MODULE 1

Add Headings (Format > Paragraph styles) and they will appear in your table of contents.

# Choix du module

Sur node on peut utiliser des modules internes ou externes. Certains modules comme fs (lecture, ecriture, manipulation de fichier) par exemple sont inclus par défaut dans nodejs. D’autres, comme async, sont des modules externes, qui sont téléchargeable via l’utilitaire de package npm.

Les modules internes peuvent être utilisés sans restrictions et ils doivent être préférés aux modules externes lorsque cela est possible (fs par exemple). Lorsqu’ils s'avèrent trop complexes à utiliser ou qu’ils ne contiennent pas les fonctionnalités voulues il est bien sur possible d’utiliser un module externe.

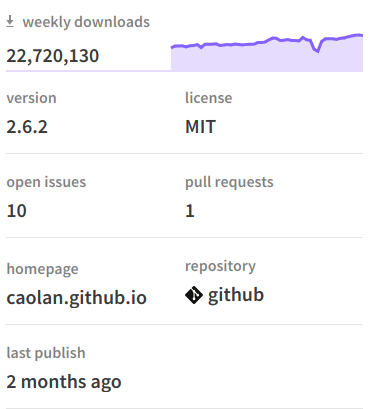
Cependant le choix d’un module externe doit être réalisé judicieusement, un module peu maintenu et peu téléchargé peut se trouver instable (beaucoup de bug et personnes pour les corrigés) et ne doit pas être séléctionné. A l’inverse un module utilisé par des millions de personnes et avec beaucoup de développeurs peut changer au gré des différentes versions (plusieurs fois par an par exemple) et obliger à mettre à jour beaucoup de code au niveau de l’application.

Exemple vécu, mongoose (module de connexion à la base de données) en version 2 permettait d’envoyer l’option suivante : {upsert: 0} ou {upsert: 1} (cela permet de créer un document en base de données).

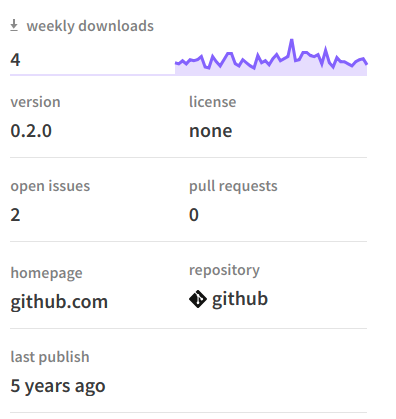
Lors du passage en version 3, les options devaient être définis non pas avec des 0 et des 1 mais des true et des false sinon la fonction renvoyait une erreur. Cela a donc entrainer un update de plusieurs milliers de lignes de codes afin de refaire fonctionner le programme.

L’utilisation d’un module doit donc se faire en fixant une version (on demande à nodejs de ne pas mettre à jour le plugin).

Exemple module populaire async (22 millions de téléchargement par semaine, 2 mois depuis la dernière correction, lien vers la documentation): <https://www.npmjs.com/package/async>



Exemple module à ne pas utiliser (4 téléchargement par semaine, dernier update il y a 5 ans) : <https://www.npmjs.com/package/fileinput>



La règle est donc la suivante :

* Ce n’est pas trop compliqué de le faire à la main => A la main
* Trop compliqué pour faire à la main, pas trop avec un module interne => Utilisation module interne
* Trop compliqué avec un module interne => On utilise un module externe populaire et on fixe la version

# 

# Fs

fs est le module interne node permettant d'interagir avec le filesystem afin de faire les opérations suivantes :

* Listing d’un dossier
* Ecriture de fichiers
* Suppression de fichiers
* Edition de fichier
* Lecture de fichiers
* ...

Toutes les fonctions existent en synchrone et en asynchrone. Pour des raisons évidentes nous détaillerons uniquement les fonctions asynchrones. Nous détaillerons uniquement les fonctions usuelles :

* Listing d’un dossier
* Lecture d’un fichier
* Ecriture d’un fichier

## Listing d’un dossier : readdir(path, callback)

var fs = require(“fs”);

fs.readdir(“./folder1”, function(err, list){

if(err || !list)

{

// erreur à l’ouverture du dossier, inexistant ou droits insuffisants, etc…

console.log(err);

}

else

{

// list est un tableau contenant les fichiers [“./folder1/file1.txt”, “./folder1/file2.txt”, …] ou un tableau vide si le dossier est vide

}

});

## Lecture d’un fichier : readFile(path, encodage, callback)

var fs = require(“fs”);

// Pour encodage utiliser systématiquement utf-8

fs.readFile(“./folder1/file1.txt”, “utf8”, function(err, datas){

if(err)

{

// erreur fichier inexistant ou droits insuffisants.

console.log(err);

}

else if(!datas)

{

// fichier vide

}

else

{

// datas est une string qui contient le contenu du fichier

}

});

## Ecriture d’un fichier : writeFile(path, encodage, callback)

var fs = require(“fs”);

var content = “<html><body>hello</body></html>”;

// Pour encodage utiliser systématiquement utf-8

fs.writeFile(“./folder1/file1.html”, content, “utf8”, function(err){

if(err)

{

// erreur droits insuffisants.

console.log(err);

}

else

{

// le fichier existe désormais, faire actions suivantes….

}

});

# Request

Request est un module permettant d’effectuer facilement des requêtes HTTP/HTTPS. Il est utilisé pour récupérer du contenu externes, envoyé du contenu vers l’extérieur, appeler des apis internes, etc…

Nous allons voir les exemples suivants:

* Requête GET vers une page web pour récupérer du HTML
* Requête POST vers une page web pour envoyer des données
* Requête GET vers une page web pour récupérer du JSON
* Requête POST vers une page web pour envoyer puis récupérer du JSON

Le prototype est le suivant:

request.get(options, callback);

request.post(options, callback);

**options** est un objet javascript qui peut / doit contenir les champs suivants.

* **url**: String, Obligatoire, url du serveur web sur lequel on fait la requête, exemple “<http://www.google.fr/>”,
* **headers** : Objet javascript (“nomduheader”: “valeur”), Optionnel, permet de spécifier des headers à la requêtes http/https
* **followRedirect**: true/false, Optionnel, permet de suivre une redirection 301/302
* **followAllRedirects**: true/false, Optionnel, permet de suivre plusieurs redirection 301/302
* **gzip**: true/false, Optionnel, permet de compresser / décompresser les requêtes si le serveur contient un module de compression
* **json**: true/false, permet d’envoyer/récupérer les données en mode JSON et non en html form standard (x-www-form-urlencoded).
* **form**: objet, permet d’envoyer des données en mode html form (x-www-form-urlencoded)

**callback** est une fonction avec le protype suivant :

**function(err, response, body**) // err contient une erreur si la requête à échouée, response contient le code HTTP de retours et les headers renvoyés, body contient une string si json est false, un objet javascript si json est à true

## Requête GET, récupération HTML

var request = require("request");

request.get({

url: "https://stage.optionizr.com/html",

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log("Erreur");

}

else

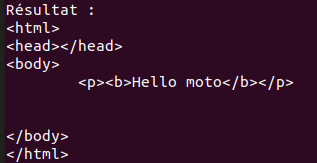
{

console.log("Résultat :");

console.log(body);

}

});



## 

## Requête GET, récupération JSON

*Attention, si le champs json:true n’est pas définis, le résultat retourné est une* ***string*** *et doit donc être parsé avec un JSON.parse*

### Sans json:true

var request = require("request");

request.get({

url: "https://stage.optionizr.com/json",

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log("Erreur");

}

else

{

console.log("Résultat :");

console.log(body.firstname, body.lastname);

}

});



### 

### Avec json:true

var request = require("request");

request.get({

url: "https://stage.optionizr.com/json",

json: true

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log("Erreur");

}

else

{

console.log("Résultat :");

console.log(body.firstname, body.lastname);

}

});



## 

## Redirection HTTP

*Une redirection HTTP (code 301 ou 302) peut être suivie automatiquement (comportement par défaut) par le module request, comme cURL, ou on peut spécifier que l’on ne veut pas suivre la requête et dans ce cas l’url de redirection est renvoyé dans le header Location.*

### Suivi redirection

var request = require("request");

request.get({

url: "https://stage.optionizr.com/redirections1",

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log("Erreur");

}

else

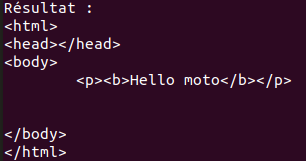
{

console.log("Résultat :");

console.log(body);

}

});



### 

### Interception redirection

var request = require("request");

request.get({

url: "https://stage.optionizr.com/redirections1",

followRedirect: false,

followAllRedirects: false

}, function(err, response, body){

if(err || !response || !response.headers)

{

console.log("Erreur");

}

else

{

console.log("Résultat :");

console.log(response.headers["location"]);

}

});



## 

## Requête POST, form

*Le module request envoi du form html (x-form-www-urlencoded) par défaut. Il n’est pas donc pas nécessaire de préciser le content-type, cependant cela peut ajouter de la lisibilité au code.*

var request = require("request");

request.post({

url: "https://stage.optionizr.com/api/formpost",

headers: {

"Content-Type": "application/x-form-www-urlencoded; charset=UTF-8"

},

form: {

monChamps1: 12,

monChamps4: 3

}

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log(err);

console.log("Erreur");

}

else

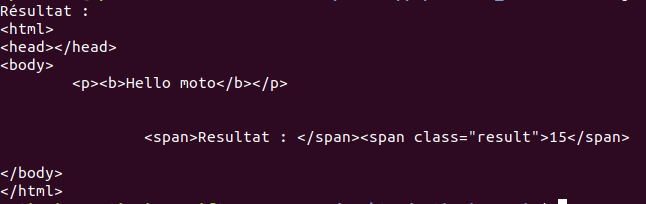
{

console.log("Résultat :");

console.log(body);

}

});



## Requête POST, JSON

var request = require("request");

request.post({

url: "https://stage.optionizr.com/api/formjson",

json: true,

form: {

monChamps1: 12,

monChamps4: 3

}

}, function(err, response, body){

if(err || !response || response.statusCode != 200)

{

console.log(err);

console.log("Erreur");

}

else

{

console.log("Résultat :");

console.log(body.result);

}

});



# 

# 

# Autre modules

Les autres modules important que nous utilisons sont express (serveur, gestion des requêtes) et mongoose (connexion à la base de données). Nous reviendrons sur ces modules car ils méritent un chapitre à part.