TD 8 : API

25 novembre 2015

Objectif: Ce TD a pour but d'introduire les APIs, avec plusieurs exemples, et voir comment les utiliser pour récupérer des données depuis des sites tiers.

La majorité des sites (communautaires, en général) à fort trafic possèdent leur propre **API** (**A**pplication **P**rogramming **I**nterface). Mais une API, qu'est-ce-que c'est ? C'est un moyen mis à la disposition de développeurs pour communiquer avec un servuce web (par exemple une base de données) d'un site et d'en récupérer des informations. Par exemple, l'API d'*Allociné* permet de récupérer toutes les informations de n'importe quel film, l'API de *VieDeMerde* permet de récupérer des VDM, etc.

Le format préféré pour communiquer avec ces APIs est JSON (voir TD précédent).

Pour des raisons de vie privée compréhensibles, il est possible d'utiliser certaines APIs (Facebook, Twitter, etc.) seulement si on possède un *token* (un jeton), et pour obtenir ceci on a besoin, normalement, d'un identifiant et d'un mot de passe.

1 Twitter API

Twitter est un réseau social, plus particulièrement un service de microblogging, qui permet aux utilisateurs de diffuser des messages courts (« tweets ») aux membres de son réseau. Le réseau d'un utilisateur est caractérisé par le nombre de personnes abonnés à ses publications (« followers ») et le nombre de personnes auxquelles l'utilisateur est lui-même abonné (« followings »).

Dans ce TD, nous allons récupérer les derniers tweets et créer un widget pour les afficher. Voici les identifiants d'un compte Twitter que vous pouvez utiliser tout au long de ce TD :

Login : imac2015_web
Mot de passe : valentin

Le profil de cet utilisateur est, par ailleurs, visible à l'adresse `https://twitter.com/imac2015_web`.

Comme bon nombre de réseaux sociaux, Twitter possède sa propre API. Cette dernière permet d'effectuer des actions de toutes sortes sur un compte twitter donné, comme par exemple :

- publier des messages,
- récupérer le nombre de followings / followers,

- récupérer les tweets postés.

Pour qu'un utilisateur puisse faire des requêtes, Twitter exige qu'il soit clairement identifié, dans le but de protéger la confidentialité des utilisateurs. Si vous ne vous identifiez pas auprès de twitter, vos requêtes vont retourner le message d'erreur suivant:

```
<error code="215">Bad Authentication data</error>
```

Twitter utilise le protocole *oAuth* pour l'identification. Ce protocole accepte 4 paramètres, dont les valeurs sont uniques et qui doivent rester secrètes : *consumer key*, *consumer secret*, *access token* et *access token secret*. Ces paramètres sont générés lors de la création d'une application sur le site des développeurs de Twitter.

Une fois authentifié, le développeur peut envoyer des requêtes PHP. L'utilisation de PHP au lieu de JavaScript (AJAX, par exemple) permet de garder les valeurs des paramètres secrètes [1].

Pour comprendre l'API Twitter sans écrire de code PHP, on va utiliser une console interactive. Elle est utile pour tester des requêtes et fournir un déboggage facile de votre code. Nous avons choisi la console **Apigee** afin d'accéder aux données de l'API Twitter.

▸ **Exercice 1:** Ouvrez la console Apigee (`https://dev.twitter.com/rest/tools/console`).

- Depuis le champs *Authentication*, choisissez l'option « *oAuth 1* ». Pour autoriser l'accès, veuillez vous identifier via votre compte Twitter (ou via le compte `imac2015_web`).
- Dans le champs « *Request URL* », gardez la methode « *GET* » et tapez : `https://api.twitter.com/1.1/statuses/user_timeline.json?count=`*`[le nombre tweets à récupérer]`*`&screen_name=`*`[le nom du compte]`*.
- Récupérez les 10 derniers tweets du compte `imac2015_web`
- Copiez le tableau des tweets renvoyé depuis la fenêtre *Reponse*, et stockez le dans un fichier `.json` dans un dossier `ajax/`.

Vous pouvez constater que le résultat obtenu est en fait un objet au format JSON. Nous allons le parser (parcourir) afin d'en récupérer les informations qui nous intéressent. Pour résumer, voici l'organisation du JSON pour un tweet :

```
{
  "text":"Le contenu du tweet",
  "created_at":"Date de creation"
  "user":{
      "L'auteur du tweet",
      "screen_name":"Son Nom",
      "profile_image_url":"Sa photo",
      "friends_count":"Nombre de followings",
  }
}
```

En utilisant ce qu'on a appris dans le TD précédent (`getJSON`, `each`, etc.) on est maintenant en mesure de parcourir les données et d'en retirer les information qui nous intéressent.

▸ **Exercice 2:** Réalisez un widget regroupant les 10 derniers tweets du compte `imac2015_web`. L'idée est de faire quelque chose similaire à l'image en Figure 1.

Vous pouvez aussi ajouter d'autres fonctionalités, par exemple vous pouvez :

- ajouter un timer pour rafraîchir automatiquement votre widget toutes les minutes (chaque minute, le site va voir si le fichier `.json` a été changé, et si oui, il va récharger les informations des tweets) ;

1. Mais comme on a dit plusieurs fois, vous allez voir ça au prochain semestre !

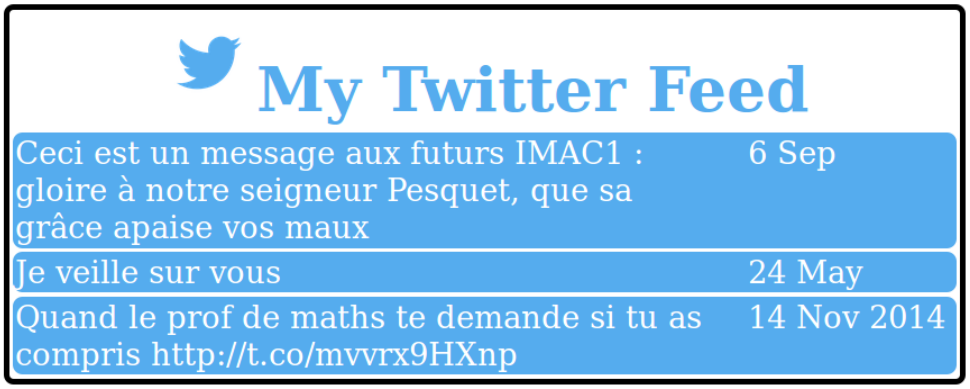

FIG. 1 – *Widget contenant des tweets.*

- afficher la date de façon relative (« Il y a 1 jour » à la place de « 24 novembre ») ;
- détecter les liens vers d'autres utilisateurs (commençant par "`@`") et rediriger vers leur compte ;
- détecter le liens vers les mot-dièses (ou *hashtags*) (commençant par"`#`") ;
- etc.

2 Papaoutai ?

L'API HTML Géolocalisation est utilisée pour obtenir la position géographique d'un utilisateur.

Avant toute chose, vous devez détecter si le navigateur de l'utilisateur permet la géolocalisation. En effet, comme toutes les nouvelles APIs, seuls les navigateurs récents (Firefox 3.5+, Safari 5+, Chrome 5+, IE9+) ont accès à ces données.

```
function trouveLieu() {
    if (navigator.geolocation) {
        navigator.geolocation.getCurrentPosition(montrePosition);
    } else {
        alert("Ce navigateur ne supporte pas la géolocalisation"); }
}

function montrePosition(position) {
  var latituteLongitude = position.coords.latitude + "," +
      position.coords.longitude;
  alert(latituteLongitude)
}
```

La méthode `getCurrentPosition()` est utilisée pour obtenir la position de l'utilisateur.

Si la géolocalisation est possible, une fonction (dans l'exemple `montrePosition`) va être exécutée avec paramètre l'objet contenant la position (dans l'exemple `position`).

Vous notez que, lors de l'appel de cette méthode, le navigateur va vous demander s'il peut accéder à vos données de géolocalisation. En effet, pour des raisons de vie privée compréhensibles, l'utilisateur peut bloquer l'utilisation de l'API s'il le souhaite.

Les propriétés « `position.coords.latitude` », « `position.coords.longitude` » et « `position.coords.accuracy` » sont toujours renvoyées. Les autres propriétés, par exemple « `position.coords.altitude` », sont retournées seulement si elles sont disponibles.

Maintenant que nous avons récupéré ces données, nous allons pouvoir les utiliser pour les afficher au sein d'une carte. On va le faire avec « `Staticmaps` » de Google, qui est simplement une demande (`url`) qui nous montre une image statique de Google Maps sur des coordonnées spécifiées. La syntaxe est la suivant:

`http://maps.googleapis.com/maps/api/staticmap?center=48.84,2.59&zoom=14&size=400x300&sensor=false`

Où êtes vous? Testez la fonction simplement en copiant le lien dans vôtre navigateur. Notez que « `center` » contient les cordonnées selon la syntaxe `center=`*`latitude`*`,`*`longitude`*. Que se passe-t-il quand vous changez le valeur de « `zoom` » ou « `size` »?

▸ **Exercice 3:** Utilisez la géolocalisation en combinaison avec les coordonnées fixes (que vous pouvez trouver sur internet, par exemple) pour créer trois plans :

- un avec votre emplacement actuelle,
- un avec la ville de naissance de votre mère et
- un avec la ville de naissance de votre père.

3 Drag & Drop

Le *drag and drop*, ou « glisser-déposer » en français, est l'un des principaux éléments d'une interface fonctionnelle. Il s'agit d'une manière de gérer une interface en permettant le déplacement de certains éléments vers d'autres conteneurs.

Ainsi, dans l'explorateur de fichiers d'un système d'exploitation quelconque, vous pouvez très bien faire glisser un fichier d'un dossier à un autre d'un simple déplacement de souris, ceci est possible grâce au concept du drag & drop.

Bien que cette fonctionnalité ait longtemps existé sur les sites Web grâce au JavaScript, avec HTML5 il est devenu possible de permettre un déplacement de texte, de fichier ou d'autres éléments depuis n'importe quelle application jusqu'à votre navigateur.

La première chose à faire est de rendre un élément déplaçable. Pour cela, on donnera la valeur « `true` » à l'attribut `draggable`.

Par exemple :

```
<h1 id="monTitre" draggable="true">Essayez de me déplacer !</h1>
```

Maintenant qu'on peut déplacer l'élément, on va utiliser les événements que l'API Drag & Drop nous fournit. Le premier événement est `ondragstart` qui, comme indique le nom, se déclenche lorsque l'élément ciblé commence à être déplacé.

Par exemple :

```
<h1 id="monTitre" draggable="true" ondragstart="glisse(event)">Essayez de me dé
    placer !</h1>
```

La fonction « `glisse` » spécifié les données qui doivent être glissées. Dans cette fonction on utilisera la méthode `dataTransfer.setData()` qui définit le type (en général `"text"` suffit même s'il s'agit d'images) et la valeur des données.

Par exemple, la fonction

```
function glisse(monEvenement) {
    monEvenement.dataTransfer.setData("text", monEvenement.target.id);
}
```

permet de déplacer l'objet même (dans l'exemple ci-dessus, le `h1` ayant pour id « `monTitre` »).

▸ **Exercice 4:** Modifiez le fichier `recherche.html` créé dans le TD1 afin de permettre le glissement de l'image de la madeleine. Quand on commence à glisser le gâteau, l'arrière plan deviendra bleu, tandis que quand on rélache la souris l'arrière plan deviendra rouge (pensez à utiliser l'événement `ondragend`).

Maintenant qu'on a appris comment glisser un élément, on voudrait aussi le déposer quelque part. Par defaut, un élément ne peut pas être déposé dans un autre élément. Pour permettre une exception à cette régle, on utilisera la méthode `preventDefault()` pour l'événement `ondragover`.

Par exemple, dans le `div` suivant

```
<div id="maBoite" ondragover="permetDepose(event)">Déposez ici svp !</div>
```

il est permis de déposer des éléments, pourvu qu'on définisse dans notre fichier `.js` la fonction :

```
function permetDepose(monEvenement) {
    monEvenement.preventDefault();
}
```

Une fois l'autorisation de déposer une donnée, on peut finalement déposer l'élément en utilisant l'événement `ondrop()`, comme dans l'exemple suivant :

```
<div id="maBoite" ondragover="permetDepose(event)" ondrop="depose(event)">Déposez
    ici svp !</div>
```

où la fonction `depose()` nous permet d'ajouter nos données (stockées dans `dataTransfer`) grâce à la méthode `dataTransfer.getData()`.

Par exemple :

```
function depose(monEvenement) {
    monEvenement.preventDefault();
    var idDeLaSource = monEvenement.dataTransfer.getData("text");
    monEvenement.target.appendChild(document.getElementById(idDeLaSource));
}
```

▸ **Exercice 5:** Créez un élément `div` ayant comme arrière plan une image d'une tasse de thé (cherchez sur Internet l'image que vous préférez).

En utilisant les exemples ci-dessus, modifiez le fichier `recherche.html` pour permettre de tremper votre madeleine dans le thé.