

# **NAAN MUDHALVAN PROJECT**

## **MERN Stack Powered By Mongodb**

**PROJECT TITLE:**

### **Online Complaint Registration And** **Management System**

**Submitted by the final year CSE 'A' team members**

<b>AXIN RICCO R</b>	<b>311421104007</b>
<b>NIKHIL SRINIVASAN S</b>	<b>311421104057</b>
<b>MUTHU BABU S</b>	<b>311421104051</b>
<b>PRASANNA VELAN V</b>	<b>311421104309</b>



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**MEENAKSHI COLLEGE OF ENGINEERING**

**12, VEMBULIAMMAN KOVIL STREET,**

**KK NAGAR WEST, CHENNAI-600078.**

# **Online Complaint Registration And Management System**

## **INTRODUCTION:**

**PROJECT TITLE:** Online Complaint Registration And Management System

### **TEAM MEMBERS:**

1. AXIN RICCO R – DataBase Administrator
2. NIKHIL SRINIVASAN S – Backend
3. MUTHU BABU S - Frontend
4. PRASANNA VELAN V- Frontend

The **Online Complaint Registration and Management System** is a modern web-based solution designed to simplify and optimize the process of registering, tracking, and resolving complaints. In today's fast-paced world, individuals and organizations often encounter issues that require swift resolution. This system provides a centralized platform for users to raise their concerns and for organizations to handle them efficiently, fostering trust and enhancing satisfaction.

## **PROJECT OVERVIEW:**

### **PURPOSE:**

The Online Complaint Registration and Management System is designed to help organizations effectively handle customer complaints, providing a centralized platform for complaint submission, tracking, and resolution. The system aims to enhance user satisfaction by reducing response times, improving transparency, and facilitating direct communication between customers and agents. Additionally, it enables organizations to manage workloads efficiently, ensuring that each complaint is routed to the most suitable personnel for prompt resolution. This streamlined approach also helps companies adhere to industry regulations and data protection standards, ultimately fostering trust and loyalty among customers.

The purpose of the Online Complaint Registration and Management System is to provide a user-friendly platform that streamlines the process of filing, tracking, and resolving complaints. This system is designed to improve customer satisfaction by offering a centralized solution for managing complaints, ensuring timely responses, secure data handling, and efficient allocation of resources to resolve issues effectively. It also supports organizational compliance with industry standards and data protection regulations.

The Online Complaint Registration and Management System serves as a transformative solution for organizations to improve their complaint resolution processes, thereby elevating customer service quality and satisfaction. By automating complaint intake, assignment, tracking, and resolution, this system minimizes manual intervention, reduces response times, and enhances transparency for both users and agents. The platform fosters stronger customer relationships by making complaint handling efficient, secure, and interactive.

## FEATURES

### 1. Advanced User Account Management:

- Offers multiple user roles (e.g., customers, agents, admins) with role-specific permissions.
- Two-factor authentication (2FA) adds an extra layer of security for users logging in.

### 2. Intelligent Complaint Routing with Machine Learning:

- Leverages machine learning to classify complaints based on their content, automatically assigning them to the most suitable agents or departments.
- Predictive analytics help the system prioritize complaints that are likely to require more immediate attention.

### 3. Real-Time Tracking with Status Indicators:

- A live status indicator (e.g., "Under Review," "In Progress," "Resolved") provides users with clear, up-to-date information on the progress of their complaint.
- Agents and admins can also see the live status of each complaint, enabling better resource allocation and follow-up.

### 4. Agent Management and Performance Monitoring:

- Managers can monitor agent workloads, performance metrics (e.g., average response time, resolution rate), and feedback scores.
- Performance reports help identify top-performing agents and those needing additional support or training.

### 5. Automated SLA (Service Level Agreement) Compliance Monitoring:

- The system automatically tracks complaint resolution times to ensure adherence to SLAs.
- Alerts are triggered if SLA deadlines are approaching or have been missed, helping prevent delayed resolutions.

### 6. Analytics Dashboard for Comprehensive Reporting:

- A detailed analytics dashboard provides insights into complaint trends, agent performance, and resolution efficiency.
- Data visualizations (graphs, charts) make it easy to spot patterns and track KPIs, helping management make data-driven improvements.

### 7. Enhanced Security Features:

- Provides role-based access control, ensuring users only see data relevant to their role (e.g., agents can see only assigned complaints).
- Data encryption ensures sensitive information remains secure, both at rest and in transit.
- Activity logs track all actions within the platform, enabling audits to ensure compliance with data protection policies.

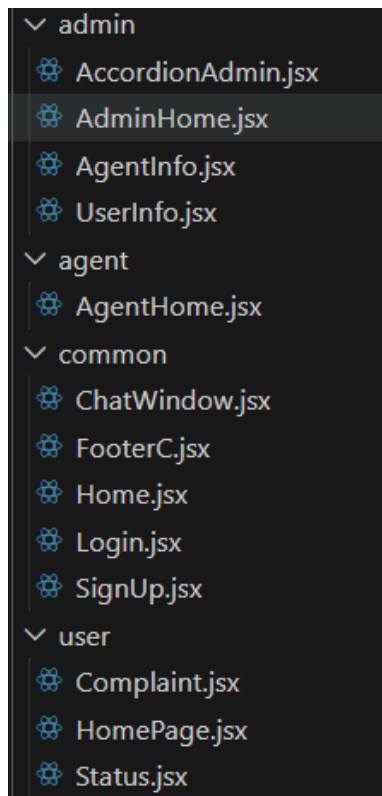
### 8. Feedback Collection and Analysis:

- After resolution, users are prompted to provide feedback, which can be analyzed to measure satisfaction and identify areas for improvement.
- A feedback scoring system helps the organization identify and address recurring issues.

## **ARCHITECTURE**

### **Frontend (React Architecture)**

The frontend is structured with React, utilizing a component-based architecture, which allows for reusability and modularity. The folders and files indicate a separation of concerns based on roles and common functionalities:

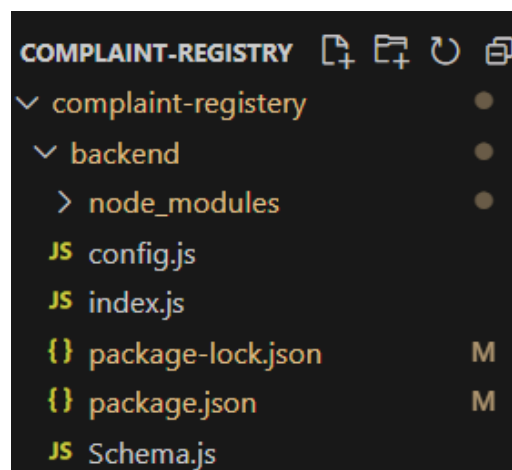


- **src/components Folder:**
  - **admin Folder:**
    - Contains components specific to admin functionalities.
    - **AccordionAdmin.jsx:** Likely used to display a list or detailed view of data in an expandable/collapsible format.
    - **AdminHome.jsx:** The main dashboard or homepage for admin users.
    - **AgentInfo.jsx** and **UserInfo.jsx:** Components to display information about agents and users, respectively.
  - **agent Folder:**
    - Contains components specific to the agent role.
    - **AgentHome.jsx:** Likely the dashboard or homepage for agents, where they can manage or respond to complaints.
  - **common Folder:**
    - Houses components shared by multiple user roles.

- **ChatWindow.jsx:** Likely a component for real-time chat, enabling communication between users and agents.
- **FooterC.jsx:** A common footer component, probably used across different pages.
- **Home.jsx:** Could be a generic home page component or a landing page.
- **Login.jsx** and **SignUp.jsx:** Components for user authentication, allowing users to log in or sign up.
- **user Folder:**
  - Contains components specific to the user role.
  - **Complaint.jsx:** Likely used for submitting a new complaint or viewing complaint details.
  - **HomePage.jsx:** The main page or dashboard for regular users, providing them with access to their complaints.
  - **Status.jsx:** Displays the status of complaints, allowing users to track the progress of their submissions.

## Backend (Node.js and Express Architecture)

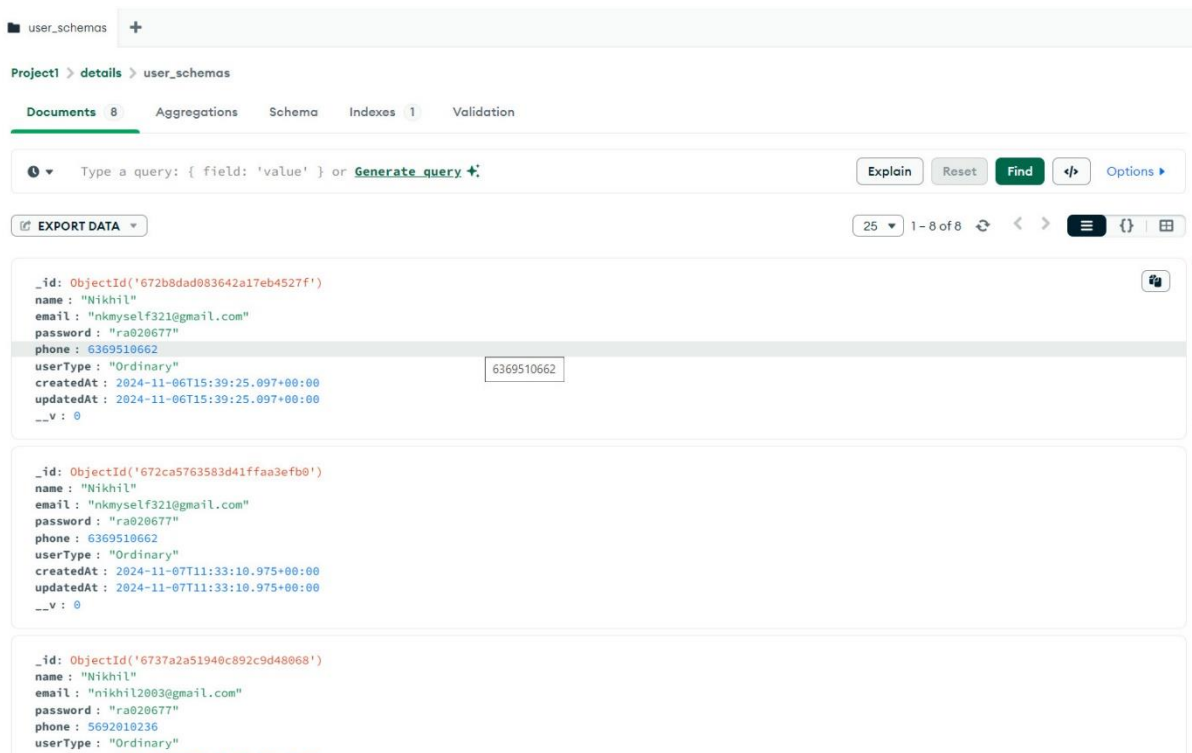
The backend architecture is built with Node.js and Express, providing RESTful APIs for the frontend to interact with:



- **index.js (Backend Entry Point):**
  - **Express Server Setup:** Initializes the Express app and defines the routes for different endpoints.
  - **Middleware:** Likely includes middlewares for JSON parsing, request logging, CORS, error handling, and authentication.
  - **Route Handling:** Imports and sets up various routes (e.g., for users, agents, complaints), each of which corresponds to a specific component on the frontend.

- **config.js:**
  - This file is typically used to store configuration settings such as database URLs, secret keys, and other environment-specific variables. It allows easy modification of these settings without changing the core code, supporting different environments (development, production).
- **Dependencies:**
  - **Express:** Manages HTTP requests and routing.
  - **Mongoose:** Used for interacting with MongoDB, with schemas defining data structure.
  - **JWT (JSON Web Token):** Likely used for authentication, securing routes based on user roles.

## Database (MongoDB and Schema Design)



The database structure seems to be defined in **Schema.js**:

- **MongoDB with Mongoose:**
  - **Schemas** define the structure of collections in MongoDB, ensuring data consistency.
  - The primary collections could include:
    - **Users:** Contains details like user credentials, roles (admin, agent, user), and contact information.

- **Complaints:** Stores complaint details such as description, category, status (open, in-progress, resolved), timestamps, and references to the user who submitted the complaint.
- **Messages or Chats:** If real-time chat is enabled, a collection for storing messages exchanged between users and agents, linked by complaint ID or user ID.

Project1 > details Refresh

Sort by: Collection Name ↑

View ☰ ⋮

<b>assigned_complaints</b>				
Storage size: 20.48 kB	Documents: 4	Avg. document size: 121.00 B	Indexes: 1	Total index size: 36.86 kB
<b>complaint_schemas</b>				
Storage size: 20.48 kB	Documents: 5	Avg. document size: 219.00 B	Indexes: 1	Total index size: 36.86 kB
<b>messages</b>				
Storage size: 20.48 kB	Documents: 5	Avg. document size: 129.00 B	Indexes: 1	Total index size: 36.86 kB
<b>user_schemas</b>				
Storage size: 20.48 kB	Documents: 8	Avg. document size: 181.00 B	Indexes: 1	Total index size: 36.86 kB

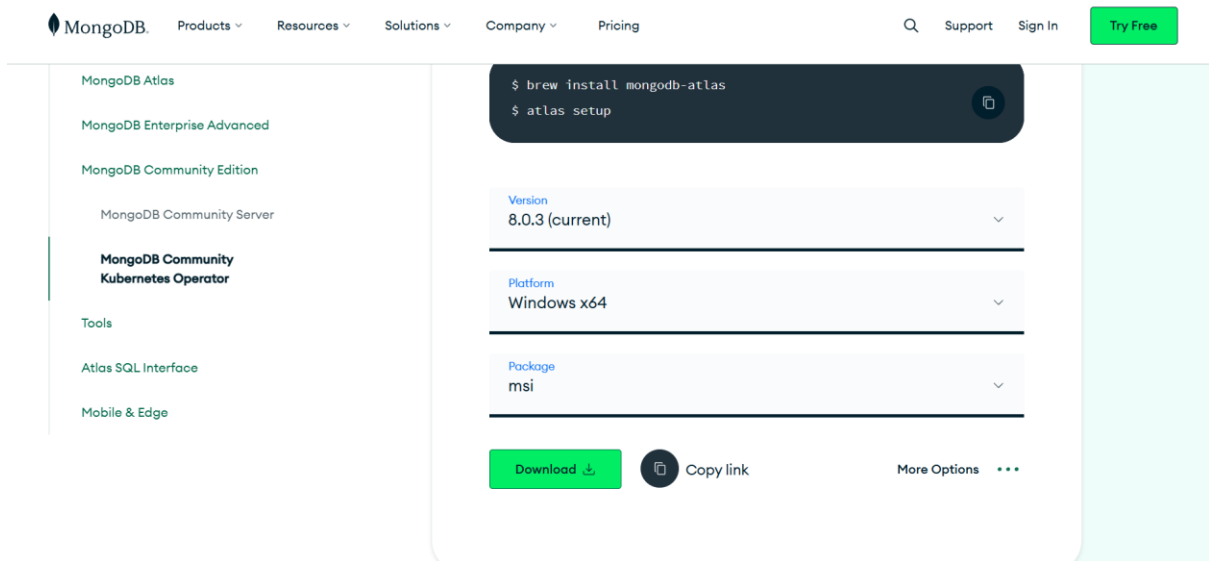
- **Data Interactions:**

- The backend APIs interact with MongoDB to perform CRUD (Create, Read, Update, Delete) operations on these collections.
- **Authentication and Authorization:** Tokens are generated and validated, with role-based access applied to ensure users can only access their data, while agents and admins have broader permissions.
- 

## Setup Instructions

### 1. Prerequisites

- **Node.js:** Required to run the server-side JavaScript - download from [Node.js official website](#).
- **MongoDB:** Database for storing user and complaint information -download from [MongoDB official website](#) .
- **Git:** Required for cloning the repository -download from [Git official website](#).



## 2. Installation Steps

### Step 1: Clone the Repository

- `git clone <https://github.com/awdhesh-student/complaint-registry.git>`
- `cd complaint-registry`

### Step 2: Install Dependencies

Navigate to the backend and frontend folders separately to install their dependencies.

#### For **Backend**:

- `cd backend`
- `npm install`

#### For **Frontend**:

- `cd ../frontend`
- `npm install`

### Step 3: Set Up Environment Variables

- In the backend, create a `.env` file inside the backend folder to store environment variables. Add the following variables, replacing the placeholders with your actual values:

### Step 4: Start the Application

To start both the frontend and backend servers:

#### For **Backend**:

- `cd backend`
- `npm start`



For **Frontend**:

- `cd ../frontend`
- `npm start`

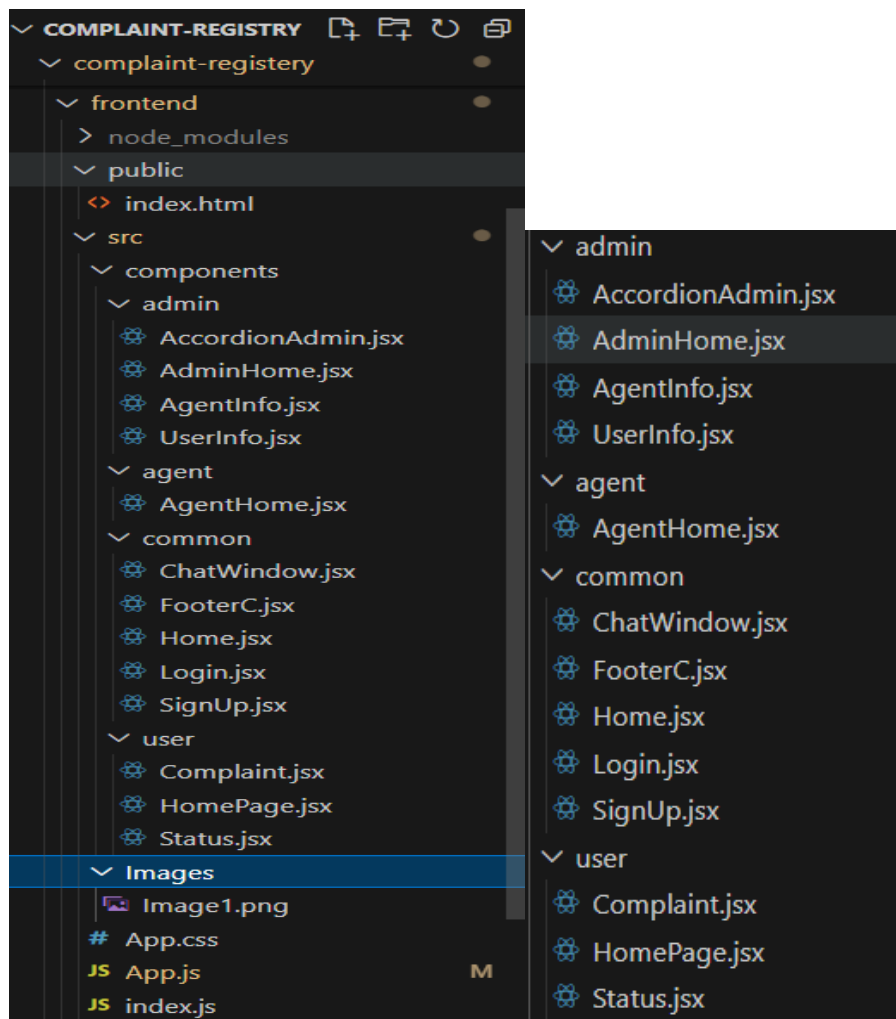
## Folder Structure

### Client (React Frontend)

The **React frontend** structure is organized to separate different components, assets, and core files for easier maintainability. Here's a breakdown:

- **public/**: Contains the static assets, like `index.html`, which is the entry point of the React app. This file is used to load the app in the browser.
- **src/**: This is the main source folder where the React app components and logic are organized.
  - **index.js**: The main file that renders the React application. It connects the React app to the `index.html` file in the public folder.
  - **App.js**: The root component that defines the main structure of the application and routes.
  - **components/**: Contains all reusable React components, organized into folders based on their purpose:
    - **admin/**: Contains components related to the admin functionality.
      - `AccordionAdmin.jsx`: Component for displaying accordion-style information specific to admins.
      - `AdminHome.jsx`: Main dashboard or home page for admin users.
      - `AgentInfo.jsx`: Displays information related to agents.
      - `UserInfo.jsx`: Shows details about users managed by the admin.
    - **agent/**: Contains components related to agent functionality.
      - `AgentHome.jsx`: Main dashboard or home page for agents.
    - **common/**: Holds components that are shared across different user roles.
      - `ChatWindow.jsx`: Component for real-time chat interactions.
      - `FooterC.jsx`: Footer component used throughout the application.
      - `Home.jsx`: The landing page or home component for general users.
      - `Login.jsx`: The login component for user authentication.

- SignUp.jsx: Component for user registration.
- **user/**: Contains components specific to general users.
  - Complaint.jsx: Allows users to submit and view complaints.
  - HomePage.jsx: Home page for logged-in users.
  - Status.jsx: Displays the status of the user's complaint.

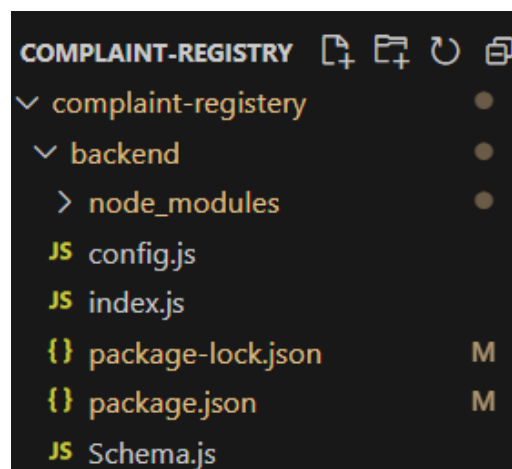


- **Images/**: This folder can be used to store any images or media assets needed in the frontend.

## Server (Node.js Backend)

The **Node.js backend** is organized to separate server logic, configurations, and routes for a modular design. Here's a breakdown:

- **backend/**
  - **index.js**: The main entry point of the backend server. It initializes the Express server, connects to MongoDB, and sets up middleware and routes.
  - **package.json** and **package-lock.json**: Contain project metadata and dependencies for the backend server.
  - **Schema.js**: Contains MongoDB schemas and models. This file defines the data structure for storing information in MongoDB, such as user and complaint details.
  - **config.js**: Stores configuration settings like database connection strings, port numbers, and any secret keys (typically using environment variables). This file is imported in index.js and other backend files that require access to environment-specific values.
  - **node\_modules/**: Contains all the installed Node.js packages required by the backend.



## Running the Application

**Frontend:** Start the React frontend server.

- `cd frontend`
- `npm start`

Run this command in the frontend directory to launch the React app. It will start a development server, typically on <http://localhost:3000>.

**Backend:** Start the Node.js backend server.

- cd backend
- npm start

It usually runs on `http://localhost:5000` or another port specified in your configuration.

## **API DOCUMENTATION**

### **1. User Registration and Authentication**

- **Endpoint:** `/api/users/register`
  - **Method:** POST
  - **Description:** Registers a new user.
  - **Request Body:**

```
{  
  "name": "John Doe",  
  "email": "johndoe@example.com",  
  "password": "securePassword123"  
}
```

**Response:**

```
{  
  "message": "User registered successfully",  
  "userId": "user123"  
}
```

**Endpoint:** `/api/users/login`

- **Method:** POST
- **Description:** Authenticates a user and returns a token.
- **Request Body:**

```
{  
  "email": "johndoe@example.com",  
  "password": "securePassword123"  
}
```

**Response:**

```
{
```

```
"token": "jwt-token-string",  
"user": {  
  "id": "user123",  
  "name": "John Doe",  
  "email": "johndoe@example.com"  
}
```

## 2. Complaint Management

- **Endpoint:** /api/complaints
  - **Method:** POST
  - **Description:** Submits a new complaint.

### Request Body:

```
{  
  "userId": "user123",  
  "description": "Product defect - screen not working",  
  "category": "Product Issue",  
  "contact": "1234567890",  
  "address": "1234 Elm Street"  
}
```

### Response:

```
{  
  "message": "Complaint submitted successfully",  
  "complaintId": "complaint456"  
}
```

### Endpoint: /api/complaints/:id

- **Method:** GET
- **Description:** Retrieves the details of a specific complaint.
- **URL Parameter:** id - the ID of the complaint.

**Response:**

```
{
  "complaintId": "complaint456",
  "status": "Pending",
  "description": "Product defect - screen not working",
  "createdAt": "2024-11-10T12:34:56Z",
  "assignedAgent": {
    "name": "Agent Sarah",
    "contact": "agent_sarah@example.com"
  }
}
```

**3. Complaint Assignment and Agent Management**

- **Endpoint:** /api/agents/assign
  - **Method:** POST
  - **Description:** Assigns a complaint to an agent.

**Request Body:**

```
{
  "complaintId": "complaint456",
  "agentId": "agent789"
}
```

**Response:**

```
{
  "message": "Complaint assigned to agent successfully",
  "agent": {
    "id": "agent789",
    "name": "Agent Sarah"
  }
}
```

**Endpoint:** /api/agents/:id/complaints

- **Method:** GET
- **Description:** Retrieves all complaints assigned to a specific agent.
- **URL Parameter:** id - the ID of the agent.

**Response:**

```
[  
  {  
    "complaintId": "complaint456",  
    "status": "In Progress",  
    "description": "Product defect - screen not working",  
    "createdAt": "2024-11-10T12:34:56Z"  
  },  
  {  
    "complaintId": "complaint789",  
    "status": "Pending",  
    "description": "Shipping delay",  
    "createdAt": "2024-11-12T08:23:45Z"  
  }  
]
```

#### 4. Complaint Status Tracking and Updates

- **Endpoint:** /api/complaints/:id/status
  - **Method:** PATCH
  - **Description:** Updates the status of a specific complaint.
  - **URL Parameter:** id - the ID of the complaint.

**Request Body:**

```
{  
  "status": "Resolved"  
}
```

**Response:**

```
{  
  "message": "Complaint status updated",  
  "complaintId": "complaint456",  
  "status": "Resolved"  
}
```

**5. Notifications**

- **Endpoint:** /api/notifications/send
  - **Method:** POST
  - **Description:** Sends a notification to the user regarding complaint status.

**Request Body:**

```
{  
  "userId": "user123",  
  "complaintId": "complaint456",  
  "message": "Your complaint has been resolved."  
}
```

**Response:**

```
{  
  "message": "Notification sent successfully"  
}
```

**AUTHENTICATION**

In this project, authentication and authorization are implemented to securely manage user access to various resources and actions within the complaint registry system. Here's an overview of how these processes work:

**1. Authentication Method****JWT (JSON Web Token) Authentication:**

- Authentication in this project is managed using JWT (JSON Web Tokens). When a user logs in, a token is generated and sent back to the user. This token is then used to authenticate future requests without requiring the user to log in again.



- **JWT Workflow:**

1. The user logs in by providing valid credentials (such as email and password) to the /login endpoint.
2. Upon successful login, the backend generates a JWT containing user information (such as user ID and role) and sends it back to the client.
3. The client stores this JWT, usually in local storage or a secure cookie.
4. For subsequent requests that require authentication, the client includes the JWT in the HTTP header (Authorization: Bearer <token>).

- **Token Structure:**

- A JWT consists of three parts: Header, Payload, and Signature.
  - **Header:** Contains the type of token (JWT) and the signing algorithm (usually HS256).
  - **Payload:** Holds the claims, including user information and custom data like role and expiration time.
  - **Signature:** Ensures the token is genuine and has not been tampered with.

## 2. Token-Based Authorization

### Role-Based Access Control (RBAC):

- The system supports different user roles (e.g., users, agents, and admins). Each role has specific permissions to access various endpoints and perform certain actions.
- **Role-Specific Access:**
  - **User:** Can register, log in, and submit complaints. They can view their own complaint statuses.
  - **Agent:** Can view and manage assigned complaints, update complaint statuses, and communicate with users.
  - **Admin:** Has higher privileges, such as viewing all complaints, assigning agents, and managing user roles.

### Implementation:

- Each protected route checks for a valid JWT token in the request header.
- The JWT is decoded, and the user's role is verified.
- Middleware functions are used to enforce role-based permissions:
  - For example, only users with an "admin" role can access certain admin-specific endpoints.

### 3. JWT Expiration and Refresh

- **Token Expiration:** To enhance security, JWT tokens are configured to expire after a certain period (e.g., 1 hour). Once expired, the user must reauthenticate to obtain a new token.
- **Refresh Token Strategy:**
  - A refresh token mechanism could be implemented to allow users to obtain a new JWT without logging in again after the initial token expires.
  - The refresh token would have a longer expiration time and could be used to issue a new JWT while maintaining the user's session.

### 4. Middleware for Authentication and Authorization

#### Authentication Middleware:

- Middleware functions validate the JWT in each request to confirm that the user is authenticated before granting access to protected routes.
- If the token is missing or invalid, the middleware denies access and returns an authentication error.

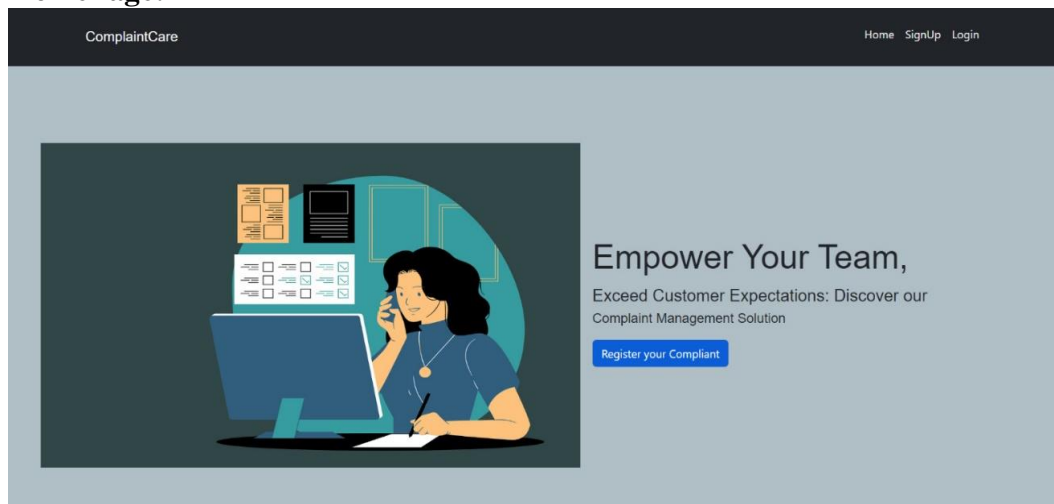
#### Authorization Middleware:

- Additional middleware checks the user's role (decoded from the JWT) and allows access only if the role has permission for the requested action.
- This ensures, for example, that only an agent can update a complaint's status, and only an admin can assign agents to complaints.

By implementing JWT-based authentication and role-based authorization, this system ensures that only authorized users can access specific resources and perform appropriate actions. This setup improves both security and usability, as users only need to authenticate once per session and can then use their token for secure, repeated access.

## USER INTERFACE

#### HomePage:



### Login and Signup Pages:

ComplaintCare

HomeSignUpLogin

Login For Registering the Complaint

Please enter your Credentials!

nikhil

Email

Password

Login

Don't have an account?

[Sign Up](#)

[Home](#) [SignUp](#) [Login](#)

## SignUp For Registering the Complaint

Please enter your Details

Full Name

Email

Password

Mobile No.

Select User ▼

Select User Type

Register

**Complaint Status Page:**

Hi Admin Axin Ricco

Dashboard

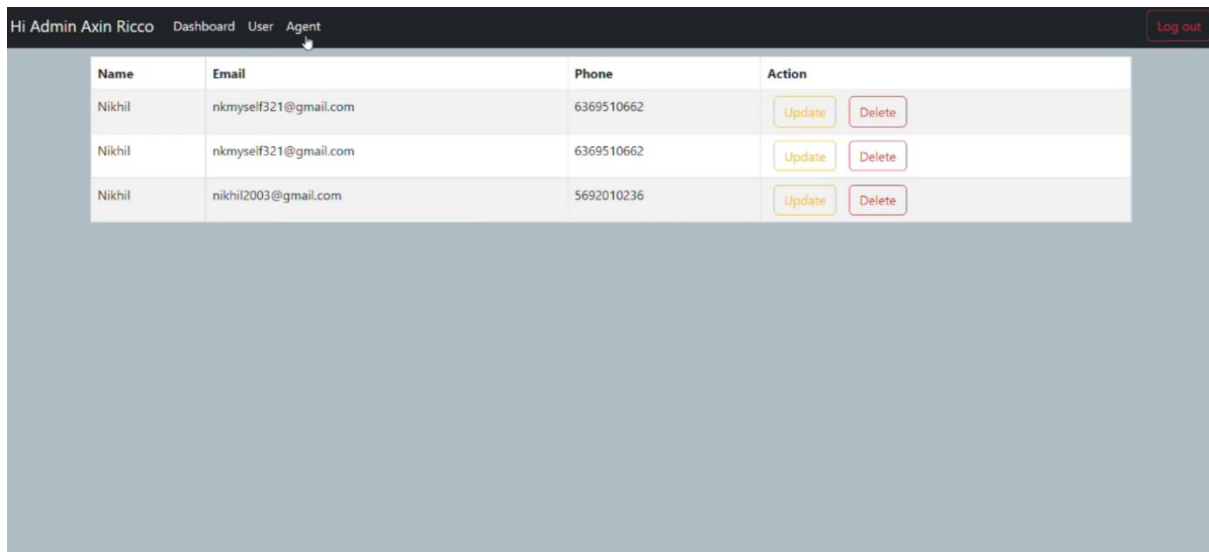
User

Agent

Log out

Name	Email	Phone	Action
Prasanna Velan	prasanna225@gmail.com	8951025662	<div>UpdateDelete</div>

## Admin Dashboard:



Hi Admin Axin Ricco   Dashboard   User   Agent <a href="#">Log out</a>			
Name	Email	Phone	Action
Nikhil	nkmyself321@gmail.com	6369510662	<a href="#">Update</a> <a href="#">Delete</a>
Nikhil	nkmyself321@gmail.com	6369510662	<a href="#">Update</a> <a href="#">Delete</a>
Nikhil	nikhil2003@gmail.com	5692010236	<a href="#">Update</a> <a href="#">Delete</a>

## TESTING

To ensure the application functions correctly and meets the requirements, a robust testing strategy is employed. This includes both manual and automated testing methods to validate the functionality, usability, performance, and security of the system.

### Testing Strategy

1. Unit Testing
  - Focuses on testing individual components or modules of the application (both frontend and backend).
  - Ensures that each unit works as expected in isolation.
  - Example: Testing React components like Login.jsx or API functions in the backend.
2. Integration Testing
  - Tests the interaction between different components or modules.
  - Example: Verifying that the frontend communicates correctly with the backend through APIs using Axios.
3. End-to-End (E2E) Testing
  - Simulates real-world user scenarios to ensure the entire application works as intended.
  - Example: A user signing up, submitting a complaint, and tracking its status.

#### 4. Functional Testing

- Verifies that the system's features work according to the specified requirements.
- Example: Testing the complaint submission form to ensure required fields are validated and data is correctly saved.

#### 5. Regression Testing

- Ensures that new updates or changes do not break existing functionality.
- Example: After implementing a new notification feature, verify the complaint submission flow is unaffected.

#### 6. Performance Testing

- Evaluates the system's speed, responsiveness, and stability under different conditions.
- Example: Testing the system with multiple simultaneous users submitting complaints.

#### 7. Security Testing

- Ensures that the application is secure and protects user data.
- Example: Testing user authentication, token validation, and access control to prevent unauthorized access.

### **Tools Used**

#### 1. Frontend Testing Tools

- Jest: For unit testing React components.
- React Testing Library: For testing user interactions with React components.
- Cypress: For end-to-end testing of the frontend application.

#### 2. Backend Testing Tools

- Mocha and Chai: For unit and integration testing backend logic and API endpoints.
- Postman: For manual testing of RESTful API endpoints.
- Supertest: For automated API testing in Node.js.

#### 3. Performance Testing Tools

- JMeter: To simulate high loads and test backend performance under stress.
- Lighthouse: For frontend performance testing.

#### 4. Security Testing Tools

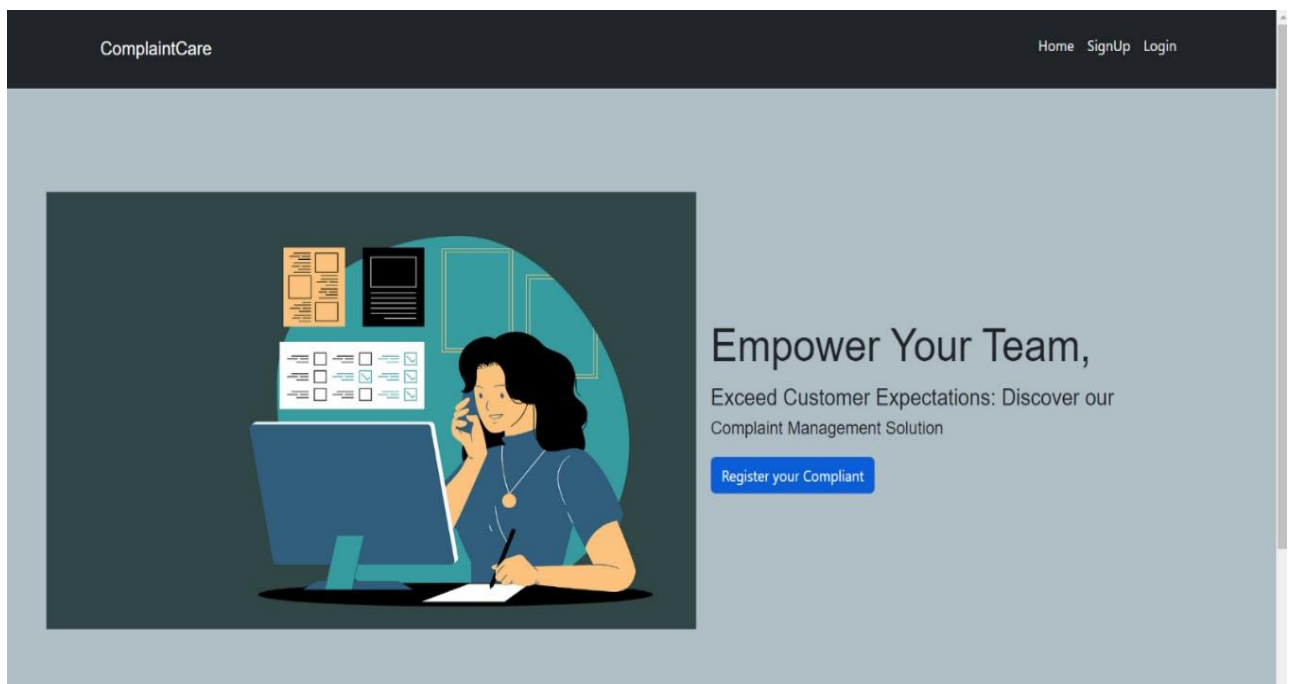
- OWASP ZAP: For automated security testing and identifying vulnerabilities.
- Postman: To check authentication mechanisms and test API security.

#### 5. Bug Tracking and Reporting Tools

- Trello or Jira: To log and manage bugs identified during testing.
- GitHub Issues: To track and resolve bugs directly within the repository.

### **SCREENSHOTS OR DEMO**

#### **HomePage:**

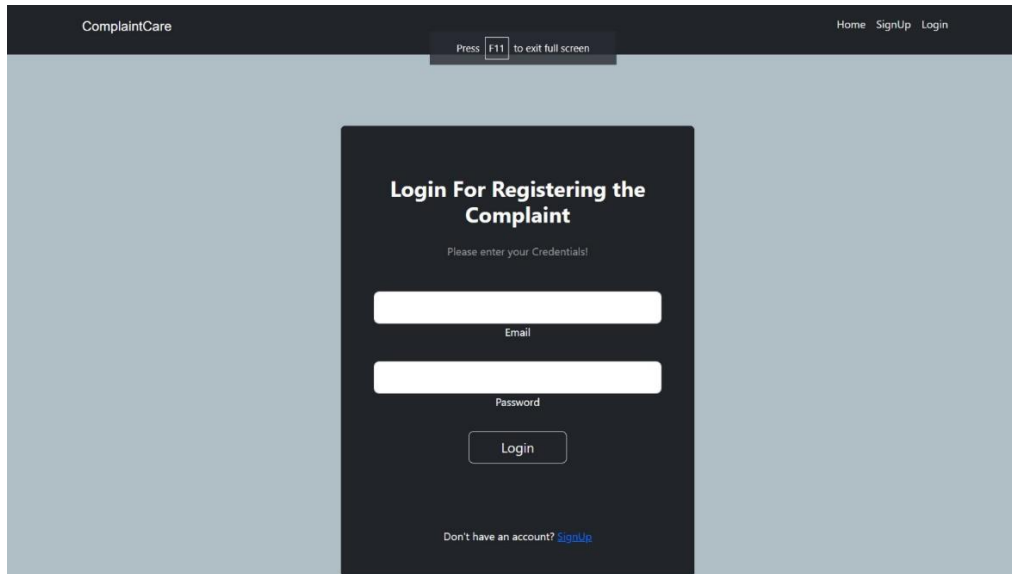


#### **Login and Signup Pages:**

##### **Signup Page:**

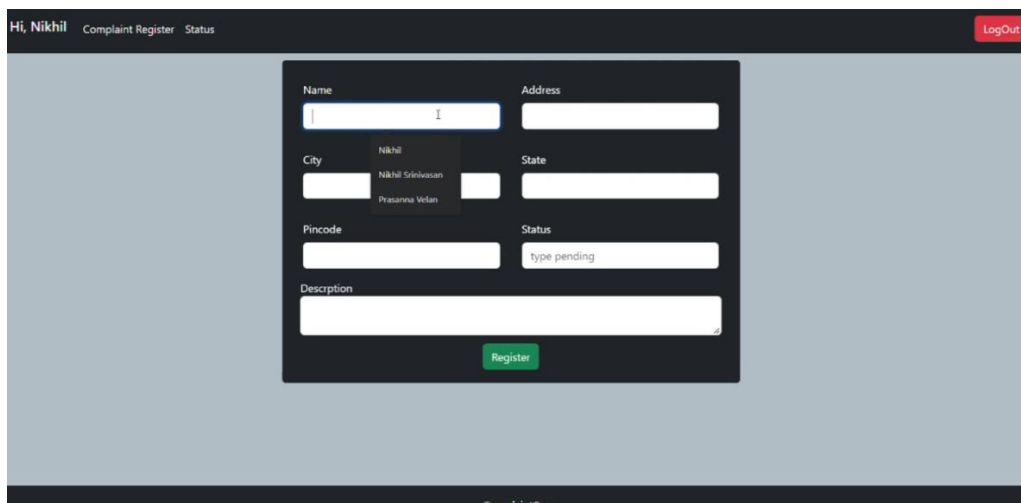
The screenshot shows the 'SignUp For Registering the Complaint' page. The header is dark grey with 'ComplaintCare' on the left and 'Home', 'SignUp', and 'Login' on the right. The main content area has a light blue background. In the center, there is a dark grey vertical card. The card has the title 'SignUp For Registering the Complaint' and a subtitle 'Please enter your Details'. Below the subtitle are four white input fields with labels: 'Full Name', 'Email', 'Password', and 'Mobile No.'. Below these fields are two dropdown menus, both labeled 'Select User \*', with the second one also having the text 'Select User Type' below it. At the bottom of the card is a white button with the text 'Register'.

## Login Page:



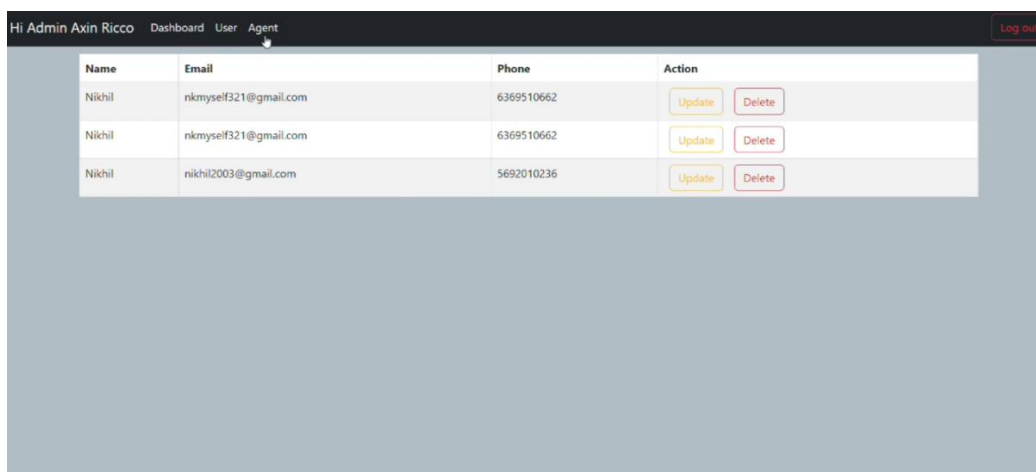
The screenshot shows the login page of the ComplaintCare application. The header includes the brand name 'ComplaintCare' on the left and navigation links 'Home', 'SignUp', and 'Login' on the right. A small instruction box says 'Press F11 to exit full screen'. The main content area features a dark grey login card with the title 'Login For Registering the Complaint' and a subtext 'Please enter your Credentials!'. The card contains two input fields for 'Email' and 'Password', a 'Login' button, and a link for users who 'Don't have an account? [signUp](#)'.

## Complaint Submission Form:



The screenshot displays the complaint submission form. The top navigation bar shows 'Hi, Nikhil', links to 'Complaint Register' and 'Status', and a 'LogOut' button. The form itself is a dark grey box with fields for 'Name', 'Address', 'City', 'State', 'Pincode', 'Status' (with a dropdown menu currently showing 'type pending'), and 'Description'. A green 'Register' button is at the bottom of the form. A dropdown menu for 'City' is open, showing options like 'Nikhil', 'Nikhil Srinivasan', and 'Prasanna Velam'.

## Admin Dashboard:



The screenshot shows the admin dashboard. The top navigation bar includes 'Hi Admin Axin Ricco', links to 'Dashboard', 'User', and 'Agent', and a 'Log out' button. The main content area contains a table with columns for 'Name', 'Email', 'Phone', and 'Action'.

Name	Email	Phone	Action
Nikhil	<a href="mailto:nkmyself321@gmail.com">nkmyself321@gmail.com</a>	6369510662	<a href="#">Update</a> <a href="#">Delete</a>
Nikhil	<a href="mailto:nkmyself321@gmail.com">nkmyself321@gmail.com</a>	6369510662	<a href="#">Update</a> <a href="#">Delete</a>
Nikhil	<a href="mailto:nikhil2003@gmail.com">nikhil2003@gmail.com</a>	5692010236	<a href="#">Update</a> <a href="#">Delete</a>

## **KNOWN ISSUES**

### **1. Form Validation Errors**

- **Issue:** Inconsistent form validation messages across different pages.
  - On the complaint submission form, validation errors sometimes do not appear for missing fields until the form is resubmitted.
- **Impact:** Users may be unaware of why their submission fails.
- **Workaround:** Ensure all required fields are filled before submitting. Clear error messages will be added in the next version.
- **Status:** To be fixed.

### **2. Real-Time Chat Delays**

- **Issue:** Occasional delays in the real-time chat feature between users and agents.
- **Impact:** Affects the responsiveness of the communication feature.
- **Cause:** High server load or suboptimal Socket.IO event handling.
- **Workaround:** Refresh the page if the chat becomes unresponsive.
- **Status:** Optimization in progress.

### **3. Mobile Responsiveness Issues**

- **Issue:** Some pages, like the admin dashboard, are not fully optimized for smaller screens.
- **Impact:** Difficult navigation for mobile users.
- **Workaround:** Use a desktop or tablet for accessing these pages.
- **Status:** Planned for improvement.

### **4. API Rate Limiting**

- **Issue:** Lack of rate-limiting on API endpoints could lead to server overload if abused.
- **Impact:** Potential security risk and performance degradation.
- **Workaround:** None currently; users are trusted to use the API responsibly.
- **Status:** Implementation planned for the next update.



## 5. Session Expiry Without Warning

- **Issue:** Users are logged out when their session token expires without a prior warning.
- **Impact:** Disrupts ongoing actions, such as form submissions or chats.
- **Workaround:** Re-login and repeat the action. Persistent notifications for nearing token expiration will be introduced.
- **Status:** To be addressed.

## 6. Admin Complaint Assignment Logic

- **Issue:** Complaints are sometimes not reassigned properly if the originally assigned agent is unavailable.
- **Impact:** Complaints may remain unresolved longer than necessary.
- **Workaround:** Admins can manually reassign complaints.
- **Status:** Fix under development.

## 7. Notification Inconsistencies

- **Issue:** Some users do not receive email or SMS notifications for status updates.
- **Impact:** Affects user awareness of complaint progress.
- **Cause:** Configuration issues in the notification service (e.g., SMTP or Twilio).
- **Workaround:** Regularly check the dashboard for updates.
- **Status:** Being investigated.

## 8. Database Schema Limitations

- **Issue:** The current schema does not support bulk actions, such as resolving multiple complaints simultaneously.
- **Impact:** Reduces efficiency for admins managing a high volume of complaints.
- **Workaround:** Perform actions one at a time.
- **Status:** Schema redesign planned.

## **FUTURE ENHANCEMENTS**

### **1. Advanced Reporting and Analytics**

- **Description:** Add a reporting dashboard for admins to generate detailed analytics.
- **Features:**
  - Complaint resolution time metrics.
  - Department or agent performance analytics.
- **Benefit:** Helps administrators monitor system efficiency and identify areas for improvement.

### **2. Multi-Language Support**

- **Description:** Introduce language options for a more inclusive user experience.
- **Features:**
  - Dynamic language selection (e.g., English, Spanish, French).
  - Localization of all UI elements and notifications.
- **Benefit:** Increases accessibility for users in diverse regions.

### **3. AI-Powered Complaint Categorization**

- **Description:** Use machine learning to categorize complaints automatically based on the description.
- **Features:**
  - Automatic tagging and routing of complaints to the relevant department.
  - Predictive analysis to suggest resolutions based on similar complaints.
- **Benefit:** Speeds up the complaint assignment process and reduces manual workload.

### **4. Integration with Social Media and Messaging Platforms**

- **Description:** Allow users to submit complaints through social media or messaging apps.
- **Features:**
  - Integration with WhatsApp, Facebook Messenger, or Twitter.
  - Automated replies and status updates via these platforms.
- **Benefit:** Enhances user convenience and engagement

## 5. Mobile App Development

- **Description:** Develop native mobile applications for Android and iOS.
- **Features:**
  - Push notifications for updates and reminders.
  - Offline mode for complaint drafting.
- **Benefit:** Improves accessibility and enhances the user experience for mobile users.

## 6. Bulk Actions for Admins

- **Description:** Enable administrators to perform bulk actions for managing complaints.
- **Features:**
  - Bulk complaint assignment.
  - Batch status updates (e.g., mark multiple complaints as resolved).
- **Benefit:** Increases efficiency in managing large volumes of complaints.

## 7. Two-Factor Authentication (2FA)

- **Description:** Enhance security for user accounts by adding 2FA during login.
- **Features:**
  - Email or SMS-based OTP for login.
  - Support for authenticator apps like Google Authenticator.
- **Benefit:** Protects against unauthorized access.

## 8. Real-Time Collaboration Tools

- **Description:** Add tools for agents to collaborate on complex complaints.
- **Features:**
  - Shared notes on complaints.
  - Internal chat or call feature for agents and admins.
- **Benefit:** Improves coordination and resolution efficiency.

## 9. Enhanced Notification System

- **Description:** Upgrade the notification system for better user communication.
- **Features:**
  - In-app notifications for all actions.
  - Scheduled reminders for pending complaints.
- **Benefit:** Keeps users informed and reduces delays in resolution.

## 10. Customizable User Roles

- **Description:** Allow admins to create and assign custom user roles.
- **Features:**
  - Define permissions for each role.
  - Support for roles beyond admin, agent, and user.
- **Benefit:** Provides flexibility for organizational needs.

## 11. Gamification for Agents

- **Description:** Introduce gamification to motivate agents and improve performance.
- **Features:**
  - Points or badges for resolving complaints quickly.
  - Leaderboards to encourage healthy competition.
- **Benefit:** Boosts morale and enhances productivity.

## 12. API for Third-Party Integration

- **Description:** Develop a public API to allow third-party platforms to integrate with the system.
- **Features:**
  - Access to complaint submission and tracking features.
  - Secure endpoints for third-party developers.
- **Benefit:** Extends the usability of the platform beyond the core system.

### 13. Video or Voice Call Support

- **Description:** Enable users to directly connect with agents through video or voice calls.
- **Features:**
  - Integration with WebRTC for real-time calls.
  - Call recording for future reference.
- **Benefit:** Improves communication and simplifies complex complaint resolution.

### 14. Dark Mode

- **Description:** Add a dark mode option for the user interface.
- **Benefit:** Provides a visually comfortable experience, especially for users working at night.

### 15. Blockchain for Complaint Transparency

- **Description:** Use blockchain technology to ensure the immutability and transparency of complaints.
- **Features:**
  - Store complaint histories on a blockchain ledger.
  - Allow users to verify the integrity of their complaints.
- **Benefit:** Builds trust and ensures transparency in the complaint resolution process.

## **CONCLUSION**

The **Online Complaint Registration and Management System** has been successfully designed and implemented to address the critical need for streamlined and efficient complaint handling processes. By leveraging state-of-the-art technologies like **React**, **Node.js**, **Express.js**, and **MongoDB**, the system provides an intuitive and scalable solution for users, agents, and administrators alike.

### **Key Achievements**

1. **Streamlined User Experience:** The system offers a clean and user-friendly interface, enabling users to submit, track, and resolve complaints with minimal effort.
2. **Centralized Management:** By providing administrators and agents with an organized dashboard and powerful tools, the system ensures efficient allocation and resolution of complaints.

3. **Real-Time Interactions:** Features like live chat between users and agents enhance communication and foster trust.
4. **Scalable Architecture:** Built with modern frameworks and database technologies, the system can handle increased demand as the user base grows.
5. **Security and Confidentiality:** Robust measures, including encrypted data transmission and secure authentication, safeguard user information and build trust.

### Impact and Benefits

- **For Users:** Empowering users to file and monitor complaints efficiently, resulting in improved satisfaction and trust.
- **For Organizations:** Enhances operational efficiency, reduces response times, and fosters accountability in complaint management.
- **For Agents and Admins:** Facilitates better workload management, data-driven decision-making, and higher productivity.

### Scope for Future Enhancements

While the project addresses its primary objectives, several potential enhancements could make it even more effective, such as:

- Incorporating AI for automatic complaint categorization and predictive insights.
- Expanding features like multilingual support, mobile apps, and video call capabilities.
- Providing advanced reporting and analytics tools to track performance and trends.

### Conclusion Statement

This project not only delivers an effective complaint management solution but also sets a foundation for continuous improvement and innovation. By addressing real-world challenges in complaint registration and resolution, it provides a critical service to organizations and users, ensuring transparency, efficiency, and satisfaction.

With its potential for scalability and future enhancements, the **Online Complaint Registration and Management System** is poised to become an indispensable tool for modern organizations striving for exceptional customer service.