Backend Documentation for Real-Time Support System

Overview

This backend uses Express and Socket.io to facilitate real-time communication between users and support agents. It manages user connections, call requests, call transfers, messaging, and agent availability status.

Technologies Used

- Node.js
- Express.js
- Socket.io

Server Setup

The server listens on port `8080` and uses Socket.io to enable real-time communication between clients.

```
const http = require('http');
const express = require('express');
const app = express();
const { Server } = require('socket.io');

const server = http.createServer(app);
const io = new Server(server, {
    cors: {
        origin: '*',
        methods: ['POST', 'GET', 'PATCH', 'PUT'],
    }
});

server.listen(8080, () => {
    console.log('Listening on port 8080');
});
```

Global Variables

- `agents`: Stores connected agents.
- `users`: Stores connected users.

```
let agents = [];
let users = [];
```

Socket.io Events

1. `connectUser`

Registers a user with a unique socket ID.

```
socket.on('connectUser', (data) => {
    data = { ...data, socketId: socket.id };
    users = [...users, data];
});
```

2. `connectAgent`

Registers an agent and sets them as available.

```
socket.on('connectAgent', (data) => {
    data = { ...data, socketId: socket.id, available: true };
    agents = [...agents, data];
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

3. `callAgent`

Finds an available agent and assigns them to a user.

```
socket.on("callAgent", (data) => {
    let user = users.find(u => u.id === data.userId);
    let agent = agents.find(a => a.available);
    if (!agent) return;

    user.roomId = data.roomId;
    user.available = false;
    agent.roomId = data.roomId;
    agent.available = false;

    io.to(agent.socketId).emit("callAgent", data);
    io.to(user.socketId).emit("callAgent", data);
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

4. `rejectCall`

Handles call rejection by updating agent availability.

```
socket.on("rejectCall", (data) => {
    let agent = agents.find(a => a.id === data.agentId);
    if (agent) agent.available = true;
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

5. `callTransfer`

Transfers a call from one agent to another available agent.

```
socket.on("callTransfer", (data) => {
    let prevAgent = agents.find(a => a.id === data.agentId);
```

```
    let nextAgent = agents.find(a => a.available);
    if (!prevAgent || !nextAgent) return;

    nextAgent.available = false;
    nextAgent.roomId = prevAgent.roomId;
    prevAgent.available = true;
    prevAgent.roomId = '';

    let user = users.find(u => u.roomId === data.roomId);
    io.to(nextAgent.socketId).emit("callTransfer", data);
    io.to(user.socketId).emit("callTransfer", { agentId:
nextAgent.id, roomId: data.roomId });
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

6. `toggleAvailability`

Updates an agent's availability status.

```
socket.on("toggleAvailability", (data) => {
    let agent = agents.find(a => a.id === data.id);
    if (agent) agent.available = data.available;
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

7. `userMessage` & `agentMessage`

Handles real-time messaging between users and agents.

```
socket.on("userMessage", (data) => {
    let agent = agents.find(a => a.roomId === data.roomId);
    if (agent) io.to(agent.socketId).emit("userMessage",
{ messageInput: data.messageInput });
});

socket.on("agentMessage", (data) => {
    let user = users.find(u => u.roomId === data.roomId);
    if (user) io.to(user.socketId).emit("agentMessage",
{ messageInput: data.messageInput });
});
```

8. `leaveCall`

Handles call termination and updates availability.

```
socket.on("leaveCall", (data) => {
    let user = users.find(u => u.roomId === data.roomId);
    let agent = agents.find(a => a.id === data.agentId);

    if (user) user.available = true;
    if (agent) agent.available = true;

    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
```

```
});
```

9. `mute`

Handles muting functionality during a call.

```
socket.on("mute", (data) => {
    let agent = agents.find(a => a.roomId === data.roomId);
    if (agent) io.to(agent.socketId).emit("mute", data);
});
```

10. `disconnect`

Handles user and agent disconnection, updating their availability.

```
socket.on('disconnect', () => {
    users = users.filter(u => u.socketId !== socket.id);
    agents = agents.filter(a => a.socketId !== socket.id);
    io.emit("getAvailableAgents", { availableAgents: agents.filter(a
=> a.available).length });
});
```

Conclusion

This backend facilitates seamless real-time communication between users and agents, allowing them to connect, transfer calls, and exchange messages efficiently. It ensures proper agent availability tracking, call transfers, and real-time messaging to enhance user support interactions.