



DÉCOUVERTE NODE.JS



date	modification
2022-10-04	Correction client.query() page 6
2023-10-12	Correction mysql2 plutôt que mysql et ajout de tests avec res.end(txt)



SOMMAIRE

1 Introduction.....	3
1.1 Prérequis.....	3
1.2 Objectifs.....	3
1.3 Mode REPL.....	4
1.4 Mode Script.....	5
2 Accès Base de données.....	6
2.1 Jeu de données.....	6
2.2 Connexion base de données.....	7
3 Interactivité des requêtes.....	9
3.1 Propriété url.....	9
3.2 méthode url.parse.....	9
3.2.1 Explication.....	11
3.2.2 Asynchronisme.....	11
3.2.3 Transaction incomplète.....	11
4 Usage de Wireshark.....	12
4.1 Trame GET.....	12
4.2 Trame POST.....	13
4.3 Non réponse.....	13
5 Annexes.....	14
5.1 Erreurs d'installation.....	14
5.1.1 PATH.....	14
5.1.2 Session active.....	14
5.2 Erreurs de modules.....	14



1 INTRODUCTION

Jusqu'à maintenant, nous n'avons utilisé JavaScript que du côté du client, dans un navigateur. Nous allons maintenant découvrir la puissance de Node.JS en programmant un serveur très léger qui lira dans une base de données MySQL et qui pourra recevoir des données en provenance d'un client.

L'écriture dans la base de données n'est pas abordé ici.

1.1 PRÉREQUIS

Node.JS est l'environnement d'exécution de scripts JavaScript, côté serveur.

Voici la liste des prérequis pour réaliser cette exploration :

- Avoir installé correctement Node.JS
- Savoir utiliser les commandes npm
- Connaître JavaScript
- Connaître SQL
- Connaître la différence entre l'envoi d'un formulaire par la méthode GET et la méthode POST

1.2 OBJECTIFS

Cette découverte doit permettre de générer les premiers codes en JavaScript pour Node.JS.

Pour cela, nous allons apprendre les connaissances suivantes :

- Différence entre le mode REPL et le mode script
- Usage du module mysql2 pour lire une base de données MySQL
- Créer de l'interactivité en lisant (côté serveur) les informations transmises par un client, soit par méthode GET, soit par méthode POST.
- Être capable d'analyser des transactions HTTP avec un analyseur de trame



1.3 MODE REPL

Utiliser Node.JS en mode REPL permet d'écrire des commandes JavaScript directement, comme dans la console de débogage des navigateurs comme Chrome ou Firefox. REPL signifie :

1. **Read** the user input
2. **Evaluate** the code
3. **Print** any results
4. **Loop** back to first step

Dans une invite de commande, tapez l'instruction suivante :

```
node
```

Un message devrait apparaître et l'invite de commande de Node.JS doit s'afficher. Si ce n'est pas le cas, consulter l'annexe des erreurs.

```
Welcome to Node.js v16.16.0.  
Type ".help" for more information.  
>
```

Pour tester ceci, tapez les commandes JavaScript suivantes :

```
let prenom = "Chuck"  
let nom = "NORRIS"  
console.log("Bonjour "+prenom+" "+nom)
```

Votre affichage devrait être similaire à ceci :

```
Bonjour Chuck NORRIS  
undefined  
>
```

Utilisez `CTRL+C` deux fois pour quitter l'environnement REPL.

Vous pouvez aussi utiliser les commandes `.help` et `.exit` !



1.4 MODE SCRIPT

Évidemment, le mode le plus utilisé est en mode script : Node.JS va donc exécuter un fichier contenant des instructions JavaScript.

Créez le fichier script.js et ajoutez-y les commandes suivantes :

script.js

```
let a = 8
let b = 2
let oct = 2**8
console.log("un octet contient",oct,"valeurs")

// afficher la version actuel de nodeJS
console.log(process.version)
```

puis lancez la commande suivante dans l'invite de commande (au même emplacement que le script) :

```
node ./script.js
```

Le résultat devrait apparaître (la valeur numérique, en couleur)

```
un octet contient 256 valeurs
```



2 ACCÈS BASE DE DONNÉES



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

Vous pouvez également utiliser une base de données en ligne : <https://www.db4free.net/about.php>

2.1 JEU DE DONNÉES

Le jeu de données est un ensemble d'informations permettant de tester les traitements dans un code (affichage, suppression, modification, ajout...). Voici les requêtes pour générer l'exemple :

```
CREATE TABLE `matable` (
  `id` int(11) NOT NULL,
  `name` varchar(50) NOT NULL,
  `address` varchar(300) NOT NULL,
  `email` varchar(50) NOT NULL,
  `phone` varchar(15) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Contenu de la table `matable`
--

INSERT INTO `matable` (`id`, `name`, `address`, `email`, `phone`) VALUES
(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');
```

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



2.2 CONNEXION BASE DE DONNÉES

Étudions l'accès à une base MySQL depuis Node.JS ; L'utilisation d'une fonction s'écrit comme suit :

```
// On utilise le module node-mysql2
var mysql = require('mysql2');

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

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

Si le module 'mysql2' n'existe pas dans votre projet, il faut l'installer avec **npm**.

Par la suite, on peut utiliser les fonctions simplement, comme ci-dessous :

```
// On se connecte à la base de données.
client.connect();
```

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 :

```
client.query('SELECT * from matable', function(err, rows, fields) {
  if (!err)
    console.log('Le résultat est : ', rows);
  else
    console.log('Erreur durant la requête...');
});
```

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](#)



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

```
let mysql = require('mysql2');

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 mysql2
```

Puis exécutez le code :

```
node mySQLExample1.js
```




3 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.

3.1 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) {
  // Préparer une réponse (status + type + text) et l'envoyer (.end())
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("je dois afficher l'application sur la page "+req.url);
  res.end();
}).listen(8080);
```

Récupère dans la requête (demande) le contenu de l'URL

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

3.2 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.

form.html

```
<html>
<body>
  <h1>Formulaires POST et GET</h1>
  <form method="GET" action="http://localhost:8080">
    nom : <input type="text" name="nom">
    age : <input type="text" name="age">
    <input type="submit" value="Envoi (post)">
  </form>

  <form method="POST" action="http://localhost:8080">
    identifiant : <input type="text" name="login">
    mot de passe : <input type="text" name="password">
    <input type="submit" value="Envoi (get)">
  </form>

</body>
</html>
```



index2.js

```
// Importation des modules ou fonction d'un module (parse)
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'}); // préparer réponse
  txt = "";
  console.log("Connexion HTTP reçue depuis", req.socket.remoteAddress);
  if (req.method == "GET") {
    // avec GET, les données sont dans l'URL
    let q = url.parse(req.url, true).query;
    txt = "<strong>nom : </strong>"+q.nom + "<br>\n<strong>age : </strong>" +
q.age;
    res.end(txt);          // envoyer réponse
  }
  else if (req.method == "POST") {
    // avec POST, il faut lire les données
    // On utilise deux listeners (data et end).
    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>\n
pwd:"+parse(body).password;
      res.end(txt);          // envoyer réponse
    });
  }
  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" (analyser) l'URL,
- dans le second cas, il faut récupérer la zone `file data` (puis l'analyser) au moyen de deux fonctions : une fonction qui se lance à chaque donnée transmise (événement 'data') et une fonction qui termine la lecture et envoie la réponse (événement 'end').



3.2.1 Explication

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 `res`.

3.2.2 Asynchronisme

Note : il serait tentant de sortir l'instruction `res.end(txt)` qui est redondante dans chaque condition et de la placer à la fin (juste avant la ligne `console.log('----')`) mais ce serait une erreur. Pour la méthode POST, Nous sommes en asynchrone. Cela signifie que la fin serait atteinte avant d'avoir pu lire le contenu de `body`.

3.2.3 Transaction incomplète

Afin de comprendre comment s'effectuent les échanges entre le serveur et le client, mettez en commentaire les lignes `res.end(txt)` et relancez le script :

- CTRL+C
- `node index2.js`

Regardez comment se comporte le navigateur après avoir envoyé un formulaire (GET ou POST) :

La page du formulaire reste affichée et le navigateur semble attendre des données, ce qui est le cas : il a reçu un signal que la connexion TCP est établie, mais aucune donnée n'est transmise.

Dé-commentez les lignes contenant `res.end(txt)`.



4 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 **wincap**. NPCAP, car il peut désormais lire les flux de l'interface locale (IPv4 127.0.0.1 ou Ipv6 ::1).

Cliquez sur les deux boutons de la page web précédente et vérifiez que vous capturez les informations de manière similaire aux deux captures d'écran suivante :

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)

4.1 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
▼ Hypertext Transfer Protocol
  > GET /?nom=david&age=46 HTTP/1.1\r\n
    Host: localhost:8080\r\n

```

Méthode GET dans l'entête

Le point faible de la méthode GET concerne la visibilité des informations dans l'URL du navigateur.

Pendant, la plus grande faiblesse, vient de l'utilisation du protocole HTTP : les communications ne sont pas chiffrées. Cela signifie qu'un pirate peut écouter toutes les informations et récupérer les données qui transitent, quelle que soit la méthode (GET/POST) utilisée.



4.2 TRAME POST

```

▼ 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 Firefox/67.0\r\n
    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
  ▼ 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"

On constate ici, que Wireshark décode facilement la trame, et affiche simplement les données transmises : un identifiant et un mot de passe seraient tout aussi faciles à afficher.

Dans le cadre des cours, nous travaillerons globalement en HTTP, pour permettre le décodage des trames et la vérification du bon fonctionnement des échanges. Nous ajouterons la sécurisation des échanges avec le protocole HTTPS, qui nécessite un certificat, plus tard.

4.3 NON RÉPONSE

Comme au chapitre précédent, commentez les lignes `res.end(txt)`, relancez le script `index2.js` et relancez une capture avec Wireshark.

On constate que le serveur informe le navigateur que sa requête est en cours de traitement, par la présence régulière de trames TCP de type 'Keep-Alive'... et ne sera jamais terminée, à cause des `res.end()` manquants. **Des pirates peuvent tirer profit de cet oubli pour saturer un serveur de requête qui ne se terminent jamais.**

TCP	127.0.0.1	127.0.0.1	66 [TCP Keep-Alive] 41990 → 8080 [ACK] Seq=976 Ack=1 Win=65536 Len=6
TCP	127.0.0.1	127.0.0.1	66 [TCP Keep-Alive ACK] 8080 → 41990 [ACK] Seq=1 Ack=977 Win=65536 L

Stoppez le script (CTRL+C dans la console Node.JS) pour stopper ce comportement.



5 ANNEXES

5.1 ERREURS D'INSTALLATION

5.1.1 PATH

Lors de l'installation de Node.JS, afin que la commande 'node' soit accessible partout, la variable d'environnement **PATH** doit contenir le chemin vers Node.JS.

Vérifiez que ce soit le cas.

<https://www.malekal.com/comment-modifier-la-variable-path-sous-windows-10-11/>

5.1.2 Session active

Si la variable d'environnement contient Node.JS, elle n'est peut-être pas prise en compte : fermez et ré-ouvrez votre session.

5.2 ERREURS DE MODULES

Les erreurs de modules proviennent généralement de l'oubli des commandes `npm init` et `npm install xxxx`.