POLYTECH MARSEILLE Aix*Marseille Université **Informatique – 4A**

Applications web et mobiles

TPs n°10 et 11 : Backend en Node/Express 1ère version

Étape 51 – Mise à jour de l'application Angular

Vous allez utiliser un nouveau *backend* en Node/Express. Dans le *frontend*, les URL du *backend* vont donc être modifiées. Par exemple, en PHP, le script assurant l'authentification des utilisateurs peut être :

`http://127.0.0.1/forum-angular-php/backend/checkLogin.php`

alors que celui en Node/Express pourra être :

`http://127.0.0.1:3000/checkLogin`

De la manière dont vous avez créé votre *frontend Angular*, seules 2 modifications sont nécessaires pour que celui-ci utilise votre *backend* en Node/Express. Elles sont situées toutes les deux dans le code de la classe `MessageService` :

1. remplacez l'URL de la racine de votre *backend* PHP par `http://127.0.0.1:3000`.
2. supprimez l'ajout de l'extension `.php` aux URLs. De plus, traditionnellement, dans les transferts d'informations vers Node/Express, on n'utilise pas les `FormData`, on transmet directement des objets Javascript. Donc, mettez entre commentaires le code que vous aviez écrit pour remplir le `FormData` et passez directement en argument de `this.http.post` les données passées en argument à votre méthode `sendMessage`.

Voilà, la mise à jour de votre `frontend` est terminée. Vous n'aurez plus aucune modification à y apporter. Notez bien que cela résulte du fait que l'on a utilisé un service de messagerie, qui abstrait les envois de messages, plutôt que d'utiliser directement dans toutes les classes des instances de `HttpClient`.

Étape 52 – Version de démonstration

Récupérez sur le site web AMeTICE du module ou sur https://pageperso.lis-lab.fr/christophe.gonzales/teaching/mobile une archive de démonstration d'un backend en node : `node-backend-demo-FR.tgz` ou `node-backend-demo-FR.zip`. Désarchivez le fichier. Dans le répertoire ainsi créé (`node-backend-demo-FR`) se trouve un fichier `coursesDemo.sql` qui contient une table appelée `coursesDemo`. Dans phpMyAdmin, importez ce fichier dans votre base de données.

Dans le répertoire `node-backend-demo-FR`, faites un `npm install`. Cela installera les packages suivants (cf. le fichier `package.json`) : `express`, `cors`, `body-parser` et `mysql2`.

Mettez à jour le fichier `config.js` afin que les informations concernant votre base de données soient correctes. Vous pouvez maintenant tester ce backend de démonstration : déplacez-vous dans le répertoire `node-backend-demo-FR` et tapez la commande :

```
node app.js
```

Vous verrez apparaître un message vous indiquant que votre serveur web écoute ses client sur le port 3000.

`postman` va vous permettre d'envoyer des requêtes à votre *backend* Node/Express. Pour cela, à gauche du bouton `send`, choisissez la méthode de transfert des données : à l'instar de votre application *Angular* actuelle, vous allez les transmettre via la méthode `POST` (et non `GET`). Indiquez également l'URL à laquelle vous souhaitez transmettre votre requête : votre serveur écoute `http://127.0.0.1:3000`. Dans ce serveur, il y a deux routes : `getCours1` et `getCours2`. Pour accéder à la première, il suffit donc de préciser l'URL : `http://127.0.0.1:3000/getCours1`

En cliquant sur le bouton `send`, vous devriez voir apparaître en bas de la fenêtre la réponse du serveur, en l'occurrence un objet JavaScript vous précisant qu'il y a une erreur. Cette erreur est due au fait que `getCours1` attend qu'on lui envoie comme *data* un champ `maxId` (autrement dit, si votre application Angular actuelle avait contacté `getCours1`, elle aurait passé à `sendMessage` un objet Javascript/TypeScript contenant un champ `maxId`). Pour réaliser ce transfert de données avec Node via `postman`, en dessous de l'URL que vous aviez indiquée, cliquez sur l'onglet `body`. Vous verrez alors apparaître les différentes manières d'envoyer des informations par `POST` : le `FormData` que vous aviez utilisé jusqu'à maintenant correspondait à l'onglet `form-data`. Ici, vous allez plutôt utiliser l'onglet `raw` et spécifier dans le menu déroulant à droite que ce que vous transmettez n'est pas du « simple » texte mais un objet JSON (donc un objet Javascript). Dans la zone de texte en dessous, spécifiez cet objet :

```
{
  "maxId": 20
}
```

Appuyez à nouveau sur le bouton « Send » et vous verrez apparaître une liste de noms de cours, comme le montre la figure 1. C'est de cette manière que vous pourrez tester facilement la mise en place de votre backend en Node/Express.

Étape 53 – Observation des fichiers du backend de démonstration

Vous pouvez maintenant regarder comment sont organisés les fichiers du backend. Précisons ici qu'ils sont tous commentés afin que vous puissiez comprendre comment ils fonctionnent. Le point d'entrée de votre serveur est le fichier `app.js`. Ce qu'il faut retenir :

1. Grâce à Express, on crée un fichier par route, comme en PHP : le fichier `getCours1.js` correspond à la route `/getCours1`, *etc.*
2. Le `session_start()` de PHP et la variable `$_SESSION` sont remplacés par deux fonctions :
   — `auth.getSession()` qui renvoie le contenu du cookie de session si ce dernier existe, ou bien un objet contenant uniquement un `userId` égal à -1 sinon.
   — `auth.sendSessionCookie()` qui envoie le cookie de session au *frontend*.
3. Pour récupérer l'ID de l'utilisateur, on utilise `auth.getUserId()`, qui permet de renvoyer -1 la première fois que l'on contacte le `backend`.
4. Contrairement à PHP, pour transmettre des messages au `frontend`, il ne faut pas exécuter simplement `sendMessage()` ou `sendError()` mais plutôt faire un `return sendMessage()` ou

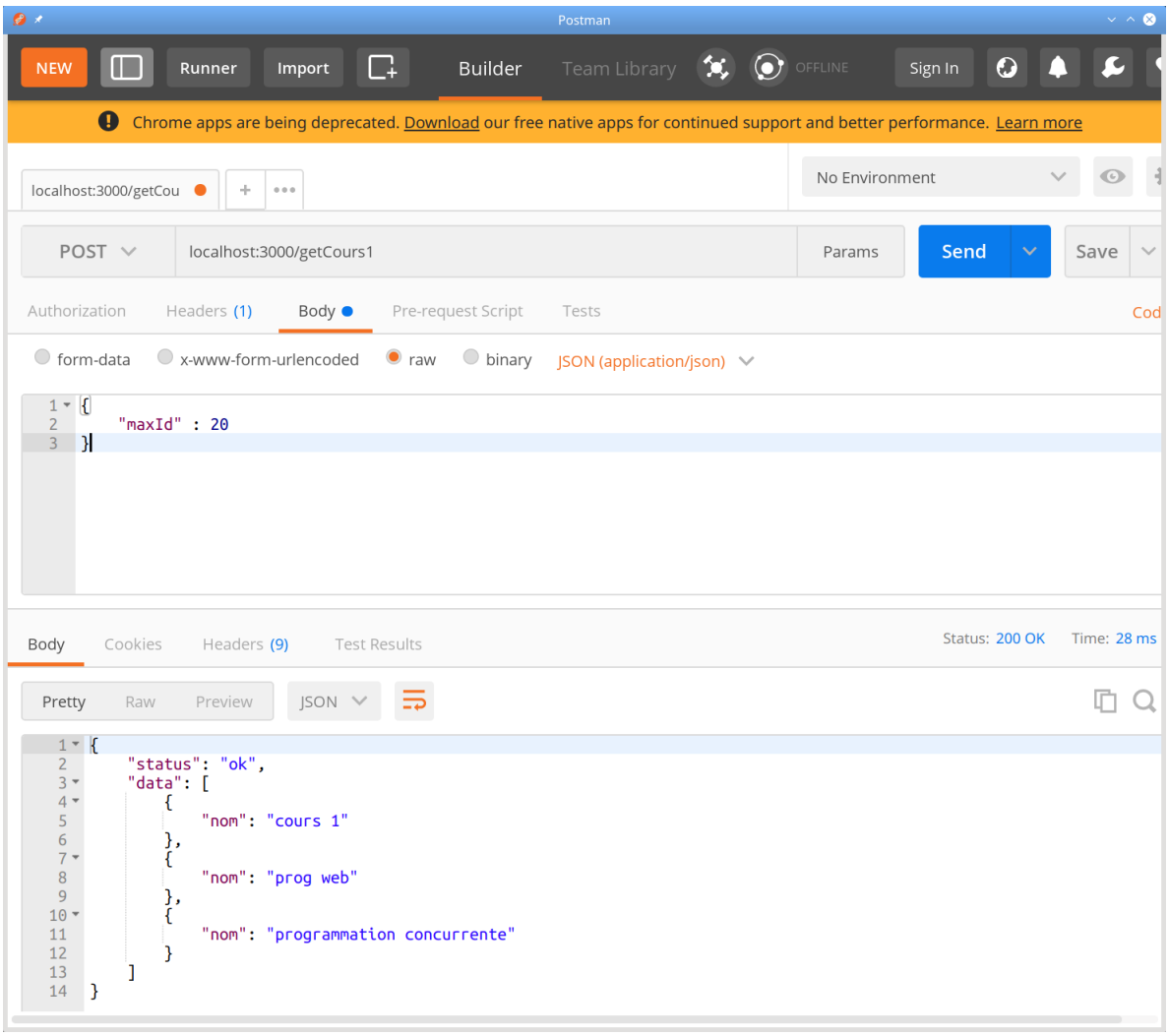

FIGURE 1 – Envoi de données par postman

`return sendError()` (cf. le fichier `getCours1.js`). En effet, en PHP, ces fonctions utilisaient un `die` pour terminer l'exécution des scripts PHP. Ici, si l'on fait la même chose, on arrêtera complètement le serveur. L'effet du return sera juste d'arrêter la fonction (comme `getCours`) qui correspond à la route demandée par l'utilisateur.

Lisez bien tous les fichiers de l'archive. Comprenez leur fonctionnement. Le seul fichier qui a peu d'intérêt pour l'instant est le fichier `auth.js` car il est pratiquement vide mais ses fonctions `getSession`, `sendSessionCookie` et `getUserId` vous seront utiles par la suite.

Étape 54 – Première version du backend en Node

En vous inspirant du backend de démonstration, écrivez votre propre backend Node/Express (pour l'instant, avec l'authentification « bidon » du fichier `auth.js` de démonstration) et testez-le avec `postman`. Autrement dit, outre les fichiers `app.js`, `auth.js`, `config.js`, `message.js`, `mysqlConnect.js` et `mysqlQueries.js`, il faut que vous réécriviez en Node l'équivalent des fichiers de votre backend PHP, à savoir `getCourses.js`, `getTopics.js`, `getPosts.js`, `saveNewPost.js`, `saveNewTopic.js`, *etc.* Écrivez ces fichiers un par un et testez chacun d'entre eux avec `postman` avant de passer au suivant. Dans l'archive de démonstration, l'utilisateur est identifié par un attribut nommé `userId`, qui est égal à 1

(cf. la fonction `getSession` du fichier `auth.js`). Si, dans vos sessions de PhP, vous avez utilisé un autre numéro, modifiez `userId` en conséquence (l'objectif, ici, est vraiment de reproduire ce que vous aviez écrit en PHP). De même, si la valeur de cet attribut est différent de 1 dans votre code PHP/base SQL, modifiez le code javascript en conséquence.

Optionnel : pour ceux/celles qui souhaitent en savoir plus

Étape 55 – Quelques références utiles

- En ce qui concerne les requêtes MySQL :
  https://www.npmjs.com/package/mysql2
  https://github.com/mysqljs/mysql#getting-the-id-of-an-inserted-row
- Si vous voulez comprendre l'idée sous-jacente à l'utilisation des await et des Promises, les deux URL suivantes pourront vous être utiles :
  https://codeburst.io/node-js-mysql-and-async-await-6fb25b01b628
  https://codeburst.io/node-js-mysql-and-promises-4c3be599909b
- En ce qui concerne le CORS :
  https://medium.com/@alexishevia/using-cors-in-express-cac7e29b005b
  https://expressjs.com/en/resources/middleware/cors.html