User Role-Based Complaint Management System Documentation

Overview:

- **Objective**: Create a web-based Support Ticketing System with role-based access (Admin & Customer).
- **Key Features**:
- Customers can create, view, and delete their tickets.
- Admins can view, reply, and update the status of tickets.
- Tech stack: React.js (Frontend), Node.js (Backend), MySQL (Database).

Core Functionality:

Authentication:

- Single login page for Admin and Customer.
- Handle login with an authentication system.
- Optionally add registration with role assignment.

Role-Based Dashboards:

- **Admin Dashboard**:
- View all tickets.
- Respond to any ticket.
- **Customer Dashboard**:
- View, create, update, and delete personal tickets.
- Tickets are automatically assigned to Admin.

Ticket Management:

- Tickets contain: Subject, Description, Status (Open, Resolved, Closed), Customer, Executive.
- Admins/Executives can reply to tickets.

Installation Steps:

Install Dependencies:

- Frontend (React.js):
```
npx create-react-app ticketing-system cd ticketing-system npm install axios react-router-dom tailwindcss
```

- Backend (Node.js):
```
mkdir backend cd backend npm init -y npm install express mysql2 bcryptjs jsonwebtoken dotenv npm install --save-dev nodemon
```

Set up MySQL:

Install MySQL:

- **Linux**: `sudo apt-get install mysql-server`
- **macOS**: `brew install mysql`
- **Windows**: Download from MySQL website.

- **Create Database**:
  ```
  CREATE DATABASE Complaint_system;
  ```

- **Create Tables**:
  ```
  CREATE TABLE Users (   id INT AUTO_INCREMENT PRIMARY KEY,   username VARCHAR(255) NOT NULL UNIQUE,   password VARCHAR(255) NOT NULL,   role ENUM('admin', 'customer') NOT NULL );  CREATE TABLE Tickets (   id INT AUTO_INCREMENT PRIMARY KEY,   subject VARCHAR(255) NOT NULL,   description TEXT NOT NULL,   status ENUM('open', 'resolved', 'closed') DEFAULT 'open',   user_id INT,   executive_id INT,   FOREIGN KEY(user_id) REFERENCES Users(id) );
  ```

Backend Setup:

Create Express Server (Node.js):

index.js (Backend entry point):

JWT Authentication:

- Add login and token generation functionality using `jsonwebtoken`.

Frontend Setup (React):

Create Pages:

- **Login Page**: Handle login and store JWT in localStorage.
- **Dashboard Pages**:
  Admin: View and manage all tickets.
  Customer: Create and manage own tickets.

Axios Setup:

Example API Call for Fetching Tickets:

```
import axios from 'axios';  const fetchTickets = async () => {
const token = localStorage.getItem('token');

const response = await axios.get('http://localhost:5000/api/tickets', {
```

```
   headers: { Authorization: `Bearer ${token}` },
});
  response.data;
};
```

React Router:

- Use `react-router-dom` for routing between login, dashboards, and ticket views.

  ```
  npm install react-router-dom
  ```

Frontend and Backend Integration:

Setup CORS in backend to allow frontend requests:

```
const cors = require('cors'); app.use(cors());
```

Handle JWT Authentication:

Attach JWT in `Authorization` header for protected routes:

```
axios.get('/api/tickets', {
   headers: {     Authorization: `Bearer ${token}`,   }
});
```

Bonus Points:

1. **Filters/Search Functionality** for tickets (e.g., by status, subject and description).
2. **Registration** functionality with role selection.

Conclusion:

This system ensures a simple yet effective way to manage customer complaints with role-based access control. The Admin can manage all tickets, while customers can only manage their own, with both roles being securely authenticated and authorized through JWT tokens.