

# **ONLINE COMPLAINT REGISTRATION AND MANAGEMENTSYSTEM WEBAPP USING MERN STACK**

**SMART INTERNZ (NAAN MUDHALVAN)**

**A Project Done By**

<b>ASHVITHA D</b>	<b>211121205003</b>
<b>DIVYA CC</b>	<b>211121205008</b>
<b>HEPSIBHA K</b>	<b>211121205012</b>
<b>SNEHA R</b>	<b>211121205026</b>

**STUDENTS OF  
INFORMATION TECHNOLOGY DEPARTMENT**



**MADHA ENGINEERING COLLEGE**

**KUNDRATHUR**

**CHENNAI-69**

**NOV 2024**

## **ABSTRACT**

The "Online Complaint Registration and Management System" is a comprehensive platform designed to streamline the process of registering, tracking, and resolving complaints efficiently. Leveraging the MERN stack (MongoDB, Express.js, React, and Node.js), the system provides a robust, scalable, and user-friendly solution for individuals and organizations. MongoDB serves as the database, efficiently storing user details, complaint records, and agent assignments. Express.js and Node.js form the backend, handling API requests, authentication, and secure data communication, while React.js powers the frontend, delivering a responsive and intuitive user interface.

Key features of the application include secure user registration and authentication, complaint submission with detailed descriptions and attachments, real-time tracking of complaint status, email or SMS notifications, and interactive communication between users and agents. The system also employs JWT-based authentication for secure user identity management. Future enhancements aim to incorporate AI for intelligent complaint categorization, multilingual support, and mobile application development for greater accessibility.

The architecture and design of the system emphasize transparency, timely resolutions, and user satisfaction, making it a versatile tool for complaint management in diverse sectors.

## Table of Contents

S.No	Topics	Page No
1	Introduction	1
	1.1 Components of the MERN Stack in the Online Complaint Registration and Management System	2
2	Pre-Requirements	3
	2.1 Tools and Technologies	3
3	Project Overview	4
	3.1 Purpose	4
	3.2 Features and Functionalities	4
	3.3 Development workflow	4
4	Architecture	5
5	Setup Instruction	6
	5.1 Setup your Development Environment	6
	5.2 Initialise your Project	6
	5.3 Setup the back-end with express & Node.js	6
	5.4 Implement API endpoint	6
6	Configure Environment variable	7
	6.1 Front-end Directory Structure	7
	6.2 Back-end Directory Structure	7
7	Folder Structure	9
8	Running the Application	10

9	Extensions & Packages	12
	9.1 Back-end Packages	13
10	API Documentation	23
	10.1 Complaint Management Endpoint	23
	10.2 User Management Endpoint	23
	10.3 Complaint status Management Endpoint	24
	10.4 Notification Management Endpoint	25
11	Authentication	26
	11.1 User Registration	26
	11.2 Login	26
	11.3 Get user Profile	26
	11.4 Admin API Step	26
12	Authentication Mechanism	27
13	Authorization	28
14	User Interface and ScreenShot	29
15	Testing Strategy	31
16	Known Issues	32
17	Future Enhancement	33
18	Conclusion	34
19	Reference	35

<b>Fig. No</b>	<b>Fig Name</b>	<b>Page No</b>
4.1	Three Tire Architecture of MERN Stack	5
7.1	Back-end	8
7.2	Front-end	8
15.1	Landing Page	29
15.2	Login Page	29
15.3	Signup Page	30

## **1. INTRODUCTION**

Welcome to the Online Complaint Registration and Management System! This innovative platform is designed to revolutionize how complaints are managed, ensuring a seamless and efficient experience for both users and administrators. Our system addresses the need for a centralized solution to register, track, and resolve complaints while fostering transparency and prompt communication.

For users, the platform offers an intuitive interface where they can easily register, submit detailed complaints, and track their status in real-time. Features like notifications via email or SMS and an interactive chat interface with agents ensure a hassle-free and engaging experience.

For administrators and agents, the system provides robust backend functionality, enabling effective complaint routing, secure data storage, and seamless communication with users. Built with cutting-edge technologies like React.js for a responsive frontend and Node.js for a powerful backend, the system is supported by MongoDB for efficient and scalable database management.

Security is at the core of our platform. We prioritize protecting user data and ensuring secure transactions through advanced authentication methods, including JWT for identity verification.

Our Online Complaint Registration and Management System is not just a tool but a bridge towards enhancing trust and satisfaction among users and organizations. We are thrilled to have you on board and look forward to assisting you in resolving complaints efficiently and transparently.

## 1.1 Components of the MERN Stack in the Online Complaint Registration and Management System

### MongoDB (Database):

- MongoDB is utilized as the primary database to store all user information, complaint records, and agent details. Its document-oriented structure allows for flexible data storage, which is ideal for handling diverse data such as complaint descriptions, user profiles, and status updates.
- The database supports efficient querying and updating of data, ensuring that complaints can be tracked and resolved in real-time. MongoDB's scalability ensures the system performs reliably under heavy data loads.

### Express.js (Backend Framework):

- Express.js powers the backend server, handling the communication between the frontend and the database. It facilitates the creation of API endpoints for user registration, complaint submission, and real-time updates.
- By managing HTTP requests (GET, POST, PUT, DELETE), Express.js ensures data is securely and efficiently processed and served to the client side.

### React.js (Frontend Library):

- React.js drives the user interface of the application, providing an intuitive and responsive experience. Users can easily submit complaints, track their statuses, and communicate with agents through interactive components.
- React's component-based architecture ensures consistency in design and functionality across the platform. Tools like the Context API or state management libraries enhance the app's ability to manage user data and maintain seamless interactions.

### Node.js (Server-side JavaScript Runtime):

- Node.js serves as the runtime environment for the backend, enabling the system to handle multiple simultaneous requests without compromising performance.
- Its non-blocking, event-driven architecture is crucial for maintaining responsiveness in a platform designed for real-time complaint tracking and communication.

By combining these components, the MERN stack ensures that the **Online Complaint Registration and Management System** is scalable, efficient, and user-friendly, addressing the needs of both users and administrators effectively.

## 2. Pre-Requirements

- **HTML, CSS, and JavaScript:** Fundamental knowledge of HTML for structuring the application, CSS for styling, and JavaScript for interactivity is crucial. These form the foundation for developing a responsive user interface.
- **Database Connectivity:** Utilize MongoDB alongside an ODM like Mongoose to enable seamless communication between the backend server and the database. This ensures efficient CRUD operations for user data, complaint details, and agent assignments.
- **Frontend Library:** Use React.js to create a dynamic and user-friendly interface for the platform. This includes user dashboards, complaint forms, and chat interfaces.
- **Version Control:** Implement Git for tracking project changes and enabling team collaboration. Platforms like GitHub or GitLab can be used to manage repositories.
- **Git Download:** Instructions for installation can be found at [Git Downloads](#).
- **Development Environment:** Choose an IDE or text editor for efficient coding. Recommended options include:
  - [Visual Studio Code](#)
  - [Sublime Text](#)
  - [WebStorm](#)

### 2.1 Tools and Technologies

1. **Node.js and npm:** Install Node.js and npm for server-side development and dependency management.
2. **MongoDB:** Set up a MongoDB database, either locally or on a cloud platform like MongoDB Atlas.
3. **Text Editor/IDE:** Use a development tool such as Visual Studio Code for efficient coding and debugging.
4. **Version Control System:** Configure Git for version control and maintain a centralized repository on platforms like GitHub.
5. **API Testing Tools:** Employ tools such as Postman or Insomnia to test API endpoints and ensure backend functionality.
6. **Browser Development Tools:** Leverage browser-based developer tools to debug and optimize the frontend.

By following these requirements and utilizing the mentioned tools, developers can ensure the successful setup and implementation of the **Online Complaint Registration and Management System**.

### 3. Project Overview

#### 3.1 Purpose:

The **Online Complaint Registration and Management System** serves as a centralized platform for users to efficiently register, track, and resolve complaints. It ensures a streamlined process, fostering transparency, timely resolutions, and improved communication between users and administrators. The platform is designed to enhance user satisfaction while enabling organizations to manage and monitor complaints effectively.

#### 3.2 Features and Functionalities:

1. **User Authentication and Authorization:**
  - Secure user registration and login process using JWT authentication.
  - Role-based access for administrators and agents to ensure restricted access to specific functionalities.
2. **Complaint Submission and Tracking:**
  - Users can lodge complaints with detailed descriptions and attachments.
  - Real-time status tracking of complaints to keep users informed about the resolution progress.
3. **Communication Tools:**
  - Interactive chat interface for direct communication between users and agents.
  - Notifications via email or SMS for complaint updates and responses.
4. **Data Security and Privacy:**
  - Secure storage of user and complaint data using MongoDB.
  - Middleware to validate and protect data during API requests.

#### 3.3 Development Workflow:

1. **Frontend Development:**
  - Utilize React.js to build a responsive and intuitive user interface.
  - Incorporate state management tools such as Context API or Redux for consistent data handling.
  - Develop components like dashboards, complaint forms, and chat interfaces with a mobile-friendly design.
2. **Backend Development:**
  - Configure the Express.js server to manage API requests and handle complaint routing.
  - Establish database connections using MongoDB for efficient data storage and retrieval.
3. **Integration and Testing:**
  - Seamlessly integrate frontend and backend components to ensure a unified platform.
  - Perform comprehensive testing, including unit, integration, and end-to-end testing, to verify functionality, reliability, and security.



## 4.ARCHITECTURE

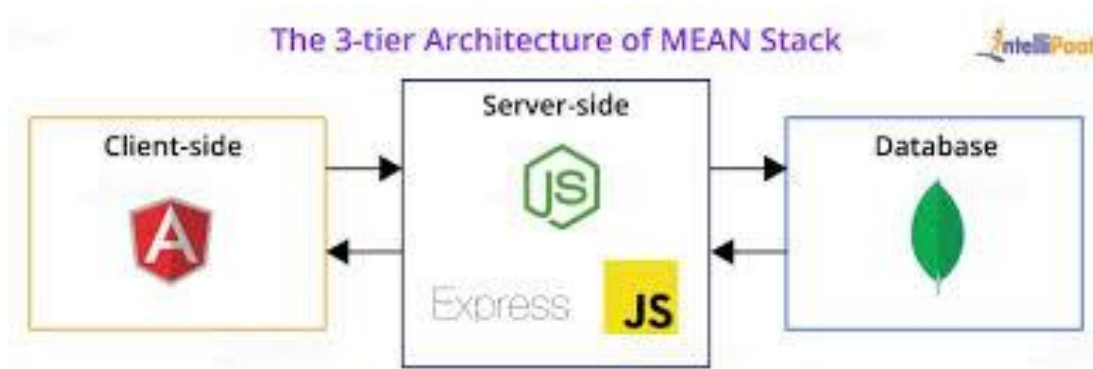


Fig 4.1 Architecture

### Frontend:

The frontend of the **Online Complaint Registration and Management System** is built using **React.js**, ensuring a responsive and interactive user experience. React components manage different functionalities such as user dashboards, complaint submission forms, and communication interfaces. The use of reusable components enhances maintainability and ensures consistency across the application.

### Backend:

The backend is powered by **Node.js** and **Express.js**, handling user requests, authentication, and business logic. It follows a **RESTful API approach** to enable seamless communication with the frontend. Express.js integrates efficiently with MongoDB, ensuring secure and efficient data handling for complaints, user profiles, and agent assignments. Middleware is utilized for authentication, data validation, and request routing.

### Database:

**MongoDB** serves as the database, providing a scalable and flexible data model to store user information, complaint records, and agent details. Its document-oriented architecture supports efficient querying and updates, making it ideal for real-time complaint tracking and resolution. MongoDB ensures the application can manage high traffic and large volumes of data without compromising performance.

## 5. Setup Instructions

### 5.1 Set Up Your Development Environment:

1. **Install Node.js and npm:** Ensure Node.js and npm (Node Package Manager) are installed on your machine.
2. **Install MongoDB:** Set up a local MongoDB instance or use a cloud-based service such as MongoDB Atlas.

### 5.2 Initialize Your Project:

1. **Create a New Project Directory:**
2. `mkdir complaint-system && cd complaint-system`
3. **Initialize npm:**
4. `npm init -y`

### 5.3 Set Up the Backend with Express and Node.js:

1. **Install Backend Dependencies:**
  - **Express.js:** `npm install express`
  - **Mongoose:** `npm install mongoose`
  - **Cors:** `npm install cors`
  - **JWT:** `npm install jsonwebtoken`
  - **bcrypt:** `npm install bcryptjs`
  - **dotenv:** `npm install dotenv`
  - **Multer:** `npm install multer`
2. **Create Server File:** Create a `server.js` file to configure a basic Express server.
3. **Connect to MongoDB:** Use Mongoose to connect the backend to the MongoDB database.

### 5.4 Implement API Endpoints:

1. **Create Routes Directory:**
2. `mkdir routes`
3. **Define API Routes:** Implement routes for user authentication, complaint management, and real-time updates.
4. **Set Up Middleware:** Configure middleware for authentication, authorization, and request validation.

## 6. Configure Environment Variables

### Create a .env File:

- Store sensitive information such as the database connection URI, JWT secret, and server port in a .env file located in the backend directory.

### Example Variables in .env:

PORT=5000

MONGODB\_URI=<your-mongodb-connection-string>

JWT\_SECRET=<your-jwt-secret-key>

### Load Environment Variables:

1. **Install dotenv Package:**
2. npm install dotenv
3. **Access Environment Variables:** Use process.env to retrieve variables in your application wherever necessary.

### 6.1 Frontend Directory Structure:

- **src/:** Contains reusable React components for UI elements.
- **src/pages/:** Houses pages such as User Dashboard, Agent Dashboard, and Admin Panel.
- **src/app.js:** Manages API calls and facilitates frontend-backend communication.

### 6.2 Backend Directory Structure:

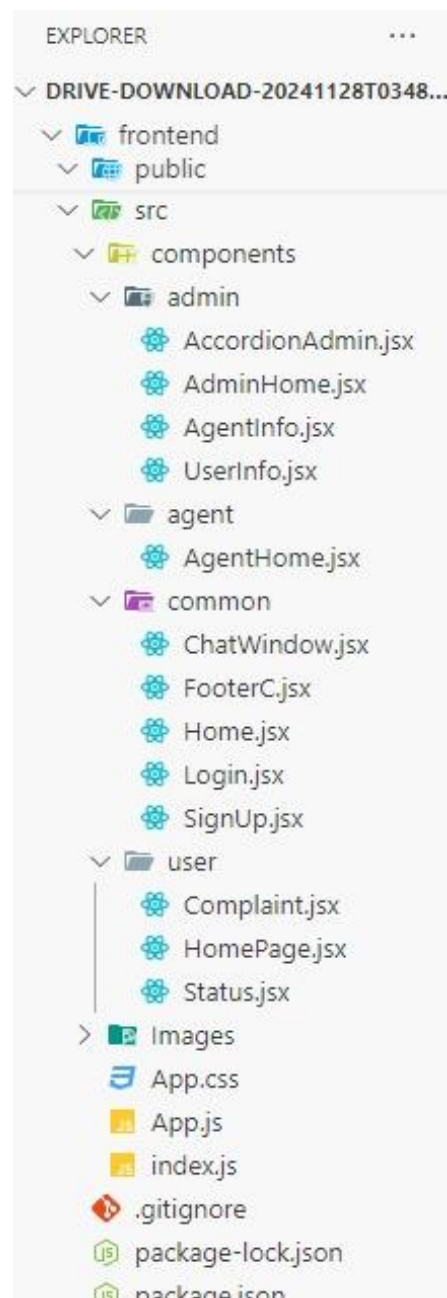
- **routes/:** Holds route files for user, complaint, and agent APIs.
- **models/:** Contains Mongoose schemas and models for MongoDB collections.
- **controllers/:** Includes logic for handling API requests and business processes.

## 7.FOLDER STRUCTURE

### Backend



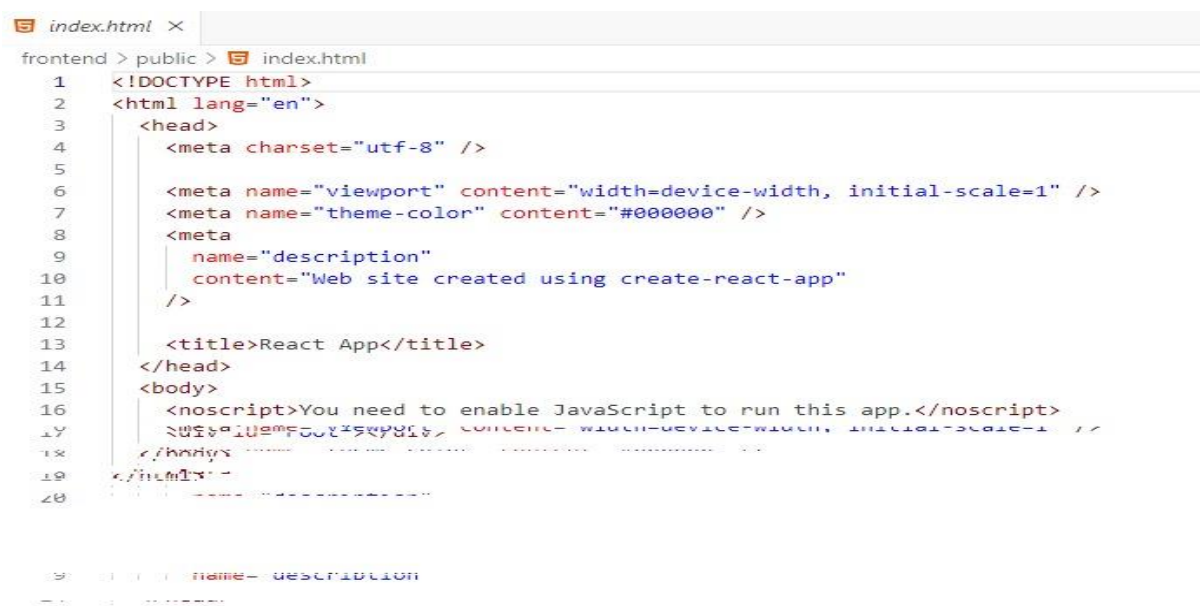
### Frontend



## 8.RUNNING THE APPLICATION

### Frontend:

- To navigate to front end use the command `cd frontend` in terminal.
- Navigate to the frontend directory and run `npm run dev` to start the React application.



```
index.html x
frontend > public > index.html
1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <meta charset="utf-8" />
5
6      <meta name="viewport" content="width=device-width, initial-scale=1" />
7      <meta name="theme-color" content="#000000" />
8      <meta
9        name="description"
10       content="Web site created using create-react-app"
11     />
12
13     <title>React App</title>
14   </head>
15   <body>
16     <noscript>You need to enable JavaScript to run this app.</noscript>
17     <!-- You can add a meta tag to enable TypeScript -->
18     <!-- <script src="main.js"></script> -->
19   </body>
20 </html>
```

### Backend:

- To navigate to backend use the command `cd backend` in terminal.
- Navigate to the backend directory and run `npm start` to start the Express.js backend.

```
index.js x
backend > index.js > ...
1 const express = require("express");
2 const cors = require("cors");
3 require("./config");
4 const {
5   ComplaintSchema,
6   UserSchema,
7   AssignedComplaint,
8   MessageSchema,
9 } = require("../Schema");
10 const app = express();
11 const PORT = 8000;
12
13 /*****message *****/
14 app.use(express.json());
15 app.use(cors());
16 /***** */
17
18 /*****message *****/
19 app.post("/messages", async (req, res) => {
20   try {
21     const { name, message, complaintId } = req.body;
22     const messageData = new MessageSchema({
23       name,
24       message,
25       complaintId,
26     });
27     const messageSaved = await messageData.save();
28     res.status(200).json(messageSaved);
29   } catch (error) {
30     res.status(500).json({ error: "Failed to send message" });
31   }
32 });
```

## 9. Extensions & Packages

To build the **Online Complaint Registration and Management System** using the MERN stack, several essential packages and tools are required to streamline development and enhance functionality.

### Frontend Packages:

- **React.js:** To build a dynamic and responsive user interface.
- **React Router:** For seamless navigation between different pages, such as dashboards and complaint submission forms.
- **State Management:** Use Redux or React Context API to handle global states like user authentication and complaint data.
- **Axios:** For making HTTP requests and managing API interactions.
- **Material-UI:** Provides pre-built UI components to speed up the design process.
- **Formik and Yup:** Useful for form handling and validation, ensuring smooth user input management.

### Backend Packages:

- **Express.js:** To set up the backend server and manage API endpoints.
- **Mongoose:** For connecting and interacting with MongoDB.
- **Cors:** To handle cross-origin requests between the frontend and backend.
- **JWT (jsonwebtoken):** For secure user authentication.
- **bcryptjs:** To hash and secure user passwords.
- **Multer:** For handling file uploads, such as complaint attachments.
- **dotenv:** For managing environment variables securely.

## Development Tools:

- **Nodemon:** Automatically restarts the server during development when changes are detected.
- **ESLint and Prettier:** For maintaining code quality and consistent formatting.
- **Jest:** To perform unit tests for both frontend and backend components.
- **Postman or Insomnia:** For API testing and debugging.
- **Docker:** For containerizing the application to ensure consistent deployment environments.

## IDE Extensions for VSCode:

- **ESLint and Prettier:** Maintain clean and formatted code.
- **GitLens:** Enhances version control visibility and Git workflow integration.
- **Path Intellisense:** Provides autocompletion for file paths, improving productivity.
- **REST Client:** For directly testing API endpoints within the IDE.

These packages and tools collectively facilitate the development, testing, and deployment of a robust and scalable **Online Complaint Registration and Management System**, ensuring a seamless development experience and high-quality output.

## 9.1 Backend Packages:

### Package.json

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "body-parser": "^1.19.0",
    "cors": "^2.8.5",
    "express": "^4.17.1",
    "jsonwebtoken": "^8.5.1",
    "mongoose": "^5.11.18",
    "nodemon": "^2.0.7"
  }
}
```

}

## Index.html

```
index.html x
frontend > public > index.html
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <meta
9       name="description"
10      content="Web site created using create-react-app"
11    />
12
13    <title>React App</title>
14  </head>
15  <body>
16    <noscript>You need to enable JavaScript to run this app.</noscript>
17    <div id="root"></div>
18  </body>
19 </html>
20
```

name=description



## App.js

```
App.js ×
frontend > src > App.js > ...
1 import './App.css';
2 import '../node_modules/bootstrap/dist/css/bootstrap.min.css';
3 import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
4
5 import HomePage from './components/user/HomePage';
6 import Login from './components/common/Login';
7 import SignUp from './components/common/SignUp';
8 import Complaint from './components/user/Complaint';
9 import Status from './components/user/Status';
10 import AdminHome from './components/admin/AdminHome';
11 import AgentHome from './components/agent/AgentHome';
12 import UserInfo from './components/admin/UserInfo';
13 import Home from './components/common/Home';
14 import AgentInfo from './components/admin/AgentInfo';
15
16 Codeium: Refactor | Explain | Generate JSDoc | X
17 function App() {
18   const isLoggedIn = !!localStorage.getItem("user");
19   return (
20     <div className="App">
21       <Router>
22         <Routes>
23           <Route exact path="/" element={<Home />} />
24           <Route path="/Login" element={<Login />} />
25           <Route path="/SignUp" element={<SignUp />} />
26           {isLoggedIn ? (
27             <>
28               <Route path="/AgentInfo" element={<AgentInfo />} />
29               <Route path="/AgentHome" element={<AgentHome />} />
30               <Route path="/UserInfo" element={<UserInfo />} />
31               <Route path="/AdminHome" element={<AdminHome />} />
32               <Route path="/HomePage" element={<HomePage />} />
33             </>
34           ) : null}
35         </Routes>
36       </Router>
37     </div>
38   );
39 }
```

## Package-jason:

```
package.json X
frontend > package.json
1  {
2    "name": "task1",
3    "version": "0.1.0",
4    "proxy": "http://localhost:8000",
5    "private": true,
6    "dependencies": {
7      "@emotion/react": "^11.11.1",
8      "@emotion/styled": "^11.11.0",
9      "@testing-library/jest-dom": "^5.16.5",
10     "@testing-library/react": "^13.4.0",
11     "@testing-library/user-event": "^13.5.0",
12     "axios": "^1.4.0",
13     "bootstrap": "^5.2.3",
14     "mdb-react-ui-kit": "^6.1.0",
15     "react": "^18.2.0",
16     "react-bootstrap": "^2.7.4",
17     "react-dom": "^18.2.0",
18     "react-router-dom": "^6.11.2",
19     "react-scripts": "5.0.1",
20     "web-vitals": "^2.1.4"
21   },
22   "scripts": {
23     "start": "react-scripts start",
24     "build": "react-scripts build",
25     "test": "react-scripts test",
26     "eject": "react-scripts eject"
27   },
28   "eslintConfig": {
29     "extends": [
30       "react-app",
31       "react-app/jest"
```

## homepage.json:

```
HomePage.jsx X
frontend > src > components > user > HomePage.jsx > ...
1 import React, { useEffect, useState } from 'react';
2 import { NavLink, useNavigate } from 'react-router-dom';
3 import Footer from '../common/FooterC';
4 import Complaint from '../user/Complaint';
5 import Status from '../user/Status';
6
7 const HomePage = () => {
8   const navigate = useNavigate();
9   const [activeComponent, setActiveComponent] = useState('Complaint');
10  const [userName, setUserName] = useState('');
11
12  useEffect(() => {
13    Codeium: Refactor | Explain | Generate JSDoc | X
14    const getData = async () => {
15      try {
16        const user = JSON.parse(localStorage.getItem('user'));
17        if (user) {
18          const { name } = user;
19          setUserName(name);
20        } else {
21          navigate('/');
22        }
23      } catch (error) {
24        console.log(error);
25      }
26    };
27    getData();
28  }, [navigate]);
29
```

## Status.js:

```
Status.jsx X
frontend > src > components > user > Status.jsx > ...
1 import axios from 'axios';
2 import React, { useEffect, useState } from 'react';
3 import Card from 'react-bootstrap/Card';
4 import Alert from 'react-bootstrap/Alert';
5 import { Button } from 'react-bootstrap';
6 import ChatWindow from '../common/ChatWindow';
7 import Collapse from 'react-bootstrap/Collapse';
8
9 Codeium: Refactor | Explain | Generate JSDoc | X
10 const Status = () => {
11   const [toggle, setToggle] = useState({});
12   const [statusCompliants, setStatusCompliants] = useState([]);
13   useEffect(() => {
14     const user = JSON.parse(localStorage.getItem('user'));
15     const { _id } = user;
16
17     axios.get(`http://localhost:8000/status/${_id}`)
18       .then((res) => {
19         setStatusCompliants(res.data);
20       })
21       .catch((err) => {
22         console.log(err);
23       });
24   }, []);
25
26   Codeium: Refactor | Explain | Generate JSDoc | X
27   const handleToggle = (complaintId) => {
```

## Home.js:

```
Home.jsx X
frontend > src > components > common > Home.jsx > ...
1 import React from 'react'
2 import Navbar from 'react-bootstrap/Navbar';
3 import Container from 'react-bootstrap/Container';
4 import Image1 from '../Images/Image1.png'
5 import { Link } from 'react-router-dom';
6 import Button from 'react-bootstrap/Button';
7 import Footer from './FooterC'
8
Codeium: Refactor | Explain | Generate JSDoc | X
9 const Home = () => {
10   return (
11     <>
12       <Navbar bg="dark" variant="dark">
13         <Container>
14           <Navbar.Brand>ComplaintCare </Navbar.Brand>
15           <ul className="navbar-nav">
16             <li className="nav-item mb-2">
17               <Link to={'/'}>
18                 className={`nav-link text-light`}
19               >
20                 Home
21               </Link>
22             </li>
23             /* <li className="nav-item mb-2">
24               <Link
25                 to={'/About'}
26                 className={`nav-link text-light`}
27               >
28                 About
29               </Link>
30             </li> */
31             <li className="nav-item mb-2">
32               <Link
```

## Login.jsx:

```
Login.jsx X
frontend > src > components > common > Login.jsx > ...
1 import axios from 'axios';
2 import React, { useState } from 'react';
3 import { Link, useNavigate } from 'react-router-dom';
4 import Container from 'react-bootstrap/Container';
5 import Navbar from 'react-bootstrap/Navbar';
6 import Footer from './FooterC'
7
Codeium: Refactor | Explain | Generate JSDoc | X
8 const Login = () => {
9   const navigate = useNavigate();
10   const [user, setUser] = useState({
11     email: "",
12     password: ""
13   });
14
Codeium: Refactor | Explain | Generate JSDoc | X
15 const handleChange = (e) => {
16   const { name, value } = e.target;
17   setUser({ ...user, [name]: value });
18 };
19
Codeium: Refactor | Explain | Generate JSDoc | X
20 const handleSubmit = async (e) => {
21   e.preventDefault();
22   await axios.post("http://localhost:8000/Login", user)
23     .then((res) => {
24       alert("Successfully logged in");
25     })
26     .catch((err) => {
27       alert("Error: " + err.message);
28     })
29   };
30 }
```

## Signup.jsx:

```
frontend > src > components > common > Signup.jsx > ...  
1 import axios from 'axios'  
2 import React, { useState } from 'react'  
3 import { Link } from 'react-router-dom'  
4 import Dropdown from 'react-bootstrap/Dropdown';  
5 import Container from 'react-bootstrap/Container';  
6 import Navbar from 'react-bootstrap/Navbar';  
7 import Footer from './FooterC'  
Codeium: Refactor | Explain | Generate JSDoc | X  
8 const Signup = () => {  
9   const [title, setTitle] = useState("Select User")  
10  const [user, setUser] = useState({  
11    name: "",  
12    email: "",  
13    password: "",  
14    phone: "",  
15    userType: ""  
16  })  
Codeium: Refactor | Explain | Generate JSDoc | X  
17  const handleChange = (e) => {  
18    setUser({ ...user, [e.target.name]: e.target.value })  
19  }  
20  
Codeium: Refactor | Explain | Generate JSDoc | X  
21  const handleTitle = (select) => {  
22    setTitle(select)  
23    setUser({ ...user, userType: select });  
24  }
```

## Chatwindow.jsx:



```

ChatWindow.jsx X
frontend > src > components > common > ChatWindow.jsx > ...
1  import React, { useState, useEffect, useRef } from 'react'
2  import axios from 'axios';
3
4
Codeium: Refactor | Explain | Generate JSDoc | X
5  const ChatWindow = (props) => {
6      const [messageInput, setMessageInput] = useState('');
7
8      const messageWindowRef = useRef(null);
9      const [messageList, setMessageList] = useState([]);
10
Codeium: Refactor | Explain | Generate JSDoc | X
11  const fetchMessageList = async () => {
12      try {
13          const response = await axios.get(`http://localhost:8000/messages/${props.complaintId}`);
14          setMessageList(response.data);
15      } catch (error) {
16          console.error('Error fetching messages:', error);
17      }
18  };
19
20  useEffect(() => {
21      fetchMessageList(props.complaintId, setMessageList);
22  }, [props.complaintId]);
23

```

## AgentHome:

```

AgentHome.jsx X
frontend > src > components > agent > AgentHome.jsx > ...
1  import React, { useState, useEffect } from 'react';
2  import Button from 'react-bootstrap/Button';
3  import Container from 'react-bootstrap/Container';
4  import Nav from 'react-bootstrap/Nav';
5  import Navbar from 'react-bootstrap/Navbar';
6  import Card from 'react-bootstrap/Card';
7  import { NavLink, useNavigate } from 'react-router-dom';
8  import axios from 'axios';
9  import Alert from 'react-bootstrap/Alert';
10 import Collapse from 'react-bootstrap/Collapse';
11 import ChatWindow from '../common/ChatWindow';
12 import Footer from '../common/FooterC';
13
Codeium: Refactor | Explain | Generate JSDoc | X
14 const AgentHome = () => {
15     const style = {
16         marginTop: '66px',
17     }
18
19     const navigate = useNavigate();
20     const [userName, setUserName] = useState('');
21     const [toggle, setToggle] = useState({})
22     const [agentComplaintList, setAgentComplaintList] = useState([]);
23     import React, { useState, useEffect } from 'react';
24     useEffect(() => {
25         const getData = async () => {
26             try {
27                 const user = JSON.parse(localStorage.getItem('user'));
28                 if (user) {
29                     const { id, name, isAgent } = user;
30                     setUserName(name);
31                     const response = await axios.get(`http://localhost:8000/allcomplaints/${id}`);
32

```

## AdminHome

```
AdminHome.jsx ×
import React, { useEffect, useState } from 'react';
import Button from 'react-bootstrap/Button';
import Container from 'react-bootstrap/Container';
import Nav from 'react-bootstrap/Nav';
import Navbar from 'react-bootstrap/Navbar';
import { NavLink, useNavigate } from 'react-router-dom';
import { AccordionAdmin } from './AccordionAdmin';

const AdminHome = () => {
  const navigate = useNavigate();
  const [activeComponent, setActiveComponent] = useState('dashboard');
  const [userName, setUserName] = useState('');

  const [user, setUser] = useState({});

  useEffect(() => {
    const user = JSON.parse(localStorage.getItem('user'));
    if (user) {
      setUserName(user.name);
    }
  }, []);

  return (
    <div>
      <Navbar>
        <Nav>
          <NavLink to="/dashboard">Dashboard</NavLink>
          <NavLink to="/users">Users</NavLink>
          <NavLink to="/products">Products</NavLink>
          <NavLink to="/orders">Orders</NavLink>
          <NavLink to="/reports">Reports</NavLink>
        </Nav>
      </Navbar>
      <Container>
        <AccordionAdmin>
          <div>Dashboard</div>
          <div>Users</div>
          <div>Products</div>
          <div>Orders</div>
          <div>Reports</div>
        </AccordionAdmin>
      </Container>
    </div>
  );
};

export default AdminHome;
```

frontend > src > components > admin > AccordionAdmin.jsx > ...

```

1  import React, { useState, useEffect } from 'react'
2  import Accordion from 'react-bootstrap/Accordion';
3  import Card from 'react-bootstrap/Card';
4  import Dropdown from 'react-bootstrap/Dropdown';
5  import Alert from 'react-bootstrap/Alert';
6  import Footer from '../common/FooterC'
7  import axios from 'axios';
8
Codeium: Refactor | Explain | Generate JSDoc | X
9  const AccordionAdmin = () => {
10     const [complaintList, setComplaintList] = useState([]);
11     const [agentList, setAgentList] = useState([]);
12     useEffect(() => {
Codeium: Refactor | Explain | Generate JSDoc | X
13         const getComplaints = async () => {
14             try {
15                 const response = await axios.get('http://localhost:8000/status');
16                 const complaints = response.data;
17                 setComplaintList(complaints);
18             } catch (error) {
19                 console.log(error);
20             }
21         };
22         getComplaints();
23
Codeium: Refactor | Explain | Generate JSDoc | X
24     const getAgentsRecords = async () => {
25         try {

```



## Agentinfo

AgentInfo.jsx ×

frontend > src > components > admin > AgentInfo.jsx > ...

```
1 import React, { useEffect, useState } from 'react';
2 import Button from 'react-bootstrap/Button';
3 import { useNavigate } from 'react-router-dom';
4 import Table from 'react-bootstrap/Table';
5 import Alert from 'react-bootstrap/Alert';
6 import { Container } from 'react-bootstrap';
7 import Collapse from 'react-bootstrap/Collapse';
8 import Form from 'react-bootstrap/Form';
9 import Footer from '../common/footer';
10 import axios from 'axios';
11
12 Codeium: Refactor | Explain | Generate JSDoc | X
13 const AgentInfo = () => {
14   const navigate = useNavigate();
15   const [ordinaryList, setOrdinaryList] = useState([]);
16   const [toggle, setToggle] = useState({})
17   const [updateAgent, setUpdateAgent] = useState({
18     name: '',
19     email: '',
20     phone: '',
21   })
22
23   Codeium: Refactor | Explain | Generate JSDoc | X
24   const handleChange = (e) => {
25     setUpdateAgent({ ...updateAgent, [e.target.name]: e.target.value })
26   }
```

## 10. API Documentation

### 10.1 Complaint Management Endpoints

1. **Submit a Complaint:**
  - **Method:** POST /complaints
  - **Description:** Submits a new complaint by a user with details and attachments.
  - **Request Body:**
    - {
    - "title": "Complaint Title",
    - "description": "Detailed complaint description",
    - "attachments": [ "file1.jpg", "file2.pdf" ],
    - "userId": "user123"
    - }
2. **View All Complaints:**
  - **Method:** GET /complaints
  - **Description:** Retrieves a list of all complaints in the system.
3. **View a Single Complaint:**
  - **Method:** GET /complaints/{id}
  - **Description:** Retrieves details of a specific complaint by its ID.
  - **Response:**
    - {
    - "complaintId": "complaint123",
    - "title": "Complaint Title",
    - "description": "Complaint details",
    - "status": "Pending",
    - "createdAt": "2024-11-01T10:00:00Z",
    - "attachments": [ "file1.jpg", "file2.pdf" ]
    - }
4. **Update a Complaint:**
  - **Method:** PUT /complaints/{id}
  - **Description:** Updates the details of a specific complaint.
  - **Request Body:**
    - {
    - "title": "Updated Complaint Title",
    - "description": "Updated complaint details",
    - "status": "Resolved"
    - }
5. **Delete a Complaint:**
  - **Method:** DELETE /complaints/{id}
  - **Description:** Deletes a complaint from the system

### 10.2 User Management Endpoints

1. **Register a User:**
  - **Method:** POST /users/register
  - **Description:** Registers a new user in the system.
  - **Request Body:**
    - {
    - "name": "User Name",

- "email": "user@example.com",
- "password": "userpassword"
- }
- 2. **Login a User:**
  - **Method:** POST /users/login
  - **Description:** Authenticates a user and generates an authentication token.
  - **Request Body:**
    - {
    - "email": "user@example.com",
    - "password": "userpassword"
    - }
  - **Response:**
    - {
    - "status": "success",
    - "token": "JWT\_Token\_String"
    - }
- 3. **Get User Details:**
  - **Method:** GET /users/{id}
  - **Description:** Retrieves details of a specific user by their ID.
- 4. **Update User Details:**
  - **Method:** PUT /users/{id}
  - **Description:** Updates the details of a specific user.
  - **Request Body:**
    - {
    - "name": "Updated User Name",
    - "email": "updatedemail@example.com"
    - }
- 5. **Delete a User:**
  - **Method:** DELETE /users/{id}
  - **Description:** Removes a user from the system.

### 10.3 Complaint Status Management Endpoints

1. **Assign Agent to Complaint:**
  - **Method:** POST /complaints/{id}/assign-agent
  - **Description:** Assigns an agent to a specific complaint.
  - **Request Body:**
    - {
    - "agentId": "agent123"
    - }
2. **Update Complaint Status:**
  - **Method:** PUT /complaints/{id}/status
  - **Description:** Updates the status of a specific complaint (e.g., Pending, Resolved, In Progress).
  - **Request Body:**
    - {
    - "status": "Resolved"
    - }
3. **Get Complaint Status:**
  - **Method:** GET /complaints/{id}/status

- **Description:** Retrieves the current status of a specific complaint.

## 10.4 Notification Management Endpoints

### 1. Send Complaint Update Notification:

- **Method:** POST /notifications/complaint-update
- **Description:** Sends an update notification to the user when their complaint status changes.
- **Request Body:**
  - {
  - "complaintId": "complaint123",
  - "userId": "user123",
  - "message": "Your complaint has been resolved."
  - }

### 2. View Notification:

- **Method:** GET /notifications/{id}
- **Description:** Retrieves a specific notification by its ID.

This API documentation provides all the necessary endpoints for managing complaints, users, statuses, and notifications in the **Online Complaint Registration and Management System**, ensuring a smooth process for both users and administrators.

## 11. Authentication

### 11.1 User Registration:

- **User Action:**
  - User sends a request with their details (name, email, password, etc.) to register.
- **System Process:**
  - The system validates the input data for required fields and correct format.
  - Password is hashed before storing to ensure security.
- **System Response:**
  - If validation passes, the system creates a new user account.
  - The system responds with a success message and user ID.

### 11.2 User Login:

- **User Action:**

- User sends login credentials (email and password).
- **System Process:**
  - The system verifies the provided credentials by checking the email and comparing the hashed password stored in the database.
- **System Response:**
  - If credentials are valid, a JWT (JSON Web Token) or similar token is generated and returned.
  - The user can now use the token to make authenticated requests to the system.

### 11.3 Get User Profile:

- **User Action:**
  - User sends a request with their authentication token to view their profile.
- **System Process:**
  - The system verifies the token and retrieves the user details from the database.
- **System Response:**
  - The system returns the user's profile information (name, email, complaint history, etc.).

### 11.4 Admin API Steps:

1. **Admin Login:**
  - **Admin Action:**
    - Admin provides login credentials to authenticate.
  - **System Process:**
    - The system verifies the credentials.
  - **System Response:**
    - If valid, a JWT is generated for admin-level access.
2. **View All Users:**
  - **Admin Action:**
    - Admin sends a request with their authentication token to fetch all user records.
  - **System Process:**
    - The system verifies the token and retrieves all user details.
  - **System Response:**
    - The system returns a list of all users.
3. **Delete a User:**
  - **Admin Action:**
    - Admin sends a request to delete a specific user by providing the user ID.
  - **System Process:**
    - The system verifies the admin token, checks if the user exists, and deletes the user record.
  - **System Response:**
    - If valid, the user is deleted and the system returns a confirmation response.

## 12. Authentication Mechanism

Authentication is managed using **JSON Web Tokens (JWT)**. When a user logs in, a JWT is generated and sent to the frontend to be stored locally (in cookies or localStorage).

### 12.1 User Registration:

- **Endpoint:** POST /register
- **Process:** Users provide their details (name, email, password) to create an account.
- **Data Handling:** Passwords are hashed before being stored in the database to enhance security.

### 12.2 User Login:

- **Endpoint:** POST /login
- **Process:** Users submit their email and password.
- **Verification:** The server verifies that the email exists and the password matches the hashed version in the database.
- **Token Generation:** On successful login, a **JWT** is generated and sent back to the user.

### 12.3 Token-Based Authentication:

- **JWT:** The token contains user information and a secret key, ensuring secure verification.
- **Storage:** The JWT is stored in the client's localStorage or cookies.
- **Authorization Header:** For subsequent requests, the JWT is included in the Authorization header as Bearer <token>.

### 12.4 Middleware Validation:

- **Token Validation:** Middleware intercepts requests to protected routes, verifying the token's validity.
- **Access Control:** If the token is valid, the request proceeds; otherwise, access is denied.

### 12.5 Database Setup:

- **Database:** MongoDB is used to store user credentials and related information.
- **Schema Design:** The schema includes fields like username, email, password, and role, with Mongoose models for users.

## 13. Authorization

Authorization ensures that users can only perform actions and access resources they are permitted to based on their role within the system. It is different from authentication, which verifies a user's identity.

### 13.1 Key Components:

#### 1. User Roles:

- **Admin:** Has full control over the system, including managing users, complaints, and system settings.
- **User:** Can submit complaints, track complaint statuses, and communicate with agents.
- **Agent:** Handles complaints assigned to them, communicates with users, and updates complaint statuses.

#### 2. Permissions:

- Each role is assigned specific permissions that define what actions they can perform within the application.
- **Admin:** Can manage users, complaints, and system configurations.
- **User:** Can create, view, and update their own complaints and view their complaint history.
- **Agent:** Can manage complaints assigned to them and communicate with users.

### 13.2 Implementation Overview:

#### 1. Defining Roles and Permissions:

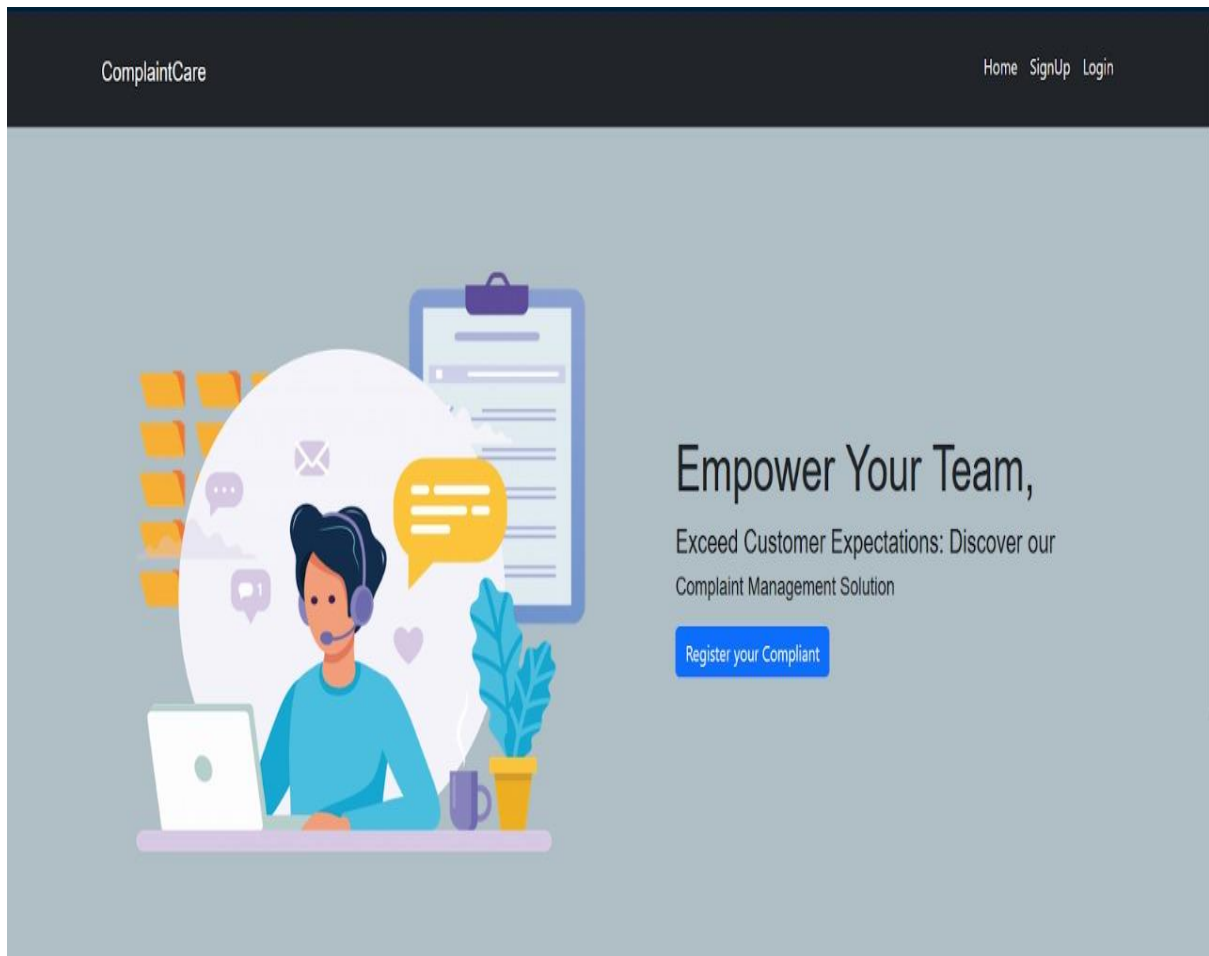
- Roles are defined based on system needs (Admin, User, Agent).
- Permissions are mapped to each role to ensure users can only access and perform actions relevant to their role.

#### 2. Assigning Roles to Users:

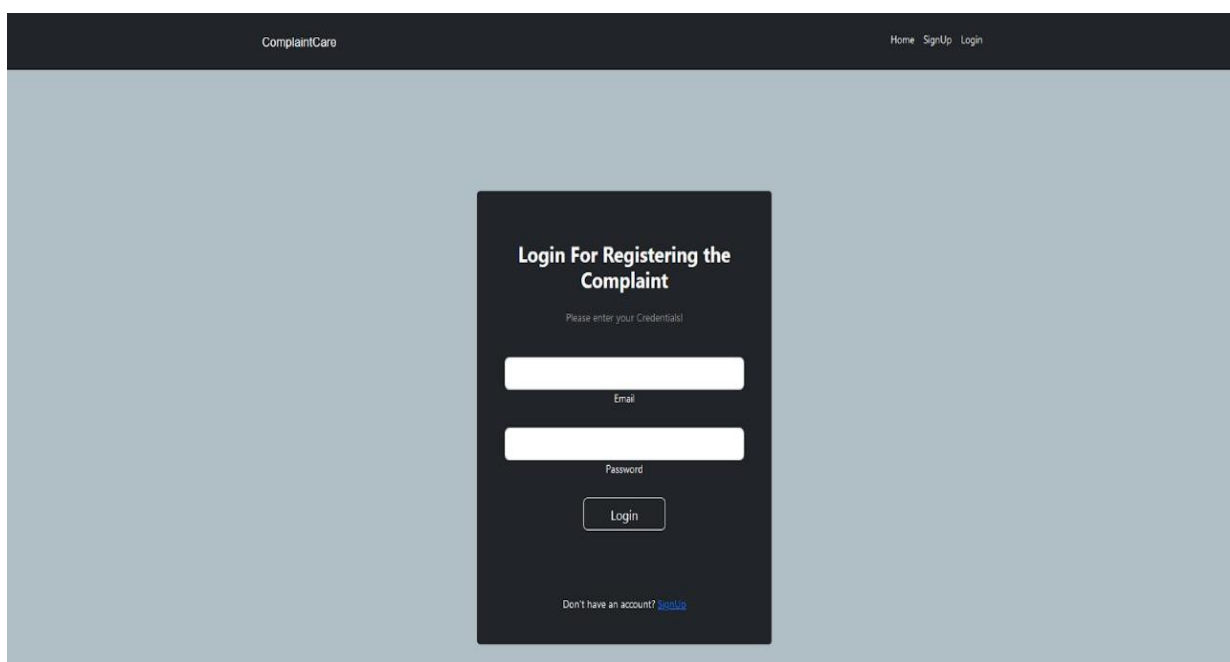
- When a user registers or is created, they are assigned a role based on their function in the system (e.g., user, agent).
- The role information is securely stored in the user's profile to control access and permissions.

## 15.USER INTERFACE AND SCREENSHOT

### Landing Page



### LOGIN PAGE:





SIGNUP PAGE:

ComplaintCare

HomeSignUpLogin

SignUp For Registering the Complaint

Please enter your Details

Full Name

Email

Password

Mobile No.

Select User \*

Select User Type

Register

Had an account? [Login](#)

Hi Admin shadeelDashboardUserAgent

Log out

Users Complaints

Name: sad

Address: sadad

City: dsadba

State: sadad

Pincode: 232131

Comment: dsadada

Status: dsd

Assign

Agents

No Agents to show

ComplaintCare

© 2024

Hi Agent sadView Complaints

Log out

Name: sad

Address: sadad

City: dsadba

State: sadad

Pincode: 232131

Comment: dsadada

Status: dsd

Status Change

Message

Message Box

Message

Send

ComplaintCare

© 2024

## 16. TESTING STRATEGY

1. **Unit Testing:**
  - **Purpose:** Ensure individual components, functions, and classes perform as expected.
  - **Scope:** Test features like user registration, complaint submission, and status updates in isolation.
  - **Tools:** Jest, Mocha, Chai (for JavaScript testing).
2. **Integration Testing:**
  - **Purpose:** Validate seamless communication between different modules and services.
  - **Scope:** Test interactions between the frontend, backend, and the database, including user-agent communication and complaint tracking.
  - **Tools:** Supertest (for API testing), Postman (for manual API testing).
3. **End-to-End (E2E) Testing:**
  - **Purpose:** Simulate real-world scenarios to test complete workflows like complaint submission and resolution tracking.
  - **Scope:** Validate the entire application flow from user login to complaint resolution.
  - **Tools:** Cypress, Selenium, Puppeteer.
4. **Performance Testing:**
  - **Purpose:** Assess the application's responsiveness and stability under load.
  - **Scope:** Test how the system performs during high complaint submission rates or under concurrent user activity.
  - **Tools:** Apache JMeter, LoadRunner.
5. **Security Testing:**
  - **Purpose:** Identify vulnerabilities and secure the system from potential threats.
  - **Scope:** Test for issues like SQL injection, XSS, CSRF, and ensure sensitive user data is protected.
  - **Tools:** OWASP ZAP, Burp Suite.

## 17. KNOWN ISSUES

- **Slow Loading Times:** High server response times or unoptimized queries may cause delays in complaint status updates.
- **Security Vulnerabilities:** Inadequate protection could expose sensitive user data or allow unauthorized access.
- **Notification Delays:** Email or SMS notifications may be delayed under high traffic conditions.
- **Database Performance:** Handling large volumes of complaint data can lead to slower query execution if proper indexing isn't applied.
- **Frontend-Backend Integration:** Issues in communication between the React.js frontend and Node.js/Express.js backend, particularly with real-time status updates.
- **Scalability:** Managing increased user and complaint data requires solutions like load balancing and cloud hosting.
- **User Experience (UX):** Complex or unintuitive interfaces might hinder user navigation and satisfaction. Regular feedback is crucial.
- **Error Handling:** Missing or vague error messages can confuse users. Proper error handling improves both user experience and debugging efficiency.
- **Deployment and Maintenance:** Setting up CI/CD pipelines, ensuring smooth deployments, and maintaining the app can be challenging. Tools like Docker or cloud platforms like AWS can assist.
- **Agent Assignments:** Ensuring agents are automatically assigned complaints efficiently and without overlap remains a challenge under high load.

## 18.FUTURE ENHANCEMENTS

- **Advanced Search and Filtering:**

Implement more sophisticated search capabilities, allowing users to filter books by various criteria such as genre, author, price range, publication date, and ratings. This can help users find exactly what they are looking for with ease.

- **Personalized Recommendations:** Develop a recommendation engine that suggests books

based on user preferences, reading history, and purchase patterns. Machine learning algorithms can be employed to analyze user behaviour and provide tailored book recommendations.

- **Social Features:** Integrate social features like book clubs, discussion forums, and the ability for users to share their reading lists and reviews on social media platforms. foster a sense of community and increase user engagement.

- **Mobile App Development:** Create a mobile application for your bookstore to seamless and optimized experience for users on the go. React Native can be used to cross-platform mobile apps.

- **Subscription Service:** Offer a subscription-based model where users can subscribe to receive a certain number of books or access exclusive content each month. This can provide a steady revenue stream and enhance user loyalty.

- **Enhanced Payment Options:** Integrate multiple payment gateways to offer users a variety of payment options, including digital wallets, credit/debit cards, and bank transfers. This can improve the checkout experience and cater to a wider audience.

- **Voice Search and Assistants:** Implement voice search capabilities and integrate with virtual assistants like Amazon Alexa or Google Assistant. This can make the user experience more interactive and accessible.

- **Analytics and Insights:** Develop an analytics dashboard for administrators to track user behaviour, sales trends, and other key metrics. This can help in making informed business decisions and optimizing the bookstore's performance.

- **Localization and Multilingual Support:** Expand your reach by adding support for multiple languages and localizing content to cater to users from different regions.

- **Sustainability Initiatives:** Introduce features that promote sustainability, such as a section for second-hand books, book donations, or eco-friendly packaging options.

## 19.CONCLUSION

- In conclusion, building a bookstore using the MERN stack (MongoDB, Express.js, React.js, and Node.js) offers a comprehensive solution for creating a robust, scalable, and user-friendly online platform. By leveraging MongoDB for efficient data management, Express.js for streamlined server-side operations, React.js for a dynamic and responsive user interface, and Node.js for handling asynchronous tasks, you can develop a high-performance application.
- Throughout the development process, it's crucial to address common challenges such as database performance, authentication and security, frontend-backend integration, scalability, user experience, error handling, deployment, and testing. Implementing best practices and using appropriate tools and libraries can help mitigate these issues and enhance the overall functionality of your bookstore.
- Future enhancements, such as advanced search and filtering, personalized recommendations, social features, mobile app development, and enhanced payment options, can further enrich the user experience and expand the capabilities of your application. By continuously iterating and improving your bookstore, you can provide a seamless and enjoyable experience for your users, ensuring the success and growth of your online bookstore.

## 20.REFERENCE

### **React.js Documentation. (2024)**

□ □ React – A JavaScript library for building user interfaces. Retrieved from <https://reactjs.org/>

### **MongoDB Documentation. (2024)**

□ □ MongoDB: The Database for Modern Applications. Retrieved from <https://www.mongodb.com/>

### **Node.js Documentation. (2024)**

□ □ Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Retrieved from <https://nodejs.org/>

### **Libraries and Frameworks:**

#### **Node.js. (2024)**

□ □ Node.js Documentation. Retrieved from <https://nodejs.org/>

#### **Express.js. (2024)**

□ □ Express - Fast, unopinionated, minimalist web framework for Node.js. Retrieved from <https://expressjs.com/>

□ □ JWT.io. (2024). JWT: JSON Web Token. Retrieved from <https://jwt.io/>

#### **Bootstrap. (2024)**

□ □ Bootstrap: The most popular HTML, CSS, and JS library in the world. Retrieved from <https://getbootstrap.com>