INFORMATIQUE
Sciences
Université Paris Cité

Programmation Web
TP n° 5a : Services Web

Dans la première partie de ce TP, nous allons réaliser des services Web avec Express qui communiquent au format JSON.

Services Web avec JSON

Un *service Web* est une fonctionnalité proposée par un serveur et utilisable par d'autres programmes via le protocole HTTP. Le programme client transfère, au serveur, via une requête HTTP, les paramètres d'*input* et le serveur lui renvoie sa réponse.

Le format *JSON* (JavaScript Object Notation) peut être utilisé pour transmettre ces paramètres [1]. Ce format est très proche de la déclaration d'objets littéraux en JavaScript : un objet javascript définie par `{name: "nom", age: 24}` sera encodé en JSON par la chaîne de caractères `'{"name": "nom", "age": 24}'` (seuls les guillemets diffèrent). Un tableau `[42, "abc"]` sera encodé par la chaîne `'[42, "abc"]'`. Les objets Javascript sont donc très facilement sérialisés (c.-à-d. représentés de manière portable au format JSON) pour pouvoir être transférés et reconstruits par le programme qui les reçoit.

JavaScript dispose des fonctions `JSON.stringify(`*Object*`)` pour transformer un objet en sa représentation JSON, et `JSON.parse(`*string*`)` pour transformer une chaîne JSON en un objet JavaScript. Cependant la plupart des fonctions qui opèrent sur le format JSON effectuent une conversion automatique ; il sera donc peu fréquent de devoir utiliser explicitement ces fonctions.

Exercice 1 : Un service web de dictionnaire

Nous allons implémenter un petit service Web permettant de gérer le contenu d'un dictionnaire. Le service devra permettre aux clients :

- d'ajouter des mots au dictionnaire ;
- de rechercher les mots commençant par un préfixe donné ;
- de récupérer l'ensemble des mots du dictionnaire.

1. Dans un fichier `Dico.js` écrivez un constructeur `function Dico(){...}`. Permettant d'initialiser un dictionnaire créé par une instruction de la forme `let dico = new Dico()`. On rappelle que dans un constructeur, l'initialisation d'un objet créé par `new` se fait par des instructions de la forme `this.p = ...` , `this.m = function(...) {...}`.

Les mots d'un dictionnaire seront simplement stockés dans une propriété de type tableau de chaînes. On souhaite en outre disposer pour tout dictionnaire des méthodes suivantes :

- `search(word)` renvoyant un booléen indiquant si le dictionnaire contient ou non `word`,
- `insert(word)` insérant un mot dans le dictionnaire s'il ne s'y trouve pas déjà,
- `words()` qui renvoie le contenu courant du dictionnaire trié,
- `prefixSearch(query)` qui renvoie un tableau contenant tous les mots du dictionnaire dont `query` est un préfixe.

Pour les écrire vous pourrez utiliser les méthodes de `Array.prototype`, invocables sur tout tableau, celles de `String.prototype`, invocables sur toute chaîne, et suivre ces indications :

- `search(word)` : vous pouvez utiliser une simple boucle, ou encore mieux invoquer sur le tableau du dictionnaire la méthode `findIndex` avec un prédicat approprié.
- `insert(word)` : combinez `search`, `push`, et `sort` (pour conserver le dictionnaire trié).

1. XML serait une alternative

– `prefixSearch(query)` : vous pouvez utiliser une boucle, ou mieux untiliser `filter` combiné avec `String.prototype.search` qui retourne l'indice de la première correspondance cherchée.

Terminez votre fichier `Dico.js` par l'instruction : `module.exports = new Dico();`, elle vous permettra, dans un second fichier, d'importer l'objet construit par ce `new` via l'instruction : `const dico = require('./Dico');`

2. Écrivez à présent un serveur Express `dictionnaire.js` :

– Commencez par récupérer l'objet `dico` créé dans `Dico.js`, et remplissez manuellement ce dictionnaire dans votre code par quelques mots, dont certains auront un prefixe commun.

– Ajoutez le paramétrage d'express suivant :

```
const express = require("express");
const server = new express();
server.use(express.json());
server.use(express.urlencoded({extended:true}));
```

On rappelle qu'alors :

(i) Le serveur pourra recevoir et émettre des données JSON.

(ii) Dans le traitement `(req,res)` des routes en `GET` d'URL à paramètres (c.à.d qui terminent sous la forme `?word=abc`) : `req.query` est un objet de la forme `{word='abc'}`, dont les noms de propriétés sont les noms des paramètres, et les valeurs de ces propriétés les valeurs des paramètres sous forme de chaînes.

(iii) Dans les routes en `POST` dont le corps contient une chaîne JSON (`'{"word" : "abc"}'`), `req.body` sera une référence vers l'objet résultant du parsing de cette chaîne (`{word='abc'}`).

(iv) La réponse du serveur doit être émise via `res.json(`*réponse*`)`

Voici le comportement attendu du serveur :

a. Une requête en `GET` sur `/dictionary` renvoie, au format JSON, l'ensemble des mots du dictionnaire.

b. Une requête en `GET` sur `/dictionary/search` avec un paramètre `word` renvoie, au format JSON, les mots du dictionnaire dont la valeur de `word` est un préfixe.

c. Une requête en `POST` sur `/dictionary`, dont le corps contient une chaîne JSON de la forme `'{"word":` *chaîne*`}'` insère *chaîne* dans le dictionnaire si elle ne s'y trouve pas déjà. La réponse du serveur sera un simple message de réussite ou d'erreur. [2]

Pour tester votre serveur il est très facile de simuler l'usage du service à l'aide d'un navigateur (`http://localhost:8080/dictionary/search?word=ab`) sur les requêtes en `GET`.

Pour tester le `POST`, mais aussi le `GET`, on peut simuler l'usage du service par l'envoi explicite de requêtes depuis le terminal à l'aide du la commande `curl` :

– `curl -X POST -H 'Content-Type:application/json' -d '{"word": "abc"}' 127.0.0.1:8080/dictionary` (attention, la commande doit s'écrire sur une seule ligne)

– `curl -X GET 127.0.0.1:8080/dictionary/search?word=abc`

2. vous pouvez utiliser la constante `undefined`