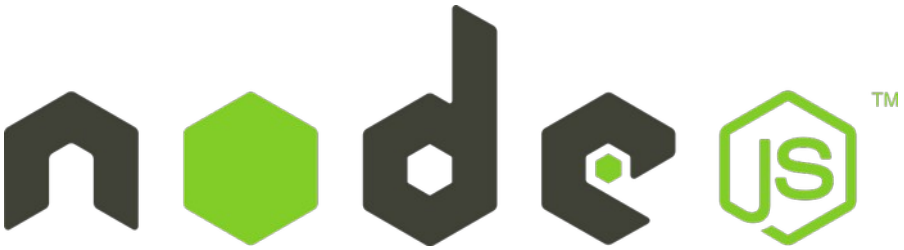


Node.JS

date	révision
juillet 2018	Création (v.40)
02/07/2019	Nombreux ajouts (GET/POST, protocole HTTP...)
27/09/2019	Corrections : p10 (Console.log() manquants), chemin fichier ./, p.25 (nodemon)
29/09/2019	Ajout des statistiques sur Node.JS

SOMMAIRE

NODE.JS

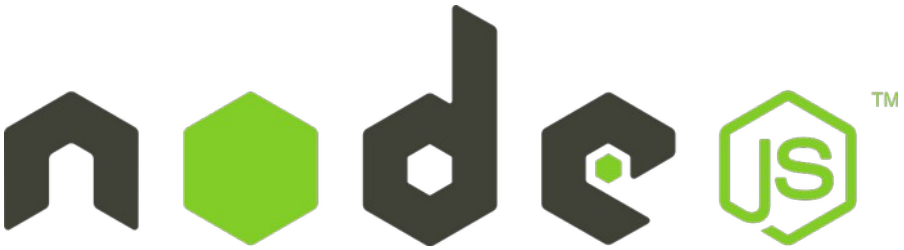
L'activité proposée ici aborde l'utilisation d'un environnement de développement asynchrone.

À l'issue de cette activité, l'étudiant devra :

- Comprendre la programmation asynchrone
- Expliquer un code JavaScript dans Node.JS
- Décrire le fonctionnement d'une application serveur sous Node.JS
- Être capable de créer une petite application dans cet environnement

 **Durée**	 **Difficulté**	 **Taxonomie Bloom**
3 séances	❶ ❷ ❸ ④	Compréhension – Application

1 SÉANCE 1 : NODE.JS



Lors de cette séance, vous allez découvrir Node.JS et son fonctionnement :

- Comment fonctionne Node.JS
- Comprendre et Utiliser les fonctions anonymes et fonctions fléchées
- Pratiquer quelques exemples Node.JS

1.1 INTRODUCTION

Node.JS est un **serveur JavaScript** qui a rapidement évolué, la version actuelle étant la 8.

Il est très utilisé par des entreprises comme Paypal, LinkedIn, Uber ou même la NASA. Ce framework créé par Ryan DAHL en 2009 s'appuie sur JavaScript mais surtout, permet un développement dit "Fullstack JS" car il permet d'écrire du code coté serveur comme coté client.

Son fonctionnement est basé sur les événements et est donc asynchrone, ce qui est non-bloquant : il ne faut pas confondre avec multithread, qui est un fonctionnement différent.

1.2 INTÉRÊT DE NODE.JS

Node.JS fonctionne différemment de PHP ou des langages comme Java, C# car il est asynchrone : lorsqu'un fichier disponible, il existe un événement qui déclenchera l'exécution d'un code Node.JS à la fin du chargement… en attendant cet événement, Node.JS peut traiter d'autres tâches !

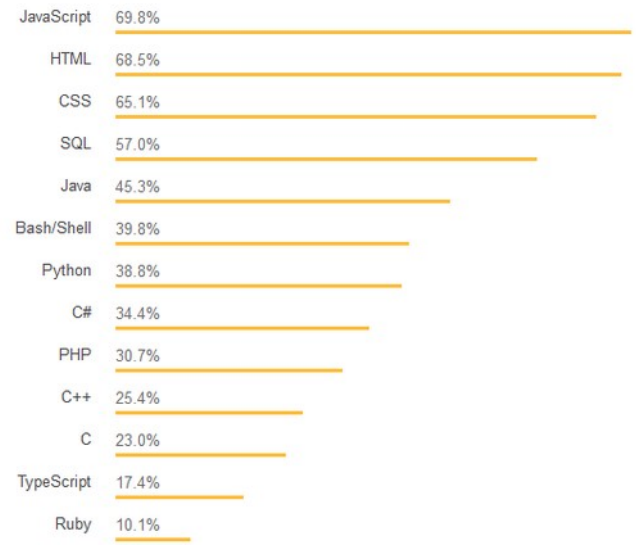
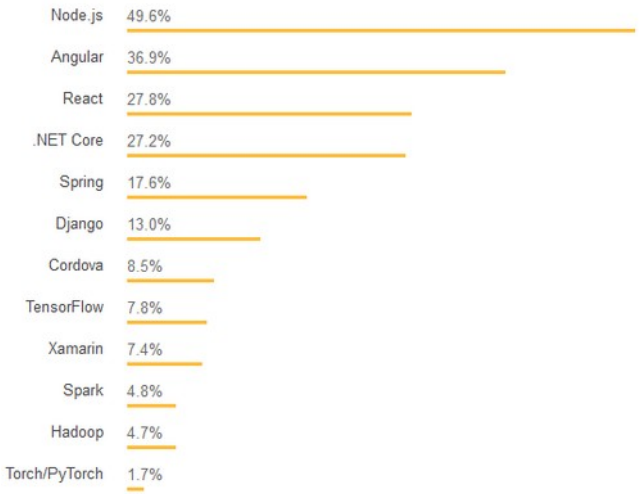
Un fichier Node.JS contient des tâches à lancer lorsque certains événements surviennent : accès à un port de communication par exemple.

Grâce à cela, Node.JS est très performant face à PHP. Le tableau ci-dessous montre la rapidité de traitement en seconde des deux langages :

Performance		
Number of iterations	**Node.JS**	**PHP**
100	2.00	0.16
10,000	3.00	9.52
1,000,000	13.00	1117.21
10,000,000	138.00	10461.29

https://www.quora.com/Is-it-better-to-learn-NodeJS-or-PHP-I-am-an-engineer-but-web-dev-hobbyist-I-am-struggling-to-choose-back-end-technology-PHP-or-NodeJS-What-is-better-in-terms-of-getting-a-job-in-the-future

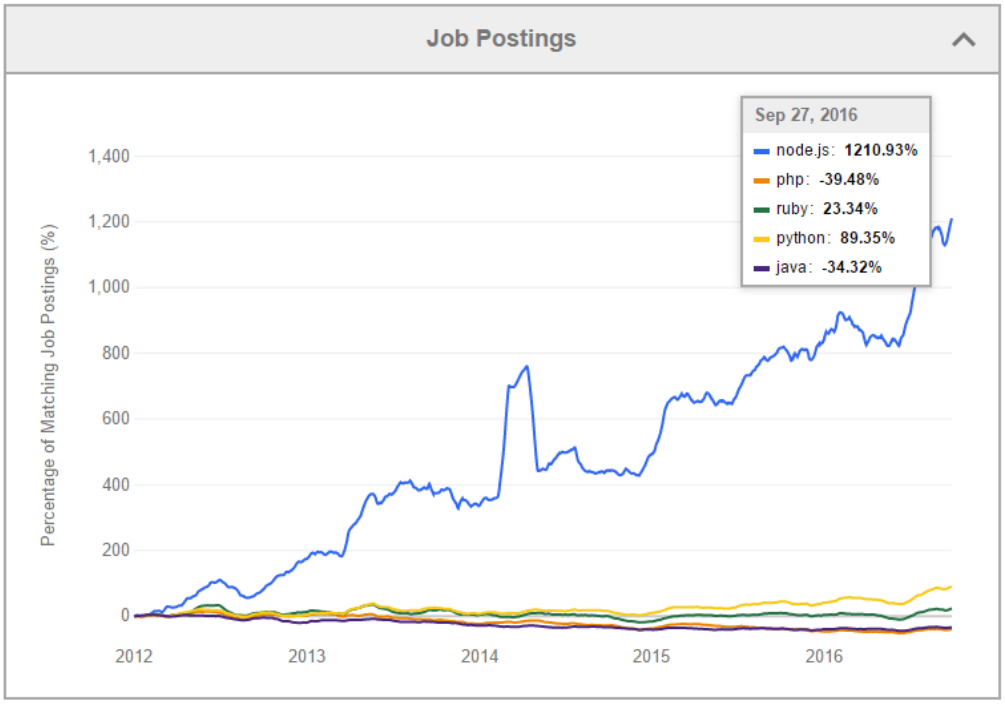
La communauté des développeurs s'intéresse énormément à cette technologie, comme le montre l'étude menée chaque année par StackOverflow.

Technologies populaires	Les framworks populaires
	

l'étude complète est disponible ici :

https://insights.stackoverflow.com/survey/2018/

Il y a également un engouement important autour de cette technologie, ce graphique du site Indeed représente la quantité d'offres d'emplois par technologies (https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/) :

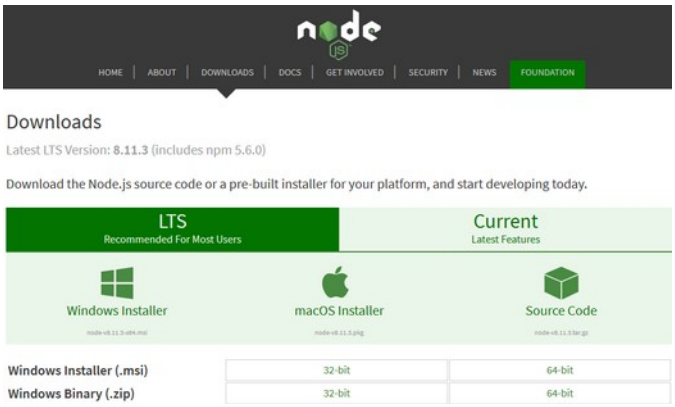


1.3 INSTALLATION DE NODE.JS

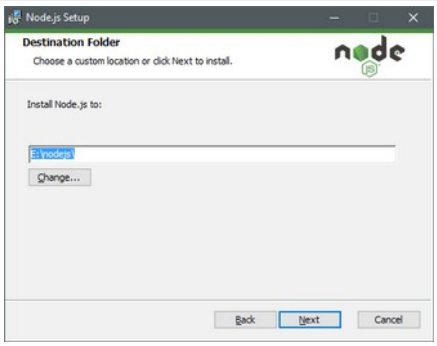
Node.JS est disponible en plusieurs versions sur le site https://Node.JS.org/en/download/

La version retenue est Windows Installer (.msi) en 64 bit. Préférez la version LTS[1].

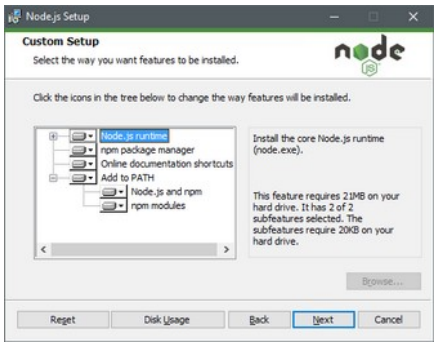
Téléchargez et installez cette version.

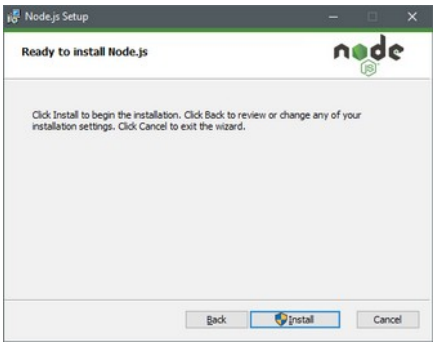


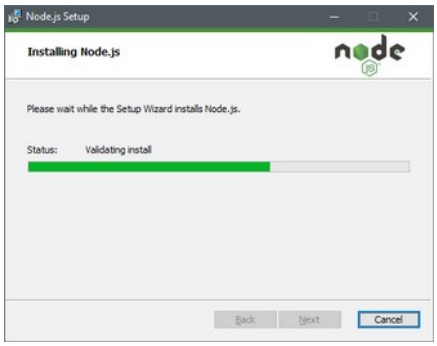
1.3.1 Options d'installation



	Cochez la licence	Choisir le chemin (dans mon cas, e:\Node.JS)

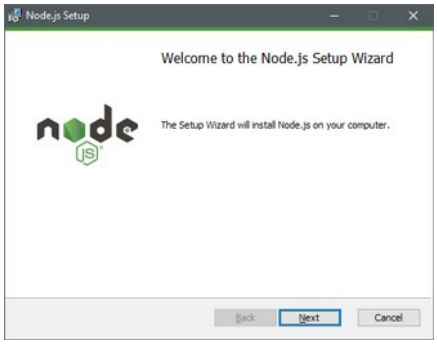


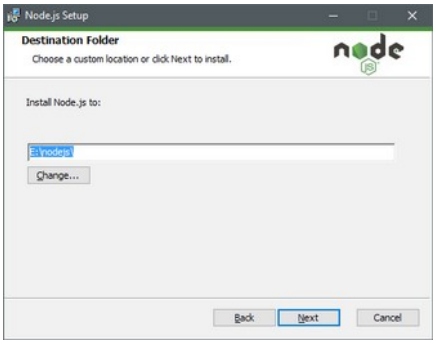




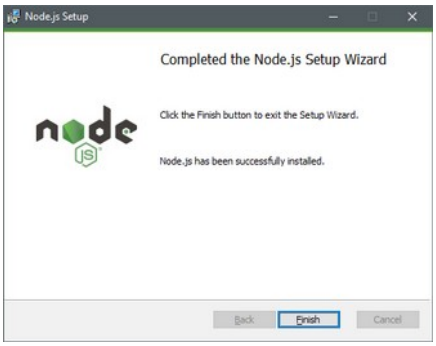
Vérifiez que npm package		

1 *Long Time Support*





manager est installé par défaut



Note : Une installation sous Linux se fait avec le gestionnaire de paquetage APT (slon le système Linux)

```
curl-sL https://deb.nodesource.com/setup_8.x |sudo-Ebash-
sudo apt install -y Node.JS
```

1.3.2 tester l'installation : "Hello world"

Le premier essai pour valider que le Node.JS est bien installé, est de créer un serveur web. Pour cela :

1. Dans un éditeur de texte, copiez le code suivant et enregistrez-le sous le nom "monApp.js"
2. Dans une invite de commande dans le répertoire de votre fichier, tapez "`node monApp.js`"
3. Ouvrez un page de navigateur sur http://127.0.0.1:3000/

Voici le code :

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('<p>Hello World</p>\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Utilisation d'un module de gestion HTTP

Création du serveur

Fonction anonyme avec 2 paramètres : req (in) et res (out)

Construction page de retour

Démarrage du serveur

Que fait ce code ?

Il crée un objet *server* qui lorsqu'il recevra une requête, renverra un texte contenant "Hello World".

Dans un environnement Visual Studio Code (l'éditeur rapide de Microsoft), il suffit de lancer le débogage (touche **F5**).

> *Pour ceux qui utilisent Notepad++ pour éditer leurs scripts, le lien suivant permet de configurer plus facilement Notepad++ pour lancer les scripts sous Node.JS.*
>
> *http://blog.aguskurniawan.net/post/notepadjs.aspx*

Notez que le script reste actif tant que l'exécution n'est pas interrompue (CTRL+C pour Notepad++ ou Shift+F5 pour Visual Studio Code). Si vous y apportez des modifications, il faut redémarrer le serveur Node.JS.

1.4 Fonctionnement Node.JS

Pour le moment, nous n'avons fait que copier un script, sans en comprendre le détail.

Voici maintenant quelques informations pour travailler avec Node.JS.

1.4.1 JavaScript sur le serveur

Node.JS utilise JavaScript comme langage de programmation.

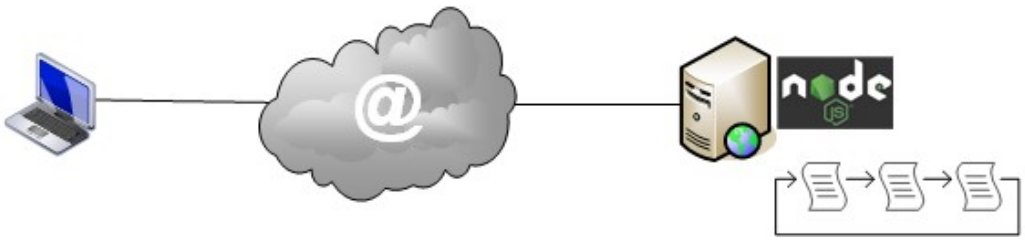
Souvent décrié, car il servait principalement à ajouter des effets dans les pages web, JavaScript est pourtant un véritable langage. Pour être exact, JavaScript est une implémentation du standard de langage ECMAScript[2] par Brendand Eich. Inventé dans les années 1995 pour le navigateur Netscape, c'est Google qui lui redonne ses lettres de noblesses dans les années 2010 avec son navigateur Chrome.

Cependant, à la différence des frameworks comme jQuery, Node.JS exécute JavaScript côté serveur !

En résumé, Node.JS remplace Apache et PHP.[3]

Mais la force de JavaScript et Node.JS réside dans le fonctionnement événementiel : là ou PHP reste séquentiel, Node.JS est optimisé pour ne pas perdre de temps à attendre sans rien faire.

Cependant, il ne peut y avoir qu'un serveur à la fois : une fois actif (en écoute sur un port), chaque tâche sera exécutée en 'callback', c'est-à-dire, que lorsque un événement survient, il activera une fonction (dite 'callback') qui affichera un résultat lorsque la tâche sera terminée.



Le cours de SI6 en JavaScript s'applique donc ici pour les instructions habituelles (boucles, conditions, variables, etc.)

En revanche, les spécificités de Node.JS sont expliquées dans ce support.

2 *ECMA International : https://www.ecma-international.org/*
3 *En réalité, il faut développer un serveur web pour remplacer Apache, c'est ce que fait le programme donné.*

1.4.1.1 Require et export

L'utilisation de **require en JavaScript** signifie la même chose que **import en Java** ou **using en C#** : simplement, le résultat du chargement du module se fait dans une variable.

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

Signifie que votre script utilisera les bibliothèques du module "http".

1.4.1.2 Modules standards

Il est bien sûr possible d'utiliser d'autres modules :

```
const connexion = require('net');
```

ce module permet de communiquer en utilisant le protocole TCP.

Les modules les plus fréquemment utilisés sont ***buffer*** (permet de gérer des flux de données brutes), ***cluster*** (permet de créer des processus enfants qui partagent les mêmes ports), ***crypto*** (qui sert dans les chiffrements et déchiffrements ainsi que les fonctions de vérification ou hachage), ***dns*** (pour effectuer des requêtes sur les résolutions de noms), ***events*** (pour la gestion des émetteurs d'événements), ***fs*** (qui signifie filesystem et dont l'objet est le traitement des fonctions POSIX sur les fichiers/dossiers), ***http***/***https*** (supportent les messages du protocole HTTP et HTTPS), ***net*** et ***dgram*** (supporte les connexions TCP et UDP), ***querystring*** (l'analyse des requêtes d'URL), ***readline*** (lire un flux ligne par ligne) et de nombreuses autres.

- Voici la liste des principaux modules fournis avec Node.JS : https://www.w3schools.com/nodejs/ref_modules.asp

1.4.1.3 Export : modules personnels

Il est possible et recommandé d'écrire ses propres modules dans des fichiers séparés.

Le fichier qui appelle votre module doit utiliser l'instruction 'require' avec le chemin d'accès du fichier :

```
const bonjour = require('./print.js');
```

Le code contenu dans votre module print.js devra cependant comporter une instruction **export**.

```
module.exports = {
  sayHelloInEnglish: function() {
    return "HELLO";
  },

sayHelloInSpanish: function() {
    return "Hola";
  }
};
```

et le code principal qui utilise le module devient :

```
// main.js
var greetings = require("./print.js");

// "Hello"
Console.log(greetings.sayHelloInEnglish());

// "Hola"
Console.log(greetings.sayHelloInSpanish());
```

Il faut toujours préciser le chemin ./ dans Node.JS, le . Signifiant la racine de l'application.

Comme pour JavaScript, il est possible d'exporter une fonction fonctionnant comme une classe :

index.js

```
var person = require('./Person.js');
var person1 = new person('James', 'Bond');
console.log(person1.fullName());
```

person.js

```
module.exports = function (firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.fullName = function () {
        return this.firstName + ' ' + this.lastName;
    }
}
```

1.4.1.4 Module Mysql : accéder à une base MySQL

Démarrez un serveur mySQL avant de tester les codes qui suivent.

Après avoir requis la dépendance mysql (qui devra être installé avec **npm**), il est possible de créer une connexion, comme suit :

```
// On utilise le module node-mysql
var mysql = require('mysql');

var client= mysql.createConnection({
  host     : 'localhost',
  user     : 'root',
  password : 'motdepasse',
  database : 'mabase',
});
```

Mettre ici les paramètres du compte de la base de données.

Par la suite, on peut utiliser la connexion pour se connecter, comme ci-dessous :

```
// On se connecte à la base de données.
client.connect((err) => {
  if (!err) { console.log("connexion réussie"); }
  else { console.log("connexion echoué !\n Erreur : " + JSON.stringify(err));
});
```

Et il est important de noter que JavaScript gère les objets, ainsi, les fonctions s'appliquent sur une variable correspondant à un module. L'objet contenant les éléments nécessaires pour se connecter, la fonction connect() n'a ici aucun paramètre.

L'envoi d'une requête et la récupération des tuples sera écrit comme suit :

```
connection.query('SELECT * from matable', function(err, rows, fields) {
  if (!err)
    console.log('The solution is: ', rows);
  else
    console.log('Error while performing Query.');
});
```

Cette fois, on passe une variable (requête SQL sous forme d'une chaîne) en paramètre… et une fonction anonyme (ou un objet contenant les attributs err, rows et fields).

- Pour passer le proxy, il peut être nécessaire de modifier la configuration de npm :
  `npm config set proxy …`
  voir les annexes, proxy

Vous pouvez utiliser WAMP ou EasyPHP (ou directement MySQL) pour créer les données suivantes :

id	name	address	email	phone
1	NORRIS Chuck	3 rue des rondins de bois 3750 NORRISTOWN	god@norristown.chuck	555-0000
2	KENT Clark	Daily Planet	ckent@daily-planet.biz	555-1697
3	PARR Hélène	5569, Indestructible street 6060 GREATTOWN	elastigirl@indestructible.com	555-7998
4	DE LA VEGA Don Diego	hacienda del signor Don Alexandro (chez papa quoi)	z@or.ro	inexistant

les commandes du fichier SQL sont en annexe Erreur : source de la référence non trouvée Erreur : source de la référence non trouvée.

Voici un exemple d'utilisation détaillé de l'accès à cette table avec Node.JS, fichier **mySQLExample1.js** :

```
let mysql = require('mysql');

let cnx = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "root",
  database: "mabase"
});

cnx.connect(function(err) {
  if (err) throw err;
  cnx.query("SELECT name, address FROM maTable;", function (err, result, fields) {
    if (err) throw err;

      console.log(result);                    // doit être la liste
      console.log('======================');
      let rows = JSON.parse(JSON.stringify(result[0]));
      console.log(rows);                      // écrire ligne 0 format JSON
      console.log('======================');
      for (res of result) {
            console.log(res.name);            // écrire les noms
      }
      console.log('======================');
      cnx.end(function(err) {
            if (err) throw err;
      });
  });
});
```

Enfin, n'oubliez pas d'installer le module MySQL avec npm :

```
npm install --save mysql
```

Puis exécutez le code :

```
node mySQLExample1.js
```

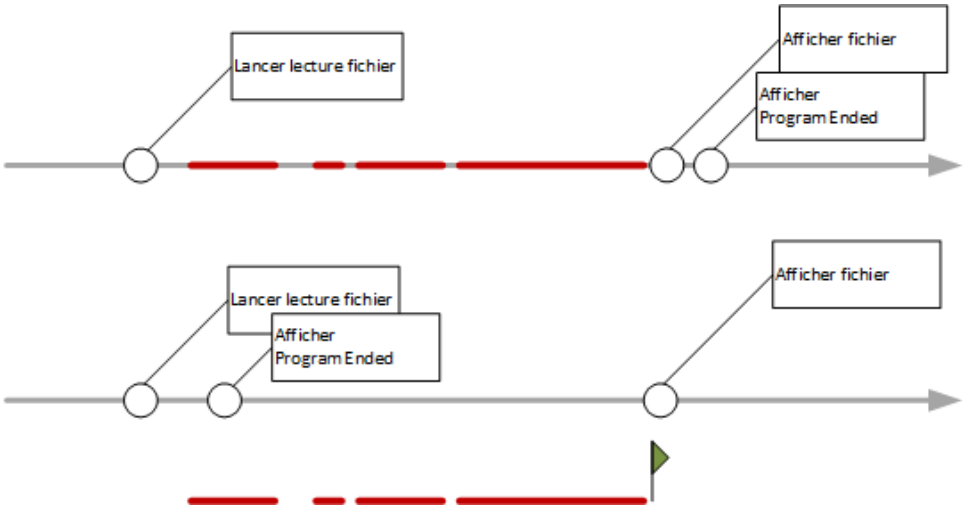
1.4.1.5 Particularité de l'asynchronisme

Nous avons vu que les fonctions peuvent prendre plusieurs formes en JavaScript. Cependant, Node.JS les exploite de manières asynchrones. Elles ne sont appelées que lorsqu'un événement survient. Programmer en Node.JS implique de dire à Node.JS quelle fonction appeler lorsqu'un événement survient.

Prenons un exemple concret en créant un fichier readme.txt et en le lisant de deux manières différentes en Node.JS. *Il est recommandé de tester les 2 exemples* :

Code synchrone	Code asynchrone
`var fs = require("fs");` `// lire et traiter` `var data = fs.readFileSync('readme.txt');` `console.log(data.toString());` `console.log("Program Ended");`	`var fs = require("fs");` `// traiter lorsque la lecture est terminée` `fs.readFile('readme.txt', function (err, data) {` `  if (err) return console.error(err);` `  console.log(data.toString());` `});` `console.log("Program Ended");`
`$ node main.js` *<le contenu du fichier texte>* `Program Ended`	`$ node main.js` `Program Ended` *<le contenu du fichier texte>*

Dans le premier cas, le programme attend que le fichier soit lu pour continuer. Dans le deuxième cas, c'est l'événement qui détermine la fin.



Il se cache derrière cette notion de callback, la notion d'événement (en anglais, stream) : dans un cas, il est nécessaire d'atteindre la fin du flux pour poursuivre l'exécution du programme ; dans le second cas, on indique à Node.JS qu'il lance la lecture du flux et continue le programme. Lorsque l'événement “fin de flux” survient, Node.JS affiche le résultat (qu'importe l'endroit du programme où il se trouve).

1.4.2 Interactivité des requêtes

Jusqu'à maintenant, nous n'avons fait qu'écrire une information figée ("Hello World"). L'objectif est toutefois de savoir ce que veut faire le visiteur.

1.4.2.1 Propriété url

L'utilisation de la propriété url sur la requête permet de déterminer le chemin de l'URL demandé :

```
let http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("je dois afficher l'application sur la page "+req.url);
  res.end();
}).listen(8080);
```

sur l'objet req (qui représente la requête transmise au navigateur), on retrouve les propriétés contenues dans l'URL.

Ainsi, interroger la page http://localhost:8080/exemple affichera

```
je dois afficher l'application sur la page /exemple
```

1.4.2.2 méthode url.parse

Nous pouvons aussi récupérer les informations transmises par la méthode GET ou POST dans l'URL. Pour cela, utilisons deux fichiers, un petit formulaire HTML et le code Node.JS.

code HTML

```
<html>
 <body>
  <h1>Formulaires POST et GET</h1>
  <form method="GET" action="/">
    nom : <input type="text" name="nom">
    age : <input type="text" name="age">
    <input type="submit" value="Envoi (post)">
  </form>

  <form method="POST" action="/">
    identifiant : <input type="text" name="login">
    mot de passe : <input type="text" name="password">
    <input type="submit" value="Envoi (get)">
  </form>

 </body>
</html>
```

Code Node.JS

```
let http = require('http');
let url  = require('url');
const { parse } = require('querystring');
let txt = "";

let serveur = http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  txt = "";
  if (req.method == "GET") {
    // avec GET, les données sont dans l'URL
    console.log("Appel par méthode GET");
      let q = url.parse(req.url, true).query;
      txt = "<strong>nom : </strong>"+q.nom + "<br>\n<strong>age : </strong>" + q.age;
      res.end(txt);
  }
  else if (req.method == "POST") {
    // avec POST, il faut lire les données
    console.log("Appel par méthode POST");
      // request.on : on bind an event with an action
      let body="";
      req.on('data', chunk => {
       body = body + chunk.toString();
    });
      req.on('end', () => {
            console.log(parse(body));      // format JSON
      let txt = "login : "+parse(body).login+"<br>\npwd : "+parse(body).password;
            res.end(txt);

      });
  }
  console.log('------');
});

serveur.listen(8080);
```

Le code Node.JS réagit aux deux méthodes (GET et POST), toutefois, les méthodes de lecture sont différentes : dans le premier cas, il suffit de "parser" l'URL, tandis que dans le second cas, il faut récupérer la zone file data.

La lecture des données pouvant être lente, on utilise deux événements, que l'on associe à deux actions différentes :

- 'data' ► à la réception de données, on place les données dans l'objet chunk puis on convertit chunk en texte que l'on ajoute à la variable txt,
- 'end' ► lorsque l'événement de fin de fichier survient, on écrit le résultat dans la page de réponse.

1.4.2.3 Usage de Wireshark

Bien que **Wireshark**↗ soit considéré comme un outil pour les gens "réseaux", il s'avère être un outil pratique pour les développeurs, depuis l'usage de la librairie **npcap** au lieu de winpcap. NPCAP, car il peut désormais lire les flux de l'interface locale (IPv4 127.0.0.1 ou Ipv6 ::1).

tcp.port == 8080

No.	Time	Source	Destination	Protocol	Length	Info
48	1.972148	::1	::1	HTTP	414	GET /?nom=david&age=46 HTTP/1.1
49	1.972163	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=1 Ack=341 Win=2618880 Len=0
62	1.989387	::1	::1	HTTP	232	HTTP/1.1 200 OK (text/html)
63	1.989413	::1	::1	TCP	74	61526 → 8080 [ACK] Seq=341 Ack=159 Win=2618624 Len=0
98	2.033585	::1	::1	HTTP	329	GET /favicon.ico HTTP/1.1
99	2.033603	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=159 Ack=596 Win=2618624 Len=0
106	2.034225	::1	::1	TCP	74	61526 → 8080 [FIN, ACK] Seq=596 Ack=159 Win=2618624 Len=0
107	2.034240	::1	::1	TCP	74	8080 → 61526 [ACK] Seq=159 Ack=597 Win=2618624 Len=0
120	2.038235	::1	::1	HTTP	244	HTTP/1.1 200 OK (text/html)
121	2.038292	::1	::1	TCP	74	61526 → 8080 [RST, ACK] Seq=597 Ack=329 Win=0 Len=0
179	4.593146	::1	::1	TCP	86	61527 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256
181	4.593188	::1	::1	TCP	86	8080 → 61527 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65
183	4.593253	::1	::1	TCP	74	61527 → 8080 [ACK] Seq=1 Ack=1 Win=2618880 Len=0
185	4.593313	::1	::1	HTTP	492	POST / HTTP/1.1 (application/x-www-form-urlencoded)

Trame GET

```
> Frame 48: 414 bytes on wire (3312 bits), 414 bytes captured (3312 bits) on interfac
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 61526, Dst Port: 8080, Seq: 1, Ack: 1, Len
v Hypertext Transfer Protocol
  > GET /?nom=david&age=46 HTTP/1.1\r\n
    Host: localhost:8080\r\n
```

Méthode GET dans l'entête

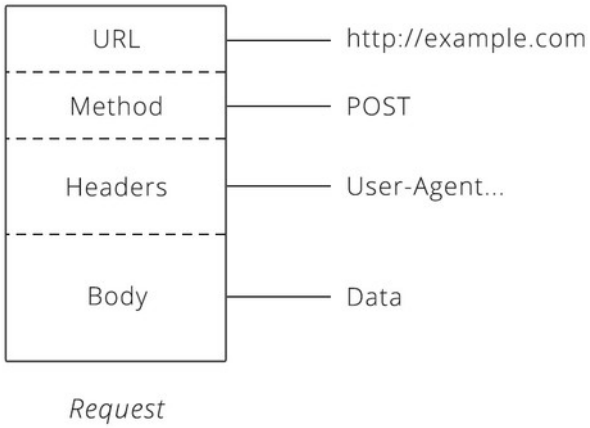
Trame POST

```
v Hypertext Transfer Protocol
  > POST / HTTP/1.1\r\n
    Host: localhost:8080\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:67.0) Gecko/20100101 Fi
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8\r\n
    Accept-Language: fr-FR\r\n
    Accept-Encoding: gzip, deflate\r\n
    Content-Type: application/x-www-form-urlencoded\r\n
  > Content-Length: 25\r\n
    DNT: 1\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://localhost:8080/]
    [HTTP request 1/2]
    [Response in frame: 203]
    [Next request in frame: 241]
    File Data: 25 bytes
v HTML Form URL Encoded: application/x-www-form-urlencoded
  > Form item: "login" = "admin"
  > Form item: "password" = "root"
```

Méthode POST dans l'entête

Données dans la partie "File Data"

Pour rappel, les échanges sont normalisés (protocole HTTP) et le navigateur du client envoie des informations en respectant cette norme.

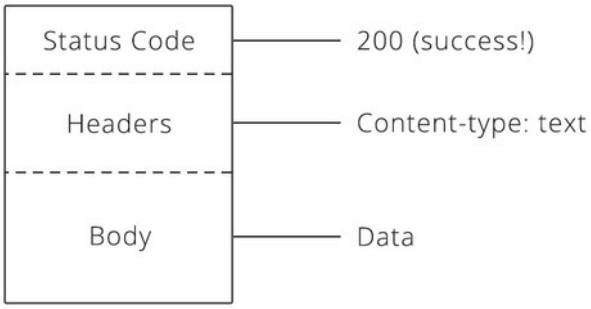


Request

Il existe réellement 4 méthodes :

- GET (demande au serveur de renvoyer une ressource : une page HTML, un fichier…)
- POST (demande au serveur de créer une ressource)
- PUT (demande au serveur la mise à jour d'une ressource existante)
- DELETE (demande au serveur de supprimer une ressource

Le serveur répondre à la requête avec un entête normalisé, contenant un code d'erreur et les données.



Response

Sachant cela, vous devriez mieux comprendre le fonctionnement de Node.JS et des formulaires HTML.

1.5 Inconvénients de Node.JS seul

Après avoir lu ces quelques pages, nous pourrions écrire une application Node.JS complète, qui affiche des pages HTML plus importante… cependant, le code généré pour obtenir une page HTML conforme aux recommandations du W3C, ressemblerait à ceci :

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

let 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 première application !</title>'+
'    </head>'+
'    <body>'+
'           <p>Voici un paragraphe <strong>HTML</strong> !</p>'+
'    </body>'+
'</html>');
    res.end();
});
server.listen(8080);
```

Il faudrait renouveler l'écriture pour chaque route (le dossier dans l'URL) ce qui deviendrait lourd et illisible.

Dans la prochaine partie, nous verrons donc comment palier un certain nombre d'inconvénients.

2 ANNEXES

2.1 SOURCES

https://www.npmjs.com/package/node-rest-client

https://www.w3schools.com/Node.JS/Node.JS_mysql_select.asp

https://oncletom.io/node.js/chapter-04/index.html

https://www.youtube.com/watch?v=Q8wacXNngXs

https://la-cascade.io/api-les-protocoles/

2.2 PROXY

2.2.1 Proxy pour npm

2.2.1.1 Activation proxy

```
npm config set proxy http://172.16.0.1:3128
npm config set https-proxy http://172.16.0.1:3128
```

vérification :

```
PS E:\david\WorkSpaces\NodeJS projects\NodeJS_RESTApi> npm config set proxy http://172.16.0.1:3128
PS E:\david\WorkSpaces\NodeJS projects\NodeJS_RESTApi> npm config set https-proxy http://172.16.0.1:3128
PS E:\david\WorkSpaces\NodeJS projects\NodeJS_RESTApi> npm install mysql
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN nodejs_restapi@1.0.0 No repository field.
```

2.2.1.2 désactivation proxy

```
npm config delete http-proxy
npm config delete https-proxy

npm config rm proxy
npm config rm https-proxy

set HTTP_PROXY=null
set HTTPS_PROXY=null
```

2.3 PROXY DANS LES APPLICATIONS NODE.JS

Le code ci-après permet à une application Node.JS d'accéder à Internet.

```
var Client = require('node-rest-client').Client;

// configure proxy
var options_proxy = {
    proxy: {
        host: "172.16.0.1",
        port: 3128,
        tunnel: true  // false = direct connexion
    }
};

var client = new Client(options_proxy);
```