**Intégrer dynamiquement un JSON à une application**

**Qu’est-ce que le JSON ?**

Le JavaScript Object Notation (JSON) est un format standard utilisé pour représenter des données structurées de façon semblable aux objets Javascript. Il est habituellement utilisé pour structurer et transmettre des données sur des sites web (par exemple, envoyer des données depuis un serveur vers un client afin de les afficher sur une page web ou vice versa).

JSON est un format de données semblable à la syntaxe des objets JavaScript. Il ne contient que des propriétés mais pas de méthodes. Malgré sa syntaxe très similaire à celle des objets littéraux JavaScript, JSON peut être utilisé indépendamment de ce langage et ainsi, de nombreux autres langages de programmation disposent de fonctionnalités permettant d'analyser la syntaxe du JSON et d'en générer.

**Structure du JSON**

Le JSON se présente sous la forme d'une chaîne de caractères, nombres, tableaux, booléens et tout autre objet littéral. Cela permet de hiérarchiser ses données. Utile lorsque l’on souhaite transmettre les données sur un réseau. Il a donc besoin d'être converti en un objet JavaScript natif lorsque l’on souhaite accéder aux données. Ce n'est pas vraiment un souci puisque le JavaScript fournit un objet global JSON disposant des méthodes pour assurer la conversion entre les deux.

**Comment manipuler le JSON sur une application ?**

Pour charger les données JSON, on utilise l'API XMLHttpRequest (qu'on appelle plus couramment **XHR**), ou plus récemment l’API Fetch. Il s'agit d'un objet JavaScript extrêmement utile qui nous permet de construire une requête afin d'interroger un serveur pour obtenir des ressources diverses (images, texte, JSON, ou n'importe quel extrait HTML) le tout en Javascript. En d'autres termes, cela permet de mettre à jour de petites sections de contenu sans avoir à recharger une page toute entière. Ceci conduit à des pages web plus réactives.

Exemple avec XHR :

On veut récupérer les données JSON et, après quelques manipulations du DOM, les afficher

**- Etape 1 :**

On charge l’url du fichier JSON que l’on veut récupérer dans une variable.

*var requestURL = 'https://mdn.github.io/learning-area/javascript/oojs/json/superheroes.json';*

**-Etape 2 :**

Afin de créer une requête, on a besoin d’instancier un nouvel objet *XMLHttpRequest* à partir de son constructeur en utilisant le mot clé *new.*

*var request = new XMLHttpRequest();*

**-Etape 3 :**

On ouvre une nouvelle requête grâce à la méthode *open()* et on ajoute la ligne suivante :

request.open('GET', requestURL);

Cette méthode prend au moins deux paramètres :

* La méthode HTTP à utiliser sur le réseau pour notre requête. La méthode *GET* est appropriée dans la mesure où nous voulons simplement récupérer quelques données.
* L'URL où adresser notre requête (il s'agit de l'url du fichier JSON précédemment évoquée.)

**-Etape 4 :**

On ajoute les lignes *request.responseType = ‘json’ ;* et *request.send() ;*

L‘attribution de la valeur *‘json’* à *responseType* signale au serveur que l’on attend une réponse au format JSON. Puis on envoie la requête à l’aide de la méthode *send()*

**-Etape 5 :**

Enfin, la réponse du serveur et son traitement. On ajoute les lignes suivantes :

*request.onload = function() {*

*var superHeroes = request.response;*

*populateHeader(superHeroes);*

*showHeroes(superHeroes);*

*}*

Ici, on stocke la réponse à notre requête (disponible au travers de la propriété [*response*](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/response)) dans la variable *superHeroes* ; cette variable contiendra désormais l'objet JavaScript basé sur le JSON ! On passe ensuite cet objet en paramètre à deux fonctions — la première remplira le <header> avec les données correspondantes tandis que la seconde créera une carte d'identité pour chaque héros de l'équipe et l'ajoutera dans la <section>.

On a encapsulé ce code dans un gestionnaire d'évènements qui s'exécutera quand l'évènement *load* sera déclenché sur l'objet *request* (voir *[onload](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequestEventTarget/onload))* — simplement parce que l'évènement *load* est déclenché quand une réponse a été renvoyée avec succès ; en procédant de la sorte, on sera certains que la propriété *request.response*sera disponible au moment où on essayera d'en faire quelque chose.

Exemple avec Fetch :

L'API Fetch est une solution moderne qui vient remplacer XHR, elle a été introduite récemment dans les navigateurs pour rendre les requêtes HTTP asynchrones plus simples en JavaScript, à la fois pour les développeurs et pour les autres APIs qui utilisent cette technologie.

Que se passe t-il dans le code Fetch ?

Tout d'abord, on invoque la méthode [fetch()](https://developer.mozilla.org/fr/docs/Web/API/WindowOrWorkerGlobalScope/fetch), en lui passant l'URL de la ressource que l’on veut récupérer. C'est la version moderne équivalente à [request.open()](https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest/open) de XHR, et n'a pas à appeler .send(), la requête est exécutée directemment.

Ensuite, la méthode [.then()](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise/then) est chaînée à la suite de fetch(), cette méthode fait partie des [Promesses](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Promise), une fonctionnalité JavaScript moderne qui permet d'effectuer des opérations asynchrones. fetch() retourne une promesse, qui est résolue lorsque la réponse est reçue du serveur, et on utilise .then() pour exécuter du code à ce moment là. C'est l'équivalent du gestionnaire d'événément onload dans la version XHR.

La fonction définie dans le .then() reçoit la réponse du serveur comme paramètre, une fois que la promesse retournée par fetch() est résolue. À l'intérieur de cette fonction, nous utilisons la méthode [text()](https://developer.mozilla.org/fr/docs/Web/API/Body/text) pour récupérer le contenu de la réponse en texte brut. C'est l'équivalent de request.responseType = 'text' dans la version XHR.

On verra que text() retourne également une promesse, nous y chaînons donc un nouveau .then(), à l'intérieur de quoi on déninit une fonction. Cette dernière récupère quant à elle le texte brut que la promesse précédente résout.

Enfin, dans le corps de la fonction, on fait la même chose que l’on a fait dans la version XHR — définir le contenu texte de l'élément [<pre>](https://developer.mozilla.org/fr/docs/Web/HTML/Element/pre) au texte récupéré.

Quelle API choisir ?

Cela dépend du projet sur lequel on travaille. XHR existe depuis longtemps maintenant et bénéficie d'un très bon support sur les différents navigateurs. Fetch et les promesses, en revanche, sont un ajout plus récent à la plateforme web, bien qu'ils soient pris en charge par la plupart des navigateurs, Internet Explorer et Safari font exception.

Si on veut un support des anciens navigateurs, alors XHR est probablement la solution préférable. En revanche, si on travaille sur un projet plus progressif, et que l’on n’est pas tant préoccupé par les anciens navigateurs, alors Fetch peut être un bon choix.

Les deux alternatives sont concevables, Fetch deviendra plus populaire au fur et à mesure que l'utilisation d'Internet Explorer diminue (IE n'est plus développé, en faveur du nouveau navigateur de Microsoft, Edge), mais on va avoir besoin de XHR pendant un moment encore.