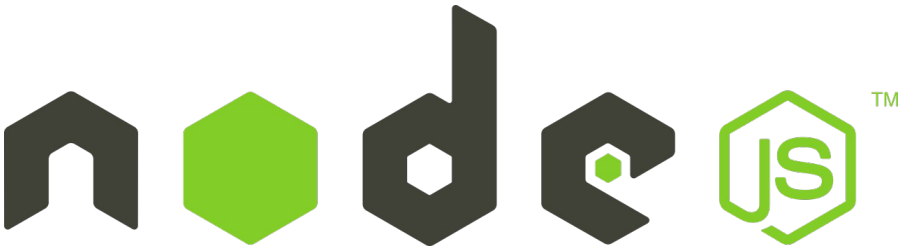


Node.JS

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
15/09/2020	V19 (corrections d'erreurs)
18/09/2020	V20 (correction p.22 GET/POST + lien localhost:8080
28/06/2021	Ajout NPM
07/09/2021	Correction statistique, performance et source
08/09/2022	Déplacement de l'installation dans un document à part, ajout import façon ES6

SOMMAIRE

NODE.JS

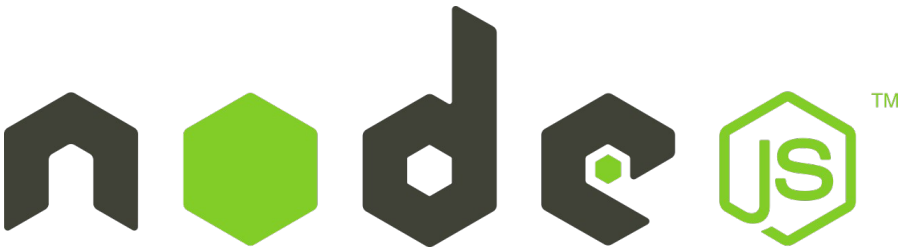
L'activité proposée ici aborde l'utilisation d'un environnement de développement asynchrone.

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 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 **runtime JavaScript** qui a rapidement évolué, la version actuelle étant la 8. Il permet de créer facilement un service web et n'est donc pas dépendant d'une application tierce (Apache, IIS, NGINX).

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.

1.1.1 Intérêt de Node.JS

Node.JS fonctionne différemment de PHP ou des langages comme Java, C# car il est asynchrone : par exemple, lorsqu'un téléchargement de fichier est lancé, Node.JS peut traiter d'autres tâches et n'exécutera le traitement du fichier qu'au déclenchement d'un événement qui indiquera la fin du chargement...

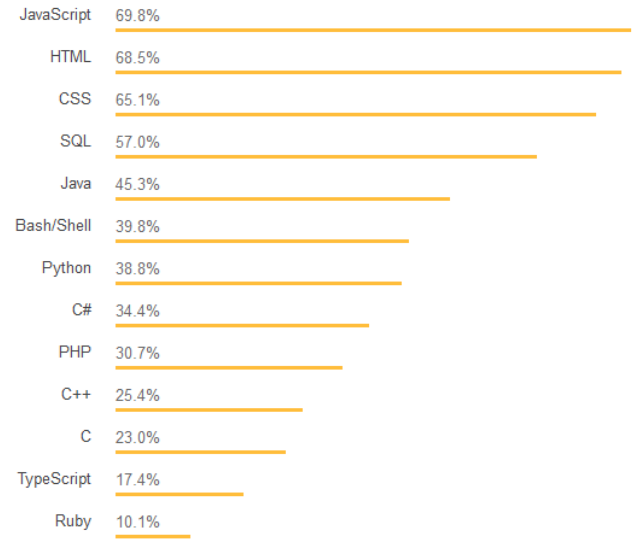
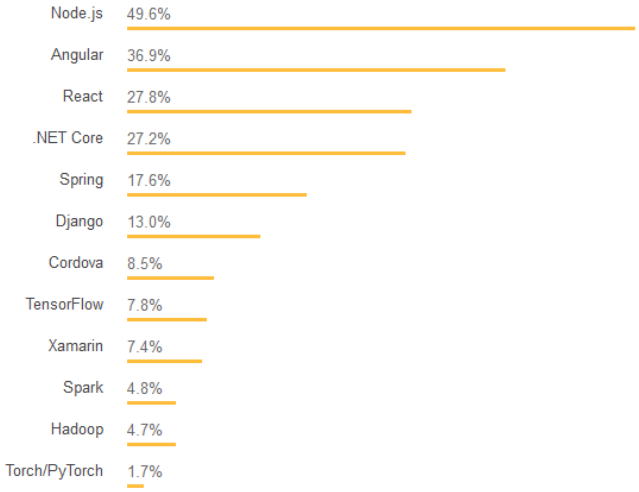
Un fichier Node.JS contient des tâches à lancer lorsque certains événements surviennent : accès à un port de communication par exemple.

Grâce à cela, Node.JS est très performant face à PHP. Le tableau ci-dessous montre la rapidité de traitement en milli-secondes 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.c-sharpcorner.com/UploadFile/2072a9/node-js-vs-traditional-scripting-languages/

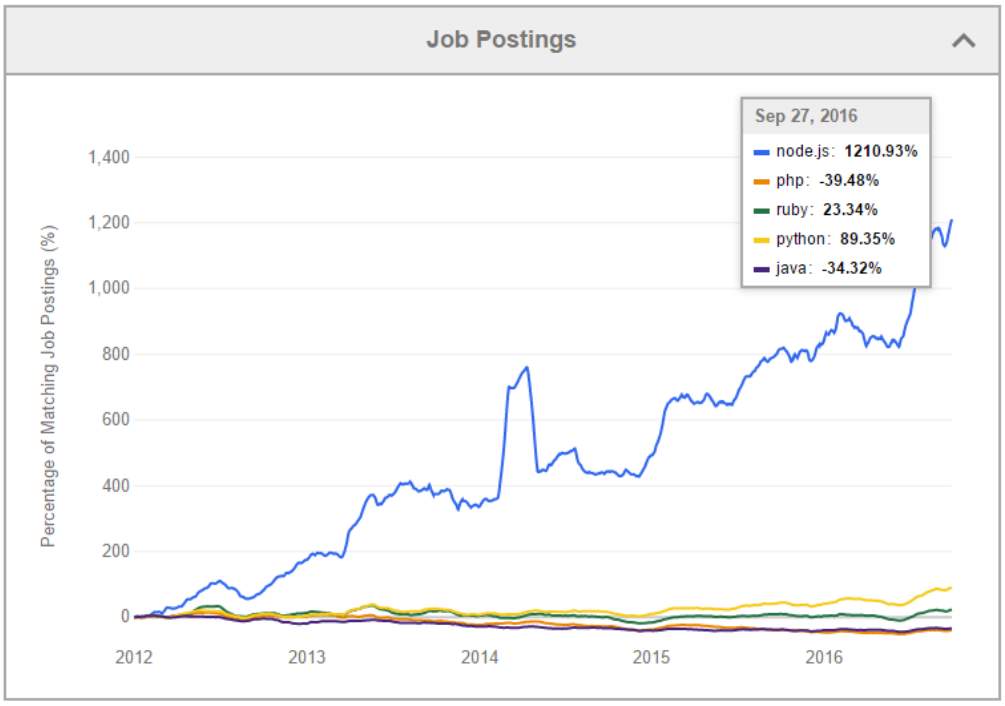
La communauté des développeurs s'intéresse énormément à cette technologie, comme le montre l'étude menée chaque année par StackOverflow.

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/) :

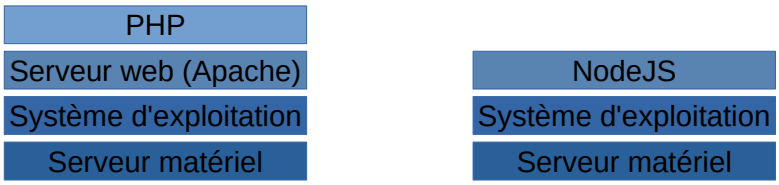


1.1.2 Comparaison PHP / NodeJS

La technologie NodeJS étant plus récente, elle cherche à répondre à un problème auquel PHP ne répond pas ou répond mal.

1.1.2.1 Serveur HTTP

Une première réponse est de reprendre le contrôle des requêtes HTTP/HTTPS directement dans le langage de programmation : tandis que PHP nécessite un serveur web pour fonctionner, un développeur NodeJS peut développer ses propres services web directement.



1.1.2.2 Modules

Node.JS dispose de bibliothèques de modules simples permettant de gérer des actions comme la connexion à un serveur (net.socket), à une base de données (mysql), etc.

l'installation se fait en utilisant l'outil **npm** que nous verrons plus loin.

Certains modules sont des modificateurs, c'est-à-dire, qu'ils se placent dans la chaîne de traitement, au milieu, et permettent de traiter, filtrer, modifier, adapter les données reçues pour le module en aval... ce sont les **middlewares**.

Côté PHP, on trouve des bibliothèques mais l'installation est plus classique (copie des fichiers).

1.1.2.3 Synchrone et asynchrone

PHP est un langage synchrone : aucune instruction suivante ne peut-être exécuté si la précédente n'est pas terminée. L'incidence est faible pour une opération courte, mais lorsqu'il s'agit de récupérer des données via Internet, un script PHP peut rester bloqué longtemps. Le serveur HTTP est donc chargé de lancer un nouveau processus à chaque nouvelle session, pour ne pas bloquer tous les utilisateurs (multithread).

Inversement, NodeJS n'utilise qu'un seul thread, mais utilise de nombreuses instructions (asynchrones) qui génèrent des événements. On utilise alors des fonctions appelées 'callback' qui effectuent le traitement nécessaire lorsqu'un événement particulier survient. JavaScript utilise pour cela une file d'attente (queue) qu'il parcourt régulièrement. Lorsqu'un événement survient, il place le code à exécuter dans la pile d'appels (stack).

1.2 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.

1.2.1 JavaScript sur le serveur

Node.JS est un runtime js , donc coder une application Node.js, c'est coder en JS ? JavaScript comme langage de programmation.

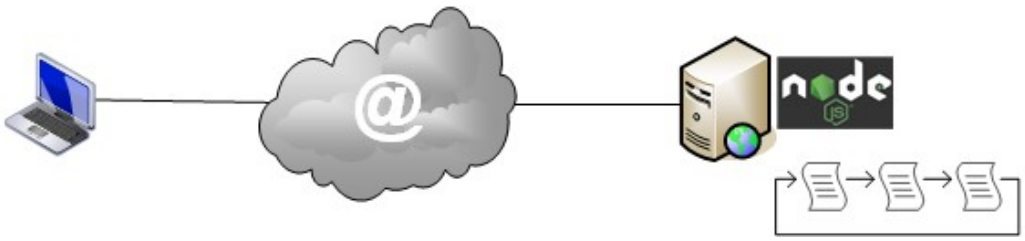
Souvent décrié, car il servait principalement à ajouter des effets dans les pages web, JavaScript est pourtant un véritable langage. Pour être exact, JavaScript est une implémentation du standard de langage ECMAScript[1] par Brendand Eich. Inventé dans les années 1995 pour le navigateur Netscape, c'est 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 est capable d'exécuter du code JavaScript côté serveur !

En résumé, Node.JS remplace Apache et PHP.[2]

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 retournera un résultat lorsque la tâche sera terminée.



Le cours de Bloc2 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.

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

1.2.1.1 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".

Notation ES6

Nous verrons qu'il existe une nouvelle manière d'intégrer des bibliothèques, compatible avec la nouvelle syntaxe ES6 (EcmaScript).

```
import http from 'http'
```

Les fichiers peuvent porter l'extension `.mjs` pour faciliter l'intégration avec Node.JS. Cependant, cette notation n'est pas la plus courante pour le moment (projet existant) et exige encore des fonctions NodeJS en beta ; Le module de Bloc 2 sera donc dans le standard le plus courant, mais vous ne devez pas être choqué par la différence de forme.

1.2.1.2 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

1.2.1.3 Export : modules personnels

Il est possible et recommandé d'écrire ses propres modules dans des fichiers séparés, pour rendre le code plus lisible.

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;
    }
}
```

1.2.2 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.

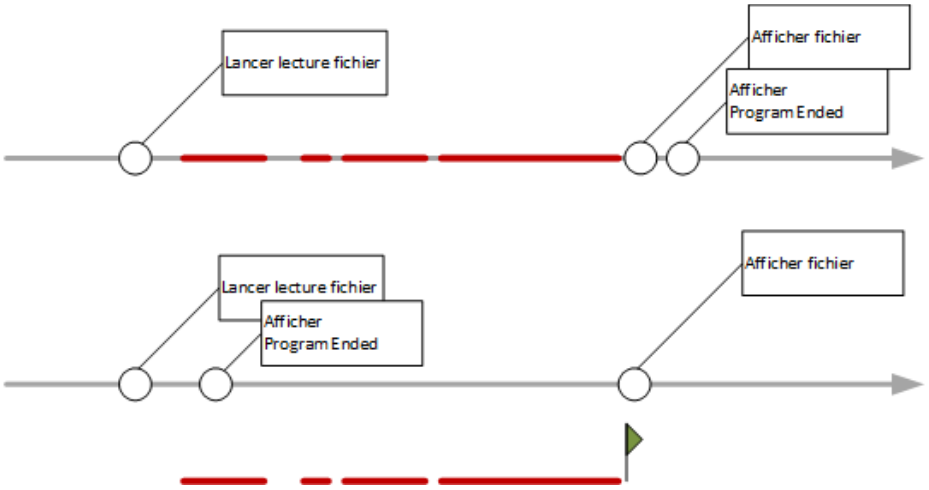
Prenons un exemple concret en créant un fichier readme.txt et en le lisant de deux manières différentes en Node.JS. *Il est recommandé de tester les 2 exemples* :

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.

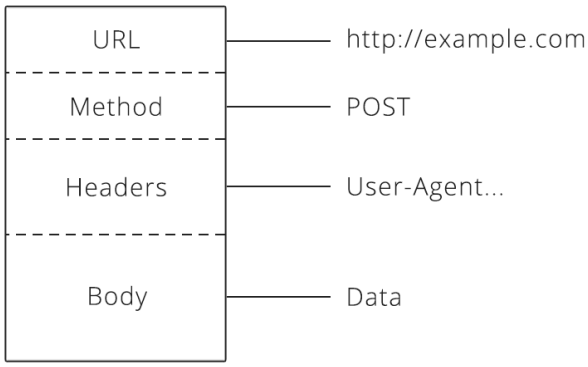


Il se cache derrière cette notion de **callback**, la notion d'événement (en anglais, stream) : dans un cas, il est nécessaire d'atteindre la fin du flux pour poursuivre l'exécution du programme ; dans le second cas, on indique à Node.JS qu'il lance la lecture du flux et continue le programme. Lorsque l'événement “fin de flux” survient (drapeau vert sur le schéma), Node.JS appelle la fonction callback (appel de retour) qui renvoie le contenu sur la sortie standard (par défaut avec console.log()) (qu'importe l'endroit du programme où il se trouve).

Les détails sont disponibles dans les **fiches** de cours.

1.2.3 Le protocole HTTP

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

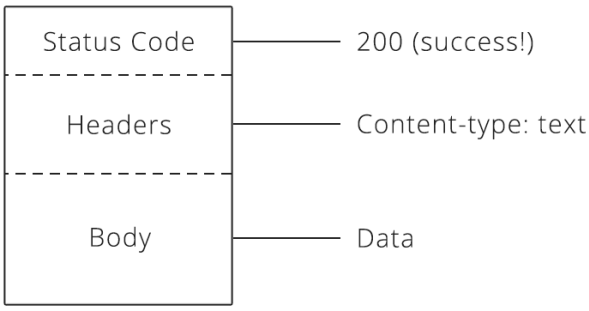


Request

Il existe principalement 4 méthodes[3] :

- **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 doit répondre à la requête avec un entête normalisé, contenant un code d'état et les données (pas toujours, cas de la redirection 302 par exemple).



Response

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

3 *Une dizaine en fait : https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode*

1.3 Inconvénients de Node.JS seul

Après avoir lu ces quelques pages, nous pourrions écrire une application Node.JS complète, qui retourne (c'est le navigateur 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://nodejs.org/fr/

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
let options_proxy = {
    proxy: {
        host: "172.16.0.1",
        port: 3128,
        tunnel: true  // false = direct connexion
    }
};
let client = new Client(options_proxy);
```