Les APIs

Table des matières

Qu'est-ce qu'une API ?

Une API, ou **Application Programming Interface** est une interface de programmation.

Les interfaces de programmation sont partout autour de nous. Elles sont utilisées pour accéder aux données qui vont permettre à plusieurs applications, sites ou services de fonctionner ensemble. C'est notamment grâce à elles que des applications nous permettent de créer un compte depuis un autre compte Google ou Facebook. On va alors faire appel à l'API Google pour récupérer les informations de l'utilisateur sur la base de son email gmail par exemple.

À quoi ça sert ?

Comme expliqué plus haut, une API a pour vocation de permettre la communication entre plusieurs ressources pour **échanger des données**. Il est important de comprendre que lorsque vous créerez votre site, vous n'aurez pas seulement besoin de **vos** données. Vous pouvez très bien avoir besoin de données **tierces**, comme par exemple une liste des arrêts de bus à proximité de votre agence, que peut vous fournir l'**API** Google Maps, une analyse poussée du trafic sur votre site, que peut vous fournir l'**API** Google Analytics…

Une API est donc un outil permettant de communiquer avec des services développés par des tiers (ou vous-même !) depuis votre site. On parle alors de **consommation d'API**.

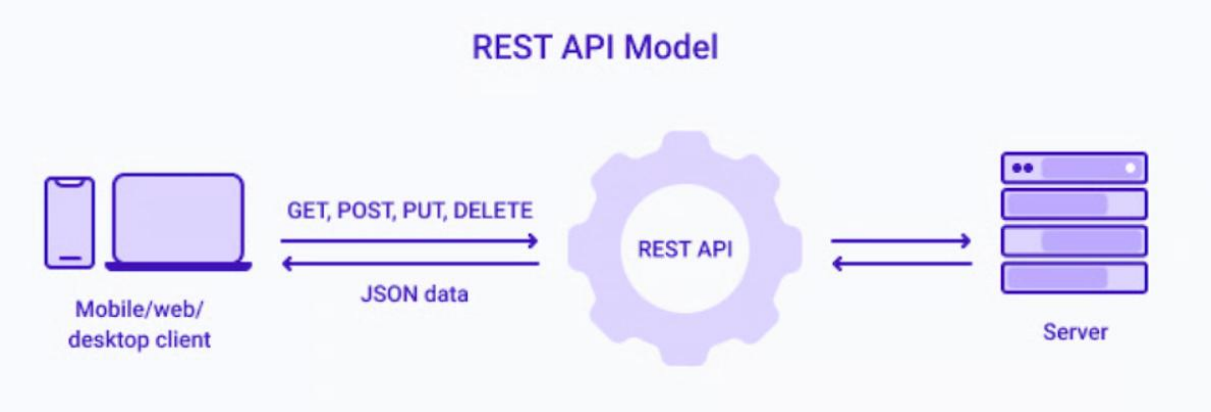
Comment ça marche ?

Dans le Web, nous utilisons un type bien spécifique d'API, les API **REST**, ou **RESTful** (Representational State Transfer). REST est simplement un standard de développement des API câblé pour fonctionner avec les **protocoles HTTP** que nous connaissons bien en langages Web.

À noter qu'il existe également un cousin des API REST, les API SOAP, qui sont un modèle de conception d'API plus strict et moins polyvalent car ils ne fonctionnent qu'en échangeant des flux XML. Nous ne nous attarderons pas vraiment dessus dans ce cours.

Les méthodes HTTP qu'une API REST utilisent sont les mêmes méthodes HTTP que l'on a vu au tout début de l'année. Que le temps passe vite…

- POST
- GET
- PUT
- DELETE



Vous reconnaissez l'architecture Client / Serveur ? C'est normal, c'est exactement comme ça que ça marche.

Sauf que cette fois ce n'est pas le navigateur qui va interpréter notre URL en une requête vers le serveur, c'est à nous de faire le travail.

Intégration PHP :

En PHP, on peut utiliser une série de fonction dédiées à la fonction Shell qui s'appelle cURL, qui nous permet d'accéder à des URLs données. cURL veut littéralement dire Client URL, on parle alors de requête cURL.

Ici on définit l'URL que l'on souhaite appeler dans la variable **$url**, puis on **initie** la requête cURL avec **curl_init**. On définit ensuite une série d'options avec la fonction **curl_setopt**, puis on demande au **curl** de s'exécuter avec la fonction **curl_exec()**. On récupère alors le résultat dans une variable **$resultat**.

```php
$curl = curl_init($url);
curl_setopt($curl, CURLOPT_URL, $url);
curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);

//Vérif du certificat de l'appelant : On ne met ces lignes que pendant le dev !
curl_setopt($curl, CURLOPT_SSL_VERIFYHOST, false);
curl_setopt($curl, CURLOPT_SSL_VERIFYPEER, false);

$resultat = curl_exec($curl);
unset($curl);
```

Intégration Javascript :

En JavaScript la syntaxe est beaucoup plus simple, à l'exception près qu'il faut spécifier un type de script bien précis dans la balise **<script>** si on utilise le mot clé **await**. Ce ne sera pas tout le temps le cas, mais je simplifie l'exemple ☺. On utilisera la fonction **fetch** pour gérer les appels.

```javascript
    <script type="module">
        const response = await fetch(url);
        const text = await response.text();
    </script>
```

Si on n'appelle pas l'API dès le chargement mais qu'au contraire on déclenche l'appel uniquement après une action de l'utilisateur, on a deux choix :

- Soit on utilise cette syntaxe de javascript avec **fetch** suivie plus loin de **response.json()** :

```
fetch(monUrl).then(response => {
    if (!response.ok) {
         throw new Error('Network response was not ok');
    }
    return response.json();
}).then(data => {
    console.log(data);
}).catch(error => {
    console.error('Error:', error);
});
```

- Soit on triche un peu et on va chercher avec AJAX un fichier PHP qui fera l'appel en PHP. Astucieux ;)

Exercices :

Allons voir l'API qui permet d'obtenir le découpage Administratif des régions et essayons un peu de la faire marcher ☺

https://geo.api.gouv.fr/decoupage-administratif

Indispensable : https://reqbin.com/curl

Générateur de syntaxe cURL depuis l'instruction Shell

Lexique :

- **Appel** : On appelle un appel le fait d'effectuer une requête à une API. Cela rejoint le vocabulaire propre aux requêtes HTTP.

- **Endpoint** : On appelle un **endpoint** l'URL de **destination** de l'API. Par exemple, dans l'API https://geo.api.gouv.fr, on peut accéder à l'endpoint **/regions** qui va nous renvoyer la liste des régions. Cela nous donne donc une URL comme ceci : https://geo.api.gouv.fr/regions
  On a généralement plusieurs **endpoints** définis dans la documentation d'une API, ici on peut également accéder à l'endpoint **/regions/{code_region}/departements** pour obtenir la liste des départements d'une région, par exemple.

- **Méthode** : Comme lors de n'importe quelle requête HTTP, un appel à une API utilise une méthode HTTP, parmi les suivantes : POST, GET, PUT, DELETE. On définira la méthode comme ceci avec **curl_setopt** et une série de constantes PHP :

```php
// POST
curl_setopt($curl, CURLOPT_POST, true);
// Ou PUT
curl_setopt($curl, CURLOPT_PUT, true);
// Ou DELETE
curl_setopt($curl, CURLOPT_DELETE, true);
// On ne définit rien pour GET, c'est son comportement par défaut.
```

En javascript, on définit simplement l'attribut **method** dans l'appel **fetch**, comme ceci :

```javascript
const response = await fetch('https://geo.api.gouv.fr/regions', {
    method: 'POST',
    // Ou
    method: 'PUT',
    // Ou
    method: 'DELETE',
    // Et on ne précise pas quand on est en GET car c'est son comportement
par défaut
});
```

- **Headers** : Les **headers** sont les données que l'on va envoyer en direction de l'API. En effet, il ne s'agit pas toujours forcément d'un simple appel à une URL, des fois il faut passer des informations comme par exemple le format de donnée envoyées. Le plus fréquent est de devoir passer un **token** dans les headers pour toutes les API qui nécessitent une **authentification**. On utilisera alors la méthode **curl_setopt** pour définir la valeur de **CURLOPT_HTTPHEADER**, comme ceci :

```php
$headers = array(
   "Content-Type: application/json",
   "Authorization: Bearer YOUR_ACCESS_TOKEN",
   "X-Custom-Header: Value",
);
```

```php
curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
```

En Javascript :

```javascript
const response = await fetch(url, {
    headers: {
        'Accept-Charset': 'utf-8',
        'Content-Type': 'application/x-www-form-urlencoded',
    },
});
```

- **Body** : Le « corps » de l'appel contient les données que l'on souhaite envoyer à l'API. On ne retrouve ce « corps », **facultatif**, que dans les requêtes POST, PUT et DELETE. On définira alors les données à envoyer comme ceci avec **curl_setopt** et la constante **CURLOPT_POSTFIELDS** **même si la méthode n'est pas POST** ! :

```php
$data = '{"productId": 1, "quantity": 10}';

curl_setopt($curl, CURLOPT_POSTFIELDS, $data);
```

En javascript, on précise l'attribut **body** dans l'appel avec **fetch**, comme ceci :

```javascript
const data = '{"username":"xyz","password":"xyz"}';

const response = await fetch(url, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
    },
    body: data,
});
```

- **Token / API Key** : un token (ou un jeton en français), est une chaîne de caractère plus ou moins longue, propre à une application, permettant de s'**identifier** pour utiliser une API. Un scénario courant est de devoir **générer** un token sur l'interface admin d'une API, pour un nom de domaine donné, afin d'authentifier notre application aux yeux de l'API. Exemples : Google Maps, Dropbox… On parle également parfois de **credentials**, c'est-à-dire les identifiants autorisant un accès.