SD203

Travaux Pratiques

Developpement Web - Côté serveur

Introduction

Le but de ce TP est d'étudier les notions abordées lors du cours SD203 sur les technologies de serveurs Web.

Plutôt que d'installer un serveur Web clés-en-main (comme Apache), de le paramètrer (e.g. modifier les fichiers de configuration httpd.conf) et d'ajouter des comportements spécifiques dans des langages comme PHP, Java, C/C++ ..., ce TP propose de créer un serveur à partir de rien (ou presque). Vous allez programmer en JavaScript, à l'aide de NodeJS et de son support pour les protocoles HTTP/HTTPS. Au fur et à mesure du TP, vous serez amenés à utiliser du code plus avancé.

Compte-Rendu

Pour ce TP, vous devrez rendre les différents fichiers JavaScript produits. Vous répondrez aux questions sous forme de commentaires dans votre code. Vous pouvez répondre en français ou en anglais. Le tout devra être zippé et téléchargé à cette adresse au plus tard le 12 février 23:59.

Pour coder, utilisez le mode "strict" de JavaScript, l'indentation et les commentaires. Vous vous assurerez que votre code est acceptable pour les validateurs (JSLint ou JSHint). Si les validateurs indiquent des "warnings", essayez de les corriger et si ce n'est pas possible expliquez pourquoi dans votre code sous forme de commentaires.

Préambule 1 - Installation de NodeJS pour le debuggage

Il existe de nombreuses versions de NodeJS. Dans ce TP nous avons besoin d'une version récente (mais pas trop). Nous avons besoin de la version v6.3.1 pour vous permettre de débugger pas-à-pas votre code dans une interface graphique. Sur les machines de salle de TP, pour installer cette version, il faut faire plusieurs étapes. Sous Windows ou Mac, installer simplement une version récente et vérfier que node-inspector fonctionne. Cela dit, l'installation de nvm (étape 1 ci-dessous) peut être intéressante: à vous de voir.

1. Installer `nvm`

   Il existe un gestionnaire de version de NodeJS appelé `nvm`. Ce gestionnaire vous permet ensuite d'installer une version spécifique de NodeJS, voire d'en installer plusieurs et de basculer facilement d'une version à l'autre, si vous avez différents projets nécessitant différentes versions. La documentation de `nvm` est ici, mais sous Linux, pour l'installer, il suffit d'executer la commande suivante:

   ```
   curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.0/install.sh |
   ```

   Après l'installation, il faut fermer la fenêtre de commande et en ré-ouvrir une. Vérifier ensuite que `nvm` est bien installé avec la commande:

   ```
   command -v nvm
   ```

   Si tout s'est bien passé, cette commande affiche:

   ```
   nvm
   ```

2. Installer la bonne version de NodeJS

   Sur les machines de TP, pour pouvoir débugger il faut la version **v6.3.1** de NodeJS. Pour l'installer, taper simplement:

   ```
   nvm install v6.3.1
   ```

   Après l'installation, vous devriez voir:

   ```
   Now using node v6.3.1 (npm v3.10.3)
   ```

3. Installation du debugger

   Le debugger s'appelle `node-inspector`. Pour l'installer, executer:

   ```
   npm install -g node-inspector
   ```

4. Debugger un programme NodeJS

   Grace à node-inspector, il est possible de debugger un programme: executer pas-à-pas votre code, placer des points d'arrêt, consulter les variables ... Quand vous lancez node-inspector, une fenêtre Google Chrome s'ouvre avec le code en cours d'execution. Pour lancer le debuggage d'un programme, par exemple appelé "main.js", il faut taper la commande suivante:

   ```
   node-debug main.js
   ```

Préambule 2 - Utilisation des modules NodeJS

NodeJS est basé sur la notion de **module**. Un module est un bloc de code JavaScript que l'on peut charger et utiliser dans son code, c'est à dire comme une bibliothèque. Pour charger un module, NodeJS utilise la fonction `require`. Elle s'utilise comme ceci:

```
var module = require('nom_de_module');
```

Il existe de nombreux modules disponibles pour NodeJS. Certains sont installés automatiquement avec NodeJS (voir la documentation ici). Dans ce cas, il faut juste utiliser `require` pour utiliser le module. C'est le cas pour le module `fs` de gestion du "File System", utilisé pour lire/écrire dans des fichiers (voir la documentation de ce module ici), ou bien encore pour le module `http` qui permet de recevoir et envoyer des messages HTTP (voir la documentation de ce module ici).

Il existe également de nombreux autres modules qu'il faut installer avant de pouvoir utiliser `require`. Pour cela, NodeJS fournit l'outil en ligne de commande `npm`. Pour télécharger et installer un module, on utilisera la ligne de commande suivante:

```
$> npm install nom_du_module
```

Vous pouvez vérifier que le module est installé en vérifiant qu'il existe bien un sous-répertoire du nom du module dans le répertoire `node_modules`. Vous pouvez ensuite utiliser `require`. Par exemple, pour installer le module `express` qui permet de développer des sites Web complexes (documentation ici), on utilisera:

```
$> npm install express
```

Et ensuite, dans le code JavaScript, on pourra faire:

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

Vous n'utiliserez le module `express` qu'à partir de l'exercice 2. Les questions précédentes ne devront pas utiliser ce module, mais uniquement le module `http`.

Exercice 1 - Serveur Web Simple

Question 1a: En utilisant uniquement le module `http`, dans un fichier appelé server1a.js, écrire un programme JavaScript NodeJS qui réalise un serveur Web sur le port 8000 qui renvoit une chaîne de caractère HTML confirmant que le serveur fonctionne quand on accède à la racine du serveur, c'est-à-dire "http://localhost:8000/". Pour cela vous pouvez utiliser la méthode `createServer` du module `http`. Cette méthode utilisera une fonction de rappel (callback) qui prend 2 paramètres: un objet `request` contenant les informations relatives à la requête (URL, méthodes, entêtes ...) et un objet `response` qui contient des méthodes pour créer et envoyer la réponse, en particulier la méthode `response.end`.

Question 1b: En utilisant le module `fs`, modifier le serveur Web précédent pour qu'il serve les fichiers présents dans le répertoire du serveur et tous les sous-répertoires. Enregistrer le nouveau code dans un fichier appele server1b.js. On ne traitera que les requêtes HTTP de type GET. L'accès à la page d'accueil devra toujours fonctionner.
Le module `fs` propose différentes méthodes pour vérifier l'existence d'un fichier, ou lire le contenu d'un fichier. Vous commenterez quelles méthodes ont été utilisées, en particulier concernant le choix synchrone/asynchrone. On s'assurera que le serveur répondra avec le code 404 si le fichier n'existe pas. On vérifiera que le serveur ne sert pas les fichiers dans les répertoires parents, c'est-à-dire utilisant "http://monserveur.com/../file.txt". On s'assurera de répondre avec le bon type MIME pour les fichiers les plus courants (HTML, CSS, JS, PNG, ...). Vous controlerez vos résultats avec l'inspecteur Réseau de votre navigateur Web.

Question 1c: Modifier le serveur Web précédent pour qu'il traite les requêtes GET pour l'adresse "http://monserveur.com/hello?name=xxxx" ou xxxx est une chaîne de caractères. Enregistrer le nouveau code dans un fichier appele server1c.js. Le serveur répondra avec du code HTML disant "Bonjour xxxx". xxxx devra pouvoir contenir des accents, des espaces ... Vous pourrez utiliser la méthode `unescape` du module `querystring`. Testez votre serveur en écrivant une page HTML (nommée exercice1c.html) contenant un formulaire qui émet ces requêtes. On pourra vérifier les vulnérabilités XSS (injection de code HTML, injection de code JavaScript) mais on ne les corrigera pas.

Question 1d: Modifier le serveur précédent pour sauvegarder en mémoire toutes les valeurs de "nom" reçues. Le serveur répondra, à chaque nouvelle requête, avec une réponse HTML du type "Bonjour xxxx, les utilisateurs suivants ont déjà visités cette page: yyyy, zzzz, aaaa, bbbb". Vérifier la vulnérabilité XSS. Expliquer en quoi cette vulnérabilité est plus grave dans ce cas. Corriger cette vulnérabilité, notamment pour `name=<b>Toto</b>` et `name=<script>alert('coucou');</script>`.

Question 1e (bonus): Modifiez votre serveur pour qu'il fonctionne aussi en HTTPS.

Question 1f (bonus): Implémentez une redirection de la page "http://serveur.com/redirect" vers la page d'accueil.

Exercice 2 - Un serveur plus flexible

Avec l'exercice 1, on voit rapidement que la fonction de gestion des requêtes devient complexe, avec des tests du type "si l'URL contient ..." ou "si la requête est de type ....". De plus, à chaque fois qu'on voudra ajouter un nouveau type d'URL ou de méthode, il faudra modifier cette fonction avec le risque d'introduire des bugs. Pour simplifier la gestion des requêtes, de nombreux développeurs NodeJS utilisent le module `express` qui implémente un principe générique des serveurs Web modernes: le routage.

Question 2a: Reproduire le serveur de l'exercice 1, en utilisant `express`. Vous pourrez en particulier utiliser la fonctionnalité `static` et vous inspirer de ce tutoriel. Enregistrer votre code dans le fichier server2a.js.

Question 2b: Une particularité du module `express` est qu'il permet de définir une méthode pour traiter des URLs variables de façon générique. La partie variable de l'URL est extraite et passée en paramètres à la fonction qui traite ces URLs. Par exemple, si on déclare une route avec l'URL `"/hello/:language"`

alors, du fait de l'utilisation du symbole ':', `express` ajoutera une propriété `params.language` à la variable `request`. Modifiez donc votre méthode de la question 1c pour générer du HTML dans une langue différente en fonction de l'URL utilisée. Sauvegarder ce résultat dans le fichier server2b.js.

Question 2c: Le module `express` permet facilement d'indiquer une fonction pour gérer les messages HTTP de type POST. Modifier votre code précédent et le fichier précédent exercice1c.html (exercice2c.html) pour envoyer les données en utilisant la méthode POST. On utilisera le module `body-parser` qui indique à `express` de mettre les données dans la variable `request.body`. Vérifiez si les vulnérabilités des questions 1c et 1d ont changé. Corrigez éventuellement.

Question 2d: Comme vous l'avez vu, il est un peu fastidieux de devoir construire une réponse HTML en concaténant des chaînes de caractères avec le contenu de variables. Une approche utilisée dans le développement moderne consiste à utiliser des modèles ("templates"), parfois appelés des vues ("views"). Le module `ejs` est entre autre utilisé pour cela. Avec ce module, construisez un template HTML/CSS pour votre page hello et peupler le template avec les informations reçues dans la requête. On pourra par exemple créer un tableau des personnes ayant utilisées la page et la langue utilisée. On n'utilisera pas nécessairement le concept de "view".

Exercice 3 - Utilisation simple d'une base de données MySQL (Bonus)

Pour cette question on utilisera le module `mysql`. Vous pouvez utiliser la base de données avec les informations suivantes:

- serveur: murillo.enst.fr
- login: tpX avec X=1 ... 120
- password: tpX
- database: tpX

Question 2a: Modifier votre serveur pour qu'il offre une API REST pour ajouter/consulter/modifier/supprimer des "news" dans la base de données. Une "news" comportera un titre, une date, un auteur, une langue et un contenu.

Question 2b: Ajouter une page de login et la gestion de cookies pour permettre l'ajout, la modification ou la suppression de données dans la base seulement quand quelqu'un est connecté.