

Adel Latibi
adel.latibi@gmail.com

Plan



- Présentation
- Pourquoi NodeJS ?
- Installation de NodeJS
- Tester NodeJs
- Creation premier serveur
- Intégrer les fichiers HTML
- Passer des paramètres via l'URL
- Evénements
- Les Streams



Présentation

NodeJS est une plateforme construite sur le moteur JavaScript V8 de Chrome qui permet de développer des applications en utilisant du JavaScript. Il se distingue des autres plateformes grâce à une approche non bloquante permettant d'effectuer des entrées/sorties (I/O) de manière asynchrone.



Pourquoi NodeJS ?

Avant de commencer à découvrir une nouvelle technologie, il est important d'en comprendre les spécificités. Pour comprendre nous allons partir de la description donnée sur le site officiel :

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.



Ce que n'est pas NodeJS

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.



Installation de NodeJS

Pour télécharger **NodeJS** il vous suffit de vous rendre sur le [site officiel](#) et de choisir la version que vous souhaitez installer. Si vous souhaitez obtenir plus d'information sur la conséquence de la mention LTS (Long term support) n'hésitez pas à jeter un oeil sur le dépôt [GitHub](#).

Windows & MacOS

Pour Windows et MacOS l'installation se fait au travers d'un exécutable qui vous guidera dans les différentes étapes d'installation et configuration.

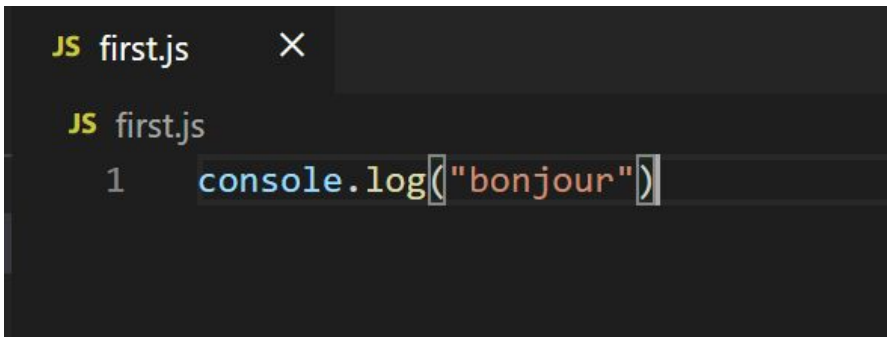
Linux

Sur linux, la méthode la plus simple reste de passer par le gestionnaire de paquets.



Tester NodeJs

Pour tester et voir si le NodeJs est bien installé on va créer un dossier **test** ensuite on met le code suivant:





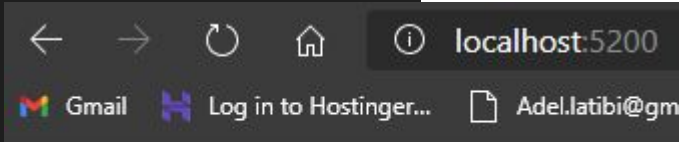
```
JS first.js  X
JS first.js
1 console.log("bonjour")
```

Pour exécuter ce code on tape la commande: **node first.js**

Notre premier serveur

1. Création d'un nouveau dossier *formation_nodejs*
2. Création d'un fichier *server.js*


```
JS server_first.js >  server.on('request') callback >  'Content-Type'
1  const http = require('http'); // importation module http
2  // creation un objet server
3  const server = http.createServer()
4  // evenement pour créer un serveur
5  server.on('request', (request,response) => {
6      // l'entete de la réponse http
7      response.writeHead(200, { 'Content-Type': 'text/plain' });
8      response.end("Bonjour à tous"); // la reponse au client
9  })
10 server.listen(5200) // le port
```



Dans l'exemple précédent on a vu comment créer un simple serveur avec la fonction `createServer()`

Mais le problème c'est que d'abord la réponse **http** n'est pas sous format **HTML**, l'encodage de texte n'est pas en **utf-8** pour les tous les caractères, pour ce faire:

JS server_first.js X


JS server_first.js >  server.on('request') callback

```
1  const http = require('http'); // importation module http
2  // creation un objet server
3  const server = http.createServer()
4  // evenement pour créer un serveur
5  server.on('request', (request,response) => {
6      // l'entete de la réponse http
7      response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
8      response.end("<h1>Bonjour à tous</h1>"); // la reponse au client
9  })
10 server.listen(5200) // le port
```

← → ↺ 🏠 ⓘ localhost:5200

 Gmail  Log in to Hostinger...  Adel.latibi@gmail

Bonjour à tous




Maintenant pour simplifier notre code on a même pas besoin de créer la constante `server` ou d'utiliser l'événement `request` on peut directement faire:

```
1  const http = require('http'); // importation module http
2  // creation d'un serveur
3  http.createServer( (request,response) => {
4      // l'entete de la réponse http
5      response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
6      response.end("<h1>Hello world</h1>"); // la reponse au client
7  }).listen(5200) // le port
```

Intégrer les fichiers HTML

Pour l'instant on a vu comment le serveur répond avec une simple chaîne de caractères, mais en réalité, on utilise des fichiers HTML pour afficher notre page. Pour cela on importe le module **fs** (filesystem) [Voir la doc NodeJs](#) et on crée le fichier **index.html**

```
<> index.html >  html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Formation NodeJs</title>
7  </head>
8  <body>
9    <h1>Bienvenue sur mon site</h1>
10   <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Modi nobis quis elige
11  </body>
12  </html>
```

JS server_second.js > ...

```
1  const http = require('http'); // importation module http
2  const fs = require('fs'); // importation module fs pour lire des fichier
3
4  http.createServer((request,response) => {
5    fs.readFile('index.html', (err, data) => {
6      // err si il y a un probelme sur le fichier
7      // data le contenu de fichier html
8      if (err) throw err; // lancer une exception
9      response.writeHead(200,
10     { 'Content-Type': 'text/html; charset=utf-8' });
11     response.end(data);
12   });
13 }).listen(5200)
```

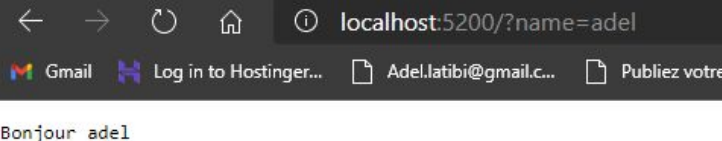
Passer des paramètres via l'URL

Si on a un URL de type: <http://localhost:5200?name=adel>

ici le paramètre est *name* qu'on passe au serveur, et c'est une requête de type *GET*

JS server.js > ...

```
1  const http = require('http'); // importat
2  const url = require('url'); // importati
3
4  http.createServer( (request,response) => {
5      let query = url.parse(request.url,true).query
6      response.writeHead(200)
7      response.end("Bonjour " + query.name)
8  }).listen(5200)
```



Événements

```
JS server_event2.js > ...
1  const EventEmitter = require('events');
2  const monEcouleur = new EventEmitter()
3
4  monEcouleur.on("saute", (a,b) => {
5    console.log("j'ai sauté",a,b)
6  })
7
8  monEcouleur.emit("saute",10,20)
9  monEcouleur.emit("saute")
10 monEcouleur.emit("saute")
```

```
PS C:\Users\adell\projects\cours_nodejs> node server_event2.js
j'ai sauté 10 20
j'ai sauté undefined undefined
j'ai sauté undefined undefined
```

Événements 2



```
JS server_event2.js > ...
1   const EventEmitter = require('events');
2   const monEcouteur = new EventEmitter()
3
4   monEcouteur.once("saute", (a,b) => {
5     console.log("j'ai sauté",a,b)
6   })
7
8   monEcouteur.emit("saute",10,20)
9   monEcouteur.emit("saute")
10  monEcouteur.emit("saute")
```


Example events

```
1  const HttpServer = require('http'); // importation module http
2  const EventEmitter = require('events'); // importation module events
3
4  const App = {
5    start: function (port) {
6      const emitter = new EventEmitter();
7      // creation d'un serveur
8      HttpServer.createServer( (request,response) => {
9        // l'entete de la réponse http
10       response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
11       emitter.emit('root',response)
12       response.end(); // la reponse au client
13     }).listen(port) // le port
14
15     return emitter;
16   }
17 }
18
19
20 const app = App.start(5050)
21 app.on('root',(response)=>{
22   response.write("<h1>Hello world of events</h1>")
23 })
24
```


Les Streams



Le code ci-dessous c'est la méthode classique pour créer des copies des fichiers qui comporte des inconvénients.

```
1  const fs = require('fs')
2
3  fs.readFile('logo.png', (err, data) => {
4    if (err) throw err;
5    fs.writeFile('copy.png', data, (err) => {
6      if (err) throw err;
7      console.log('le fichier a bien été copié')
8    })
9  })
```

Les Streams 2

```
JS stream.js > [?] read
1   const fs = require('fs')
2
3   file = "logo.png"
4   const read = fs.createReadStream(file)
5   read.on('data', (chunk) => {
6     |   console.log("j'ai lu "+chunk.length)
7   })
8
9
10  read.on('end', ()=>{
11    |   console.log("j'ai fini de lire le fichier")
12  })
```

Les Streams 3 stats

```
JS stream_stats.js > ...
1  const fs = require('fs')
2
3  fs.stat("logo.png", (err, stat) => {
4    const total = stat.size
5    let progress = 0
6
7    const read = fs.createReadStream("logo.png")
8    read.on('data', (chunk) => {
9      progress += chunk.length
10     p = Math.round(100 * progress / total)
11     console.log("j'ai lu "+p+" %")
12   })
13
14   read.on('end', () => {
15     console.log("j'ai fini de lire le fichier")
16   })
17 })
18
```

Les Streams 4

```
JS stream_copy.js > ...
1  const fs = require('fs')
2
3  file = "logo.png"
4  const read = fs.createReadStream(file)
5  const write = fs.createWriteStream("logo_cp.png")
6  |
7  read.on('data', (chunk) => {
8  |    console.log("j'ai lu "+chunk.length)
9  |  })
10
11 read.pipe(write)
12
13 read.on('finish', ()=>{
14 |   console.log("j'ai fini de lire le fichier")
15 | })
```



Références

1. Grafikart
2. openclassroom
3. [NodeJs Docs](#)
4. [DevDocs](#)