Full Stack Development with MERN

1. Introduction

- **Project Title:** Online Complaint Registration And Management System
- **Team Members:**

  Kinjal.K.Dave – Project Report , Project Implementation

  Nandana Krishna – Video Voice over , Project Implementation

  Niveditha.S – Frontend Development , Project Implementation

  Sharmila.S – Backend Development , Project Implementation

  Suroochi.G.C – Video Editing , Project Implementation

2. Project Overview

- **Purpose:** An online complaint registration and management system is a software application or platform that allows individuals or organizations to submit and track complaints or issues they have encountered. It can help optimize the complaint handling process and empower organizations to develop a safety management system to efficiently resolve customer complaints, while staying in line with industry guidelines and regulatory compliance obligations. It provides a centralized platform for managing complaints, streamlining the complaint resolution process, and improving customer satisfaction.

- **Features:**

1. User registration: Users can create accounts to submit complaints and track their progress.
2. Complaint submission: Users can enter details of their complaints, including relevant information such name, description of the issue, address etc.
3. Tracking and notifications: Users can track the progress of their complaints, view updates, and receive notifications via email or SMS when there are any changes or resolutions.
4. User can interact with the agent who has assigned the complaint.
5. Assigning and routing complaints: The system assigns complaints to the appropriate department or personnel responsible for handling them. It may use intelligent routing algorithms to ensure efficient allocation of resources.
6. Security and confidentiality: The system ensures the security and confidentiality of user data and complaint information through measures such as user authentication, data encryption, access controls, and compliance with relevant data protection regulations.

3. Architecture

- **Frontend:** The frontend architecture is built using **React.js**, emphasizing modularity and reusability. The application starts with a structured directory setup and necessary library installations. Core functionalities are divided into reusable UI components, such as forms, buttons, and navigation bars, ensuring a consistent design across the application. React's state management and data binding handle dynamic updates seamlessly, connecting the frontend to backend APIs for real-time interactions. Features like complaint registration, tracking, and admin dashboards are implemented with Material-UI and Bootstrap for enhanced styling and responsiveness.
- **Backend:** The backend is developed using **Node.js** and **Express.js**, providing a scalable server-side framework. It features a well-organized directory structure with modularized routes for authentication, complaint handling, and admin functions. The API endpoints interact with a MongoDB database using **Mongoose**, while middleware like body-parser and cors ensures smooth data parsing and cross-origin communication. Authentication is managed through **JWT tokens**, securing sensitive operations and user data. Error handling middleware guarantees standardized responses, enhancing reliability and maintainability.

- **Database:** The database architecture uses **MongoDB** for efficient storage and management of user and complaint data. Key entities, such as users, complaints, assigned complaints, and messages, are modeled using **Mongoose** schemas, ensuring seamless CRUD operations. Relationships are established using object references like userId and complaintId to link users with their complaints and assigned agents. Each schema is optimized for performance, with data stored in separate collections to support scalability and complex queries, ensuring robust data handling and retrieval.

4. Setup Instructions

- **Prerequisites:** For the backend to function well, Those libraries includes

1) Node.js.
2) MongoDB.
3) Bcrypt
4) Body-parser

Also, for the frontend we use the libraries such as

1) React Js.
2) Material UI
3) Bootstrap
4) Axios

- **Installation:**

1) **Node.js & npm**: Required for running server-side JavaScript and managing dependencies. Download Node.js
2) **Express.js**: Web application framework for building APIs and handling routing. Install using: npm install express.
3) **MongoDB**: NoSQL database for storing application data. Download MongoDB

4) **React.js**: JavaScript library for creating interactive user interfaces. React.js Setup Guide
5) **HTML, CSS, JavaScript**: Essential for structuring, styling, and adding interactivity to the app.
6) **Database Connectivity**: Used Mongoose to connect Node.js server with MongoDB for CRUD operations. Guide
7) **UI Libraries**: Utilized Material-UI and Bootstrap for styling and enhancing the user interface.
8) **Version Control**: Managed code versions with Git. Download Git
9) **Development Tools**: Used Visual Studio Code as the code editor. Download VS Code

5. Folder Structure

- **Client:** The React frontend is organized into modular and reusable components, enabling a clean and scalable design. Key aspects include:

1) **Component-Based Architecture**: Individual components (e.g., Header, Footer, ComplaintForm, AdminDashboard) handle specific UI elements.
2) **Routing**: React Router is used to navigate between pages such as the complaint submission form, status view, and admin dashboard.
3) **State Management**: Managed locally within components or using libraries like Context API for global state sharing.
4) **Styling**: Material-UI and Bootstrap are used for consistent and responsive design.
5) **APIs**: Axios or Fetch API is used to communicate with the backend server for CRUD operations.

- **Server:** The Node.js backend follows an MVC (Model-View-Controller) architecture for maintainability and scalability. Key components include:

1) **Routing**: Express.js handles API endpoints for managing complaints, users, and admin operations.
2) **Controllers**: Contain the business logic for processing requests and responding to the client.
3) **Models**: Use Mongoose to define the structure of MongoDB collections (e.g., User, Complaint).
4) **Middleware**: Implements functions for request validation, authentication (e.g., JWT), and error handling.
5) **Database Integration**: MongoDB is connected using Mongoose for seamless CRUD operations.
6) **Environment Variables**: Managed securely with dotenv for configurations like database URIs and API keys.

6. Running the Application

- Provide commands to start the frontend and backend servers locally.
  - **Frontend:**

    (a) **Navigate to the Frontend Directory**:
    <u>**Command-**</u> cd complaint-registery/frontend

(b) **Install Dependencies**:
Ensure all required packages are installed:
<u>Command-</u> npm install

(c) **Start the Development Server**:
Launch the React frontend:
<u>Command-</u> npm start

(d) **Access the Application**:
The application will be available at http://localhost:3000 by default.

- **Backend:**

(e) **Navigate to the Backend Directory**:
Change the terminal directory to where the backend code is located:
<u>Command-</u>cd <project-directory>/backend

(f) **Install Dependencies**:
Use npm to install all required packages such as Express.js, Mongoose, etc.:
<u>Command-</u>npm install

(g) **Set Up Environment Variables**:
Ensuring the .env file is properly configured with values like:

- Database connection string (MONGO_URI)
- JWT secret (JWT_SECRET)
- Server port (PORT)

(h) **Start the Server**:
Start the development server with the following command:
<u>Command-</u>npm start

(i) **Access the Backend**:
The backend will run on the port specified in the .env file (default: http://localhost:5000).

7. API Documentation

- Document all endpoints exposed by the backend.
- Include request methods, parameters, and example responses.

8.Authentication :

Authentication in the project is managed using **JSON Web Tokens (JWT)**. JWT is a widely used token-based authentication mechanism that allows for secure transmission of information between the client and server.

1. **User Registration**
   During the user registration process, the following details are captured from the user: name, email, password, phone, and userType. The password is hashed and stored in the database. This ensures that even if the database is compromised, user passwords remain secure.
   - The **user's email** is used as the unique identifier for login.
   - The **password** is hashed using a hashing algorithm (e.g., bcrypt) before storing it in the database.

2. **User Login**
   When a user logs in, the system checks if the provided credentials (email and password) match any records in the database.
   - If the credentials are valid, the system generates a JWT token that is sent back to the client.
   - This token includes an encrypted payload containing user information and an expiration date, which ensures that the token is valid for a limited time (e.g., 1 hour).
3. **Token-based Authentication**
   - The token is sent to the client after successful login, and it is stored in the client's local storage or cookies.
   - For subsequent requests to protected routes (e.g., creating a complaint, fetching user-specific complaints, etc.), the client sends the JWT token in the **Authorization header** as a Bearer token.

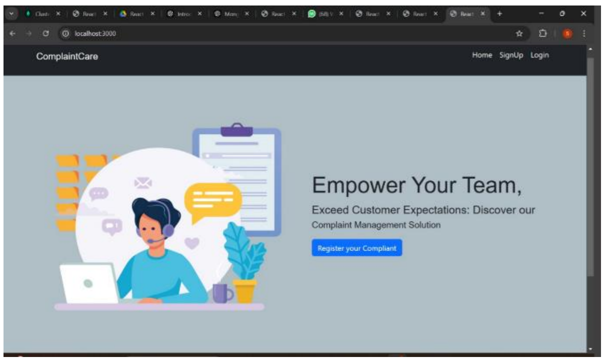
Authorization

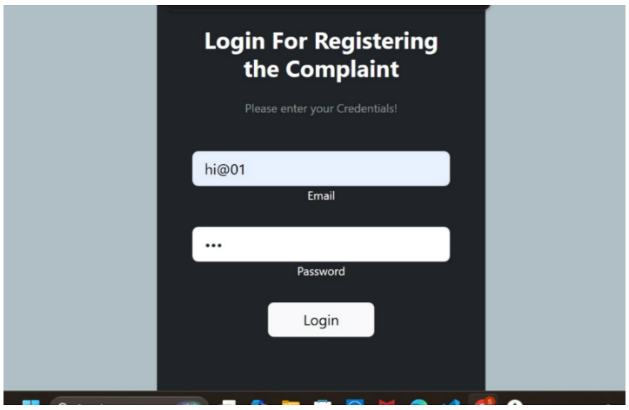
Authorization ensures that a user has permission to access specific resources or perform certain actions. The system uses the following strategies to enforce authorization:

1. **Role-based Access Control (RBAC)**
   The system supports multiple user roles (e.g., customer, agent, admin). Based on the role stored in the JWT payload, the user is granted access to different endpoints:
   - **Customer**: Can create complaints, view their own complaints, and chat with agents.
   - **Agent**: Can view complaints assigned to them and communicate with customers.
   - **Admin**: Can access all complaints, assign complaints to agents, view user data, and manage the entire system.
2. **Protected Routes**
   Certain API endpoints are protected, meaning that only authenticated and authorized users can access them. For example:
   - Only **admins** can access routes to fetch all complaints or assign complaints.
   - Only **agents** can view complaints assigned to them.

In these cases, before accessing the route, the server checks if the token is valid and if the user's role matches the required role for that endpoint.

9. User Interface

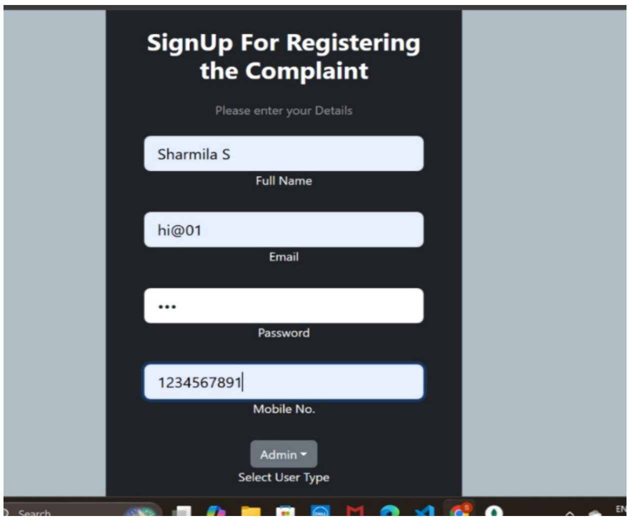



Login For Registering the Complaint

Please enter your Credentials!

Email: hi@01

Password: •••


SignUp For Registering the Complaint

Please enter your Details

Full Name: Sharmila S

Email: hi@01

Password: •••

Mobile No.: 1234567891

Select User Type: Admin

10. Testing

The testing approach for the Online Complaint Registration and Management System ensures that the application operates as intended across all components, including the frontend, backend, and database. The strategy involves the following stages:

1. Unit Testing
    - Objective: Validate individual components, functions, or modules in isolation.
    - Examples:
        - Frontend: Ensure React components render correctly with expected props.
        - Backend: Test utility functions and middleware like token verification or request parsing.
        - Database: Verify CRUD operations on MongoDB models
2. Integration Testing

- **Objective**: Test the interaction between different modules and services.
- **Examples**:
    - Frontend & Backend: Verify API endpoints work correctly when called from React components.

    - Backend & Database: Ensure database queries return expected results for API requests.

3. End-to-End (E2E) Testing

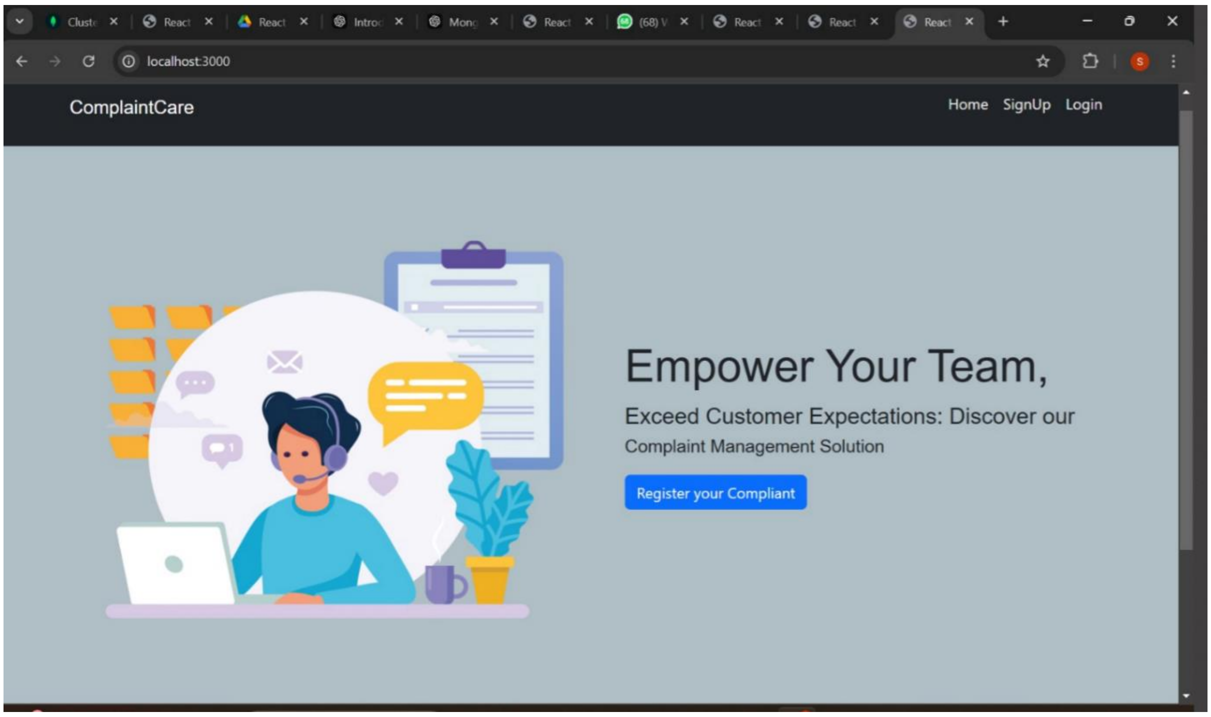
- **Objective**: Simulate real user interactions to ensure the system works as a whole.
- **Examples**:
    - A customer logs in, registers a complaint, and views the complaint status.
    - An admin assigns a complaint to an agent, and the agent marks it as resolved.

4. Performance Testing

    - **Objective**: Evaluate the application's performance under various conditions.
    - **Examples**:
        - Test the backend's response time for API calls under load.
        - Check database query execution time for high-volume transactions.

5. Security Testing
    - **Objective**: Identify vulnerabilities and ensure data security.
    - **Examples**:
        - Test for SQL injection in API endpoints.
        - Ensure sensitive user data like passwords are hashed correctly.
6. Manual Testing
    - **Objective**: Perform exploratory testing to catch edge cases or unexpected behaviors.
    - **Examples**:
        - Test UI responsiveness on various screen sizes.
        - Verify that role-based access control restricts users to their intended functionalities.

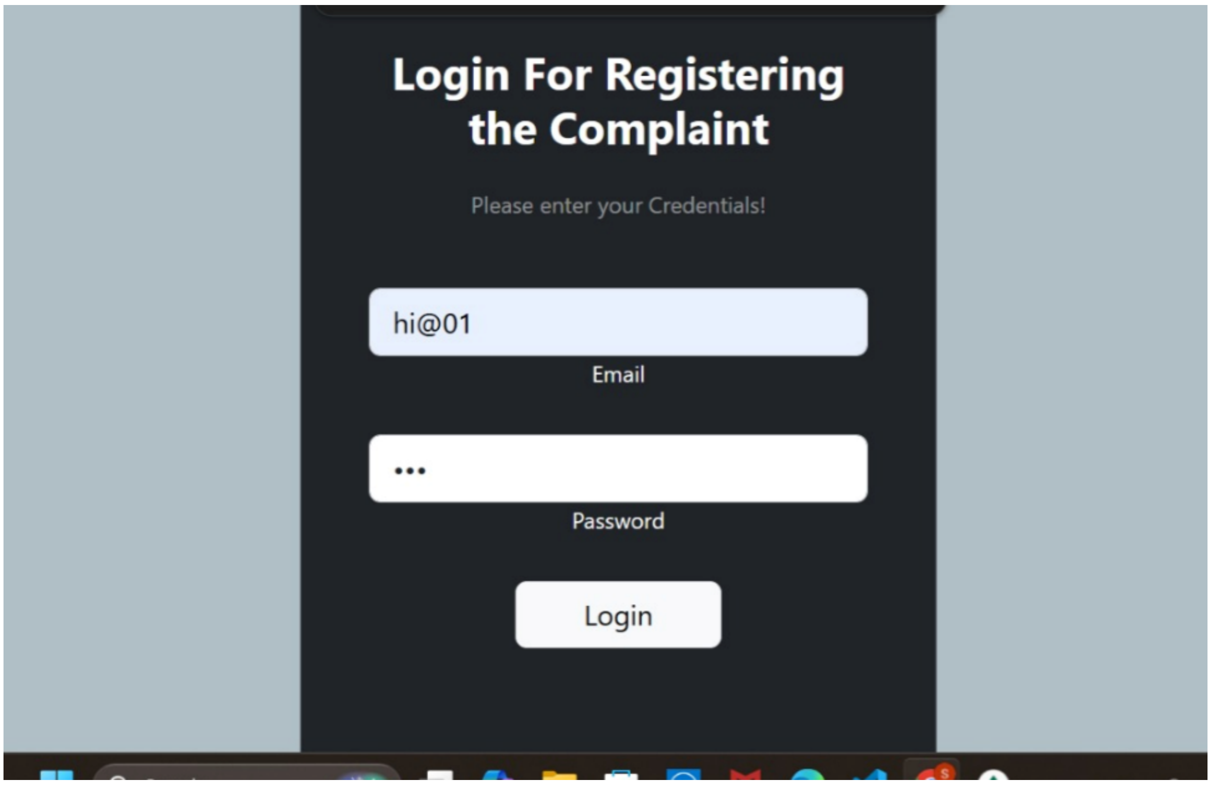
11. Screenshots or Demo



Empower Your Team,

Exceed Customer Expectations: Discover our

Complaint Management Solution

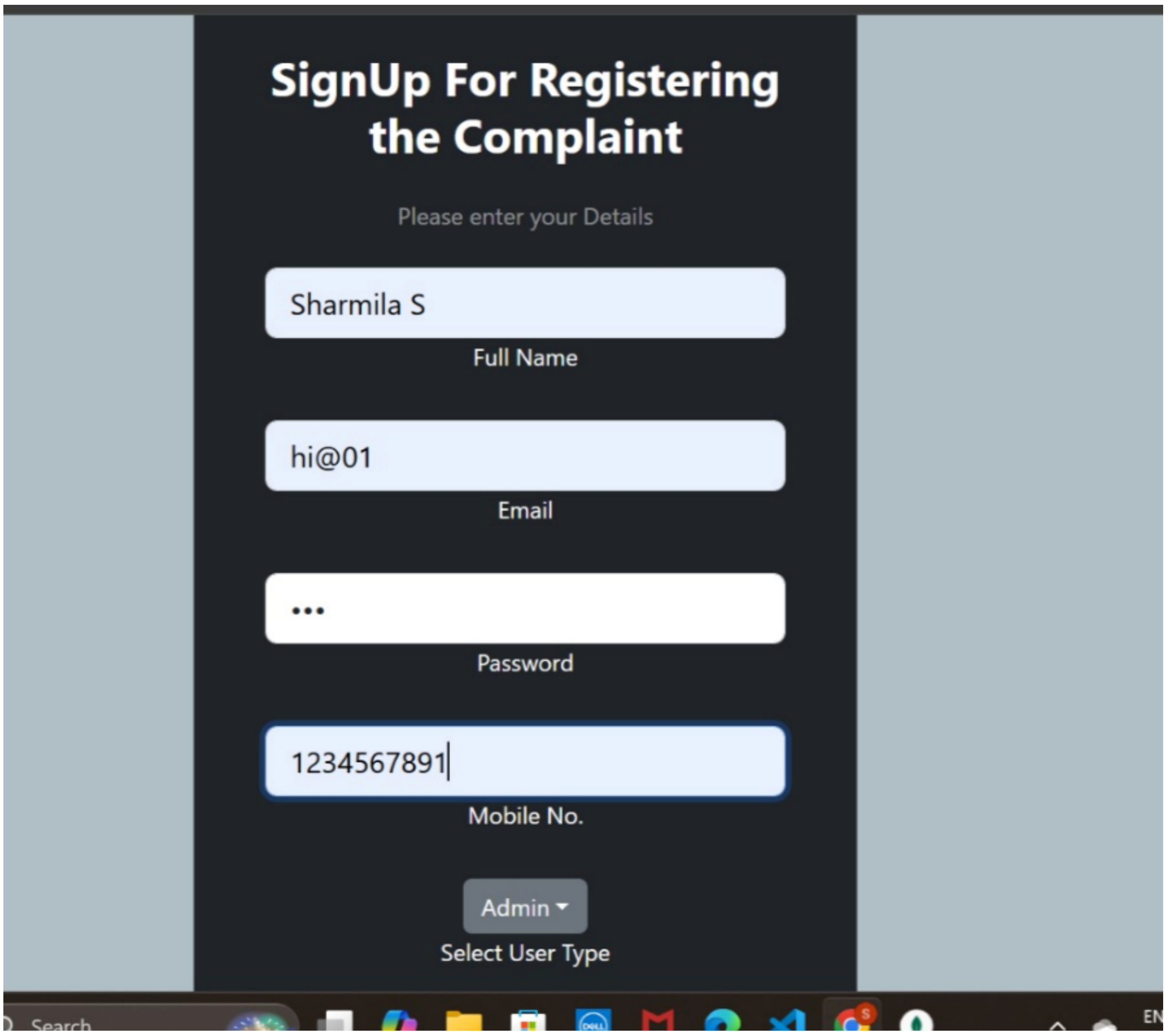


Login For Registering the Complaint

Please enter your Credentials!

hi@01

•••


SignUp For Registering the Complaint

Please enter your Details

Full Name: Sharmila S

Email: hi@01

Password: •••

Mobile No.: 1234567891

Select User Type: Admin

12. Known Issues

1. **Delayed Real-Time Updates**:
   - In some cases, the chat interface does not update immediately after a new message is sent.
   - **Workaround**: Refresh the chat page manually.
2. **Pagination Not Implemented**:
   - Large datasets (e.g., complaints) may slow down the dashboard due to the absence of pagination.
3. **Error Message Clarity**:
   - Some error responses lack user-friendly descriptions, which may confuse end-users.

13. Future Enhancements

1. **Improved Chat Functionality**:
   - Implement WebSockets for real-time chat updates without requiring manual refresh.

2. **Advanced Analytics for Admin**:
    - Add visual dashboards for complaint trends, resolution times, and agent performance.
3. **Mobile App Integration**:
    - Develop a React Native-based mobile app for users and agents.
4. **Enhanced User Notifications**:
    - Introduce email and SMS alerts for status updates on complaints.
5. **Multilingual Support**:
    - Add support for multiple languages to serve a diverse user base.