Programmation Web
TP n° 6 : Serveurs en Node.js

Dans ce TP on se familiarisera d'abord avec **node.js** en utilisant des modules relativement bas niveau : **fs** et **http**. Ensuite on utilisera le module **express.js**, qui propose des fonctionnalités plus haut niveau pour le développement d'un serveur Web, ainsi que le langage de template **ejs**, pour faciliter l'écriture des pages web produites à la volée. Enfin, nous rappelons les concepts des **websocket**.

Vous aurez besoin de l'installer à partir de la page **http://nodejs.org/** avant de continuer.

I) Node.js

1. 'Hello World' en node.js.

Créer un fichier **testNode.js** contenant le code

`console.log('Hello');` exécutez-le à l'aide de la commande en ligne
`node testNode.js`

2. Lecture de fichiers avec le module fs.

A l'aide du module **fs** écrire un programme qui lit un fichier textuel et le reproduit dans la console. Prenez le soin d'afficher un message "reading file, please wait" en attendant la lecture du fichier.

Lire un fichier :

```
var fs = require('fs');

fs.readFile('file.txt', function(err, data) {
    …
});
```

a) A l'aide du module **fs** écrire un programme qui lit les nombres stockés dans un fichier textuel IN et stocke la somme dans un fichier OUT. Le nombre des lignes concerné par la somme est donné dans la 1ère ligne (ici 3).

in.txt
3 4 5 6 5 -5 7 7 5

out.txt
9 11 2

Créer un fichier :

```
var fs = require('fs');
```

```
fs.appendFile('newfile.txt', 'Hello content!', function (err) {
  if (err) throw err;
        console.log('Saved!');
});

var fs = require('fs');

fs.open('newfile.txt', 'w', function (err, file) {
  if (err) throw err;
  console.log('Saved!');
});
var fs = require('fs');

fs.writeFile('newfile.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

C'est quoi la différence entre les trois fonctions ?

- fs.appendFile()
- fs.open()
- fs.writeFile()

Supprimer le fichier in.txt après la production du résultat avec la fonction unlink.

3. Un serveur minimal avec le module http.

a. En utilisant le module http, créer un serveur minimal qui se contente de renvoyer un message textuel (Bonjour). Attacher le serveur au port **8080**, le démarrer avec la commande en ligne **node** et le tester depuis un navigateur.

b. Modifier le serveur pour qu'il lise et renvoie le contenu d'un fichier textuel stocké dans le serveur. Veillez à envoyer un message d'erreur, et à modifier le **status** code de l'entête HTTP, si une erreur se produit dans la lecture du fichier.

c. Modifier le serveur du point précédent pour qu'il renvoie une page web. La page doit être un document HTML bien formé ayant un header avec un titre (h1), un paragraphe contenant le texte lu du fichier, et un footer.

d. Est-ce que la réponse renvoyée par le serveur est bien affichée comme une page Web par le navigateur ? Corriger si besoin.

II) Express.js

Avant de continuer, installer express avec npm (installation locale).

4. Un serveur qui écrit un log.

Créez un serveur express et attachez-lui trois routes de telle sorte qu'il ait le comportement suivant. A la réception de toute sorte de requête, le serveur écrit sur la console "requête reçue à" suivi de l'heure actuelle (en utilisant Date.now() ; ne cherchez pas à rendre le résultat lisible). Ensuite :

- Si la requête est un GET vers '/about' le serveur écrit sur la console "envoie des infos" et répond à la requête avec un message (par exemple "auteur : ...")
- Si la requête n'est pas GET vers '/about' le serveur écrit "abort" sur la console et termine la réponse dans un état d'erreur.

5. Un premier router.

Ecrire un serveur express qui répond aux requêtes GET aux URI suivants : **/ (page d'accueil)** ; **/private** et **/private/mine**, ainsi que **/pictures**. Le serveur renvoie des messages différents pour chaque URI. De plus sur l'URI **/pictures** il envoie un fichier pour le téléchargement.

Prévoir un appel à **.use()** pour traiter les demandes de pages non admises. A l'aide d'un deuxième middleware modifiez le code précédent de façon à écrire la route demandée sur la console à chaque requête.

6. Un router dynamique.

Ecrire un router express qui répond à des URI dynamiques de la forme cours/:numeroducours/descr et qui renvoie le message "Vous avez demandez le cours numero" suivi du numéro de cours qui apparait dans l'URI.

7. Templates.

On veut maintenant écrire un serveur express qui renvoie non plus du simple texte ou des pages HTML minimalistes, mais des pages HTML plus complexes. A tel fin on utilisera des templates.

Installer le système de templates **EJS** (Embedded Java Script) avec npm.

Ecrire une page Web **cours.ejs** qui dépend de trois paramètres : un titre, un descriptif de cours et une liste d'enseignants. La page contient un header avec le logo de l'école et des liens de navigation, une section avec le titre du cours et son descriptif, et une section affichant la liste des enseignants du cours. Ajouter également un footer qui indique le premier enseignant de la liste comme contact.

Ecrire un serveur express monté sur un URI dynamique de la forme cours/:numeroducours/descr.

Le serveur a accès à un tableau contenant, pur chaque numéro de cours, un titre, un descriptif et une liste d'enseignants. (Pour ce tableau, on pourra utiliser une variable globale du serveur, qui simule une plus réaliste base de données.)

La route renvoie la page **cours.ejs** construite à partir des paramètres associés au cours :numeroducours.

Tester le résultat de l'appel à différentes pages web avec uri /cours/i/descr.

III) Traiter un formulaire avec express

8. Formulaire d'inscription.

Créer en HTML un formulaire d'inscription comportant plusieurs champs dont au moins un nom, prénom, login, mdp et confirmation mdp. Le but de cet exercice est de faire en sorte que le serveur vérifie que le mot de passe est correct, et qu'il réponde un message du type "Bonjour Prénom Nom, ton compte est bien créé".

Faire en sorte que la soumission du formulaire provoque l'envoi (submit) des réponses à la page http://localhost:8080/ à l'aide de la m´méthode POST.

a. Créer un serveur express et y ajouter une route POST qui récupère les champs du formulaire.

b. Faire en sorte que cette route renvoie un message de confirmation d'inscription ou d'erreur selon que le mot de passe soit bien dupliqué ou non. En cas d'erreur l'utilisateur est invité à retaper son mot de passe dans le formulaire (sans que les autres champs soient effacés).

c. Lorsqu'un formulaire est soumis, enregistrer les données dans une bases de données (que pour l'instant on simule avec un tableau : les données sont perdues lorsque le serveur s'arrête). Si un utilisateur avec les mêmes nom/prénom remplit à nouveau le formulaire, on le détecte et on envoie un message de modification au lieu du message d'inscription.

9. QCM interactif.

Créer un QCM avec 5 questions, qui auront chacune 3 réponses possibles. Y ajouter un bouton « Corriger » qui envoie le formulaire au serveur.

Faire en sorte qu'initialement seule la première question soit proposée. Les bonnes réponses sont cachées sur le serveur, et la question suivante apparait lorsque la bonne réponse a été donnée. Tant que la réponse n'est pas la bonne, l'utilisateur peu ressayer. (Suggestion : on associera une route à chaque question, pensez à utiliser des routes dynamiques.)

IV) Web Socket

L'objectif est de mettre en place une simple application permettant de trier un tableau en utilisant le tri à bulle optimisé. Pour ce faire, on demande de :

a. Créer un serveur à l'aide des web socket permettant de trier un tableau de n entiers envoyé par client donné.

b. Créer un client permettant d'afficher le tableau avant et après le tri.

c. Modifier le serveur afin d'envoyer chaque itération vers le client après un délai de 1,5 seconde.

d. Modifier le client afin d'afficher chaque itération renvoyée par le serveur.