
UE 702 - MIOA702T -

Partie « Framework web côté serveur »

Nathalie Hernandez
nathalie.hernandez@univ-tlse2.fr



Objectifs

- Apprendre à faire des scripts en Javascript avec NodeJS
- Savoir mettre en place un serveur web léger avec NodeJS et Express

Plan de cours

➤ NodeJS, le moteur Javascript

- Rappels Javascript serveur / Javascript client
- Module, export, require et npm
- L'accès au système de fichiers
- Tester son script nodeJS
- Mini TP
 - Installation de NodeJS
 - petit script de lecture de fichiers
- NodeJS comme serveur web
 - Rappels HTTP
 - Les requêtes et les réponses HTTP
 - Les templates
- Mini TP : appel du script de lecture via une api

➤ Express, le framework web pour NodeJS

- Introduction et fonctionnement d'Express
- Le module nodejs path
- Les objets Request et Response
- Mini TP
- Les templates
- Les middlewares d'Express
- Le routing avec Express

➤ Création d'un squelette d'application avec express-generator

NodeJS, le moteur javascript

Rappels Javascript serveur / Javascript client

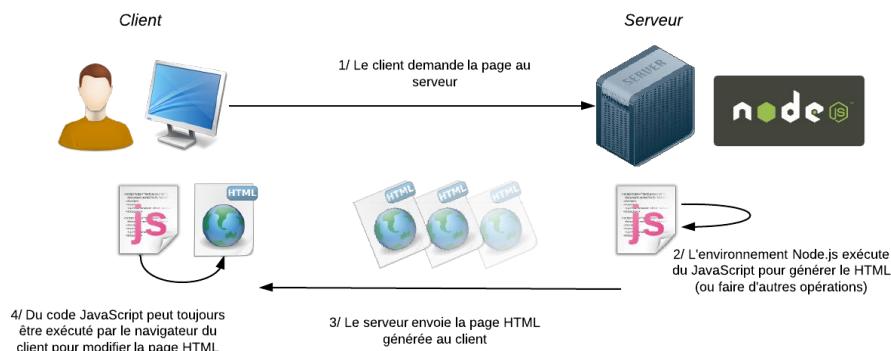
NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

- Le javascript est un langage de script qui peut être exécuté sur différents environnements :
 - Dans un navigateur web, on parle de javascript côté client



- Dans une machine virtuelle directement exécuté sur le système d'exploitation, on parle de javascript côté serveur

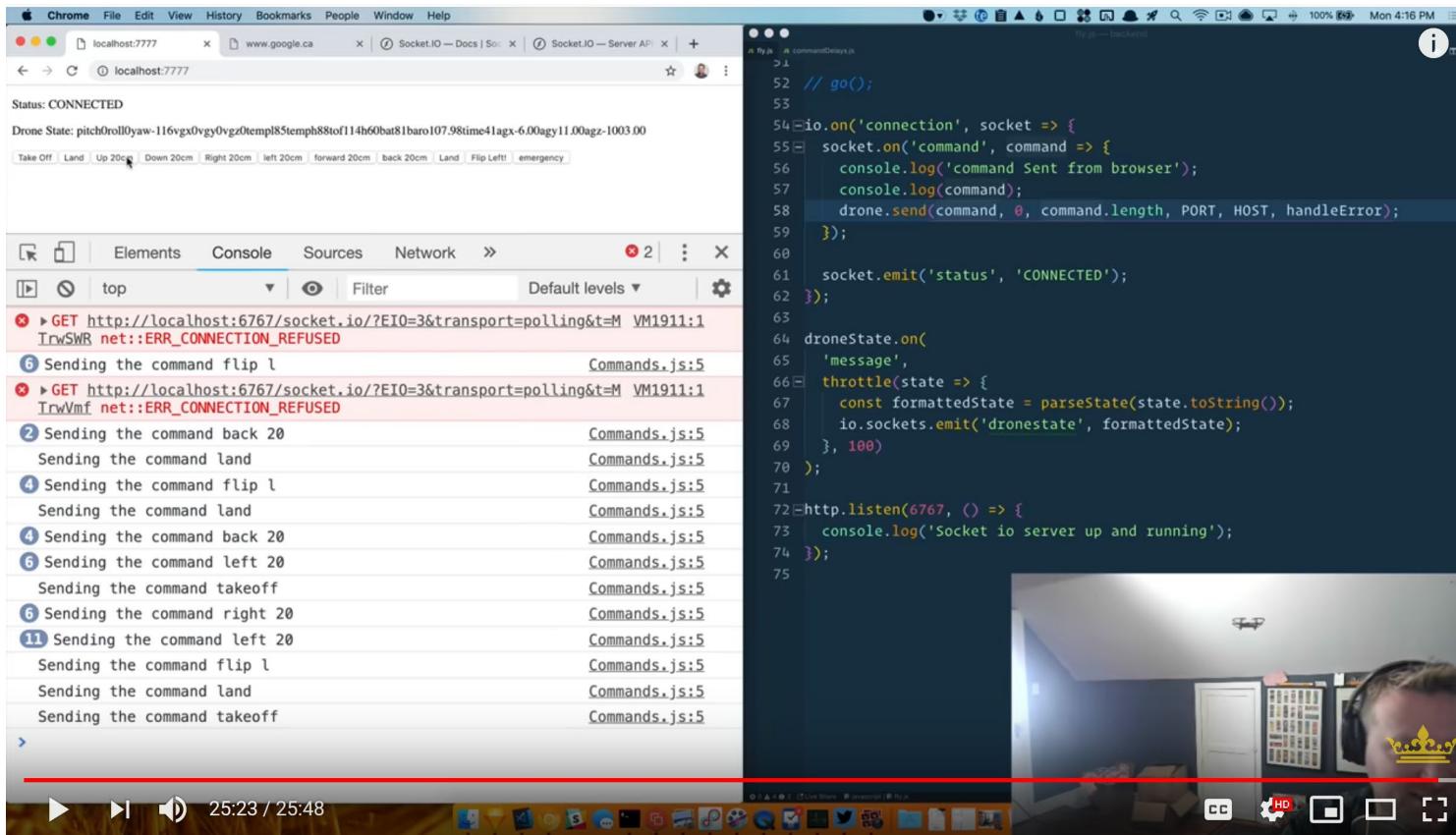


NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

- On peut également utiliser le javascript pour faire de l'embarqué ou des objets connectés!

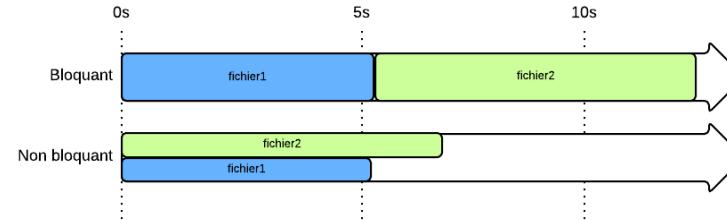
Par exemple, faire voler un drone : <https://www.youtube.com/watch?v=JzFvGf7Ywkk>



NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

- NodeJS est un environnement côté serveur, développé par Ryan Dahl en 2009, maintenu par la société Joyent.
- Il s'exécute sur la machine virtuelle V8, développé pour javascript par Google
- Sa rapidité vient de son modèle non bloquant, qui permet l'exécution de plusieurs instructions simultanément



- Via ses modules natifs, il peut
 - Répondre à des requêtes HTTP
 - Accéder au système de fichiers
 - Tout ce que vous faites en java peuvent être fait en node JS

<https://v8.dev>

NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

Quelques différences notables entre du javascript serveur et du javascript client :

- Le JS client est isolé dans le navigateur.
 - Il a accès aux api du navigateur (l'objet *document* par exemple)
 - Mais il n'a pas le droit d'accéder au système de fichiers
 - Il peut utiliser les périphériques de l'utilisateur (micro, caméra...) en demandant au navigateur de demander à l'utilisateur (« Autorisez vous www.google.com à accéder à votre micro? »)
 - Il est mono thread
- Le JS serveur s'exécute sur le système d'exploitation
 - Il a donc accès au système de fichiers
 - *Single Threaded Event Loop*

NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

Exemple d'un script « hello world » avec NodeJS

```
JS helloWorld.js
1   console.log("hello world !");|
```

Pour exécuter le script, via le terminal :

```
$ node helloWorld.js
hello world !
```

Et voilà!

NodeJS, le moteur Javascript

Rappels Javascript serveur/Javascript client

Exemple d'un script avec le nom de l'utilisateur en argument d'entrée

Pour l'exécuter :

```
$ node helloWorld.js blandine
Hello blandine!
```

Et si l'on ne donne pas d'argument, le script retourne une erreur :

```
$ node helloWorld.js
Missing argument! Example: node helloWorld.js YOUR_NAME
```

```
JS helloWorld.js > ...
1  #!/usr/bin/env node
2
3  'use strict';
4
5  /*
6   * The command line arguments are stored in the `process.argv` array,
7   * which has the following structure:
8   * [0] The path of the executable that started the Node.js process
9   * [1] The path to this application
10  * [2-n] the command line arguments
11
12  Example: [ '/bin/node', '/path/to/yourscript', 'arg1', 'arg2', ... ]
13  src: https://nodejs.org/api/process.html#process\_process\_argv
14 */
15
16 // Store the first argument as username.
17 const username = process.argv[2];
18
19 // Check if the username hasn't been provided.
20 if (!username) {
21     // Give the user an example on how to use the app.
22     console.error('Missing argument! Example: node helloWorld.js YOUR_NAME');
23
24     // Exit the app (success: 0, error: 1).
25     process.exit(1);
26 }
27
28 // Print the message to the console.
29 console.log('Hello %s!', username);
```

NodeJS, le moteur javascript

Module, export, require et npm

NodeJS, le moteur Javascript Module, export, require et npm

NodeJS a un système de gestion de module intégré(CommonJS).

Un fichier NodeJS peut importer une fonctionnalité exposée par un autre fichier NodeJS.

Pour importer une fonctionnalité d'un fichier, utiliser le mot clé **require** :

```
const library = require('./library');
```

Dans le fichier *library.js*, la fonctionnalité doit être exposée via l'api **module.exports** :

```
const library = {
  brand: 'Ford',
  model: 'Fiesta'
}
module.exports = library
```

<https://nodejs.dev/expose-functionality-from-a-nodejs-file-using-exports>

NodeJS, le moteur Javascript Module, export, require et npm

npm est le gestionnaire de paquets standard pour NodeJS.

- Il existe plus de 300 000 paquets sur le registre npm. C'est le plus gros registre de paquet pour un langage de programmation.
- Il existe des paquets pour à peu près tout (et n'importe quoi aussi ^^)
- Certains paquets sont exclusifs pour nodeJS, d'autres pour le javascript client (dans le navigateur donc).
- Certains paquets peuvent être utilisés sur les deux plate formes.
- Des alternatives à *npm* existent. Par exemple, *yarn*.

NodeJS, le moteur Javascript

Module, export, require et npm

Installer un paquet se fait via la commande *install* :

```
npm install <package-name>
```

- Cette commande installe le paquet *package-name* et crée :
 - un fichier *package-lock.json* qui contient la version du paquet installé
 - Un dossier *node_modules* contenant le paquet installé

package-lock.json et *node_modules* sont créés dans le dossier où l'on a exécuté la commande *npm install*

NodeJS, le moteur Javascript Module, export, require et npm

Lorsque l'on récupère un projet javascript existant, *npm* va nous aider à récupérer la liste de toutes les dépendances facilement.

Il faut :

- un fichier *package.json* à la racine du projet
- Et lancer la commande *npm install*

Le fichier *package.json* contient

- les informations essentielles du projet (nom, type de licence, auteur...)
- Les dépendances npm
- Des tâches pour lancer des scripts de tests, de packagin ou autre

NodeJS, le moteur Javascript

Module, export, require et npm

Exemple de fichier package.json :

```
{} package.json > ...
1  {
2    "name": "m1-ice-example",
3    "version": "1.0.0",
4    "description": "",
5    "main": "script.js",
6    "dependencies": {
7      "lodash": "^4.17.15"
8    },
9    "devDependencies": {},
10   "scripts": {
11     "test": "echo \\\"Error: no test specified\\\" && exit 1"
12   },
13   "author": "Blandine",
14   "license": "MIT"
15 }
```

<https://www.npmjs.com/>

NodeJS, le moteur Javascript Module, export, require et npm

Pour générer ce fichier package.json, lancer la commande

```
npm init
```

Et suivre les instructions.

Pour ajouter de nouveaux paquets au projet, et les enregistrer :

```
npm install --save-prod <package-name> //pour un paquet utilisé en production
```

```
npm install --save-dev <package-name>
```

```
// pour un paquet utilisé en développement (par exemple, les outils de tests)
```

<https://docs.npmjs.com/cli/install>

NodeJS, le moteur javascript

L'accès au système de fichier

NodeJS, le moteur Javascript

L'accès au système de fichiers

Il existe deux manières de lire un fichier avec NodeJS

- Synchrone : la lecture bloque l'exécution du script
- Asynchrone : la lecture est non bloquante, le script continue à s'exécuter et un callback est appelé lorsque la lecture est terminée.

Toute la puissance de NodeJS est justement sa capacité à traiter les instructions en asynchrone, nous allons donc nous concentrer sur cette façon de faire.

Pour lire un fichier en asynchrone, on utilise la méthode *readFile* de la class *fs* de NodeJS :

- On récupère la class *fs* via *require*
- *readFile* a trois paramètres en entrée :

- Le chemin du fichier
- L'encoding du fichier (attention : si on ne le précise pas, on récupère un buffer)
- Une fonction de callback, qui sera appelée de manière asynchrone, lorsque la lecture du fichier sera terminée. Le callback a deux paramètres : soit une erreur, si la lecture ne s'est pas bien passée, soit les données du fichier

```
fs.readFile('/Users/joe/test.txt', 'utf8', (err, data) => {
  if (err) {
    console.error(err)
    return
  }
  console.log(data)
})
```

<https://nodejs.org/api/fs.html>

NodeJS, le moteur javascript

Mini TP

NodeJS, le moteur Javascript

Mini TP : Script de lecture de fichier

- Installer NodeJS
- Créer un dossier *my-app*
- Dans ce dossier *my-app*, créer le fichier package.json avec *npm init*
- Créer un fichier *data.csv* contenant :

```
User1;toulouse;  
User2;toulouse;
```

- Créer un fichier *script.js* qui contiendra le code de l'application
- Coder un script qui lit le fichier *data.csv* et affiche son contenu dans le terminal

Résultat attendu :

```
$ node script.js  
data :  
: User1;toulouse;  
User2;toulouse;
```

- **Ajout** : le nom du fichier de données est fourni en argument lors de l'appel au script.
- **Ajout** : une nouvelle ligne de données est fourni en argument et est rajoutée au fichier de données
- **Ajout** : une nouvelle version du script doit être réalisée pour prendre en compte un fichier au format JSON

NodeJS, le moteur javascript

NodeJS comme serveur web

NodeJS, le moteur Javascript NodeJs comme serveur web

NodeJS a des fonctionnalités natives pour fonctionner en serveur web, c'est-à-dire être capable de répondre à des requêtes HTTP.

Le module nodeJS pour cela est *http*.

NodeJS, le moteur javascript

NodeJS comme serveur web

Rappels HTTP

NodeJS, le moteur Javascript

NodeJs comme serveur web – Rappels HTTP

- HTTP : HyperText Transfert Protocol, est un protocole de communication client/serveur pour le web
 - Un client ouvre une connexion vers un serveur
 - Le client effectue une requête et attend
 - Le serveur répond
- Les requêtes HTTP peuvent être de type :
 - GET : Récupération de données
 - POST : Envoi de données au serveur
 - PUT : Mise à jour de données
 - DELETE : Suppression de données
- Les réponses ont un code, et éventuellement des données. Les codes les plus courants sont :
 - 200 : tout s'est bien passé
 - 404 : la ressource n'a pas été trouvée
 - 401 : accès non autorisé
 - 500 : une erreur s'est produite

<https://developer.mozilla.org/fr/docs/Web/HTTP>

<https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode/GET>

<https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode/POST>

<https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode/PUT>

<https://developer.mozilla.org/fr/docs/Web/HTTP/M%C3%A9thode/DELETE>

NodeJS, le moteur javascript

NodeJS comme serveur web

Requetes et réponses HTTP

NodeJS, le moteur Javascript

NodeJs comme serveur web – Requetes et réponses HTTP

Exemple d'un serveur web simple :

```
js server.js > ...
1 const http = require('http') // import du module http
2
3 const port = 3000; // numéro de port sur lequel le serveur va écouter
4
5 const server = http.createServer( // création du serveur HTTP
6   (req, res) => { // à chaque requête arrivant au serveur, cette fonction sera appelée avec req (contenu de la requête) et res (objet de réponse à compléter et renvoyer)
7     res.statusCode = 200 // le code HTTP de retour sera 200
8     res.setHeader('Content-Type', 'text/html') // on informe qu'on va renvoyer du HTML
9     res.end('<h1>Hello, World!</h1>') // HTML que l'on envoie
10  }
11
12 server.listen(port, () => { // quand le serveur a été créé, il se met à écouter sur le port 3000
13   console.log(`Server running at port ${port}`)
14 })
```

On exécute le serveur de la même façon qu'un script simple, puis on peut l'interroger via le navigateur (ou curl, ou autre...)

```
$ node server.js
Server running at port 3000
```



<https://nodejs.dev/build-an-http-server>

NodeJS, le moteur Javascript

NodeJs comme serveur web – Requetes et réponses HTTP

Utiliser le module *http* de nodeJS, sans surcouche, est assez fastidieux. C'est pour cela que nous verrons en seconde partie de ce cours, *express*, un framework web pour Node qui nous simplifiera la vie.

Mais en attendant ...

L'objet *request* passé en paramètre de la fonction *createServer* est un stream. Il faut donc lire ce stream, morceau par morceau (appelé *chunk*) pour récupérer le contenu de la requête.

Imaginons que l'on souhaite envoyer une requête contenant du JSON. Voici le script pour que le serveur puisse le lire :

```
 5  const server = http.createServer((req, res) => {
 6    let data = [];
 7    req.on("data", chunk => {
 8      data.push(chunk);
 9    });
10   req.on("end", () => {
11     JSON.parse(data).todo; // 'Achète du pain!'
12   });
13   res.statusCode = 200;
14   res.setHeader("Content-Type", "text/html");
15   res.end("<h1>Bien reçu!</h1>");
16 });


```

<https://nodejs.dev/get-http-request-body-data-using-nodejs>

NodeJS, le moteur javascript

NodeJS comme serveur web

Les templates

NodeJS, le moteur Javascript

NodeJs comme serveur web – Les templates

Comme avec d'autres serveurs web (Django, php...), on peut renvoyer directement des templates HTML.

Il n'y a pas de solution de templating natif avec NodeJS, mais beaucoup de librairies existent :

- Pug : <https://github.com/pugjs/pug>
- Ejs : <https://github.com/tj/ejs>
- React (et oui, mais généré sur le serveur!) : <https://github.com/reactjs/express-react-views>

Dans ce cours, nous allons utiliser Pug.

Installation via *npm* : `npm install pug`

Création d'un fichier template.pug

```
template.pug
1  p #{name}'s Pug source code!
```

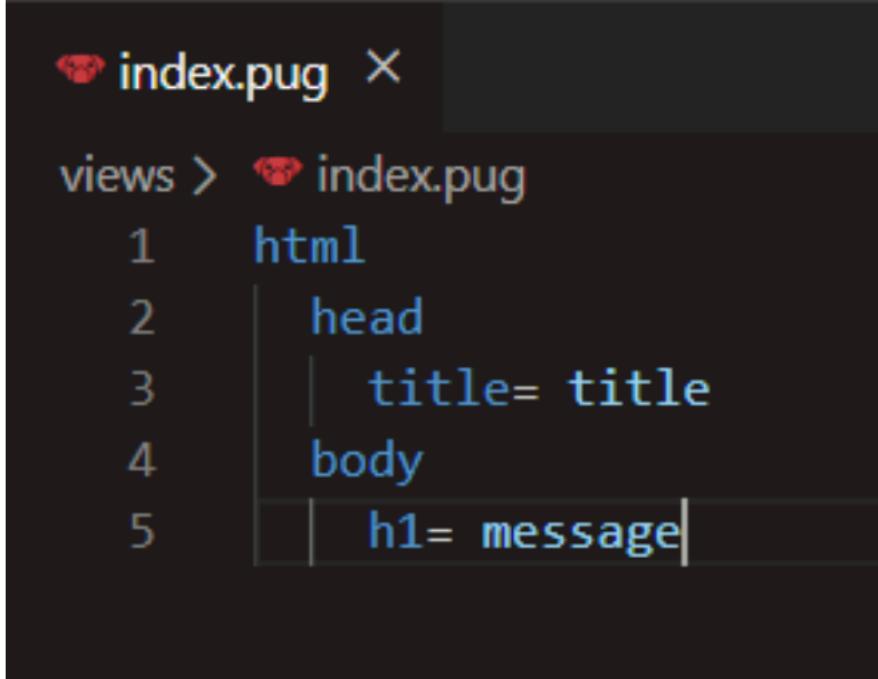
On renvoie ce fichier avec notre serveur :

NodeJS, le moteur Javascript

NodeJs comme serveur web – Les templates

Server.js :

```
JS server.js > ...
1  const http = require("http");
2  const pug = require("pug"); // import du module pug
3
4  const compiledFunction = pug.compileFile('template.pug'); // compilation du template
5  const port = 3000;
6
7  const server = http.createServer((req, res) => {
8      const generatedTemplate = compiledFunction({
9          name: 'Blandine'
10     }); // génération du template avec une variable pour "name"
11
12     res.statusCode = 200;
13     res.setHeader("Content-Type", "text/html");
14     res.end(generatedTemplate); // on renvoie le template
15 });
16
17 server.listen(port, () => {
18     console.log(`Server running at port ${port}`);
19 })
```



The image shows a code editor window with a dark theme. The title bar says "index.pug" with a close button. Below the title bar, the path "views > index.pug" is displayed. The code itself is a Pug template:

```
1  html
2   |   head
3   |     title= title
4   |   body
5     h1= message
```

NodeJS, le moteur javascript

Mini TP

NodeJS, le moteur Javascript

Mini TP : appel du script de lecture via une API web

- Transformer le script de lecture de fichier pour en faire un serveur web
- Lorsque l'on envoie une requête au serveur, il renvoie une page HTML contenant :
 - Le titre : « Voici le contenu du fichier »
 - Un tableau du contenu du fichier, avec des entêtes :

Identifiant	Ville
User1	Toulouse
User2	Toulouse

<https://pugjs.org/language/interpolation.html>
<https://pugjs.org/language/iteration.html>

Bonus : le nom du fichier de données est fourni en paramètre de la requête.