Chapitre 8
Temps réel

Plan

- SSE
- WebSockets

Temps réel

La majorité des concepts vus jusqu'à présent suffisent à construire une application web typique **complète** et **fonctionnelle**.

…. Par contre, certains autres **défis** doivent être résolus par des approches un peu plus '**exotiques**'…

Notamment tout ce qui touche au **temps réel**.

- Chat
- Résultats sportifs
- Bourse, bitcoin
- …

Temps réel

Que faire pour une application de ***chat*** (messagerie instantanée) ?

Approche typique : **long-polling**

À un intervalle de **X** secondes, on envoie une **requête** au serveur afin de recueillir les nouveaux messages.

→ Ce n'est pas que ça ne fonctionne pas, mais clairement pas la technique la plus efficace.

Temps réel

Revient à l'application d'un flot **événementiel** autant purement *client-side* que lors des interactions client-serveur.

- Souvent, seulement certaines parties de l'application suivront ce paradigme (complexifie tout de même le flot)
- Intéressant lorsque votre application est majoritairement basée sur des événements, peu intéressant s'il s'agit de simple CRUD.
- Difficulté d'implémentation varie selon le langage…

SSE (Server-Side Event)

Repose sur l'inversion de la communication entre le serveur et le client… donne la possibilité au serveur de mettre à jour l'application *client-side* !

- Possibilité de *broadcaster* des événements à tous les clients
- Un appel d'API pourrait lancer une mise à jour de d'autres parties de l'application
- Plus facile de gérer les tâches en *background* de votre API

SSE (Server-Side Event)

Utilise **EventSource** côté client

https://developer.mozilla.org/en/docs/Web/API/EventSource

```
<html>
<head>
 <meta charset="utf-8" />
</head>
<body>
 <script>
   var source = new EventSource('/events');
   source.onmessage = function(e) {
     document.body.innerHTML += e.data + '<br>';
   };
 </script>
</body>
</html>
```

SSE (Server-Side Event)

Facile d'ajouter des **event handlers** spécifiques:

```
source.addEventListener('userlogon', function(e) {
 var data = JSON.parse(e.data);
 console.log('User login:' + data.username);
}, false);
```

Ne pas oublier de **closer** la source...

```
source.close();
```

SSE (Server-Side Event)

Le côté serveur est toujours un peu plus complexe...

```
const http = require('http');
const sys = require('sys');
const fs = require('fs');

http.createServer((req, res) => {
 if (req.headers.accept && req.headers.accept == 'text/event-stream') {
   if (req.url == '/events') {
     sendSSE(req, res);
   } else {
     res.writeHead(404);
     res.end();
   }
 }
}).listen(8000);
```

Exemple complet: https://www.html5rocks.com/en/tutorials/eventsource/basics/

SSE (Server-Side Event)

Le côté serveur est toujours un peu plus complexe...

```
function sendSSE(req, res) {
 res.writeHead(200, {
   'Content-Type': 'text/event-stream',
   'Cache-Control': 'no-cache',
   'Connection': 'keep-alive'
 });

 var id = (new Date()).toLocaleTimeString();
 constructSSE(res, id, (new Date()).toLocaleTimeString());
}

function constructSSE(res, id, data) {
 res.write('id: ' + id + '\n');
 res.write("data: " + data + '\n\n');
}
```

SSE (Server-Side Event)

Remarquez l'introduction d'un nouveau **content type** spécifique: **text/event-stream**.

Exemple complet en Java:

http://www.cs-repository.info/2016/08/server-sent-events.html

Exemple en Ruby avec **Sinatra**:

https://www.eriwen.com/javascript/server-sent-events/

WebSockets

Pourquoi pas ajouter une couche et établir une communication bidirectionnelle?

WebSockets :

- Communication **full-duplex** basé sur le protocole **TCP**.
- Doit être implémenté **client** ET **server-side**.

- Basé sur l'envoi de **messages** (au lieu de bytes en TCP).
  → On peut donc envoyer n'importe quoi!

- Conçu pour fonctionner exclusivement avec un **navigateur web**.

WebSockets

Le client doit d'abord faire une demande d'***upgrade*** vers une communication WebSocket.

Requête **HTTP** correspondante :

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

Demande d'upgrade

En-têtes typiques de WebSockets.

WebSockets

Le serveur renvoie ensuite une confirmation de la mise à jour du protocole utilisé.

Réponse **HTTP** correspondante :

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

Demande d'upgrade

En-têtes typiques de WebSockets.

WebSockets

- WebSockets utilise un nouveau schéma d'uri : **ws://** ou **wss://**
- WebSockets est supporté par la grande majorité des navigateurs récents.
- Comme SSE, souvent utilisé en combinaison avec un REST API classique… tout n'a pas besoin d'être temps réel.

WebSockets

WebSockets en JavaScript est supporté nativement: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API

Ouvrir une **connexion** :

```
let webSocket = new WebSocket("ws://myWebSocketUrl");
```

Fermer une **connexion** :

```
webSocket.close();
```

Envoi de **message** :

```
webSocket.send({"artist": "Borgore"});
```

WebSockets

WebSockets en JavaScript …

Le côté client repose sur une **gestion des événements** reçus ou envoyés lors de la communication WebSockets.

4 types d'événements sont disponibles :

- **onopen**
- **onmessage**
- **onclose**
- **onerror**

WebSockets

Explorons WebSockets en Javascript …

Événements :

```
webSocket.onopen = function() {
     updateStatus("wsStatus", "Connected to WebSocket server!");
}

webSocket.onmessage = function(event) {
     displayMessage(event.data);
}

webSocket.onclose = function() {
     updateStatus("wsStatus", "WebSocket closed!");
}

webSocket.onerror = function(event) {
     updateStatus("wsStatus", "WebSocket error : " + event.data);
}
```

WebSockets

Différentes implémentations possibles côté serveur

- NodeJS (que nous allons explorer)
- Java (http://www.baeldung.com/java-websockets)
- Go (https://github.com/gorilla/websocket)
- Python (https://pypi.python.org/pypi/websocket-client)

Existe surement en PHP...

WebSockets

Vraiment beaucoup de *use cases* possibles…

- Collaboration en temps réel (Drive)
- Réseaux sociaux
- Jeux

Simplement garder en tête qu'il ne s'agit pas d'un ***silver bullet***, mais bien d'un outil avec un but particulier… n'est pas une alternative mais bien un complément au classique REST.

Socket.IO

Extension WebSockets pour NodeJS

Voir : http://socket.io/

- Devenu la référence avec NodeJS
- Étonnamment, SocketIO utilise **Adobe Flash** si installé.
- Les librairies client et serveur sont indépendantes et peuvent être utilisées avec d'autres technos… mais évidemment fonctionnent mieux lorsque combinées.

socket.io

Socket.IO

SocketIO est composé de deux parties:

- Serveur WebSockets, qui utilise le serveur HTTP de NodeJS. (côté serveur). Souvent utilisé en combinaison avec Express.

- Client SocketIO, qui doit être importé dans votre JavaScript **client** (wrapper des WebSockets HTML5). https://github.com/socketio/socket.io-client

Socket.IO

Établir une connection simple:

```
// Du côté serveur
var app = require('express')();
var http = require('http').createServer(app);
var io = require('socket.io')(http);

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

http.listen(3000);

// Du côté client
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect("http://localhost:3000/");
</script>
```

Socket.IO

Envoi de messages:

```
// Du côté serveur
io.on('connection', function(socket){
  // Événement nommé chat message.
  socket.on('chat message', function(msg){
    console.log('message: ' + msg);
  });
});

http.listen(3000);

// Du côté client
<script src="/socket.io/socket.io.js"></script>
<script>
  var socket = io.connect("http://localhost:3000/");
  socket.emit('chat message', 'Hi!');
</script>
```

⇒ Importance de la **cohérence** du noms des événements!

Socket.IO

Envoi d'événements depuis le serveur:

```
// Du côté serveur
io.on('connection', function(socket){
  // Événement nommé chat message.
  socket.on('chat message', function(msg){
    io.emit('chat message', msg);
  });
});
```

io.emit envoie un message à tous les clients connectés, ***socket.broadcast.emit*** envoie un message à tous les clients *sauf* celui connecté.

Socket.IO

Réception de messages du côté client:

```
// Du côté client
<script>
  var socket = io();

  // Envoi d'un événement vers le serveur
  socket.emit('chat message', 'Hi!');

  // Réception d'un événement côté client
  socket.on('chat message', function(msg){
    $('#messages').append($('<li>').text(msg));
  });
</script>
```

MVC?

Probablement plus sage d'utiliser le client WebSocket JavaScript natif…

Des exemples de librairies ici:

https://github.com/icebob/vue-websocket

https://www.npmjs.com/package/vue-native-websocket

MVC?

Notez bien :

Le mélange de frameworks n'est pas toujours une bonne idée. Notamment, intégrer une logique de WebSockets avec un framework tel que **AngularJS** peut s'avérer difficile…

- Posez vous vraiment la question si votre application nécessite le tout.
- Si oui, assurez de choisir les bonnes technologies pour faciliter un heureux mariage :)