Module 16 : Jquery/AJAX/JSON

1. **Interaction de JQuery avec Ajax**

* Tel que vu précédemment nous savons que chaque appel Ajax est une requête http et tel que nous savons également JavaScript est assez puissant et évolué pour envoyer des requetés http.
* JQuery nous permet de simplifier l’appel d’AJAX
* Jusqu’à maintenant nous avons utilisé JQuery sur nos pages du côté client : modification du DOM, ajout d'animations, modification du CSS, etc.
* Nous pouvons également étendre notre utilisation du JQuery pour une utilisation du côté serveur, ou la principale fonction du JQuery est d’intercepter les données et mettre à jour les pages Web

1. **XmlHttpRequest avec jQuery**

* **XmlHttpRequest** est plus facile à instancier avec jQuery qu’avec JavaScript. Cela grâce à l’utilisation de la fonction **jQuery.ajax()** ou **$.ajax()**
* La fonction **$.ajax()**permet d'instancier directement l'objet XmlHttpRequest.
* Le code d’instanciation est tel que ci-dessous :

$.(document).ready(function(){

/\*

\* Utilisons $.ajax pour créer une instance de XmlHttpRequest

\*/

$.ajax();

});

* Après l’instanciation de l’objet **XmlHttpRequest** ci-dessus nous pouvons commencer l’envoie d’une requête http
* Nous pouvons également coupler cette avec tous les évènements, effets ou autre.

**Exemple :**

* Nous prenons pour exemple un fil de commentaire qui se recharge en AJAX dès que l'on cliquerait sur un bouton « **plus de commentaires** ».
* Imaginons que ce bouton existe dans le DOM, et qu'il dispose d’un identifiant **#more\_com**.
* On va écouter l'évènement **click()** sur ce bouton et dès qu'il sera réalisé, on instanciera un objet **XmlHttpRequest** . tel que ci-dessous :

$(document).ready(function(){

/\*

\* Ecoutons l'évènement click()

\*/

$("#more\_com").click(function(){

$.ajax();

});

});

* Le code ci-dessus fonctionne sur tous les navigateurs qui supportent **XmlHttpRequest** et **ActiveX** de Microsoft
* **XmlHttpRequest** permet d'envoyer des requêtes HTTP depuis JavaScript, cette classe est donc à la base des appels AJAX.
* La fonction **$.ajax()** de jQuery nous permet d'instancier un objet très rapidement à partir de cette classe.
* Après l’instanciation de l’objet **XmlHttpRequest** nous pouvons commencer à l’exploiter un peu tel que vu dans le module précèdent avec JavaScript

1. **Le fonctionnement de $.ajax()**

* L’ensemble des appels AJAX avec JQuery passent par la fonction **$.ajax()**

### Les paramètres principaux de $.ajax()

##### **Le fichier ciblé**

* Lorsque vous envoyez une requête HTTP, vous demandez quelque chose au serveur.
* Dans notre cas de figure, il s'agit simplement du script côté serveur qui va aller chercher les commentaires dans la base de données et nous les retourner.
* Le paramètre url de $.ajax() permet de spécifier la ressource ciblée (notre script PHP) tel que ci-dessous :

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php' // La ressource ciblée

});

});

##### **Le type de la requête : POST ou GET ?**

* Nous allons envoyer une requête de type **GET**. Cela passe ici par le paramètre **type**.
* GET est le type de requête que jQuery prend par défaut

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php', // La ressource ciblée

type : 'GET' // Le type de la requête HTTP.

});

});

##### **Envoyer des éléments en GET**

* Si la requête est de type **GET**, rien ne vous empêche de faire passer des paramètres avec **GET** pour les utiliser côté serveur dans le tableau **$\_GET**.
* Pour cela, vous pouvez utiliser le paramètre **data** tel que ci-dessous.

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php', // La ressource ciblée

type : 'GET' // Le type de la requête HTTP.

data : 'utilisateur=' + nom\_user;

});

});

* Avec ceci, du côté serveur, **$\_GET['utilisateur']** contiendra la valeur de la variable **nom\_user**.

##### **Le type de données à recevoir**

* À ce niveau nous devons spécifier = le type de données que nous allons recevoir de PHP.
* Nous pouvons recevoir tout et n'importe quoi : du XML, du HTML, du texte, du JSON..., etc.
* Dans notre exemple nous allons utiliser du HTML afin de montrer que c'est tout à fait utilisable. Tel que ci-dessous :

$("#more\_come").click(function(){

$.ajax({

url : 'more\_com.php', // La ressource ciblée

type : 'GET', // Le type de la requête HTTP

/\*\*

\* Le paramètre data n'est plus renseigné, nous ne faisons plus passer de variable

\*/

dataType : 'html' // Le type de données à recevoir, ici, du HTML.

});

});

* Tout ce qu'il nous reste à faire maintenant, c'est de les traiter côté serveur.

##### **Envoyer des requêtes de type POST**

* Nous allons prendre pour exemple l’envoi d'un mail depuis une page de contact
* Le visiteur tape un message dans un formulaire HTML, et ce message vous est envoyé par email en AJAX.
* Si nous souhaitons réaliser ce système, il faut envoyer des informations au serveur avec notre requête HTTP, pour cela, il faut commencer à spécifier un type POST pour **$.ajax()**, tel que ci-dessous :

$("#envoyer").click(function(){

$.ajax({

url : 'send\_mail.php', // Le nom du script a changé, c'est send\_mail.php maintenant !

type : 'POST', // Le type de la requête HTTP, ici devenu POST

dataType : 'html'

});

});

* Le code ci-dessous permet de faire passer des variables JavaScript à notre script PHP.
* Ces variables contiennent les informations du formulaire, et PHP en a besoin pour agir côté serveur.
* Pour ce transfert nous allons utiliser l’argument **data**

$("#envoyer").click(function(){

$.ajax({

url : 'send\_mail.php',

type : 'POST', // Le type de la requête HTTP, ici devenu POST

data : 'email=' + email + '&contenu=' + contenu\_mail, // On fait passer nos variables, exactement comme en GET, au script more\_com.php

dataType : 'html'

});

});

### Les paramètres success, error et complete de la fonction $.ajax( )

### Partant de l’étape ou le serveur nous renvoie des données en réponse d’une requête de l’utilisateur.

### Nous allons gérer kle retour du serveur avec les paramètres

### Les Votre requête HTTP est donc prête, et pour peu que votre script côté serveur soit présent et fonctionnel, elle a toutes les chances de marcher. Après avoir fait son boulot, le serveur nous renvoie quelque chose. Manque de bol, nous n'avons rien pour gérer le retour de $.ajax(). Les paramètres complete, success et error vont nous être utilisés.

### Traiter les retours de la fonction

##### **L'appel AJAX a réussi : success**

### Le paramètre success permet de gérer le retour de la fonction $.ajax()en cas de réussite de l’appel AJAX.

### Le paramètre success prend en charge une fonction qui sera exécutée lorsque l’appel AJAX aura réussi.

### Par exemple :

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php',

type : 'GET',

dataType : 'html', // On désire recevoir du HTML

success : function(code\_html, statut){

// code\_html contient le HTML renvoyé

}

});

});

### L'argument statut est une chaîne de caractère automatiquement générée par jQuery pour vous donner le statut de votre requête.

##### **L'appel AJAX n’a pas réussi : error**

### Le paramètre error permet de gérer le retour de la fonction $.ajax()en cas d’échec de l’appel AJAX.

### Le paramètre error prend en charge une fonction qui sera exécutée

### Lorsque l’appel AJAX a échoué.

### Par exemple:

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php',

type : 'GET',

dataType : 'html',

success : function(code\_html, statut){

// success est toujours en place, bien sûr !

},

error : function(resultat, statut, erreur){

}

});

});

### La fonction exécutée par error prend en charges trois arguments : le résultat, le statut, et l'erreur. :

### L’argument resultat est un l’objet XHR renvoyé par jQuery

### L'argument statut est une chaîne de caractère automatiquement générée par jQuery pour vous donner le statut de votre requête.

### L’argument erreur est une fonction qui peut contient par exemple une chaine de caractère lorsque l’appel AJAX n’a pas fonctionné.

##### **Le paramètre complete de $.ajax() :**

### Le paramètre complete permet de gérer le retour de la fonction $.ajax()lorsque l’appel AJAX est effectué.

### Le paramètre complete prend en charge une fonction qui sera exécutée lorsque l’appel AJAX est effectué.

### Par exemple:

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php',

type : 'GET',

dataType : 'html',

success : function(code\_html, statut){

},

error : function(resultat, statut, erreur){

},

complete : function(resultat, statut){

}

});

});

### complete prend deux arguments :

### L’argument resultat est un l’objet XHR renvoyé par jQuery

### L'argument statut est une chaîne de caractère automatiquement générée par jQuery pour vous donner le statut de votre requête.

### C'est à l'intérieur de ces fonctions que nous allons devoir traiter la suite des évènements :

### Nous voulons maintenant ajouter les commentaires reçus du serveur à notre fil de commentaires sur notre page.

### Imaginons un bloc div portant l'identifiant #commentaires qui contienne les commentaires déjà chargés dans la page.

### Nous allons vouloir ajouter à cette division les commentaires reçus.

### Pour cela, il faut ajouter du contenu au niveau de la fonction de success. Tel que ci-dessous :

// On reprend le même id que dans le précédent chapitre

$("#more\_com").click(function(){

$.ajax({

url : 'more\_com.php',

type : 'GET',

dataType : 'html',

success : function(code\_html, statut){

$(code\_html).appendTo("#commentaires"); // On passe code\_html à jQuery() qui va nous créer l'arbre DOM !

},

error : function(resultat, statut, erreur){

},

complete : function(resultat, statut){

}

});

});

### Les raccourcis $.get() et $.post()

Tel que vu jusqu’à maintenant nous pouvons gérer facilement un appel AJAX avec la fonction **$.ajax()**et ses paramètres. Nous pouvons également passer par deux raccourcis d’utilisation **$.get()**et **$.post()**

**Les raccourcis $.get() et $.post()**

* Les méthodes **$.get()**et **$.post()permettent** permettant de créer directement une requête GET ou POST.
* Les deux fonctions font appel implicitement à **$.ajax()** en lui spécifiant un type GET pour **$.get()** et un type POST pour **$.post()**.

**Par exemple :**

/\*\*

\* $.get() vaut $.ajax() avec un type get

\* Ces lignes reviennent à faire un $.get();

\*/

$.ajax({

type : 'get'

});

/\*\*

\* De la même manière $.post() vaut $.ajax() avec un type post

\* Ces lignes reviennent à faire un $.post();

\*/

$.ajax({

type: 'post'

});

**Un appel AJAX en GET avec$.get()**

* La fonction **$.get()** est un raccourci de **$.ajax()** avec un type GET.
* Ceci dit nous avons besoin de la même liste d’arguments que précédemment.
  + L'URL du fichier appelé ;
  + Les données envoyées ;
  + Une fonction qui va gérer le retour ;
  + Le format des données reçues.

**Par exemple :**

$.get(

'fichier\_cible.php', // Le fichier cible côté serveur.

'false', // Nous utilisons false, pour dire que nous n'envoyons pas de données.

'nom\_fonction\_retour', // Nous renseignons uniquement le nom de la fonction de retour.

'text' // Format des données reçues.

);

function nom\_fonction\_retour(texte\_recu){

// Du code pour gérer le retour de l'appel AJAX.

}

* Le fonctionnement est exactement le même qu’avec l’utilisation précédente (format classique de **$.ajax()**)

**Un appel AJAX en POST avec$.post()**

* **$.post()**fonctionne de la même manière que **$.get():** c'est un raccourci pour créer rapidement un appel AJAX de type POST.
* Les arguments qu'elle prend sont identiques à ceux de **$.get()**
  + L'URL du fichier appelé ;
  + Les données envoyées ;
  + Une fonction qui va gérer le retour ;
  + Le format des données reçues.

Par exemple :

$.post(

'send\_mail.php', // Le fichier cible côté serveur.

{

sujet : $("#sujet").val(), // Nous supposons que ce formulaire existe dans le DOM.

contenu : $("#contenu").val()

},

nom\_fonction\_retour, // Nous renseignons uniquement le nom de la fonction de retour.

'text' // Format des données reçues.

);

function nom\_fonction\_retour(texte\_recu){

// Du code pour gérer le retour de l'appel AJAX.

}

### Sérialisation des formulaires

* Lorsque nous gérons un appel AJAX de type POST impliquant l'utilisation d'un formulaire, nous pouvons faciliter notre travail avec la **sérialisation**.

##### **La méthode serialize() de jQuery**

* Prenons l’exemple de l’utilisation de la fonction $.ajax()pour envoyer les variables issues d'un formulaire HTML à un script PHP.
* Un problème va se poser lorsque vous devrez spécifier vos variables et leurs contenus : vous devrez concaténer dans une chaîne de caractères toutes ces variables. Tel que ci-dessous :

.ajax({

/\*\*

\* Nous réutilisons $.ajax() pour illustrer l'exemple

\*/

// Concentrons nous sur le paramètres data

data : 'valeur1=' + valeur1 + '&valeur2=' + valeur2 + '&valeur3=' + valeur3,

});

* Pour faciliter cette tâche nous avons la possibilité de sérialiser le formulaire avec la **serialize()**de jQuery
* Le but de la méthode **serialize()**est de transformer les champs d'un formulaire en chaîne de caractères avec les variables et leurs contenus concaténés.

**Exemple d’utilisation :**

Vous avez ci-dessous le formulaire que nous voulons sérialiser

<form id="formulaire" method="POST" action="traitement.php">

<input type="text" name="valeur1" />

<input type="text" name="valeur2" />

<input type="text" name="valeur3" />

<input type="submit" name="submit" />

</form>

Le but est d'obtenir un format tel que ci-dessous :

data : 'valeur1=' + valeur1 + '&valeur2=' + valeur2 + '&valeur3=' + valeur3 pour l’envoyer par la suite en AJAX.

Le code ci-dessous permet de sérialiser notre formulaire :

$("#formulaire").submit(function(e){ // On sélectionne le formulaire par son identifiant

e.preventDefault(); // Le navigateur ne peut pas envoyer le formulaire

var donnees = $(this).serialize(); // On créer une variable content le formulaire sérialisé

$.ajax({

//...

data : donnees,

//...

});

});

### Charger des données JSON

### 5.1 Le Format JSON

### JSON (JavaScript Object Notation) est un format de données textuel qui permet de représenter des informations structurées.

* Vous avez ci-dessous une liste des signes de de ponctuations qui permettent de structurer les données :
  + {...} : les **accolades** définissent un objet."
  + language":"Java" : Les **guillemets** (double-quotes) et les **double-points** définissent un couple clé/valeur (on parle de membre).
  + [...] : Les **crochets** définissent un tableau
  + {"id":1, "language":"json", "author":"Douglas Crockford"} : Les **virgules** permettent de séparer les membres d'un tableau ou, comme ici, d'un objet . A noter : pas de virgule pour le dernier membre d'un objet, sinon, il ne sera pas valide et vous aurez des erreurs lors de l'analyse du fichier.

Ci-dessous un premier exemple d’un document JSON de base :

**Exemple 1**

{

"titre\_album":"Abacab",

"groupe":"Genesis",

"annee":1981,

"genre":"Rock"

}

L’exemple ci-dessus contient un objet (**{...}**) qui, ici, contient 4 membres identifiés par leurs paires **clé/valeur**. Les valeurs possibles sont :

* Une chaîne de caractères : "titre":"Le format json", "description":"Le format <strong>simple</strong> et <strong>léger</strong>, "contenu":"<p>L'avantage de json est son incroyable simplicité d'apprentissage et de mise en oeuvre. C'est le \"Petit Poucet\" de l'échange de données.</p>"
* Un nombre (pas de guillemets requis dans ce cas) : "pi":3.14, "g":9.81, "v\_son":340
* Un tableau : [...]
* Un objet : {...}
* D'autres valeurs possibles : un booléen (true ou false), null, rien ("alsanaute":true, "autrenaute":null,"bisounaute":""). Attention, ces valeurs doivent écrites en minuscule.

### Ci-dessous un exemple d’un fichier JSON avec des objets et des tableaux

{

"fruits": [

{ "kiwis": 3,

"mangues": 4,

"pommes": null

},

{ "panier": true }

],

"legumes": {

"patates": "amandine",

"poireaux": false

},

"viandes": ["poisson","poulet","boeuf"]

}

### Dans l’exemple ci-dessus nous avons :

### 3 membres (fruits, legumes et viandes).

### fruits est constitué d'un seul membre qui est un tableau de 2 objets :

### Le premier objet contient 3 membres et le second un seul.

### legumes est défini par un objet constitué par 2 membres.

### viandes, est défini par un tableau de 3 éléments.

### Remarqué que le niveau d’imbrication est illimité

### Interprétation d’un document JSON

### Afin de valider et visualiser le contenu d’un contenu JSON, vous pouvez le faire directement sur internet sur (à titre d’exemple ) l’adresse <http://jsonviewer.stack.hu/>.

### Vous avez aussi la possibilité de transformer un document JSON en XML sur (à titre d’exemple ) l’adresse <https://jsonformatter.org/>

### D'autres outils disponibles en ligne :

* + Formatage et validation : [JSON Formatter & Validator](http://jsonformatter.curiousconcept.com/)
  + Editeur et formatage : [JSON Editor Online](https://www.jsoneditoronline.org/)

### Charger des données codées en JSON

* Afin de manipuler des données JSON en JQuery, nous pouvons utiliser la fonction JQeury $.getJSON()qui permet de charger un fichier de données au format JSON
* Après le chargement du document, nous pouvons travailler sur les différentes données qui le composent en utilisant la fonction de rappel.

**Par exemple :**

En partant du document JSON ci-dessous :

{

"nom": "Pierre Durand",

"age": "27",

"ville": "Paris",

"domaine": "HTML5, CSS3, JavaScript"

}

Nous avons ci-dessous le code permettant de charger et manipuler les données du fichier JSON :

<button id="charger">Charger et traiter les données</button>

<div id="r">Cliquez sur "Charger et traiter les données" pour lancer la lecture et le traitement des données JSON</div>

<script src="jquery.js"></script>

<script>

$(function() {

$('#charger').click(function() {

$.getJSON('fichier.json', function(donnees) {

$('#r').html('<p><b>Nom</b> : ' + donnees.nom + '</p>');

$('#r').append('<p><b>Age</b> : ' + donnees.age + '</p>');

$('#r').append('<p><b>Ville</b> : ' + donnees.ville + '</p>');

$('#r').append('<p><b>Domaine de compétences</b> : ' + donnees.domaine + '</p>');

});

});

});

</script>

* Au clique sur le bouton la fonction getJSON() est exécutée pour charger le fichier de données fichier.json

$.getJSON('fichier.json', function(donnees) {

* Le deuxième paramètre de la fonction getJSON() correspond à la fonction de rappel.
* La fonction de rappel est exécutée lorsque le fichier de données a été entièrement chargé.
* Remarquez le mot donnees passé comme paramètre de la fonction de rappel. C'est par son intermédiaire que les données JSON seront accessibles.
* la valeur correspondant au nom (donnees.nom) est extraite du fichier de données et placée sous une forme HTML (html()) dans la balise <div>#r.
* Comme nous passons par la méthode html()pour remplir la balise <div>, il est possible d'utiliser des attributs de mise en forme. Ici, le mot « Nom » est mis en gras avec la balise HTML <b>.

$('#r').html('<p><b>Nom</b> : ' + donnees.nom + '</p>');

* La donnée **age** (**donnees.age**) est extraite du fichier de données et placée à la suite du nom, dans un nouveau paragraphe. Ici aussi, le nom du champ est mis en gras en utilisant la balise HTML <b>.
* Deux instructions similaires extraient les données **ville** et **domaine** du fichier de données JSON et les affichent à la suite du nom et de l'âge :

$('#r').append('<p><b>Ville</b> : ' + donnees.ville + '</p>');

$('#r').append('<p><b>Domaine de compétences</b> : ' + donnees.domaine + '</p>');

**Mise en pratique :**

En partant du document XML ci-dessous :

<sites>

<site id="2">

<title>Microsoft</title>

<url>http://www.microsoft.ca</url>

<desc>

<brief>Résumé</brief>

<long>Description longue</long>

</desc>

</site>

<site id="3">

<title>Intel</title>

<url>http://www.intel.com</url>

<desc>

<brief>Résumé</brief>

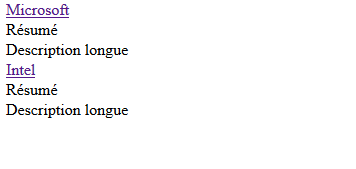
<long>Description longue</long>

</desc>

</site>

</sites>

* Utiliser les méthodes JQuery afin d’importer le contenu du document XML ci-dessus et d’afficher le contenu au moment du chargement d’une page Web tel que ci-dessous :



Au clique sur les liens Microsoft ou Intel envoie sur les adresses url de la balise <url>

**Liste des références :**

Sébastien de la Marck et Johann Pardanaud. Créez des pages web interactives avec JavaScript. Repéré au : <https://openclassrooms.com/courses/simplifiez-vosdeveloppements-javascript-avec-jquery/plus-loin-dans-la-selection-d-elements>

Michel Martin. [Créez des pages web interactives avec JavaScript. Repéré au : https://openclassrooms.com/courses/simplifiez-vos-developpements-javascript-avec-jquery/plus-loin-dans-la-selection-d-elements](\\\\sv27\\profs\\ytagmouti\\Module16\\Créez des pages web interactives avec JavaScript. Repéré au :  https:\\openclassrooms.com\\courses\\simplifiez-vos-developpements-javascript-avec-jquery\\plus-loin-dans-la-selection-d-elements)