ÉCOLE POLYTECHNIQUE DE L'UNIVERSITE FRANÇOIS RABELAIS DE TOURS
DEPARTEMENT INFORMATIQUE
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 18
www.polytech.univ-tours.fr

Rapport de Project d'option web

Classifieur multimédia, interactif et en ligne

Etudiants :
Fabien TOUCHARD
fabien.touchard@etu.univ-tours.fr
Amaury GRENIER
amaury.grenier@etu.univ-tours.fr
DI5 2011-2012

Enseignant :
Gilles VENTURINI
gilles.venturini@univ-tours.fr

Version du 9 Mai 2012

Table de matières

Présentation du sujet

1. Contexte

Notre projet fait partie d'un projet plus vaste mené par un groupe d'étudiants en 4ème année dans le cadre de leur projet collectif. Leur client, INNOPHYT, a formulé le besoin de posséder un ensemble d'outils performants d'aide à la décision dans la reconnaissance d'insectes. INNOPHYT est un centre d'expertise et de transfert Universitaire (CETU), rattaché à l'université de Tours, spécialisé dans le domaine de la lutte antiparasitaire durable.

Dans leur travail du quotidien, ils ont besoin de pouvoir classifier correctement une espèce (notamment pour savoir si elle est nuisible). Aujourd'hui, ceci est réalisé à l'aide d'un fichier excel qui répertorie les questions qui permettent de déterminer une espèce. Le but du projet est donc de moderniser cet outil afin de proposer un ensemble d'applications améliorant le confort de travail. La suite d'applications comportera au final :

- Une application mobile permettant à l'utilisateur de déterminer une espèce
- Une application PC sous la forme d'un site web proposant les mêmes fonctionnalités
- Une application PC permettant d'administrer cet ensemble

Dans le cadre de notre projet d'option, nous avons en charge la réalisation du site web.

2. Objectifs

L'objectif est donc de fournir une application web qui, à partir de la lecture d'un fichier XML représentant un arbre de décision, permettra à l'utilisateur de naviguer dans cet arbre afin de classer une espèce.
Le produit final devra avoir les fonctionnalités suivantes:

- Une connexion sécurisée mais sans base de donnée
- Un système de navigation, type fil d'Ariane, pour naviguer dans les questions auxquelles on a déjà répondu. Ceci afin de pouvoir changer d'avis
- L'affichage d'une webcam connectée à la machine, afin de pouvoir filmer l'insecte observé
- La lecture de données multimédias (image, vidéo, son) et leur présentation sous la forme d'une galerie.

Une interface ergonomique et conviviale.

3. Choix technologiques

Pour répondre à ces différents objectifs, nous avons choisi un ensemble de solutions et bibliothèques disponibles sur internet. Nous avons choisi d'utiliser ces bibliothèques pour les raisons suivantes:

- Gagner du temps sur le développement
- Leur documentation est bien fournie
- Ce sont des bibliothèques stables et qui sont amenées à évoluer, apportant des fonctionnalités supplémentaires à notre projet.
- Profiter de ce projet pour nous former sur des outils utilisé en entreprise

Nous avons donc choisi des bibliothèques bien documentées, avec une communauté active afin d'assurer la stabilité et l'évolutivité de notre architecture.

Pour poser les bases de notre design, nous avons choisi d'utiliser Bootstrap une bibliothèque développée et utilisée par Twitter. En plus des feuilles de style CSS, Bootstrap comprend un certain nombre de composants intéressants comme des boutons, des menus, une barre de chargement… Il contient aussi quelques plugins JavaScript comme le plugin Carrousel, qui va nous servir pour afficher les galeries de médias.

Nous avons ajouté à cela une autre bibliothèque, ShadowBox, qui permet de lire tous type d'objets multimédias (vidéo, son, image) dans une fenêtre modale, et ainsi profiter de toute la largeur de l'écran.

Pour l'affichage de la webcam, nous avons simplement utilisé un code minimaliste en Flash, qui nous permet de démarrer la webcam après autorisation de l'utilisateur, et d'afficher à l'écran l'image filmée.

Enfin, afin de rendre l'interface plus conviviale, nous avons décidé de mettre en place une architecture AJAX. Cela nous permet notamment de supprimer les rechargements de pages inutiles dans notre cas car comme nous allons le voir, il suffit de changer la question et les réponses en fonction de l'avancement dans l'arbre. Nous avons utilisé JQuery et ces fonctions spécialisées dans l'architecture AJAX pour simplifier la gestion de notre application.

Le fichier XML

1. Présentation

L'arbre de décision qui contient toutes les données à afficher sur notre site et contenu dans un fichier XML. Le format de ce fichier a été défini par les 4èmes années. Il est important de ne pas modifier la structure de ce fichier afin d'assurer la compatibilité entre les différentes applications développées.

Voici un exemple de fichier XML :

```
<?xml version = '1.0' encoding="UTF8"?>
<!DOCTYPE arbre SYSTEM "arbre.dtd">

<arbre>
<branche id="b1" type="Accueil" date="jj/mm/yyy">
    <question id="q1" texte="Que voyez vous ?">
        <media>
            <legende>Vous devez nous dire ce que vous avez vue !!!</legende>
        </media>
        <reponse id="r1" texte="6 pattes">
            <media>
                <img id="img1" src="toto6pattes.png" />
                <legende>C'est toto avec 6 pattes</legende>
            </media>
            <branche id="b2">
                <!-- ... -->
            </branche>
        </reponse>
        <reponse id="r2" texte="8 pattes, ailes et antennes absentes">
          <branche id="b3" type="Arachnide">
            <question id="q2" texte="Quelle est la taille de l'arachnide ?">
                <reponse id="r3" texte="Supérieur à 5 mm" >
                  <resultat id="res1">
                    <nom>Nom de la morpho-espèce trouvée</nom>
                    <type>MEL1</type>
                    <regimeAlimentaire>Prédateur</regimeAlimentaire>
                    <informations>Informations complementaires sur la morpho-espèce</informations>
                    <media>
                        <img id"img2" src="coucou.jpg" />
                    </media>
                  </resultat>
                </reponse>
                <reponse id="r4" texte="Inférieur à 5 mm">
                    <!-- ... -->
                </reponse>
            </question>
          </branche>
        </reponse>
    </question>
</branche>
</arbre>
```

Figure 1 : Exemple de fichier XML (incomplet)

Les règles de formalisme attachées à ce fichier XML sont définies dans le DTD « arbre.dtd » qui est appelé à la 2ème ligne du fichier XML. Le DTD actuel est le suivant :

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!ELEMENT arbre (branche)>
4
5  <!ELEMENT branche (question)>
6  <!ATTLIST branche id ID #REQUIRED>
7  <!ATTLIST branche type CDATA #IMPLIED>
8  <!ATTLIST branche date CDATA #IMPLIED>
9
10 <!ELEMENT question (reponse+, media*)>
11 <!ATTLIST question id ID #REQUIRED>
12 <!ATTLIST question texte CDATA #REQUIRED>
13
14 <!ELEMENT reponse ((branche|resultat), media*)>
15 <!ATTLIST reponse id ID #REQUIRED>
16 <!ATTLIST reponse texte CDATA #REQUIRED>
17
18 <!ELEMENT resultat (nom, type, regimeAlimentaire, informations, media*)>
19 <!ELEMENT nom (#PCDATA)>
20 <!ELEMENT type (#PCDATA)>
21 <!ELEMENT regimeAlimentaire (#PCDATA)>
22 <!ELEMENT informations (#PCDATA)>
23 <!ATTLIST resultat id ID #REQUIRED>
24
25 <!ELEMENT media ((img?, sound?, video?), legende?)>
26 <!ELEMENT img EMPTY>
27 <!ATTLIST img id ID #REQUIRED>
28 <!ATTLIST img src CDATA #REQUIRED>
29 <!ELEMENT sound EMPTY>
30 <!ATTLIST sound id ID #REQUIRED>
31 <!ATTLIST sound src CDATA #REQUIRED>
32 <!ELEMENT video EMPTY>
33 <!ATTLIST video id ID #REQUIRED>
34 <!ATTLIST video src CDATA #REQUIRED>
35 <!ELEMENT legende (#PCDATA)>
```

Figure 2 : DTD du fichier XML

Il s'agit donc d'un ensemble de règles qui peuvent être traduites ainsi:

- Un arbre contient un unique nœud fils de type branche (ligne 3)
- Une branche contient un unique nœud fils de type question (ligne 5)
- Une branche a pour attribut obligatoire un identifiant (ligne 6) et des attributs type et date optionnels (lignes 7 et 8)
- Une question contient au moins une réponse et 0,1 ou plusieurs médias qui lui sont attachés (ligne 10) et possède comme attributs obligatoires un identifiant et un texte (lignes 11 et 12)
- Une réponse peut contenir soit une autre branche (cas récursif), soit un résultat (feuille de l'arbre) et on peut y attacher 0,1 ou plusieurs médias (ligne 14). Comme la question, elle possède comme attribut un identifiant et un texte (lignes 15 et 16).
- Un résultat correspond à une espèce identifiable. Il est composé d'informations sur cette espèce: nom, type, régime alimentaire, informations (lignes 19-22). Ainsi que 0,1 ou N médias pour la reconnaître. Enfin, un résultat possède un identifiant (ligne 23).

- Pour finir, un média peut être une image, un son ou une vidéo avec éventuellement une légende. Les différents types de médias sont des balises auto-fermantes (lignes 26, 29, 32) donc sans contenu et possède comme attributs obligatoires un identifiant et une url source.

2. Lecture du fichier XML

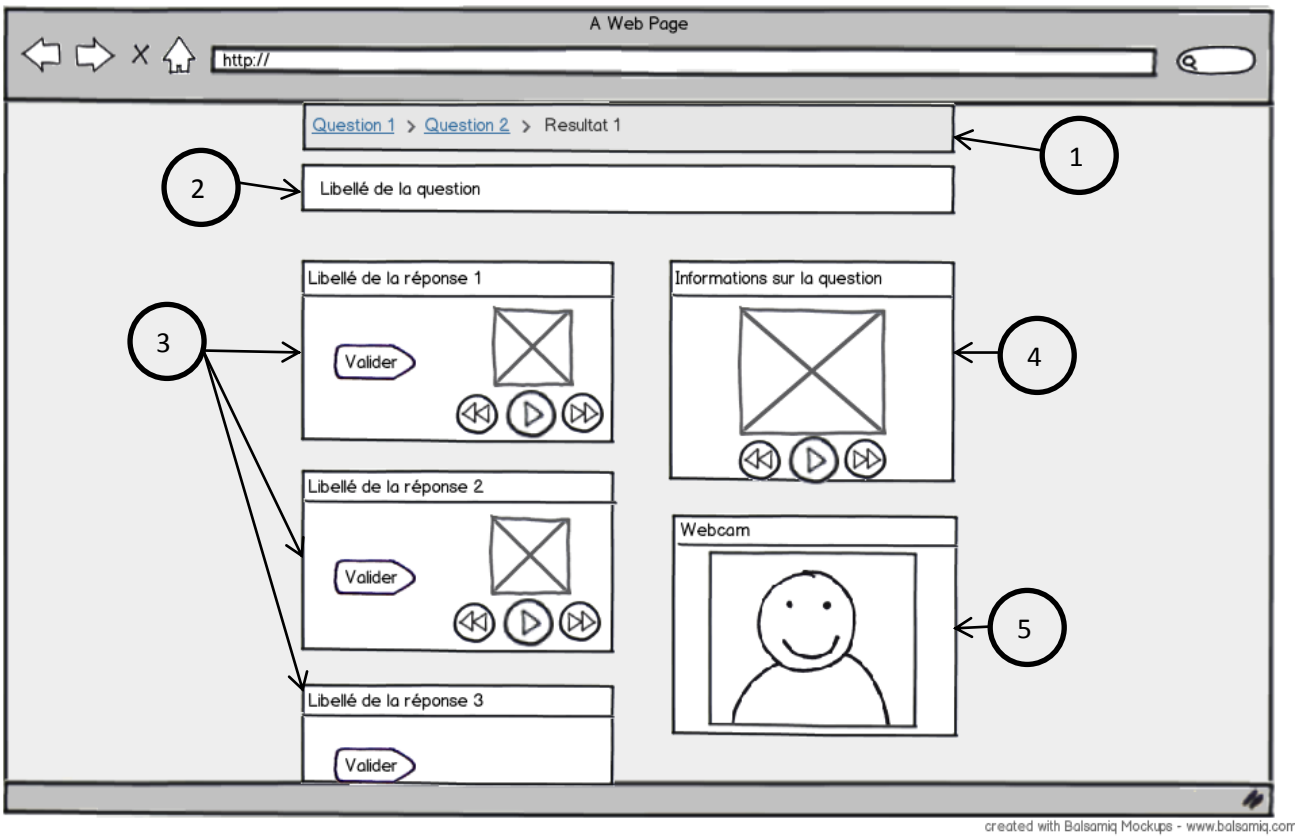
Nous avons besoin de lire ce fichier pour récupérer au besoin les informations relatives à une question, à une réponse ou un résultat. Pour cela, nous avons utilisé l'extension DOM qui permet d'utiliser des documents XML via l'API DOM de PHP 5. Un serveur web capable d'interpréter un script PHP est donc nécessaire pour le fonctionnement de cette application.

Cette extension est normalement installée par défaut avec PHP mais si besoin, toute la documentation se trouve ici : http://www.php.net/manual/fr/intro.dom.php. Dans cette bibliothèque, nous avons plus précisement utilisé la classe DOMDocument, dont la document est disponible ici : http://www.php.net/manual/fr/class.domdocument.php. Cette classe nous a permis de générer facilement un arbre DOM à partir du fichier XML, et de n'en garder que la partie qui nous intéresse en fonction d'un identifiant de question ou de réponse passé en paramètre.

L'ensemble des fonctions de lecture du fichier XML sont situées dans le fichier xmlParser.php dans le dossier php/ de notre projet.

Maquette de l'application

Nous avons essayé de mettre en place une interface conviviable. L'interface tient dans une unique page qui contient l'ensemble des informations nécessaires découpées en bloc bien distincts. On peut schématiser l'application comme ceci:



En haut de la page, collé à la barre du navigateur, on trouve le fil d'Ariane qui permet de naviguer dans les questions déjà posées (1). Juste au-dessous, le bloc contenant le texte de la question en cours (2). Ce bloc reste visible même si on descend en bas de la page pour voir l'ensemble des réponses. Puis le reste de la page est divisé en deux colonnes. Sur la gauche, une succession de blocs contenant chacun une réponse (3), avec le libellé de réponse, une galerie de médias et un bouton valider. Sur la droite, un premier bloc contenant les médias associés à la question (4) et un dernier bloc contenant la webcam (5).

ShadowBox

ShadowBox est une bibliothèque JavaScript qui permet de mettre en place des diaporamas sur un site web. Elle présente plusieurs avantages :

- Bonne intégration avec JQuery que nous utilisons déjà
- Les diaporamas sont affichés sur toute la largeur de la page grâce à une fenêtre modal, ce qui ajoute au confort de visionnage
- Elle permet l'ajout de nombreux formats de vidéos (flv, avi, wmv), de pistes audios (wav, mp3,…) et bien sûr de nombreux formats d'images.
- Elle est utilisable gratuitement pour des fins non-commerciales (20$ pour une licence commerciale)

Pour plus d'informations sur ShadowBox, voir le site du projet www.shadowbos-js.com .

1. Installation

ShadowBox se présente sous la forme d'une archive qu'il suffit de placer dans le répertoire de notre projet. Cette archive contient un fichier shadowbox.js qui contient la bibliothèque de fonction. Il faut inclure ce fichier dans notre page web pour l'utiliser. En plus de ce fichier, l'archive contient un ensemble d'images et un fichier CSS pour styliser l'affichage des galeries, et un player flash pour lire les vidéos et les sons.

2. Utilisation

Tout d'abord, on inclus la bibliothèque dans notre site.

```
<link rel="stylesheet" type="text/css" href="shadowbox/3.0.3/shadowbox.css">
<script type="text/javascript" src="shadowbox-3.0.3/shadowbox.js"></script>
```

Il faut ensuite créer de manière dynamique avec JavaScript (ou statique en HTML) un ensemble de balises <a> contenant chacune un média. Pour que ces médias soient regroupés au sein de la même galerie, il faut leur donner le même attribut **rel**, comme expliqué dans la documentation :

```
<a rel="shadowbox[Mixed];" href="myimage.jpg">jpg</a>
<a rel="shadowbox[Mixed];width=520;height=390" href="myswf.swf">swf</a>
<a rel="shadowbox[Mixed];width=292;height=218" href="mymovie.mp4">movie</a>
<a rel="shadowbox[Mixed]" href="mywebsite.html">iframe</a>
```

Nous avons ici un ensemble de quatre medias de types différents, regroupés dans une galerie "Mixed". Enfin après avoir créé toutes nos galeries, il suffit d'initialiser l'objet shadowBox comme ceci :

```
$(document).ready(function() {
    Shadowbox.init();
});
```

Pour plus d'informations, voir le site du projet et notamment la page de tutorial : http://www.shadowbox-js.com/usage.html .

Twitter Bootstrap

Le Bootstrap de Twitter permet de poser les bases d'un design. Il a été développé par Twitter et est utilisé sur leur site. On peut le télécharger et avoir des exemples de ce qu'il nous permet de faire son site : http://twitter.github.com/bootstrap/.

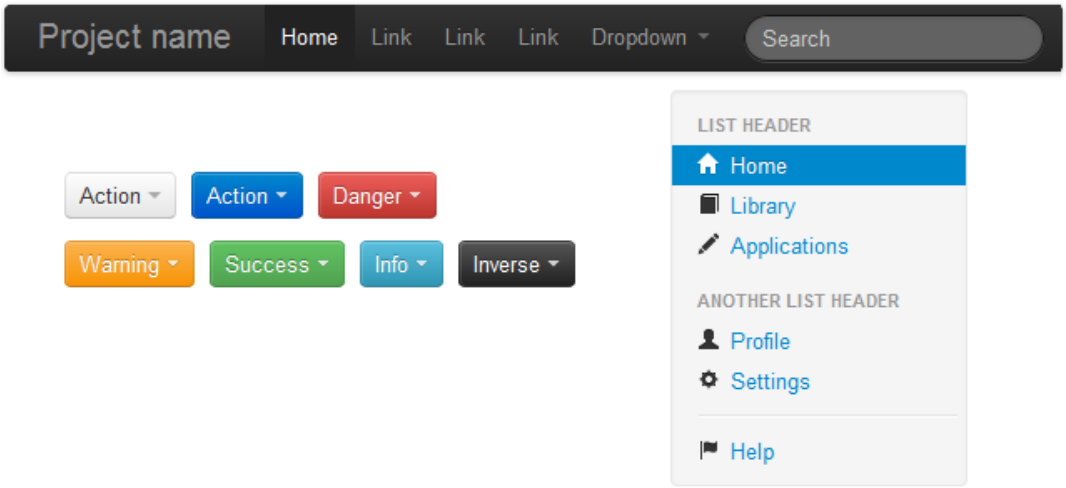


Figure 4. Exemples de composants offert par le Bootstrap

Le Bootstrap nous a permis de commencer le développement de notre design sur de bonnes bases. Il est complétement paramétrable et modifiable, on peut aussi lui adjoindre des feuilles de style que développer de notre côté.

Il contient plusieurs composants comme des boutons, des menus ou même des barres de chargement graphiquement stylisés. Il continent aussi des plugins JavaScript utilisant la célèbre bibliothèque jQuery (http://jquery.com). Nous avons notamment utilisé de carrousel (http://twitter.github.com/bootstrap/javascript.html#carousel) permettant de générer un diaporama intégré à la page.

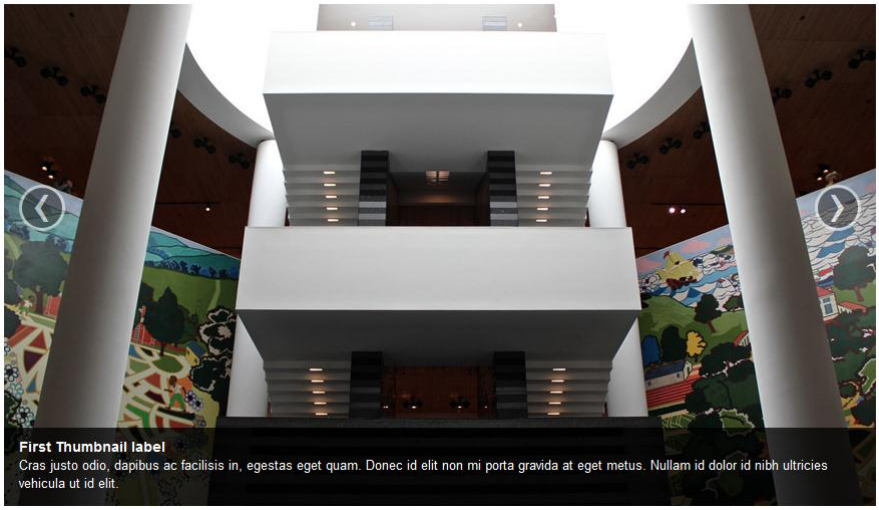


Figure 5. Exemple de caroussel

Authentification HTTP

Un point à développer pour notre application était de restreindre son accès uniquement à des utilisateurs autorisés. N'ayant pas de base de données pour gérer des comptes utilisateurs nous avons mis en œuvre une authentification HTTP. Nous allons donc voir dans cette partie en quoi elle consiste et comment la configurer.

Une authentification HTTP permet de s'identifier auprès d'un serveur web à l'aide d'un couple nom d'utilisateur, mot de passe. Si les identifiants correspondent alors l'accès à la ressource restreinte est accordé. Sur le serveur web Apache – car c'est celui qui est utilisé pour notre application – la restriction d'accès se fait à l'aide de 2 fichiers que je vais vous présenter.

Le premier fichier est le fichier « .htaccess », il se place à la racine du dossier à protéger – dans notre cas il s'agit de la racine de l'application, étant donné que nous voulons restreindre l'accès à la totalité du site. Voyons ensemble ce qui va composer ce fichier.

.htaccess

```
AuthUserFile /var/www/site/password/.htpasswd
AuthGroupFile /dev/null
AuthName "Acces restreint"
AuthType Basic
<limit GET>
   require valid-user
</Limit>
```

- **AuthUserFile** : Chemin d'accès jusqu'au fichier contenant les nom d'utilisateurs et mot de passes – il s'agit du fichier que nous verrons par la suite. Le chemin d'accès doit nécessairement être absolu, et donc partir de la racine du serveur.

- **AuthGroupFile** : Permet de définir des groupes d'utilisateur. Nous ne nous en servons pas, sa valeur est donc placée à « /dev/null ».

- **AuthName** : Défini le texte qui apparaitra à l'écran lors de la demande d'authentification.

- **AuthType** : Il s'agit du type d'authentification. La valeur peut aussi être « Digest » pour que l'idenfiant et le mot de passe ne transitent pas en clair lors de la communication, mais cette fonctionnalité n'est pas prise en charge sur tous les navigateur. On se contentera donc ici d'utiliser la valeur « Basic ».

- **Require valid-user** : Cette ligne permet simplement d'autoriser l'accès aux utilisateurs que l'on retrouvera dans la liste contenue dans le fichier « .htpasswd ».

L'étape suivante est donc logiquement de spécifier le fichier « .htpasswd » qui va contenir les utilisateurs. Il faut placer se fichier au bon endroit, conformément à ce qui a été spécifié dans le fichier « .htacces », dans l'attribut « AuthUserFile ».

.htpasswd

```
admin:admin
utilisateur:mot_de_passe
```

```
robert:pomme
```

Dans ce fichier une ligne correspond à un utilisateur. On va retrouver sur chaque ligne le nom d'utilisateur et le mot de passe séparés par le caractère ':'. On voit donc dans le fichier d'exemple que l'utilisateur « robert » va pouvoir se connecter avec le mot de passe « pomme ». Pour ajouter un utilisateur il suffit de rajouter une ligne au fichier, et de la structurer de la bonne manière.

Structure du site

1. Arborescence du site

Dans cette partie nous allons voir comment est structuré le site, les différents dossiers et fichiers qui le compose.

Voici donc la structure du site :

```
Racine du site
|    .htaccess
|    arbre.dtd
|    index.php
|    structure_xml.xml
|
+---bootstrap
|
+---css
|
+---images
|
+---js
|        contentManagment.js
|        jquery-1.7.2.min.js
|        script.js
|
+---medias
|
+---n8g779hf9hdcx9
|        .htpasswd
|
+---php
|        xmlparser.php
|
+---shadowbox-3.0.3
|
\---webcam
```

- **Racine du site** : A la racine du site on retrouve le fichier « .htaccess » que l'on a vu précédemment. On retrouve aussi le fichier XML décrivant l'arbre des questions et réponses et son DTD associé permettant de définir la structure du fichier XML. Et enfin il y a le fichier « index.php » qui est le point d'entrée du site.

- **bootstrap** : Ce dossier contient tous les fichiers relatif au Bootstrap : JavaScript et feuilles de style.

- **images** : On retrouve dans ce dossier les images relative au design.

- **js** : Ici il y a les différents fichiers JavaScript. « contentManagment.js » contient différents fonction javaScript pour modifier le contenu de la page. « script.js » contient le reste des fonctions JavaScript utile qui ont été développées. Et enfin « jquery-1.7.2.min.js » est la bibliothèque jQuery qui vide la simplification du développement en JavaScript.

- **media** : Dans ce dossier on retrouve les différents média utilisés pour l'arbre de questions/réponses.

- **n8g779hf9hdcx9** : Ce dossier au nom improbable contient le fichier mettant à disposition la liste des utilisateur autorisés. Il est nommé ainsi afin de limiter les chance de le trouver, car le fichier « .htpasswd » contient les mots de passe en clair.

- **php** : Ce dossier a en son sein le script PHP permettant de faire le parsing du fichier XML.

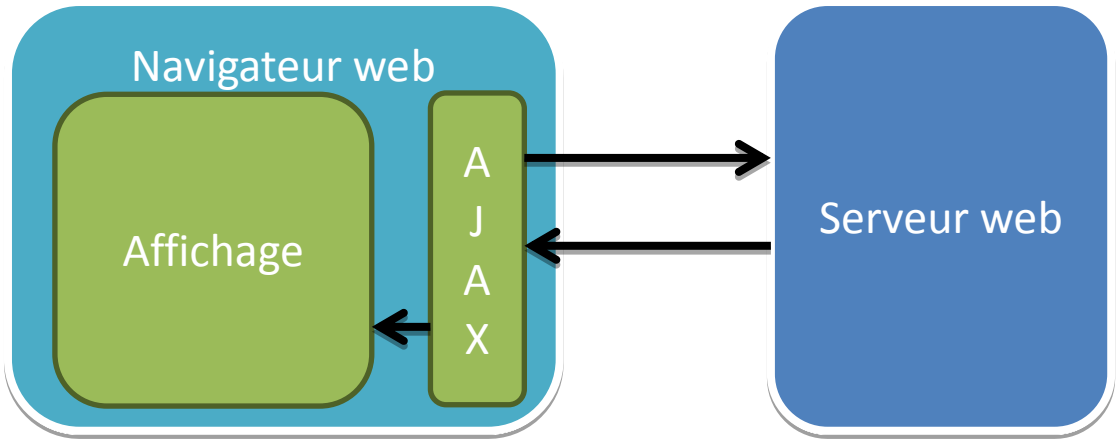
- **Shadowbox-3.0.3** : Il s'agit des fichiers de la bibliothèque ShadowBox.

- **webcam** : Ici on retrouve le fichier flash permettant d'afficher la webcam sur la page.

1. AJAX

AJAX, pour Asynchronous JavaScript And XML, permet notamment d'exécuter du code côté serveur sans pour autant avoir à recharger la page.

Dans une architecture « classique », lorsque l'on clique sur un lien pour demander une page, la page qui est demandé au serveur web est directement affichée dans le navigateur. La page courante est donc déchargée pour ensuite charger la page demandée. Cette demande est relativement lourde pour le navigateur et le rechargement est visible pour l'utilisateur, ce qui n'est pas très agréable.



Dans une architecture AJAX, le navigateur va faire une demande de page mais celle-ci ne va pas être relayé directement vers l'affichage. La réponse envoyé par le serveur va être analysée par du code JavaScript – exécuter dans le navigateur. Ainsi peut modifier uniquement certaines parties de la page et non recharger la page entière.

Ceci présente 2 avantages majeurs :

- Cela permet au serveur d'éviter d'envoyer des données qui ne changeront pas d'une page à une autre – comme les éléments de design par exemple. Ce qui évite donc de générer du trafic inutile, et cela évite aussi au navigateur d'avoir à traiter plus de données que nécessaire.

- Cette architecture permet aussi d'avoir un affichage plus fluide pour l'utilisateur en évitant le rechargement complet de la page. Avec ce genre de structure on s'approche donc plus d'un résultat que l'on pourrait obtenir avec une application de bureau.

C'est donc pour cela que nous avons utilisé AJAX dans notre application.

Conclusion

Un projet de DI4 prévoit actuellement l'ajout d'une visualisation en 3D des insectes afin de pouvoir répondre plus précisément aux questions posées.

Un point important évoquer, est le fait que la compatibilité avec l'application doit être assurée en cas de changement de structure du fichier XML. Ceci est loin d'être impossible sachant que l'application visant à générer le XML est en cours de développement.