PENDING MESSAGES AND NOTIFYING THE USER

To address the issue of notifying a user when a message is sent to them while they are in a different chat room and ensuring they receive the "deferred" message when they select the sender, you can implement a notification system and enhance the message handling logic. Below, I’ll outline the necessary changes to your code, focusing on these two requirements:

1. \*\*Notifying the selected user of a new message in another room.\*\*

2. \*\*Ensuring the selected user receives the deferred message when they select the sender.\*\*

I’ll provide specific code modifications for both the backend (`server.js`) and frontend (`app.js`), along with explanations. The solution will involve storing messages for users who are not in the same room, notifying them of pending messages, and displaying those messages when they switch to the sender’s chat.

---

### Solution Overview

- \*\*Backend Changes\*\*:

- Store messages in a temporary queue for users who are not in the same room.

- Emit a notification event to the target user when a message is sent to them.

- Deliver queued messages when the user joins the sender’s private room.

- \*\*Frontend Changes\*\*:

- Display notifications (e.g., a badge or alert) for users with pending messages.

- Load and display queued messages when the user selects the sender.

- \*\*Assumptions\*\*:

- Messages are stored temporarily on the server until delivered or the user disconnects.

- Notifications are visual (e.g., a badge on the user’s name in the user list).

- Messages are persisted in `localStorage` on the client side as per your existing code.

---

### Backend Modifications (`server.js`)

We’ll add a message queue to store messages for users who are not in the same room and introduce a notification system.

1. \*\*Add a Message Queue\*\*:

Create a data structure to store pending messages for users.

2. \*\*Modify the `message` Event Handler\*\*:

Check if the target user is in the specified room. If not, queue the message and send a notification.

3. \*\*Deliver Queued Messages\*\*:

When a user joins a private room, deliver any queued messages for that room.

Here’s the updated `server.js`:

```javascript

// server.js

const express = require('express');

const http = require('http');

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

const path = require('path');

const app = express();

const server = http.createServer(app);

const io = socketIo(server, {

cors: {

origin: ["http://localhost:3500", "http://127.0.0.1:5500"],

methods: ["GET", "POST"],

},

maxHttpBufferSize: 10e6

});

require('dotenv').config();

const PORT = process.env.PORT || 3500;

const SYSTEM = "Admin";

app.use(express.static(path.join(\_\_dirname, '../FrontEnd')));

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

res.sendFile(path.join(\_\_dirname, '../FrontEnd', 'index.html'));

});

const UsersState = {

users: [],

setUsers: function (newUsersArray) {

this.users = newUsersArray;

}

};

// Store pending messages for users

const PendingMessages = {

messages: {}, // { userId: [{ message, room, senderId }], ... }

addMessage: function (userId, message, room, senderId) {

if (!this.messages[userId]) {

this.messages[userId] = [];

}

this.messages[userId].push({ message, room, senderId });

},

getMessages: function (userId, room) {

if (!this.messages[userId]) return [];

return this.messages[userId].filter(msg => msg.room === room);

},

clearMessages: function (userId, room) {

if (this.messages[userId]) {

this.messages[userId] = this.messages[userId].filter(msg => msg.room !== room);

if (this.messages[userId].length === 0) {

delete this.messages[userId];

}

}

}

};

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

socket.emit('message', buildMsg(SYSTEM, "Welcome to WinChat!"));

socket.on('enterApp', ({ name }) => {

const user = activateUser(socket.id, name, null);

io.emit('userList', {

users: getAllUsers(),

pendingMessages: getPendingMessagesForAllUsers()

});

socket.emit('message', buildMsg(SYSTEM, `You have joined WinChat`));

socket.broadcast.emit('message', buildMsg(SYSTEM, `${user.name} has joined WinChat`));

});

socket.on('joinPrivateRoom', ({ name, targetUser }) => {

const user = getUser(socket.id);

if (!user) return;

const room = getPrivateRoomId(name, targetUser);

const prevRoom = user.room;

if (prevRoom) {

socket.leave(prevRoom);

}

user.room = room;

UsersState.setUsers([...UsersState.users]);

socket.join(room);

socket.emit('message', buildMsg(SYSTEM, `You have started a chat with ${targetUser}`));

// Deliver any pending messages for this room

const targetUserId = getUserSocketIdByName(targetUser);

const pendingMessages = PendingMessages.getMessages(socket.id, room);

pendingMessages.forEach(({ message }) => {

socket.emit('message', message);

});

PendingMessages.clearMessages(socket.id, room);

// Update user list with pending message counts

io.emit('userList', {

users: getAllUsers(),

pendingMessages: getPendingMessagesForAllUsers()

});

});

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

const user = getUser(socket.id);

userLeavesApp(socket.id);

if (user) {

io.emit('message', buildMsg(SYSTEM, `${user.name} has left WinChat`));

io.emit('userList', {

users: getAllUsers(),

pendingMessages: getPendingMessagesForAllUsers()

});

// Clear pending messages for disconnected user

delete PendingMessages.messages[socket.id];

}

});

socket.on('message', ({ name, text, room, type, fileName }) => {

const message = buildMsg(name, text, room, type, fileName);

if (room) {

// Find the target user (other user in the private room)

const usersInRoom = room.split('\_');

const targetUserName = usersInRoom.find(u => u !== name);

const targetUser = UsersState.users.find(u => u.name === targetUserName);

if (targetUser) {

// Check if target user is in the same room

if (targetUser.room === room) {

io.to(room).emit('message', message);

} else {

// Queue the message and notify the target user

PendingMessages.addMessage(targetUser.id, message, room, socket.id);

io.to(targetUser.id).emit('notification', {

from: name,

room

});

// Emit to sender to display the message immediately

socket.emit('message', message);

}

} else {

// If target user is not found, still emit to sender

socket.emit('message', message);

}

} else {

// Broadcast to all users

io.emit('message', message);

}

});

socket.on('activity', ({ name, room }) => {

if (room) {

socket.broadcast.to(room).emit('activity', name);

} else {

socket.broadcast.emit('activity', name);

}

});

// Voice call handlers remain unchanged

socket.on('voice-offer', ({ target, offer }) => {

const targetSocket = getUserSocketIdByName(target);

if (targetSocket) {

io.to(targetSocket).emit('voice-offer', {

from: getUser(socket.id).name,

offer

});

}

});

socket.on('voice-answer', ({ target, answer }) => {

const targetSocket = getUserSocketIdByName(target);

if (targetSocket) {

io.to(targetSocket).emit('voice-answer', {

answer

});

}

});

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

const targetSocket = getUserSocketIdByName(target);

if (targetSocket) {

io.to(targetSocket).emit('ice-candidate', {

candidate

});

}

});

});

function getUserSocketIdByName(name) {

const user = UsersState.users.find(u => u.name === name);

return user?.id;

}

function buildMsg(name, text, room = null, type = 'text', fileName = '') {

return {

name,

text,

date: new Intl.DateTimeFormat('default', {

year: 'numeric',

month: 'numeric',

day: 'numeric'

}).format(new Date()),

time: new Intl.DateTimeFormat('default', {

hour: 'numeric',

minute: 'numeric',

second: 'numeric'

}).format(new Date()),

room,

type,

fileName

};

}

function activateUser(id, name, room) {

const user = { id, name, room };

UsersState.setUsers([

...UsersState.users.filter(user => user.id !== id),

user

]);

return user;

}

function userLeavesApp(id) {

UsersState.setUsers(

UsersState.users.filter(user => user.id !== id)

);

}

function getUser(id) {

return UsersState.users.find(user => user.id === id);

}

function getAllUsers() {

return UsersState.users;

}

function getPrivateRoomId(user1, user2) {

return [user1, user2].sort().join('\_');

}

// Helper function to get pending message counts for all users

function getPendingMessagesForAllUsers() {

const pendingCounts = {};

Object.keys(PendingMessages.messages).forEach(userId => {

const user = getUser(userId);

if (user) {

const counts = {};

PendingMessages.messages[userId].forEach(({ room }) => {

const otherUser = room.split('\_').find(name => name !== user.name);

counts[otherUser] = (counts[otherUser] || 0) + 1;

});

pendingCounts[user.name] = counts;

}

});

return pendingCounts;

}

server.listen(PORT, () => {

console.log(`WinChat Server is running on http://localhost:${PORT}`);

});

```

\*\*Key Backend Changes\*\*:

- \*\*PendingMessages Object\*\*: Stores messages for users not in the same room, keyed by user ID, with each message including the message data, room, and sender ID.

- \*\*Message Handler\*\*: Checks if the target user is in the room. If not, queues the message and emits a `notification` event to the target user.

- \*\*JoinPrivateRoom Handler\*\*: Delivers queued messages when a user joins a private room and clears them from the queue.

- \*\*UserList Update\*\*: Includes `pendingMessages` to inform clients about pending message counts for display.

- \*\*Disconnect Handler\*\*: Clears pending messages for a user who disconnects.

- \*\*getPendingMessagesForAllUsers\*\*: Aggregates pending message counts per user for notification badges.

---

### Frontend Modifications (`app.js`)

We’ll update the frontend to display notifications (e.g., a badge on the user’s name) and ensure messages are displayed when the user switches to the sender’s chat.

1. \*\*Display Notifications\*\*:

Add a badge to the user’s name in the user list to indicate pending messages.

2. \*\*Handle Notification Events\*\*:

Update the user list when a notification is received.

3. \*\*Load Queued Messages\*\*:

Your existing `loadMessages` function already handles displaying messages from `localStorage`. Ensure it works with the delivered queued messages.

Here’s the updated `app.js`:

```javascript

// app.js

import { MessageBox } from "./functlib.js";

import { initVoiceCallFeatures } from './voiceCall.js';

const msgInput = document.querySelector('#message');

const nameInput = document.querySelector('#name');

const activity = document.querySelector('.activity');

const userName = document.querySelector('.user-name');

const chatDisplay = document.querySelector('.chat-display');

const chatEraser = document.querySelector('#chatEraser');

const chatSmile = document.querySelector('#chat-smile');

const chatImages = document.querySelector('#chat-images');

const chatPaperClip = document.querySelector('#chat-paperclip');

chatSmile.addEventListener('click', sendSmiley);

chatPaperClip.addEventListener('click', send\_a\_File);

chatImages.addEventListener('click', sendImage);

document.querySelector('.form-msg').addEventListener('submit', sendMessage);

document.querySelector('.form-join').addEventListener('submit', enterApp);

const socket = io('http://localhost:3500');

let selectedUser = null;

socket.on("message", (data) => {

const { name, text, time, room, type, fileName } = data;

if (

(selectedUser && room === getPrivateRoomId(nameInput.value, selectedUser.name)) ||

room === null

) {

let fromUser = name === nameInput.value;

const li = document.createElement('li');

li.className = fromUser ? 'post post--right' : 'post post--left';

if (name === 'Admin') {

li.innerHTML = `<div class="post\_\_admin">${text}</div>`;

li.className = "post\_\_admin";

} else if (type === 'image') {

li.innerHTML = `

<div class="post\_\_text ${fromUser ? 'post\_\_text--user' : 'post\_\_text--reply'}">

<img src="${text}" alt="${fileName}" style="max-width: 200px; max-height: 200px;" />

</div>

<div class="post\_\_header ${fromUser ? 'post\_\_header--user' : 'post\_\_header--reply'}">

<span class="post\_\_header--name">${fromUser ? '' : name}${room === null ? ' (All)' : ''}</span>

<span class="post\_\_header--time">${time}</span>

</div>`;

} else if (type === 'file') {

li.innerHTML = `

<div class="post\_\_text ${fromUser ? 'post\_\_text--user' : 'post\_\_text--reply'}">

<a href="${text}" download="${fileName}">${fileName}</a>

</div>

<div class="post\_\_header ${fromUser ? 'post\_\_header--user' : 'post\_\_header--reply'}">

<span class="post\_\_header--name">${fromUser ? '' : name}${room === null ? ' (All)' : ''}</span>

<span class="post\_\_header--time">${time}</span>

</div>`;

} else {

li.innerHTML = `

<div class="post\_\_text ${fromUser ? 'post\_\_text--user' : 'post\_\_text--reply'}">${text}</div>

<div class="post\_\_header ${fromUser ? 'post\_\_header--user' : 'post\_\_header--reply'}">

<span class="post\_\_header--name">${fromUser ? '' : name}${room === null ? ' (All)' : ''}</span>

<span class="post\_\_header--time">${time}</span>

</div>`;

}

chatDisplay.appendChild(li);

chatDisplay.scrollTop = chatDisplay.scrollHeight;

}

saveMessages(data, room);

});

let activityTimer;

socket.on("activity", (name) => {

if (selectedUser) {

activity.textContent = `${name} is typing...`;

clearTimeout(activityTimer);

activityTimer = setTimeout(() => {

activity.textContent = "";

}, 2000);

}

});

socket.on('userList', ({ users, pendingMessages }) => {

showUsers(users, pendingMessages);

});

socket.on('notification', ({ from, room }) => {

// Update user list to reflect new notification

socket.emit('requestUserList'); // Request updated user list

});

// Request user list to refresh notifications

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

if (nameInput.value) {

socket.emit('enterApp', { name: nameInput.value });

}

});

function showUsers(users, pendingMessages) {

userName.innerHTML = '';

if (users) {

users.forEach((user) => {

if (user.name !== nameInput.value) { // Don't show self

const userItem = document.createElement('li');

userItem.className = 'userItem';

const initials = getUserInitials(user.name);

const userIcon = document.createElement('div');

userIcon.className = 'userIcon';

userIcon.innerHTML = initials;

userIcon.style.backgroundColor = getRandomColor();

// Add notification badge if there are pending messages

const pendingCount = pendingMessages[user.name] && pendingMessages[user.name][nameInput.value] || 0;

if (pendingCount > 0) {

const badge = document.createElement('span');

badge.className = 'notification-badge';

badge.textContent = pendingCount;

badge.style.backgroundColor = '#ff4d4f';

badge.style.color = '#fff';

badge.style.borderRadius = '50%';

badge.style.padding = '2px 6px';

badge.style.fontSize = '12px';

badge.style.position = 'absolute';

badge.style.top = '-5px';

badge.style.right = '-5px';

userIcon.appendChild(badge);

}

if (selectedUser && selectedUser.name === user.name) {

userItem.classList.add('selected');

}

userItem.addEventListener('click', () => {

selectedUser = user;

chatDisplay.innerHTML = '';

userName.querySelectorAll('.userItem').forEach(item => item.classList.remove('selected'));

userItem.classList.add('selected');

activity.textContent = `Chatting with ${user.name}`;

const room = getPrivateRoomId(nameInput.value, user.name);

socket.emit('joinPrivateRoom', {

name: nameInput.value,

targetUser: user.name

});

loadMessages(room);

updateChatDisplay(`Started chat with ${user.name}`);

});

userItem.appendChild(userIcon);

const userNameText = document.createElement('span');

userNameText.textContent = user.name;

userItem.appendChild(userNameText);

userName.appendChild(userItem);

}

});

}

chatEraser.addEventListener('click', () => {

const room = selectedUser ? getPrivateRoomId(nameInput.value, selectedUser.name) : null;

deleteMessages(room);

chatDisplay.innerHTML = '';

updateChatDisplay('Chat history cleared');

});

}

// ... (rest of the functions remain unchanged: getRandomColor, getUserInitials, updateChatDisplay, sendMessage, enterApp, getPrivateRoomId, getTime\_Now, getDate\_Now, saveMessages, loadMessages, deleteMessages, sendSmiley, sendImage, send\_a\_File)

msgInput.addEventListener('keypress', (event) => {

const sendAllUsers = document.querySelector('#sendAllUsers').checked;

if (sendAllUsers) {

socket.emit('activity', { name: nameInput.value, room: null });

} else if (selectedUser) {

socket.emit('activity', { name: nameInput.value, room: getPrivateRoomId(nameInput.value, selectedUser.name) });

}

if (event.key === "Enter") document.querySelector("#sendMessage").click();

});

initVoiceCallFeatures({

socket,

nameInput,

selectedUserGetter: () => selectedUser

});

```

\*\*Key Frontend Changes\*\*:

- \*\*User List with Notifications\*\*: The `showUsers` function now accepts `pendingMessages` and displays a badge with the number of pending messages for each user.

- \*\*Notification Handler\*\*: Listens for the `notification` event and requests an updated user list to refresh notification badges.

- \*\*CSS for Notification Badge\*\*: Added styles for the notification badge (you’ll need to add this to your CSS file, see below).

- \*\*Message Display\*\*: The existing `loadMessages` function works with messages delivered from the server’s queue, as they are emitted via the `message` event.

---

### CSS Additions

Add the following styles to your CSS file (e.g., `styles.css`) to style the notification badge:

```css

.notification-badge {

background-color: #ff4d4f;

color: white;

border-radius: 50%;

padding: 2px 6px;

font-size: 12px;

position: absolute;

top: -5px;

right: -5px;

}

```

---

### How It Works

1. \*\*Notifying the Selected User\*\*:

- When User A sends a message to User B in a private room (e.g., `room = "A\_B"`), the backend checks if User B is in that room.

- If User B is not in the room, the message is queued in `PendingMessages` for User B’s socket ID, and a `notification` event is sent to User B.

- User B’s client receives the `notification` event, requests an updated user list, and displays a badge on User A’s name in the user list (e.g., showing “2” if there are two pending messages).

2. \*\*Receiving Deferred Messages\*\*:

- When User B clicks on User A in the user list, they join the private room `A\_B`.

- The backend emits any queued messages for `A\_B` to User B’s client via the `message` event.

- The client’s `message` event handler displays these messages and saves them to `localStorage` using `saveMessages`.

- The badge is removed because the `userList` event includes updated `pendingMessages` counts (zero after delivery).

---

### Additional Improvements

To enhance the user experience and robustness, consider these optional improvements:

1. \*\*Persistent Notifications\*\*:

- Store notification states in `localStorage` to persist across page refreshes.

- Modify `saveMessages` to track unread messages and display notifications even after a refresh.

2. \*\*Visual Feedback\*\*:

- Add a toast or popup notification when a new message arrives, in addition to the badge.

- Example: Use a library like Toastify or create a custom toast component.

3. \*\*Sound Notifications\*\*:

- Play a sound when a notification is received.

- Example:

```javascript

socket.on('notification', ({ from, room }) => {

const audio = new Audio('/path/to/notification-sound.mp3');

audio.play();

socket.emit('requestUserList');

});

```

4. \*\*Clear Notifications on Read\*\*:

- Clear the notification badge as soon as the user views the message, even before switching rooms, by tracking read status.

5. \*\*Message Queue Persistence\*\*:

- Persist the `PendingMessages` queue in a database (e.g., MongoDB) instead of memory to handle server restarts.

- This requires integrating a database and modifying `PendingMessages` to read/write from it.

6. \*\*User Feedback for Offline Users\*\*:

- If the target user is offline, notify the sender that the message will be delivered when the user comes online.

7. \*\*Rate Limiting\*\*:

- Add rate limiting to prevent spam notifications or messages.

8. \*\*UI Enhancements\*\*:

- Highlight the user list item with a subtle animation when a new notification arrives.

- Example CSS:

```css

.userItem.notify {

animation: pulse 1s ease-in-out;

}

@keyframes pulse {

0% { background-color: transparent; }

50% { background-color: rgba(255, 77, 79, 0.2); }

100% { background-color: transparent; }

}

```

---

### Testing the Solution

1. \*\*Setup\*\*:

- Ensure your server is running (`node server.js`).

- Open two browser windows (e.g., Chrome and Firefox) to simulate User A and User B.

- Log in as User A in one window and User B in another.

2. \*\*Test Notification\*\*:

- Have User A select User B and send a message while User B is in a different room (e.g., chatting with User C or in the global chat).

- Verify that User B sees a badge on User A’s name in the user list.

3. \*\*Test Message Delivery\*\*:

- Have User B click on User A in the user list.

- Verify that the queued messages appear in the chat display and the badge disappears.

4. \*\*Edge Cases\*\*:

- Test with User B offline (close their window) and then reconnecting.

- Test with multiple messages from User A to User B.

- Test with User A sending messages to multiple users.

---

This solution addresses both requirements efficiently while maintaining compatibility with your existing code. If you need help implementing additional features or debugging specific issues, let me know!