AJAX
avec jQuery

Table des matières

L'architecture informatique AJAX (Asynchronous JavaScript and XML) permet de construire des applications Web et des sites web dynamiques interactifs sur le poste client en se servant de différentes technologies ajoutées aux navigateurs web entre 1995 et 2005.

1. les requêtes HTTP

AJAX 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.

Pour interagir avec une base de données, jQuery va devoir appeler des scripts côté serveur. AJAX repose sur ce fondement même, la communication asynchrone d'un langage côté client, avec un langage côté serveur. Ainsi, pour mettre en place un appel AJAX sur un site, il faut :

- Un langage côté client : JavaScript, avec jQuery
- Un langage côté serveur : PHP
- Un appel AJAX

La base de chaque appel AJAX est une requête HTTP. jQuery va simplifier énormément l'appel et le suivi de cette requête HTTP, et le traitement du retour que fait PHP.

1.1. Le fonctionnement du web

Le web est un ensemble d'ordinateurs fonctionnant en réseau. On peut ranger ces ordinateurs dans deux catégories : les clients et les serveurs. Les serveurs sont des ordinateurs sur lesquels se trouvent les sites web, ils sont généralement très très puissants et fonctionnent en permanence.

Pour obtenir une page web, un client demande à un serveur une page web. Le serveur cherche dans son disque dur à la recherche de la page demandée, et il la renvoie au client. Si des fichiers JavaScript sont rattachés à cette page, le serveur les fait parvenir aussi au client. Le navigateur web du client lit ensuite le code HTML et interprète le code JavaScript que le serveur lui a renvoyé, et il affiche la page au visiteur.

Néanmoins, pour que le web fonctionne, il faut que le client et le serveur parlent la même langue. Le protocole utilisé sur le World Wide Web est le protocole HTTP. La "demande" que le client fait est ce que l'on appelle une « requête HTTP » ; ce que le serveur répond, c'est la « réponse HTTP ».

Il existe plusieurs types de requête HTTP. L'un d'eux est le type GET[1]. On l'utilise lors des rappels AJAX pour obtenir des données (exemple : un système pour recharger des commentaires dès que l'on clique sur un bouton « Plus de commentaires »).

Il est également possible d'envoyer des données grâce à GET. Généralement les caractères ?, = et & y sont présents :

```
http://promethee.eu.org/index.php?item=32&IDftp=1
```

1 obtenir

Le type POST est un type dédié à l'envoi de données, mais il est possible aussi d'en recevoir (exemple : un système pour envoyer un email).

Un grand nombre de types de requêtes HTTP existent, mais ils ne nous seront pas nécessaires pour traiter les appels AJAX car les types GET et POST couvriront 99% des besoins.

1.1.1. AJAX par Javascript

JavaScript a la particularité d'intégrer en natif une classe appelée XmlHttpRequest (XHR). En instanciant un objet à partir de cette classe, on peut envoyer une requête HTTP vers un serveur grâce à cet objet XHR.

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 qu'un appel AJAX soit mis en œuvre sur ces ceux-ci :

```
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("Le navigateur ne supporte pas l'objet XMLHTTPRequest...");
    return;
    }
```

1.1.2. XmlHttpRequest avec jQuery

Instancier un objet XmlHttpRequest devient extrêmement simple, grâce à la fonction jQuery.ajax() ou $.ajax() :

```
$.(document).ready(function(){
    /*
     * Utilisons $.ajax pour créer une instance de XmlHttpRequest
     */

    $.ajax();
});
```

Exemple : soit un fil de commentaire qui se recharge en AJAX en cliquant sur un bouton « plus de commentaires » d'identifiant #more_com. On va écouter l'évènement click() sur ce bouton et dès qu'il sera réalisé, on instanciera un objet XHR.

```
$(document).ready(function(){
    /*
     * Ecoutons l'évènement click()
     */
    $("#more_com").click(function(){
       $.ajax();
```

```
    });
});
```

XHR permet d'envoyer des requêtes HTTP depuis JavaScript, cette classe est donc à la base des appels AJAX. La fonction $.ajax() de jQuery permet d'instancier un objet très rapidement à partir de cette classe.

2. La fonction $.ajax()

La fonction $.ajax() est incontournable pour les appels AJAX en jQuery.

2.1. $.ajax() et ses paramètres

Lorsque l'on envoie une requête HTTP, on demande quelque chose au serveur. Dans ce 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.

Pour spécifier quelle est la ressource ciblée, il faut utiliser le paramètre url de $.ajax() :

```
$("#more_com").click(function(){

    $.ajax({
       url : 'more_com.php' // La ressource ciblée
    });

});
```

Le fichier PHP exécuté côté serveur s'appelle more_com.php. C'est un lien relatif, le fichier PHP se trouve donc dans le même répertoire que le fichier JavaScript, et le fichier HTML auquel il est lié. Ce code fonctionne, mais il ne fait rien. Pour envoyer des informations au serveur, il faut envoyer une requête de type GET :

```
$("#more_com").click(function(){

    $.ajax({
       url : 'more_com.php',  // La ressource ciblée
       type : 'GET'           // Le type de la requête HTTP
    });

});
```

NB : le type GET est le type que jQuery prend par défaut.

On peut faire passer des paramètres avec GET pour les utiliser côté serveur dans l'array $_GET. Pour cela, il faut utiliser le paramètre data :

```
$("#more_com").click(function(){

    $.ajax({
       url : 'more_com.php',  // La ressource ciblée
       type : 'GET'           // Le type de la requête HTTP.
       data : 'utilisateur=' + nom_user;
    });

});
```

Du côté serveur, $_GET['utilisateur'] contiendra la valeur de la variable nom_user.

Il faut spécifier maintenant le type de données à recevoir de PHP avec dataType. On peut recevoir tout et n'importe quoi : du XML, du HTML, du texte, du JSON... (on utilisera ici du HTML) :

```
$("#more_come").click(function(){

    $.ajax({
       url : 'more_com.php',  // La ressource ciblée
       type : 'GET',          // Le type de la requête HTTP
       /**
        * data n'est plus renseigné, on ne fait plus passer de variable
        */
       dataType : 'html'      // Le type de données à recevoir, ici, du HTML
    });

});
```

Pour envoyer des informations au serveur avec une requête HTTP, il faut commencer à spécifier un type POST :

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

    $.ajax({
       url : 'send_mail.php', // Le nom du script a changé : c'est send_mail.php
       type : 'POST',         // Le type de la requête HTTP, ici devenu POST
       dataType : 'html'
    });

});
```

Pour faire passer les variables JavaScript qui contiennent les informations du formulaire au script PHP, il faut uiliser l'argument data :

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

    $.ajax({
       url : 'send_mail.php',
       type : 'POST', // Le type de la requête HTTP, ici devenu POST
       // On fait passer les variables au script more_com.php
       data : 'email=' + email + '&contenu=' + contenu_mail,
       dataType : 'html'
    });

});
```

2.2. Les paramètres success, error et complete

jQuery propose un paramètre pour gérer le retour de la fonction $.ajax() en cas de réussite. Le paramètre s'appelle success, et il ne sera exécuté QUE si l'appel AJAX a abouti. Ce paramètre prend en charge une fonction qui sera exécutée :

```
$("#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 donner le statut de la requête.

Si l'appel AJAX a rencontré une erreur, le paramètre qui va être employé sera error. Le paramètre exécute une fonction si l'appel AJAX a échoué :

```
$("#more_com").click(function(){

    $.ajax({
       url : 'more_com.php',
       type : 'GET',
       dataType : 'html',
       success : function(code_html, statut){
            // success est toujours en place
       },
       error : function(resultat, statut, erreur){
       }
    });

});
```

La fonction exécutée par error prend en charges trois arguments : le résultat, le statut, et l'erreur. Le second argument est le même que pour success, il ne nous sera pas utile. Le premier est un objet XHR renvoyé par jQuery. Le dernier est une exception : vous pouvez ici placer une chaîne de caractère à afficher à votre visiteur si l'appel AJAX n'a pas fonctionné.

Un dernier paramètre de $.ajax() : complete, qui va s'exécuter une fois l'appel AJAX effectué.

Le paramètre va prendre deux arguments, l'objet resultat dont nous avons parler plus haut ainsi qu'un statut :

```
$("#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){
       }
    });

});
```

C'est à l'intérieur de ces fonctions qu'il faut traiter la suite des évènements.

Exemple : ajouter les commentaires reçus du serveur à un fil de commentaires sur une page. Imaginons un bloc div portant l'identifiant #commentaires qui contienne les commentaires déjà chargés dans la page. Nous allons y ajouter les commentaires reçus.

```
// 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){
           // On passe code_html à jQuery() qui va nous créer l'arbre DOM !
           $(code_html).appendTo("#commentaires");
       },
       error : function(resultat, statut, erreur){
       },
       complete : function(resultat, statut){
       }
    });

});
```

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

Écrire une fonction $.ajax() pour ensuite définir son type GET/POST dans le paramètre adéquat, c'est lourd. Pour créer directement une requête GET ou POST, il existe les fonctions $.get() et $.post(). Toutes les deux fonctionnent de la même manière, elles font appel implicitement à $.ajax() en lui spécifiant un type GET pour $.get() et un type POST pour $.post() :

```
/**
 * $.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'
});
```

Les arguments demandés sont les mêmes :

- l'URL du fichier appelé
- les données envoyées*
- une fonction qui va gérer le retour
- le format des données reçues

L'argument marqué d'une astérisque désigne les données envoyées. On peut également faire passer des variables en GET avec des URL ayant le bon format ; ?prenom=lorem&nom=ipsum en est un exemple.

```
$.get(
    'fichier_cible.php', // Le fichier cible côté serveur.
```

```
    'false',            // false indique 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){
    // code pour gérer le retour de l'appel AJAX.
}
```

La fonction de retour prend un argument : il est automatiquement créé par $.get() et il contient les données reçues du serveur.

$.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 sa soeur :

- l'URL fichier appelé
- les données envoyées
- une fonction qui va gérer le retour
- le format des données reçues

Avec POST, il devient obligatoire d'envoyer des variables, c'est la raison d'être même de ce type.

Imaginons l'envoi d'un email en AJAX avec la fonction $.post() :

```
$.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){
    // code pour gérer le retour de l'appel AJAX.
}
```

2.4. Sérialisation des formulaires

Pour envoyer les variables issues d'un formulaire HTML à un script PHP en AJAX, il faut concaténer dans une chaîne de caractères toutes ces variables. Pour cela, on utilise la sérialisation du formulaire avec la méthode serialize(). Son but est de transformer les champs d'un formulaire en chaîne de caractères avec les variables et leurs contenus concaténés :

```
<!-- Formulaire HTML super simple à 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 data : 'valeur1=' + valeur1 + '&valeur2=' + valeur2 + '&valeur3=' + valeur3 afin

de l'envoyer en AJAX :

```
$("#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,
    //...
    });
});
```

3. Un formulaire de connexion avec AJAX

Nous allons créer ensemble un formulaire de connexion en AJAX grâce à jQuery.

Nous devrons d'abord créer le formulaire de connexion en HTML qui demandera un nom d'utilisateur ainsi qu'un mot de passe. Ensuite, nous utiliserons jQuery afin de lancer un appel AJAX vers un script PHP appelé login.php. Enfin, nous enverrons à ce script les données que l'utilisateur a tapé dans le formulaire.

Ce fichier PHP comparera les informations reçues avec des données pré-enregistrées. Si elles correspondent, nous renverrons un texte indiquant le succès de l'opération. Sinon, nous renverrons une erreur.

3.1. Côté client : du HTML et un appel AJAX

Pour commencer, nous allons créer une page HTML. Elle devra être liée à jQuery, mais surtout proposer un formulaire HTML afin de laisser le visiteur entrer un pseudo et un mot de passe :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Un formulaire de connexion en AJAX</title>
</head>
<body>
    <h1>Un formulaire de connexion en AJAX</h1>
    <form>
    <p>
        Nom d'utilisateur : <input type="text" id="username" />
        Mot de passe : <input type="password" id="password" />
        <input type="submit" id="submit" value="Se connecter !" />
    </p>
    </form>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
</body>
</html>
```

Ajoutons l'appel AJAX : étant donné que nous souhaitons utilisons un formulaire, nous allons faire transiter nos données grâce à POST et donc utiliser la fonction $.post().

```
<!DOCTYPE html>
```

```
<html>
<head>
<!-- ... -->
</head>
<body>
<!-- ... -->
<script>
$(document).ready(function(){
    $("#submit").click(function{
        $.post(
            'connexion.php', // Un script PHP que l'on va créer juste après
            {
                login : $("#username").val();  // Nous récupérons la valeur de
nos inputs que l'on fait passer à connexion.php
                password : $("#password").val();
            },
            function(data){ // Cette fonction ne fait rien encore, nous la
mettrons à jour plus tard
            },
            'text' // Nous souhaitons recevoir "Success" ou "Failed", donc on
indique text !
         );
    });
});
</script>
</body>
</html>
```

Remarque : une bonne pratique est de créer un nouveau fichier pour chaque appel AJAX à effectuer sur la page web et ensuite de le lier à la page HTML grâce à la balise <script>.

3.2. Côté serveur : un script PHP de connexion

Nous allons ici créer un script PHP qui va recevoir les données AJAX et les traiter afin de connecter le membre.

Côté serveur, il faut vérifier si les données ont bien été entrées dans le formulaire HTML. Nous utiliserons le tableau $_POST pour effectuer ces vérifications. Cela marche exactement comme si on avait soumis le formulaire en HTML. Des données sont passées en POST et sont présentes dans la requête HTTP. Nous pouvons y accéder grâce à $_POST. La première chose à faire est de vérifier que les données nommées username et password sont bien créées et ne sont pas vides.

```
<?php
    if( isset($_POST['username']) && isset($_POST['password']) ){
    }
?>
```

Prochaine étape : charger les données utilisateur.

Il faut comparer les données entrées dans le formulaire à des données pré-enregistrées. En réalité, vous comparerez certainement les données entrées à des valeurs en base de données. Ici, nous allons faire les choses plus simplement, et comparer de manière très basique les données à des variables dans notre script :

```
<?php
    /**
     * Nous créons deux variables : $username et $password
```

```
     * qui valent respectivement "Sdz" et "salut"
     */
    $username = "Sdz";
    $password = "salut";

    if ( isset($_POST['username']) && isset($_POST['password']) ){
    }
?>
```

Si les informations correspondent, nous allons renvoyer le texte « Success » en démarrant une session ; si les informations ne correspondent pas, nous allons renvoyer le texte « Failed ».

```
<?php
    /**
     * Nous créons deux variables : $username et $password
     * qui valent respectivement "Sdz" et "salut"
     */
    $username = "Sdz";
    $password = "salut";

    if ( isset($_POST['username']) && isset($_POST['password']) ){
        if ( $_POST['username'] == $username && $_POST['password'] == $password ){
            // Si les infos correspondent...
            session_start();
            $_SESSION['user'] = $username;
            echo "Success";
            }
        else   // Sinon
            echo "Failed";
        }
?>
```

3.3. De retour côté client

Quel que soit le résultat côté serveur, il faut afficher quelque chose au visiteur. C'est ici qu'il faut gérer le résultat de l'appel AJAX grâce aux attributs de $.post().

On sait que notre script PHP nous renvoie forcément deux chaînes de caractères : « success » ou « failed ». On va donc simplement jouer du côté jQuery avec quelques conditions... la valeur data a été remplie automatiquement par jQuery grâce à l'affichage que nous avons fait en PHP !

Rajoutons un bloc div en HTML qui portera l'identifiant #resultat afin d'y ajouter un retour pour le visiteur :

```
<!DOCTYPE html>
<html>
<head><!-- ... --></head>
<body>
    <div id="resultat">
        <!-- Nous allons afficher un retour en jQuery au visiteur -->
    </div>
    <form>
        <!-- Le formulaire donné plus haut-->
    </form>
<script>
$(document).ready(function(){
    $("#submit").click(function{
        $.post(
```

```
            'connexion.php', // Un script PHP que l'on va créer juste après
            {
                login : $("#username").val();  // Nous récupérons la valeur de
nos input que l'on fait passer à connexion.php
                password : $("#password").val();
            },

            function(data){
                if ( data == 'Success' ){
                    // Le membre est connecté. Ajoutons lui un message dans la
page HTML.
                    $("#resultat").html("<p>Vous avez été connecté avec
succès !</p>");
                }
                else{
                    // Le membre n'a pas été connecté. (data vaut ici "failed")
                    $("#resultat").html("<p>Erreur lors de la
connexion...</p>");
                }
            },

            'text'
        );
    });
});
</script>
</body>
</html>
```

4. Les évènements AJAX

4.1. Les requêtes AJAX sur écoute

Les évènements que l'on va écouter concernent directement AJAX, mais ils fonctionnent de la même manière que les autres : ils s'activent en fonction de l'état de la requête AJAX.

On pourra employer la méthode on(), mais aussi des fonctions qui leurs sont dédiées.

L'évènement ajaxStart() permet d'écouter le début d'une requête AJAX, il est réalisé dès que l'utilisation de la fonction $.ajax() - et donc $.get() ou $.post() - est faite :

```
$("p").ajaxStart(function(){
    console.log("L'appel AJAX est lancé !");
});
```

Dès qu'un appel AJAX présent sur cette page sera lancé, une fonction va envoyer dans la console le texte : « L'appel AJAX est lancé ! »

Pour écouter le succès d'une requête AJAX se déroulant depuis la page sur laquelle nous travaillons, on utilise l'évènement ajaxSuccess(). Il s'utilise de la même manière qu'ajaxStart(), mais écoute quand à lui le succès d'une requête, pas son début :

```
$("p").ajaxSuccess(function(){
    console.log("L'appel AJAX a réussi !");
});
```

Pour écouter l'échec d'une requête AJAX, un autre évènement est disponible : ajaxError(). Il est

réalisé quand une requête AJAX en cours sur la page échoue.

```
$("p").ajaxError(function(){
    console.log("L'appel AJAX a échoué !");
});
```

4.2. Cas d'utilisation des évènements

Nous allons ici aborder l'utilisation des évènements AJAX dans un cas concret : un loader.

Reprenons ici l'exemple lorsque nous voulions récupérer des commentaires en base de données, lors du clic sur le bouton portant l'identifiant #more_com. Ici, nous afficherons un loader (une image indiquant le chargement) animé lorsque la requête AJAX démarrera :

```
// Nous ajoutons un élément après le bouton
$("<div id='loading'></div>").insertAfter("#more_com");

// Nous appliquons du CSS à cet élément pour y afficher l'image en background
$("#loading").css({
    background : "url(load.gif)",    // On affiche l'image en arrière-plan
    display : "none"                 // Nous cachons l'élément
});

$("#more_com").click(function(){
    $.get(

        'more_com.php',
        false,
        'fonction_retour',
        'text'
    );

    // Nous ciblons l'élément #loading qui est caché
    $("#loading").ajaxStart(function(){
        $(this).show(); // Nous l'affichons quand la requête AJAX démarre
    });
});
```

Il existe également un évènement ajaxComplete() que l'on peut utiliser pour cacher à nouveau le loader. Ainsi, son affichage ne durera que la durée de la requête AJAX.

5. La méthode load()

La fonction $.ajax() et ses dérivées ne sont pas les seules manières de gérer un appel AJAX en jQuery sur un site web. Une autre fonction un peu moins connue existe : load().

La première chose que load() va faire sera d'instancier un objet XmlHttpRequest.

Son fonctionnement est en revanche beaucoup plus simple et basique que les fonctions reposant sur $.ajax(). Le but de load() est de récupérer le contenu qu'on lui spécifie en paramètre, puis l'injecter dans le DOM. C'est là sa principale utilité, dès qu'il faut récupérer un contenu qui ne nécessite pas forcément une logique (appel à une BDD, conditions...) qui se trouve sur le serveur et que l'on doit ajouter au DOM.

La méthode load() va prendre d'abord en paramètre la ressource ciblée. Nous allons opter pour un fichier HTML statique, étant donné que l'utilité première de la fonction est de charger du contenu

statique en AJAX.

Nous considérons qu'un bloc div portant l'identifiant #container existe dans le DOM. C'est avec ce bloc que load() va devoir travailler. Nous allons charger à l'intérieur un fichier appelé contenu.html :

```
// contenu.html se trouve au même niveau dans l'arborescence.
$("#container").load("contenu.html");
```

Ce qui se trouve dans le fichier contenu.html est alors injecté par jQuery dans le bloc div.

Lorsque on l'utilise de cette manière, tout le contenu de ce fichier HTML va être ajouté à l'élément donné, et cela concerne également les feuilles de style qui pourraient être rattachées au fichier inclus.

Pour éviter cela, nous allons devoir affiner la fonction. Par exemple, récupérer uniquement une certaine balise dans ce contenu mais pas tout le reste :

```
// un élément portant l'id "content" existe dans contenu.html
$("#container").load("contenu.html #content");
```

On ne récupérera que l'élément portant l'identifiant qui nous intéresse dans la page contenu.html, et on pourra l'ajouter simplement à un élément courant.

Bien que ce ne soit pas son utilisation première, load() permet également d'envoyer des données au serveur en requête POST. Pour cela, il faut lancer un appel AJAX avec load() en ciblant un fichier PHP, et en lui faisant passer des données.

```
/**
 * Nous reprenons notre exemple de formulaire de connexion
 *
 */
$("#container").load("connexion-avec-ajax.php", { // N'oubliez pas l'ouverture
des accolades !
    login : $("login").val(),
    password : $("password").val()
});
```

Côté serveur, les valeurs deviennent récupérables dans le tableau POST avec $_POST['login'] et $_POST['password'] comme nous l'avons vu dans les chapitres précédents. Le rôle de load() étant d'injecter le retour dans l'élément ciblé.

6. Un tchat en AJAX

Nous allons créer un tchat en AJAX, qui permettra aux visiteurs de laisser un message qui sera rendu visible directement à tous les autres participants à la conversation.

6.1. Le système de tchat

Les tchats en AJAX sont très répandus sur le web, on en trouve généralement sur les pages d'accueil des forums où les membres peuvent laisser un message qui sera instantanément transmis à tous les autres membres, pour peu qu'ils soient connectés au forum.

À son arrivée sur la page, l'utilisateur verra deux zones qui lui seront proposées. Sur le haut de la page, il verra les messages du tchat qui seront postés. Sur la partie inférieure de la page, il trouvera un petit formulaire HTML, dans lequel il pourra renseigner son pseudo et son message. Un bouton lui permettra d'envoyer son message, et celui-ci sera ajouté au tchat sans rechargement de page.

Toutes les 5 secondes, une vérification sera faite : des nouveaux messages ont-ils été postés ? Si oui, ils seront affichés. Si non, on affichera un petit message d'information : « aucun message n'a été ajouté dans les 5 dernières secondes ».

Nous avons évoqué l'idée que l'utilisateur doit pouvoir poster son message sans recharger la page, il va donc nous falloir utiliser de l'AJAX et utiliser plusieurs scripts côté serveur : l'un enverra les messages dans une base de données, et l'autre surveillera la base de données à intervalle de 5 secondes, pour voir si de nouveaux messages ont été postés.

6.2. Codage

6.2.1. Le fichier HTML

L'utilisateur doit avoir un formulaire HTML pour lui permettre de renseigner pseudo et message :

```
<!DOCTYPE html>
<html>
    <head>
    <title>Le tchat en AJAX !</title>
    </head>

    <body>
        <div id="messages">
            <!-- les messages du tchat -->
        </div>
    <form method="POST" action="traitement.php">
        Pseudo : <input type="text" name="pseudo" id="pseudo" /><br />
        Message : <textarea name="message" id="message"></textarea><br />
        <input type="submit" name="submit" value="Envoyez votre message !"
id="envoi" />
    </form>
    </body>
</html>
```

6.2.2. PHP et SQL

Pour réaliser une application utilisant de l'AJAX, on la créera d'abord en PHP classique, puis on ajoutera AJAX. Il y a deux avantages à cette pratique :

- Une meilleure organisation.
- L'assurance que l'application fonctionnera même si JavaScript est désactivé.

Le fichier PHP va se charger d'envoyer les informations tapées dans le formulaire en base de données. Il faut donc créer une base de données : elle sera très simple, elle contiendra une seule table message contenant le pseudo de l'auteur et son message.

```
CREATE TABLE IF NOT EXISTS `messages` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `auteur` varchar(100) NOT NULL,
  `message` text NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;
```

Nous créons ensuite un fichier PHP qui envoie en base de données ce qui a été tapé dans le formulaire HTML :

```
<?php
```

```
// on se connecte à la base de données
try
{
    $bdd = new PDO('mysql:host=localhost;dbname=tchat', 'root', '');
}
catch (Exception $e)
{
    die('Erreur : ' . $e->getMessage());
}

if ( isset($_POST['submit']) ){ // si on a envoyé des données avec le formulaire
    // si les variables ne sont pas vides
    if ( !empty($_POST['pseudo']) AND !empty($_POST['message']) ){
        $pseudo  = mysql_real_escape_string($_POST['pseudo']);
        $message = mysql_real_escape_string($_POST['message']); // on sécurise
nos données
        // puis on entre les données en base de données :
        $insertion = $bdd->prepare('INSERT INTO messages VALUES("", :pseudo,
:message)');
        $insertion->execute(array(
            'pseudo' => $pseudo,
            'message' => $message
        ));
    }
    else
        echo "Vous avez oublié de remplir un des champs !";
}
?>
```

6.2.3. La couche jQuery

On va initialiser une requête AJAX pour récupérer les données qui nous intéressent dans le formulaire. Comme la requête est de type POST, on utilisera soit $.ajax(), soit $.post() :

```
// on sécurise les données
var pseudo  = encodeURIComponent( $('#pseudo').val() );
var message = encodeURIComponent( $('#message').val() );

// on vérifie que les variables ne sont pas vides
if ( pseudo != "" && message != "" ){
    $.ajax({
        url : "traitement.php",      // on donne l'URL du fichier de traitement
        type : "POST",               // la requête est de type POST
        data : "pseudo=" + pseudo + "&message=" + message // et on envoie nos
données
    });
}
```

Le problème, c'est que ce code va s'exécuter dès le chargement de la page. Pour envoyer des données après le clic sur le bouton d'envoi, il suffit alors d'utiliser un évènement ici :

```
$('#envoi').click(function(e){
    e.preventDefault(); // on empêche le bouton d'envoyer le formulaire
    // on sécurise les données
    var pseudo    = encodeURIComponent( $('#pseudo').val() );
    var message   = encodeURIComponent( $('#message').val() );

    // on vérifie que les variables ne sont pas vides
    if ( pseudo != "" && message != "" ){
```

```
        $.ajax({
            url : "traitement.php", // on donne l'URL du fichier de traitement
            type : "POST",          // la requête est de type POST
            data : "pseudo=" + pseudo + "&message=" + message // et on envoie
nos données
        });
    }
});
```

Pour un maximum de dynamisme, nous allons ajouter le message directement dans la zone prévue à cet effet :

```
$('#envoi').click(function(e){
    e.preventDefault(); // on empêche le bouton d'envoyer le formulaire
    // on sécurise les données
    var pseudo  = encodeURIComponent( $('#pseudo').val() );
    var message = encodeURIComponent( $('#message').val() );

    // on vérifie que les variables ne sont pas vides
    if ( pseudo != "" && message != "" ){
        $.ajax({
            url : "traitement.php", // on donne l'URL du fichier de traitement
            type : "POST",          // la requête est de type POST
            data : "pseudo=" + pseudo + "&message=" + message // et on envoie
nos données
        });

        // on ajoute le message dans la zone prévue
        $('#messages').append("<p>" + pseudo + " dit : " + message + "</p>");
    }
});
```

Il faut maintenant s'occuper de récupérer les messages en base de données. D'abord, nous allons récupérer les dix derniers postés, en PHP. Puis, à intervalle de 5 secondes, il s'agira de vérifier s'il y en a eu de nouveaux, et si c'est le cas, de les afficher :

```
<!DOCTYPE html>
<html>
    <head>
    <title>Le tchat en AJAX !</title>
    </head>

    <body>
        <div id="messages">
            <!-- les messages du tchat -->
            <?php
                // on se connecte à la base de données
                try
                {
                    $bdd = new PDO('mysql:host=localhost;dbname=tchat', 'root',
'');
                }
                catch (Exception $e)
                {
                    die('Erreur : ' . $e->getMessage());
                }
                // on récupère les 10 derniers messages postés
                $requete = $bdd->query('SELECT * FROM messages ORDER BY id DESC
```

```
LIMIT 0,10');
                while($donnees = $requete->fetch()){
                    // on affiche le message (l'id servira plus tard)
                    echo "<p id=\"" . $donnees['id'] . "\">" . $donnees['pseudo']
. " dit : " . $donnees['message'] . "</p>";
                }
                $requete->closeCursor();
            ?>
        </div>

    <form method="POST" action="traitement.php">
        Pseudo : <input type="text" name="pseudo" id="pseudo" /><br />
        Message : <textarea name="message" id="message"></textarea><br />
        <input type="submit" name="submit" value="Envoyez votre message !"
id="envoi" />
    </form>

        <script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js"></script>
        <script src="main.js"></script>
    </body>
</html>
```

Le plus récent se trouvera en première position. Pour lancer une action toutes les 5 secondes, il faut utiliser setTimeout(). En créant une fonction se répétant indéfiniment, il suffit d'indiquer un intervalle de 5000 millisecondes.

```
function charger(){
    setTimeout( function(){
        // on lance une requête AJAX
        $.ajax({
            url : "charger.php",
            type : GET,
            success : function(html){
                // on veut ajouter les nouveaux messages au début du bloc
#messages
                $('#messages').prepend(html);
            }
        });

        charger(); // on relance la fonction
    }, 5000); // on exécute le chargement toutes les 5 secondes
}

charger();
```

La base étant posée, on doit penser depuis quel message il faut commencer à compter dans la base de données . On ne doit pas charger tous les messages, mais seulement les nouveaux. Il faut donc passer l'id du message le plus récemment affiché au fichier PHP, pour qu'il récupère tous les messages ayant un id plus élevé.

Pour cela, on va le prendre dans l'identifiant du premier paragraphe :

```
function charger(){
    setTimeout( function(){
        // on récupère l'id le plus récent
        var premierID = $('#messages p:first').attr('id');
```

```
        $.ajax({
            // on passe l'id le plus récent au fichier de chargement
            url : "charger.php?id=" + premierID,
            type : GET,
            success : function(html){
                $('#messages').prepend(html);
            }
        });

        charger();
    }, 5000);
}

charger();
```

Il ne reste plus qu'à aller chercher les messages en base de données :

```
<?php
// ...
// on se connecte à notre base de données
if(!empty($_GET['id'])){ // on vérifie que l'id est bien présent et pas vide
    $id = (int) $_GET['id']; // on s'assure que c'est un nombre entier
    // on récupère les messages ayant un id plus grand que celui donné
    $requete = $bdd->prepare('SELECT * FROM messages WHERE id > :id ORDER BY id
DESC');
    $requete->execute(array("id" => $id));
    $messages = null;

    // on inscrit tous les nouveaux messages dans une variable
    while($donnees = $requete->fetch()){
        $messages .= "<p id=\"" . $donnees['id'] . "\">" . $donnees['pseudo'] .
" dit : " . $donnees['message'] . "</p>";
    }

    echo $messages; // enfin, on retourne les messages à notre script JS
}
?>
```

6.3. Améliorations

Ce tchat en AJAX n'est pas directement réutilisable car il est bourré de failles de sécurité ! En effet, il n'y a que des vérifications des données basiques, aussi bien côté serveur que côté client. Ex : empêcher les utilisateurs d'utiliser les mêmes pseudos, vous utiliserez des sessions pour y arriver.