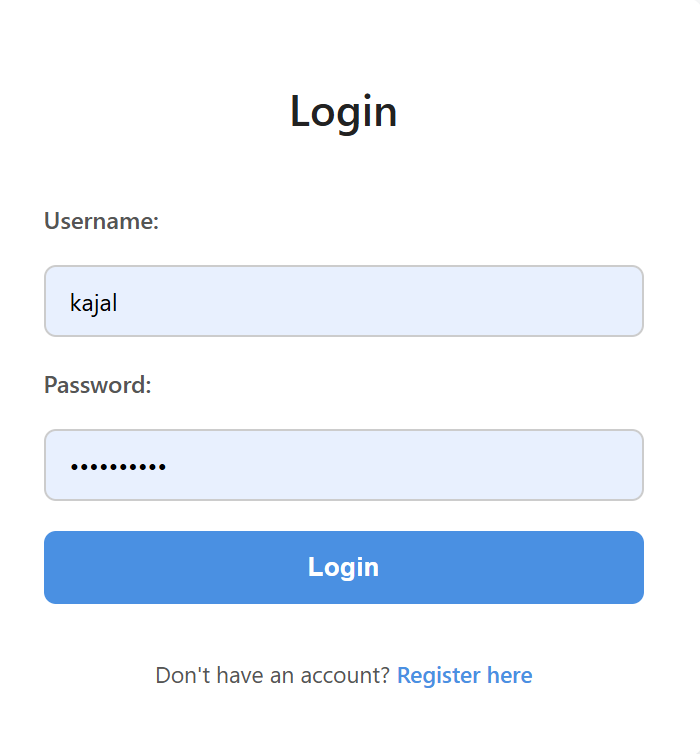
Modularization refers to breaking down the application into smaller, independent modules. Each module handles a specific functionality and communicates with others to create a complete working system

|  |  |
| --- | --- |
| Module Name | Description |
| **User Module** | Handles user registration, login, logout, and session management. |
| **Complaint Module** | Allows users to submit complaints and view their own complaints. |
| **Admin Module** | Lets admin view, update status, and manage all user complaints. |
| **Authentication Module** | Manages secure user authentication and permission handling. |
| **Dashboard Module** | Displays relevant data to users and admins after login. |

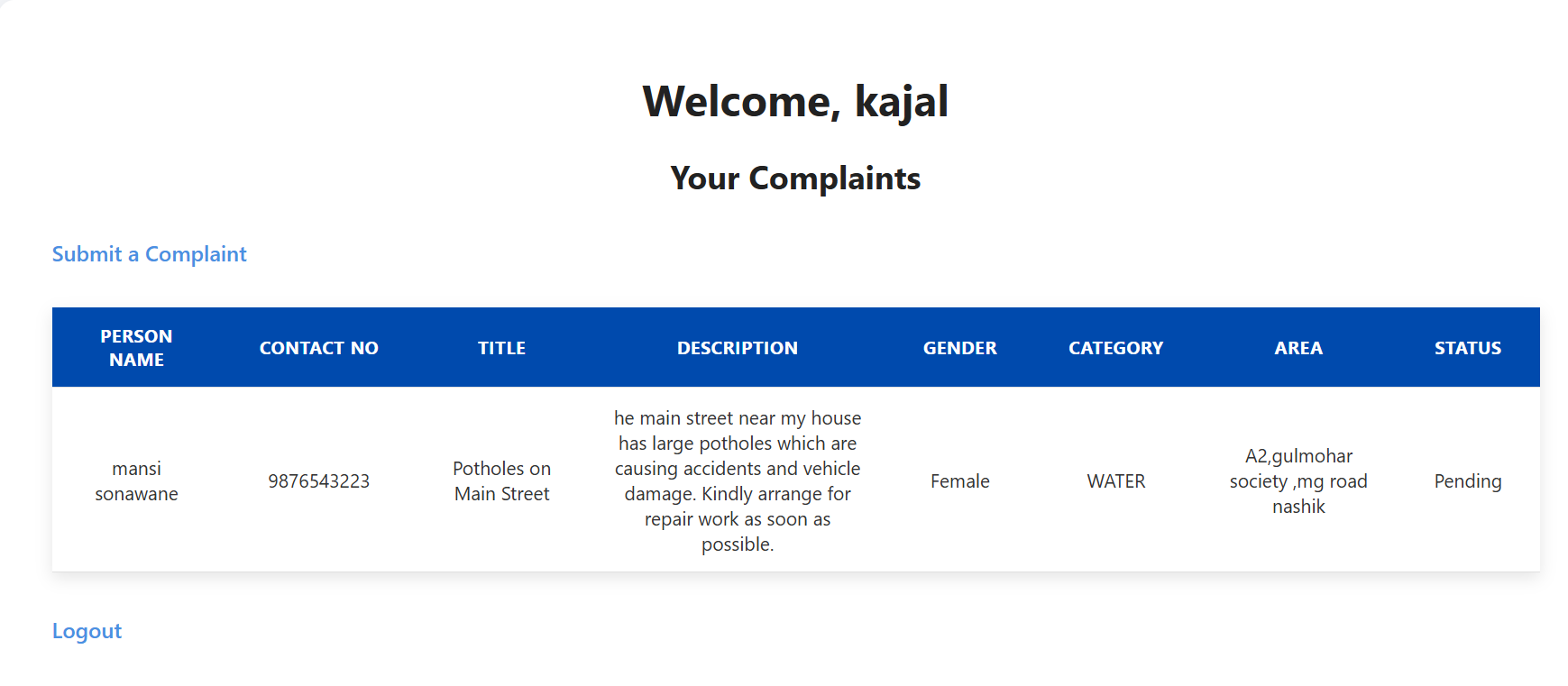
INPUT SCREENS

***Register Page :***

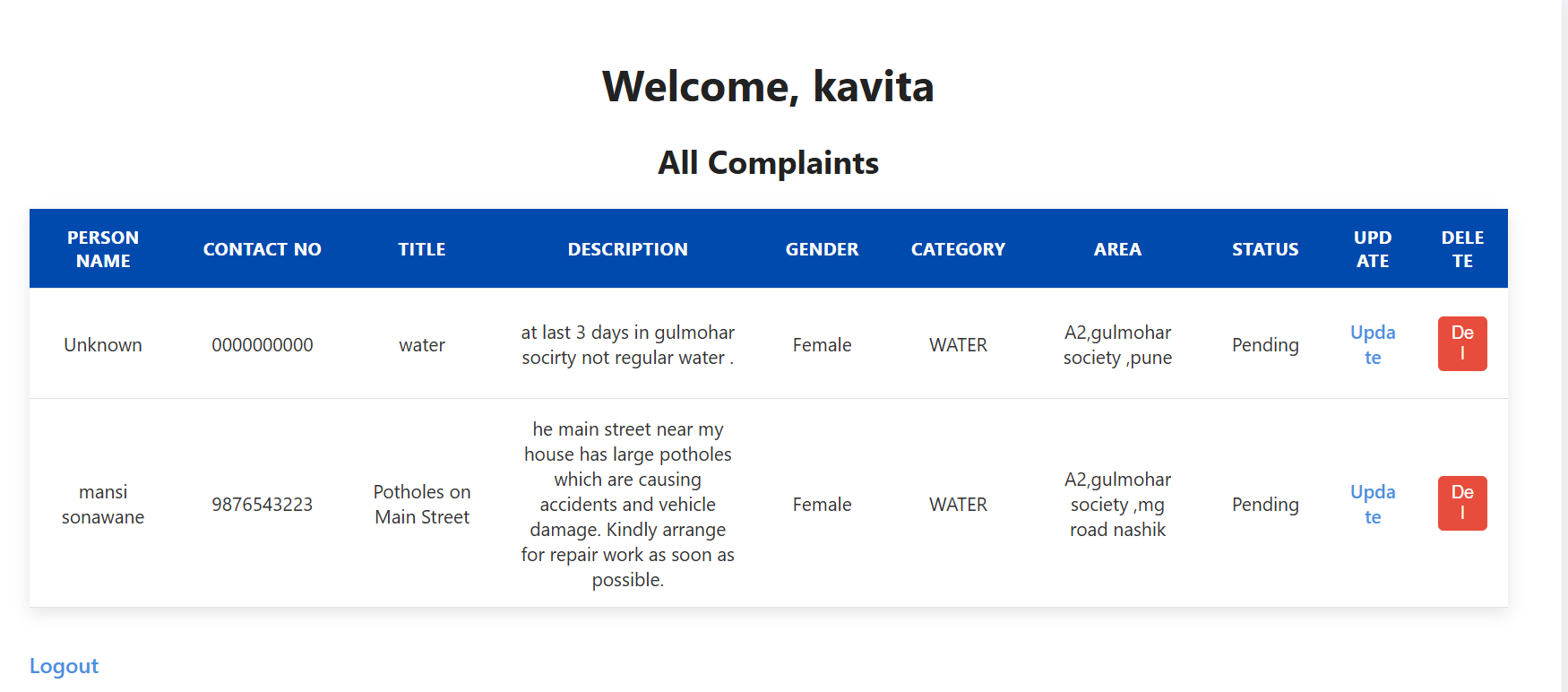
***Login page:***

******

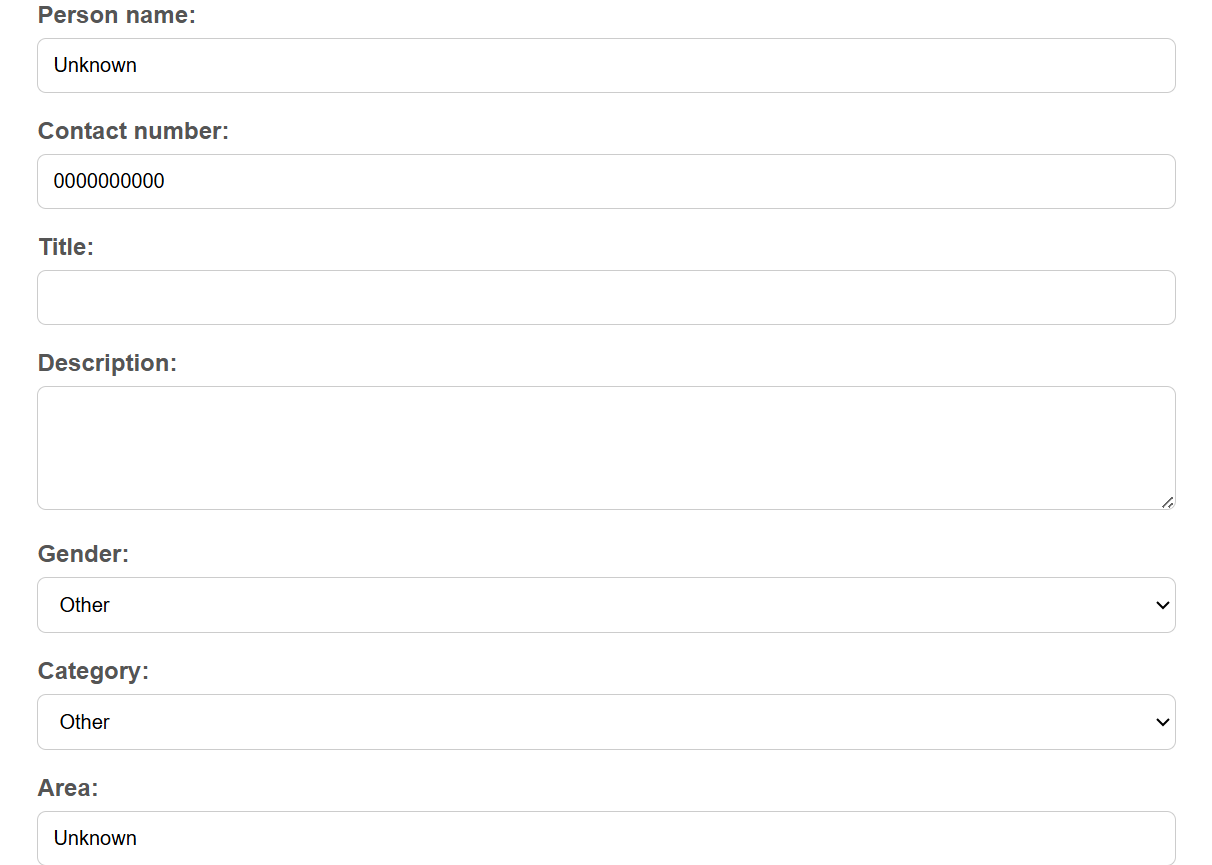
# *User dashboard page:*



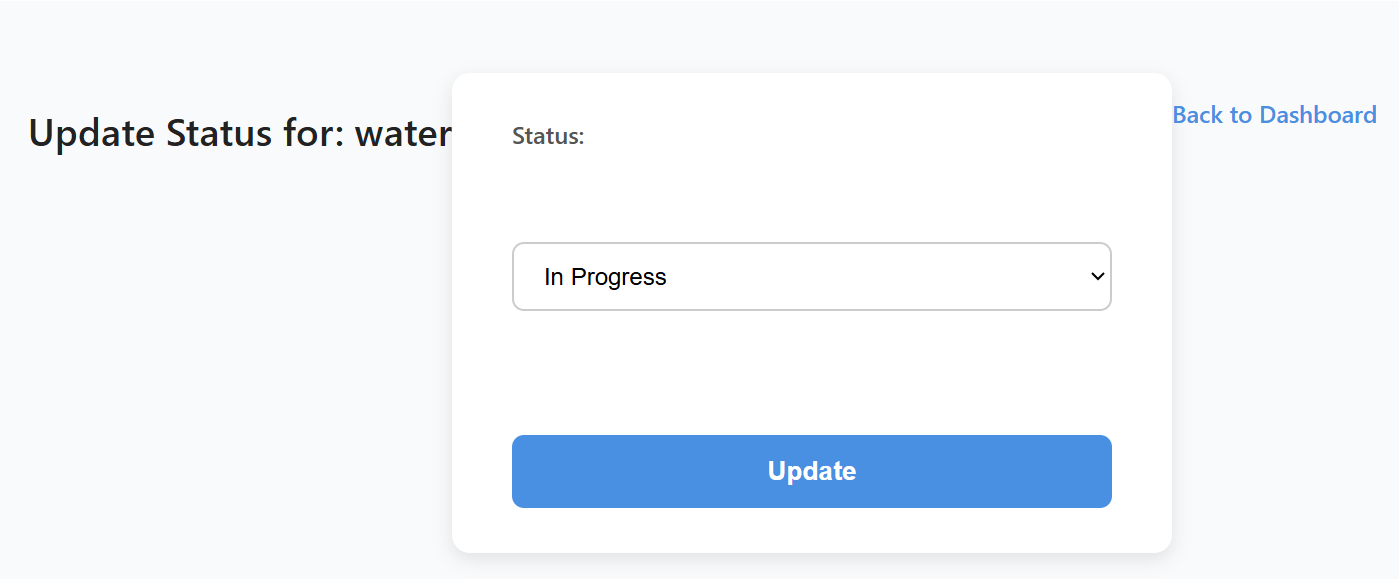
# *Admin dashboard page:*



**Compliant submit page**:



**Update Status page:**

****

**Coding**

**Views.py**

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login, logout

from django.contrib.auth.decorators import login\_required, user\_passes\_test

from .forms import ComplaintForm, UserRegisterForm

from .models import Complaint

@login\_required

def dashboard(request):

if request.user.username == 'kavita':

# Admin sees all complaints

complaints = Complaint.objects.all()

else:

# Normal users see only their own complaints

complaints = Complaint.objects.filter(user=request.user)

return render(request, 'dashboard.html', {'complaints': complaints})

def register\_view(request):

if request.method == 'POST':

username = request.POST['username']

password = request.POST['password']

if User.objects.filter(username=username).exists():

return render(request, 'register.html', {'error': 'Username already exists'})

User.objects.create\_user(username=username, password=password)

return redirect('login')

return render(request, 'register.html')

def is\_admin(user):

return user.is\_superuser

def home(request):

return redirect('login')

def register\_view(request):

if request.method == 'POST':

form = UserRegisterForm(request.POST)

if form.is\_valid():

user = form.save(commit=False)

user.set\_password(form.cleaned\_data['password'])

user.save()

return redirect('login')

else:

form = UserRegisterForm()

return render(request, 'register.html', {'form': form})

def login\_view(request):

if request.method == 'POST':

user = authenticate(username=request.POST['username'], password=request.POST['password'])

if user:

login(request, user)

return redirect('dashboard')

return render(request, 'login.html')

def logout\_view(request):

logout(request)

return redirect('login')

@login\_required

def dashboard(request):

if request.user.is\_superuser:

complaints = Complaint.objects.all()

else:

complaints = Complaint.objects.filter(user=request.user)

return render(request, 'dashboard.html', {'complaints': complaints})

@login\_required

def submit\_complaint(request):

return render(request, 'complaint\_form.html', {'form': form})

if request.method == 'POST':

form = ComplaintForm(request.POST)

if form.is\_valid():

complaint = form.save(commit=False)

complaint.user = request.user

complaint.save()

return redirect('dashboard')

else:

form = ComplaintForm()

return render(request, 'complaint\_form.html', {'form': form})

@user\_passes\_test(is\_admin)

def update\_status(request, complaint\_id):

complaint = Complaint.objects.get(id=complaint\_id)

if request.method == 'POST':

complaint.status = request.POST['status']

complaint.save()

return redirect('dashboard') sett

return render(request, 'update\_status.html', {'complaint': complaint})

**Models.py**

from django.db import models

from django.contrib.auth.models import User

class Complaint(models.Model):

STATUS\_CHOICES = [

('Pending', 'Pending'),

('In Progress', 'In Progress'),

('Resolved', 'Resolved'),

]

GENDER\_CHOICES = [

('Male', 'Male'),

('Female', 'Female'),

('Other', 'Other'),

]

CATEGORY\_CHOICES = [

('WATER', 'Water'),

('Hostel', 'Hostel'),

('Elctricity', 'Elctricity'),

('Other', 'Other'),

]

user = models.ForeignKey(User, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

description = models.TextField()

gender = models.CharField(max\_length=10, choices=GENDER\_CHOICES, default='Other')

category = models.CharField(max\_length=50, choices=CATEGORY\_CHOICES, default='Other')

date\_created = models.DateTimeField(auto\_now\_add=True)

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='Pending')

def \_\_str\_\_(self):

return f"{self.title} - {self.user.username}"

**urls.py**

from django.urls import path

from . import views

urlpatterns = [

path('', views.home, name='home'),

path('register/', views.register\_view, name='register'),

path('login/', views.custom\_login\_view, name='login'),

path('logout/', views.logout\_view, name='logout'),

path('dashboard/', views.dashboard, name='dashboard'),

path('submit/', views.submit\_complaint, name='submit\_complaint'),

path('update/<int:complaint\_id>/', views.update\_status, name='update\_status'),

]

<!-- complaints/templates/submit\_complaint.html -->

<!DOCTYPE html>

<html>

<head>

<title>Submit Complaint</title>

</head>

<body>

<h2>Submit Complaint</h2>

<form method="post">

{% csrf\_token %}

<label>Title:</label><br>

<input type="text" name="title" required><br><br>

<label>Description:</label><br>

<textarea name="description" required></textarea><br><br>

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

</form>

<p><a href="{% url 'dashboard' %}">Back to Dashboard</a></p>

</body>

</html>

**<!-- complaints/templates/dashboard.html -->**

<!DOCTYPE html>

<html>

<head>

<title>Dashboard</title>

{% load static %}

<link rel="stylesheet" href="{% static 'complaints/style.css' %}">

</head>

<body>

<div class="container">

<h2>Welcome, {{ user.username }}</h2>

<p>

<a href="{% url 'submit\_complaint' %}">Submit a Complaint</a> |

<a href="{% url 'logout' %}">Logout</a>

</p>

<hr>

<h3>View Complaints:</h3>

{% if user.username == "admin" %}

<p><strong>All Complaints:</strong></p>

{% else %}

<p><strong>Your Submitted Complaints:</strong></p>

{% endif %}

<ul>

{% for complaint in complaints %}

<li>

<strong>{{ complaint.title }}</strong><br>

{{ complaint.description }}<br>

Status: {{ complaint.status }}

{% if user.username == "admin" %}

<br><a href="{% url 'update\_status' complaint.id %}">Update Status</a>

{% endif %}

</li>

{% empty %}

<li>No complaints found.</li>

{% endfor %}

</ul>

</div>

</body>

</html>

<!-- complaints/templates/login.html -->

<!DOCTYPE html>

<html>

<head>

<title>Login - Complaint Management</title>

</head>

<body>

<h2>Login</h2>

<form method="post">

{% csrf\_token %}

<label>Username:</label><br>

<input type="text" name="username" required><br><br>

<label>Password:</label><br>

<input type="password" name="password" required><br><br>

<button type="submit">Login</button>

</form>

<p>Don't have an account? <a href="{% url 'register' %}">Register here</a></p>

</body>

</html>

<!DOCTYPE html>

<html>

<head>

<title>User Registration</title>

</head>

<body>

<h2>Register</h2>

<form method="post">

{% csrf\_token %}

<label>Username:</label><br>

<input type="text" name="username" required><br><br>

<label>Password:</label><br>

<input type="password" name="password" required><br><br>

<button type="submit">Register</button>

</form>

<p>Already have an account? <a href="{% url 'login' %}">Login</a></p>

</body>

</html>

<!-- complaints/templates/submit\_complaint.html -->

<!DOCTYPE html>

<html>

<head>

<title>Submit Complaint</title>

</head>

<body>

<h2>Submit Complaint</h2>

<form method="post">

{% csrf\_token %}

<label>Title:</label><br>

<input type="text" name="title" required><br><br>

<label>Description:</label><br>

<textarea name="description" required></textarea><br><br>

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

</form>

<p><a href="{% url 'dashboard' %}">Back to Dashboard</a></p>

</body>

</html>

<!-- complaints/templates/update\_status.html -->

<!DOCTYPE html>

<html>

<head>

<title>Update Complaint Status</title>

</head>

<body>

<h2>Update Status for: {{ complaint.title }}</h2>

<form method="post">

{% csrf\_token %}

<label>Status:</label><br>

<select name="status">

<option value="Pending" {% if complaint.status == "Pending" %}selected{% endif %}>Pending</option>

<option value="In Progress" {% if complaint.status == "In Progress" %}selected{% endif %}>In Progress</option>

<option value="Resolved" {% if complaint.status == "Resolved" %}selected{% endif %}>Resolved</option>

</select><br><br>

<button type="submit">Update</button>

</form>

<p><a href="{% url 'dashboard' %}">Back to Dashboard</a></p>

</body>

</html>

Models.py

from django.db import models

from django.contrib.auth.models import User

class Complaint(models.Model):

STATUS\_CHOICES = [

('Pending', 'Pending'),

('In Progress', 'In Progress'),

('Resolved', 'Resolved'),

]

user = models.ForeignKey(User, on\_delete=models.CASCADE)

title = models.CharField(max\_length=200)

description = models.TextField()

date\_created = models.DateTimeField(auto\_now\_add=True)

status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='Pending')

def \_\_str\_\_(self):

return f"{self.title} - {self.user.username}"

urls.py

from django.urls import path

from . import views

path('', views.home, name='home'),

urlpatterns = [

path('', views.home, name='home'),

path('register/', views.register\_view, name='register'),

path('login/', views.login\_view, name='login'),

path('logout/', views.logout\_view, name='logout'),

path('dashboard/', views.dashboard, name='dashboard'),

path('submit/', views.submit\_complaint, name='submit\_complaint'),

path('update/<int:complaint\_id>/', views.update\_status, name='update\_status'),

path('', views.submit\_complaint, name='submit\_complaint'),

]

Setting.py

"""

Django settings for complaint\_system project.

Generated by 'django-admin startproject' using Django 5.2.

For more information on this file, see

https://docs.djangoproject.com/en/5.2/topics/settings/

For the full list of settings and their values, see

https://docs.djangoproject.com/en/5.2/ref/settings/

"""

from pathlib import Path

import os

# Build paths inside the project like this: BASE\_DIR / 'subdir'.

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

# Quick-start development settings - unsuitable for production

# See https://docs.djangoproject.com/en/5.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!

SECRET\_KEY = 'django-insecure-x2ep\*mgvd5&#l3!9obe&1esvgv2(cg$nhmimr\_ts5#1p&gq0i0'

# SECURITY WARNING: don't run with debug turned on in production!

DEBUG = True

ALLOWED\_HOSTS = []

# Application definition

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'complaints',

]

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]

ROOT\_URLCONF = 'complaint\_system.urls'

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

'DIRS': [BASE\_DIR / 'complaints' / 'templates'],

# Use pathlib to build path

'APP\_DIRS': True,

'OPTIONS': {

'context\_processors': [

'django.template.context\_processors.request',

'django.contrib.auth.context\_processors.auth',

'django.contrib.messages.context\_processors.messages',

],

},

},

]

WSGI\_APPLICATION = 'complaint\_system.wsgi.application'

# Database

# https://docs.djangoproject.com/en/5.2/ref/settings/#databases

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3',

'NAME': BASE\_DIR / 'db.sqlite3',

}

}

# Password validation

# https://docs.djangoproject.com/en/5.2/ref/settings/#auth-password-validators

AUTH\_PASSWORD\_VALIDATORS = [

{

'NAME': 'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.MinimumLengthValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.CommonPasswordValidator',

},

{

'NAME': 'django.contrib.auth.password\_validation.NumericPasswordValidator',

},

]

# Internationalization

# https://docs.djangoproject.com/en/5.2/topics/i18n/

LANGUAGE\_CODE = 'en-us'

TIME\_ZONE = 'UTC'

USE\_I18N = True

USE\_TZ = True

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/5.2/howto/static-files/

STATIC\_URL = 'static/'

# Default primary key field type

# https://docs.djangoproject.com/en/5.2/ref/settings/#default-auto-field

DEFAULT\_AUTO\_FIELD = 'django.db.models.BigAutoField'

**Css file**

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

background-color: #f5f7fa;

color: #333;

line-height: 1.6;

padding: 40px 20px;

min-height: 100vh;

display: flex;

justify-content: center;

align-items: flex-start;

}

/\* Container \*/

.container {

background-color: #ffffff;

max-width: 700px;

width: 100%;

padding: 30px 40px;

border-radius: 12px;

box-shadow: 0 4px 20px rgba(0, 0, 0, 0.1);

transition: box-shadow 0.3s ease;

}

.container:hover {

box-shadow: 0 6px 28px rgba(0, 0, 0, 0.15);

}

/\* Headings \*/

h2, h3 {

font-weight: 600;

margin-bottom: 24px;

color: #222;

text-align: center;

font-size: 1.5rem;

}

/\* Form \*/

form {

display: flex;

flex-direction: column;

gap: 18px;

margin-bottom: 20px;

}

/\* Labels \*/

label {

font-weight: 500;

color: #555;

margin-bottom: 6px;

display: inline-block;

font-size: 1rem;

}

/\* Inputs and textarea \*/

input[type="text"],

input[type="password"],

input[type="email"],

select,

textarea {

padding: 12px 16px;

border: 1px solid #e0e0e0; /\* very light, thin border \*/

border-radius: 8px;

font-size: 1rem;

background-color: #fafafa;

color: #555;

font-family: inherit;

resize: vertical;

width: 100%;

box-shadow: inset 0 0 5px rgba(0,0,0,0.03);

transition: border-color 0.3s ease, box-shadow 0.3s ease;

}

input[type="text"]:focus,

input[type="password"]:focus,

input[type="email"]:focus,

select:focus,

textarea:focus {

outline: none;

border-color: #7aa9f7;

box-shadow: 0 0 8px rgba(122, 169, 247, 0.5);

}

/\* Textarea \*/

textarea {

min-height: 100px;

}

/\* Button \*/

button {

padding: 14px 0;

background-color: #4a90e2;

color: #fff;

font-size: 1.1rem;

font-weight: 600;

border: none;

border-radius: 8px;

cursor: pointer;

transition: background-color 0.3s ease, transform 0.15s ease;

}

button:hover {

background-color: #357ab8;

transform: translateY(-2px);

}

button:active {

transform: translateY(0);

}

/\* Links \*/

a {

color: #4a90e2;

text-decoration: none;

font-weight: 600;

transition: color 0.3s ease;

}

a:hover {

color: #357ab8;

text-decoration: underline;

}

/\* Lists \*/

ul {

list-style-type: none;

padding-left: 0;

margin-top: 12px;

}

li {

background-color: #f1f3f5;

border: 1.5px solid #ddd;

padding: 16px 22px;

margin-bottom: 14px;

border-radius: 10px;

transition: background-color 0.3s ease;

color: #333;

}

li:hover {

background-color: #e6ebf1;

}

/\* Status badges \*/

.status {

display: inline-block;

padding: 6px 14px;

border-radius: 20px;

font-weight: 600;

font-size: 0.85rem;

color: white;

margin-top: 8px;

}

.status.Pending {

background-color: #f5a623; /\* amber \*/

}

.status.InProgress {

background-color: #4a90e2; /\* blue \*/

}

.status.Resolved {

background-color: #2ecc71; /\* green \*/

}

/\* Responsive \*/

@media (max-width: 640px) {

body {

padding: 20px 10px;

}

.container {

padding: 25px;

}

button {

font-size: 1rem;

}

}

**Validation**

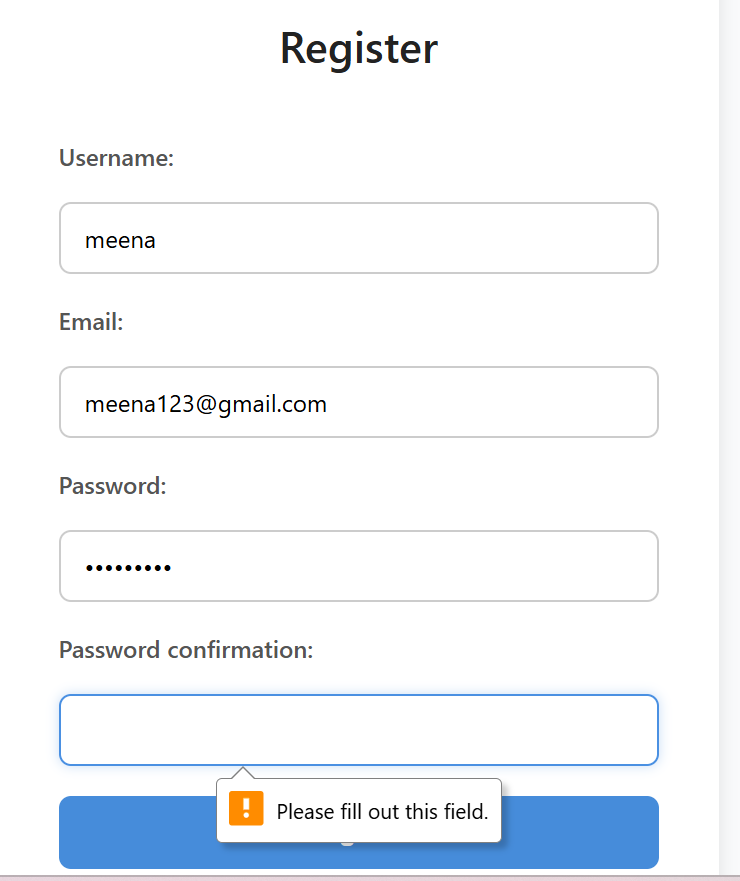
## ✦ Types of Validation

Validation checks in our Complaint Management System include:

* 🟢 **Empty Field Validation**
* 🟢 **Email Format Validation**

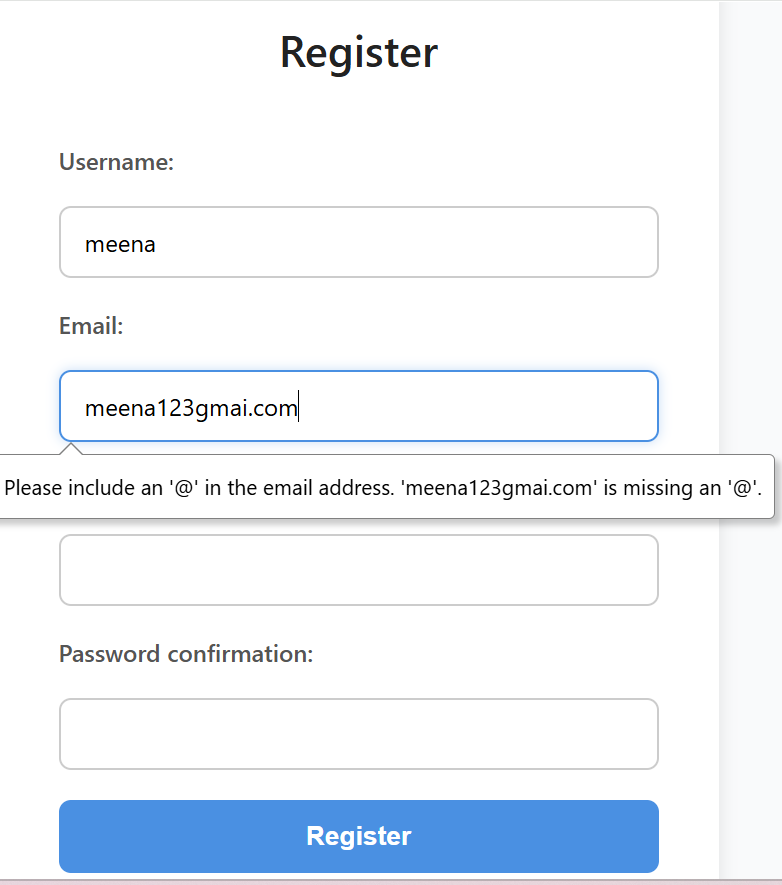
### ✅ Empty Field Validation

This validation ensures that no required input fields are left blank during form submissions. It helps in maintaining data integrity and user accountability.



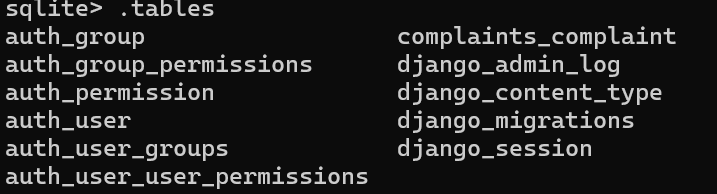
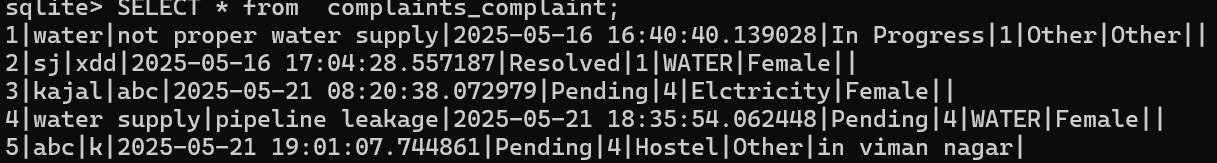
### ✅ Email Format Validation

This validation checks if the entered email address follows the correct format (e.g., includes '@' and domain name). It prevents invalid or incorrectly typed emails from being saved.



**Database:**

show the all tables:



### TESTING

### TESTING STRATEGY

### Testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test-Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

### Software testing can also be stated as the process of validating and verifying that a software program/application/product:

### 1. meets the business and technical requirements that guided its design and development;

### 2. works as expected; and

### 3. Can be implemented with the same characteristics.

### Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

### Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

### Testing can never completely identify all the defects within software. Instead, it furnishes a criticism or comparison that compares the state and behavior of the product against oracles-principles or mechanisms by which someone might recognize a

### problem. These oracles may include (but are not limited to) specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

### Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. Software testing is the process of attempting to make this

### assessment.

### Testing methods

### Software testing methods are traditionally divided into and white- and black-box testing. These two approaches are used to describe the point of view that a test engineer takes

### when designing test cases.

### Unit testing

### Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

### In unit testing, the smallest unit of the software, i.e. modules, is tested. This test focuses on the internal process logic and data structure within the boundaries of a component. These types of tests are usually written by developers as they work on code

### (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other. Unit testing is also called component testing.

### Integration testing

### Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localized more quickly and fixed.

### Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

### In integration testing, all the modules are tested as a whole. Database connectivity is checked which means data can be retrieved from the database for displaying reports.

### System testing

### System testing tests a completely integrated system to verify that it meets its requirements.

### System testing is actually a series of different tests whose primary purpose is to fully exercise the computer based system. Although each test has a different purpose but all works to verify that all system elements have been properly integrated and perform allocated functions.

### This kind of testing checks whether the whole system is working correctly or not. This checks that the outputs that are being generated correspond to the input that was supplied by the user. This checks the complete working of the project.

### Conclusion

The Complaint Management System successfully fulfills its purpose of providing a user-friendly platform where users can easily submit complaints regarding various issues, while administrators can efficiently manage and resolve these complaints. The system incorporates essential features such as user registration and authentication, complaint submission with detailed information (including person name, contact number, title, description, gender, category, and area), and a dynamic dashboard that displays complaints in an organized manner.

The dashboard has been enhanced with **Update** and **Delete** functionalities, allowing users and administrators to maintain accurate complaint records. Administrators can update complaint statuses to keep users informed about the progress, and authorized users can delete complaints when necessary, ensuring data remains relevant and clutter-free.

By implementing role-based access controls, the system guarantees that sensitive operations like status updates and deletions are only performed by authorized personnel. This enhances the security and integrity of the complaint data.

Overall, this system streamlines complaint handling processes, reduces manual workload, and promotes transparency between users and the administration. It serves as an effective communication channel that can significantly improve response times and customer satisfaction. With further enhancements, this platform can be scaled and customized to meet the needs of diverse organizations and communities.

## ****Future Scope****

The current Complaint Management System provides a streamlined platform for users to submit complaints and track their statuses. While the core features are functional and efficient, there are numerous possibilities for future improvements and extensions. These enhancements can significantly improve usability, efficiency, and scalability.

### ****Potential Enhancements:****

1. **Multi-User Role Management**  
   Introduce roles such as moderator, department officer, or super admin, each with specific permissions and dashboards.
2. **Categorization and Department Routing**  
   Automatically route complaints to the correct department based on predefined categories (e.g., IT, Maintenance, HR).
3. **File Attachment Support**  
   Allow users to upload images, documents, or videos as evidence when submitting a complaint.
4. **Notification System**  
   Integrate email or SMS alerts to notify users and admins of complaint updates or resolutions.
5. **Mobile Application Development**  
   Extend the system with an Android/iOS app to improve accessibility and usage across platforms.
6. **Search and Filter Functionality**  
   Enable users and admins to search complaints by date, status, or keywords for better navigation and analysis.
7. **Analytics and Reporting Dashboard**  
   Add an admin dashboard to display complaint statistics (e.g., total complaints, resolved/unresolved status, department-wise stats).
8. **Feedback and Rating System**  
   Allow users to rate the resolution of their complaints and provide feedback to improve services.
9. **Escalation System**  
   Automatically escalate unresolved complaints to higher authorities after a certain time threshold.
10. **AI-Powered Chatbot**  
    Implement a chatbot for assisting users in submitting complaints, checking status, or resolving common queries instantly.

### BIBLIOGRAPHY

### ****Books:****

1. **Fundamentals of Software Engineering** – Rajib Mall
2. **An Introduction to Database Systems** (4th ed.) – C.J. Date, Addison-Wesley
3. **Introduction to Database Management Systems** – Bipin C. Desai
4. **MySQL: The Complete Reference** – Kevin Loney and George Koch
5. **Learning Python** – Mark Lutz
6. **Web Development with Django Cookbook** – Jake Kronika and Aidas Bendoraitis
7. **HTML and CSS: Design and Build Websites** – Jon Duckett
8. **Python Crash Course** – Eric Matthes

### ****Websites and Online Resources:****

* <https://docs.djangoproject.com> — Django official documentation
* <https://www.w3schools.com> — HTML/CSS/Python tutorials
* <https://stackoverflow.com> — Community Q&A and debugging help
* <https://developer.mozilla.org> — HTML/CSS/JavaScript documentation (MDN)
* <https://phptutorial.com> — (As per your image)