Communication avec socket.io

socket.io est l'une des bibliothèques les plus prisées par ceux qui développent avec Node.js (préalablement installé). Pourquoi ? Parce qu'elle permet de faire très simplement de la communication synchrone dans votre application, c'est-à-dire de la communication en temps réel ! socket.io se base sur plusieurs techniques différentes qui permettent la communication en temps réel (et qui pour certaines existent depuis des années !). La plus connue d'entre elles, et la plus récente, est WebSocket (API JavaScript). Elle permet un échange bilatéral synchrone entre le client et le serveur.

WebSocket est une nouveauté du Web qui permet de laisser une sorte de "tuyau" de communication ouvert entre le client et le serveur. Le navigateur et le serveur restent connectés entre eux et peuvent s'échanger des messages dans un sens comme dans l'autre dans ce tuyau. Désormais, le serveur peut donc lui-même décider d'envoyer un message au client comme un grand !

Pour installer socket.io, il faut exécuter la commande : « npm install socket.io »

Avant d'utiliser socket.io, il faut prendre le fichier socket.io.js dans le dossier socket.io-client (situé dans « node_modules » automatiquement fourni par le serveur node.js via le module socket.io) et le mettre dans un dossier (que vous créerez) nommé socket.io situé au même endroit que les fichiers serveur et client.
Il faudra l'inclure ainsi :

```
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
<script src="/socket.io/socket.io.js"></script>
```

Quand on utilise socket.io, on doit toujours s'occuper de deux fichiers en même temps :

- Le fichier serveur (ex : robot.js) : c'est lui qui centralise et gère les connexions des différents clients connectés au site.
- Le fichier client (ex : client.html) : c'est lui qui se connecte au serveur et qui affiche les résultats dans le navigateur.

Dans robot.js, on charge le serveur et on récupère et renvoie le contenu de la page index.html.

```
var http = require('http');
var fs = require('fs');

// Chargement du fichier index.html affiché au client
var server = http.createServer(function(req, res) {
    fs.readFile('./client.html', 'utf-8', function(error, content) {
        res.writeHead(200, {"Content-Type": "text/html"});
        res.end(content);
    });
});
```

Ce code renvoie le fichier client.html quand un client demande à charger la page dans son navigateur.
Ensuite, on charge socket.io et on gère les événements de socket.io :

```
// Chargement de socket.io
var io = require('socket.io').listen(server);

io.sockets.on('connection', function (socket) {
```

Ce code se prépare à recevoir des requêtes via socket.io. Ici, on s'attend à recevoir un seul type de message : la connexion. Lorsqu'on se connecte via socket.io, on logge ici l'information dans la console.

```
server.listen(PORT, console.log("Ecoute sur le port " + PORT + '...'));
```

Ouvrez le navigateur à l'adresse « localhost:8080 » ce qui va charger le fichier « client.html ».
Un code JavaScript se connecte au serveur via socket.io (et non http) avec les WebSockets.
Ainsi, le client se connecte au serveur en HTTP pour charger la page client.html et se connecte en temps réel avec le serveur avec les WebSockets grâce à socket.io.

```
ADRESSE_IP = "localhost";
PORT = 8080;
var socket = io.connect('http://'+ ADRESSE_IP + ':' + PORT);
```

Le fichier client.html est envoyé par le serveur nodejs.
Pour lancer l'application , il faut entre la commande : node robot.js et ensuite se rendre sur le navigateur Web à l'adresse (ici « localhost:8080 ») renseigné lors de la connexion du socket.
Maintenant, dès la connexion, le serveur pourrait envoyer un message au client :

```
// Quand un client se connecte, on lui envoie un message
socket.emit('message', 'Vous êtes connecté au serveur du robot !');
```

Et s'il y a plusieurs clients :

```
// On signale aux autres clients qu'il y a un nouveau venu
socket.broadcast.emit('message', 'Un autre client vient de se connecter au serveur du robot ! ');
```

Du côté du client, nous avons la réception du message :
La fonction alert (fonction js) sert à afficher une boîte de dialogue et un bouton.

```
// On affiche une boite de dialogue quand le serveur nous envoie un "message"
socket.on('message', function(message) {
    alert( message);
});
```

Remarque : Lorsque nous utilisons la fonction « emit » le 1er paramètre sert à donner un nom au message envoyé ce qui le différencie des autres fonctions « emit » et le 2e paramètre correspond au message envoyé.

Sinon, cela pourrait être le client qui envoie un message au serveur :
La fonction prompt (fonction js) est une boîte de dialogue qui permet d'afficher un message, un champ à remplir (input de type texte) et un bouton « OK » retournant la valeur qui a été entré dans le champ par l'utilisateur.

```
var solution = prompt('Pour quel méthode voulez-vous opter ?\n 1 => le client décide des valeurs d\'angles\n 2 =>
socket.emit('solution', solution);
```

Du côté du serveur, nous stockons le message afin que le serveur se rappelle de celui-ci :

```
socket.on('solution', function(solution){
    socket.solution = solution;
    if (solution == 1) {
        socket.on('angle1', function (angle1) {
            console.log(angle1);
        });
    }
    else if (solution == 2) {
        socket.on('angle2', function (angle2) {
            console.log(angle2);
        });
    }
    else {
    }

});
```

Dans la deuxième version, nous avons enlevé la possibilité au client d'envoyer des données au serveur. Ainsi, le serveur envoie 6 valeurs toutes les secondes grâce au code suivant :

Toutes les captures ci-dessous contiennent des commentaires qui expliquent directement le code.

Fonction qui envoie les valeurs aléatoires :

```
// On renvoie un nombre aléatoire entre une valeur min (incluse)
// et une valeur max (exclue)
function getRandomArbitrary(min, max) {
    return Math.floor(Math.random() * (max - min) + min);
}
```

Fonction qui exécute 6 fois cette fonction toutes les secondes :

```
var interval = setInterval(() => {
    //if (i === 10) clearInterval(interval);
    //socket.emit("angle2", "RESET");
    socket.emit("reset");
    for (let i = 0; i < 6; i++){
        var angle2 = getRandomArbitrary(0, 100);
        socket.emit('angle2', angle2);
    }
}, 1000);
```

La fonction SetInterval a pour 1er argument le code à exécuter et le 2e est le délai d'exécution.

D'autre part, côté serveur nous avons la possibilité d'envoyer au(x) client(s) le nombre de clients connectés à l'aide du code suivant :

```
if (io.engine.clientsCount > 1){ // on vérifie si le nombre de client connecté est supérieur à 1
    socket.emit('message', "Il y a " + io.engine.clientsCount + " clients actuellement connectés !");
    socket.broadcast.emit('message', "Il y a " + io.engine.clientsCount + " clients actuellement connectés !");
}
else {
    socket.emit('message', "Il y a " + io.engine.clientsCount + " client actuellement connecté !");
    socket.broadcast.emit('message', "Il y a " + io.engine.clientsCount + " client actuellement connecté !");
}
```

Du côté du client, nous avons les boutons (connexion, déconnexion, réception des données et arrêt de la réception) ainsi que la division où vont s'afficher les valeurs envoyées par le serveur :

```
<button onclick="receive()">Recevoir les données du serveur</button><br><br>
<button onclick="stop()">Arrêter la réception des données du serveur</button><br><br>
<button onclick="connect()">Se connecter au serveur</button><br><br>
<button onclick="disconnect()">Se déconnecter au serveur</button><br><br>
<!-- boucle for en twig sur le nbr d'axes dans la BDD -->
<div id="demo" class="display"></div>
```

Lorsqu'un client est connecté, en plus du message envoyé par le serveur, nous avons aussi un message qui s'affiche dans le division destinée aux valeurs du serveur :

```
document.getElementById("demo").innerHTML = "CONNECT !";
```

Et si le client est déconnecté, il y a la fonction « connect » :

```
function connect(){
    if (socket.connected) alert ("Vous êtes déjà connecté !"); // on vérifie si le client est déjà connecté
    else {
        let choix = confirm("Vous allez être connecté au serveur !");
        if (choix){
            socket.connect(); // on connecte le client
            document.getElementById("demo").innerHTML = "CONNECT !";
        }
        else alert ("Vous êtes toujours déconnecté !");
    }
    /*if (socket.on('reconnect')) {
        alert ("Vous êtes de retour !");
    }*/
}
```

Si le client est connecté, il y a la fonction « deconnect » :

```
function disconnect() {
    if (socket.connected) { // on vérifie si le client est déjà connecté
        if (on_socket >= 1) { // on vérifie si la réception est en cours...
                              // avec un compteur qui s'incrémente à chaque clic du bouton "receive"
            alert("Veuillez arrêter la réception avant de vous déconnecter !");
        }
        else {
            let choix = confirm("Vous allez être déconnecté du serveur !");
            if (choix) {
                socket.disconnect(); // on déconnecte le client
                alert("Vous êtes déconnecté du serveur du robot !");
                document.getElementById("demo").innerHTML = "DISCONNECT !";
            }
            else {
                alert ("Vous êtes toujours connecté !");

            }
        }
    }
    else alert ("Vous êtes déjà déconnecté !");
}
```

Si le client souhaite lancer la réception, il y a la fonction « receive » :

```
function receive(){
    //var solution = 2;
    //socket.emit('solution', solution);
    //alert(Object.keys(socket.on('angle2')).length);
    if (socket.connected) {
        alert("Vous allez recevoir des valeurs aléatoires d\'angles issues du serveur");
        document.getElementById("demo").innerHTML = "START !";
        /*showValue = function (angle2) {
            //on_event ++;
            socket.emit('angle2', angle2);
            if (angle2 === "RESET") {
                document.getElementById("demo").innerHTML  = "";
            }
            document.getElementById("demo").innerHTML += angle2 + '<br>';
            //function affiche() {}
            //setInterval(affiche, 1000);
        }*/
        showValue = function (angle2) { // subscriber qui va afficher les séries
            socket.emit('angle2', angle2);
            document.getElementById("demo").innerHTML += angle2 + '<br>'; // on affiche dans la division "demo"...
                                                          // toutes les séries de 6 valeurs envoyées par le serveur à la suite des autres
        }
        resetValue = function () { // subscriber qui va reset l'affichage de la division "demo"
            document.getElementById("demo").innerHTML = "";
        }
        socket.on('angle2', showValue);
        socket.on('reset', resetValue);
        on_socket++; // incrémentation du compteur
        if (on_socket > 1) { // si le compteur est supérieur à 1, c'est que plusieurs subscribers s'exécutent à même temps
            socket.removeListener("angle2", showValue); // on supprime un subscriber
            socket.removeListener("reset", resetValue); // on supprime un subscriber
        }
        /*socket.emit('message1', {1: 'Le serveur a choisi aléatoirement les valeurs d\'angles.',
                                 2: 'test solution 2'});*/
    }
    else alert("Vous ne pouvez pas recevoir de données !");
}
```

Dans cette capture, il y a 2 façons d'afficher les valeurs envoyés par le serveur mais une des 2 est en commentaires. En effet, la 1ere (en commentaire) fait le reset et le show en 1 seul subscriber alors que la 2e sépare les 2 subscribers ayant chacun leurs tâches.

Si le client souhaite arrêter la réception, il y a la fonction « stop » :

```
function stop() {
    if (socket.connected) { // on vérifie si le client est connecté
        if (on_socket === 0) { // on vérifie si la réception n'est pas en cours...
            alert("Aucune réception n'est en cours !");
        }
        else {
            on_socket = 0; // sinon on arrete la réception
            let choix = confirm("Vous allez arrêter la réception des données du serveur !");
            if (choix) {
                document.getElementById("demo").innerHTML = "STOP !";
                //socket.removeListener("angle2", subscriber);
                socket.removeListener("angle2", showValue); // on supprime un subscriber
                socket.removeListener("reset", resetValue); // on supprime un subscriber
                alert("Réception arrêtée !");
            }
            else {
                //alert(on_socket);
                on_socket++; // on relance la réception
                alert("Vous recevez toujours les données du serveur !");
            }
        }
    }
    else alert("Vous êtes déconnecté.\nIl n'y a donc aucune réception !");
}
```

Bibliographie :

https://socket.io/

https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1057825-socket-io-passez-au-temps-reel