



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER

**KEEP
CALM
AND
<CODE/>**

Programmation back

Réalisation et test d'une application en node.js

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



**KEEP
CALM
AND
<CODE/>**

Programmation back

Express

Faire un projet express

Installer Express (en gérant les dépendances) :

```
npm install express --save
```

Installer le générateur d'application (en **global**) :

```
npm install express-generator -g
```

Créer l'appli web monAppli (avec le moteur de vues ejs) :

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

Création du « squelette de l'application »

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

Fichier package.json

```
{
  "name": "monappli",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "ejs": "~2.6.1",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1"
  }
}
```

Les dépendances sont les fichiers qui sont liés à votre projet

Elles sont indiquées dans package.json

```
"dependencies": {      "express": "^4.16.1"  
}
```

→ accepte les changements mineurs de version

Installer les dépendances

```
npm install
```

→ Les modules sont ajoutés dans le répertoire node_modules

Lors de la mise au point

Le module `nodemon` : relance l'exécution du fichier à chaque modification

Installer le module : `npm install nodemon`

Remplacer `node` par `nodemon` dans `package.json`



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER

**KEEP
CALM
AND
<CODE/>**

Programmation back

Moteur de vue

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

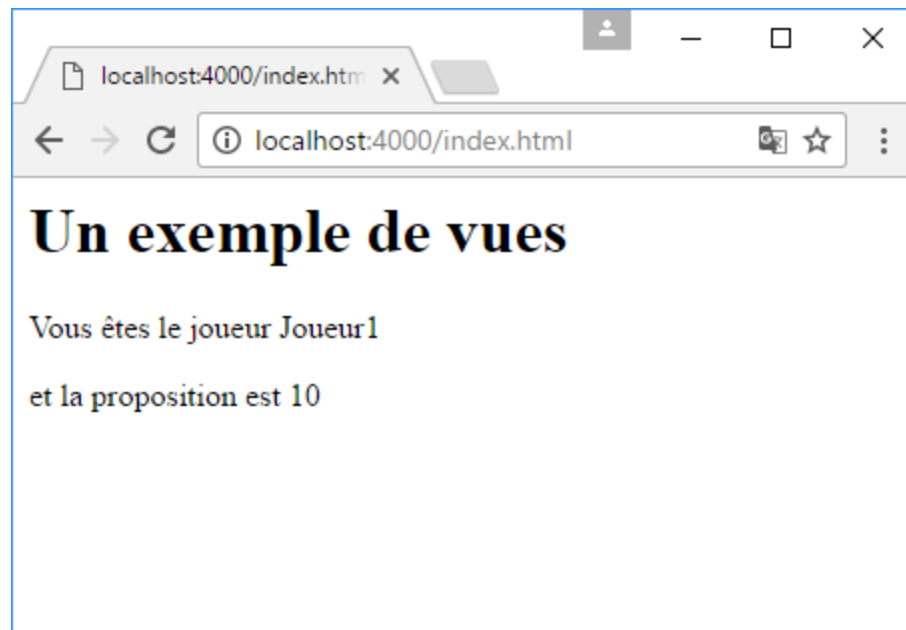
/views/reponse.ejs

```
<!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});  
}
```





**KEEP
CALM
AND
<CODE/>**

Programmation back

Utiliser les web sockets

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

</head>
<body>
<div id="main"> Hello !!! </div>
<script>
  const socket = io();
  // message envoyé au serveur
  socket.emit("event1", "eh ! je t'appelle");
</script>
</body>
</html>
```

2 méthodes

méthode	rôle
<code>io.connect(url)</code>	Crée une liaison avec le serveur et retourne un objet socket qui servira à la communication entre ce client et le serveur
<code>socket.emit(event, params)</code>	Envoie un événement au serveur en lui transmettant les paramètres indiqués, qui sont soit sous forme d'objet 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
<code>require("socket.io").listen(server)</code>	Transforme le serveur http en serveur pouvant dialoguer par web sockets avec les clients qui vont se connecter au serveur. Retourne un 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("event1", 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
<code>io.connect(url)</code>	Crée une liaison avec le serveur et retourne un objet socket qui servira à la communication entre ce client et le serveur
<code>socket.on(event, callback)</code>	Positionne un gestionnaire d'événements permettant de traiter l'événement event reçu du serveur. La fonction de callback est de la forme <code>function(data)</code> dans laquelle data représente les données transmises (objet ou chaîne)

Diffusion à plusieurs clients

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



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER

**KEEP
CALM
AND
<CODE/>**

Programmation back

Tests unitaires avec Mocha

Les tests

Tests unitaires : tester **chaque fonction** individuellement.

→ rédiger un ou plusieurs tests unitaires qui permettent de valider le comportement de la fonction, même en lui passant des paramètres incorrects (null, ...)

Test d'intégration : vérifier que tous les **éléments marchent ensemble**, et que ce qu'ils font est bien ce qui a été défini.

Tests unitaires

Fonctionnement classique en 4 phases

Initialisation (set up) : définition d'un environnement de test complètement reproductible

Exercice : le module à tester est exécuté

Vérification (assert) : comparaison des résultats obtenus avec un vecteur de résultats défini. Ces tests définissent le résultat du test : `SUCCESS` OU `FAILURE`

Désactivation (clean up) : désinstallation pour retrouver l'état initial du système.

Tous les tests doivent être indépendants et reproductibles unitairement.

Automatisation des tests : framework Mocha

Framework Mocha : <https://www.npmjs.com/package/mocha>

Installation (locale au projet) :

```
npm install --save-dev mocha
```

Enregistrer les fichiers de test dans le répertoire **test**

Commande **node node_modules\mocha\bin\mocha**

→ Ajout dans `package.json` de

```
"devDependencies": {  
  "mocha": "^5.0.0" }
```


Pour lancer le projet

L'attribut **"scripts"**

(<https://docs.npmjs.com/misc/scripts>)

Compléter **scripts** dans **package.json** :

```
"scripts": {  
  "test": "mocha" ,  
  "start": "node index.js"  
}
```

NB : `npm test` → comme `node node_modules\mocha\bin\mocha`

Automatisation des tests : framework Mocha

```
describe
('Test du fichier XX', function()
{
  // le fichier xx contient la fonction quoi
  describe
  ('test de quoi', function()
    {
      // vérification avec le module assert
    }
  );
});
```

Automatisation des tests : framework Mocha

`ok(value, [msg])` : l'expression est vraie

`equal(actual, expected, [msg])` : deux expressions sont
égales

`notEqual(actual, expected, [msg])`

Exemple

Créer un répertoire `lib`

Ajouter un fichier `utilitaire.js` avec la fonction

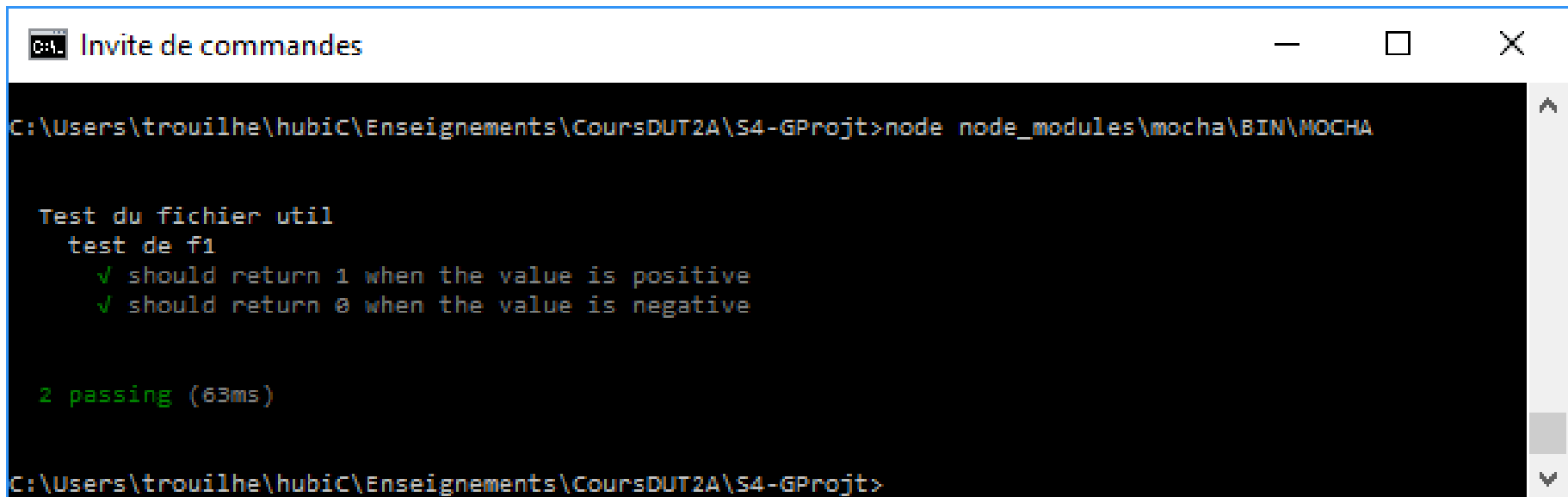
```
function quoi(v1)
{
  let n=1,f=1;
  while (f < v1) f*=++n;
  return f==v1;
}
module.exports.quoi = quoi;
```

Automatisation des tests : framework Mocha

```
const assert = require('assert');
const m=require('../lib/utilitaire.js');

describe
('Test du fichier util', function()
  { describe
    ('test de f1', function()
      {
        it('should return true when ...',
function() {assert.equal(true, m.quoi(120));});
        it('should return false when ...',
function() {assert.equal(false, m.quoi(100));});
      }
    );
  }
);
```

Automatisation des tests : framework Mocha



```
C:\Users\trouilhe\hubiC\Enseignements\CoursDUT2A\S4-GProjt>node node_modules\mocha\BIN\MOCHA

Test du fichier util
  test de f1
    ✓ should return 1 when the value is positive
    ✓ should return 0 when the value is negative

2 passing (63ms)

C:\Users\trouilhe\hubiC\Enseignements\CoursDUT2A\S4-GProjt>
```

Automatisation des tests : framework Mocha

initialisation :

before, qui sera exécuté avant la série de tests

beforeEach, qui sera exécuté avant chaque test

désactivation :

afterEach, qui sera exécuté après chaque test

after, qui sera exécuté après la série de tests