W1 - Stack JavaScript

W-JSC-502

Piscine MERN J01

Node.JS

3.1

Piscine MERN J01

repository name: Piscine_MERN_Jour_01
repository rights: ramassage-tek
language: Node.js

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

COMPÉTENCE À ACQUÉRIR

- Javascript
- Node.js
- Express

Détails Administratifs

- Sauf indication contraire, chaque exercice doit être rendu dans un fichier nommé `'server.js'` dans son propre dossier nommé 'ex_`<numero_exercice>`' situé à la racine du projet (par exemple : ex_42/server.js pour l'exercice numéro 42).

Introduction

Pour cette première journée, nous allons apprendre à nous servir de Node.js. Il s'agit en effet du principal outil (serveur web) compris dans la stack MERN.
MERN comprends 4 outils :

- MongoDB : La base de données NoSQL qui vous permet d'appréhender les données au format JSON.
- Express.js : Le Framework en JS, côté serveur, qui vous aide à créer des applications web fonctionnant sous Node.js.
- React.js : Le Framework en JS, côté client, qui vous facilite la vie pour la relation interface utilisateur <-> Serveur.
- Node.js : Le serveur web permettant d'interpréter du code JS et de répondre aux requêtes HTTP des navigateurs.

EXERCICE 1 (2 POINTS)

Dans ce premier exercice, vous devez apprendre à installer votre environnement de base.
Pour cela, vous devez installer l'outil Node.js avec la configuration suivante:

- Le serveur Node.js doit se lancer avec Express.
- Le serveur Node.js doit se lancer sur le port 4242.
- Vous devez lancer le serveur en mode « development ».
- Le nom d'hôte dans la configuration doit être « localhost »

Conseil : Pour réaliser cette configuration, il est fortement conseillé de créer vous-même votre propre fichier de configuration « config.js ». Cependant, il existe des modules pour Node.js comme « nconf » qui vous permettent de gérer aisément la configuration de votre serveur.

EXERCICE 2 (2 POINTS)

Vous devez créer votre première application avec Node.js.
Quand nous nous rendons sur la page : http://localhost:4242/ ; il doit s'afficher « Great ! It works. ».
Pour cela, vous devez créer une première route `GET /` via express, qui renvoie le texte demandé.

EXERCICE 3 (1 POINT)

Maintenant que notre application Node.js renvoie du texte aux visiteurs, il serait mieux de renvoyer du code HTML. C'est pourquoi vous aller devoir changer la route pour qu'elle renvoie du HTML depuis un fichier (et toujours afficher la même chose que l'exercice précédent).

fs

Votre page de retour doit également être du HTML valide W3C et afficher toujours le même texte qu'à l'exercice précédent.

EXERCICE 4 (3 POINTS)

Nous allons apprendre à utiliser les routes.
Lorsqu'un visiteur se rend sur la page http://localhost:4242/name/`<name>`, il doit s'afficher (toujours avec une page HTML valide W3C) : « Hello `<name>` ».
Exemples :

- http://localhost:4242/name/Mike affiche « Hello Mike ».
- http://localhost:4242/name/Thomas affiche « Hello Thomas ».
- http://localhost:4242/name/ affiche « Hello unknown ».

Pour réussir cet exercice, il faut apprendre comment fonctionne les routes dans Node.js avec Express et utiliser du parsing simple.

Se renseigner sur l'utilisation du middleware express

EXERCICE 5 (3 POINTS)

Avoir le paramètre directement dans le path de l'url c'est bien, mais l'avoir sous forme valide des requêtes http usuelles, c'est mieux.
Lorsqu'un visiteur se rend sur la page http://localhost:4242/name/`<name>`?age=`<age>`, il doit s'afficher (toujours avec une page HTML valide W3C) : « Hello `<name>`, you have `<age>` yo ».
Exemples :

- http://localhost:4242/name/Mike?age=24 affiche « Hello Mike, you have 24 yo ».
- http://localhost:4242/name/Thomas?age=23 affiche « Hello Thomas, you have 23 yo ».
- http://localhost:4242/name/ affiche « Hello unknown, i dont know your age ».

EXERCICE 6 (5 POINTS)

Pour cet exercice, **vous ne devez pas rendre de fichier** `server.js`, mais uniquement un fichier `'myMERN_module.js'` dans un dossier `'ex_06'` situé à la racine de dépôt.
Vous allez devoir créer votre propre module qui sera appelé avec la ligne suivante :

```
var myMERN_module = require('./myMERN_module.js');
```

Les fonctions à implémenter sont :

- `myMERN_module.create(name)` doit créer un fichier avec le nom « name », et afficher « Create "name" : OK » si le fichier est créé, et « Create "name" : KO » si la création a échouée.
- `myMERN_module.read(name)` doit lire le contenu du fichier « name » et l'afficher dans le terminal. Si l'action échoue, on affiche « Read "name" : KO »
- `myMERN_module.update(name, content)` doit remplacer le contenu du fichier « name » par le contenu « content ». Si l'action réussie, on affiche « Update "name" : OK ». Si l'action échoue, on affiche « Update "name" : KO »
- `myMERN_module.delete(name)` doit supprimer le fichier « name ». Si l'action réussie, on affiche « Delete "name" : OK ». Si l'action échoue, on affiche « Delete "name" : KO »

Les fonctions dites "synchrones" dont le nom se terminent normalement par `Sync` sont interdites.

Une best practice est de ne garder qu'un export par fichiers.

Tous les affichages doivent se faire dans la console (`console.log(VOTRE_MESSAGE)`)

Pour mieux comprendre les fonctions asynchrones, exécutez le code suivant et observez l'ordre d'affichage.

```
var fs = require("fs");
fs.readFile('async.js','utf8', function(err,data) {
    console.log("File read");
});
console.log("Done");
```

EXERCICE 7 (4 POINTS)

Vous allez devoir créer les routes correspondantes à votre module.
De plus, vous devez modifier votre module pour renvoyer les données, au lieu de les afficher.

- `GET /files/:name` : Apelle `myMERN_module.read(name)`, en lui passant le paramètre « name »
- `POST /files/:name` : Apelle `myMERN_module.create(name)`, en lui passant le paramètre « name »
- `PUT /files/:name/:content` : Apelle `myMERN_module.update(name, content)`, en lui passant le paramètre « name » et « content »
- `DELETE /files/:name` : Apelle `myMERN_module.delete(name)`, en lui passant le paramètre « name »