Socket.IO

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Introduction

2. Utilisation des sockets
   - Installation
   - Préparation du serveur
   - Préparation du client
   - Envoi de message : du client au serveur
   - Réception de message chez le serveur
   - Diffusion de message : du serveur à tous les clients
   - Réception de message chez le client
   - Broadcasting : du serveur à tous les clients sauf l'émetteur initial

Plan

3. Données sur les clients
   - Nombre de clients connectés
   - Identifiant du client connecté
   - IP du client connecté

4. Multicasting avec les rooms

Express.js

Socket.io

- Bibliothèque **JavaScript**
- Permettant la communication en temps réel entre les clients et le serveur web
- Fournissant une connexion bidirectionnelle persistante entre le client et le serveur

© Ach

Express.js

Socket.io

- Bibliothèque **JavaScript**
- Permettant la communication en temps réel entre les clients et le serveur web
- Fournissant une connexion bidirectionnelle persistante entre le client et le serveur

© Ach

Remarque

Ne pas confondre **Socket.io** et **WebSocket**.

Express.js

Socket.io vs **WebSocket**

- **WebSocket** : protocole de communication client/serveur, bidirectionnel et basé sur **TCP**.
- **Socket.io** : bibliothèque utilisant **WebSocket** comme l'un de ses moyens de transport, ou un autre si **WebSocket** n'est pas disponible.
- **WebSocket** : supporté par la plupart des navigateurs modernes mais pas les anciens.
- **Socket.io** : compatible avec un large éventail de navigateurs et offrant une rétrocompatibilité avec les anciens navigateurs.

Express.js

Démarche

- Créez un répertoire `cours-socket` dans votre espace de travail
- Lancez **VSC** et allez dans `File > Open Folder...` et choisissez `cours-socket`
- Dans `cours-socket`, créez un fichier `app.js`

Express.js

Initiation d'un projet `Node.js`

```
npm init -y
```

© Achref EL MOUELHI ©

Express.js

Initiation d'un projet `Node.js`

```
npm init -y
```

Installation de `Express.js`

```
npm i express
```

© AOUJELHI ©

Express.js

Initiation d'un projet `Node.js`

```
npm init -y
```

Installation de `Express.js`

```
npm i express
```

© AOUJELHI ©

Lancement du projet

```
nodemon .
```

Express.js

Créons `index.html` **dans** `client`

```
<!DOCTYPE html>
<html lang="fr">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale
      =1.0">
    <title>Socket.io</title>
</head>

<body>
    <h1>Socket.io</h1>
    <ul id="messages"></ul>
    <form id="form">
        <input id="input" autocomplete="off">
        <button>Envoyer</button>
    </form>
</body>

</html>
```

Express.js

Préparons aussi notre `app.js`

```
const express = require('express');
const app = express();
const { join } = require('path');

app.use(express.static(join(__dirname, 'client')));

app.get('/', (req, res) => {
    res.sendFile(join(__dirname, 'client', 'index.html'));
});

app.listen(3000, () =>
    console.log('Adresse du serveur : http://localhost:3000')
);
```

Express.js

Préparons aussi notre `app.js`

```
const express = require('express');
const app = express();
const { join } = require('path');

app.use(express.static(join(__dirname, 'client')));

app.get('/', (req, res) => {
    res.sendFile(join(__dirname, 'client', 'index.html'));
});

app.listen(3000, () =>
    console.log('Adresse du serveur : http://localhost:3000')
);
```

Pour tester, allez à l'URL `localhost:3000` et vérifiez qu'elle est accessible.

Express.js

Installation de `Socket.IO`

```
npm install socket.io
```

© Achref EL MOUELHI

Express.js

Installation de `Socket.IO`

```
npm install socket.io
```

© Achref EL MOUELHI

Pour utiliser les sockets, il faut commencer par ajouter les imports suivants

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

Express.js

Pour créer une instance de `socker.io`, il nous faut une instance du serveur `HTTP`

```
const server = createServer(app);
```

© Achref EL MOU

Express.js

Pour créer une instance de `socker.io`, il nous faut une instance du serveur `HTTP`

```
const server = createServer(app);
```

Ainsi nous pourrons créer une instance de `socker.io`

```
const io = new Server(server);
```

© Achref EL MOU HI ©

Express.js

Définissons une fonction qui sera exécutée chaque fois qu'un nouveau client se connecte à notre serveur

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
});
```

© Achref EL M

Express.js

Définissons une fonction qui sera exécutée chaque fois qu'un nouveau client se connecte à notre serveur

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
});
```

© Achref EL M

Pour tester, allez à l'URL `localhost:3000` et vérifiez qu'elle est accessible.

Express.js

À la fin du `body` d'index.html, ajoutons le code suivant

```
<script src="./client-socket.js"></script>
```

© Achref EL MOUELHI ©

Express.js

À la fin du `body` d'index.html, ajoutons le code suivant

```
<script src="./client-socket.js"></script>
```

Créons donc `client-socket.js` dans `client` et établissons la connexion avec le serveur

```
<script src="/socket.io/socket.io.js"></script>
const socket = io();
```

© AC[illegible] EL MOUELHI ©

Express.js

À la fin du `body` d'index.html, ajoutons le code suivant

```
<script src="./client-socket.js"></script>
```

Créons donc `client-socket.js` dans `client` et établissons la connexion avec le serveur

```
<script src="/socket.io/socket.io.js"></script>
const socket = io();
```

© AC[illegible] EL MOUELHI ©

Vérifiez que le message `Un client vient de se connecter` s'affiche dans la console.

Pour envoyer une message du client au serveur, on utilise la méthode `emit` de l'objet `socket`

```
const socket = io();

const button = document.querySelector('button');
const input = document.querySelector('input');

button.addEventListener('click', (e) => {
  if (input.value) {
    socket.emit('new message', input.value);
    input.value = '';
  }
});
```

© Achref EL MOUEL

Pour envoyer une message du client au serveur, on utilise la méthode `emit` de l'objet `socket`

```
const socket = io();

const button = document.querySelector('button');
const input = document.querySelector('input');

button.addEventListener('click', (e) => {
  if (input.value) {
    socket.emit('new message', input.value);
    input.value = '';
  }
});
```

Explication

- `socket.emit` : permet d'émettre un évènement et un paramètre
- `new message` : le nom qu'on a donné à l'évènement (Il n'est pas prédéfini)

Pour envoyer une message du client au serveur, on utilise la méthode `emit` de l'objet `socket`

```
const socket = io();

const button = document.querySelector('button');
const input = document.querySelector('input');

button.addEventListener('click', (e) => {
  if (input.value) {
    socket.emit('new message', input.value);
    input.value = '';
  }
});
```

EL MOUL

Explication

- `socket.emit` : permet d'émettre un évènement et un paramètre
- `new message` : le nom qu'on a donné à l'évènement (Il n'est pas prédéfini)

Saisissez un message dans l'`input`, cliquez sur le bouton et vérifiez que le message disparaît.

Express.js

Remarque

Dans la fonction `emit`, on peut envoyer un nombre variable d'arguments

© Ach

Express.js

Pour récupérer les messages du client, on utilise la méthode `on` de l'objet `socket`

```
const io = new Server(server);
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
    });
});
```

© Achref EL MOUELHI

Express.js

Pour récupérer les messages du client, on utilise la méthode `on` de l'objet `socket`

```
const io = new Server(server);
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
    });
});
```

EL MOUELH

Explication

- `socket.on` : permet d'écouter l'événement `new message` sur la connexion du client.
- L'évènement utilisé à l'émission doit être le même que celui à la réception (à savoir `new message`)

Express.js

Pour récupérer les messages du client, on utilise la méthode `on` de l'objet `socket`

```
const io = new Server(server);
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
    });
});
```

Explication

- `socket.on` : permet d'écouter l'événement `new message` sur la connexion du client.
- L'évènement utilisé à l'émission doit être le même que celui à la réception (à savoir `new message`)

Envoyez un message et vérifiez qu'il s'affiche dans la console.

Express.js

Rappel

`socket.emit` permet d'émettre un événement à un seul client spécifique.

© Achref EL MOU

Express.js

Rappel

`socket.emit` permet d'émettre un événement à un seul client spécifique.

Remarque

Pour émettre un évènement à tous les clients connectés, on peut utiliser `io.emit`

Express.js

Pour envoyer une message aux clients connectés, on utilise la méthode `emit` de l'objet `io`

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        io.emit(msg);
    });
});
```

Express.js

Pour récupérer les messages du serveur, on utilise la méthode `on` de l'objet `socket`

```
const messages = document.getElementById('messages');

socket.on('new message', (msg) => {
    const item = document.createElement('li');
    item.textContent = msg;
    messages.appendChild(item);
});
```

© Achref EL M

Express.js

Pour récupérer les messages du serveur, on utilise la méthode `on` de l'objet `socket`

```
const messages = document.getElementById('messages');

socket.on('new message', (msg) => {
    const item = document.createElement('li');
    item.textContent = msg;
    messages.appendChild(item);
});
```

Achref EL M

Pour tester

- Ouvrez la page `http://localhost:3000/` dans plusieurs onglets du navigateur.
- Envoyer un message depuis un onglet
- Vérifiez que le message s'affiche dans tous les onglets (y compris celui de l'émetteur)

Express.js

Résumé

- `socket.emit` permet d'émettre un événement à un seul client spécifique.
- `io.emit` permet d'émettre un évènement à tous les clients connectés.
- `socket.broadcast.emit` permet d'émettre un événement à tous les clients connectés, sauf à celui qui a initié l'événement.

Express.js

Pour envoyer une message aux clients sauf l'émetteur initial, on utilise `broadcast`

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message',msg);
    });
});
```

© Achref EL MO

Express.js

Pour envoyer une message aux clients sauf l'émetteur initial, on utilise `broadcast`

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message',msg);
    });
});
```

Achref EL MO

Pour tester

- Ouvrez la page `http://localhost:3000/` dans plusieurs onglets du navigateur (au moins 3).
- Envoyer un message depuis un onglet
- Vérifiez que le message s'affiche dans tous les onglets (sauf celui de l'émetteur)

Express.js

Pour récupérer le nombre de clients connectés

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
});
```

© Acme

Express.js

Pour récupérer le nombre de clients connectés

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
});
```

© Acme

Relancez le serveur, ouvrez des nouveaux onglets et vérifiez que le nombre s'incrémente.

Express.js

Et pour gérer la déconnexion

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
    socket.on('disconnect', () => {
        console.log(`#clients connectés après déconnexion : ${io.engine
           .clientsCount}`);
    });
});
```

Express.js

Et pour gérer la déconnexion

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter');
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
    socket.on('disconnect', () => {
        console.log(`#clients connectés après déconnexion : ${io.engine
           .clientsCount}`);
    });
});
```

Fermez quelques onglets et vérifiez que le nombre se décrémente.

Express.js

Et pour récupérer l'identifiant du client connecté

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter. Son ID : ' + socket.
       id);
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
    socket.on('disconnect', () => {
        console.log(`#clients connectés après déconnexion : ${io.engine
           .clientsCount}`);
    });
});
```

Express.js

Et pour récupérer l'IP du client connecté

```
io.on('connection', (socket) => {
    console.log('Un client vient de se connecter. Son ID : ' + socket.
       id);
    console.log('Adresse IP du client : ' + socket.handshake.address);
    console.log(`#clients connectés : ${io.engine.clientsCount}`);
    socket.on('new message', (msg) => {
        console.log('message: ' + msg);
        socket.broadcast.emit('new message', msg);
    });
    socket.on('disconnect', () => {
        console.log(`#clients connectés après déconnexion : ${io.engine
           .clientsCount}`);
    });
});
```

Express.js

Rooms

- Mécanisme permettant de regrouper des connexions/socket en un espace logique
- Facilitant :
  - la gestion des messages
  - la diffusion sélective de ces messages à un sous-ensemble spécifique des connexions

Express.js

Rooms

- Mécanisme permettant de regrouper des connexions/socket en un espace logique
- Facilitant :
  - la gestion des messages
  - la diffusion sélective de ces messages à un sous-ensemble spécifique des connexions

Cas d'utilisation

Organiser les utilisateurs en groupes dans les salons de discussion

Express.js

Pour créer une room

```
io.on('connection', (socket) => {
    socket.join('room1');
});
```

© Achref EL MOUELHI ©

Express.js

Pour créer une room

```
io.on('connection', (socket) => {
    socket.join('room1');
});
```

Pour envoyer un message à une room spécifique

```
io.to('room1').emit('évènement', 'Un message pour la room1');
```

© Achref EL MOUELHI ©

Express.js

Pour créer une room

```
io.on('connection', (socket) => {
    socket.join('room1');
});
```

Pour envoyer un message à une room spécifique

```
io.to('room1').emit('évènement', 'Un message pour la room1');
```

© Achref EL MOUELHI ©

Pour quitter une room

```
io.on('connection', (socket) => {
    socket.leave('room1');
});
```

Express.js

Pour lister les rooms d'une connexion

```
io.on('connection', (socket) => {
    console.log(socket.rooms);
});
```

© Achi HI ©