TP4 - Programmation serveur et API REST

Polytech Nantes - Info 3

Objectifs du TP

Dans le TP1 vous avez vu comment créer et mettre en forme une page HTML statique. Dans le TP2 vous avez fait vos premiers programmes JavaScript en manipulant des données au format JSON. Dans le TP3 vous avez programmé la partie client d'une application web permettant de parcourir des données musicales (genres, artistes, albums).

Dans ce TP4, nous vous proposons de **programmer** (en JavaScript) **la partie serveur de l'application web** permettant de parcourir des données musicales. Vous verrez en particulier les notions suivantes :

- **Mise en place d'un serveur web** directement avec l'objet http de nodejs ainsi que via le framework web ExpressJS.
- **Utilisation des entêtes HTTP** pour répondre à une requête HTTP provenant d'un navigateur web.
- Utilisation du système de routing d'ExpressJS pour **mettre en place une API REST** simple.
- **Faire des requêtes fetch coté serveur** afin de récupérer des données et les renvoyer (transformées) vers le client.
- **Utiliser xpath** afin de transformer des données XML en Json.

Outils

Pour réaliser ce TP vous aurez besoin de :

- Un éditeur de texte au choix (gedit, sublime text, notepad++, etc.)
- Votre navigateur préféré.

Nous allons également utiliser deux outils pour simplifier le développement :

- **npm** (node package manager) nous permettra d'installer les outils de validation de votre code ainsi qu'un outil permettant de recharger votre page HTML à chaque fichier modifié.
- **gulp** va permettre d'automatiser la réalisation des tâches décrites ci-dessus à chaque modification des fichiers JavaScript.

Installation

Vous trouverez sur Madoc une archive contenant les données et scripts nécessaires à la réalisation de ce TP. Téléchargez cette archive et extrayez son contenu.

Pour chaque exercice, vous devrez :

- A la racine du répertoire de l'exercice (là ou est stocké votre fichier `package.json`) lancer l'installation de tous les outils en tapant la commande suivante dans un terminal : `npm install`
- Maintenant que les outils sont installés vous pouvez lancer gulp. On va le lancer via une commande npm : `npm test`.
- Cette commande devrait commencer à surveiller vos fichiers JavaScript (comme au TP2).
- Pour lancer notre serveur, nous devrons également saisir (**dans une autre console**) la commande : `npm start`.
- Si le serveur est bien lancé, vous devez pouvoir consulter votre application web à l'adresse suivante : http://127.0.0.1:8080. Pour l'instant ça ne marche pas puisque vous devez écrire le code du serveur dans la suite de ce TP…

Préliminaires

Dans cette première partie, nous allons réaliser deux serveurs web très simples, n'utilisant pas le travail fait au TP3. **Les exemples de code serveur nodejs donnés en cours pourront vous être utiles.**

Exercice 1.1 - Serveur texte

Nous allons réaliser un serveur web minimaliste dont la seule fonctionnalité est de renvoyer le texte `Bonjour tout le monde !` à chaque fois qu'une requête est effectuée. La réponse sera renvoyée au format texte (pas HTML), quelle que soit la méthode HTTP utilisée pour la requête (GET, POST, PUT, etc.).

Pour cela vous devrez :

- Importer le module `http` de nodejs via un `require`.
- Utiliser la méthode createServer() de l'objet `http` afin de créer un serveur web.
  - Cette méthode prend en paramètre une fonction callback qui sera appelée à chaque requête avec deux paramètres : `request`, de type IncomingMessage et `response` de type ServerResponse.
  - On ne se servira pour l'instant pas de `request` qui permet d'accéder à toutes les informations de la requête HTTP (url, entête, corps de la requête, etc.).
  - On utilisera la méthode response.writeHead() afin d'envoyer un code de statut 200 pour indiquer au navigateur que la requête s'est bien passée.
  - On utilisera la méthode response.write() afin d'envoyer notre message au format texte brut.
  - Enfin, on terminera le traitement de la réponse avec un appel à la méthode response.end() afin de signaler que le traitement de la réponse est terminé et que la réponse peut être envoyée au navigateur.
- N'oubliez pas l'appel à la méthode listen() une fois votre objet `http` créé.

Vous pouvez tester votre serveur après l'avoir lancé avec la commande `npm start`. Une visite de l'adresse http://127.0.0.1:8080 devrait afficher votre message dans le navigateur.

Exercice 1.2 - Serveur html

Le serveur précédent fonctionne, mais est très (très) simple. Il répond toujours la même chose, répond même lorsque la méthode HTTP utilisée n'est pas GET, et ne renvoie aucun entête HTTP (pas même `Content-Length`, pourtant requis…).

Nous allons donc dans cet exercice modifier le code de l'exercice précédent (faites en une copie avant) afin de :

- Ne répondre par un statut 200 que lorsque la méthode HTTP utilisée est GET. Sinon, un code 405 sera renvoyé. Vous aurez pour cela besoin de l'attribut `request.method`.
- Renvoyer une réponse au format HTML, encodée en utf8 (avec un doctype, un entête, un body, etc.). Celle-ci aura pour titre 'Bonjour' et affichera le message suivant (pour une requête à l'adresse http://127.0.0.1:8080/un/repertoire/index.html?arg1=12&arg2=23) :

  ```
  Analyse de votre requête:
  Vous accédez à l'url: un/repertoire/index.html
  La chaine de requête est: arg1=12&arg2=23
  ```

  La séparation du chemin de l'url et de la chaine de requête peut être réalisée à l'aide d'un simple split() sur `request.url` ou via la classe URL de nodejs et son attribut `search`.
- Compléter les entêtes `Content-Type` et `Content-Length` lors de votre appel à response.writeHead(). Pour connaitre la longueur (en octets) de la chaine de caractères contenant votre réponse, vous pouvez utiliser la méthode Buffer.byteLength().

Exercice 2 - API REST via fichier JSON

Dans l'exercice 1.2 nous avons réalisé un serveur web directement avec le module `http` de nodejs. Bien que fonctionnelle cette mise en œuvre n'est pas très efficace : il est nécessaire de traiter "à la main" tous les aspects du protocole HTTP. Pour cet exercice et le suivant, nous utiliserons le framework ExpressJS, qui simplifie (entre autres) la gestion du protocole HTTP.

Mise en place du serveur web

Complétez le fichier serveur.js situé à la racine du répertoire de l'exercice 2. Les étapes à suivre sont les suivantes :

- Importez le module `express` (via un require) et créez une instance d'`express` via l'appel à son constructeur. Cette instance sera stockée dans une variable appelée `app`.
- Utilisez la méthode app.use() d'express ainsi que la méthode express.static() afin d'indiquer à express que tout ce qui sera à la racine du site sera servi comme du contenu statique et que ce contenu sera situé dans le répertoire `public`. Il vous faudra également utiliser la méthode app.listen() afin de créer le serveur et le faire écouter sur le port 8080.
- Importez le module `api`, situé à l'emplacement `./api/api`. Ce module ne contient pour l'instant pas de code, nous l'écrirons dans la partie 'mise en place de l'API'.
- Utilisez la méthode app.use() d'express afin de créer une route qui redirigera tous les appels vers http://127.0.0.1:8080/api/n_importe_quoi_d_autre vers le module `api`.

Si tout va bien votre application doit maintenant servir le contenu du répertoire `public` lorsqu'on visite la page http://127.0.0.1:8080.

Le contenu affiché ne doit cependant pas être très intéressant puisqu'il s'agit du code du début du TP3. Pour améliorer les choses et permettre à la suite du TP de fonctionner vous devez recopier le code que vous avez produit au TP3 dans le répertoire `public`. Bien entendu, tout ne fonctionnera pas encore puisqu'il va falloir que nous réalisions l'API REST renvoyant les genres, artistes et albums. Il vous faut aussi modifier légèrement le code Javascript du TP3 afin que vos requêtes se fassent sur des adresses relatives à votre API (ex :`api/genres`) plutôt qu'absolues (ex :`localhost:3000/genres`).

Mise en place de l'API

Dans cette partie nous allons mettre en place une partie de l'API REST que nous avons utilisé au TP3. On ne traitera ici que des requêtes `GET` :

- /genres/ renverra un document JSON contenant la liste des genres. Le format des données sera exactement le même que la partie `genres` du fichier `db.json` situé dans le répertoire `api/data/`.
- /genre/id_de_genre/artists renverra un document JSON contenant la liste des artistes correspondant au genre `id_de_genre`.
- /artist/id_de_l_artist/albums renverra un document JSON contenant la liste des albums correspondant a l'artiste `id_de_l_artist`. **Cette partie de l'API est à faire en bonus**.

Le format des données sera exactement le même que les parties `genres`, `artists` et `albums` du fichier `db.json` situé dans le répertoire `api/data/`.

Dans cet exercice on chargera simplement les données à partir du fichier `api/data/db.json`. **Il faut donc le charger via un (simple)** `require()`.

Liste des genres (/genres/)

- A l'aide d'un appel à app.get(), créez une route qui répondra aux requêtes GET à l'adresse /genres/.
- Dans le callback de app.get(), récupérez l'ensemble des genres présents dans `db.json` dans une chaine de caractères (via JSON.stringify()).
- Le callback de app.get() prend en paramètre un objet `req` de type Request et un objet `res` de type Response. Utilisez la méthode res.set() afin de mettre l'entête `Content-Type` à `application/json; charset=utf-8`.
- Toujours dans le callback, utilisez res.send() afin de renvoyer vos données JSON.

Lancez votre serveur et vérifiez que la liste des genres s'affiche bien dans la liste déroulante de votre page web.

Liste des artistes correspondant à un genre (/genre/id_de_genre/artists)

Faites de même pour /genre/id_de_genre/artists. Les choses sont un peu plus compliquées, car on ne doit renvoyer que les artistes correspondant au genre passé dans l'url. Voici donc quelques indications :

- Vous pouvez récupérer le genre passé dans l'url en découpant la chaine à l'aide d'un simple String.prototype.split() ou via le mécanisme de paramètre de route d'`express`.
- L'url est 'url-encodé', vous pouvez la décoder avec decodeURIComponent() (ça évite par exemple d'avoir des %20 à la place des espaces dans les noms de genre).

- Le filtrage des artistes peut être réalisé grâce à Array.prototype.filter.
- Si le genre passé dans l'url n'existe pas dans votre base de données, vous devez renvoyer une erreur 404.
- Vous pouvez remplacer l'appel à res.set() + res.send() utilisé tout à l'heure par un appel à res.json() plus simple…

Exercice 3 - API REST via `fetch`

Dans ce dernier exercice, nous allons récupérer les données via l'API REST de Lastfm. Pour corser un peu le travail, les données seront récupérées en XML et devront être transformées en JSON pour être envoyées au navigateur. **Les IDIA n'ayant pas eu de cours de XML, ils récupéreront directement les données en JSON et ignoreront les instructions concernant le XML.**

Votre fichier `serveur.js` sera le même que pour l'exercice 2, vous n'avez pas à le modifier. Il va par contre falloir réécrire le fichier `api.js`.

Vous aurez besoin des modules suivants (déjà installés via `npm` et votre fichier `package.json`), **à charger via des appels à `require()`** :

- node-fetch permet d'utiliser l'API fetch dans nodejs. Par défaut nodejs ne fournit pas d'implémentation de `fetch()`. Une fois ce module chargé vous pouvez utiliser fetch comme vous l'avez fait pour la partie client du TP3.
- urlencode permet d'encoder une chaine de caractères afin qu'elle puisse faire partie d'une URL.
- xpath permet de faire des requêtes XPath dans un document XML.
- xmldom permet de construire une représentation DOM d'un contenu XML. Cette représentation DOM pourra ensuite être utilisée par XPath pour effectuer nos requêtes.

Liste des genres (/genres/)

- A l'aide d'un appel à app.get(), créez une route qui répondra aux requêtes GET à l'adresse /genres/.
- Dans le callback de app.get(), récupérez le top des genres de Lastfm via un appel à fetch() à l'adresse suivante : http://ws.audioscrobbler.com/2.0/?method=tag.getTopTags&api_key=2c08f218f45c6f367a0f4d2b350bbffc.
  - Votre premier `then` permettra de récupérer le texte de la requête (via `res.text()`).
  - Votre second `then` devra utiliser la méthode `parseFromString()` du module `xmldom` afin d'obtenir une représentation DOM des données XML. Un appel à la méthode `select()` du module `xpath` devrait vous permettre de récupérer tous les ids de genres (appelés tags dans l'API Lastfm). A vous de construire la requête XPath permettant cette récupération.
- Affichez dans la console la liste des genres afin de vérifier que ce que vous avez fait est correct.
- Pour chaque genre récupéré, utilisez la méthode `map()` afin d'effectuer une nouvelle requête qui permettra de récupérer la description du genre. Celle-ci est récupérable à une adresse de la forme http://ws.audioscrobbler.com/2.0/?method=tag.getinfo&tag=ID_DU_GENRE&api_key=2c08f218f45c6f367a0f4d2b350bbffc.
- Il faut stocker le résultat de votre appel à `map()` dans une variable que vous nommerez `promises`. Cette variable contiendra un tableau de promesses sur lequel nous allons pouvoir nous synchroniser. En effet, la réalisation des promesses se faisant de manière asynchrone

il nous faut nous assurer qu'elles auront toutes été réalisées avant de renvoyer la réponse à notre requête.
- Pour cela vous devrez utiliser la méthode Promise.all() qui permet d'attendre la réalisation d'un ensemble de promesses et d'appeler un callback sur un tableau contenant le résultat de toutes ces promesses (il est également possible de se passer de callback en utilisant `await` sur `Promise.all()`). C'est dans ce callback que vous pourrez renvoyer votre résultat sous forme de chaine de caractères à l'aide de la fonction `JSON.stringify()`.

Bravo, si tout s'est bien passé vous avez réussi à remplir la liste des genres ainsi que leur description. Pour la récupération des informations concernant les artistes liés à ce genre, rendez-vous dans la partie Bonus !

Bonus

- Complétez l'API de l'exercice 3 afin de récupérer les informations concernant les artistes correspondant à un genre (l'url à requête sur l'API lastfm sera de la forme http://ws.audioscrobbler.com/2.0/?method=tag.gettopartists&tag=ID_DU_GENRE&api_key=2c08f218f45c6f367a0f4d2b350bbffc)

Remarque : pour des raisons de copyright, lastfm ne renvoie plus les images des artistes lors des appels à leur API. Ne soyez donc pas surpris de voir des images d'étoiles à la place des images d'artistes.

- Complétez l'API des exercices 2 et 3 afin de renvoyer les informations concernant les albums d'un artiste (pour la partie Lastfm nous vous laissons trouver les urls à utiliser, sachant que l'`api_key` sera toujours la même). Il vous faudra aussi certainement modifier un peu le code de la partie client de l'exercice 3, car Lastfm renvoie beaucoup d'albums pour chaque artiste et notre mise en page n'est pas adaptée…

A rendre…

Pour le rendu de ce TP vous créerez une archive (type zip) contenant l'ensemble des fichiers créés et modifiés dans chaque exercice (1.1, 1.2, 2 et 3), à l'exception du dossier `node_modules`.

L'évaluation du TP sera faite sur deux critères :

- Quantité de fonctionnalités mises en place et respect des consignes.
- Pas d'erreur au linter Javascript (modulation de la note jusqu'à -50%).