Node.js (Séance 8)
https://tinyurl.com/ybu27uyc

Dans la plupart des langages de programmation côté serveur comme C ou PHP, les codes sont en général exécutés de manière séquentielle. Cela signifie qu'un serveur ne traite une nouvelle instruction du code que lorsque l'instruction précédente a fini d'être exécutée et a livré un résultat. Dans ce cas, on parle également de **tâches synchrones**. L'exécution des autres codes est ainsi bloquée aussi longtemps qu'une tâche en cours n'est pas terminée. Ceci peut mener à des ralentissements importants, particulièrement pour les accès au système de données, aux bases de données ou aux services Web.

De nombreux langages de programmation ou environnements d'exécution vont alors donner la possibilité de réaliser les tâches de manière parallèle grâce à ce que l'on appelle les **threads (process).** Il s'agit de fils exécutés dans le cadre d'un processus avec lequel les actions sont traitées pendant que le reste est également exécuté. L'inconvénient de cette méthode : plus des fils sont exécutés et plus il est nécessaire de disposer de temps CPU et de mémoire vive (RAM). En d'autres termes, le **multithreading** est particulièrement gourmand en ressources. De plus, les fils d'exécution supplémentaires s'accompagnent d'un besoin plus grand en programmation. En revanche, l'implémentation de JavaScript côté serveur permet de contourner ce problème. C'est pourquoi Node.js a été créé.

Node.js est une **plateforme logicielle avec une architecture orientée événements** qui permet **d'utiliser le langage de script JavaScript**, initialement développé pour une utilisation côté client, **pour une utilisation côté serveur**. Cela fonctionne comme PHP, Java, .NET, Ruby ou Python pour écrire du code pour le serveur. Il est utilisé pour le développement d'applications JavaScript côté serveur qui doivent assumer de fortes charges en temps réel.

La plateforme logicielle de Node.js se base sur le **moteur V8 du JavaScript de Google** qui intervient aussi sur le navigateur Web Chrome.

Node.js comprend une librairie de **modules JavaScript divers**, qui peuvent être chargés grâce à une fonction simple et servir directement comme élément structurel pour le développement d'applications Web. Un exemple est le module HTTP, qui permet grâce à une fonction unique de créer un serveur Web rudimentaire. En outre, le package intégré (dit « NPM » pour Node Package Manager) permet d'installer des modules complémentaires.

L'avantage majeur de Node.js est son architecture orientée événements, avec laquelle un code de programme s'exécute de manière **asynchrone**. Node.js est donc basé sur **un seul fil d'exécution** et un système entrée/sortie (ou I/O pour Input/Output) qui permet un traitement parallèle de plusieurs opérations d'écriture et de lecture.

- **I/O asynchrone :** parmi les tâches classiques effectuées par un serveur, on compte la réponse aux requêtes, la sauvegarde de données dans une base de données, la lecture de fichiers depuis le disque dur, ou encore l'établissement de connexions vers d'autres composants du réseau. Ces activités sont désignées comme sortantes/entrantes. Les opérations entrantes/sortantes sont exécutées de manière synchrone dans des langages de programmation tels que C ou Java. Ainsi, les tâches sont effectuées les unes après les autres. Cela signifie que le système I/O est bloqué tant que la tâche en cours n'est pas terminée. En revanche, Node.js utilise un I/O asynchrone, qui délègue les opérations d'écriture ou de lecture directement au système d'exploitation ou aux bases de données. Ceci permet de travailler sur un grand nombre de tâches en parallèle sans avoir de blocage. Les applications basées sur Node.js et JavaScript présentent ainsi l'énorme avantage de gagner en rapidité dans de nombreux cas de figure.

- **Fil d'exécution unique :** pour compenser le temps d'attente dans les I/O synchrones, des applications serveur basées sur des programmes de langage classiques orientés

serveurs s'appuient sur plusieurs threads (avec les inconvénients du multithreading). Ainsi, un serveur Apache HTTP commence par exemple un nouveau fil à chaque requête détaillée. Le nombre de fils possibles (et donc aussi le nombre de requêtes qui peuvent être traitées parallèlement dans un système multi-threadé synchrone) est limité par l'espace mémoire disponible. Node.js en revanche ne fait intervenir qu'un fil d'exécution du fait du system I/O décentralisé ce qui permet de réduire considérablement la complexité mais aussi l'exploitation des ressources.

- **Architecture orientée événements :** l'exécution asynchrone d'opérations I/O est réalisée grâce à l'architecture orientée évènements de Node.js. Il se base en réalité essentiellement sur un seul fil (mono-thread) qui se trouve sur une boucle d'évènements. Cette dernière a pour mission d'attendre des évènements et de les gérer. Les évènements sont alors classés en tant que tâches ou résultats. Si la boucle enregistre une tâche, par exemple une requête de base de données, cette dernière va être mise en arrière-plan du processus avec ce que l'on appelle une fonction de rappel (Callback). Le traitement de la tâche n'a pas lieu sur le même fil sur lequel la boucle fonctionne, ce qui permet à cette dernière de passer immédiatement à l'événement suivant. Si une tâche transférée est exécutée, les résultats du processus transféré sont retournés à la boucle comme nouvel évènement grâce à la fonction callback. Par conséquent, la boucle peut déclencher la remise du résultat.

Ce modèle non bloquant orienté évènement a l'avantage de ne jamais laisser inactive l'application sur Node.js en attendant un résultat. Il est ainsi par exemple possible **d'exécuter différentes requêtes des bases de données simultanément** sans que le programme ne soit retardé. L'établissement d'un site, qui exige différentes requêtes externes, se déroule bien plus rapidement avec Node.js qu'avec un processus de traitement synchrone.

Node.js est disponible ici https://nodejs.org/et peut être téléchargé avec l'installeur et/ou le paquet binaire, et ce suivant les systèmes d'exploitation

Modules de base

De base, Node.js ne sait en fait pas faire grand chose. Pourtant, Node.js est très riche grâce à son extensibilité. Ces extensions de Node.js sont appelées **modules**. Il existe des milliers de modules qui offrent des fonctionnalités . Il y a à peu près tout ce dont on peut rêver et de nouveaux modules apparaissent chaque jour.

Pour charger n'importe quel module de base dans une application Node.js, vous aurez simplement besoin de la fonction *require()*. Cette dernière nécessite **une chaîne (« string »)**, qui déclare comme paramètre le module qui doit être chargé. S'il s'agit d'un module de base, le **nom de module** suffit. Pour les modules installés ultérieurement, la chaîne doit contenir l'emplacement du module, même quand le module se trouve dans le répertoire actuel (dans ce cas, vous pouvez simplement écrire « ./ »). Les lignes de code suivantes montrent le schéma de base, selon lequel un module peut être chargé dans une application Node.js :

```
var nommodule = require('emplacement/nommodule');
```

exemple (0SampleServer)

```
var os = require('os');
var freemem = os.freemem();
console.log(freemem);
```

Installation de modules

L'installation de modules de programmes s'effectue avec Node.js grâce au Package Manager (NPM) intégré. Les modules d'extension ne doivent en effet pas être téléchargés manuellement depuis des pages externes puis copiés dans un répertoire. L'installation se limite à une seule ligne de code, qui doit être entrée dans la **console du système d'exploitation** (et non dans la console Node.js !) Avec les systèmes d'exploitation Windows, il s'agit de l'interpréteur de commandes *cmd.exe*, qui s'appelle grâce à l'entrée établie dans la boîte de dialogue d'exécution Windows (touche Windows + R). Seule la commande *npm install* est nécessaire ainsi que le nom du module installé :

```
npm install nommodule
```

Le module sera installé *localement* spécialement pour votre projet.
Si vous avez un autre projet, il faudra donc relancer la commande pour l'installer à nouveau pour cet autre projet. Cela vous permet d'utiliser des versions différentes d'un même module en fonction de vos projets.

Allez faisons un test. On va installer le module colors qui permet de bénéficier de couleur dans la console node.js

`npm install colors`
NPM va télécharger automatiquement la dernière version du module et il va la placer dans un sous-dossier node_modules. Vérifiez donc bien que vous êtes dans le dossier de votre projet Node.js avant de lancer cette commande !

(01SampleServer)

```
var colors = require('colors');
console.log('hello en vert'.green);
console.log('hello en rouge'.bold.red)
console.log('hello en reverse'.inverse);
console.log('Hello Arc en Ciel!'.rainbow);
```

Notre premier Serveur (02SampleServer)

```
var http = require('http');
var port = 8080;

var server = http.createServer(function(req, res) {
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello world!');
});
server.listen(port);

console.log('Le serveur répond sur http://127.0.0.1:' + port + '/');
```