INE3: Communication P2P en temps réel sur le web

TP2 : la plateforme NODEJS

Introduction

L'objectif de ce TP est de mettre en pratique certains des concepts discutés dans le cours et ce à travers le développement d'une application temps réel sur web en se basant sur les technologies HTML, HTTP et NODEJS. Des extensions sont exigées à la fin pour couvrir d'aspects du développement temps réel sur le web, surtout la plateforme «MEAN STACK».

Cahier de charges

L'application à développer doit permettre à ses utilisateurs d'interagir avec le serveur pour lui demander d'exécuter des requêtes sur son système de fichiers et retourner les résultats en les affichant sur la fenêtre du navigateur client. Pour pouvoir utiliser l'application, un utilisateur doit d'abord s'inscrire auprès du système en introduisant son nom et son prénom et en choisissant un login et un mot de passe pour des fins d'authentification. Les informations saisies par l'utilisateur doivent être inscrites dans une collection en mémoire ou dans une base de données NoSQL telle que MongoDB.

Les utilisateurs possédant un compte dans le système peuvent accéder à un ensemble de fonctionnalités/services en fournissant l'URI correspondant (login, logout, upload, show, find, …). Le service *login* doit servir à ouvrir une session avec le serveur alors le service *logout* permet de mettre fin à la session courante. Le service *upload* permet à l'utilisateur de choisir une image pour son profil et la sauvegarder sur le disque du serveur pour pouvoir la retourner plus tard. Le service *show*, quant à lui, permet à l'utilisateur de retourner et afficher sur son navigateur la dernière image qu'il a chargée sur le serveur, sinon une image par défaut sur le système de fichiers. Enfin, le service *find* doit être demandé par l'utilisateur lorsqu'il veut afficher sur son navigateur l'ensemble des fichiers du serveur. Il est à noter que si l'utilisateur ne spécifie pas le service demandé lors de la formulation d'une requête (c'est-à-dire, seul l'URL du serveur est indiqué) le serveur lui affichera un message d'accueil avec possiblement l'ensemble des services disponibles. Par contre, lorsque l'utilisateur demande un service non-existant le serveur lui présentera un message d'erreur.

Pour implémenter l'ensemble des services de l'application, on vous demande d'utiliser la plateforme NodeJS pour la mise en place d'un serveur modulaire pour faciliter la maintenance et l'évolution de l'application. Je vous demande aussi de penser à la modélisation de la solution en vous servant de certains diagrammes UML.

Environnement de travail

Comme a été indiqué auparavant, l'implémentation de l'application doit se faire en utilisant la plateforme NODEJS. NodeJS est une plateforme construite sur le moteur JavaScript V8 de Chrome qui permet de

développer des applications en utilisant du JavaScript. Il se distingue des autres plateformes grâce à une approche non bloquante permettant d'effectuer des entrées/sorties (I/O) de manière asynchrone.

La première des choses à faire est de télécharger et installer NODEJS et les modules nécessaires pour le développement des services de votre solution. Pour ce faire, il faut aller sur le site officiel de la plateforme (https://nodejs.org/) et télécharger la version correspondante à votre système d'exploitation. Vous pouvez aussi installer la plateforme NODEJS en utilisant un gestionnaire de packages tel que dnf/yum ou apt-get. À la fin de l'installation, vous aurez les composants *node* et *npm* disponibles sur votre système : *node* est l'environnement d'exécution de NODEJS alors que *npm* (Network Package Manager) est le gestionnaire de modules de NODEJS pour installer les modules externes nécessaires pour les applications à développer. Rappelons que NODEJS vient avec un ensemble de modules-core qui ne nécessitent aucune installation particulière de la part de l'utilisateur. Cependant, pour installer un nouveau module externe il suffit de taper la commande : *npm install nom-du-module*

Implémentation de la solution

Pour pouvoir comprendre le fonctionnement de NODEJS, nous allons procéder à une implémentation de l'application par raffinement successif. Nous allons surtout nous intéresser à l'aspect architectural de l'application en mettant la logique-métier à côté. Une telle logique-métier peut être ajoutée facilement en écrivant le code de chaque composant de l'architecture.

1ère version de l'application :

1) Créer un fichier server.js et y mettre le code suivant :

```
var http = require("http");
http.createServer(function(request, response) {
response.writeHead(200, {"Content-Type": "text/plain"});
response.write("Hello World");
response.end();
}).listen(8888);
```

2) Ouvrir un terminal sur votre système et exécuter la commande : *node server.js*
3) Ouvrir un navigateur puis taper l'URL : *http://localhost:8888/*
4) Vérifier le résultat et penser à comment améliorer cette implémentation.

2ème version de l'application :

Dans cette implémentation, nous allons essayer de rendre modulaire notre application. Pour ce faire, nous allons décomposer l'implémentation en deux modules :

1) Le module server.js contenant le code suivant :

```
var http = require("http");

function start() {
  function onRequest(request, response) {
  console.log("Request received.");
  response.writeHead(200, {"Content-Type": "text/plain"});
```

```
  response.write("Hello World");
  response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
exports.start = start;
```

2) Le module index.js contenant le code suivant :

```
var server = require("./server");
server.start();
```

3) Exécuter la commande : *node index.js* sur un terminal
4) Ouvrir un navigateur puis taper l'URL : http://localhost:8888/
5) Vérifier le résultat et comparer la solution à la précédente.
6) Penser à comment améliorer cette implémentation.

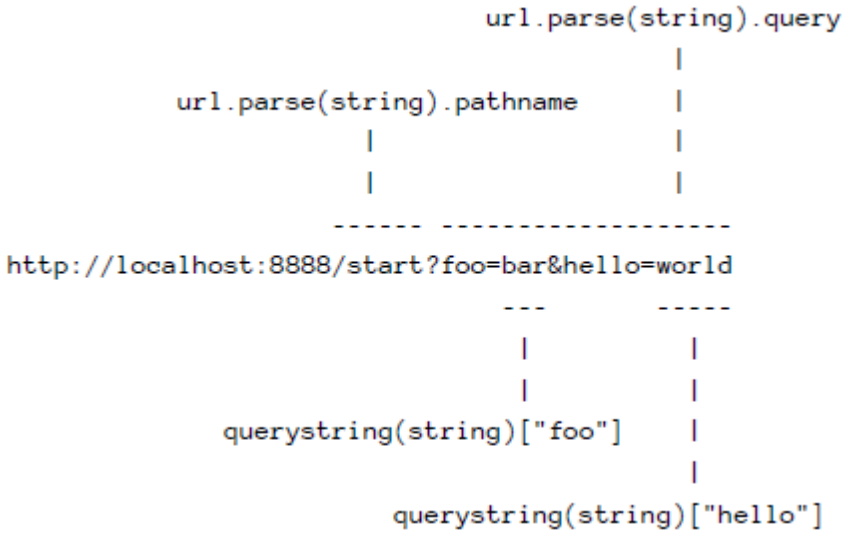
3ème version de l'application :

Dans cette implémentation, nous allons introduire un 3ème module pour se charger de routage des requêtes vers leurs propres processeurs («request handlers» en anglais). Ainsi, nous aurons les trois modules suivants :

1) Le module server.js contenant le code suivant :

```
var http = require("http");
var url = require("url");

function start(route) {
  function onRequest(request, response) {
      var pathname = url.parse(request.url).pathname;
      console.log("Request for " + pathname + " received.");

      route(pathname);

      response.writeHead(200, {"Content-Type": "text/plain"});
      response.write("Hello World");
      response.end();
      }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
  }
exports.start = start;
```

Dans ce code, nous avons utilisé le module "url" qui permet ce découpage:



2) Le module index.js contenant le code suivant :

```
var server = require("./server");
var router = require("./router");

server.start(router.route);
```

3) Le module router.js contenant le code suivant :

```
function route(pathname) {
  console.log("About to route a request for " + pathname);
}

exports.route = route;
```

4) Exécuter la commande : *node index.js* sur un terminal
5) Ouvrir un navigateur puis essayer de demander les différents services de l'application.
6) Vérifier le résultat et comparer la solution à la précédente.
7) Penser à comment améliorer cette implémentation.

4ème version de l'application :

Dans cette version, notre implémentation est composée de quatre modules dont un de ces modules est appelé «*requestHandlers*.js» et sert à traiter les requêtes différemment. Le code de l'implémentation se présente comme suit :

1) Le module server.js contenant le code suivant :

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
```

```
    console.log("Request for " + pathname + " received.");

    route(handle, pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}
exports.start = start;
```

2) Le module index.js contenant le code suivant :

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
handle["/find"] = requestHandlers.find;
handle["/show"] = requestHandlers.show;
handle["/login"] = requestHandlers.login;
handle["/logout"] = requestHandlers.logout;

server.start(router.route, handle);
```

3) Le module router.js contenant le code suivant :

```
function route(handle, pathname) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
     handle[pathname]();
  } else {
     console.log("No request handler found for " + pathname);
  }
}

exports.route = route;
```

4) Le module requestHandlers.js contenant le code suivant :

```
function start() {
  console.log("Request handler 'start' was called.");
}

function upload() {
  console.log("Request handler 'upload' was called.");
}
```

```
function find() {
  console.log("Request handler 'find' was called.");
}

function show() {
  console.log("Request handler 'show' was called.");
}

function login() {
  console.log("Request handler 'login' was called.");
}

function logout() {
  console.log("Request handler 'start' was called.");
}

exports.start = start;
exports.upload = upload;
exports.find = find;
exports.show = show;
exports.login = login;
exports.logout = logout;
```

5) Exécuter la commande : *node index.js* sur un terminal
6) Ouvrir un navigateur puis essayer de demander les différents services de l'application.
7) Vérifier le résultat et comparer la solution à la précédente.
8) Penser à comment améliorer cette implémentation.

5ème version de l'application :

Dans cette version, notre implémentation est améliorée en permettant aux différents modules de retourner leurs résultats au serveur pour qu'il puisse les retourner à son tour au client/utilisateur moyennant son objet requête HTTP. Le code des quatre modules se présente alors comme suit :

1) Le module server.js contenant le code suivant :

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    var content = route(handle, pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write(content);
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

2) Le module index.js contenant le code suivant :

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
handle["/find"] = requestHandlers.find;
handle["/show"] = requestHandlers.show;
handle["/login"] = requestHandlers.login;
handle["/logout"] = requestHandlers.logout;

server.start(router.route, handle);
```

3) Le module router.js contenant le code suivant :

```
function route(handle, pathname) {
  console.log("About to route a request for " + pathname);
```

```
  if (typeof handle[pathname] === 'function') {
     return handle[pathname]();
  } else {
     console.log("No request handler found for " + pathname);
     return "404: Resource not found";
  }
}

exports.route = route;
```

4) Le module requestHandlers.js contenant le code suivant :

```
function start() {
  console.log("Request handler 'start' was called.");
  return "Hello start";
}

function upload() {
  console.log("Request handler 'upload' was called.");
  return "Hello upload";
}

function find() {
  console.log("Request handler 'find' was called.");
  return "Hello find";
}

function show() {
  console.log("Request handler 'show' was called.");
  return "Hello show";
}

function login() {
  console.log("Request handler 'login' was called.");
  return "Hello login";
}

function logout() {
  console.log("Request handler 'start' was called.");
  return "Hello logout";
}

exports.start = start;
exports.upload = upload;
exports.find = find;
exports.show = show;
exports.login = login;
exports.logout = logout;
```

5) Exécuter la commande : *node index.js* sur un terminal
6) Ouvrir un navigateur puis essayer de demander les différents services de l'application.

7) Changer le code de la fonction *start()* comme suit et exécuter les étapes 5 et 6 surtout en demandant les deux services en même temps en commençant par le services start.

```
function start() {
  console.log("Request handler 'start' was called.");

  function sleep(milliSeconds) {
    var startTime = new Date().getTime();
    while (new Date().getTime() < startTime + milliSeconds);
  }

  sleep(60000);
  return "Hello Start";
}
```

7) Vérifier le résultat et comparer la solution à la précédente.
8) Penser à comment améliorer cette implémentation.

6ème version de l'application :

Dans cette version de notre implémentation, nous tenons en considération l'exécution asynchrone des différents services. Pour ce faire, nous allons éliminer les valeurs retournées par les fonctions services et nous les remplaçons par des écritures directement dans l'objet réponse HTTP passé comme argument aux différentes fonctions. Nous obtenons le code suivant :

1) Le module server.js contenant le code suivant :

```
var http = require("http");
var url = require("url");

function start(route, handle) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(handle, pathname, response);
}

http.createServer(onRequest).listen(8888);
console.log("Server has started.");
}

exports.start = start;
```

2) Le module index.js contenant le code suivant :

```
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}
```

```
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/upload"] = requestHandlers.upload;
handle["/find"] = requestHandlers.find;
handle["/show"] = requestHandlers.show;
handle["/login"] = requestHandlers.login;
handle["/logout"] = requestHandlers.logout;

server.start(router.route, handle);
```

3) Le module router.js contenant le code suivant :

```
function route(handle, pathname, response) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
     handle[pathname](response);
  } else {
     console.log("No request handler found for " + pathname);
     response.writeHead(404, {"Content-Type": "text/plain"});
     response.write("404: Resource not found");
     response.end();
  }
}

exports.route = route;
```

4) Le module requestHandlers.js contenant le code suivant :

```
var exec = require("child_process").exec;

function start(response) {
  console.log("Request handler 'start' was called.");
  setTimeout( function() {
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello start");
    response.end();}, 60000);
}

function find(response) {
  console.log("Request handler 'start' was called.");
  exec("find /",
      { timeout: 10000, maxBuffer: 20000*1024 },
        function (error, stdout, stderr) {
          response.writeHead(200, {"Content-Type": "text/plain"});
          response.write(stdout);
          response.end();
      });
}

function upload(response) {
  console.log("Request handler 'upload' was called.");
```

```
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Upload");
  response.end();
}

function show(response) {
  console.log("Request handler 'show' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello show");
  response.end();
}

function login(response) {
  console.log("Request handler 'login' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello login");
  response.end();
}

function logout(response) {
  console.log("Request handler 'start' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello logout");
  response.end();
}

exports.start = start;
exports.upload = upload;
exports.find = find;
exports.show = show;
exports.login = login;
exports.logout = logout;
```

5) Exécuter la commande : *node index.js* sur un terminal.
6) Ouvrir un navigateur puis essayer de demander les différents services de l'application Possiblement en même temps.
7) Vérifier le résultat et comparer la solution à la précédente.
8) Penser à comment améliorer cette implémentation.

Extensions à faire et à remettre

Pour bénéficier au maximum de ce TP, je vous demande de faire les extensions suivantes :

1) Compléter le code (la logique-métier) de chacun des services figurant dans le module *requestHandlers.js*
2) Migrer l'implémentation de l'application en *Socket.io* en fournissant le code de la partie client et celui de la partie serveur.
3) Donner l'architecture générale d'une application écrite en MEAN STACK.
4) Intégrer le Framework Express dans l'application pour implémenter l'aspect routage.
5) Intégrer le Framework Angular dans l'application pour implémenter l'aspect présentation.

6) Intégrer le Framework Mongoose et la base de données MongoDB pour implémenter le stockage et la persistance de données de l'application.
7) Esquisser quelques diagrammes UML de la solution que vous jugez importants.
8) Dresser une synthèse de tout ce que vous avez appris dans ce TP.

Bon apprentissage!