Météo : le retour

Benoit Favre, Jérémie Auguste, Raquel Ureña

généré le 4 février 2020

1 Introduction

L'objectif de cette séance est de créer un serveur web qui aille chercher des informations sur un autre site en exploitant une API REST, les formate en utilisant des templates et renvoie la page web générée à l'utilisateur.

Il est recommandé pour tous les codes javascript, d'utiliser le mode stricte (ajouter `"use strict"` seul sur une ligne en haut de tous les scripts).

2 l'API de weatherstack.com

Le site weatherstack.com offre une API REST pour accéder à des informations sur le temps qu'il fait dans une ville dans le monde. Le compte gratuit permet d'accéder à ces données avec des limitations. Notez que ce TP peut être fait avec une autre API sans trop de problèmes.

1. Créez un compte sur le site et récupérez la clé d'accès à l'API (`https://weatherstack.com/signup/free`). La clé est ensuite visible dans `https://weatherstack.com/dashboard` une fois connecté.
2. Dans un terminal, utilisez `curl` pour faire quelques requêtes. Lisez la documentation pour voir ce que vous pouvez faire. Par exemple, l'url `http://api.weatherstack.com/current?access_key=API_KEY&query=Marseille`, dans laquelle vous avez préalablement remplacé `API_KEY` par votre clé, permet de récupérer la météo sur Marseille au format JSON [1].

1. Le format JSON est un sous-ensemble de javascript. Il s'agit d'une combinaison de nombres, chaînes de caractères, booléens, tableaux et dictionnaires. Vous pouvez obtenir une version formatée du JSON avec `curl -s URL|python -m json.tool`

3 Charger une URL depuis node.js

En node.js, le module `request` permet d'effectuer une requête HTTP sur un serveur distant. On peut l'installer avec `npm install request` en ligne de commande dans le répertoire du serveur. Ceci créé un répertoire `node_modules` qui contient tous les modules installés de cette manière. Il faut noter que `npm` s'occupe tout seul d'installer les dépendances de `request`, ces autres modules étant aussi rangés dans le même répertoire. Une fois installé, un module peut être utilisé dans un script `node` avec l'instruction `let module = require('nom du module');`.

Il se trouve que `request` est un module simple. Il n'est constitué que de la fonction `request(url, callback)` à qui on passe l'url que l'on veut charger et une fonction qui sera exécutée une fois que les données ont été chargées, ou lorsqu'une erreur arrive. Comme expliqué dans la documentation [2] du module, ce callback prend trois arguments : l'erreur s'il y en a eu une, un objet qui représente la réponse du serveur (avec par exemple les entêtes et le code HTTP) et le body renvoyé par le serveur. C'est ce dernier qui nous intéresse.

Voilà un exemple de script pour demander au site weatherstack le temps qu'il fait à Marseille. Le serveur renvoie dans le body un texte représentant les données au format JSON. On peut les convertir en objet javascript avec `JSON.parse(texte)`.

```
let request = require('request');
let key = '...';
let city = 'Marseille';
let url = `http://api.weatherstack.com/current?access_key=${key}&query=${city}`;

console.log(url)
request(url, (err, res, body) => {
  if(!err) {
    let data = JSON.parse(body);
    console.log(data);
  }
})
```

1. Après avoir testé ce script, modifiez-le pour pouvoir passer le nom de la ville en argument au programme, en tapant dans le terminal quelque chose comme `node get-weather.js Marseille`. Indice : utilisez `process.argv`.
2. Modifiez le script pour qu'il affiche à partir des données renvoyées par weatherstack :
   — Le nom de la ville
   — La température
   — La vitesse du vent
   — La description du temps qu'il fait (`weather_descriptions`)

4 Un serveur web

Voici un court exemple de serveur web qui écoute sur le port 3000 sur localhost. Quelle que soit l'url demandée, il renvoie la même page HTML.

```
let url_parser = require('url')
let http = require('http')

let server = http.createServer((req, res) => {
  let url = url_parser.parse(req.url, true);
```

2. https://www.npmjs.com/package/request

```
  console.log(url);
  res.writeHead(200, {'content-type': 'text/html'});
  res.end('<h1> Température à Marseille </h1>');
  }
});

server.listen(3000);
```

Modifiez ce serveur pour qu'il aille chercher le temps qu'il fait dans une ville de votre choix, et génère une page HTML simple contenant le nom de la ville, la température, la vitesse du vent, la description textuelle du temps qu'il fait et l'image représentant le temps qu'il fait (`weather_icons`).

5 Rendu avec des templates

Il existe de nombreux moteurs de templates. Ces derniers permettent de séparer le code de l'application et le HTML, en rendant certains parties de ce dernier paramétrables. Le principe est de mettre toutes les données que l'on veut inclure dans le HTML dans un objet javascript, puis dans la page HTML d'indiquer où chacun des champs de l'objet doivent être remplacés.

Le moteur de template le plus simple pourrait ressembler à cela :

```
let template = '<p> Température : {{temperature}} degrés</p>';
let temperature = 25;
let html = template.replace('{{temperature}}', temperature);
```

Certains moteurs de templates sont très puissants et permettent d'utiliser des conditions et des boucles dans le HTML. Pour ce TP, nous allons en utiliser un très simple qui s'appelle Mustache (`https://github.com/janl/mustache.js/`).

On peut installer Mustache avec `npm install mustache`. Ensuite, un script qui l'utilise ressemblerait à :

```
let mustache = require('mustatche');
let data = {x: 10, y: 3};
let template = "Les coordonnées sont {{x}} et {{y}}.";

console.log(mustache.render(template, data));
```

1. Lisez la documentation de Mustache pour comprendre comment il fonctionne. Deux façon d'employer les templates vont nous intéresser : les valeurs (`{{temperature}}`) et les tableaux (`{{#jours}} {{*}} {{/jours}}`). L'idée est de passer l'objet json retourné par weatherstack au moteur de template pour qu'il génère la page à partir des valeurs s'y trouvant.
2. Écrivez une page html pour présenter les données de weatherstack (par exemple le nom de la ville, le temps qu'il fait, la température et le vent). Pour l'instant, utilisez des exemples de valeurs.
3. Remplacez les exemples de valeurs par des templates. Pour tester, écrivez un script nodejs qui charge le template depuis un fichier, instancie un objet javascript avec des données et appelle mustache pour générer et afficher le HTML correspondant.
4. Intégrez ce code dans votre serveur.

Notez que l'on peut charger un fichier dans une chaîne de caractères avec `fs.loadFileSync('nom du fichier').toString()`. Cette fonctionnalité utilise le module `fs` qui fait partie de la librairie standard de node.

6 Sélection de la ville

Maintenant que le serveur sait aller chercher les données sur un serveur tiers, en faire une page HTML à partir d'un template, et les renvoyer au client, il ne reste plus qu'à laisser le client choisir la ville dont il veut la météo. Deux méthodes seront mises en place.

6.1 Dans l'URL

Dans la première méthode, on va considérer que l'utilisateur doit écrire le nom de la ville dans l'URL qu'il demande au serveur. L'URL sera de la forme `http://127.0.0.1:3000/Nom-de-la-ville`. Pour récupérer le nom de ville demandé par l'utilisateur, il suffit d'utiliser le champ `pathname` de l'objet url renvoyé par `url_parser` à partir de son analyse de `req.url`. Ce champ sera de la forme `/Nom-de-la-ville`, donc il faudra enlever le slash.

Implémentez et testez cette manière de choisir la ville avec `curl` et dans le navigateur.

6.2 Avec un formulaire

Il faut se l'avouer, devoir entrer la ville dans la barre d'URL du navigateur n'est pas très intuitif pour un utilisateur. Vous allez donc ajouter un formulaire à la page HTML pour que l'utilisateur puisse saisir le nom de la ville dans un champ prévu à cet effet et cliquer sur un bouton pour valider sa sélection. Voici à quoi pourrait ressembler un tel formulaire :

```
<form action="/" method="GET">
  <input type="text" name="city" placeholer="Ville">
  <input type="submit" value="Valider">
</form>
```

Il faut noter les éléments suivants :

- Le formulaire est une balise `<form>` contenant deux balises `<input>`.
- La première est de type `text` et a pour nom `city`. L'attribut `placeholder` permet juste d'afficher un texte dans le champ pour aider l'utilisateur. Ce champ est celui dans lequel on va saisir la ville.
- La seconde est de type `submit`. Ce champ est le bouton sur lequel l'utilisateur doit cliquer pour soumettre le formulaire.
- Pour le navigateur, soumettre le formulaire veut dire appeler l'URL spécifiée dans le champ `action` de la balise `<form>` avec en paramètres la valeur des champs ayant un attribut `name`, et afficher la page renvoyée par le serveur.

Par exemple, si on entre `Marseille` comme nom de ville puis on clique sur `Valider`, le navigateur va charger l'URL : `http://127.0.0.1:3000/?city=Marseille`. Côté serveur nodejs, après traitement de l'URL par `url_parser`, on reçoit la valeur du paramètre `city` dans `url.query.city`.

1. Modifiez le template de la page HTML pour y ajouter le formulaire.
2. Prenez en compte la ville choisie dans le formulaire à travers `url.query.city`.

Bravo, vous avez conçu un site web simple qui va chercher des informations sur un serveur REST en fonction de ce que veut l'utilisateur et présente le résultat sous forme lisible par un humain. Ouf.

7 Pour aller plus loin

1. Redemander à weatherstack le temps qu'il fait à chaque requête n'est pas très économe (surtout qu'un compte gratuit est limité à 1000 requêtes par jour). Ajoutez un cache à votre serveur. Vous pouvez utiliser un dictionnaire qui contient pour une ville donnée la réponse de weatherstack et la date à laquelle la requête a été effectuée (utiliser `new Date()`). Lors d'une nouvelle requête

pour cette ville, si la date est moins récente que 10 minutes, retournez la valeur du cache, sinon mettez à jour cette valeur avant de la retourner.

2. Choisissiez une API sur `https://apilist.fun/` et développez un serveur web pour donner accès aux données proposées par cette API.