### **Backend (Node.js, Express, MongoDB, Socket.io)**

#### Folder Structure:

/backend

/models

- User.js

- ChatRoom.js

- Task.js

/routes

- authRoutes.js

- chatRoutes.js

- taskRoutes.js

server.js

#### server.js (Backend Entry Point)

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const cors = require('cors');

const socketio = require('socket.io');

const http = require('http');

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

const chatRoutes = require('./routes/chatRoutes');

const taskRoutes = require('./routes/taskRoutes');

const app = express();

const server = http.createServer(app);

const io = socketio(server);

app.use(cors());

app.use(bodyParser.json());

mongoose.connect('mongodb://localhost:27017/your-database', {

useNewUrlParser: true,

useUnifiedTopology: true,

});

// Socket.io implementation for real-time features

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

console.log('User connected');

// Handle chat events

socket.on('chatMessage', (message) => {

io.emit('chatMessage', message); // Broadcast the message to all connected clients

});

// Handle task board events

socket.on('taskUpdate', (updatedTask) => {

io.emit('taskUpdate', updatedTask); // Broadcast task updates to all clients

});

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

console.log('User disconnected');

});

});

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

app.use('/api/chat', chatRoutes);

app.use('/api/tasks', taskRoutes);

const PORT = process.env.PORT || 5000;

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

#### authRoutes.js (Example Authentication Routes)

const express = require('express');

const router = express.Router();

// Authentication routes: login, register, logout, etc.

module.exports = router;

#### chatRoutes.js (Example Chat Routes)

const express = require('express');

const router = express.Router();

// Chat-related routes: create/join rooms, send messages, etc.

module.exports = router;

#### taskRoutes.js (Example Task Routes)

const express = require('express');

const router = express.Router();

// Task-related routes: CRUD operations on tasks, etc.

module.exports = router;

### **Frontend (React with Socket.io Client)**

#### Folder Structure

/frontend

/src

/components

- ChatRoom.js

- TaskBoard.js

/services

- api.js

- App.js

- index.js

#### api.js (Frontend API Service)

import socketIOClient from 'socket.io-client';

const socket = socketIOClient('http://localhost:5000'); // Backend socket connection

const api = {

// API functions for chat and task board operations

};

export default api;

#### ChatRoom.js

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

import api from '../services/api';

const ChatRoom = () => {

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

const [newMessage, setNewMessage] = useState('');

useEffect(() => {

// Socket.io event listeners for chat messages

socket.on('chatMessage', (message) => {

setMessages([...messages, message]);

});

return () => {

// Clean up socket.io event listeners

socket.off('chatMessage');

};

}, [messages]);

const sendMessage = () => {

// Send new message to the server and broadcast it to all clients

api.sendMessage(newMessage);

setNewMessage('');

};

return (

<div>

{/\* Chat interface \*/}

</div>

);

};

export default ChatRoom;

#### TaskBoard.js

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

import api from '../services/api';

const TaskBoard = () => {

const [tasks, setTasks] = useState([]);

useEffect(() => {

// Socket.io event listeners for task updates

socket.on('taskUpdate', (updatedTask) => {

// Update tasks in real-time

setTasks(tasks.map((task) => (task.id === updatedTask.id ? updatedTask : task)));

});

return () => {

// Clean up socket.io event listeners

socket.off('taskUpdate');

};

}, [tasks]);

const updateTask = (updatedTask) => {

// Update task on the server and broadcast it to all clients

api.updateTask(updatedTask);

};

return (

<div>

{/\* Task board interface \*/}

</div>

);

};

export default TaskBoard;