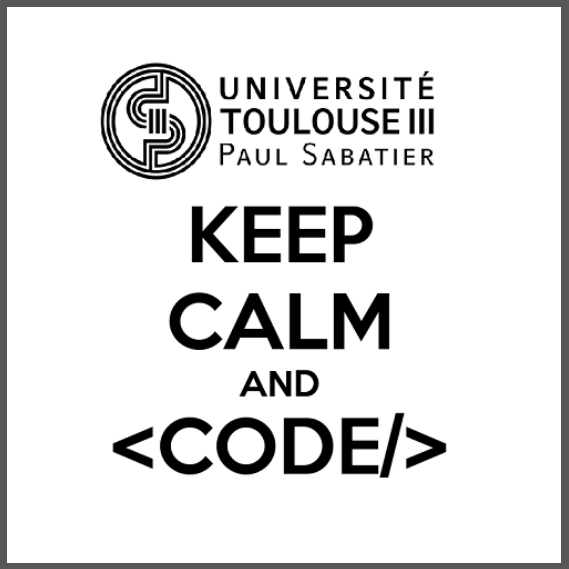


Programmation back

Express

Sylvie TROUILHET - www.irit.fr/~Sylvie.Trouilhet

Générateur express

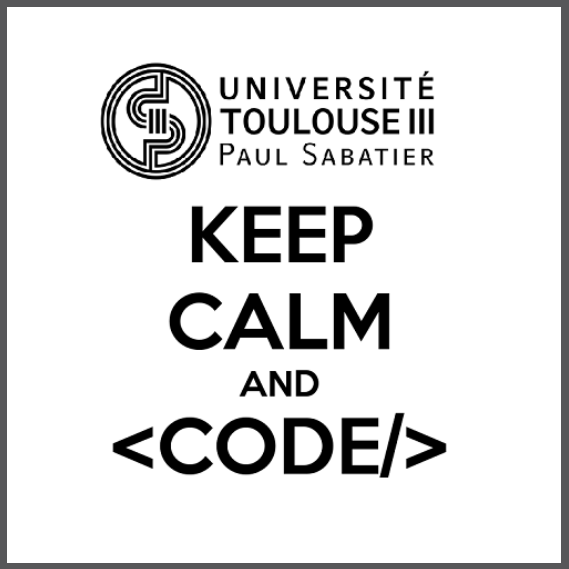
Installer en **global** : `npm install express-generator` **`-g`**

Créer l'appli web `monAppli`

```
express --view=ejs monAppli
```

Création du « squelette de l'appli »

```
 create : monAppli
create : monAppli/package.json
create : monAppli/app.js
create : monAppli/public
create : monAppli/views
create : monAppli/views/index.ejs
create : monAppli/views/error.ejs
create : monAppli/routes
create : monAppli/routes/index.js
create : monAppli/routes/users.js
create : monAppli/bin
create : monAppli/bin/www
create : monAppli/public/javascripts
create : monAppli/public/images
create : monAppli/public/stylesheets
create : monAppli/public/stylesheets/style.css
install dependencies:
  > cd monAppli && npm install
run the app:
  > SET DEBUG=monappli:* & npm start
```



Programmation back

Moteur de vue

Sylvie TROUILHET - www.irit.fr/~Sylvie.Trouilhet

Moteur de vues : ejs|hbs|hjs|jade|pug|twig|vash

view (modèle MVC):

afficher un document HTML à partir du modèle de données

modules **`pug`** ou **`ejs`** (Embedded JavaScript) comme moteur de vues

ejs : module à installer avec npm

Ejs (www.embeddedjs.com)

La vue se trouve dans un fichier .ejs dans le répertoire **views**

Les éléments variables sont indiqués par des balises :

```
<%=  nom  %>
```

Dans le fichier .js:

```
app.set('view engine', 'ejs');
```

les balises sont remplacées par les valeurs indiquées entre accolades **`{nom: valeur}`**

```
response.render(fichier, valeurs);
```

Exemple

/views/reponse.ejs

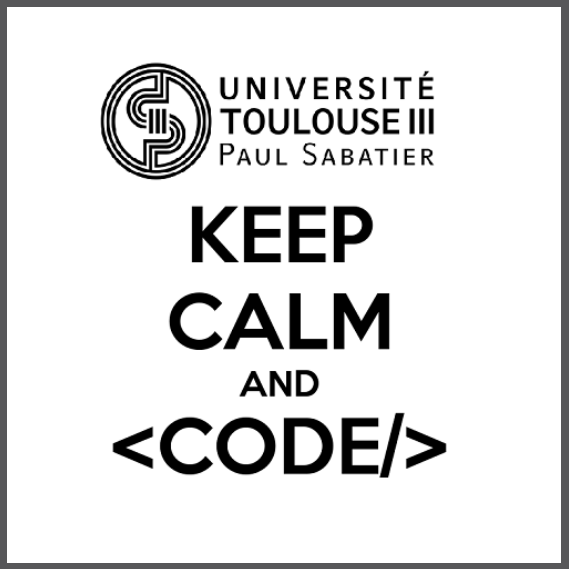
```
<!DOCTYPE html>
<head> … </head>
<body>
…
<h1>Un exemple de vues</h1>

<p>Vous êtes le joueur <%= nom %> </p>
<p>et la proposition est <%= proposition %> </p>
…
</body>
```

Fichier index.js

```
const express = require('express');
…
app.set('view engine', 'ejs');
…
app.get('/index.html', construireVue);
function construireVue(request, response){
 response.render('reponse.ejs',{nom:'Joueur1', proposition:10});
 }
```


localhost:4000/index.htm
localhost:4000/index.html
Un exemple de vues
Vous êtes le joueur Joueur1
et la proposition est 10



Programmation back

Utiliser les web sockets

Sylvie TROUILHET - www.irit.fr/~Sylvie.Trouilhet

Plan

1. Principes
2. Requête du client
3. Requête du serveur
4. Mise en œuvre

Le module socket.io

Communication client-serveur : c'est le client qui est à l'initiative de la communication (envoi d'une requête), le serveur ne fait que transmettre la réponse.
Question : comment permettre au serveur d'être à l'initiative de l'envoi d'une requête ?

Par exemple : proposer un canvas partagé où les modifications des uns sont visualisées par tous.

→Web sockets : une fois la connexion entre un client et un serveur établie, la communication peut s'effectuer à tout instant dans les deux sens.

Le module **`socket.io`**

Il faut un programme côté client + un programme côté serveur

Côté client : page HTML incluant un fichier Javascript
Côté serveur : programme écrit en Node

La communication se fait par événements : lorsque le client veut communiquer au serveur, il lui envoie un événement que celui-ci peut recevoir et traiter (et inversement).

Requête du client vers le serveur

Fichier **index.html**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Les web sockets</title>
  <script src="http://localhost:4000/socket.io/socket.io.js"></script>

</head>
<body>
<div id="main"> Hello !!! </div>
<script>
    const socket = io.connect('http://localhost:4000');
    //socket.emit("event1", "eh ! je t'appelle");
</script>
</body>
</html>
```

2 méthodes

méthode	**rôle**
io.connect(url)	Crée une liaison avec le serveur et retourne un objet socket qui servira à la communication entre ce client et le serveur
socket.emit(event, params)	Envoie un événement au serveur en lui transmettant les paramètres indiqués, qui sont soit sous forme d'objest JSON, soit sous forme de chaînes de caractères

Traitement de la requête par le serveur

Fichier **serveur.js**

```
const http = require('http');
const fs = require('fs');
const server = http.createServer
(
  function (request, response) {
     fs.readFile('./index.html', 'utf-8', function (error, content) {
             response.writeHead(200, {"Content-Type": "text/html"});
             response.write(content);
             response.end();
            } );
 });
const io = require("socket.io")(server);

io.sockets.on("connection", function (socket) {
        console.log("un client s'est connecté");
            });

server.listen(4000);
```

1 méthode / 1 événement

méthode	rôle
require("socket.io").listen(server)	Transforme le serveur http en serveur pouvant dialoguer par web sockets avec les clients qui vont se connecter au serveur. Retourne une objet io sur lequel on pourra écouter les connections des clients.

événement	rôle
"connection"	Événement reçu sur l'objet io.sockets, servant à recevoir les connexions des utilisateurs sur le serveur. Une fois cet événement réceptionné, tous les autres événements transmis par le client sont reçus dans la fonction de callback traitant cet événement

Déconnexion d’un client

Le serveur doit positionner un écouteur d’événement **`"disconnect"`**

```
socket.on("disconnect", callback)
```

Requête du serveur vers le client

Fichier serveur.js

```
…
io.sockets.on("connection", function (socket) {
        console.log("un client s'est connecté");
        socket.emit("event2", "Message envoyé par le serveur");
        socket.on("event2", function(data) {})
});
```

1 méthode

méthode	**rôle**
socket.emit(events, params)	Envoie un événement au client en lui transmettant les paramètres indiqués, qui sont soit sous la forme d'objets JSON, soit sous la forme de chaînes de caractères.

Traitement de la requête par le client

Fichier **index.html**

```
…
socket.on("event2", function (data) {
            console.log(data);
        }
);
```

2 méthodes

méthode	**rôle**
io.connect(url)	Crée une liaison avec le serveur et retourne un objet socket qui servira à la communication entre ce client et le serveur
socket.on(event, callback)	Positionne un gestionnaire d'événements permettant de traiter l'événement event reçu du serveur. La fonction de callback est de la forme function(data) dans laquelle data représente les données transmises (objet ou chaîne)

Diffusion à plusieurs clients

méthode	**rôle**
socket.emit(event, params)	Diffusion vers un seul client, celui représenté par socket
socket.broadcast.emit(event, params)	Diffusion de l'événement à tous les clients connectés, sauf nous-mêmes (celui représenté par la variable socket)
io.sockets.emit(event, params)	Diffusion de l'événement à tous les clients connectés, y compris nous-mêmes