NodeJS API - A4 - Communication Web Socket

Imaginez une conversation téléphonique, mais au lieu de parler, vous envoyez des messages écrits. C'est un peu comme ça que fonctionnent les WebSockets.

Au lieu d'attendre qu'une personne appelle, vous avez une connexion ouverte en permanence. Dès qu'il y a quelque chose de nouveau à partager, vous l'envoyez instantanément. C'est comme une conversation en temps réel, sans attendre que quelqu'un appelle ou envoie un message.

Nous allons donc mettre en place ce système de communication en temps réel.

01 - Mise en place

Pour commencer, nous allons attaquer le coté serveur. Il nous faudra installer un package

```
npm install ws
```

Créez un fichier `websocketserver.js` à la racine.

```
const WebSocket = require('ws');
// Lancement du serveur Web socket sur le port 8181
const wss = new WebSocket.Server({ port: 8181 });
wss.on('connection', function connection(client) {
    client.on('message', function incoming(message) {
        console.log('received: %s', message);
        client.send(`S: ${message}`);
    });
    client.send('This is a message');
});
exports.module = wss;
```

Dans ce code, nous avons :

1. L'instance du serveur WebSocket (indépendant du serveur express)
2. Nous gérons les connexions au serveur
3. Nous gérons la réception de l'évènement `message` de la part du client
4. Nous renvoyons un message en retour à la connexion.

Importez votre module dans le fichier `server.js`

Relancez le serveur. Puis, créez une requête WebSocket. Pointez l'url suivante `ws://localhost:8181`. En vous connectant, vous devriez voir le message du serveur, puis en envoyant un message, il devrait apparaitre dans le console NodeJS.

```
A l'heure où le sujet est rédigé, seul Postman semble prendre en compte
les websocket.
```

02 - Front-end

L'intérêt est aussi d'avoir un vrai client web et non juste un appel Postman. Nous allons donc construire un petit site web pour faire office de chat.

Dans un dossier `front` (Je vous recommande en dehors de votre dossier projet API), créez un fichier `index.html` puis un fichier `main.js`.

Pour aller au plus simple, voici le code html simple que je vous propose (et reste entièrement customisable)

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Chat App</title>
    <link
href="https://cdn.jsdelivr.net/npm/tailwindcss@2.0.1/dist/tailwind.min.c
ss" rel="stylesheet">
</head>
<body class="bg-gray-100">
    <div class="container mx-auto p-4">
        <h2 class="text-2xl font-bold mb-2">Chat</h2>
        <div id="messages" class="bg-white p-4 h-64 overflow-auto mb-4">
</div>
        <input type="text" id="messageInput" class="border p-2 w-full">
        <button id="sendButton" class="bg-blue-500 hover:bg-blue-700
text-white font-bold py-2 px-4 rounded">Envoyer</button>
    </div>
    <script src="main.js"></script>
</body>
</html>
```

Vous devriez avoir une interface de chat simplissime.

Passons maintenant au `main.js`. Je vous propose de récupérer dans des constante les différents éléments important du front.

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

Puis il est temps d'utiliser l'api de WebSocket implémenter par JS (donc aucune dépendance). L'adresse peut être différente.

```
const ws = new WebSocket(`ws://localhost:8181`);
```

Il nous reste à gérer les différents évènements du WebSocket . A vous de les implémenter

```
ws.onopen = () => {
    //Affiche un message comme quoi la connexion est réussie
};
ws.onmessage = (event) => {
    // Crée un div
    // récupérer les donnée de l'event pour le mettre en tant que
contenu du la div
    // Ajoute la div à messageS
};
ws.onerror = (error) => {
    //Affiche l'erreur dans la console
};
ws.onclose = () => {
    //Affiche un message de déconnexion dans la console
};
```

Si tout fonctionne, vu que notre serveur devrait renvoyer 'This is a message' à la connexion (pour peu que vous ne l'ayez pas supprimé), celui-ci devrait apparaitre dans le chat. Ouvre la console du navigateur, vérifier que votre message à la connexion s'affiche.

Tentez de déconnecter votre API, le message de déconnexion devrait apparaitre dans la console.

Il ne nous reste plus qu'a gérer le bouton d'envoi. A vous de jouer, la seule aide que je vous donne c'est que l'objet `ws` possède une méthode `send()`

Une fois que tout fonctionne, une fois le message envoyé, il devrait être reçu par le serveur, loggé dans VSCode, puis renvoyé avec un préfix `S: message`. Seul soucis, si vous tentez d'ouvrir 2 fois votre front, quand l'un envoi le message, le 2ème ne le reçois pas.

Daaaamn.... Mais non pas de soucis, il y a un moyen de parer à ce problème ! En effet, même si la librairie WebSocket ne permet pas la diffusion de masse (broadcast), elle garde une liste des clients connecté. Il nous suffit alors de boucler dessus pour envoyer le message à tous les clients ! (N'oubliez pas d'enlever le message que l'on envoi au client qui à envoyer le message initial !).

A vous de jouer. Une petite aide, le module `ws` possède un attribue client sur lequel on peut faire une boucle !

Vous devriez pouvoir maintenant échanger des messages entre les 2 fenêtres et celle-ci s'actualisent bien ! Testez même avec le client Postman, il fait aussi interface de chat !

03 - Aller plus loin

Vous êtes peut être séré niveau temps, donc je garde cette partie pour ceux qui ont l'envie et les moyens d'aller plus loin.

Actuellement, tous les messages ont le même préfixe, donc impossible de savoir qui nous a communiqué quoi que ce soit.

Sachez qu'il est possible de passer des paramètres à votre url de connexion au serveur WebSocket. Des query params pour être précis. Imaginez comme ceci :

```
ws://localhost:8181?token=hjtry456dgf5614
```

Nous pouvons donc récupérer le token soit à la connexion, soit à la requête. Faire valider celui-ci. S'il n'est pas valide, on refuse la requête ou bien nous déconnectons l'utilisateur. Si tout va bien, alors la requête est accepté / La connexion est validée.

Dans notre modèle actuel, nous n'avons pas de nom d'utilisateur mais seulement son mail. Il serait d'usage d'en avoir un. Mais pour l'exercice, vous pouvez vous contenter d'afficher la partie gauche du mail par exemple !

Vous pouvez gérer la récupération du token à l'aide d'une page email/password classique ou vous contenter de le récupérer à la main via Postman.

Je laisse une ressource en anglais pour vous aider dans ce challenge ! Bon courage et évidemment, certaines choses que l'on à faite diffèrent avec ce qui est montré sur ce site.

Auth over WebSockets