1819_INFO2_PWCS_NodeJS_TP1

Objectifs

A l'issue de cette séance de TP, vous serez capables de

- Mettre en place un serveur NodeJS
- Faire un site web simple avec NodeJS

NodeJS en deux mots

NodeJs est en fait le moteur (javascript) V8 de Google. Il permet donc d'interpréter du Javascript en dehors d'un navigateur. Il est accompagné d'un ensemble de modules qui permettent de s'interfacer avec le système d'exploitation et en particulier avec le système de fichier et les ports de communication. En particulier, il est possible de rédiger rapidement un serveur http ou de récupérer les mouvements de la souris ou d'un game-pad. L'intérêt de nodeJS est que son execution est parallèle (merci la programmation évenementielle et les appels non-bloquant): il peut servir un très grand nombre de requêtes simultanément sans que le développeur ai à se soucier de concurrence d'accès. Comme il traite du JS, vous pouvez réutiliser des framework existant du côté client (jQuery, ...).

Références

Sur la page Moodle, vous trouverez deux livres sur ES6 et sur NodeJS : « Javascript : Novice to Ninja » (jsninja2) et « Exploring JS » (explJS). Nous nous reporterons dans ce TP. Le site nodejs.org est bien sur le site de référence.

Premier programme

Node est installé sur les machines virtuelles Linux de TP ENSSAT. Dans un répertoire de votre choix, créez un fichier **monPremierProgramme.js** (avec l'éditeur de votre choix, atom étant très bien) contenant le simple appel :

```
console.log( 'Hello World' ) ;
```

Exécutez le code par la commande suivante (dans un terminal) :

```
>node monPremierProgramme.js
```

Vous verrez apparaître la chaîne ''Hello World''.

Premier serveur Web

NodeJS a une structure modulaire. Le noyau ne contient que le moteur JS. Il faut lui adjoindre des modules pour développer ses capacités (par exemple accès au système de fichiers, ...). Ici, nous allons permettre à notre programme de communiquer via le protocole HTTP. Oui, nous allons faire notre premier serveur HTTP en node. Éditez un fichier **monPremierServeurHTTP.js,**

```
const http = require('http'); //chargement du module http

const hostname = '127.0.0.1'; //notez la notation ES6
const port = 3000;

//que doit faire le serveur lorsq' une requête lui parvient ?
const server = http.createServer((req, res) => { //req et res sont les requête et réponses HTTP
  res.statusCode = 200; //code retour portée par la réponse HTTP
  res.setHeader('Content-Type', 'text/plain'); //type MIME de la réponse HTTP
  res.end('Hello World¥n'); //corps de la réponse et envoi dans une même fonction
});

// lancement du serveur
server.listen(port, hostname, () => { //que faire si le serveur s' est bien lancé ?
  console.log(`Server running at http://${hostname}:${port}/`);
```

```
});
```

exécutez-le et rendez vous sur http://localhost:3000.

Premier module

Vous serez amenés à produire du code que vous utiliserez souvent. Afin de ne pas le réécrire, vous pouvez l'encapsuler dans un module, un fichier JS proposant une interface. On trouve un bel exemple dans explJS, en 16.3.

```
//------ lib.js ------
var sqrt = Math.sqrt;
function square(x) {
    return x * x;
}
function diag(x, y) {
    return sqrt(square(x) + square(y));
}
module.exports = {
    sqrt: sqrt,
    square: square,
    diag: diag,
};

//------ main.js ------
var square = require('lib').square;
var diag = require('lib').diag;
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```

Le module est la bonne échelle pour définir des classes comme par exemple ici :

```
const Student = require('./student.js');

class Students{
  constructor(){
    this.students= new Map();
  }
  get size(){
    return this.students.size;
  }
  addStudent(newStudent){
    this.students.set(newStudent.numetudiant,newStudent);
  }
}

module.exports = Students;
```

Proposez une classe *personne*, dans un module **personne.js.** Une personne est définie par un nom, un prénom, une adresse, ... (inspirez vous d'une vCard). Les méthodes associées sont des getters et setters. Utilisez cette classe dans une application nodeJS pour manipuler 2 ou 3 objets personnes. En particulier, observez l'affectation d'un objet créé à une variable ... s'agit il de deux objets ? d'un seul ?

Premier projet (npm)

NodeJS dispose d'un outil de construction de projet très simple : npm (Node Paquet Manager). Il permet d'importer des modules depuis un dépot nommé npmjs.com. Les modules sont téléchargés dans un répertoire nommé **node_modules**. Les dépendances de votre projet avec ces modules importés sont formalisés dans un fichier nommé **package.json**. Ainsi, lorsque vous déployez votre application, vous n'exportez que vos fichiers et la liste des dépendances. Ces dépendances sont résolues à la lecture du **package.json**.

Pour créer un nouveau projet, utilisez la commande `npm init` puis répondez aux questions (des valeurs pas défaut sont proposées : elles conviennent très bien). Un fichier package.json est crée, il doit ressembler à cela :

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo ¥"Error: no test specified¥" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Test Driven Developement

Nous allons utiliser notre premier module importé : JEST. Ce module permet de faire du développement diérigé par le test (TDD, https://jestjs.io/). Suivez les étapes proposées à l'url suivante : https://jestjs.io/docs/en/getting-started.html.

```
npm install --save-dev jest
```

l'option save-dev précise que la dépendance ne se fait qu'à l'étape de développement. Ainsi, JEST ne sera pas importé lors du déploiement.

Transformez les fonctions développées lors de la section 'Premier module' pour les intégrer dans des tests unitaires JEST.

Documentation

La réutilisation des modules ne peut être efficace que si vous les avez bien documentés. Je vous propose d'utiliser l'outils jsDoc . Installez le module jsdoc (https://github.com/jsdoc3/jsdoc) puis écrivez la documentation de votre module (http://usejsdoc.org/about-getting-started.html).

Pour automatiser la génération de la documentation, ajoutez une ligne script à **package.json**.

Express

Vous allez utiliser tout ce que vous avez appris jusqu'à présent pour faire un premier site web (pas une application) hébergée sous node. Pour cela, vous allez utiliser le mini framework express (http://expressjs.com/). Suivez les étapes de 'Getting Started' pour servir des pages staiques.

Pour aller plus loin, suivez les étapes de http://expressjs.com/en/guide/using-template-engines.html. Vous utiliserez le moteur de template EJS.

Les étapes suivantes consisteront à intégrer une base de donnée (http://expressjs.com/en/guide/database-integration.html#mongodb) MongoDb. Vous n'utiliserez pas d'instance de Mongodb locale mais une instance dans le Cloud. Je recommande https://www.mongodb.com/cloud.