Chapitre 1 : Introduction sur Nodejs

Objectifs :

- ✓ Découvrir et comprendre le fonctionnement de Node.js
- ✓ Installer et configurer un serveur Node.js

Prérequis : Avant d'entamer cette séquence, vous avez besoins d'une connaissance préalable en JavaScript et en HTML

1. Historique

Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge.

Elle utilise la machine virtuelle V8 et implémente sous licence MIT les spécifications CommonJS.

Parmi les modules natifs de Node.js, on retrouve http qui permet le développement de serveur HTTP. Il est donc possible de se passer de serveurs web tels que **Nginx** ou Apache lors du déploiement de sites et d'applications web développés avec Node.js.

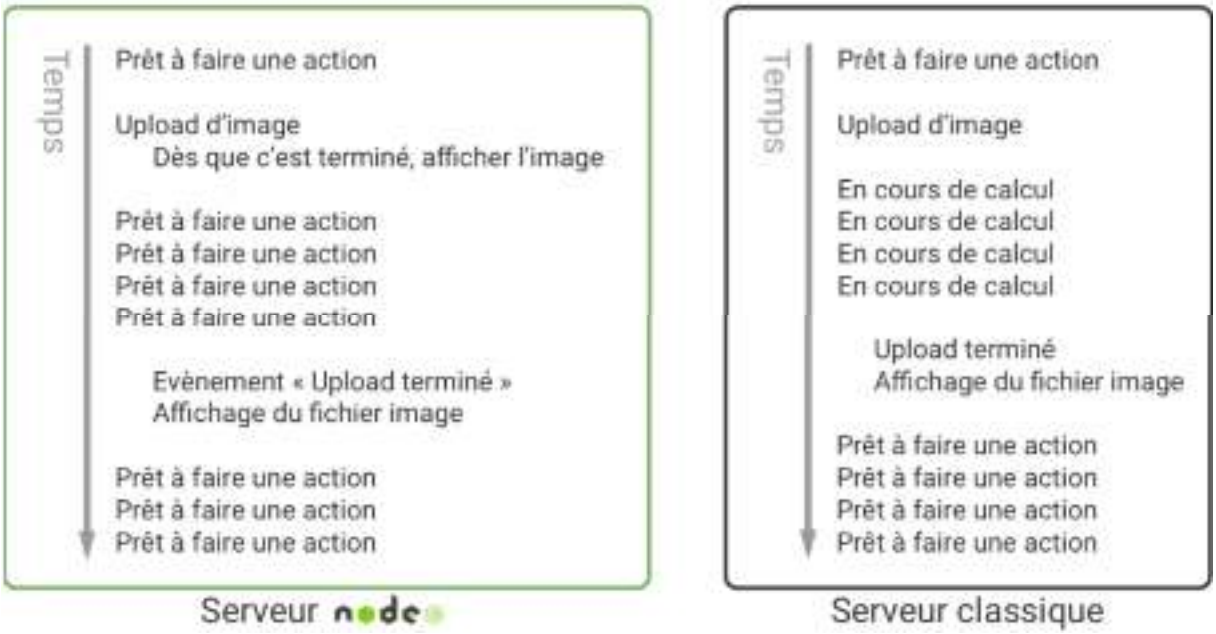
Concrètement, Node.js est un environnement bas niveau permettant l'exécution de JavaScript côté serveur.

Node.js est utilisé notamment comme plateforme de serveur Web, elle est utilisée par Groupon, Vivaldi, SAP, LinkedIn, Microsoft, Yahoo!, Walmart, Rakuten, Sage et PayPal.

2. C'est quoi NodeJS

Avant Node.js, JavaScript était uniquement utilisé pour le développement frontend (côté client). Il était nécessaire d'utiliser un autre langage de programmation pour la partie backend (côté serveur). En pratique, vous étiez obligé d'avoir deux équipes de développeurs, une pour le frontend et une autre pour le backend. Avec la popularité croissante de Node.js, le mythe du développeur fullstack est devenu réalité. Aujourd'hui, il est possible de coder les deux parties d'une application web en JavaScript. C'est un gain de temps pour le développeur et une économie d'argent pour l'entreprise.

Node.js est une plateforme logicielle libre en JavaScript intégrant un serveur HTTP. Son fonctionnement est basé sur une boucle événementielle lui permettant de supporter de fortes montées en charge.



L'utilisation de Node.js en tant que serveur web permet de traiter un gros volume de requêtes simultanément de manière efficace. Cette performance élevée s'explique par une conception asynchrone (modèle non bloquant) permettant d'éviter les attentes. Ainsi, plusieurs requêtes peuvent être lancées en parallèle. Les résultats sont traités au fil de l'eau.

Node utilise le compilateur JavaScript V8 de Google focalisé sur les performances et la sécurité. A l'origine, la machine virtuelle V8 a été créée pour interpréter le JavaScript dans le navigateur Chrome. Le moteur V8 est et restera à la pointe de la technologie. Par ricochet, les progrès de V8 impactent directement Node.js.

En résumé, Nodejs présente les spécificités suivantes :

built on Chrome's V8 JavaScript engine : Afin d'interpréter le code que l'on va écrire en JavaScript NodeJS se repose sur le **moteur V8** qui équipe actuellement Chrome. Ce moteur a été amélioré grâce à la concurrence entre les navigateurs et permet de créer un script avec un langage familier tout en gardant un temps d'exécution optimal.

event-driven model : Le code que l'on va écrire va être basé sur un système d'évènement. Les objets que l'on va créer émettront des évènements lors de leur cycle de vie. Il sera ensuite possible de souscrire et d'écouter ces évènements afin d'effectuer des opérations spécifiques lorsqu'ils sont émit. C'est une méthode qui ressemble à celle que l'on utilise actuellement côté navigateur avec la méthode addEventListener

non-blocking I/O model : Lors du déroulement d'un script il y a souvent des phases "d'attente" ou le script bloque en attendant une entrée ou une sortie. Pendant ce temps d'attente, le script ne fais rien et ne peut traiter de nouvelles tâches. Au sein de NodeJS la plupart de ces entrée/sorties vont se dérouler de manière asynchrone grâce à la librairie **libuv**. Ceci permet de gérer plus de concurrences en évitant les phases d'attentes.
Node.js' package ecosystem : **NodeJS** dispose de son gestionnaire de paquet officiel **NPM** qui permettra de télécharger et de partager des librairies. Il dispose d'une communauté très importante et d'un très grand nombre de paquets.

NB : **NodeJS** n'est pas un framework. Ce n'est pas un outil qui vous permettra de mettre en place une application web rapidement avec peu de code. C'est un outil plus bas niveau qui vous permettra de communiquer avec le système à travers différentes librairies C++ et avec un langage familier. Comme vu précédemment, c'est un outil que l'on va sélectionner si on a besoin de gérer un grand nombre de demandes sur un seul thread en évitant les lenteurs dû à la nature synchrone d'autres langages.

3. Installation de NodeJS

3.1. Télécharger Node.js

Allez dans le site officiel de nodeJS : https://nodejs.org/en/download/current/

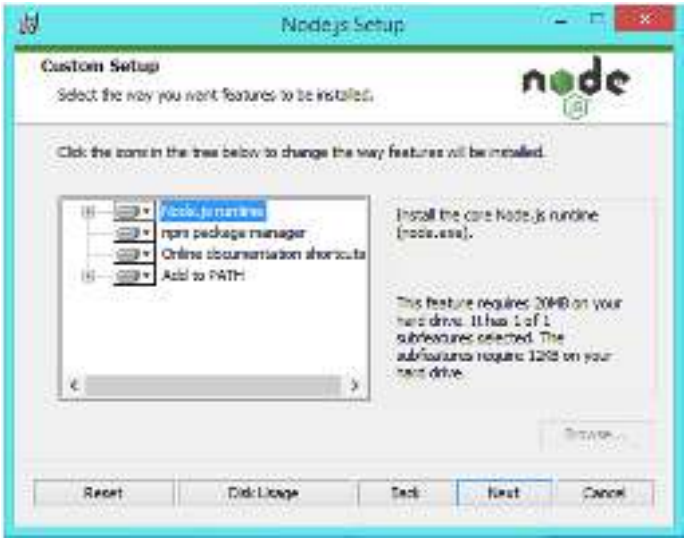
Depuis la page de téléchargement, vous devriez voir un bouton « Windows installer » (si vous êtes sur Windows). Cliquez dessus pour télécharger automatiquement le zip le plus adapté à votre système. Vous pouvez également choisir vous même votre zip dans Download pour obtenir au choix le Windows Installer (.msi) et/ou le Windows Binary (.exe) en 32/64 bit.

3.2. Intaller node et npm

Exécutez votre fichier

Cliquez sur suivant puis acceptez la licence et déposez-le dans "Program files".

Vous aurez l'interface suivante :



Ce package va vous installer :

L'exécuteur node.js : le programme permettant d'exécuter des fichiers **.js** (comme php.exe le ferait avec des .php).

Le module npm (Node Package Manager) : un gestionnaire de modules qui va vous permettre simplement d'ajouter et retirer les librairies dont vous aurez besoin pour vos applications (pas de surplus, seulement le nécessaire donc).

Un raccourci vers la documentation en ligne.

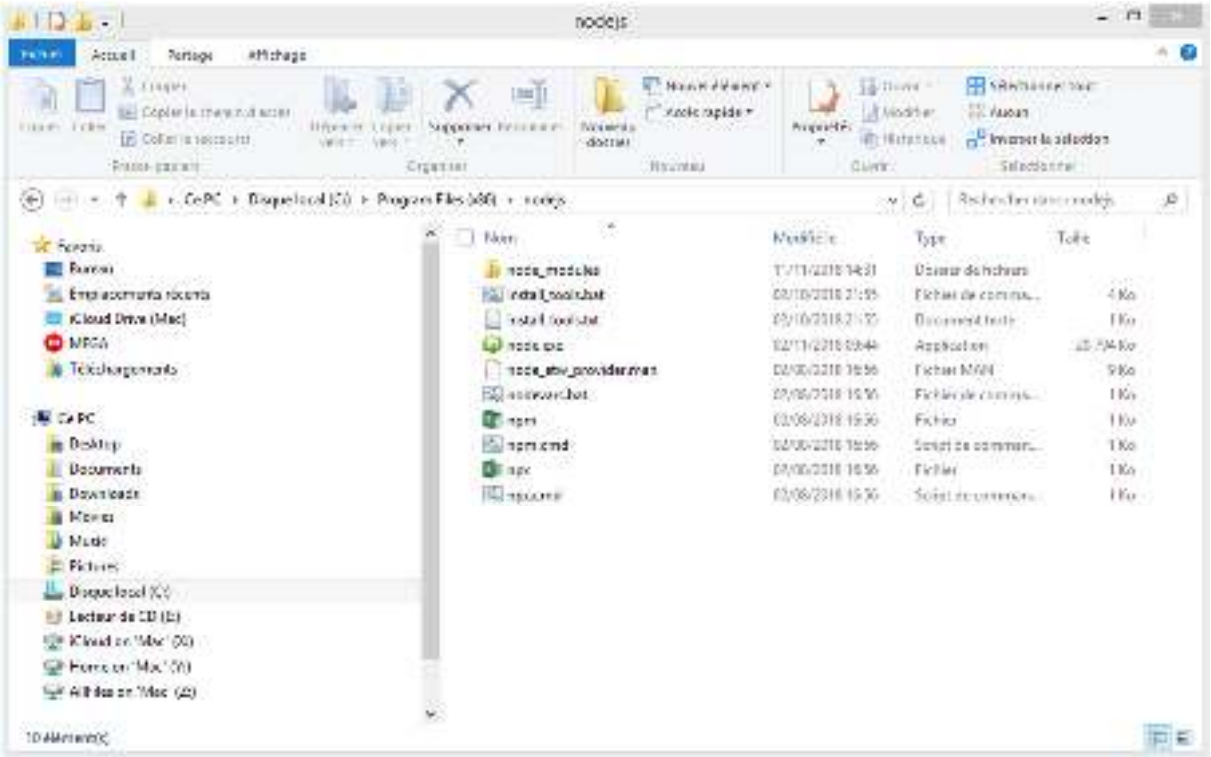
Des variables d'environnements : Ainsi vous pourrez exécuter les commandes **node** et **npm** dans votre invité de commande.

Cliquez sur **Next >Next** pour lancer officiellement l'installation de **NodeJs**

A la fin de l'installation vous aurez la fenêtre suivante :

3.3. Dossier de Node.js après installation

Quand l'installeur aura fini vous aurez un joli dossier contenant entre autre **node.js** et **npm.cmd** qui seront appelables depuis n'importe quel dossier avec les commandes **node** et **npm.**



4. Notre premier projet Nodejs

Juste après l'installation, tentons notre premier « **Hello World** ». Tout d'abord, créez le répertoire **projetsNodejs** dans « **Mes Document** », pour y mettre vos projet **nodeJS**.. Créez un fichier **hello-world.js** accessible à cette adresse sur mon poste : **C:\ projetsNodejs\hello-world.js**.
Dans le fichier hello-word.js, copiez et collez le code suivant.

```
var http = require('http');
var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Salut tout le monde !');
});
server.listen(8080);
```

Ile programme suivant crée un mini-serveur web qui renvoie un message "Salut tout le monde" dans tous les cas, quelle que soit la page demandée. Ce serveur est lancé sur le port 8080 à la dernière ligne.

Décomposition du code :

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

require effectue un appel à une bibliothèque de **Node.js**, ici la bibliothèque **"http"** qui nous permet de créer un serveur web. Il existe des tonnes de bibliothèques comme celle-là, la plupart pouvant être téléchargées avec **NPM**, le gestionnaire de paquets de Node.js.

La variable **http** représente un objet JavaScript qui va nous permettre de lancer un serveur web. C'est justement ce qu'on fait avec :

```
var server = http.createServer()
```

On appelle la fonction **createServer()** contenue dans l'objet http et on enregistre ce serveur dans la variable server. Vous remarquerez que la fonction createServer prend un paramètre... et que ce paramètre est une fonction ! C'est pour ça que l'instruction est un peu compliquée,

puisqu'elle s'étend sur plusieurs lignes :

```
var server = http.createServer(function(req, res) {
  res.writeHead(200);
  res.end('Salut tout le monde !');
});
```

Tout le code ci-dessus correspond à l'appel à **createServer().** Il comprend en paramètre **la fonction à exécuter quand un visiteur se connecte à notre application**.

Dans Node.js Il y a des fonctions de callback de partout, et en général elles sont placées à l'intérieur des arguments d'une autre fonction comme le code ci-dessus.

N'oubliez pas de bien fermer la fonction de callback avec une accolade, puis de fermer les parenthèses d'appel de la fonction qui l'englobe, puis de placer le point-virgule. C'est pour ça que vous voyez les symboles});à la dernière ligne du code.

La fonction de callback est donc appelée à chaque fois qu'un visiteur se connecte à notre site. Elle prend 2 paramètres :

- ✓ **La requête du visiteur (reqdans mes exemples) :** cet objet contient toutes les informations sur ce que le visiteur a demandé. On y trouve le nom de la page appelée, les paramètres, les éventuels champs de formulaires remplis...
- ✓ **La réponse que vous devez renvoyer (resdans mes exemples) :** c'est cet objet qu'il faut remplir pour donner un retour au visiteur. Au final, **res** contiendra en général le code HTML de la page à renvoyer au visiteur.

Dans la fonction callback on a les instruction suivantes :

```
  res.writeHead(200);
  res.end('Salut tout le monde !');
```

On renvoie le code **200** dans l'en-tête de la réponse, qui signifie au navigateur **"OK tout va bien"** (on aurait par exemple répondu **404** si la page demandée n'existait pas). Il faut savoir qu'en plus du code HTML, **le serveur renvoie en général tout un tas de paramètres en en-tête**. Il faut connaître la norme HTTP qui indique comment clients et serveurs doivent

communiquer pour bien l'utiliser. Ce qui est dûe au fait que **Node.js est bas niveau**.

Ensuite, on termine la réponse **(avec end())** en envoyant le message de notre choix au navigateur. Ici, on n'envoie même pas de HTML, juste du texte brut.

Enfin, le serveur est lancé et "écoute" sur le port 8080 avec l'instruction :

```
server.listen(8080);
```

NB : On évite d'utiliser ici le port **80** qui est normalement réservé aux serveurs web, car celui-ci est peut-être déjà utilisé par votre machine. Ce port **8080** sert juste pour nos tests évidemment, une fois en production il est conseillé au contraire d'écouter cette fois sur le port **80** car c'est à cette porte (à ce port) que vos visiteurs iront taper en arrivant sur votre serveur.

Lancement de notre code

Pour tester votre premier serveur, rendez-vous dans la console et tapez puis allez dans le répertoire ou se trouve votre projet(« **hello-world.js** ») .

Ensuite lancez la commande suivante :

```
node hello-word.js
```

Avec votre navigateur allez su le lien : http://localhost:8080

Vous verrez le message « **Salut tout le monde** »

5. Retourner du code HTML

Nous avons créé notre première application avec son serveur web embarqué. Mais l'application est pour l'instant minimaliste :

- ✓ Le message renvoyé est du texte brut, il ne comporte même pas de HTML !
- ✓ L'application renvoie toujours le même message, quelle que soit la page appelée (http://localhost:8080,http://localhost:8080/mapage, http://localhost:8080/dossier/autrepage)

Il y a des règles à respecter entre le client et le serveur. Ils communiquent en se basant sur la norme HTTP inventée par Tim Berners-Lee. Cette norme est à la base du Web (tout comme le langage HTML qui a aussi été inventé par ce même monsieur ;)).

La norme HTTP dit que le serveur doit indiquer le type de données qu'il s'apprête à envoyer au client. Un serveur peut renvoyer différents types de données :

- ✓ Du texte brut : text/plain
- ✓ Du HTML : text/html
- ✓ Du CSS : text/css
- ✓ Une image JPEG : image/jpeg
- ✓ Une vidéo MPEG4 : video/mp4
- ✓ Un fichier ZIP : application/zip
- ✓ etc.

Ce sont ce qu'on appelle les types MIME. Ils sont envoyés dans l'en-tête de la réponse du serveur.

Dans notre premiere application Nous avons seulement indiqué le code de réponse 200 qui signifie "OK, pas d'erreur". Nous devons rajouter un paramètre qui indique le type MIME de la réponse. Pour HTML, ce sera donc :

```
res.writeHead(200, {"Content-Type": "text/html"});
```

Au final, notre code ressemble donc maintenant à ceci :

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

var server = http.createServer(function(req, res) {
    res.writeHead(200, {"Content-Type": "text/html"});
    res.end('<p>Voici un paragraphe <strong>HTML</strong> !</p>');
});
server.listen(8080);
```

Jusqu'ici, nous avions toujours écrit le code HTML dansres.end(). Pour mieux découper le code,

on utilise la commande **res.write()** qui permet d'écrire la réponse en plusieurs temps. Ca revient au même, mais notre code est mieux découplé comme ça.

res.end() doit toujours être appelé en dernier pour terminer la réponse et faire en sorte que le serveur envoie le résultat au client.

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

var server = http.createServer(function(req, res) {
   res.writeHead(200, {"Content-Type": "text/html"});
   res.write('<!DOCTYPE html>'+
'<html>'+
  ' <head>'+
    ' <meta charset="utf-8" />'+
    ' <title>Ma page Node.js !</title>'+
    ' </head>'+
   '<body>'+
       '<p>Voici un paragraphe <strong>HTML</strong> !</p>'+
   '</body>'+
'</html>');
   res.end();
});
server.listen(8080);
```

Il existe des moyens de séparer le code HTML du code JavaScript : ce sont les systèmes de templates.

6. Déterminer la page appelée et les paramètres

Dans une application web, Il faut qu'on sache quelle est la page demandée par le visiteur. Pour l'instant, vu qu'on ne fait aucun test, notre application renvoie toujours la même chose.

Pour récupérer la page demandée par le visiteur, on va faire appel à un nouveau module de Node appelé **"url".** On demande son inclusion avec :

```
var url = require("url");
```

Ensuite, il nous suffit de "parser" la requête du visiteur comme ceci pour obtenir le nom de la page demandée :

```
url.parse(req.url).pathname;
```

Voici un code très simple qui nous permet de tester ça :

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
   var page = url.parse(req.url).pathname;
   console.log(page);
   res.writeHead(200, {"Content-Type": "text/plain"});
   res.write('Bien le bonjour');
   res.end();
});
server.listen(8080);
```

Une fois ce scripte exécuté, si on entre l'adresse **http://localhost:8080/accueil** on doit voir le résultat suivant sur la console:

```
/accueil
```

Maintenant pour envoyé un résultat en fonction de la page qui est demandée, il suffit juste d'utiliser une condition comme le montre l'exemple suivant :

```
var http = require('http');
var url = require('url');

var server = http.createServer(function(req, res) {
   var page = url.parse(req.url).pathname;
   console.log(page);
   res.writeHead(200, {"Content-Type": "text/plain"});
   if (page == '/') {
      res.write('Vous êtes à l\'accueil, que puis-je pour vous ?');
   }
   else if (page == '/sous-sol') {
      res.write('Vous êtes dans la cave à vins, ces bouteilles sont à moi !');
   }
   else if (page == '/etage/1/chambre') {
      res.write('Hé ho, c\'est privé ici !');
   }
   res.end();
});
server.listen(8080);
```

Récupération des paramètres de la requête

Les paramètres sont envoyés à la fin de l'URL, après le chemin du fichier. Prenez cette URL par exemple :

http://localhost:8080/page?prenom=Robert&nom=Dupont

Les paramètres sont contenus dans la chaîne **?prenom=Robert&nom=Dupont**. Pour récupérer cette chaîne, il suffit de faire appel à:

```
var params=url.parse(req.url).query
```

Le problème, c'est qu'on vous renvoie toute la chaîne sans découper au préalable les différents paramètres. Heureusement, il existe un module **Node.js** qui s'en charge pour nous **: querystring.**

Vous devez inclure ce module :

```
var querystring = require('querystring');
```

Vous pourrez ensuite faire :

```
var params = querystring.parse(url.parse(req.url).query);
```

Vous disposerez alors d'un tableau de paramètres **"params"**. Pour récupérer le paramètre **"prenom"** par exemple, il suffira d'écrire :**params['prenom']**

Exemple : Une application qui affiche le prenom et le nom donnés en paramètre.

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');

var server = http.createServer(function(req, res) {
   var params = querystring.parse(url.parse(req.url).query);
   res.writeHead(200, {"Content-Type": "text/plain"});
   if ('prenom' in params && 'nom' in params) {
      res.write('Vous vous appelez ' + params['prenom'] + ' ' + params['nom']);
   }
   else {
      res.write('Vous devez bien avoir un prénom et un nom, non ?');
   }
   res.end();
});
server.listen(8080);
```

Références

https://www.grafikart.fr/formations/nodejs/nodejs-intro

https://makina-corpus.com/blog/metier/2014/introduction-a-nodejs

https://blog.lesieur.name/installer-et-utiliser-nodejs-sous-windows/

http://wdi.supelec.fr/appliouaibe/Cours/JSserveur