TD3 : Mise en place d'un serveur de chat temps réel basique, sous Debian

Objectif :

Maitriser l'utilisation de socket.io pour échanger des messages à temps réel entre clients et serveur node.

Prérequis

- Node.js : environnement d'exécution JavaScript open source et multiplateforme, conçu pour créer des applications réseau évolutives.
- Express : framework populaire dans node. C'est un framework d'application Web Node.js populaire, minimaliste et flexible, qui fournit un ensemble robuste de fonctionnalités pour les applications Web et mobiles.
- Socket.io : une bibliothèque qui permet d'établir une communication bidirectionnelle, à faible latence et en temps réel, entre serveur et clients, basée sur des événements. Il se base sur le protocole WebSocket et offre des fonctionnalités supplémentaires telle que la reconnexion automatique. Le protocole WebSocket permet d'ouvrir un canal de communication bidirectionnel sur un socket TCP pour navigateurs et serveurs web.
- nodemon : un outil qui aide à développer des applications basées sur Node.js, en redémarrant automatiquement l'application node lorsque des modifications dans les fichiers sources sont détectées.
- npm

Préparer votre environnement de travail

Installer npm
Installer nodejs
Créer un nouveau répertoire «SimpleSocketChat»
Se déplacer dans «SimpleSocketChat»
installer express
installer nodemon

Initialisation du projet Node.js

Créer un fichier package.json avec les informations de base sur notre application

Initialisation du code front-end (le client)

1. Créer un répertoire «public» à la racine de votre projet, dans lequel nous allons placer le code client.
2. Se déplacer dans «public».
3. Créer une page Web HTML «index.html» permettant de saisir et d'afficher des messages. «index.html» doit contenir un formulaire et une liste de messages.
   - ✓ Le formulaire est composé des éléments suivants :

- un champs de saisie
- un bouton

✓ La liste des messages est une liste html non ordonnée.

4. Ajouter une feuille de style «styleChat.css», permettant de mettre en forme, à votre goût, le contenu de la page «index.html».

Initialisation du serveur

1. Sortir du répertoire «public», pour se positionner dans le répertoire principal de votre projet.
2. Créer le fichier principal de votre application : « server.js ». Pour le moment, ce fichier permettra seulement d'écouter sur un port (3000) et de renvoyer le message «Mon premier chat avec socket.io» aux utilisateurs.

```
/* initialiser l'application avec le Framework Express et la bibliothèque http intégrée à node*/
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);

/* définir un gestionnaire de route qui est appelé lorsque on accède à l'accueil de votre site Web : cela permet de gérer les requêtes HTTP des utilisateurs connectés en leur renvoyant le message «Mon premier chat avec socket.io» */
app.get('/', (req, res) => {
  res.send('<h1> Mon premier chat avec socket.io </h1>');
});

/* lancer le serveur en écoutant les connexions arrivant sur le port 3000 */
server.listen(3000, () => {
  console.log('listening on *:3000');
});
```

3. Exécutez la commande «*node server.js*» depuis le dossier où se trouve votre application pour la démarrer.
4. Ouvrir un navigateur et saisir l'adresse http://localhost:3000. Ouvrir d'autres onglets avec la même adresse.
5. Arrêter votre serveur (vous pouvez utiliser les touches «ctrl + C») et redémarrer le en utilisant nodemon : *nodemon server.js*

Intégration de Socket.io

1. Socket.IO est composé de deux parties :
   - Une partie serveur qui s'intègre avec le serveur HTTP Node.JS

- Une bibliothèque cliente qui se charge côté navigateur socket.io-client

2. Exécuter la commande : *npm install socket.io*. Cela permettra d'installer socket.io et ajouter la dépendance à «*package.json*»
3. Initialiser une nouvelle instance de socket.io dans «*server.js*» :

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server); // la constante io, va nous permettre de travailler avec Socket.io

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

server.listen(3000, () => {
  console.log('listening on *:3000');
});
```

4. Dans le fichier index.html, ajouter le code js suivant juste avec la fermeture de la balise <body>

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

A noter qu'ici nous ne spécifions aucune URL dans l'appel de io() (URL à laquelle le client Socket.io doit se connecter), car par défaut il va tenter de se connecter sur le serveur qui héberge la page client.

Ajout de l'événement 'connection'

1. Nous allons ajouter un premier événement spécifique à Socket.io : l'événement «connection». A chaque fois qu'un utilisateur se connecte sur la page de notre site, l'événement est déclenché. Notre code va donc l'écouter et afficher un message dans la console à chaque déclenchement.

   Ajoutez le code suivant à server.js.

```
io.on('connection', function(socket){
  console.log('a user connected');
});
```

2. Ouvrir une page http://localhost:3000/ et observer la console. Ouvrir d'autres onglets http://localhost:3000/.

Ajout de l'événement 'disconnect'

1. Nous allons écouter l'événement disconnect qui est déclenché à chaque fois qu'un utilisateur se déconnecte de la socket sur laquelle il était connecté. Cet événement est rattaché à la socket d'un utilisateur en particulier et non au module Socket.io en général. C'est donc sur l'instance de l'objet socket (passé en argument à la fonction de callback de l'événement connection) qu'il faut écouter.

   Modifier le code de server.js :

```
io.on('connection', function(socket){
  console.log('a user connected');
  socket.on('disconnect', function(){
    console.log('user disconnected');
  });
});
```

2. Tester votre application : ouvrir et fermer (ou bien actualiser) des onglets http://localhost:3000/. Comme vous pouvez le constater, le serveur reçoit des événements du client en temps réel, comme par exemple ici la connexion et la déconnexion d'un utilisateur.

Emettre des événements par le client

Outre l'émission et la réception d'événements à temps réel entre client et serveur, l'idée principale derrière Socket.IO est que vous pouvez envoyer et recevoir tous les événements que vous souhaitez, avec les données que vous souhaitez. Nous allons tester cet échange de message.
Faites en sorte que lorsque l'utilisateur saisit un message, le serveur le reçois en tant qu'événement de message de chat. Modifier le script js inclus dans le fichier index.html (juste avant la balise <body> :

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

  var form = document.getElementById('form');
  var input = document.getElementById('input');
```

```
 form.addEventListener('submit', function(e) {
   e.preventDefault();
   if (input.value) {
     socket.emit('chat message', input.value);
     input.value = '';
   }
 });
</script>
```

Maintenant que l'événement est émis, il faut le réceptionner côté serveur. modifier le fichier server.js :

```
io.on('connection', (socket) => {
 socket.on('chat message', (msg) => {
   console.log('message: ' + msg);
 });
});
```

Tester votre application : ouvrir des onglets http://localhost:3000/, et écrire de nouveaux messages. Observez la console (sur votre terminal Debian).

Gestion des événements envoyés par le serveur

Pour l'instant, nos événements ont été émises par un client vers le serveur.
L'inverse est aussi important et possible avec socket.io.
Il existe trois types d'événements envoyés par le serveur :

- L'événement simple, envoyé au travers d'une seule socket (vers un seul utilisateur)
  Exemple :
  ```
  socket.emit('random-event', randomContent);
  ```
- le broadcast, envoyé à tout le monde sauf à la socket courante (c'est à dire sauf à l'utilisateur courant)
  Exemple :
  ```
  socket.broadcast.emit('random-event', randomContent);
  ```
- la combinaison des deux : l'émission d'un événement à tous les clients connectés au serveur
  Exemple :
  ```
  io.emit('random-event', randomContent);
  ```

Nous allons transmettre les messages émiss par le serveurs à tous les utilisateurs. Modifier server.js comme suit :

```
io.on('connection', (socket) => {
 socket.on('chat message', (msg) => {
   io.emit('chat message', msg);
```

```
  });
});
```

Voilà ce qui se passe maintenant avec notre application :
un client accède à la page (il se connecte ainsi au server via socket.io)
le client envoie un message et émet donc un événement « chat-message »
le serveur reçoit l'événement, ce qui déclenche l'émission d'un autre événement (ici on l'appelle chat-message, mais vous pouvez l'appeler autrement) qui sera envoyé à tous les utilisateurs connectés.

Nous allons ajouter le code permettant de réceptionner l'événement côté client et afficher le message reçu. Ajouter le code suivant à index.html.

```
socket.on('chat message', function(msg) {
   var item = document.createElement('li');
   item.textContent = msg;
   messages.appendChild(item);
   window.scrollTo(0, document.body.scrollHeight);
 });
```

Tester votre application.

Pour aller plus loin…

1. N'envoyez pas le même message à l'utilisateur qui l'a envoyé. Au lieu de cela, ajoutez le message directement dès qu'il appuie sur Entrée.
2. Diffusez un message aux utilisateurs connectés lorsque un autre utilisateur se connecte ou se déconnecte.
3. Ajoutez la prise en charge des login : un utilisateur doit se connecter avant de pouvoir accéder à l'envoi de messages.