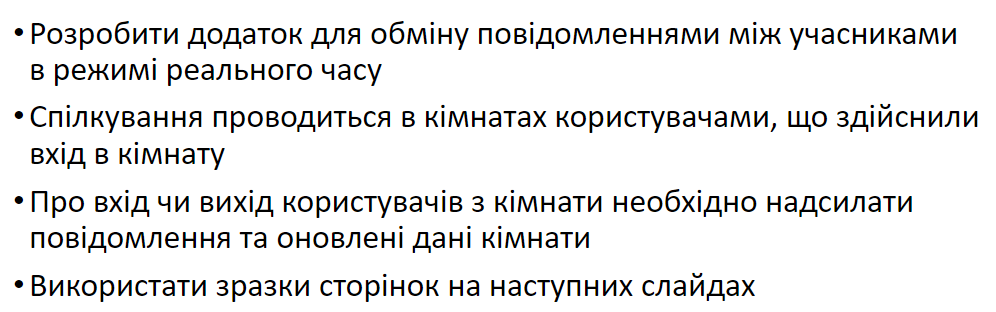
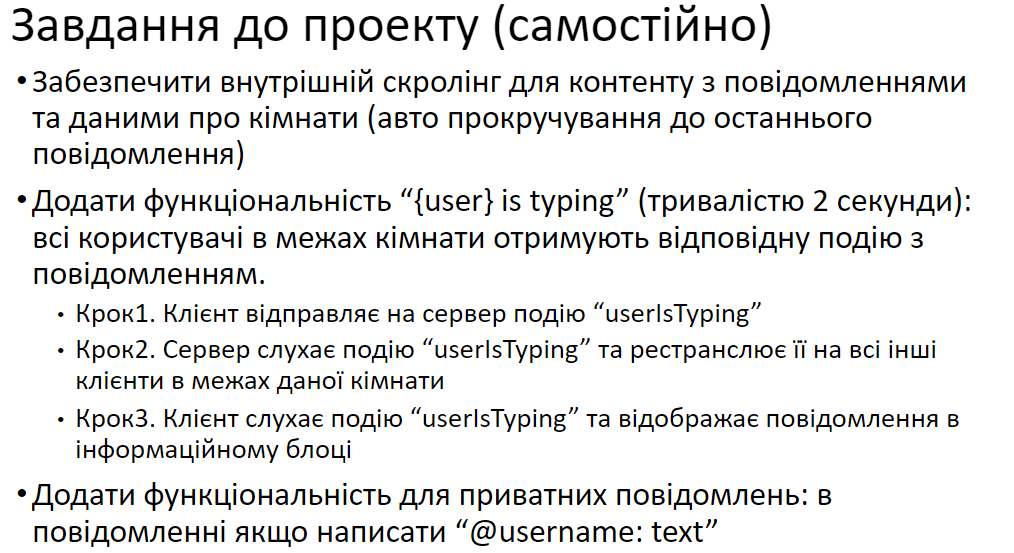
**ЛАБОРАТОРНА РОБОТА № 6**

**Протокол WebSocket. Використання Socket.io для розробки чат-додатків**

**Хід роботи:**

**Завдання:**

****

****

**Лістинг програми:**

**index.html**

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">  
 <link rel="stylesheet" href="./css/style.css">  
 <title>Приєднатись до чату</title>  
</head>  
<body>  
<div class="vw-100 vh-100 d-flex justify-content-center align-items-center">  
 <div class="form-container bg-info p-4">  
 <form action="/chat.html">  
 <div class="form-group">  
 <input type="text" class="form-control" id="nickname" name="username" placeholder="Логін">  
 </div>  
 <div class="form-group">  
 <input type="text" class="form-control" id="room" name="room" placeholder="Кімната">  
 </div>  
 <button type="submit" class="btn btn-warning">Приєднатись</button>  
 </form>  
 </div>  
</div>  
  
  
</body>  
</html>

**chat.html**

<!doctype html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">  
 <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="anonymous">  
 <link rel="stylesheet" href="./css/style.css">  
 <title>Document</title>  
</head>  
<body>  
<section>  
 <div class="container-fluid">  
 <div class="row">  
 <div id="sidebar" class="col-4 vh-100 bg-info p-4">  
  
 </div>  
  
 <div id="chat" class="col-8 d-flex flex-column flex-grow-1 vh-100 p-0">  
  
 <div id="messages" class="flex-grow-1">  
  
 </div>  
 <div id="typing-message">  
  
 </div>  
 <div id="compose">  
 <form id="message-form" class="form-inline">  
 <div class="form-group flex-grow-1">  
 <input type="text" name="message" placeholder="Message" required autocomplete="off">  
 </div>  
 <div class="form-group">  
 <button class="btn btn-warning btn-lg">Send</button>  
 </div>  
  
 </form>  
 </div>  
 </div>  
 </div>  
 </div>  
  
  
  
</section>  
  
<script id="message-template" type="text/html">  
 <div class="alert message {{bgClass}}">  
 <div>  
 <span class="message-name">{{username}}</span>  
 <span class="message-meta">{{createdAt}}</span>  
 </div>  
 <div>{{message}}</div>  
 </div>  
</script>  
  
<script id="private-message-template" type="text/html">  
 <div class="alert message {{bgClass}}">  
 <div>  
 <span class="message-name">From {{username}}, To {{to}} (private)</span>  
 <span class="message-meta">{{createdAt}}</span>  
 </div>  
 <div>{{message}}</div>  
 </div>  
</script>  
  
<script id="sidebar-template" type="text/html">  
 <div class="alert alert-primary">  
 Room: <strong>{{room}}</strong><br>  
 Username: <strong>{{currentUser}}</strong>  
 </div>  
  
 <ul class="list-group" id="list-group">  
 <li class="list-group-item list-group-item-primary">Users in the chatroom</li>  
 {{#users}}  
 <li class="list-group-item user-part" onclick="active(this)">{{username}}</li>  
 {{/users}}  
 </ul>  
</script>  
  
<script id="typing-message-template" type="text/html">  
 <ul class="list-group list-group-horizontal">  
 {{#users}}  
 <li class="list-group-item list-group-item-dark">{{username}} is typing...</li>  
 {{/users}}  
 </ul>  
</script>  
  
<script src="https://cdnjs.cloudflare.com/ajax/libs/mustache.js/3.1.0/mustache.min.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.24.0/moment-with-locales.min.js"></script>  
<script src="https://cdnjs.cloudflare.com/ajax/libs/qs/6.9.1/qs.min.js"></script>  
  
<script src="/socket.io/socket.io.js"></script>  
<script>  
 //elements  
 const ***$messageForm*** = ***document***.querySelector("#message-form")  
 const $messageFormInput = $messageForm.querySelector("input");  
 const $messageFormButton = document.querySelector("button")  
 const $messages = document.querySelector("#messages")  
 const $sidebar = document.querySelector("#sidebar")  
 const $typingMessage = document.querySelector("#typing-message")  
 const $usersList = document.getElementsByClassName('user-part')  
  
 //Templates  
 const messageTemplate = document.querySelector("#message-template").innerHTML  
 const privateMessageTemplate = document.querySelector("#private-message-template").innerHTML  
 const sidebarTemplate = document.querySelector('#sidebar-template').innerHTML  
 const typingMessageTemplate = document.querySelector('#typing-message-template').innerHTML  
  
 let typing = false;  
 let timeout;  
 let privateUsername = null;  
 const {username, room} = Qs.parse(location.search, {ignoreQueryPrefix: true})  
 let socket = io();  
 socket.emit('join', {username, room}, (error) => {  
 if (error) {  
 alert(error);  
 location.href = '/';  
 }  
 });  
  
 socket.on('message', (message) => {  
 const html = Mustache.render(messageTemplate, {username: message.username, message: message.text, createdAt: moment(message.createdAt).format("h:mm a"), bgClass: 'alert-info'});  
 $messages.innerHTML += html;  
 });  
  
 socket.on('sendMessage', (message) => {  
 const html = Mustache.render(messageTemplate, {username: message.username, message: message.text, createdAt: moment(message.createdAt).format("h:mm a"), bgClass: message.bgClass});  
 $messages.innerHTML += html;  
 });  
  
 socket.on('sendPrivateMessage', (message) => {  
 if(message.username == username || message.to == username) {  
 console.log(username, message.to, message.from)  
 const html = Mustache.render(privateMessageTemplate, {username: message.username, to: message.to, message: message.text, createdAt: moment(message.createdAt).format("h:mm a"), bgClass: 'alert-danger'});  
 $messages.innerHTML += html;  
 }  
 });  
  
 socket.on('roomData', (users) => {  
 users.forEach(user => {  
 if (user.username == username)  
 user.username = username + ' (Це ви)';  
 })  
 let html = Mustache.render(sidebarTemplate, {room: room, currentUser: username, users: users});  
 $sidebar.innerHTML = html  
 });  
  
 socket.on('display', (typingUsers) => {  
 let html = Mustache.render(typingMessageTemplate, {users: typingUsers});  
 $typingMessage.innerHTML = html  
 });  
  
 $messageFormButton.addEventListener('click', (e) => {  
 e.preventDefault();  
 if ($messageFormInput.value && privateUsername)  
 socket.emit('sendPrivateMessage', $messageFormInput.value, privateUsername);  
 else if ($messageFormInput.value)  
 socket.emit('sendMessage', $messageFormInput.value);  
 $messageFormInput.value = '';  
 });  
  
 function typingTimeout(){  
 typing = false;  
 socket.emit('typing', {user:username, room: room, typing:false});  
 }  
  
 async function active(elem) {  
 console.log(username)  
 if (elem.innerHTML.slice(0, -8) != username.trim().toLowerCase()) {  
 elem.classList.toggle('active')  
 if (elem.classList.contains('active'))  
 privateUsername = elem.innerHTML  
 else  
 privateUsername = null  
 for (let user of $usersList) {  
 if (user != elem) {  
 user.classList.remove('active');  
 user.classList.remove('disabled');  
 }  
 }  
 }  
 }  
 $messageFormInput.addEventListener('keypress', () => {  
 typing = true;  
 socket.emit('typing', {user:username, room:room, typing:true});  
 clearTimeout(timeout);  
 timeout = setTimeout(typingTimeout, 2000);  
 });  
</script>  
</body>  
</html>

**style.css**

.form-container button {  
 width: 100%;  
}  
  
/\* General Styles \*/  
  
\* {  
 margin: 0;  
 padding: 0;  
 box-sizing: border-box;  
}  
  
html {  
 font-size: 16px;  
}  
  
input {  
 font-size: 14px;  
}  
  
body {  
 line-height: 1.4;  
 color: #333333;  
 font-family: Helvetica, Arial, sans-serif;  
}  
  
h1 {  
 margin-bottom: 16px;  
}  
  
label {  
 display: block;  
 font-size: 14px;  
 margin-bottom: 8px;  
 color: #777;  
}  
  
input {  
 border: 1px solid #eeeeee;  
 padding: 12px;  
 outline: none;  
}  
  
button {  
 cursor: pointer;  
 padding: 12px;  
 background: #7C5CBF;  
 border: none;  
 color: white;  
 font-size: 16px;  
 transition: background .3s ease;  
}  
  
button:hover {  
 background: #6b47b8;  
}  
  
button:disabled {  
 cursor: default;  
 background: #7c5cbf94;  
}  
  
/\* Join Page Styles \*/  
  
.centered-form {  
 background: #333744;  
 width: 100vw;  
 height: 100vh;  
 display: flex;  
 justify-content: center;  
 align-items: center;  
}  
  
.centered-form\_\_box {  
 box-shadow: 0px 0px 17px 1px #1D1F26;  
 background: #F7F7FA;  
 padding: 24px;  
 width: 250px;  
}  
  
.centered-form button {  
 width: 100%;  
}  
  
.centered-form input {  
 margin-bottom: 16px;  
 width: 100%;  
}  
  
/\* Chat Page Layout \*/  
  
.chat {  
 display: flex;  
}  
  
.chat\_\_sidebar {  
 height: 100vh;  
 color: white;  
 background: #333744;  
 width: 225px;  
 overflow-y: scroll  
}  
  
/\* Chat styles \*/  
  
.chat\_\_main {  
 flex-grow: 1;  
 display: flex;  
 flex-direction: column;  
 max-height: 100vh;  
}  
  
.chat\_\_messages {  
 flex-grow: 1;  
 padding: 24px 24px 0 24px;  
 overflow-y: scroll;  
}  
  
/\* Message Styles \*/  
  
.message {  
 margin: 8px;  
}  
  
.message-name {  
 font-weight: 600;  
 font-size: 14px;  
 margin-right: 8px;  
}  
  
.message-meta {  
 font-size: 14px;  
}  
  
.message a {  
 color: #0070CC;  
}  
  
/\* Message Composition Styles \*/  
  
#compose input {  
 border: 1px solid #eeeeee;  
 width: 100%;  
 padding: 12px;  
 margin: 0 16px 0 0;  
 flex-grow: 1;  
}  
  
#compose button {  
 font-size: 14px;  
 width: 150px;  
}  
  
/\* Chat Sidebar Styles \*/  
  
.room-title {  
 font-weight: 400;  
 font-size: 22px;  
 background: #2c2f3a;  
 padding: 24px;  
}  
  
.list-title {  
 font-weight: 500;  
 font-size: 18px;  
 margin-bottom: 4px;  
 padding: 12px 24px 0 24px;  
}  
  
.users {  
 list-style-type: none;  
 font-weight: 300;  
 padding: 12px 24px 0 24px;  
}  
#messages {  
 overflow: auto;  
}

**index.js**

const express = require('express');  
const app = express();  
const server = require('http').createServer(app);  
const io = require('socket.io')(server);  
const {addUser, removeUser, getUser, getUsersInRoom, addTypingUser, getTypingUsersInRoom, removeTypingUser} = require('./utils/users')  
const {generateMessage, generateLocationMessage} = require('./utils/messages')  
  
app.use(express.static(\_\_dirname + `/../public`))  
  
io.on("connection", (socket) => {  
 console.log("New user connected");  
 socket.on('join', (options, callback) => {  
 const {error, user} = addUser({id: socket.id, username: options.username, room: options.room});  
 if (error) {  
 return callback(error);  
 }  
 socket.join(user.room);  
 io.to(user.room).emit('roomData', getUsersInRoom(user.room));  
 socket.emit('message', generateMessage('Admin', `Well Cum, ${user.username}`));  
 socket.broadcast.to(user.room).emit('message', generateMessage('Admin', `${user.username} has joined`));  
 callback()  
 });  
 socket.on('sendMessage', (message) => {  
 let user = getUser(socket.id);  
 io.to(user.room).emit('sendMessage', generateMessage(user.username, message, user.bgClass));  
 });  
  
 socket.on('sendPrivateMessage', (message, to) => {  
 let user = getUser(socket.id);  
 console.log(message, to)  
 io.to(user.room).emit('sendPrivateMessage', generateLocationMessage(user.username, message, to, 'alert-danger'));  
 });  
  
 socket.on('typing', (data) => {  
 if(data.typing)  
 addTypingUser(socket.id);  
 else  
 removeTypingUser(socket.id);  
 io.to(data.room).emit('display', getTypingUsersInRoom(data.room));  
 })  
  
 socket.on('disconnect', () => {  
 let user = getUser(socket.id);  
 removeUser(socket.id);  
 console.log('a user disconnected');  
 if (user) {  
 socket.broadcast.to(user.room).emit('message', generateMessage('Admin', `${user.username} has disconnected`));  
 io.to(user.room).emit('roomData', getUsersInRoom(user.room));  
 }  
 });  
});  
  
  
server.listen(3000, function () {  
 console.log('listening on 3000 port');  
});

**messages.js**

const generateMessage = (username, text, bgClass) => {  
 return {  
 username,  
 text,  
 bgClass,  
 createdAt: new ***Date***().getTime()  
 }  
}  
  
const generateLocationMessage = (username, text, to, bgClass) => {  
 return {  
 username,  
 text,  
 to,  
 bgClass,  
 createdAt: new ***Date***().getTime()  
 }  
}  
  
module.exports = {  
 generateMessage,  
 generateLocationMessage  
}

**users.js**

const users = [];  
let typingUsers = [];  
  
const addUser = ({ id, username, room}) => {  
 // Clean the data  
 username = username.trim().toLowerCase()  
 room = room.trim().toLowerCase()  
  
 // Validate the data  
 if (!username || !room) {  
 return {  
 error: 'Username and room are required!'  
 }  
 }  
  
 // Check for existing user  
 const existingUser = users.find((user) => {  
 return user.***room*** === room && user.username === username  
 })  
  
 // Validate username  
 if (existingUser) {  
 return {  
 error: 'Username is in use!'  
 }  
 }  
  
 // Store user  
 let backgrounds = [  
 'alert-primary',  
 'alert-secondary',  
 'alert-success',  
 'alert-warning',  
 ];  
 let bgClass = backgrounds[***Math***.floor(***Math***.random()\*backgrounds.length)];  
 const user = { id, username, room, bgClass}  
 users.push(user)  
 return { user }  
}  
const removeUser = (id) => {  
 const index = users.findIndex((user) => user.id === id)  
  
 if (index !== -1) {  
 return users.splice(index, 1)[0]  
 }  
}  
  
const getUser = (id) => {  
 return users.find((user) => user.id === id)  
}  
  
const getUsersInRoom = (room) => {  
 room = room.trim().toLowerCase()  
 return users.filter((user) => user.***room*** === room)  
}  
  
const addTypingUser = (id) => {  
 let user = getUser(id);  
 // Check for existing user  
 const isTypingUser = typingUsers.find((typingUser) => {  
 return typingUser.id === id  
 })  
 // Validate username  
 if (!isTypingUser) {  
 typingUsers.push(user)  
 }  
}  
  
const removeTypingUser = (id) => {  
 const index = typingUsers.findIndex((user) => user.id === id);  
  
 if (index !== -1) {  
 return typingUsers.splice(index, 1)[0];  
 }  
}  
  
const getTypingUsersInRoom = (room) => {  
 room = room.trim().toLowerCase();  
 return typingUsers.filter((user) => user.***room*** === room);  
}  
  
module.exports = {  
 addUser,  
 removeUser,  
 getUser,  
 getUsersInRoom,  
 addTypingUser,  
 removeTypingUser,  
 getTypingUsersInRoom  
}

**Результат виконання завдання:**

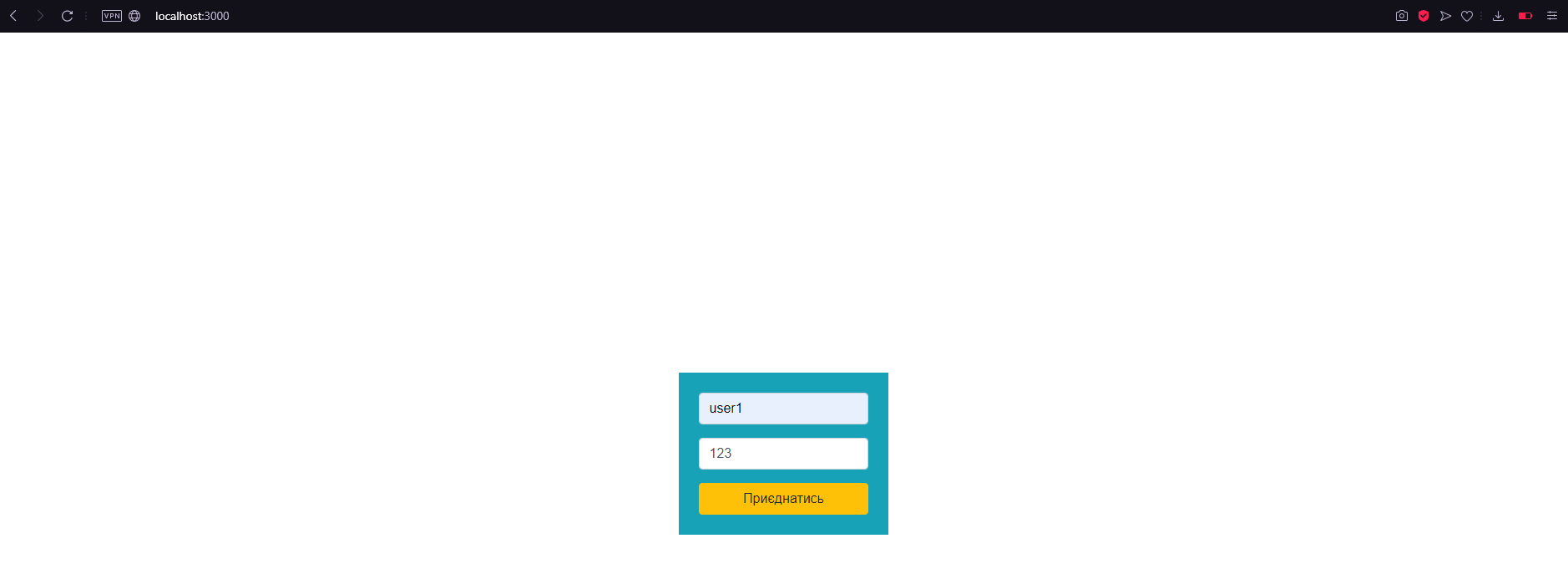
****

Рис. 1. Сторінка входу у кімнату

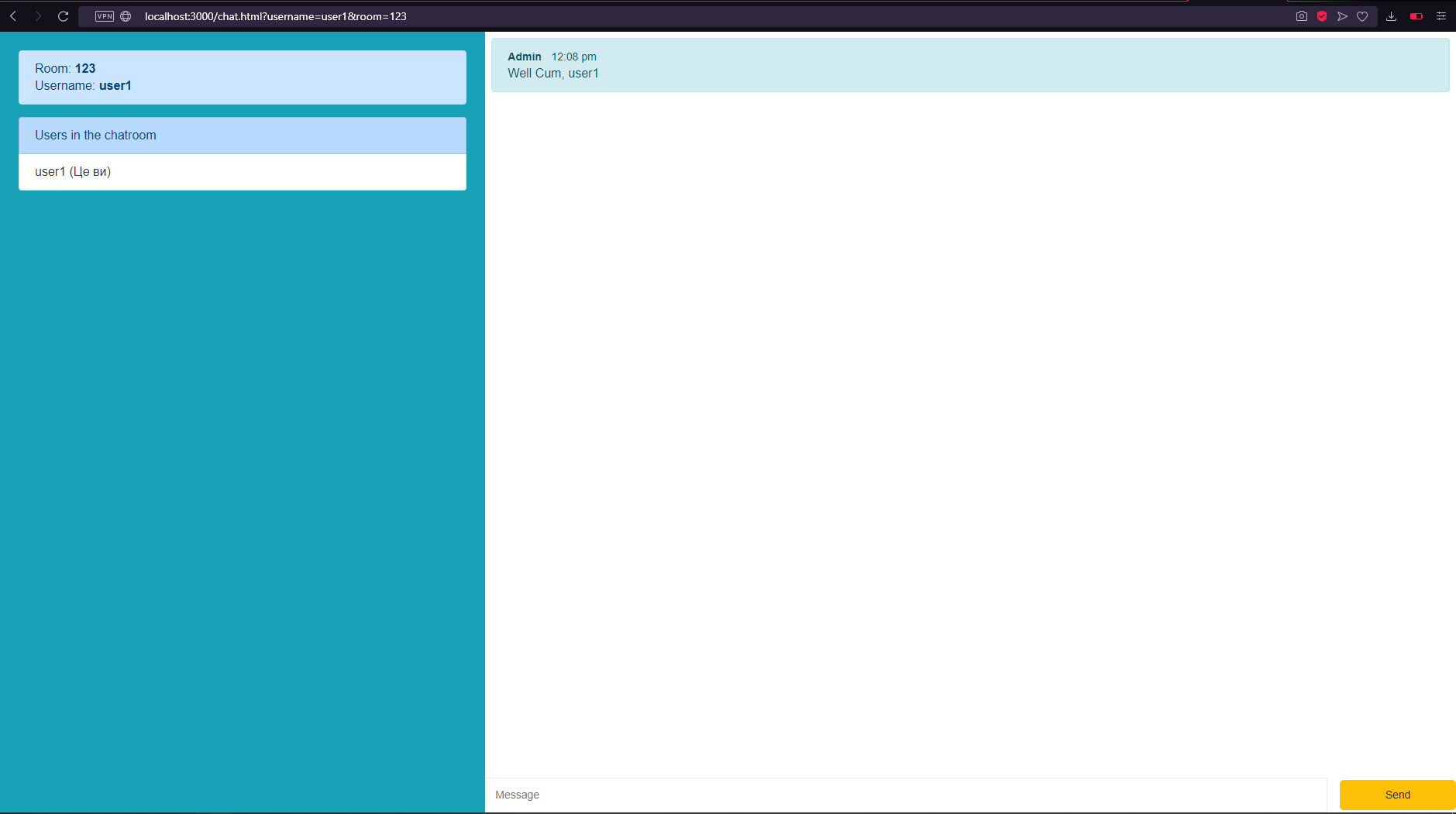


Рис. 2. Привітання користувачу

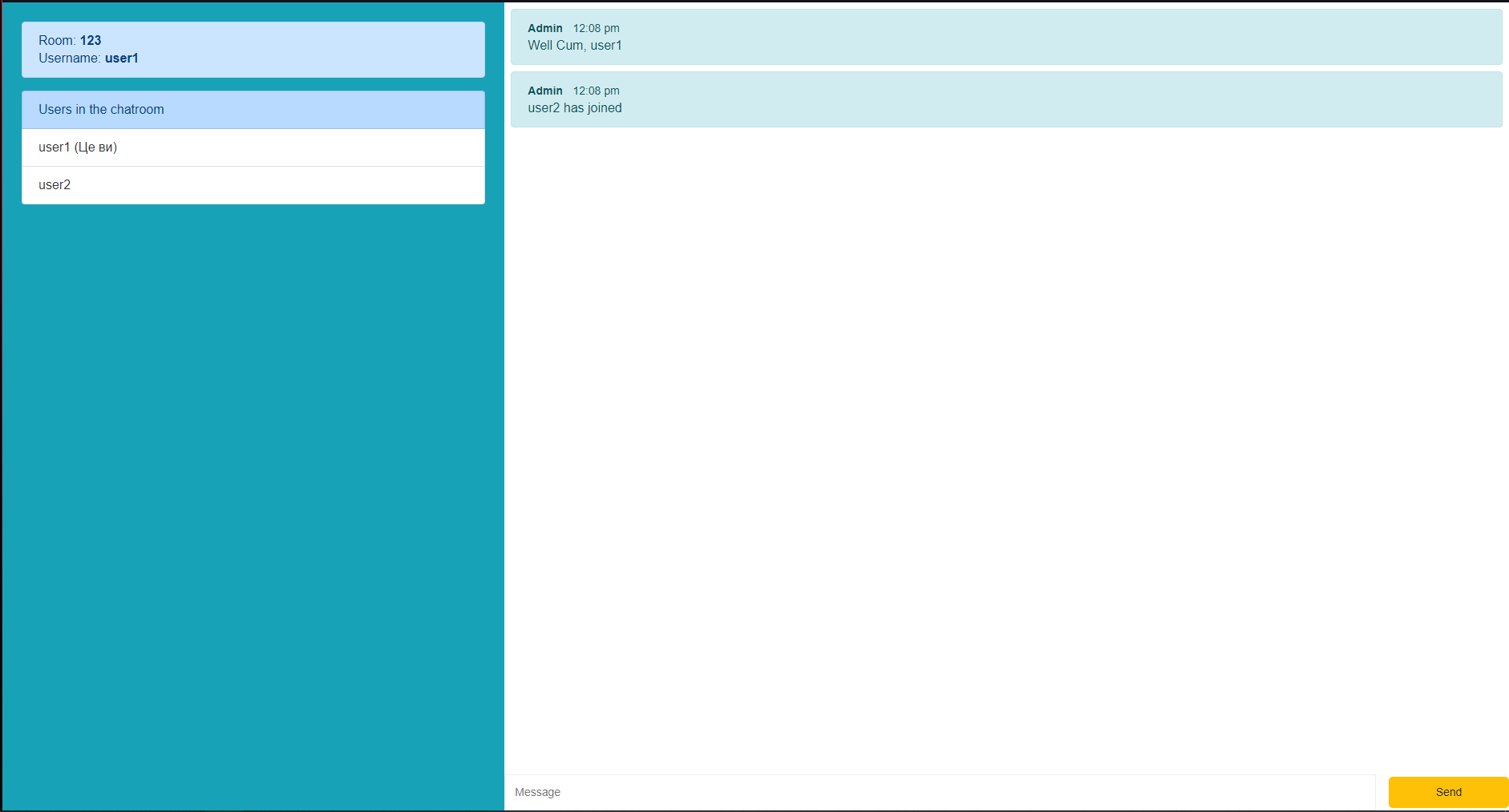


Рис. 3. Повідомлення про приєднання іншого користувача

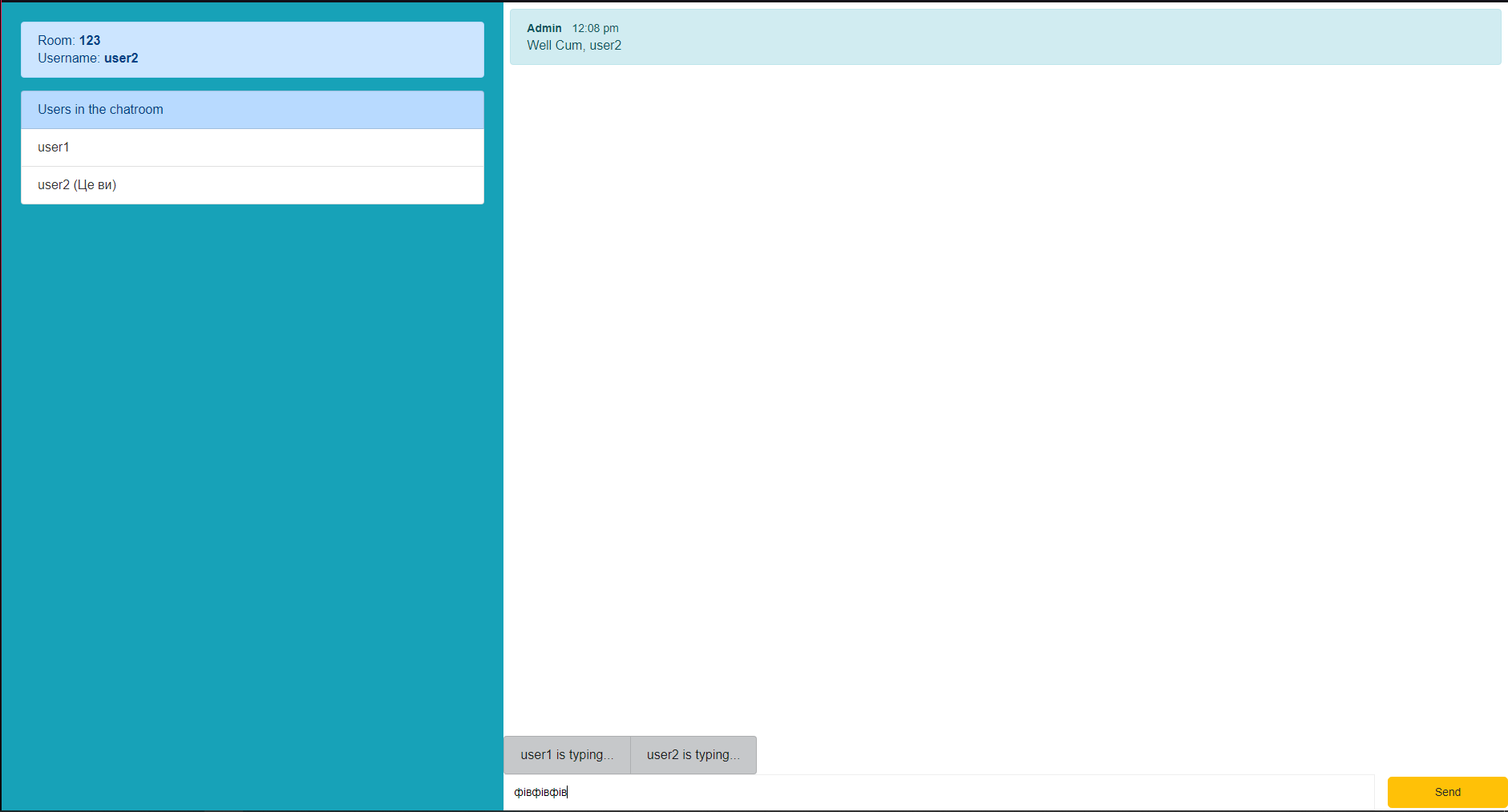


Рис. 4. Повідомлення, якщо користувач(користувачі) набирають повідомлення

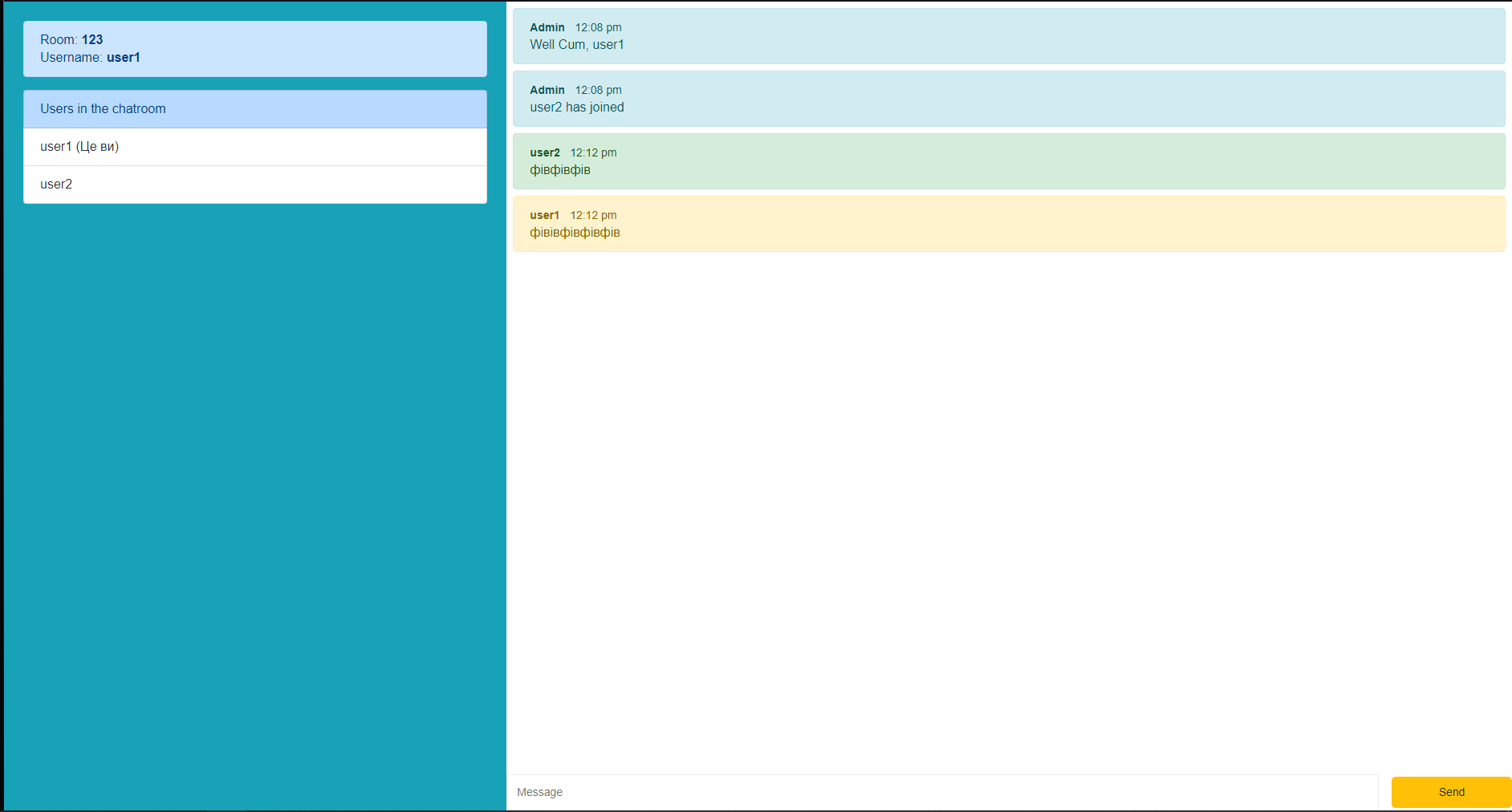


Рис. 5. Відображення надісланих повідомлень

Для відправлення приватного повідомлення, потрібно обрати користувача зі списку зліва

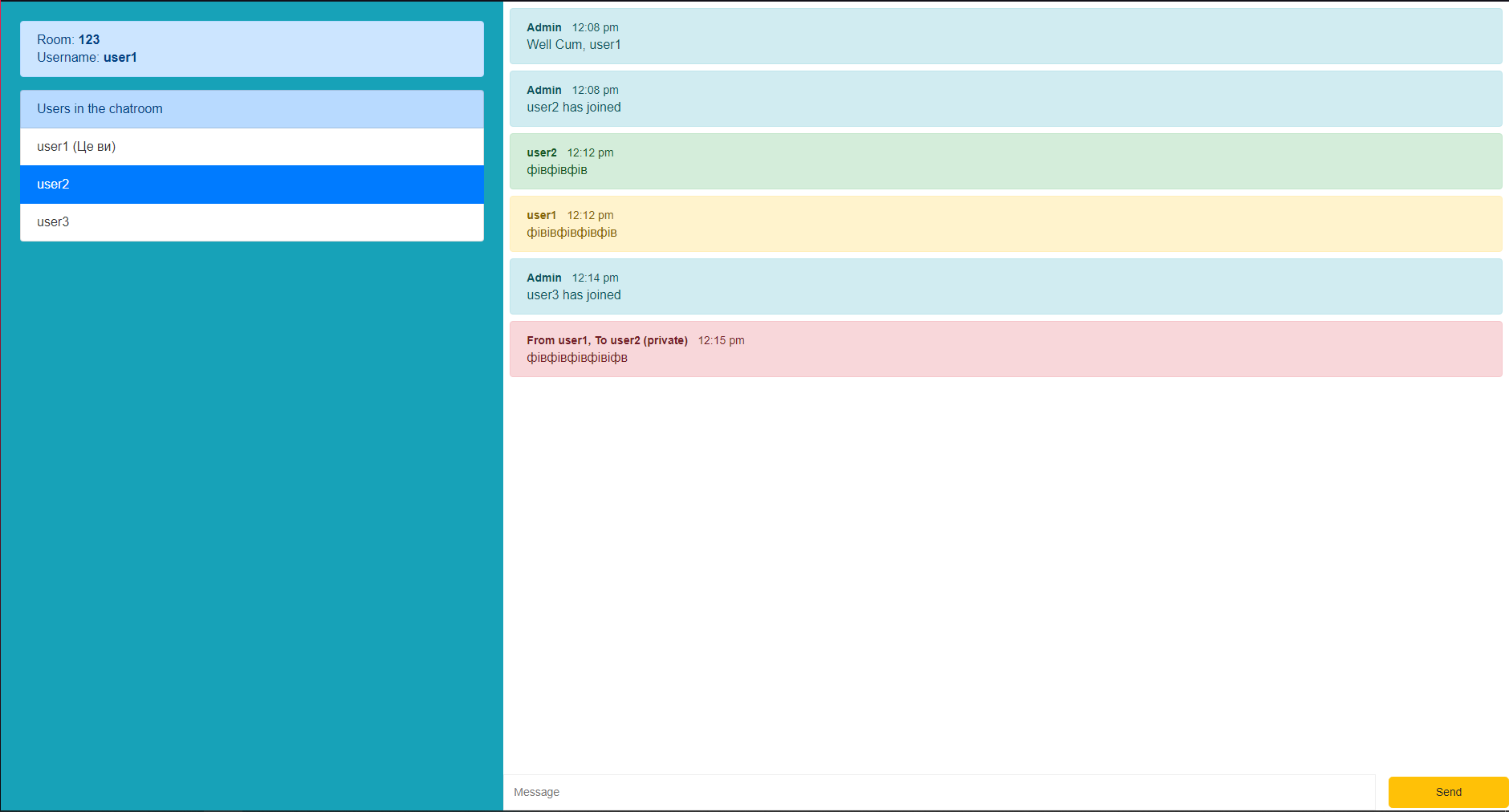


Рис. 6. Відправлення приватного повідомлення

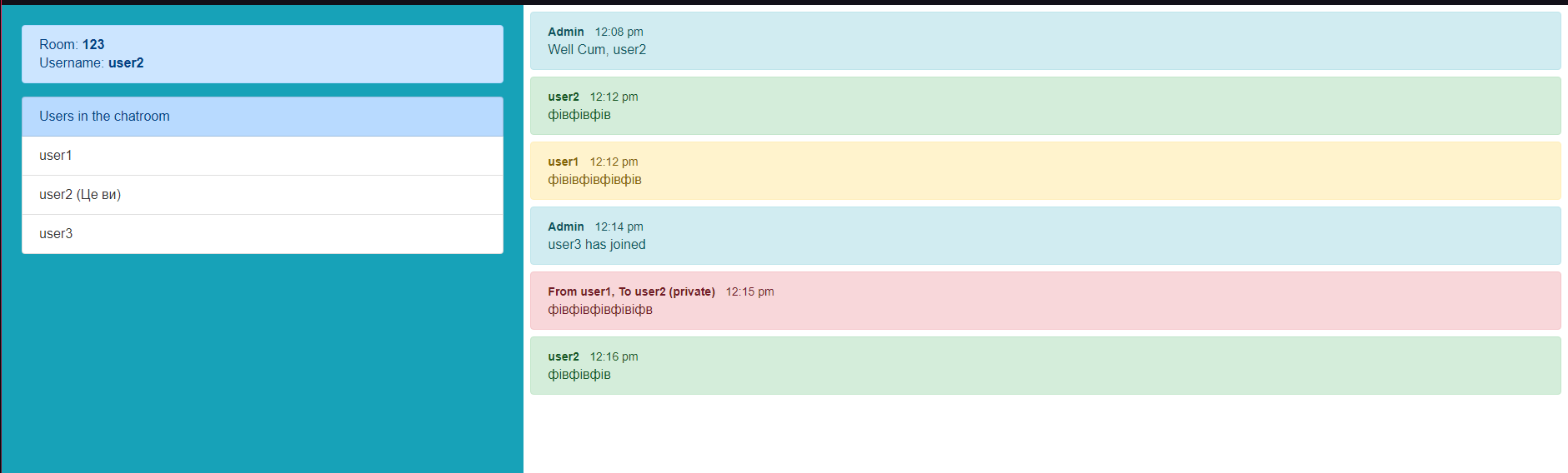


Рис. 7. Приймач бачить повідомлення

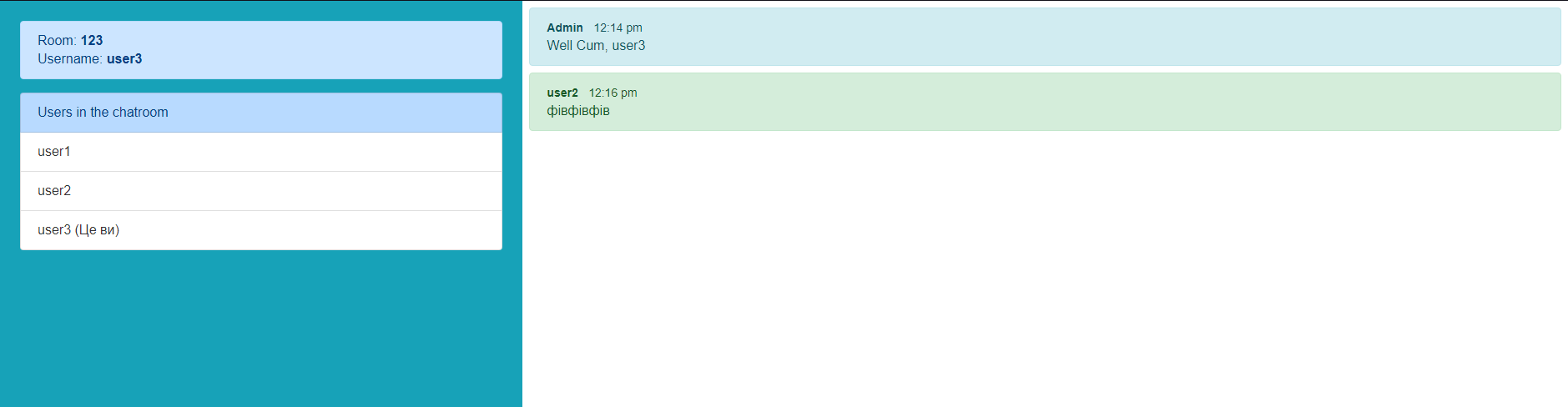


Рис. 8. Інші користувачі не бачять приватне повідомлення

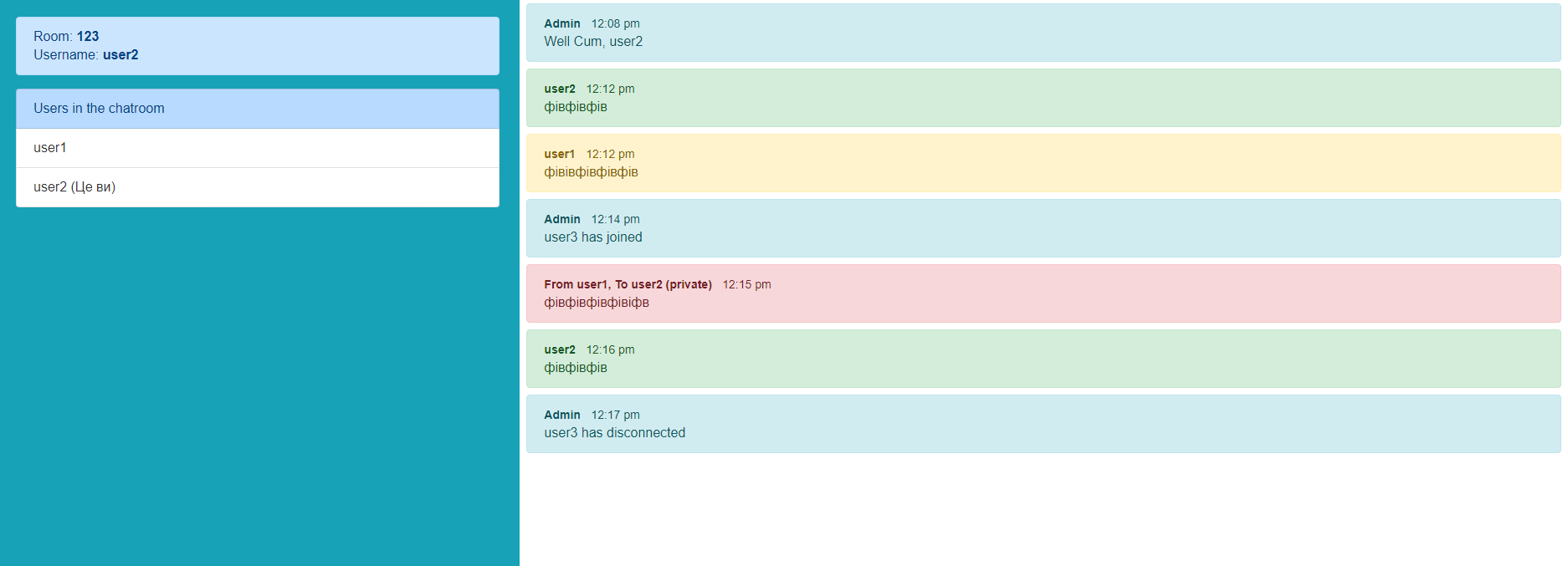


Рис. 9. Повідомлення при виході користувача

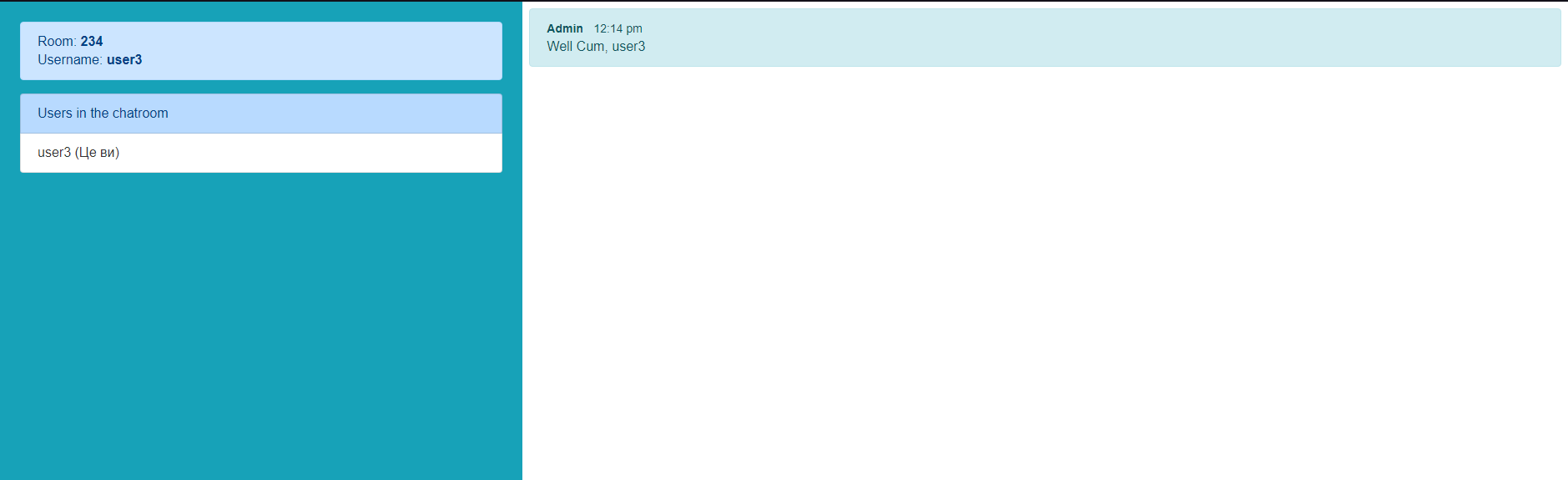


Рис. 10. Користувач в іншій кімнаті не бачить повідомлення з іншої кімнати

***Висновки:*** в ході виконання лабораторної роботи було отримано базові навички для роботи з протоколом WebSocket. Використано Socket.io для розробки чат-додатку