**Backend Code:**

**user.js: (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\backend\models)**

const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({

  name: { type: String, required: true },

  email: { type: String, required: true, unique: true },

  password: { type: String, required: true },

  role: { type: String, enum: ['patient', 'doctor'], required: true },

});

module.exports = mongoose.model('User', userSchema);

**auth.js: (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\backend\routes)**

const express = require('express');

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const User = require('../models/User');

const router = express.Router();

// Predefined Users

const predefinedUsers = [

    { email: 'patient@hospital.com', password: 'patient123', role: 'patient', name: 'John Doe', age: 30, gender: 'male', contact: '1234567890' },

    { email: 'doctor@hospital.com', password: 'doctor123', role: 'doctor', name: 'Dr. Smith', age: 40, gender: 'male', contact: '0987654321' },

    { email: 'fahim1915021@stud.kuet.ac.bd', password: 'fahim1915021', role: 'admin', name: 'Fahim Admin', age: 0, gender: 'N/A', contact: 'N/A' } // New Admin

];

// Seed Database with Predefined Users

predefinedUsers.forEach(async (user) => {

    const existingUser = await User.findOne({ email: user.email });

    if (!existingUser) {

        const hashedPassword = await bcrypt.hash(user.password, 10);

        await User.create({ ...user, password: hashedPassword }); // Ensure all required fields are included

    }

});

// Login Route (Updated)

router.post('/login', async (req, res) => {

  const { email, password } = req.body;

  try {

      const user = await User.findOne({ email });

      if (!user) return res.status(400).json({ message: 'Invalid credentials' });

      const isMatch = await bcrypt.compare(password, user.password);

      if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

      const token = jwt.sign({ id: user.\_id, role: user.role }, 'secret', { expiresIn: '1h' });

      res.json({ token, role: user.role });

  } catch (err) {

      res.status(500).json({ message: 'Server error' });

  }

});

// Register Route

router.post('/register', async (req, res) => {

    const { email, password, name, age, gender, contact, medicalHistory, role } = req.body;

    try {

      const existingUser = await User.findOne({ email });

      if (existingUser) {

        return res.status(400).json({ message: 'User already exists' });

      }

      const hashedPassword = await bcrypt.hash(password, 10);

      const newUser = new User({

        email,

        password: hashedPassword,

        name,

        age: parseInt(age, 10),  // Convert age to an integer

        gender,

        contact,

        medicalHistory,

        role

      });

      await newUser.save();

      console.log('User saved successfully:', newUser);

      res.status(201).json({ message: 'Registration successful' });

    } catch (err) {

      console.error('Error during registration:', err);

      res.status(500).json({ message: 'Server error' });

    }

  });

// Fetch all registered users (For Admin)

router.get('/users', async (req, res) => {

    try {

        const users = await User.find({ role: { $in: ['patient', 'doctor'] } }).select('-password');

        res.json(users);

    } catch (err) {

        console.error(err); // Log the error

        res.status(500).json({ message: 'Server error' });

    }

});

// Remove a user (For Admin)

router.delete('/users/:id', async (req, res) => {

    try {

        await User.findByIdAndDelete(req.params.id);

        res.json({ message: 'User deleted successfully' });

    } catch (err) {

        res.status(500).json({ message: 'Server error' });

    }

});

// In-memory storage for availability

const availability = {

    patient: false, // Initially, the patient is not available

    doctor: false,  // Initially, the doctor is not available

};

// Update availability status

router.post('/update-availability', (req, res) => {

    const { role, status } = req.body; // Get the role (doctor/patient) and status (true/false) from the request body

    // Check if the role is valid (either 'patient' or 'doctor')

    if (role === 'patient' || role === 'doctor') {

        availability[role] = status; // Update the availability status

        return res.json({ message: `${role} availability updated`, availability });

    }

    // If the role is invalid, return an error

    return res.status(400).json({ message: 'Invalid role' });

});

// Get availability status

router.get('/availability', (req, res) => {

    res.json(availability); // Respond with the current availability status

});

module.exports = router;

**server.js: (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\backend)**

const express = require('express');

const mongoose = require('mongoose');

const cors = require('cors');

const http = require('http');

const { Server } = require('socket.io');

const app = express();

const server = http.createServer(app);

const io = new Server(server, {

    cors: {

        origin: '\*', // Allow all origins (for development purposes)

        methods: ['GET', 'POST'],

    },

});

// Middleware

app.use(cors());

app.use(express.json());

// MongoDB Connection

mongoose.connect('mongodb://127.0.0.1:27017/hospital', {

    useNewUrlParser: true,

    useUnifiedTopology: true,

})

    .then(() => console.log('MongoDB connected'))

    .catch((err) => console.log(err));

// Routes

const authRoutes = require('./routes/auth');

app.use('/api/auth', authRoutes);

// WebSocket for Video Call & Chat

io.on('connection', (socket) => {

    console.log('A user connected:', socket.id);

    // Handle joining a room

    socket.on('join-room', (roomId, userId) => {

        console.log(`${userId} joined room: ${roomId}`);

        socket.join(roomId);

        socket.to(roomId).emit('user-connected', userId);

        // Handle WebRTC signaling

        socket.on('offer', (data) => {

            console.log('Offer received:', data);

            socket.to(roomId).emit('offer', data);

        });

        socket.on('answer', (data) => {

            console.log('Answer received:', data);

            socket.to(roomId).emit('answer', data);

        });

        socket.on('ice-candidate', (data) => {

            console.log('ICE Candidate received:', data);

            socket.to(roomId).emit('ice-candidate', data);

        });

        // Handle user disconnection

        socket.on('disconnect', () => {

            console.log(`${userId} disconnected`);

            socket.to(roomId).emit('user-disconnected', userId);

        });

    });

    // Handle real-time chat messages

    socket.on('chat-message', ({ roomId, userId, message }) => {

        console.log(`Chat Message from ${userId} in ${roomId}: ${message}`);

        io.to(roomId).emit('chat-message', { userId, message });

    });

});

// Start Server

const PORT = 5000;

server.listen(PORT, () => console.log(`Server running on port ${PORT}`));

**Fronted Code:**

**App.js: (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src)**

import React from 'react';

import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

import Dashboard from './components/Dashboard';

import Login from './components/Login';

import VideoCall from './components/VideoCall';

import Availability from './components/Availability';

import Register from './components/Register';

import AdminDashboard from './components/AdminDashboard';

function App() {

    return (

        <Router>

            <Routes>

                <Route path="/" element={<Dashboard />} />

                <Route path="/login" element={<Login />} />

                <Route path="/register" element={<Register />} />

                <Route path="/availability" element={<Availability />} />

                <Route path="/video-call" element={<VideoCall />} />

                <Route path="/admin-dashboard" element={<AdminDashboard />} />  {/\* New Admin Route \*/}

            </Routes>

        </Router>

    );

}

export default App;

**Availability.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React, { useState, useEffect } from 'react';

import { useNavigate } from 'react-router-dom';

import axios from 'axios';

import Chat from './Chat';

import '../styles.css'; // ✅ Import CSS file

const Availability = () => {

const [availability, setAvailability] = useState({ patient: false, doctor: false });

const [canStartCall, setCanStartCall] = useState(false);

const navigate = useNavigate();

const userEmail = sessionStorage.getItem('userEmail');

const userRole = sessionStorage.getItem('userRole');

const userId = userEmail;

const roomId = 'hospital\_chat\_room';

useEffect(() => {

const fetchAvailability = async () => {

const res = await axios.get('http://localhost:5000/api/auth/availability');

setAvailability(res.data);

setCanStartCall(res.data.patient && res.data.doctor);

};

fetchAvailability();

const interval = setInterval(fetchAvailability, 5000);

return () => clearInterval(interval);

}, []);

const handleStartCall = () => {

navigate('/video-call');

};

return (

<div className="availability-container"> {/\* ✅ Center the entire content \*/}

<div className="availability-content">

<h1>Availability</h1>

<p>Patient Available: {availability.patient ? 'Yes' : 'No'}</p>

<p>Doctor Available: {availability.doctor ? 'Yes' : 'No'}</p>

<button onClick={handleStartCall} disabled={!canStartCall}>

Start Call

</button>

</div>

{/\* ✅ Centered Chat Box \*/}

<Chat roomId={roomId} userId={userId} />

</div>

);

};

export default Availability;

**Dashboard.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React from 'react';

import { useNavigate } from 'react-router-dom';

import '../styles.css'; // Importing the CSS file

const Dashboard = () => {

    const navigate = useNavigate();

    return (

        <div className="dashboard"> {/\* Apply the "dashboard" class \*/}

            <h1>Hospital Management System</h1>

            <button onClick={() => navigate('/login?role=patient')}>Patient</button>

            <button onClick={() => navigate('/login?role=doctor')}>Doctor</button>

            <button onClick={() => navigate('/login?role=admin')}>Admin</button>

        </div>

    );

};

export default Dashboard;

**Login.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React, { useState } from 'react';

import { useNavigate, useLocation } from 'react-router-dom';

import axios from 'axios';

import '../styles.css'; // Importing the CSS file

const Login = () => {

    const [email, setEmail] = useState('');

    const [password, setPassword] = useState('');

    const navigate = useNavigate();

    const location = useLocation();

    const role = new URLSearchParams(location.search).get('role') || 'user';  // Default to 'user' if role is null

    <h1>{role.charAt(0).toUpperCase() + role.slice(1)} Login</h1>

    const handleLogin = async () => {

        try {

            const res = await axios.post('http://localhost:5000/api/auth/login', { email, password });

            if (res.data.role) {

                sessionStorage.setItem('userEmail', email);

                sessionStorage.setItem('userRole', res.data.role);

                if (res.data.role === 'admin') {

                    navigate('/admin-dashboard'); // Redirect to Admin Dashboard

                } else {

                    await axios.post('http://localhost:5000/api/auth/update-availability', {

                        role: res.data.role,

                        status: true,

                    });

                    navigate('/availability');

                }

            } else {

                alert('Invalid role');

            }

        } catch (err) {

            alert('Invalid credentials');

        }

    };

    return (

        <div className="login-container"> {/\* Apply the "login-container" class \*/}

            <h1>{role.charAt(0).toUpperCase() + role.slice(1)} Login</h1>

            <input type="email" placeholder="Email" value={email} onChange={(e) => setEmail(e.target.value)} />

            <input type="password" placeholder="Password" value={password} onChange={(e) => setPassword(e.target.value)} />

            <button onClick={handleLogin}>Login</button>

            {/\* Add the registration link below the login button \*/}

            <p>

                Don't have an account? <a href="/register">Register here</a>

            </p>

        </div>

    );

};

export default Login;

**VideoCall.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React, { useEffect, useRef, useState } from 'react';

import { io } from 'socket.io-client';

import '../styles.css'; // Importing the CSS file

const VideoCall = () => {

const myVideo = useRef(null);

const remoteVideo = useRef(null);

const socket = useRef();

const peerConnection = useRef(null);

const [roomId] = useState('room1');

const [callStarted, setCallStarted] = useState(false);

const [isVideoOn, setIsVideoOn] = useState(true);

const [isAudioOn, setIsAudioOn] = useState(true);

const [localStream, setLocalStream] = useState(null);

const servers = {

iceServers: [

{ urls: 'stun:stun.l.google.com:19302' }, // Public STUN server

],

};

useEffect(() => {

// Initialize Socket.IO

socket.current = io('http://192.204.1.107:5000'); // Replace with your backend URL

// Get local media stream

navigator.mediaDevices.getUserMedia({ video: true, audio: true })

.then((stream) => {

setLocalStream(stream);

myVideo.current.srcObject = stream;

// Ensure video plays after loading

myVideo.current.onloadedmetadata = () => {

myVideo.current.play().catch(err => console.error('Error playing local video:', err));

};

// Initialize PeerConnection

peerConnection.current = new RTCPeerConnection(servers);

// Add local stream tracks to peer connection

stream.getTracks().forEach(track => {

peerConnection.current.addTrack(track, stream);

});

// Handle ICE candidate exchange

peerConnection.current.onicecandidate = (event) => {

if (event.candidate) {

socket.current.emit('ice-candidate', event.candidate);

}

};

// Handle remote stream

peerConnection.current.ontrack = (event) => {

if (remoteVideo.current) {

remoteVideo.current.srcObject = event.streams[0];

remoteVideo.current.play().catch(err => console.error('Error playing remote video:', err));

}

};

// Handle connection state changes

peerConnection.current.onconnectionstatechange = () => {

if (peerConnection.current.connectionState === 'disconnected') {

console.warn('Peer disconnected, attempting reconnection...');

}

};

// Join the room

socket.current.emit('join-room', roomId, socket.current.id);

})

.catch(err => console.error('Error accessing media devices:', err));

// Socket event listeners

socket.current.on('offer', async (data) => {

if (peerConnection.current.signalingState !== "stable") {

console.warn("Received offer but signaling state is", peerConnection.current.signalingState);

return;

}

try {

await peerConnection.current.setRemoteDescription(new RTCSessionDescription(data));

const answer = await peerConnection.current.createAnswer();

await peerConnection.current.setLocalDescription(answer);

socket.current.emit('answer', answer);

} catch (error) {

console.error("Error handling offer:", error);

}

});

socket.current.on('answer', async (data) => {

if (peerConnection.current.signalingState !== "have-local-offer") {

console.warn("Received answer but signaling state is", peerConnection.current.signalingState);

return;

}

try {

await peerConnection.current.setRemoteDescription(new RTCSessionDescription(data));

} catch (error) {

console.error("Error setting remote description:", error);

}

});

socket.current.on('ice-candidate', async (data) => {

try {

await peerConnection.current.addIceCandidate(data);

} catch (err) {

console.error('Error adding received ICE candidate', err);

}

});

// When a new user joins, create and send an offer

socket.current.on('user-connected', async () => {

const offer = await peerConnection.current.createOffer();

await peerConnection.current.setLocalDescription(offer);

socket.current.emit('offer', offer);

});

// Cleanup on component unmount

return () => {

if (socket.current) {

socket.current.disconnect();

}

if (peerConnection.current) {

peerConnection.current.close();

peerConnection.current = null;

}

};

}, [roomId]);

// Toggle Video On/Off

const toggleVideo = () => {

if (localStream) {

const videoTrack = localStream.getVideoTracks()[0];

videoTrack.enabled = !videoTrack.enabled;

setIsVideoOn(videoTrack.enabled);

}

};

// Toggle Audio On/Off

const toggleAudio = () => {

if (localStream) {

const audioTrack = localStream.getAudioTracks()[0];

audioTrack.enabled = !audioTrack.enabled;

setIsAudioOn(audioTrack.enabled);

}

};

// End Call

const endCall = () => {

if (localStream) {

localStream.getTracks().forEach(track => track.stop());

}

if (socket.current) {

socket.current.disconnect();

}

setCallStarted(false);

};

const handleStartCall = () => {

setCallStarted(true);

if (remoteVideo.current) {

remoteVideo.current.play().catch(err => console.error('Error playing remote video:', err));

}

};

return (

<div>

<div className="video-container">

<div>

<h3>My Camera</h3>

<video ref={myVideo} className="video-frame" autoPlay muted />

</div>

<div>

<h3>Remote Camera</h3>

<video ref={remoteVideo} className="video-frame" autoPlay />

</div>

</div>

{/\* Buttons for Video, Audio, and Call End \*/}

<div className="video-controls">

<button onClick={toggleVideo}>

{isVideoOn ? 'Turn Off Camera' : 'Turn On Camera'}

</button>

<button onClick={toggleAudio}>

{isAudioOn ? 'Mute' : 'Unmute'}

</button>

<button className="end-call-btn" onClick={endCall}>

End Call

</button>

</div>

{!callStarted && (

<button className="start-call-btn" onClick={handleStartCall}>Start Call</button>

)}

</div>

);

};

export default VideoCall;

**Chat.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React, { useState, useEffect, useRef } from 'react';

import { io } from 'socket.io-client';

import '../styles.css'; // Importing the CSS file

const Chat = ({ roomId, userId }) => {

    const [messages, setMessages] = useState([]);

    const [input, setInput] = useState('');

    const socket = useRef(null);

    useEffect(() => {

        socket.current = io('http://localhost:5000'); // Connect to backend

        socket.current.emit('join-room', roomId, userId);

        socket.current.off('chat-message').on('chat-message', ({ userId: senderId, message }) => {

            setMessages((prev) => [...prev, { senderId, message }]);

        });

        return () => {

            socket.current.disconnect();

        };

    }, [roomId, userId]);

    const sendMessage = () => {

        if (input.trim()) {

            socket.current.emit('chat-message', { roomId, userId, message: input });

            setInput(''); // ✅ Clear input after sending

        }

    };

    return (

        <div className="chat-container">

            <div className="chat-header">Chat</div>

            <div className="chat-messages">

                {messages.map((msg, index) => (

                    <p

                        key={index}

                        className={`message ${msg.senderId === userId ? 'sent' : 'received'}`}

                    >

                        <strong>{msg.senderId}:</strong> {msg.message}

                    </p>

                ))}

            </div>

            <div className="chat-input-container">

                <input

                    type="text"

                    value={input}

                    onChange={(e) => setInput(e.target.value)}

                    className="chat-input"

                    placeholder="Type a message..."

                />

                <button onClick={sendMessage} className="send-button">Send</button>

            </div>

        </div>

    );

};

export default Chat;

**Register.js (Path: C:\Users\Fahim\OneDrive\Desktop\Fahim\frontend\src\components)**

import React, { useState } from 'react';

import { useNavigate } from 'react-router-dom';

import axios from 'axios';

const Register = () => {

  const [formData, setFormData] = useState({

    email: '',

    password: '',

    name: '',

    age: '',

    gender: 'male',

    contact: '',

    medicalHistory: '',

    role: 'patient'

  });

  const navigate = useNavigate();

  const handleChange = (e) => {

    setFormData({ ...formData, [e.target.name]: e.target.value });

  };

  const handleRegister = async () => {

    try {

      const res = await axios.post('http://localhost:5000/api/auth/register', formData);

      alert(res.data.message);

      navigate('/login');

    } catch (err) {

      alert(err.response?.data?.message || 'Registration failed. Please try again.');

    }

  };

  return (

    <div className="login-container">

      <h1>Registration</h1>

      {/\* Role Selection at the Top \*/}

      <label>Register as a: </label>

      <select name="role" value={formData.role} onChange={handleChange}>

        <option value="patient">Patient</option>

        <option value="doctor">Doctor</option>

      </select>

      <input type="text" name="name" placeholder="Full Name" value={formData.name} onChange={handleChange} />

      <input type="number" name="age" placeholder="Age" value={formData.age} onChange={handleChange} />

      {/\* Gender Label and Dropdown \*/}

      <label></label>

      <select name="gender" value={formData.gender} onChange={handleChange}>

        <option value="male">Male</option>

        <option value="female">Female</option>

        <option value="other">Other</option>

      </select>

      <input type="text" name="contact" placeholder="Contact Number" value={formData.contact} onChange={handleChange} />

      <textarea

        name="medicalHistory"

        placeholder={formData.role === 'doctor' ? "Doctor's Additional Information" : "Medical History"}

        value={formData.medicalHistory}

        onChange={handleChange}

      />

      <input type="email" name="email" placeholder="Email" value={formData.email} onChange={handleChange} />

      <input type="password" name="password" placeholder="Password" value={formData.password} onChange={handleChange} />

      <button onClick={handleRegister}>Register</button>

    </div>

  );

};

export default Register;