Université Toulouse III – Paul sabatier
L2 Informatique

Antoine de Roquemaurel (antoine.de-roquemaurel@univ-tlse3.fr)
Groupe 2.2

Conception d'une application de folksonomies

Systèmes d'information et applications Web

Toulouse, le 2 mai 2013

Avant-propos

Ce dossier comporte les différentes de réalisation d'une application Web de folksonomies, site permettant de partager des URL et de les associer à des tages. Il à été conçut par Antoine de Roquemaurel dans le cadre du module *Systèmes d'Information et Application Web* de la L2 Informatique de l'université Toulouse III – Paul Sabatier.

Tester le projet

L'archive que vous avez reçus est organisée comme ceci :

scriptCreationBd.sql Contient le script de création de la base de données, celui-ci contient un jeu d'essai. Il est également possible de supprimer le schéma, et de le créer via l'application.

folksonomies/ Contient tous les fichiers du Site Web.

rapport.pdf Le présent rapport que vous êtes en train de lire

Afin de tester le projet, vous pouvez avoir accès au site web fonctionnel directement en ligne à l'adresse `http://dev.joohoo.fr/dev/folksonomies/`. Il est également possible d'utiliser notre code source avec votre propre serveur web et votre base de données, pour cela les paramétrage des accès à la base de données sont présent dans le fichier `folksonomies/database/connect.php`. Toutes les adresses web étant en relatifs, aucun problème ne devrait avoir lieu.

Ce site Web utilisant des fonctionnalités de HTML5 et CSS3, il est recommandé d'utiliser un navigateur récent. Ce site à été développé sous Google Chrome, ainsi l'affichage sera optimal sur ce navigateur, cependant il devrait s'afficher correctement sur les autres.

Il est possible de créer le schéma de la base de données grâce au menu Base de données → Création de la base.

Rédigé le 2 mai 2013 par Antoine de Roquemaurel.

Table des matières

Chapitre 1

Les besoins de l'application

Ce projet consiste en la création d'un site web utilisant les technologies HTML, CSS, PHP et la base de données MySQL permettant de gérer des tags sur des sites web. Ces tags permettent de retrouver facilement des URL, d'effectuer des recherches en fonction de tags, ...

Pour cela, un modèle de base de données nous à été fournis, mon application ayant été légèrement améliorée afin de pouvoir gérer une multitude d'utilisateurs, celui-ci à été légèrement modifié. Figure ? ? ? est présent le nouveau modèle.

Insertion d'un nouveau document Le site doit permettre d'ajouter un nouveau document, et de le lier à des tags. Si l'adresse du site existe déjà dans la base de données, celle-ci n'est pas ajoutée de nouveau, seuls les nouveaux tags sont ajoutés.

Lister les documents enregistrés sur le site Il doit être possible de lister tous les documents présent dans la base de données, ainsi que les tags associés. Il est également possible de savoir quels utilisateur ont enregistrés un document.

Afficher un nuage de tag L'application doit pouvoir afficher un « tag cloud » ou nuage de tag. Les différents tag s'affiche d'une taille différente et d'une couleur plus foncée en fonctions de leur importance, c'est-à-dire du nombre de document liés à ce tag.

Afficher tous les docuemnts liés à un tag donné Il doit être possible d'afficher tous les documents étant liés à un tag donné.

Rechercher un document en fonction d'un ou plusieurs tags On peut rechercher un document en fonction d'un ou plusieurs tags.

Inscription d'un nouvel utilisateur Il est également possible de s'inscrire afin d'utiliser le système, pour cela un pseudo, une adresse e-mail et un mot de passe sont demandés, les mots de passes sont cryptés dans la base de données. L'inscription d'un utilisateur lui permet ensuite d'ajouter de nouveaux documents, de leur donner une description personnaliser et de lister tous les documents lui appartenant.

Afficher les documents enregistrés par un utilisateur Enfin, un utilisateur peut afficher uniquement les documents lui appartenant.

Chapitre 2

Organisation du travail

Pour ce projet, j'étais seul, ainsi cela fut plus simples que lors du projet précédent où nous étions deux à travailler dessus.

Cependant, afin de bien organiser mon travail, de n'oublier aucune fonctionnalités et de ne rien perdre en cas de problème technique, j'ai choisis pour ce projet d'utiliser un outil de gestion de projet et un logiciel de versionnement de la même manière que le projet précédent.

2.1 Un outil de gestion de projet : Redmine

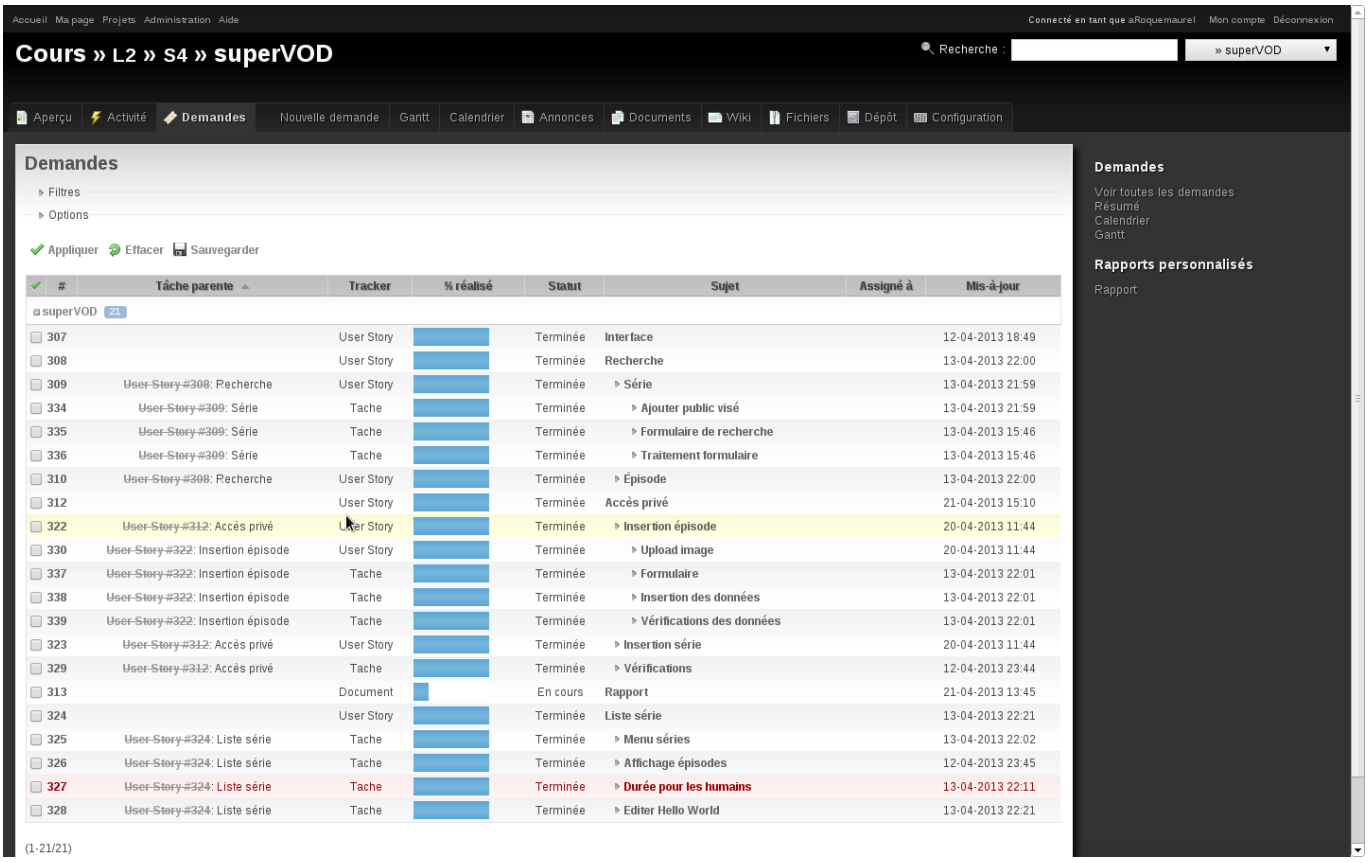

Figure 2.1 – Affichage des demandes dans Redmine

Pour le projet, j'ai utilisé *Redmine*, une plateforme web de gestion de projet (Cf figure 2.1). Elle m'a permis de simplifier le travail, et de ne rien oublier.

En effet, je pouvais créer des tâches, signaler qu'elles sont en cours/terminés/en tests, leur donner des dates limites, etc. . .

2.2 Un logiciel de versionnement : Git

Afin de limiter les problèmes liés à une perte de données, j'ai utilisé un logiciel de versionnement Git. Il a deux intérêt, tout d'abord, je pouvais travailler de plusieurs ordinateurs en parallèle sur le projet sans me soucier de fusionner le travail [1].

D'autre part, tous les logs étant enregistrés, je pouvais voir l'avancée du projet.

Enfin, toutes les modification sont stockées sur le serveur, ainsi en cas de problème, il est très facile de revenir à la version précédente ou même de comparer deux versions afin de voir les changements et de comprendre rapidement pourquoi une fonctionnalité à régressé.

1. À condition de ne pas travailler sur deux lignes de code identiques

Chapitre 3

Implémentation et conception

La conception du projet est l'étape la plus importante, ainsi j'ai préféré repenser à notre manière le projet, je suis partis sur une architecture respectant le patron de conception MVC [1], c'est à dire que toute la partie affichage (notamment le HTML), Modèle (c'est-à-dire les requêtes SQL) sont séparés, c'est le Contrôleur qui dirige le modèle et la vue, c'est ainsi le *chef d'orchestre*.

3.1 Mise en forme : les vues

Afin d'avoir un site web élégant sans passer beaucoup de temps à réinventer la roue en CSS, j'ai choisi d'utiliser un framework [2] CSS appelé *bootstrap*. Celui-ci nous permet d'avoir une mise en forme sobre très rapidement. La mise en forme est organisé comme suit :

3.1.1 Le CSS

Le CSS est présent dans le dossier `style/css`, dedans est présent d'une part le CSS fourni par *bootstrap*, auquel je n'ai pas touché, d'autre part, mon CSS qui surcharge certains affichage de *bootstrap*, celui-ci est dans `style/css/style.css`.

```
.menuSeries {
    padding-left: 30px; /* On ne colle pas le menu à gauche de l'écran */
}
#series {
    float: left; /* flottement lors de la recherche d'une série */
}

/* Taille/affichage du carousel */
.carousel h1 {
    color: #d5d5d5;
}
.carousel {
    height: 250px;
    width: 800px;
    background-color: black;
    margin: auto;
    margin-bottom: 15px;
}
```

Listing 3.1 – Exemple d'une partie du CSS

1. Modèles Vue Contrôleur ou *pattern MVC* : Model View Controller en anglais
2. Kit de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture). — D'après *wikipédia*

3.1.2 Le HTML

Le HTML correspond aux vues, tous les fichiers contenant du HTML sont dans le dossier `views/`, les formulaires eux sont dans le dossier `views/forms`.

Les vues peuvent ne comporter que du HTML, comme c'est le cas dans les formulaires, mais ils peuvent également comporter des instructions simples de PHP comme des boucles ou des conditions.

Le dossier `views` contient également les fonctions d'affichage des documents ou une fonction permettant d'afficher une alerte.

```php
<?php
/**
 * Affiche une liste de documents
 * @param $xaDocuments Un tableau contenant tous les documents
 * @para $desc Vrai si la description des documents doit être affiché
 * @param $pseudo Vrai si les pseudos doivent être affichézs
 */
function listDocuments($xaDocuments, $desc=false, $pseudo=true) {
  echo '
  <table class="table table-striped"> ';
  foreach($xaDocuments as $doc) {
    echo '<tr>';
    echo '<td><a href="'.$doc['url'].'">'.$doc['titre'].'</a>   ';
    echo' </td>';
    if($desc) {
      echo '<td>'.$doc['description'].'</td>';
    }
    echo '<td>';
    $doc['motCle'] = array_unique($doc['motCle']); // on s'assure que les mots clés sont uniques
    foreach($doc['motCle'] as $keyword) {
      echo '<a href="search.php?k='.$keyword.'"><span class="label label-info">'.$keyword.'</span></a> ';
    }
    if($pseudo) {
      $doc['utilisateur'] = array_unique($doc['utilisateur']);
      echo '<td>';
      foreach($doc['utilisateur'] as $user) {
        echo '<a href="search.php?u='.$user.'"><span class="label">'.$user.'</span></a> ' ;
      }
      echo  '</td>';
    }
    echo '</td>';
    echo '</tr>';
  }
  echo '</table>';
}
```

Listing 3.2 – Fonction d'affichage d'une liste de documents

3.2 Connexions à la base de données : Les modèles

L'application comporte trois modèles : un pour les documents, un pour les termes, et un pour les utilisateurs. Ces trois fichiers contiennent des fonctions effectuant des requêtes et retournant un tableau avec les attributs, ceci afin de bien séparer le SQL du PHP.

Également présent dans le dossier `database`, le fichier `connect.php` contient les identifiants et les instructions de

connexion à la base de données. Le fichier `database.php` quant à lui contient les requêtes de création et suppression de la base.

```
select document.numeroD, titre, url, motCle, description
from document

-- Jointures de l'intégralité du modèle
left join decrit on decrit.numeroD = document.numeroD
left join terme on terme.numeroT = decrit.numeroT
left join utilisateur on utilisateur.idUtilisateur = decrit.idUtilisateur
left join sauvegarde on sauvegarde.idUtilisateur = utilisateur.idUtilisateur
-- Selection de l'utilisateur
where utilisateur.idUtilisateur = &parametreIdUtilisateur
-- On joint également la table document
  and   document.numeroD = sauvegarde.numeroD
order by titre;
```

Listing 3.3 – Requête sélectionnant les documents d'un utilisateur donné

```
<?php
function insertDocument($xsTitle, $xsUrl, $xasKeywords, $xiIdUser, $xsDescription) {
  // si l'url n'existe pas dans la base, on l'insère
  if(!($urlFound = urlExist($xsUrl))) {
    $iIdDocument = uniqid(rand(), true);
    mysql_query("insert into document(numeroD, titre, url) values('$iIdDocument', '$xsTitle', '$xsUrl')") or die(mysql_error());
  } else { // sinon, on récupère son ID
    $iIdDocument = $urlFound->numeroD;
  }

  // maintenant on insère les mots clefs, s'ils n'existent pas déjà
  foreach($xasKeywords as $keyword) {
    if(!($keywordFound = keywordExist($keyword))) {
      $iIdTerme = uniqid(rand(), true);
      mysql_query("insert into terme(numeroT, motCle) values
        ('$iIdTerme', '".trim($keyword)."')") or die(mysql_error());
    } else {
      $iIdTerme = $keywordFound->numeroT;
    }

    if(!decritExist($iIdDocument, $iIdTerme)) {
      mysql_query("insert into decrit(numeroD, numeroT, idUtilisateur) values
        ('$iIdDocument', '$iIdTerme', '$xiIdUser')") or die(mysql_error());
    }

    if(!documentUserExist($iIdDocument, $xiIdUser)) {
      mysql_query("insert into sauvegarde(numeroD, idUtilisateur, description) values
        ('$iIdDocument', '$xiIdUser', '$xsDescription')") or die(mysql_error());
    }
  }
}
```

Listing 3.4 – Fonction d'insertion d'un document

3.3 Liaison des vues aux modèles : Les contrôleurs

Chaque contrôleur correspond aux pages que verra un utilisateur lambda dans l'URL, les contrôleurs sont en général assez simples, ils font appels au modèle puis font une inclusion de la vue, ceux sont eux qui feront les éventuels vérifications sur des paramètres, sur une connexion pour l'insertion de document etc...

```php
<?php
$page = 'Saisir un document';
require_once('database/connect.php');
require_once('database/db_document.php');

include_once('views/header.php');
include_once('views/documentInsert.php');

if(isset($_POST['titre']) && isset($_POST['url']) && ↩
    isset($_POST['keywords'])) {
  $aKeywords = explode(';', $_POST['keywords']);
  foreach($aKeywords as $keyword) {
    $keyword = trim($keyword);
  }
  insertDocument($_POST['titre'], $_POST['url'], $aKeywords, $_SESSION['id'], ↩
     $_POST['description']);
}
```

Listing 3.5 – Contrôleur de l'insertion d'un document

Comme dit plus haut, c'est relativement simple, on inclus les vues et fonctions nécessaires, on vérifie qu'on a le droit d'être ici, on créer le tableau de mots clés, puis on insère le tout e faisant appel au modèle.

Chapitre 4

Résultats obtenus

Dans cette partie, nous allons regrouper quelques captures d'écrans montrant les résultats obtenus sur notre application Web. L'application est répartie en 3 fonctionnalités :
- L'affichage des séries et épisodes présents
- La recherche de série ou épisodes
- L'ajout de série ou épisodes

L'ajout d'épisodes implique la possibilité de se connecter afin que seul l'administrateur puisse utiliser l'administration.

R Le couple login/mot de passe de l'application est admin/admin

4.1 Affichage des séries et épisodes

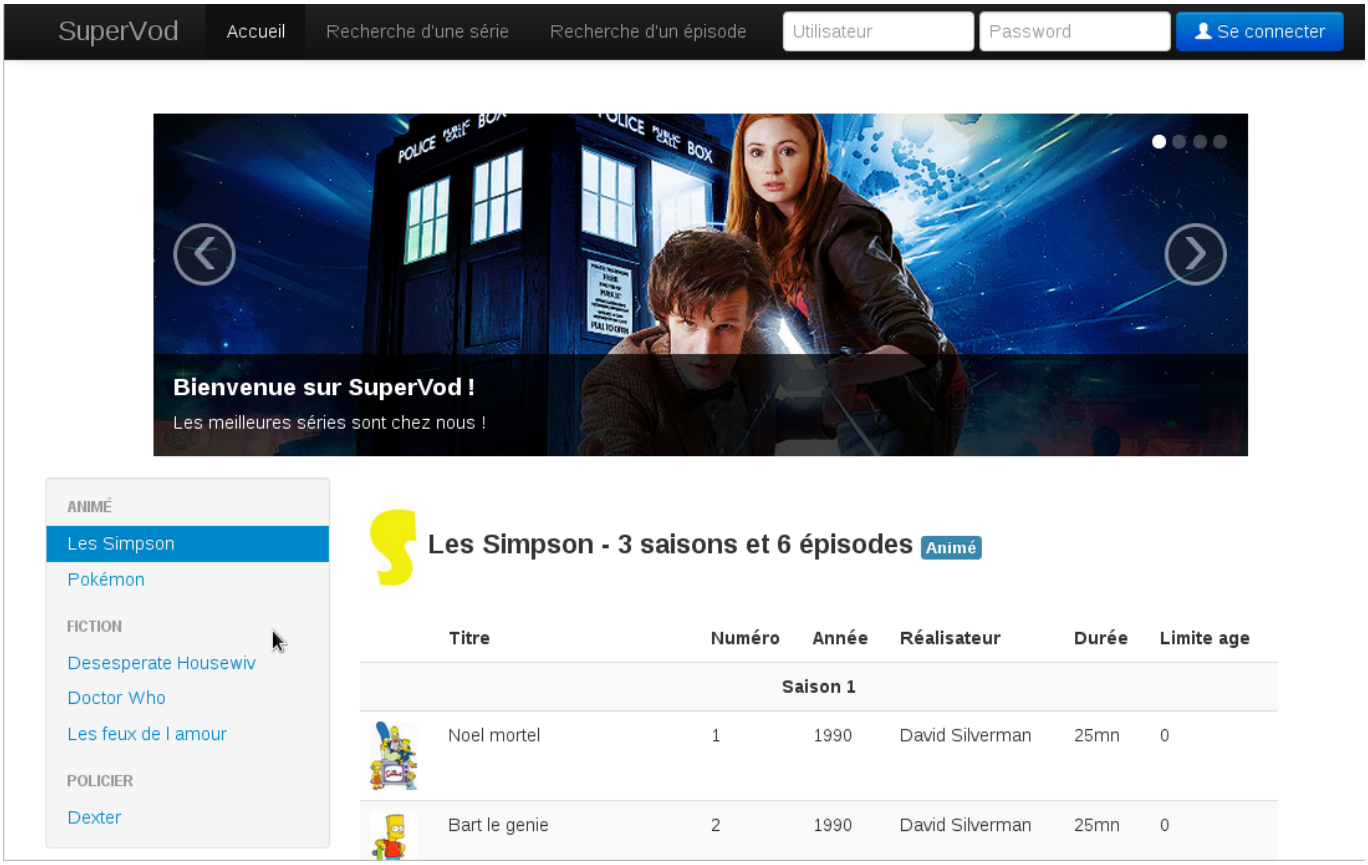

Figure 4.1 – Page d'accueil de l'application

La figure 4.1 nous montre la page d'accueil du site, avec la possibilité de sélectionner une série via le menu de gauche, affichant ainsi tous ses épisodes. On peut remarquer sur cette figure que le site possède un thème qui est commun à tout le site : la barre de navigation en haut de l'écran permet d'accéder d'un clique à chacune des fonctionnalités présentes sur le site.

4.2 Recherche

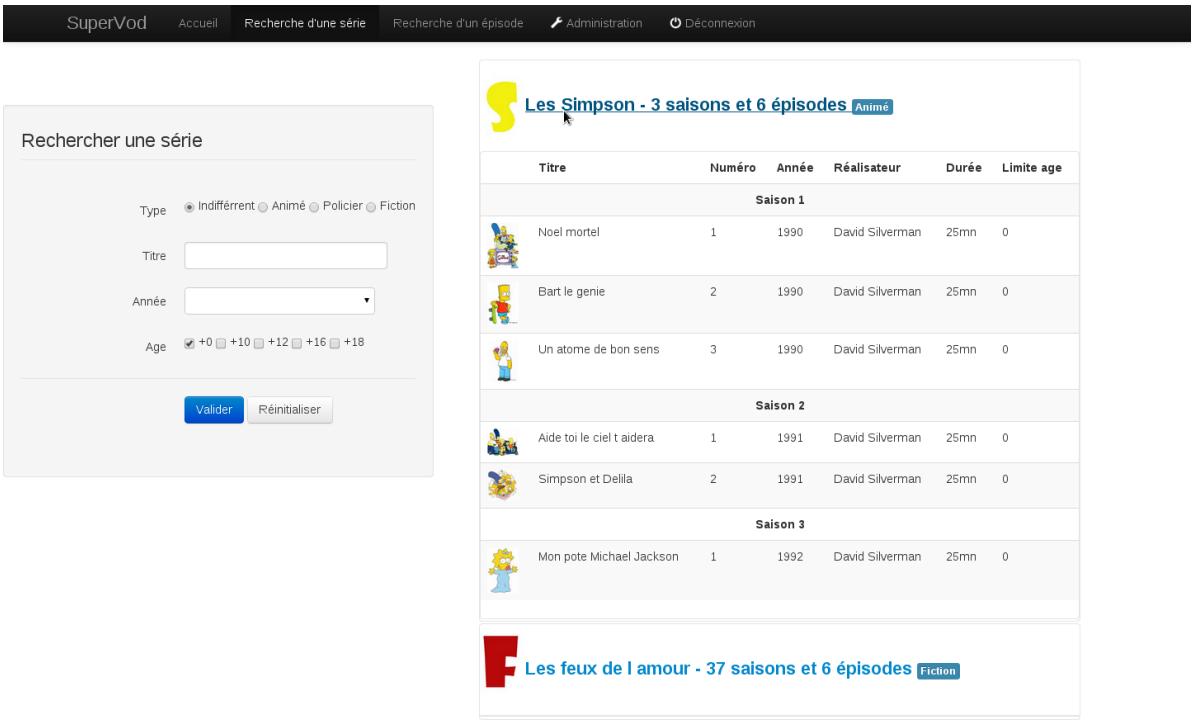

Figure 4.2 – Recherche d'une série

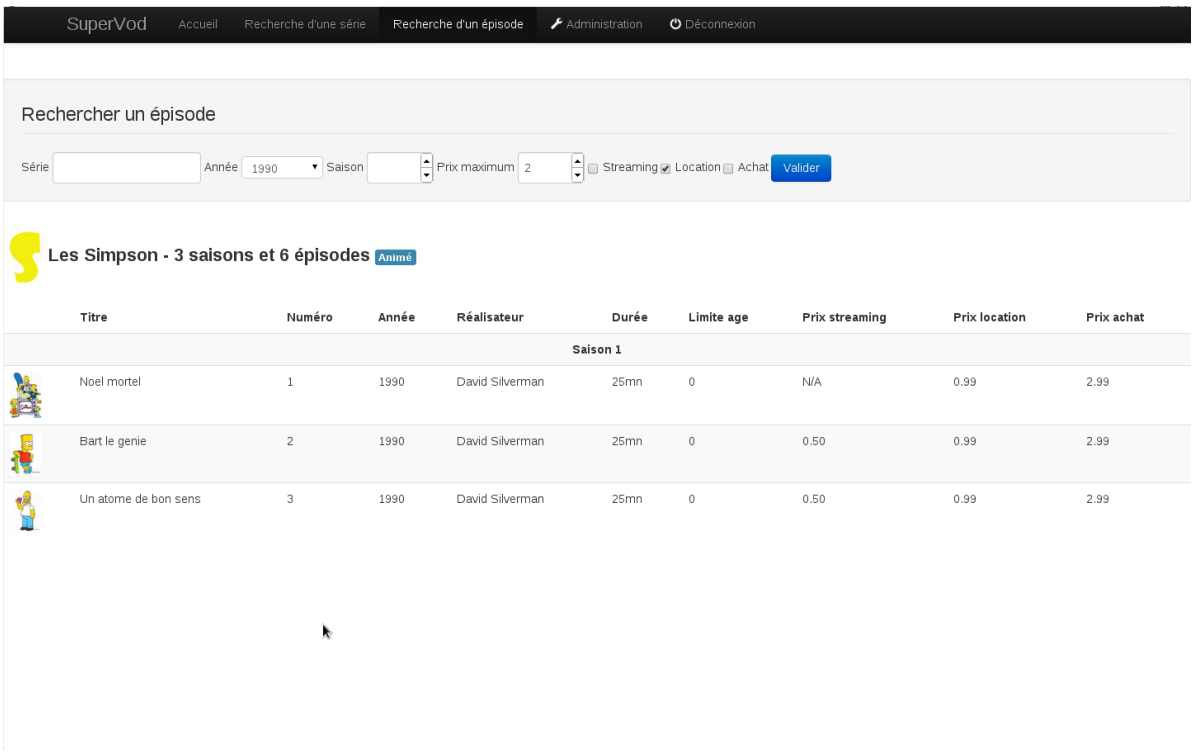

Figure 4.3 – Recherche d'un épisode

Les figures 4.2 et 4.3 nous montrent respectivement la recherche d'une série et la recherche d'un épisode.

Ces deux fonctionnalités sont assez similaires, bien que les critères de recherches soit différents. Ainsi, la recherche d'une série affiche les séries répondants au critères renseignés [1], ainsi que la possibilité d'affiche les épisodes correspondants de la série.

La recherche d'un épisode quand à elle à une mise en forme différente, étant donné qu'il y a plus d'informations a afficher pour un épisode, cependant le principe reste le même via les critères de recherche [2]

4.3 Ajout de séries et épisodes

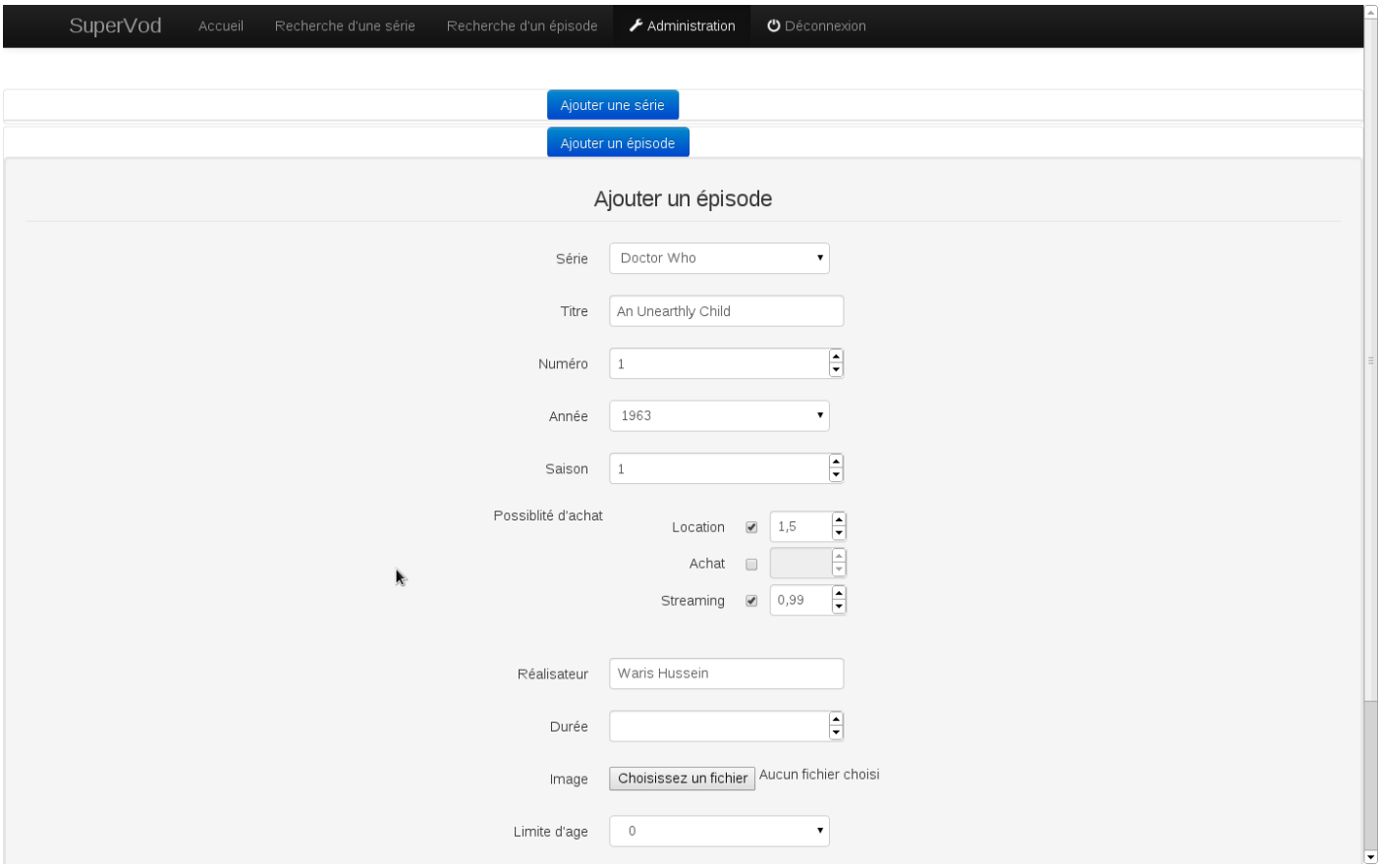

FIGURE 4.4 – Administration

La figure 4.4 nous montre la partie administration, ici l'ajout d'un nouvel épisode à la série *Doctor Who*, tous les champs présents précédemment peuvent être renseignés, cependant seuls les champs Série, Titre, Numéro et Saison sont obligatoires, les autres sont facultatifs.
Il est également possible de sélectionner une image pour la série depuis notre ordinateur via un champ d'upload.

Nous n'avons pas fait de capture d'écrans de l'ajout d'une série, cependant le principe est le même bien que les champs soient moins nombreux, pour une série, les champs Type et Nom sont obligatoires.

1. Une série peut être cherché par son type, son titre, ses années de diffusion ainsi que son age
2. Un épisode peut être trouvé grâce à sa série, son année de diffusion, sa saison, le prix maximum auquel on veut l'acheter, ainsi que sa disponibilité (Streaming, Location, Achat)

4.4 Valide HTML5

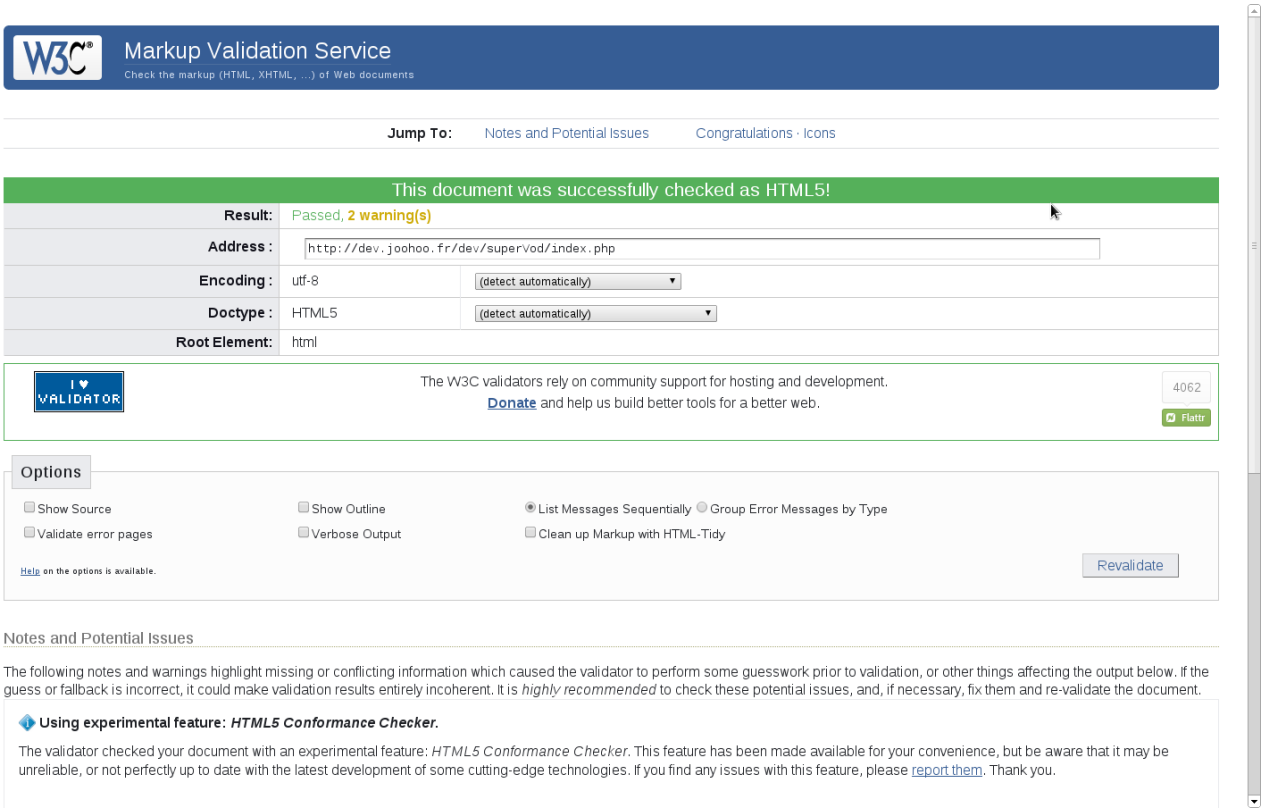

Figure 4.5 – HTML5 validator – Passed

La figure 4.5 montre simplement notre site web passant la validation du w3c [3] avec la norme HTML5.

3. World Wide Web Consortium

Annexe A

Table des figures

Annexe B

Liste des codes sources