Online Complaint Registration and Management System using MERN Stack

Introduction

The Complaint Management and Registration System is a comprehensive full-stack application, leveraging React for an interactive and user-friendly frontend and Node.js with Express for a robust and scalable backend. The frontend empowers users to easily file complaints, view updates, and track the resolution progress in real-time. Meanwhile, the backend handles critical operations such as data processing, user authentication, secure complaint storage, and efficient assignment of complaints to agents. By integrating these technologies, the system offers a seamless and reliable platform for users to address and resolve their issues effectively.

TEAM MEMBERS

E.VASANTH	513421104047
R.SRIRAM	513421104043
R.DINISH KANNA	513421104304
K.REVANTH KUMAR	513421104031

Purpose and Goals

The Complaint Registration and Management System is designed to streamline the process of handling complaints, ensuring efficiency, accountability, and timely resolution. This project aims to bridge the communication gap between users and administrators by offering a digital platform where complaints can be logged, assigned, tracked, and resolved seamlessly. The primary goals are:

Enhanced User Experience

Simplify the complaint registration process for users, enabling them to submit issues quickly and track their resolution status effortlessly.

Optimized Complaint Handling:

Provide administrators with an intuitive dashboard for viewing, assigning, and managing complaints, allowing for better prioritization and resource allocation.

Improved Accountability and Transparency:

Ensure that each complaint is traceable from submission to resolution, giving both users and admins a clear view of complaint statuses and timelines.

Data-Driven Insights:

Offer insightful analytics on complaint patterns and trends, enabling organizations to address recurring issues and improve overall service quality.

Key Features and Functionalities

User-Friendly Complaint Registration:

Intuitive Form-Based Input: Users can quickly log complaints via an accessible form interface that captures essential details, including complaint type, location, and a brief description.

Automatic Acknowledgments: Upon submission, users receive immediate confirmation that their complaint is registered, enhancing transparency from the outset.

Real-Time Complaint Tracking:

Status Updates: Users can monitor the progress of their complaints with clear status indicators like “Pending,” “Assigned,” and “Resolved.”

Notification Alerts: Both users and administrators receive timely notifications as complaints progress through various stages, ensuring no complaint is overlooked.

Admin Dashboard for Comprehensive Management:

Complaint Assignment: Admins can view all pending complaints and assign them to specific agents, along with setting priority levels for timely action.

Role-Based Access Control: The system differentiates between user roles, granting admins and agents customized access to features based on their responsibilities.

Agent-Specific Views and Workflows:

Assigned Complaints View: Agents can see the list of complaints assigned to them, allowing them to manage their workload effectively.

Status Management: Agents can update the complaint status (e.g., to “pending” or
“Assigned” or “Completed”), providing real-time updates to the admin and user dashboards.

Frontend Architecture for Complaint Management and Registration System (React)

The frontend architecture is designed to efficiently handle three user roles—Admin, Ordinary User, and Agent—each with distinct functionalities. The architecture includes modular component structures, role-based routing, state management, and a robust UI framework, making the application scalable and user-friendly.

1. Component Structure

The UI is split into key components, each focusing on the functionality required for the different user roles.

User Components: Allow ordinary users to register complaints, view complaint status, and communicate with agents.

RegisterComplaint: Form to submit a new complaint.

ComplaintStatus: Displays status updates for submitted complaints.

Admin Components: Enable admins to oversee all complaints, assign them to agents, and manage the workflow.

AdminDashboard: Displays a list of all complaints with filtering options (e.g., pending, assigned, completed).

AssignComplaint: Allows the admin to assign complaints to available agents and track their status.

Agent Components: Allow agents to view assigned complaints, communicate with users, and update the complaint status.

AgentDashboard: Lists complaints assigned to the agent, providing an option to update complaint status and message users.

ComplaintDetail: Detailed view for each complaint, including messaging features for communication with the user.

2. Routing and Navigation

Role-Based Routes: Using React Router, different routes are accessible depending on the user's role (e.g., Admin, User, or Agent).

Protected Routes: Only authenticated users can access protected routes.

Role-Specific Navigation: Routes like `/admin-dashboard` (Admin only), `/agent-dashboard` (Agent only), and `/user-complaints` (User only) ensure each user sees only relevant components.

3. Role-Based Access Control

Conditional Rendering: Each role has access to only relevant features:

Admin: Full control over complaints, including viewing, filtering, and assigning to agents.

Agent: Access only to assigned complaints, with options to update statuses and communicate with the user.

Ordinary User: Limited to complaint registration and viewing complaint status.

UI-Based Restrictions: Specific buttons and components appear based on the user's role. For example, the "Assign" button is visible only for Admin users, while the "Update Status" button is available for Agents.

4. UI Design

Responsive Layout: Ensures a seamless experience across devices.

Component-Specific Styling: Each role's dashboard (Admin, Agent, User) is customized to provide intuitive navigation and reduce clutter.

Dynamic Status Indicators: Visual indicators (colors, icons) help users understand complaint statuses at a glance.

Messaging Feature: Enables Agents to communicate directly with Users within the `ComplaintDetail` view.

5. Form Handling and Validation

Form for Complaint Submission: Simplifies data handling in forms, ensuring smooth registration for users.

Error and Success Feedback: Alerts and notifications enhance user experience, providing real-time feedback on complaint submission, status updates, and assignment changes.

This architecture facilitates a streamlined experience for all three user roles, with optimized navigation, clear role-based access, and real-time complaint management, making it a robust solution for complaint handling and resolution.

Backend Architecture for Complaint Management and Registration System (Node.js and Express.js)

The backend architecture supports the primary user roles—Admin, Ordinary User, and Agent—focusing on modularity, scalability, and role-based permissions. The architecture is implemented using **Node.js** and **Express.js**, with **MongoDB** for the database, **Mongoose** for ORM, and **JWT** for secure authentication.

Key Components

1. Models

The models define the structure of the data stored in MongoDB and include:

- **userModel.js**:
    - Manages user information (name, email, password, userType, etc.).
    - Differentiates roles (Admin, Agent, User) using the userType field.
- **complaintModel.js**:
    - Represents complaints submitted by users.
    - Includes fields like userId, address, city, state, status (e.g., pending, assigned, completed).
- **assignedModel.js**:
    - Links complaints to agents for task assignments.
    - Tracks assignment statuses (e.g., assigned, in-progress, completed).
- **messageModel.js**:
    - Facilitates communication between agents and users.
    - Tracks sender, receiver, content, and timestamps.

2. Controllers

Controllers handle the logic for various actions within the system:

- **authController.js**:
    - Manages user registration and login.

    - Generates and validates JWT tokens.
- **complaintController.js**:
    - Handles complaint submission, retrieval, and updates.
    - Allows users to view the status of their complaints.
- **assignedController.js**:
    - Handles assigning complaints to agents.
    - Updates assignment statuses and ensures only admins perform this action.
- **messageController.js**:
    - Manages message exchanges between agents and users.
    - Enables efficient complaint-specific communication.
- **adminController.js**:
    - Handles admin-specific actions like viewing all complaints and assigning them to agents.

3. Routes

Routes expose endpoints for the client to interact with the backend:

- **authRoutes.js**:
    - Endpoints for user login (/login) and registration (/register).
- **complaintRoutes.js**:
    - Endpoints for users to file complaints and check their statuses.
- **assignedRoutes.js**:
    - Endpoints for assigning complaints to agents (/assign) and updating statuses.
- **messageRoutes.js**:
    - Endpoints for sending and retrieving messages related to complaints.
- **adminRoutes.js**:
    - Endpoints for admin operations like managing assignments and viewing all complaints.

4. Middleware

Middleware ensures security and proper handling of requests:

- **authenticateToken.js**:
    - Verifies JWT tokens for secure access.
    - Restricts endpoints based on user roles.

5. Utilities

Utility functions provide reusable logic:

- **generateToken.js**:
    - Generates JWT tokens for authenticated sessions.

6. Entry Point

- **server.js**:
    - Initializes the Express app and connects it to MongoDB.
    - Configures middleware and mounts all routes.
    - Starts the server on a specified port, making the application accessible.

Data Flow Overview

1. **User Registration and Authentication**:
    - Users register via the authRoutes and log in to obtain a JWT token.
    - Tokens are used to authenticate subsequent requests.
2. **Complaint Management**:
    - Users submit complaints via complaintRoutes, which are stored in the database.
    - Admins and agents interact with the complaints through respective routes to update statuses.
3. **Complaint Assignment**:
    - Admins assign complaints to agents via assignedRoutes.
    - Agents update their progress, which is reflected in the system.
4. **Messaging System**:
    - Agents and users exchange messages using messageRoutes.

- Messages are stored in the database and retrieved on demand.

This modular architecture ensures separation of concerns, ease of maintenance, and scalability for future enhancements.

Database Schemas

1.1 User Schema (userSchema)

The userSchema defines the structure of the user data. This schema differentiates users by type (admin, agent, and ordinary user) and stores essential profile details like name, email, and phone.

```
const userSchema = new mongoose.Schema(
{
    name: { type: String, required: true },
    email: { type: String, required: true },
    password: { type: String, required: true },
    phone: { type: Number, required: true },
    userType: { type: String, required: true } // Values: 'admin', 'agent', 'user'
    },
{ timestamps: true }
);
```

- Fields:
    - name, email, password, phone: Store basic user details.
    - userType: Specifies the user role (admin, agent, or user) for role-based access control.
- Interactions:

- During registration, the user's data is stored in MongoDB.
- The userType is essential for controlling access to certain parts of the system, like admin privileges for complaint assignment.
- The timestamps option automatically adds createdAt and updatedAt fields.

1.2 Complaint Schema (complaintSchema)

The complaintSchema represents individual complaints filed by users. Each complaint is linked to the user who created it and contains details like location and complaint status.

```
const complaintSchema = new mongoose.Schema(
    {
    userId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "user" },
    name: { type: String, required: true },
    address: { type: String, required: true },
    city: { type: String, required: true },
    state: { type: String, required: true },
    phone: { type: Number, required: true },
    comment: { type: String, required: true },
status: { type: String, required: true, default: "pending" } // Values: 'pending', 'assigned', 'completed'
    }
);
```

- Fields:
    - userId: Links to the user who submitted the complaint.
    - status: Tracks the complaint's state. Initially set to "pending" and updated as it progresses.
    - comment: User’s description of the complaint.
    - location fields (address, city, state, pincode): Specifies where the complaint is located.
- Interactions:
    - Complaints are created by users and saved to MongoDB.
    - Status is updated by an admin when assigned to an agent and by an agent when resolved.

1.3 Assigned Complaint Schema (assignedSchema)

The assignedSchema represents the assignment of complaints to agents by the admin. It includes both the complaint and agent details, along with the assignment status.

```
const assignedSchema = new mongoose.Schema(
{
    agentId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "user" },
    complaintId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "complaint" },
    status: { type: String, required: true }, // e.g., 'assigned', 'in-progress', 'completed'
    agentName: { type: String, required: true }
}
```

);

- Fields:
    - agentId: Refers to the assigned agent.
    - complaintId: Refers to the associated complaint.
    - status: Tracks the progress of the assignment.
    - agentName: Stores the agent’s name for easy reference.
- Interactions:
    - Admins create an assignment, linking an agent to a complaint.
    - Agents update the status based on their progress (e.g., in-progress, completed).

1.4 Message Schema (messageSchema)

The messageSchema facilitates communication between agents and users about a specific complaint.

```
const messageSchema = new mongoose.Schema(
{
    senderId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "user" },
    receiverId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "user" },
    complaintId: { type: mongoose.Schema.Types.ObjectId, required: true, ref: "complaint" },
    content: { type: String, required: true },
    timestamp: { type: Date, default: Date.now }
    }
```

);

- Fields:
    - senderId, receiverId: Identify the message sender and receiver.
    - complaintId: Links to the specific complaint the message pertains to.
    - content: Stores the message text.
    - timestamp: Automatically records when each message is sent.
- Interactions:
    - Agents and users exchange messages via this schema.
    - Each message is associated with a complaint for easy tracking and contextual discussion.

2. Schema Interactions with MongoDB

Each schema interacts with MongoDB collections to store and retrieve data as per application needs:

- User Registration and Authentication:
    - User data is saved to MongoDB during registration.
    - The userType field differentiates between roles and is used for access control.
- Complaint Management:
    - Complaints are created by users and stored in MongoDB with an initial pending status.
    - Admins query complaints to manage them and assign agents.
- Complaint Assignment:
- Admins create entries in the assigned collection by linking a complaintId to an agentId.
    - The assigned schema is updated by the agent as they work on the complaint.

- Messaging:
    - Messages between agents and users about complaints are stored in the messages collection.
    - The complaintId field enables easy lookup of all messages related to a specific complaint.

3. Relationships

- User - Complaint Relationship: Each complaint is tied to a user (userId).
- Complaint - Assigned Complaint Relationship: Each assignment links a complaint to an agent.
- User - Message Relationship: Messages reference users as senders and receivers, establishing communication about specific complaints.

This architecture keeps data interactions efficient, enables easy tracking of complaint status, and facilitates role-based access. Each schema is purpose-driven, supporting a specific part of the complaint management process.

Setup Instructions for Complaint Management and Registration System

Prerequisites

Before setting up the project, ensure the following software and tools are installed on your system:

1. **Node.js** (v16.x or later)
2. **npm** (Comes with Node.js) or **yarn** - For managing dependencies.
3. **MongoDB** (v4.x or later)
    - Alternatively, use a cloud MongoDB service like **MongoDB Atlas**.
4. **Git** - For cloning the repository.

Installation

1. **Clone the Repository**
   Open your terminal and run the following command:

```
git clone repo_url
cd complaint-management-system
```

2. **Install Dependencies**
   Install the required Node.js packages by running:

   ```
   npm install
   ```

3. **Set Up Environment Variables**
   Create a .env file in the project root directory and add the following configuration:

   ```
   env
   PORT=5000
   MONGO_URI=mongodb://localhost:27017/complaint-management
   JWT_SECRET=your_secret_key
   ```

   - Replace MONGO_URI with your MongoDB connection string.
   - Replace JWT_SECRET with a strong secret key for JWT token signing.

4. **Start the MongoDB Server**
   If you're running MongoDB locally, start it using the following command:

   ```
   mongod
   ```

5. **Start the Development Server**
   Run the following command to start the Node.js server:

   ```
   npm start
   ```

   Or for development mode with hot-reloading:

   ```
   npm run dev
   ```

6. **Access the Application**
   Open your browser and navigate to: http://localhost:5000

Folder Structure

Client: React Frontend Structure

The React frontend is organized for scalability and maintainability, with a clear separation of components, pages, and utilities

```
frontend/
├── public/
│   ├── index.html              # Main HTML file
│   └── logo.svg                # Logo and other static assets
├── src/
│   ├── images/                 # Images or other media files
│   ├── pages/                  # Main pages folder
│   │   ├── admin/              # Admin-specific pages
│   │   │   ├── Admin.jsx       # Admin component
│   │   │   ├── AdminHome.jsx   # Admin dashboard/home page
│   │   │   ├── Agent.jsx       # Component for managing agents
│   │   │   └── User.jsx        # Component for managing users
│   │   ├── agent/              # Agent-specific pages
│   │   │   └── AgentHome.jsx   # Agent dashboard/home page
│   │   ├── common/             # Shared components across roles
│   │   │   ├── About.jsx       # About page
│   │   │   ├── ChatWindow.jsx  # Chat interface component
│   │   │   ├── Home.jsx        # Home page component
│   │   │   ├── Login.jsx       # Login form
│   │   │   ├── Profile.jsx     # Profile page component
│   │   │   └── Signup.jsx      # Signup form
│   │   └── user/               # User-specific pages
│   │       ├── Complaint.jsx   # Component for viewing complaints
│   │       ├── Status.jsx      # Component for checking complaint status
│   │       └── UserHome.jsx    # User dashboard/home page
│   ├── App.js                  # Main App component
│   ├── App.test.js             # Test file for App component
│   ├── index.js                # Entry point for React
│   └── index.css               # Global styles
├── node_modules/               # Node dependencies
└── package.json                # Project dependencies and scripts
```

Server: Node.js Backend Structure

The Node.js backend follows a modular structure for scalability, with dedicated directories for models, controllers, routes, and middleware.

```
backend/
├── models/                    # Database schemas
│   ├── userModel.js           # User schema
│   ├── complaintModel.js      # Complaint schema
│   ├── assignedModel.js       # Assignment schema
│   └── messageModel.js        # Message schema
│
├── controllers/               # Request handlers
│   ├── adminController.js     # Admin-specific operations
│   ├── assignedController.js  # Assignment-related operations
│   ├── authController.js      # Authentication operations
│   ├── complaintController.js # Complaint-related logic
│   └── messageController.js   # Messaging logic
│
├── routes/                    # API endpoints
│   ├── adminRoutes.js         # Admin-specific routes
│   ├── assignedRoutes.js      # Assignment routes
│   ├── authRoutes.js          # Authentication routes
│   ├── complaintRoutes.js     # Complaint-related routes
│   └── messageRoutes.js       # Messaging routes
│
├── middleware/                # Middleware functions
│   ├── authenticateToken.js   # JWT authentication
│   └── errorHandler.js        # Global error handling
│
├── utilities/                 # Utility functions
│   └── generateToken.js       # Helper to generate JWT tokens
│
├── config/                    # Configuration files
│   └── db.js                  # MongoDB connection setup
│
├── .env                       # Environment variables
├── server.js                  # Main server file
├── package.json               # Dependencies and scripts
└── README.md                  # Documentation
```

Running the Application

1. **Frontend**
    - Navigate to the frontend directory:

      cd frontend

    - Start the frontend server:

      npm start

    - Open a browser and go to http://localhost:3000 to view the frontend.
2. **Backend**
    - Navigate to the backend directory:

      cd backend

    - Start the backend server:

      npm start

    - The backend server will run on http://localhost:5000 (or as specified in your environment settings).

API Documentation

User API

GET /api/users/agents

Description: Get a list of all agents.
Response:


```
{
 "agent": [
  {
   "_id": "ObjectId",
   "name": "Agent Name",
   "email": "agent@example.com",
   "phone": 1234567890,
   "userType": "Agent",
```

```
    "createdAt": "2024-11-15T00:00:00Z",
    "updatedAt": "2024-11-15T00:00:00Z"
   }
  ]
}
```


GET /api/users/ordinary

Description: Get a list of all ordinary users.
Response:


```
{
  "user": [
    {
      "_id": "ObjectId",
      "name": "User Name",
      "email": "user@example.com",
      "phone": 9876543210,
      "userType": "Ordinary",
      "createdAt": "2024-11-15T00:00:00Z",
      "updatedAt": "2024-11-15T00:00:00Z"
    }
  ]
}
```


GET /api/users/agent/:agentId

Description: Get details of a single agent.
Response:


```
{
  "_id": "ObjectId",
  "name": "Agent Name",
  "email": "agent@example.com",
  "phone": 1234567890,
  "userType": "Agent",
  "createdAt": "2024-11-15T00:00:00Z",
  "updatedAt": "2024-11-15T00:00:00Z"
}
```

DELETE /api/users/:userId

Description: Delete a user by ID.
Response:

```
{
  "message": "User Deleted Successfully"
}
```

PUT /api/users/:id

Description: Update user information.
Response:

```
{
  "updatedUser": {
    "_id": "ObjectId",
    "name": "Updated Name",
    "email": "updated@example.com",
    "phone": 1234567890,
    "userType": "Ordinary",
    "createdAt": "2024-11-15T00:00:00Z",
    "updatedAt": "2024-11-15T00:00:00Z"
  },
  "message": "User updated Successfully"
}
```

Authentication API

POST /api/auth/signup

Description: Register a new user.
Response:

```
{
  "_id": "ObjectId",
  "name": "User Name",
  "email": "user@example.com",
  "phone": 9876543210,
  "userType": "Ordinary"
}
```

POST /api/auth/login

Description: Login an existing user.
Response:


```
{
  "_id": "ObjectId",
  "name": "User Name",
  "email": "user@example.com",
  "phone": 9876543210,
  "userType": "Ordinary"
}
```


Complaint API

POST /api/comp/add-complaint/:userId

Description: Create a new complaint for a user.
Response:


```
{
  "_id": "ObjectId",
  "userId": "ObjectId",
  "name": "Complaint Title",
  "address": "123 Main St",
  "city": "City Name",
  "state": "State Name",
  "pincode": 123456,
  "comment": "Complaint description",
  "status": "pending",
  "createdAt": "2024-11-15T00:00:00Z",
  "updatedAt": "2024-11-15T00:00:00Z"
}
```

GET /api/comp/all-complaints

Description: Get all complaints.
Response:


```
{
 "complaints": [
   {
    "_id": "ObjectId",
    "userId": "ObjectId",
    "name": "Complaint Title",
    "address": "123 Main St",
    "city": "City Name",
    "state": "State Name",
    "pincode": 123456,
    "comment": "Complaint description",
    "status": "pending",
    "createdAt": "2024-11-15T00:00:00Z",
    "updatedAt": "2024-11-15T00:00:00Z"
   }
 ]
}
```


GET /api/comp/user-complaints/:userId

Description: Get all complaints for a specific user.
Response:


```
{
 "complaints": [
   {
    "_id": "ObjectId",
    "userId": "ObjectId",
    "name": "Complaint Title",
    "address": "123 Main St",
    "city": "City Name",
    "state": "State Name",
    "pincode": 123456,
    "comment": "Complaint description",
    "status": "pending",
```

```
    "createdAt": "2024-11-15T00:00:00Z",
    "updatedAt": "2024-11-15T00:00:00Z"
   }
  ]
}
```


PUT /api/comp/update-complaint/:complaintId

Description: Update the status of a complaint.
Response:


```
{
 "updatedComplaint": {
  "_id": "ObjectId",
  "userId": "ObjectId",
  "name": "Complaint Title",
  "address": "123 Main St",
  "city": "City Name",
  "state": "State Name",
  "pincode": 123456,
  "comment": "Complaint description",
  "status": "resolved",
  "createdAt": "2024-11-15T00:00:00Z",
  "updatedAt": "2024-11-15T00:00:00Z"
 },
 "message": "Complaint updated successfully"
}
```


Assigned Complaint API

POST /api/comp/assign-complaint

Description: Assign a complaint to an agent.
Response:


```
{
 "message": "Complaint assigned successfully"
}
```

GET /api/comp/agent-complaints/:agentId

Description: Get all complaints assigned to a specific agent.
Response:


```
{
  "updatedComplaints": [
    {
      "_id": "ObjectId",
      "agentId": "ObjectId",
      "complaintId": "ObjectId",
      "status": "assigned",
      "agentName": "Agent Name",
      "name": "Complaint Title",
      "address": "123 Main St",
      "city": "City Name",
      "state": "State Name",
      "pincode": 123456,
      "comment": "Complaint description"
    }
  ],
  "message": "Complaints fetched successfully"
}
```


Message API

POST /api/mesg/add-message

Description: Add a message to a complaint.
Response:


```
{
  "_id": "ObjectId",
  "name": "Agent Name",
  "message": "Complaint message text",
  "complaintId": "ObjectId",
  "createdAt": "2024-11-15T00:00:00Z",
  "updatedAt": "2024-11-15T00:00:00Z"
}
```

GET /api/mesg/messages/:complaintId

Description: Get all messages for a specific complaint.
Response:


```
{
  "messages": [
    {
      "_id": "ObjectId",
      "name": "Agent Name",
      "message": "Complaint message text",
      "complaintId": "ObjectId",
      "createdAt": "2024-11-15T00:00:00Z",
      "updatedAt": "2024-11-15T00:00:00Z"
    }
  ]
}
```


Authentication

Authentication is the process of verifying the identity of a user, device, or system. It ensures that the entity attempting to access a resource is genuinely who it claims to be.

Methods:

- **Username and Password**: The most common method, where a user provides a username and password to verify their identity.
- **Token-based Authentication (JWT)**: After the user logs in, a JSON Web Token (JWT) is issued. This token is sent with each subsequent request to authenticate the user.
- **OAuth**: An authorization framework that allows third-party applications to access user data without exposing their credentials.

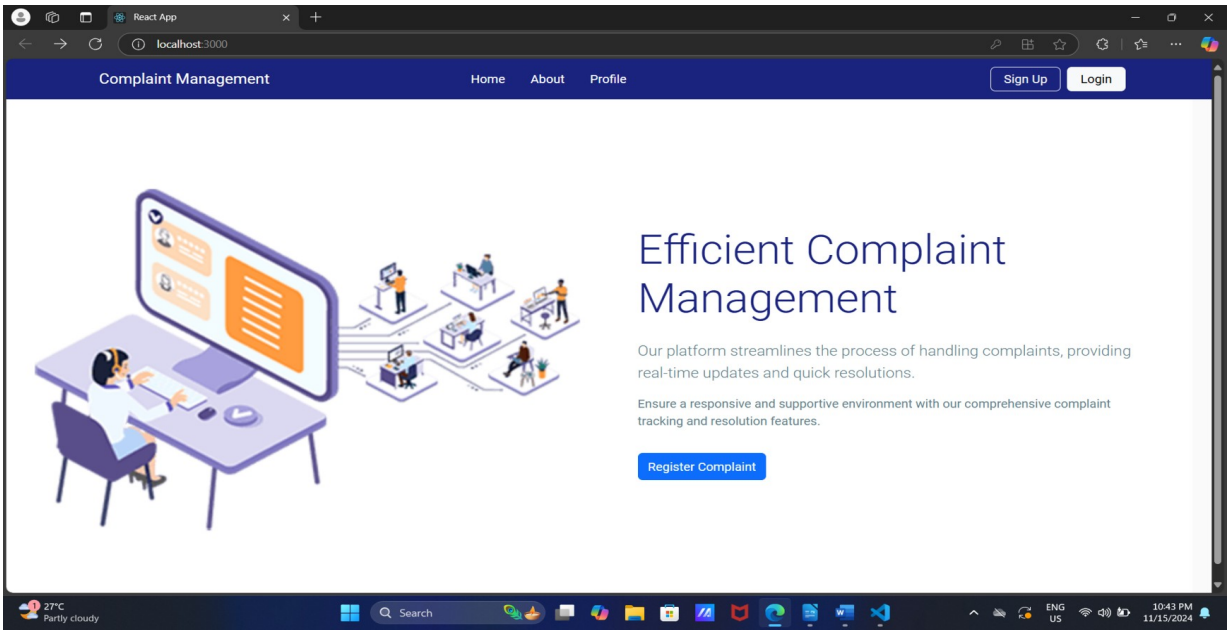
Authorization

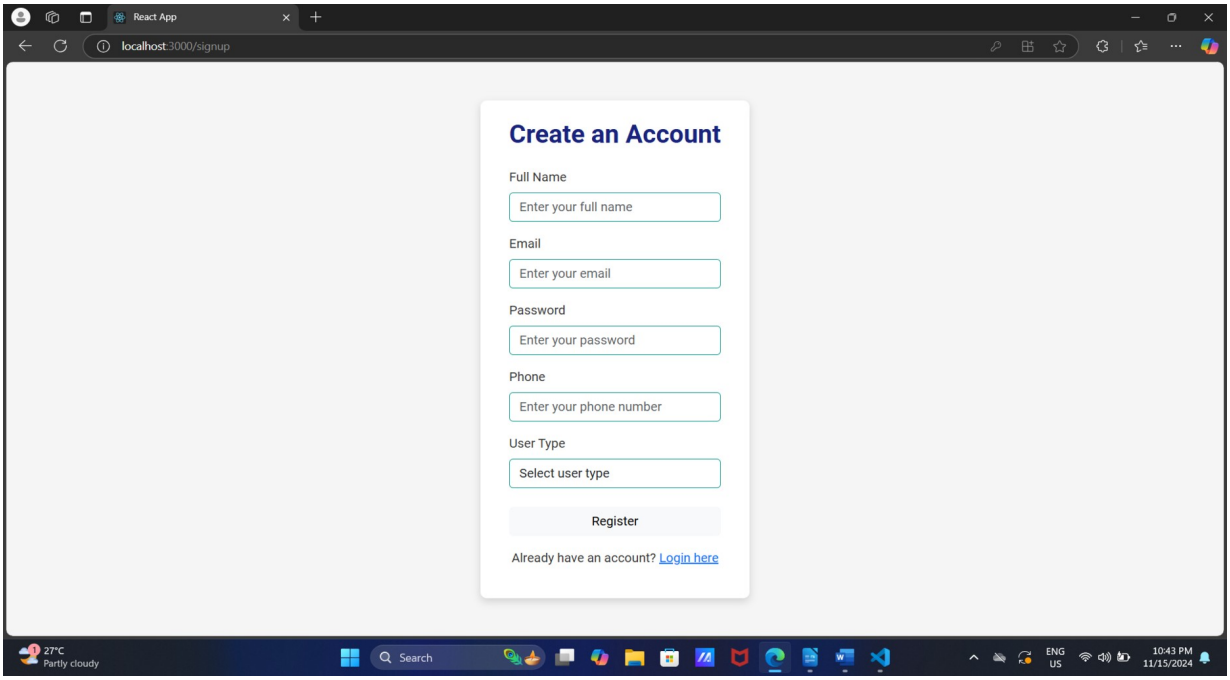
Authorization is the process of granting or denying access to specific resources or actions based on the authenticated user's permissions or roles. After a user is authenticated, the system decides whether they are permitted to perform an action or access certain data.

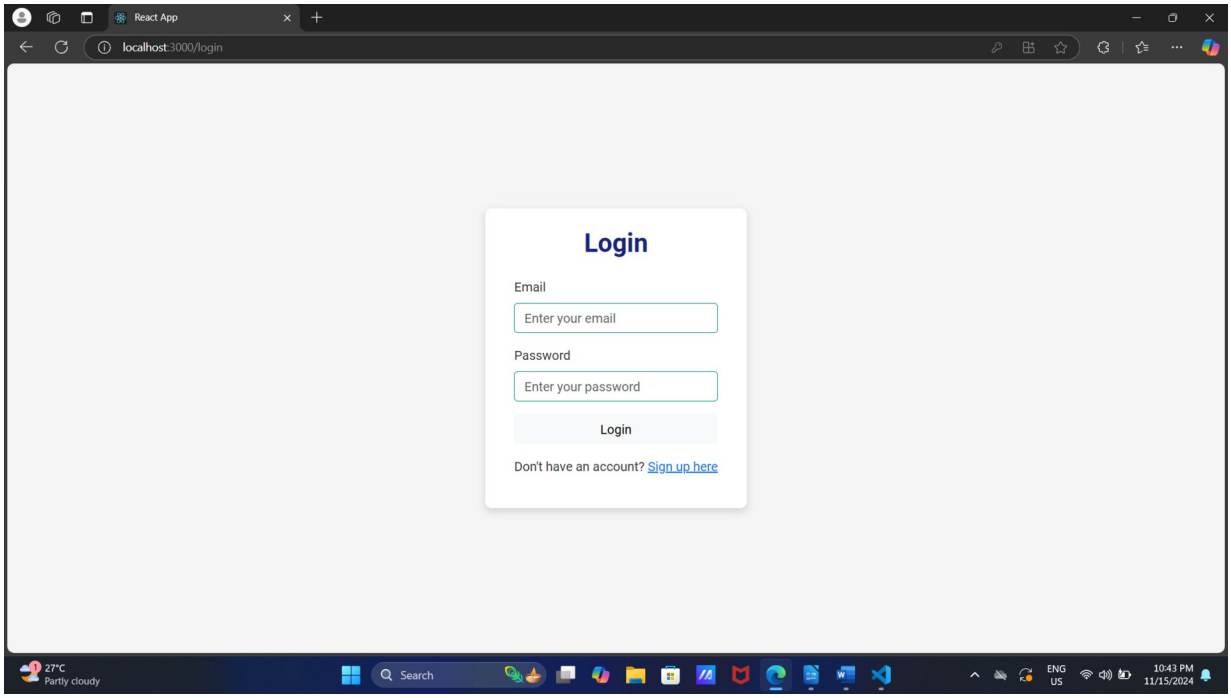
Methods:

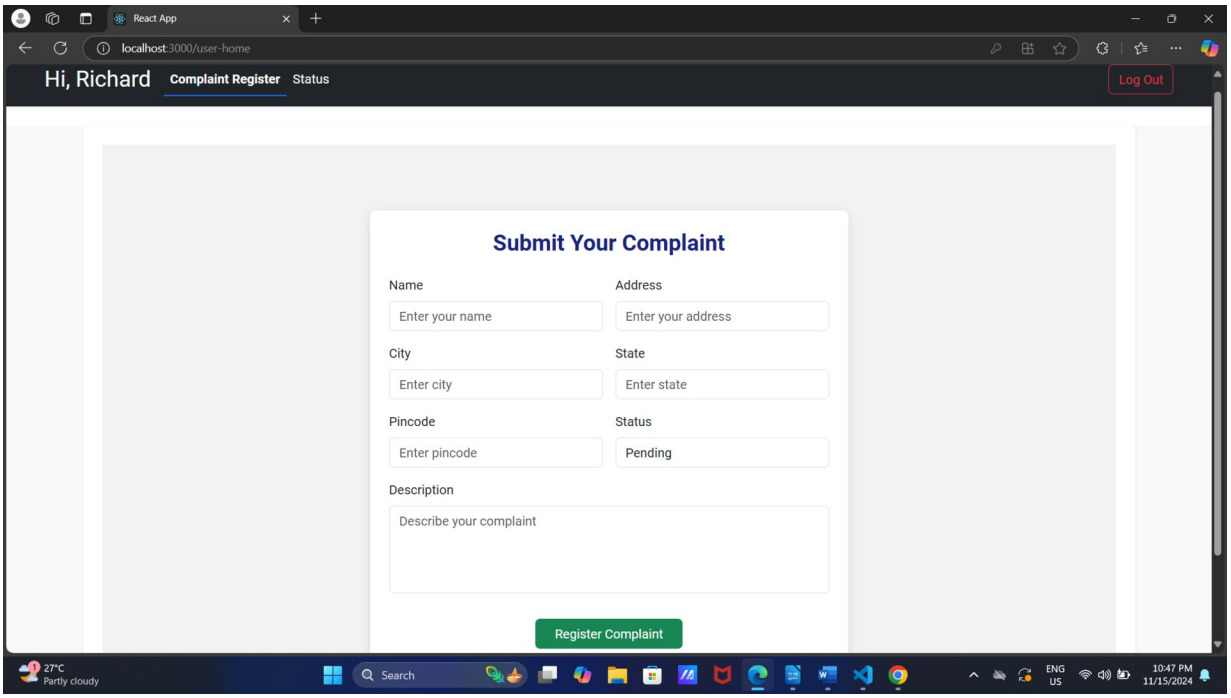
- **Role-based Access Control (RBAC)**: Users are assigned roles (e.g., admin, user) that define their level of access to resources.
- **Permission-based Access Control**: Permissions are explicitly assigned to users or groups, defining what actions they can perform (e.g., read, write, delete).

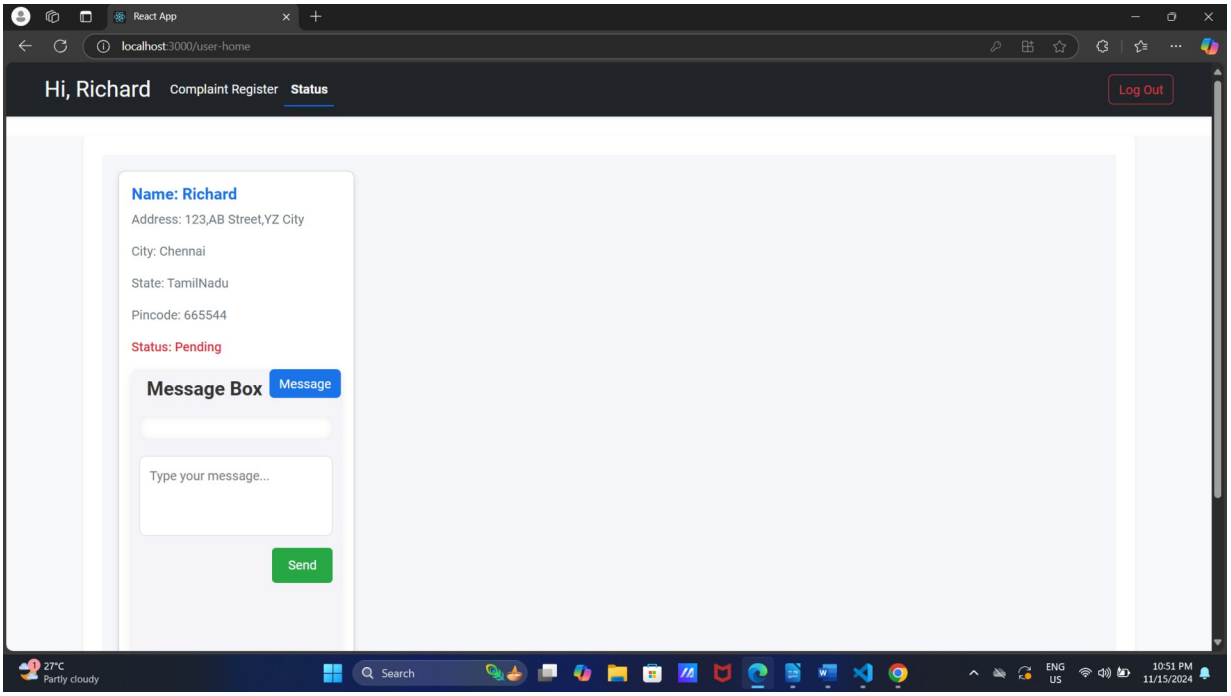
User Interface

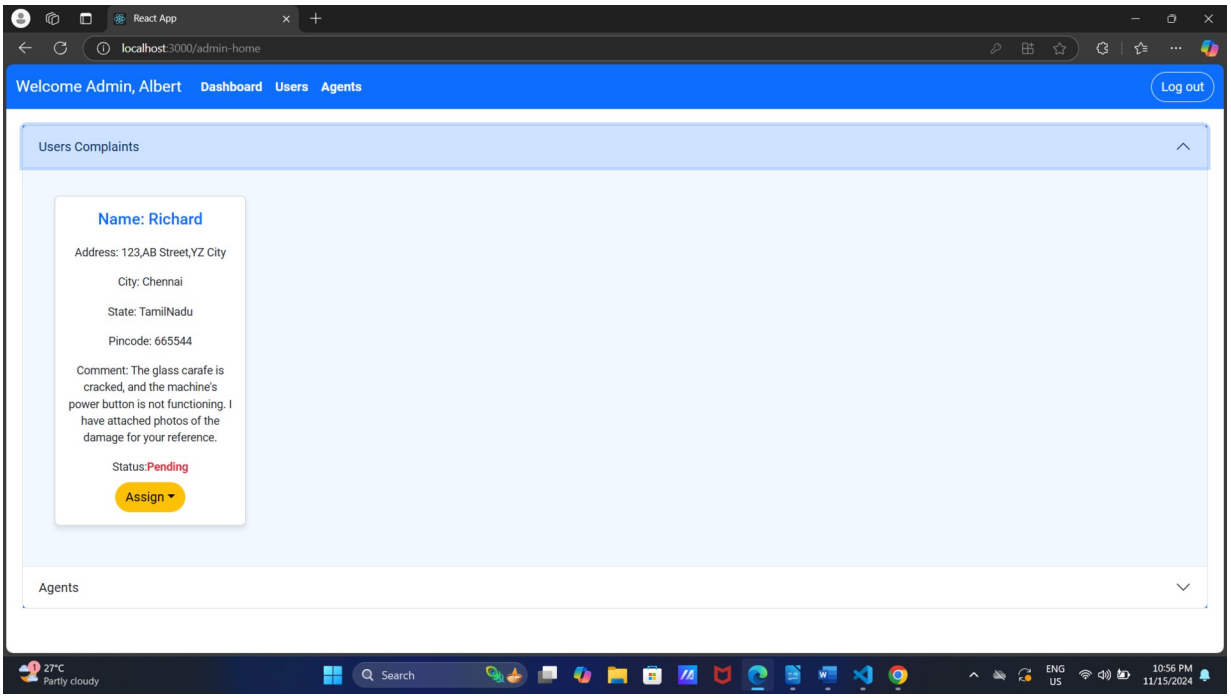


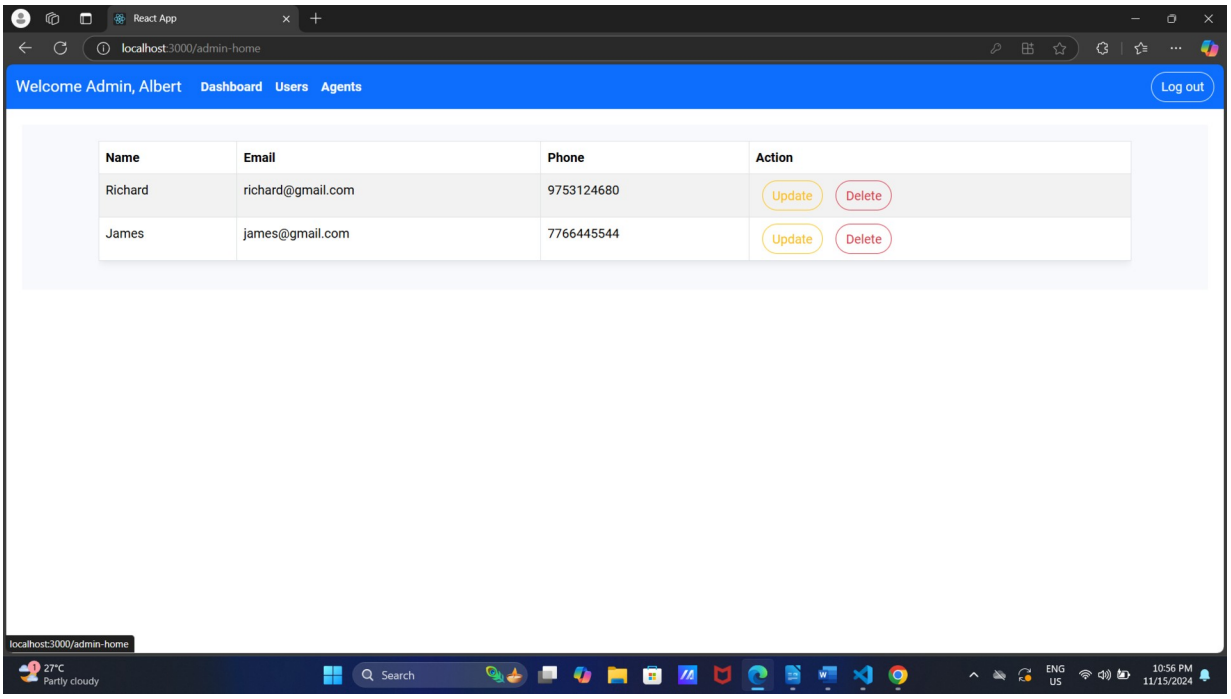

React App
localhost:3000/signup
Create an Account
Full Name
Enter your full name
Email
Enter your email
Password
Enter your password
Phone
Enter your phone number
User Type
Select user type
Register
Already have an account? Login here
27°C
Partly cloudy
Search
ENG
US
10:43 PM
11/15/2024

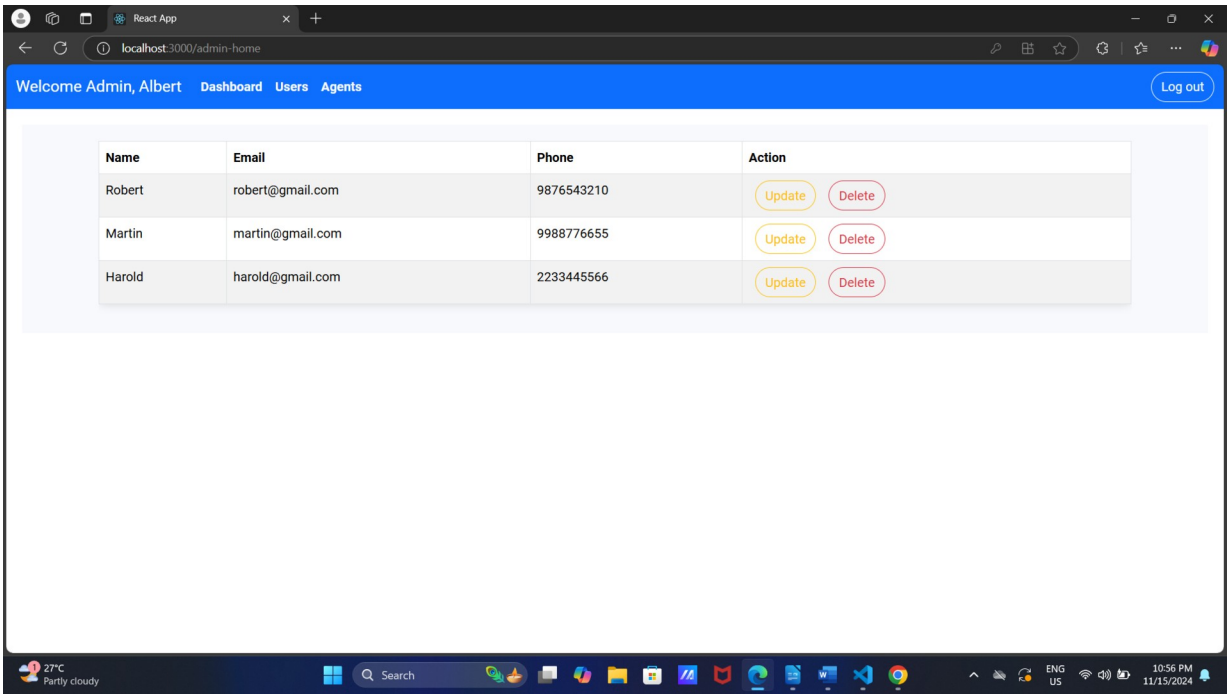

React App
localhost:3000/login
Login
Email
Enter your email
Password
Enter your password
Login
Don't have an account? Sign up here
27°C
Partly cloudy
Search
ENG
US
10:43 PM
11/15/2024

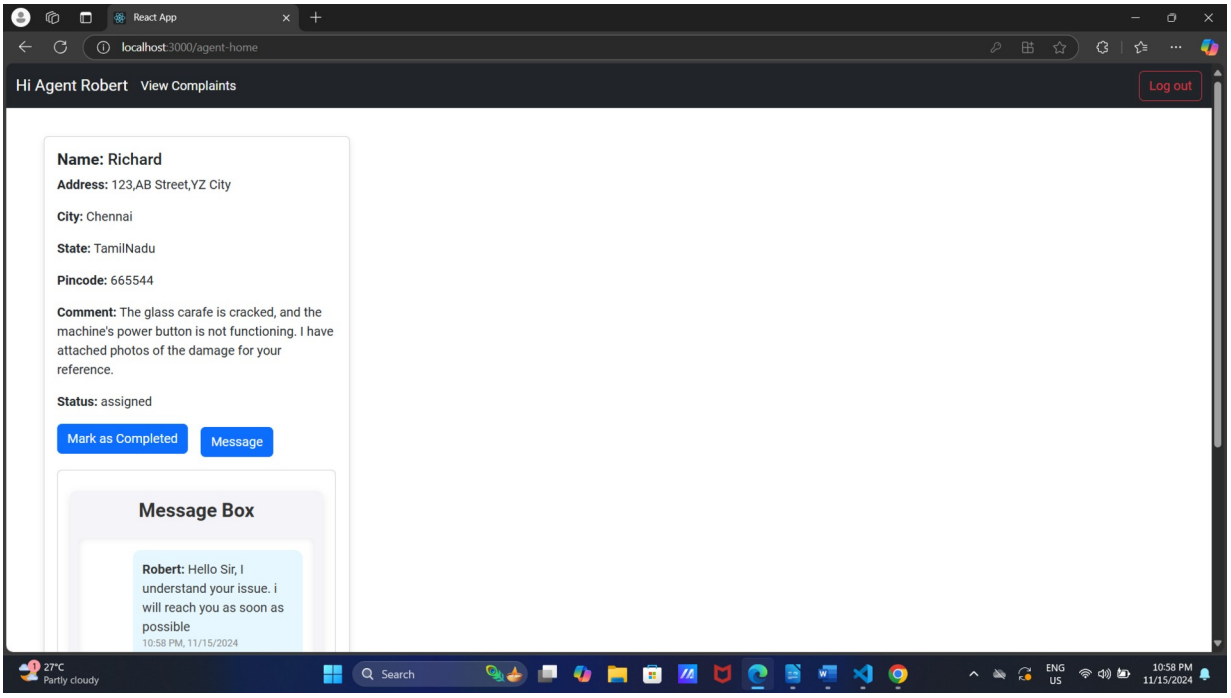

React App
localhost:3000/user-home
Hi, Richard
Complaint Register
Status
Log Out
Submit Your Complaint
Name
Enter your name
Address
Enter your address
City
Enter city
State
Enter state
Pincode
Enter pincode
Status
Pending
Description
Describe your complaint
Register Complaint
27°C
Partly cloudy
Search
ENG
US
10:47 PM
11/15/2024

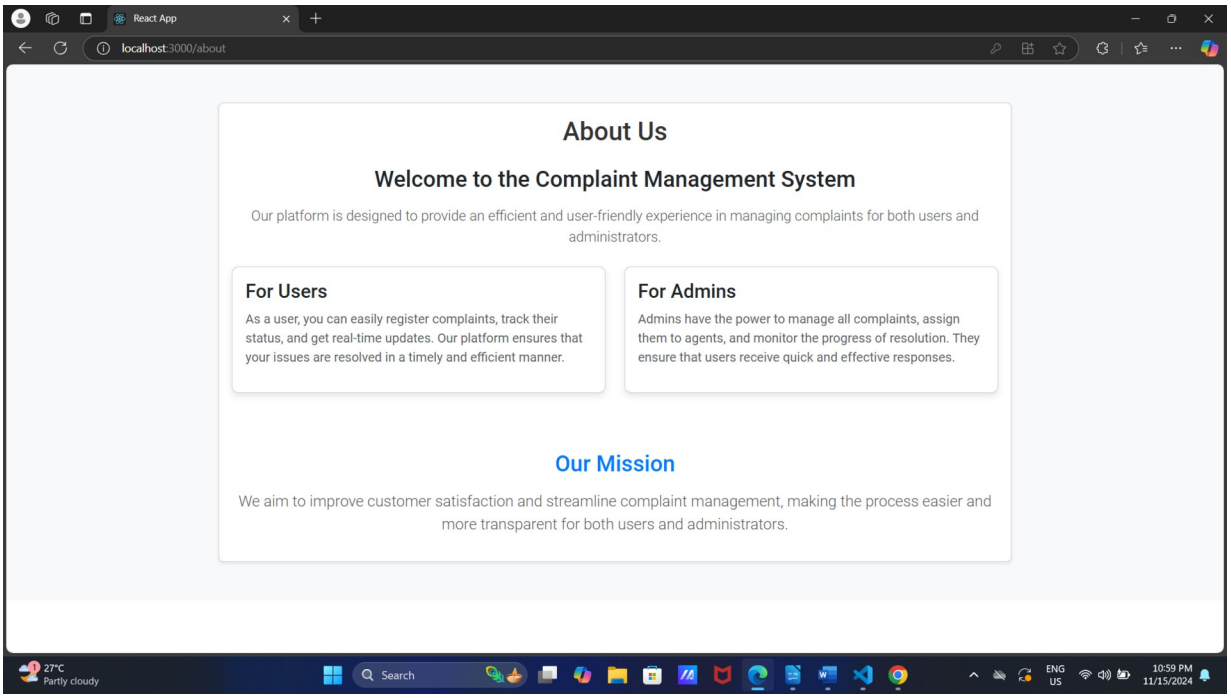

React App
localhost:3000/user-home
Hi, Richard
Complaint Register
Status
Log Out
Name: Richard
Address: 123,AB Street,YZ City
City: Chennai
State: TamilNadu
Pincode: 665544
Status: Pending
Message Box
Message
Type your message...
Send
27°C
Partly cloudy
Search
ENG
US
10:51 PM
11/15/2024

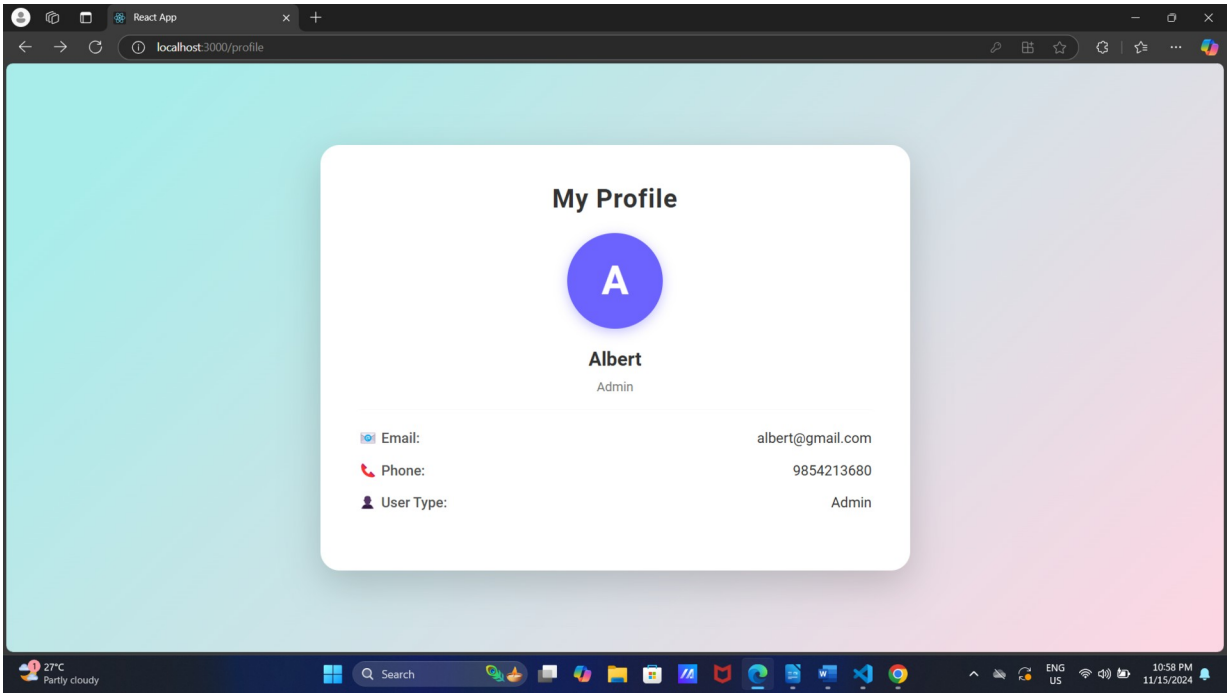

React App
localhost:3000/admin-home
Welcome Admin, Albert
Dashboard
Users
Agents
Log out
Users Complaints
Name: Richard
Address: 123,AB Street,YZ City
City: Chennai
State: TamilNadu
Pincode: 665544
Comment: The glass carafe is cracked, and the machine's power button is not functioning. I have attached photos of the damage for your reference.
Status:Pending
Assign
Agents
27°C
Partly cloudy
Search
ENG
US
10:56 PM
11/15/2024


React App
localhost:3000/admin-home
Welcome Admin, Albert
Dashboard
Users
Agents
Log out
Name
Email
Phone
Action
Richard
richard@gmail.com
9753124680
Update
Delete
James
james@gmail.com
7766445544
Update
Delete
localhost:3000/admin-home
27°C
Partly cloudy
Search
ENG
US
10:56 PM
11/15/2024


React App
localhost:3000/admin-home
Welcome Admin, Albert
Dashboard
Users
Agents
Log out
Name
Email
Phone
Action
Robert
robert@gmail.com
9876543210
Update
Delete
Martin
martin@gmail.com
9988776655
Update
Delete
Harold
harold@gmail.com
2233445566
Update
Delete
27°C
Partly cloudy
Search
ENG
US
10:56 PM
11/15/2024


React App
localhost:3000/agent-home
Hi Agent Robert
View Complaints
Log out
Name: Richard
Address: 123,AB Street,YZ City
City: Chennai
State: TamilNadu
Pincode: 665544
Comment: The glass carafe is cracked, and the machine's power button is not functioning. I have attached photos of the damage for your reference.
Status: assigned
Mark as Completed
Message
Message Box
Robert: Hello Sir, I understand your issue. i will reach you as soon as possible
10:58 PM, 11/15/2024
27°C
Partly cloudy
Search
ENG
US
10:58 PM
11/15/2024


React App
localhost:3000/about
About Us
Welcome to the Complaint Management System
Our platform is designed to provide an efficient and user-friendly experience in managing complaints for both users and administrators.
For Users
As a user, you can easily register complaints, track their status, and get real-time updates. Our platform ensures that your issues are resolved in a timely and efficient manner.
For Admins
Admins have the power to manage all complaints, assign them to agents, and monitor the progress of resolution. They ensure that users receive quick and effective responses.
Our Mission
We aim to improve customer satisfaction and streamline complaint management, making the process easier and more transparent for both users and administrators.
27°C
Partly cloudy
Search
ENG
US
10:59 PM
11/15/2024


React App
localhost:3000/profile
My Profile
A
Albert
Admin
Email:
albert@gmail.com
Phone:
9854213680
User Type:
Admin
27°C
Partly cloudy
Search
ENG
US
10:58 PM
11/15/2024

Testing Strategy and Tools Used

Testing is an essential part of ensuring the functionality, performance, and reliability of the application. In this project, the following testing approaches and tools are used:

1. **Manual Testing**:
    - **Postman** is used for testing API endpoints manually. This ensures that APIs are working correctly, handling various input cases, and returning the expected responses.
2. **Unit Testing**:
    - **Jest** is employed for unit testing individual functions and components to validate their correctness.
    - **Mocha/Chai** are used for testing backend logic, such as validating API responses and database interactions.
3. **Integration Testing**:
    - Integration tests ensure that different components of the application, like the API and the database, work correctly together. These tests check if the complete flow (e.g., creating a complaint or assigning it) functions as expected.

Demo

Video Link

https://drive.google.com/file/d/1umlmbdQIWj_AliezHxdqvBDcGlSBdhNB/view?usp=sharing

Known Issues

There are a few issues within the application that may affect user experience. These include problems with data persistence, such as certain changes not being reflected after page refreshes. Additionally, there are occasional bugs related to UI elements not updating correctly, such as buttons reappearing after actions are taken. Login and signup functionalities may also experience validation errors. Efforts are ongoing to address and resolve these issues in future updates.

Future Enhancements

In the coming phases, several exciting features and improvements will be introduced to elevate the functionality and user experience of the project:

1. **Advanced Search and Filtering:** Enhance the complaint management system by integrating more sophisticated search and filter options, allowing users to quickly find complaints based on status, agent, or time.
2. **Real-Time Notifications:** Implement real-time push notifications to alert users and agents about updates or new assignments, ensuring timely responses and better communication.
3. **User Role Permissions:** Extend user role capabilities by introducing customized permissions, providing a more granular level of control for admins, agents, and regular users.
4. **Analytics Dashboard:** Build an insightful analytics dashboard for administrators to view detailed reports on complaint status, user activity, and agent performance to make data-driven decisions.
5. **AI-Driven Complaint Routing:** Introduce AI algorithms to automatically assign complaints to the most suitable agents based on skills, workload, and availability, improving efficiency.
6. **Mobile App Integration:** Develop a mobile app to complement the web interface, ensuring users and agents can manage complaints and receive updates on-the-go.
7. **Multilingual Support:** Incorporate multilingual support to cater to a wider audience, ensuring accessibility and inclusivity for diverse user groups.

These enhancements will significantly improve system usability, operational efficiency, and user satisfaction, setting the stage for future scalability and success.