Centre de Réadaptation de Mulhouse

TP1- UTILISATION D'AJAX AVEC TRAITEMENT SERVEUR ET JQUERY

1 INITIATION MIS EN ŒUVRE OBJET AJAX

AJAX (Asynchronous JavaScript and XML) n'est pas une technologie, c'est le résultat d'un ensemble de technologies du web qui fonctionnent ensemble pour arriver à un objectif simple : rafraîchir une partie de page web sans recharger la page complète.

AJAX... C'EST SIMPLE !

Exemple mettre à jour le fil de commentaire sans que la page soit rechargée entièrement quand vous laissez des « post »(s) sur un site. Le but du jeu est donc qu'au moment de la soumission du formulaire, une logique se mette en route côté AJAX (JQuery), qui fera que le commentaire sera ajouté en base de données, et qu'il soit automatiquement ajouté au fil, le tout sans recharger la page !

C'est totalement possible... et c'est même très répandu. Si vous êtes fan des réseaux sociaux, comme Facebook, vous avez forcément été confronté à l'un de ces fils de commentaire dans votre vie de geek.

JQUERY NE SUFFIT PAS !

Nous avons parlé d'un fil de commentaire qui se recharge en AJAX plus haut, et nous avons même évoqué l'idée d'une base de données. Vous devriez donc comprendre que jQuery est aussi capable d'interagir avec ce qu'il se passe côté serveur !

Si c'est ce que vous pensez, sachez que ce n'est que partiellement vrai. Pour interagir avec une base de données, jQuery va devoir appeler des scripts côté serveur, qui eux sont capables de le faire. Vous commencez à comprendre ? **AJAX repose sur ce fondement même, la communication asynchrone d'un langage côté client, avec un langage côté serveur.** Vous devinez la suite, pour mettre en place un appel AJAX sur son site, jQuery ne va plus nous suffire. Voilà ce dont on va avoir besoin :

Un langage côté client : nous utiliserons bien sûr JavaScript, avec ou sans jQuery.

Un langage côté serveur : nous utiliserons ici le PHP.

Le script PHP appelé fais son travail : envoi de mail, insertion en base de données... et surtout, il renvoie un résultat que jQuery va intercepter. Ce résultat, c'est ce que jQuery utilisera pour mettra à jour la page

UN APPEL AJAX SANS LE VOCABULAIRE TECHNIQUE

Pour une première approche nous utiliserons les fonctions Ajax de jQuery pour éviter une surcharge de vocabulaire technique propre aux objets Ajax XMLHttpRequest surnommé objet « XHR » surtout pour la compatibilité avec les anciens navigateurs.

En effet :

INSTANCIER XHR : UN CALVAIRE

Instancier un objet XHR peut devenir difficile... car il faut prendre en compte le problème de compatibilité entre les navigateurs. Les navigateurs Internet Explorer antérieurs à la version 7 utilisaient une implémentation différente de XHR : ActiveX, développé par Microsoft. Il va donc falloir prendre en compte ces navigateurs pour que notre appel AJAX soit mis en œuvre sur ces ceux-ci.

```
Exemple : var xhr = null;

if(window.XMLHttpRequest || window.ActiveXObject){
        if(window.ActiveXObject){
                try{
                        xhr = new ActiveXObject("Msxml2.XMLHTTP");
                }catch(e){
                        xhr = new ActiveXObject("Microsoft.XMLHTTP");
                }
        }else{
                xhr = new XMLHttpRequest();
        }
}else{
        alert("Votre navigateur ne supporte pas l'objet
XMLHTTPRequest...");
        return;
}
```

Ça fait peur n'est-ce pas ?

XMLHTTPREQUEST SANS JQUERY

Pour effectuer une requête asynchrone au serveur en utilisant l'objet XMLHttpRequest, nous allons toujours devoir suivre 4 étapes :

On crée un objet XMLHttpRequest ;

1) On initialise notre requête, c'est à dire on choisit le mode d'envoi des données, l'URL à demander, etc. ;

2) On envoie la requête ;

3) On crée des gestionnaires d'événements pour prendre en charge la réponse du serveur.

4) Pour créer un objet XMLHttpRequest, nous allons utiliser le constructeur XMLHttpRequest() avec la syntaxe classique new XMLHttpRequest().

Ensuite, pour initialiser notre requête, nous allons utiliser la méthode open() de XMLHttpRequest. On va pouvoir passer 2, 3, 4 ou 5 arguments à cette méthode :

Le premier argument (obligatoire) correspond au type de méthode de requête HTTP à utiliser. On va pouvoir choisir entre GET, POST, PUT, DELETE, etc. Dans la grande majorité des cas, on choisira GET ou POST.

Pour être tout à fait précis, on préférera GET pour des requêtes non destructives, c'est-à-dire pour effectuer des opérations de récupération simple de données sans modification tandis qu'on utilisera POST pour des requêtes destructives, c'est-à-dire des opérations durant lesquelles nous allons modifier des données sur le serveur ainsi que lorsqu'on souhaitera échanger des quantités importantes de données (POST n'est pas limitée sur la quantité de données, au contraire de GET).

Le deuxième argument (obligatoire) représente l'URL de destination de la requête, c'est-à-dire l'URL où on souhaite envoyer notre requête.

Le troisième argument (facultatif) est un booléen indiquant si la requête doit être faite de manière asynchrone ou pas. La valeur par défaut est true.

Les quatrième et cinquième arguments (facultatifs) permettent de préciser un nom d'utilisateur et un mot de passe dans un but d'authentification.

Une fois notre requête initialisée ou configurée grâce à open(), on va spécifier le format dans lequel le serveur doit nous renvoyer sa réponse en passant ce format en valeur de la propriété responseType de notre objet XMLHttpRequest. Les valeurs possibles sont les suivantes :

"" (chaine de caractères vide) : valeur par défaut; demande au serveur de renvoyer sa réponse sous forme de chaine de caractères ;

"text" : demande au serveur de renvoyer sa réponse sous forme de chaine de caractères ;

"arraybuffer" : demande au serveur de renvoyer sa réponse sous forme d'objet ArrayBuffer ;

"blob" : demande au serveur de renvoyer sa réponse sous forme d'objet Blob ;

"document" : demande au serveur de renvoyer sa réponse sous forme de document XML ;

"json" : demande au serveur de renvoyer sa réponse sous forme JSON.

Dans la majorité des cas, on demandera au serveur de nous renvoyer des données sous forme JSON (elles seront alors interprétées automatiquement) ou texte.

Une fois qu'on a défini le format de la réponse, nous allons pouvoir envoyer notre requête. Pour cela, nous allons utiliser la méthode send() de XMLHttpRequest.

Cette méthode ouvre la connexion avec le serveur et lui envoie la requête. On peut lui passer le corps de la requête en argument facultatif.

```
let xhr = new XMLHttpRequest();

    xhr.open("GET", "listeobjets.json", true);

    // xhr.responseType = "json";

    xhr.send();
```

PRENDRE EN CHARGE LA REPONSE RENVOYEE PAR LE SERVEUR

Une fois notre requête envoyée, nous allons devoir réceptionner la réponse du serveur. Pour connaitre l'état d'avancement de notre requête, nous avons deux options.

Utiliser les gestionnaires d'événements load, error et progress

Nous allons pour cela utiliser des gestionnaires d'événements définis par l'interface XMLHttpRequestEventTarget qui vont nous permettre de prendre en charge différents événements déclenchés par notre requête et en particulier :

L'événement load qui se déclenche lorsque la requête a bien été effectuée et que le résultat est prêt ;

l'événement error qui se déclenche lorsque la requête n'a pas pu aboutir ;

L'événement progress qui se déclenche à intervalles réguliers et nous permet de savoir où en est notre requête.

Au sein du gestionnaire d'événement load, on va déjà vouloir tester la valeur du statut code HTTP pour savoir si notre requête a bien abouti ou pas. Pour cela, nous allons observer la valeur de la propriété status de l'objet XMLHttpRequest.

Les statuts code HTTP les plus fréquents sont les suivants :

100 Continue : tout fonctionne jusqu'à présent; le client devrait continuer avec la requête ;

200 OK : Les requête a été un succès ;

301 Moved Permanently : L'identifiant de ressource unique (URI) relatif à la ressource demandée a changé de localisation de façon permanente ;

302 Found : L'identifiant de ressource unique (URI) relatif à la ressource demandée a changé de localisation de façon temporaire ;

304 Not Modified : Indique au client que la réponse n'a pas été modifiée depuis le dernier accès et qu'il peut utilisée la version en cache ;

401 Unauthorized : Indique que le client doit s'identifier s'il veut accéder à la réponse ;

403 Forbidden : Indique que le client n'a pas l'autorisation d'accéder à ce contenu ;

404 Not Found : Le serveur n'a pas pu trouver la ressource demandée ;

500 Internal Server Error : Le serveur a rencontré une situation qu'il ne peut pas gérer.

Ici, on va généralement tester si le statut code de notre réponse est bien égal à 200 en testant donc si la propriété status contient bien cette valeur. Si c'est le cas, on va pouvoir manipuler les données envoyées par le serveur.
Pour accéder à ces données, on va pouvoir utiliser la propriété response de l'objet XMLHttpRequest qui contient la réponse du serveur sous le format précisé par reponseType lors de l'envoi de la requête.

```
xhr.onload = function() {
        //Si le statut HTTP n'est pas 200...
        if (xhr.status != 200) {
           //...On affiche le statut et le message correspondant
           alert("Erreur " + xhr.status + " : " + xhr.statusText);
           //Si le statut HTTP est 200, on affiche le nombre d'octets téléchargés et la réponse
        } else {
           let tabdata = JSON.parse(xhr.responseText);
           // console.log(tabdata);
           generer_diapo(tabdata);
}
 };
```

Autre exemple :

```
//On veut une réponse au format JSON
xhr.responseType = "json";

//On envoie la requête
xhr.send();
```

```
//Dès que la réponse est reçue...
xhr.onload = function(){
   //Si le statut HTTP n'est pas 200...
   if (xhr.status != 200){
      //...On affiche le statut et le message correspondant
      alert("Erreur " + xhr.status + " : " + xhr.statusText);
   //Si le statut HTTP est 200, on affiche le nombre d'octets téléchargés et la réponse
   }else{
      alert(xhr.response.length + " octets  téléchargés\n" + JSON.stringify(xhr.response));
   }
};
```

Si vous travaillez en local, il est normal que la requête ci-dessus échoue (même en renseignant une bonne URL). Cela est dû à la politique CORS (Cross Origin Resource Sharing) qui interdit certaines requêtes pour protéger les utilisateurs. Nous reparlerons de cela plus tard.

Utiliser la propriété readyState et le gestionnaire onreadystatechange

Une autre méthode consiste à observer la valeur de la propriété readyState de l'objet XMLHttpRequest pour déterminer l'état d'avancement de la requête. On préfère cependant aujourd'hui utiliser les gestionnaires d'événements load, progress et error.

Pour information, les valeurs possibles de readyState sont les suivantes :

Valeur	Etat	Description
0	UNSENT	Le client a été créé mais open() n'a pas encore été appelée
1	OPENED	open() a été appelée
2	HEADERS_RECEIVED	send() a été appelée et l'en-tête et le statut sont disponibles
3	LOADING	Les données sont en train d'être téléchargées

4 DONE L'opération est complète

Si on choisit d'utiliser readyState pour suivre l'avancement de notre requête, on va alors utiliser un gestionnaire d'événement onreadystatechange qui va être appelé dès que la valeur de readyState change.

On passe une fonction anonyme à ce gestionnaire pour gérer la réponse. Dans cette fonction anonyme, on teste déjà que la valeur de readyState est bien égale à 4 ou à DONE. Cela signifie qu'on a reçu la réponse du serveur dans son intégralité et qu'on va donc pouvoir l'exploiter.

Ensuite, on teste la valeur du statut code de la réponse HTTP en observant la valeur de la propriété status de l'objet XMLHttpRequest pour savoir si notre requête est un succès ou pas.

Liste des propriétés et méthodes de l'objet XMLHttpRequest

Pour information, vous pourrez trouver ci-dessous un récapitulatif des propriétés et des méthodes de l'objet XMLHttpRequest ainsi qu'une rapide description.

Pour aller plus loin…

LES PROPRIETES DE L'OBJET XMLHTTPREQUEST

readyState = retourne un entier entre 0 et 4 qui correspond à l'état de la requête ;

onreadystatechange = gestionnaire d'événements appelé lorsque la valeur de readyState change ;

responseType = chaine de caractères précisant le type de données contenues dans la réponse ;

response = renvoie un objet JavaScript, ArrayBuffer, Blob, Document ou DOMString, selon la valeur de responseType, qui contient le corps de la réponse ;

responseText = retourne un DOMString qui contient la réponse serveur sous forme de texte ou null si la requête a échoué ;

responseURL = retourne une URL sérialisée de la réponse serveur ou la chaine de caractères vide si l'URL est null ;

responseXML = retourne un Document qui contient la réponse serveur ou null si la requête a échoué ;

status = retourne de statut code HTTP de la réponse ;

statusText = retourne un DOMString contenant la réponse complète HTTP (statut code + texte de la réponse) ;

timeout = représente le nombre de millisecondes accordées à un requête avant qu'elle ne soit automatiquement close ;

ontimeout = gestionnaire d'événements appelé lorsque la requête dépasse le temps accordé par timeout ;

upload = objet XMLHttpRequestUpload représentant la progression du téléchargement ;

withCredentials = booléen représentant si les requêtes d'accès et de contrôle cross-sites doivent être faites en utilisant des informations d'identification telles que les cookies ou les en-têtes d'autorisation.

LES METHODES DE L'OBJET XMLHTTPREQUEST

open() = initialise une requête en JavaScript ;

send() = envoie une requête asynchrone par défaut ;

abort() = abandonne la requête si celle-ci a déjà été envoyée ;

setRequestHeader() = définit la valeur de l'en-tête HTTP de la requête ;

getResponseHeader() = retourne la chaine de caractère contenant le texte de l'en-tête de la réponse spécifié ou null si la réponse n'a pas été reçue ;

getAllResponseHeaders() = retourne la réponse de tous les en-têtes ou null si aucune réponse n'a été reçue ;

overrideMimeType() = surcharge le type MIME retourné par le serveur.

PRESENTATION ET UTILISATION DE L'API FETCH EN JAVASCRIPT

Alternative à l'objet XHR

Dans cette leçon, nous allons étudier l'API Fetch et sa méthode fetch() qui correspondent à la "nouvelle façon" d'effectuer des requêtes HTTP.

Cette API est présentée comme étant plus flexible et plus puissante que l'ancien objet XMLHttpRequest.

PRESENTATION DE L'API FETCH ET DE LA METHODE FETCH()

L'API Fetch fournit une définition pour trois interfaces Request, Response et Headers et implémente également le mixin Body qu'on va pouvoir utiliser avec nos requêtes.

Les interfaces Request et Response représentent respectivement une requête et la réponse à une requête. L'interface Headers représente les en-têtes de requête et de réponse tandis que le mixin Body fournit un ensemble de méthodes nous permettant de gérer le corps de la requête et de la réponse.

L'API Fetch va également utiliser la méthode globale fetch() qui représente en quelques sortes le coeur de celle-ci. Cette méthode permet l'échange de données avec le serveur de manière asynchrone.

La méthode fetch() prend en unique argument obligatoire le chemin de la ressource qu'on souhaite récupérer. On va également pouvoir lui passer en argument facultatif un liste d'options sous forme d'objet littéral pour préciser la méthode d'envoi, les en-têtes, etc.

La méthode fetch() renvoie une promesse (un objet de type Promise) qui va se résoudre avec un objet Response. Notez que la promesse va être résolue dès que le serveur renvoie les en-têtes HTTP, c'est-à-dire avant même qu'on ait le corps de la réponse.

La promesse sera rompue si la requête HTTP n'a pas pu être effectuée. En revanche, l'envoi d'erreurs HTTP par le serveur comme un statut code 404 ou 500 vont être considérées comme normales et ne pas empêcher la promesse d'être tenue.

On va donc devoir vérifier le statut HTTP de la réponse. Pour cela, on va pouvoir utiliser les propriétés ok et status de l'objet Response renvoyé.

La propriété ok contient un booléen : true si le statut code HTTP de la réponse est compris entre 200 et 299, false sinon.

La propriété status va renvoyer le statut code HTTP de la réponse (la valeur numérique liée à ce statut comme 200, 301, 404 ou 500).

Pour récupérer le corps de la réponse, nous allons pouvoir utiliser les méthodes de l'interface Response en fonction du format qui nous intéresse :

La méthode text() retourne la réponse sous forme de chaine de caractères ;

La méthode json() retourne la réponse en tant qu'objet JSON ;

La méthode formData() retourne la réponse en tant qu'objet FormData ;

La méthode arrayBuffer() retourne la réponse en tant qu'objet ArrayBuffer ;

La méthode blob() retourne la réponse en tant qu'objet Blob ;

Code source :

```
fetch("https://www.une-url.com")
.then(response => response.json())
.then(response => alert(JSON.stringify(response)))
.catch(error => alert("Erreur : " + error));
```

Expliquons ce code ensemble. Tout d'abord, la méthode fetch() a besoin d'un argument obligatoire, qui correspond à l'URL des ressources à récupérer. On utilise ici une/url (remplacez bien évidemment par une vraie URL en pratique).

fetch() retourne ensuite une promesse contenant la réponse (si tout se passe bien). On ne peut pas exploiter la réponse renvoyée dans cette promesse en l'état : il faut indiquer le format de réponse souhaité. Ici, on choisit JSON avec response.json().

response.json() renvoie également une promesse contenant la réponse à votre demande en JSON. On utilise JSON.stringify() pour transformer notre objet JSON en une chaine JSON et on affiche cette chaine.

Finalement, on traite les erreurs avec le bloc catch et on affiche l'erreur rencontrée si on en rencontre effectivement une.

PASSER DES OPTIONS A FETCH()

Comme on l'a dit plus tôt, la méthode fetch() accepte un deuxième argument. Cet argument est un objet qui va nous permettre de définir les options de notre requête. On va pouvoir définir les options suivantes :

method : méthode utilisée par la requête. Les valeurs possibles sont GET (défaut), POST, etc.) ;

headers : les en-têtes qu'on souhaite ajouter à notre requête ;

body : un corps qu'on souhaite ajouter à notre requête ;

referrer : un référant. Les valeurs possibles sont "about:client" (valeur par défaut), "" pour une absence de référant, ou une URL ;

referrerPolicy : spécifie la valeur de l'en-tête HTTP du référent. Les valeurs possibles sont no-referrer-when-downgrade (défaut), no-referrer, origin, origin-when-cross-origin et unsafe-url ;

mode : spécifie le mode qu'on souhaite utiliser pour la requête. Les valeurs possibles sont cors (défaut), no-cors et same-origin ;

credentials : les informations d'identification qu'on souhaite utiliser pour la demande. Les valeurs possibles sont same-origin (défaut), omit et include ;

cache : le mode de cache qu'on souhaite utiliser pour la requête. Les valeurs possibles sont default (défaut), no-store, reload, no-cache, force-cache et only-if-cached ;

redirect : le mode de redirection à utiliser. Valeurs possibles : follow (défaut), manual, error ;

integrity : contient la valeur d'intégrité de la sous-ressource de la demande. Valeurs possibles : "" (défaut) ou un hash ;

keepalive : permet à une requête de survivre à la page. Valeurs possibles : false (défaut) et true ;

signal : une instance d'un objet AbortSignal qui nous permet de communiquer avec une requête fetch() et de l'abandonner.

Code source exemple :

```
let promise = fetch(url, {
  method: "GET", //ou POST, PUT, DELETE, etc.
  headers: {
    "Content-Type": "text/plain;charset=UTF-8" //pour un corps de type chaine
  },
  body: undefined, //ou string, FormData, Blob, BufferSource, ou URLSearchParams
```

```
    referrer: "about:client", //ou "" (pas de réferanr) ou une url de l'origine
    referrerPolicy: "no-referrer-when-downgrade", //ou no-referrer, origin, same-origin...
    mode: "cors", //ou same-origin, no-cors
    credentials: "same-origin", //ou omit, include
    cache: "default", //ou no-store, reload, no-cache, force-cache, ou only-if-cached
    redirect: "follow", //ou manual ou error
    integrity: "", //ou un hash comme "sha256-abcdef1234567890"
    keepalive: false, //ou true pour que la requête survive à la page
    signal: undefined //ou AbortController pour annuler la requête
});
```

XMLHTTPREQUEST AVEC JQUERY

XmlHttpRequest devient nettement plus facile à instancier avec jQuery. Une seule ligne de code, cela va aller très vite !

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

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

    $.ajax();

});
```

Intéressons-nous de plus près, à la fonction la plus importante en JQuery pour manipuler l'objet XHR et donc faire des requêtes Ajax…

```
$.ajax(options);
```

JQuery permet d'émettre des requêtes Ajax vers un serveur, en effectuant l'appel Ajax via la méthode $.ajax(options), et ceci indépendamment du navigateur utilisé.

Options de la methode $.ajax(options)

url : Correspond à l'url du programme sur le serveur qui traitera la requête. Cette option est obligatoire, sinon aucun traitement sur le serveur ne peut avoir lieu…

Data : Objet ou chaine de caractères qui sera transmis au serveur. Si on utilise une chaine de caractères, elle doit être de la forme name1=value1 & name2=value2…, chaque name étant le nom d'un paramètre et value la valeur correspondante, encodée en UTF-8.

Si on utilise un objet, jQuery encode lui-même en UTF-8 chacune des valeurs et transmet au serveur une chaine de la forme name1=value1& name2=value2…

Type : Mode de transmission des paramètres indiqués dans l'option data précédente. 2 valeurs sont possibles : « GET » (valeur par défaut) ou « POST »

Complete : Méthode de la forme : complete(xhr, textStatus) appelé à la fin de la requête Ajax, que celle-ci ait réussie ou non.

Le paramètre textStatus contient « success » si la requête a réussi sinon un message d'erreur, comme « error » , « timeout» ou « parseerror ».

Dans le cas où la requête a réussi, le paramètre « xhr » donne accès à l'objet « XMLHttpRequest » contenant la réponse serveur (dans xhr.responseText).

Username : Nom d'utilisateur à indiquer si le serveur nécessite une autorisation.

Password :Mot de passe associé à l'option username.

MISE EN APPLICATION

Vous réaliserez l'exemple suivant d'utilisation Ajax. On saisit un nom dans un champs text, puis on envoie à l'aide d'un bouton, le nom saisie vers le serveur qui le transforme en majuscule et le retourne vers le navigateur, qui affiche le nom transformé.

Le code de la partie serveur se trouve dans le fichier suivant :

```
< ?

$nom =$_REQUEST[ 'nom' ] ;

echo strtoupper($nom) ; //

?>
```

A vous de faire la page client.html contenant le code Javascript qui effectue le traitement souhaité dans l'énoncé.

2 CHARGEMENT DE DONNEES AU FORMAT JSON

Le format de données **JSON** (JAVASCRIPT OBJECT NOTATION) constitue le standard actuel pour les échanges de données sur le Web, notamment avec AJAX.

Il s'agit d'une syntaxe pour décrire des informations structurées sous une forme proche des objets JavaScript. Voici un exemple de document JSON qui décrit deux voitures (source).

```
{
  "voitures" : [
    { "modèle" : "Peugeot",
      "couleur" : "bleu",
      "immatriculation" : 2008,
      "révisions" : [ 2012, 2014 ]
    },
    { "modèle" : "Citroën",
      "couleur" : "blanc",
      "immatriculation" : 1999,
      "révisions" : [ 2003, 2005, 2007, 2009, 2011, 2013 ]
    }
  ]
}
```

Vous allez vous rendre compte que ce format ne présente pas beaucoup de défauts, principalement parce qu'il reste vraiment très simple à lire, comprendre et utiliser.

LES PLUS

Format compréhensible par tous (humain et machine). Aucun apprentissage n'est requis puisque la syntaxe n'utilise que quelques marques de ponctuations (nous le verrons plus tard).

Ne dépend d'aucun langage (format d'échange de données ouvert).Comme ce format est très ouvert, il est pris en charge par de nombreux langages : JavaScript, PHP, Perl, Python, Ruby, Java,...

Permet de stocker des données de différents types : chaînes de caractères (y compris des images en base64), nombres, tableaux (array), objets, booléens (true, false), la valeur null.

Sa structure en arborescence et sa syntaxe simple lui permet de rester très "léger" et efficace.

LES MOINS

Comme pour tout moyen de stockage de données, il faudra prendre des mesures de sécurité pour garantir la sécurité des données sensibles.

Le fait que la syntaxe soit rudimentaire peut être un inconvénient dans certains cas. Par exemple, contrairement à XML, il n'y a pas d'identification précise des données (sous forme de balise par exemple), la structure doit donc être connue avant utilisation.

LA SYNTAXE

Je vous parlais précédemment de quelques signes de ponctuations qui permettent de structurer les données dans un fichier .json. En voici la liste (vous ne serez normalement pas dépaysés puisqu'ils sont utilisés exactement de la même manière dans d'autres langages, comme JavaScript par exemple) :

- `{...}` : 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 (ou array en anglais).
- `{"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.

C'est tout ? Ben oui, c'est tout ! Une dernière précision cependant, et elle n'est pas anodine : tout doit être encodé en utf-8 (ou utf-16 ou utf-32).

Il est temps de découvrir un exemple de fichier .json basique :

```
{
    "titre_album":"Abacab",
    "groupe":"Genesis",
    "annee":1981,
    "genre":"Rock"
}
```

Suffisamment simple pour qu'il y ai très peu d'explication…

Un fichier .json 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 (`"crmnaute":true, "autreinternaute":null,"auteur":""`). Attention, ces valeurs doivent écrites en minuscule.

Sachez qu'il est possible d'ajouter autant d'espaces (au sens large du terme, c'est à dire que les retours à la ligne et les tabulations sont également considérés comme des espaces) que l'on veut. Le fichier ci-dessus pourra donc également s'écrire de la manière suivante, par exemple :

```
{
    "titre_album"
       :"Abacab",
    "groupe"
       :"Genesis",
    "annee"
       :1981,
    "genre"
       :"Rock"
 }
```

Voyons à présent un exemple plus complexe contenant des objets et des tableaux :

```
{
  "fruits": [
    { "kiwis": 3,
      "mangues": 4,
      "pommes": null
    },
    { "panier": true }
  ],
  "legumes": {
      "patates": "amandine",
```

```
      "poireaux": false
    },
    "viandes": ["poisson","poulet","boeuf"]
}
```

Dans cet exemple nous pouvons observer qu'il y a 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, quant à lui, est défini par un tableau de 3 éléments.

Cet exemple démontre l'extraordinaire imbrication (illimitée) de ce type de format…
Pour vérifier votre syntaxe json lors de la création du fichier, vous pouvez utiliser le site « parser » suivant

http://jsonviewer.stack.hu/

Webographie :

Pour de la doc. détaillée sur le format Json :

http://json.org/

http://www.tutorialspoint.com/json/index.htm

Site en Français.

https://www.xul.fr/ajax-format-json.php

MISE EN APPLICATION AJAX JQUERY ET BOOTSTRAP

ETAPE 1 – COMPOSANT BOOTSTRAP

A l'aide du code ci –dessous et du Framework CSS Bootstrap vous réaliserez le diaporama de photo ci-dessous en mettant en œuvre le composant « Carrousel » de Bootstrap avec les flèches de défilement.

---------- Code html avec les « class » Bootstrap ----------

```
<div id="carouselExampleControls" class="carousel slide" data-ride="carousel">
 <div class="carousel-inner">
  <div class="carousel-item active">
```

```
    <img class="d-block w-100" src="photos_volcans/1.jpg" alt="First slide">
   </div>
   <div class="carousel-item">
    <img class="d-block w-100" src="photos_volcans/2.jpg" alt="Second slide">
   </div>
   <div class="carousel-item">
    <img class="d-block w-100" src="photos_volcans/4.jpg" alt="Third slide">
   </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleControls" role="button" data-slide="prev">
   <span class="carousel-control-prev-icon" aria-hidden="true"></span>
   <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleControls" role="button" data-slide="next">
   <span class="carousel-control-next-icon" aria-hidden="true"></span>
   <span class="sr-only">Next</span>
  </a>
</div>
```

----------fin de code ----------

Le résultat doit ressembler au visuel ci-dessous…Les photos libres de droits vous sont fournis dans le dossier nommé « photos_volcans » …

ETAPE 2 INTEGRATION DE CONTENUS DYNAMIQUES AJAX AVEC JQUERY

A l'aide du fichier document word fourni qui contient les titres et numéro d'image. Vous devez créer un document Json structuré qui contient une liste d'objet Json…Chaque objet aura 3 propriétés un id :numero d'image , un titre (le texte correspondant) , un « alt » le texte correspondant plus les mots «- volcans en indonésie » pour le référencement

Ensuite à l'aide de ce fichier Json que vous devez charger Dynamiquement en Ajax avec les fonctions jquery vous devez afficher le titre et le numéro de chaque image, et le contenu de l'attribut alt qui s'affichera en légende en bas de chaque image …

ETAPE 3 INTEGRATION DE CONTENUS DYNAMIQUES AJAX SANS JQUERY

Même opération que précédemment mais les balises sont générées dynamiquement en javascript en fonction du nombre d'image, le chargement de données Json également… Le framework JQuery ne sera pas utilisé dans cet exercice…

ETAPE 4 AJAX AVEC JQUERY ET TRAITEMENT SERVEUR (PHP)

Vous devez faire en sorte que :

- Avec un mini-formulaire dans la page et un script serveur en php, on puisse déterminer le nombre d'images du diaporama avec une variable « min » et une variable « max ».