# Project: Real-time Admin and User Inquiry System

## Description:

The Real-time Admin and User Inquiry System is a web application that enables users to submit inquiries and receive real-time responses from administrators. The system ensures instant communication through WebSockets, allowing seamless interaction between users and admins. The project includes role-based authentication, an admin dashboard for managing inquiries, and user dashboards for tracking responses.

## Features:

1. **User Authentication & Roles**

- Secure login system with JWT.
- Role-based access control (Admin & User).

2. **Real-time Inquiry System**

- Users can submit inquiries through a form.
- Admins receive live notifications of new inquiries.
- WebSockets ensure real-time updates.
- Inquiry tracking: Pending, In Progress, Resolved.

3. **Admin Dashboard**

- Overview of total, active, and resolved inquiries.
- Real-time inquiry list with timestamps.
- Assign inquiries to specific admins.
- Chat-based inquiry response system.

4. **User Dashboard**

- View submitted inquiries with live status updates.
- Receive real-time admin responses.
- Update or close an inquiry after resolution.

5. **Notifications & Alerts**

- Push/email notifications for new inquiries.
- Real-time user updates when admins respond.
- Optional Twilio/email notifications.

6. **Deployment & Security**

- Hosted on platforms like Vercel (frontend) and Render/AWS (backend).

- Data stored securely in MongoDB or Firebase Firestore.
- Security measures include JWT authentication and role-based access.

## Technologies Used:

#### Frontend

- React.js (Dynamic UI)
- Next.js (Optional for SSR & Performance)
- Tailwind CSS / Bootstrap (Styling)
- Socket.io-client (Real-time communication)

#### Backend

- Node.js (Server-side logic)
- Express.js (Backend framework)
- WebSockets (Socket.io) (Real-time updates)

#### Database

- MongoDB (Primary database)
- Firebase Firestore (Optional for real-time DB)
- Redis (Optional for caching)

#### Authentication & State Management

- JWT (JSON Web Token)
- Firebase Auth (Optional)
- Redux / Context API (State management)

#### Hosting & Deployment

- Frontend: Vercel / Netlify
- Backend: Render / AWS / Heroku
- Database: MongoDB Atlas / Firebase Firestore

#### Version Control & CI/CD

- Git & GitHub
- GitHub Actions (Optional for automated deployment)

## Console Code (Node.js + Express + WebSockets)

1. **Backend (Express & WebSockets)**

```
Const express = require(“express”);
Const http = require(“http”);
Const { Server } = require(“socket.io”);
Const cors = require(“cors”);
Require(“dotenv”).config();
Const app = express();
Const server = http.createServer(app);
Const io = new Server(server, {
    Cors: {
        Origin: “*”,
        Methods: [“GET”, “POST”]
    }
});
App.use(cors());
App.use(express.json());
Let inquiries = [];
// WebSocket connection
Io.on(“connection”, (socket) => {
    Console.log(“A user connected:”, socket.id);
    Socket.on(“newInquiry”, (inquiry) => {
        Inquiries.push(inquiry);
        Io.emit(“updateInquiries”, inquiries);
    });
    Socket.on(“disconnect”, () => {
        Console.log(“User disconnected:”, socket.id);
    });
});
// API to get inquiries
App.get("/inquiries", (req, res) => {
    Res.json(inquiries);
});
Const PORT = process.env.PORT || 5000;
Server.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

2. **Frontend (React + Socket.io)**

```
Import React, { useState, useEffect } from "react";
Import io from "socket.io-client";
Const socket = io(http://localhost:5000);
Const InquirySystem = () => {
    Const [inquiries, setInquiries] = useState([]);
    Const [message, setMessage] = useState("");
    useEffect(() => {
        socket.on("updateInquiries", (data) => {
            setInquiries(data);
        });
    }, []);
 Const submitInquiry = () => {
        Const newInquiry = { message, status: "Pending" };
        Socket.emit("newInquiry", newInquiry);
        setMessage("");
    };
    Return (
        <div>
```

```
            <h2>Real-time Inquiry System</h2>
            <input
                Type="text"
                Value={message}
                onChange={€ => setMessage(e.target.value)}
                placeholder="Enter your inquiry"
            />
            <button onClick={submitInquiry}>Submit</button>
            <h3>Inquiries</h3>
            <ul>
                {inquiries.map((inq, index) => (
                    <li key={index}>{inq.message} – {inq.status}</li>
                ))}
            </ul>
        </div>
    );
};
Export default InquirySystem;
```

## Expected Output

**Console Output**

```
Server running on port 5000
A user connected: BxD12fZ
User disconnected: BxD12fZ
```

**Frontend UI**

- A text input field and submit button for users to send inquiries.
- A list of inquiries appearing in real-time as they are submitted.

- Admins see live updates and can respond.

## Deployment

1. Backend Deployment: Use Render, AWS, or Heroku.
2. Frontend Deployment: Deploy via Vercel or Netlify.
3. Database Setup: Use MongoDB Atlas or Firebase Firestore.
4. GitHub Repository: Push the project to GitHub and enable CI/CD.

## Final Deliverables

- Fully functional Real-time Inquiry System
- Admin & User Dashboards with live updates
- Secure Authentication & Role-based Access
- Deployed Web Application for real-time inquiries