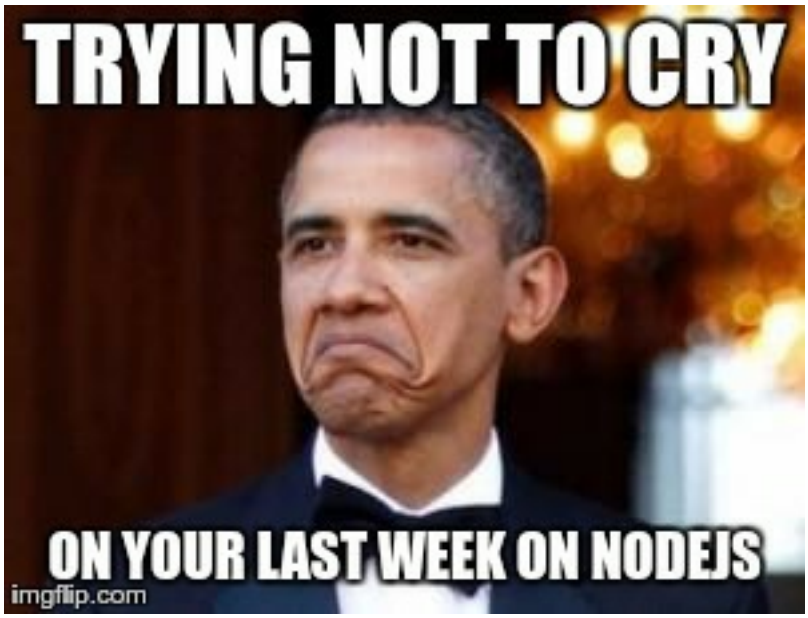
TP 8 : Socket.io et plugins



Introduction

Socket.io est une implémentation du protocole websocket au sein de node.JS. Comparé à une connexion HTTP standard, le protocole websocket effectue qu'une seule connexion entre le client et le serveur. Par la suite, les messages transitent grâce à des channels et des événements.

Le serveur peut communiquer à la fois avec un client propre ou bien aux clients inscrits à un channel spécifique ou bien à tous les clients (broadcast).

Pour bien comprendre l'utilisation et le fonctionnement de socket.io ainsi que la différence entre les événements individuels, de room ou en broadcast, vous trouverez, en suivant ce lien, un exemple de créaction d'un tchat : http://socket.io/get-started/chat/.

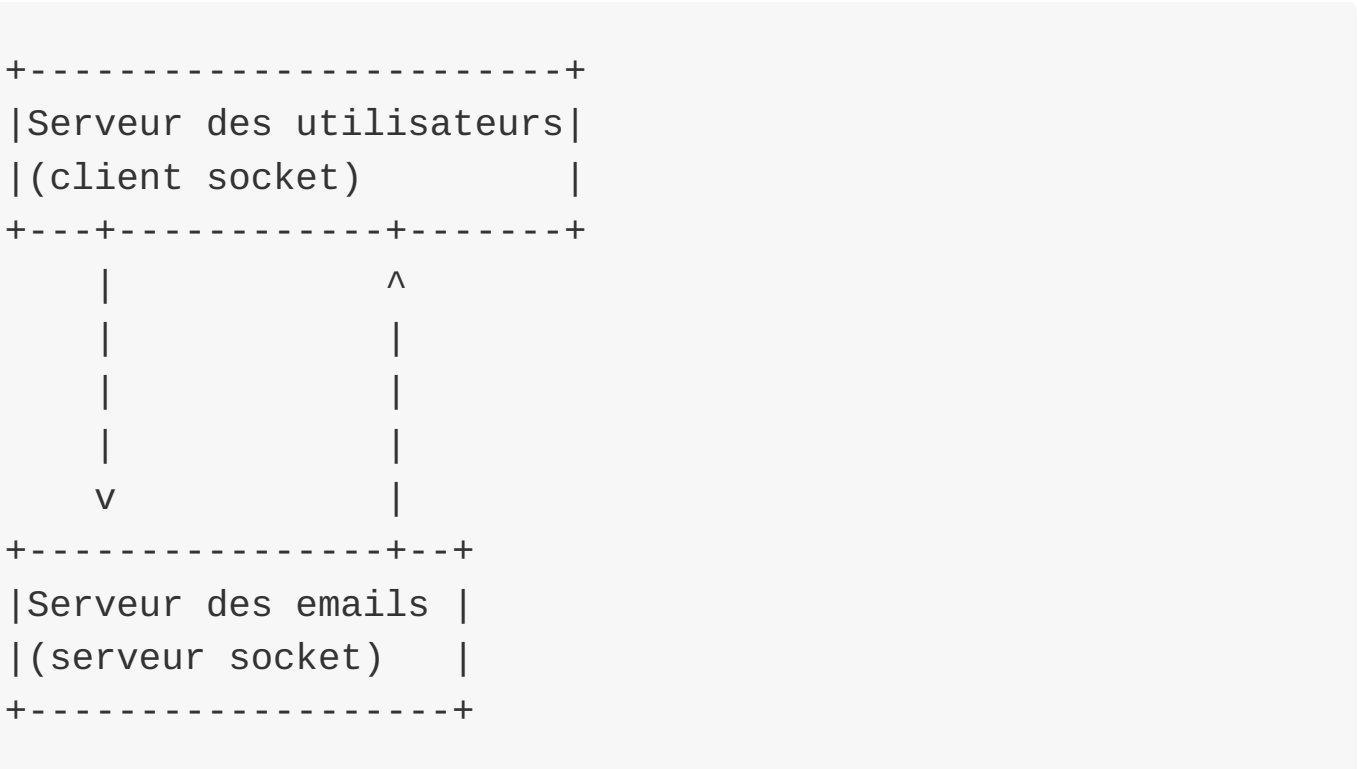
Mise en pratique au sein d'Hapi

Comme vous l'avez vu, socket.io fonctionne avec un système de client et de serveur. Dans ce TP, nous allons voir comment faire communiquer deux serveurs Hapi ensemble via ce protocole.

Attention : Ce qui est fait ici sert juste à illustrer socket.io au travers de Hapi. Dans un cas réel en production, pour faire communiquer deux API entre elles, le mieux est de passer par RabbitMQ ou Kafka qui sont des services de messaging et utilisent le protocole AMQP.

Actuellement, vous n'avez qu'une seule API servant à enregistrer des utilisateurs et à envoyer un email à la création/modification de chacun d'entre eux.

Maintenant, nous allons partir vers une approche dite de microServices. Nous allons créer un serveur Hapi par parties métier. Ce qui va nous donner ici un serveur Hapi pour gérer uniquement les utilisateurs et un serveur Hapi pour gérer uniquement l'envoi d'emails. Ce qui devrait donner le schéma structurel suivant :



Comme vous pouvez le voir, le serveur des mails sera aussi le serveur Socket pour des raisons de simplicité et pour savoir qui est-ce qui a demandé l'envoi de mails.

Serveur Socket.io

Afin d'instancier un serveur socket au sein de Hapi, nous allons passer pour un plugin nous simplifiant tout ça : `hapi-io`.

Ce plugin permet de définir une route standard de Hapi et de l'utiliser comme un événement socket comme ceci :

```
server.route([
    {
      method: 'GET',
      path: '/users/{id}',
      config: {
        plugins: {
          'hapi-io': 'get-user'
        }
      },
      handler(request, reply) {
        db.users.get(request.params.id, (err, user) => {
          reply(err, user);
        });
      }
    }
]);
```

Ici, la partie qui nous intéresse est l'objet `plugins` dans la partie `config`. Comme vous pouvez le voir, on dit que sur cette route, on active le plugin hapi-io et que cette route est accessible via l'événement `get-user`.

Au niveau du handler, il y a plusieurs possibilités pour interragir avec le client :

- Via un emit sur le socket (seul le client est notifié s'il est inscrit sur cet événement) :

```
let socket = request.plugins['hapi-io'].socket;
if (socket) {
  socket.emit('test-result');
}
```

- Via un emit sur `io` (effectue un broadcast) :

```
let io = request.plugins['hapi-io'].io;
if (io) {
  io.sockets.emit('global-result');
}
```

- Via un emit sur une room (seul les inscrits à cette room y ont accès) :

```
let io = request.plugins['hapi-io'].io;
if (io) {
  io.to('room:a').emit('test-result', { room : 'A' });
}
```

- Via l'appel au callback du `client.emit(event, cb)` (seul le client effectuant l'emit est notifié) :

```
reply(null, { msg : 'ok' });
```

Client Socket.io

De base, Hapi n'a aucun plugin permettant d'avoir un client socket.io au sein du serveur (et normalement un serveur n'est jamais un client socket.io). Mais heureusement, voici un plugin perso qui règle le souci :

```
const socket = require('socket.io-client');


module.exports.register = (server, options, next) => {
    let io = socket(options.server, {reconnect: true});

    io.on('connect', function () {
        console.log('socket.io is connected');
    });

    server.decorate('server', 'ioClient', io);

    next();
};

module.exports.register.attributes = {
    name :'ioClient'
};
```

Attention : Ce plugin a comme dépendance `socket.io-client`. Pensez donc à l'installer dans votre `package.json`.

Ce fichier est à placer dans le répertoire `/app/plugins` mais n'oubliez surtout pas de le charger dans le fichier `plugins.js` du manifest.

Important : Le paramètre `server` devra **obligatoirement** contenir le `http://` dans l'adresse du serveur. Ce qui pourra par exemple donner :

```
server.register({
    register : require('../../app/plugins/ioClient'),
    options : {
        server : 'http://0.0.0.0:8080'
    }
});
```

Ensuite vous pouvez envoyer/écouter un événement au/du serveur Socket.io de la façon suivante :

```
// inscription à un emit du serveur
server.ioClient.on('test-result', (params) => {
    console.log(params);
});

// envoie d'un emit au serveur
server.ioClient.emit('send-message', { mesg : 'Hello' });

// envoie d'un emit au serveur et traitement du callback
server.ioClient.emit('get-user', 1, (user) => {
    console.log(user);
});
```

Consignes fin de module

Pour rappel, les deux projets que vous devrez avoir à la fin (le serveur hapi pour les utilisateurs et le serveur hapi pour l'envoi de mails) doivent être impérativement rendus **avant le 24 février au soir**. Pour des raisons de simplicités et pour pouvoir vous donner facilement une correction à chacun, vous devrez envoyer **les liens de vos projets GIT** à benjamin.besse.aff@unilim.fr.

La note prendra en compte le bon fonctionnement des projets, que le fichier `package.json` soit bien rempli pour que le lancement de vos projets sur un autre poste que le votre se fasse sans encombre, que la partie hapi soit bien configurée et que la qualité du code soit au rendez-vous.