Module socket.io de Node.js

Côté serveur

Initialisation

On crée un nouveau serveur Express :

```
var app = require('express')();
var server = require('http').createServer(app);
server.listen(8080);
```

On initialise une websocket (WS) qui va écouter sur le serveur :

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

Réception de messages

La fonction suivante est la base de la websocket. Elle se déclenche lors de l'initialisation du serveur :

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

On voit qu'elle prend une fonction en callback `function(socket) {...}` qu'on va compléter avec chacun des évènements qu'on veut écouter. Par exemple :

```
io.sockets.on('connection', function (socket) {
	socket.on('anEvent', function (data1) {...});
	socket.on('anAnotherEvent', function (data1) {...});
	socket.on('aThirdEvent', function (data1, data2) {...});
});
```

Ici, on a ici trois évènements. Chaque évènement possède la structure suivante :

```
socket.on('aThirdEvent', function (data1, data2) {...});
```

Le premier paramètre correspond au nom de l'évènement. Il est choisi arbitrairement, il faut juste garder les mêmes noms entre le serveur et le client.
Le deuxième paramètre est une fonction de callback appelée à chaque déclenchement d'un évènement. Elle prend autant de paramètres qu'on souhaite avec les noms qu'on veut (par exemple si on a décidé d'envoyer un évènement à deux paramètres, cette fonction de callback doit avoir deux paramètres, et on a choisi arbitrairement de les appeler data1 et data2). Elle contient le code de traitement de l'évènement. Par exemple :

```
socket.on('aThirdEvent', function (data1, data2) {
	console.log('data1: ' + data1);
	console.log('data2: ' + data2);
	var somme = data1 + data2;
	console.log('Somme: ' + somme);
});
```

D'où le code final de la websocket :

```
var app = require('express')();
var server = require('http').createServer(app);
server.listen(8080);
var io = require('socket.io').listen(server);

io.sockets.on('connection', function (socket) {
	socket.on('anEvent', function (data1) {
		// do something
	});
	socket.on('anAnotherEvent', function (data1) {
		// do something
	});
	socket.on('aThirdEvent', function (data1, data2) {
		console.log('data1: ' + data1);
		console.log('data2: ' + data2);
		var somme = data1 + data2;
		console.log('Somme: ' + somme);
	});
});
```

Émission de messages

Le serveur est capable de distinguer les clients qui se connectent. La librairie socket.io attribue un identifiant unique à chaque client, on n'a donc pas besoin de s'en occuper nous-même.

Pour émettre un évènement vers un client, on utilise le code suivant :

```
socket.emit('anEvent', data1, data2);
```

Pour émettre un évènement à tous les clients connectés, on utilise le code suivant :

```
socket.broadcast.emit('anEvent', data1, data2);
```

Gestion des groupes ("rooms")

On peut aussi créer des groupes (les "rooms") pour regrouper certains clients. Lors du traitement d'un évènement reçu d'un client, on utilise les codes suivants :

```
socket.join('newRoomName'); // Ajoute le client à une room
socket.leave('oldRoomName'); // Retire le client d'une room
```

Ainsi, si on veut émettre un évènement à tous les clients connectés à une "room", on utilise le code suivant :

```
socket.broadcast.to('roomName').emit('anEvent', data1, data2);
```

Côté client

Initialisation

Le code de socket.io étant écrit en JavaScript, on le place soit dans un fichier .js soit à l'intérieur d'une balise `<script>` chez le client.

Le fonctionnement des websockets est bidirectionnel. On retrouve donc des codes presque similaires chez le client et sur le serveur.

Dans le head de la page HTML du client, on rajoute une référence vers la librairie de socket.io :

```
<script src="/socket.io/socket.io.js"></script>
```

Puis on initialise la socket en lui passant en paramètre l'adresse du serveur :

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

Réception de messages

On utilise l'objet socket pour écouter et traiter les évènements qu'on reçoit :

```
socket.on('anEvent', function (data1, data2) {...});
```

Émission de messages

On utilise l'objet socket pour envoyer des évènements :

```
var ville = 'Nice';
var codepostal = '06000';
socket.emit('anOtherEvent', ville, codepostal);
```

On peut également transmettre des objets en les écrivant au format JSON :

```
socket.emit('anOtherEvent', 'Ville', { nom: 'Nice', codepostal: '06000' });
```