SOULARD Alexandre & MINIER-MANCINI Titouan Cao

Compte-rendu de TP

Le premier sujet proposé dans le TP se concentrait sur l'utilisation de The Movie Database et la mise en place d'une page internet affichant les informations utiles sur le film demandé (entré en paramètre du lien url).

1) Exploration

Le film correspondant à l'adresse url fournie est le film "Fight Club". Le format de réponse est du JSON car cela permet ainsi de stocker des informations sur le film de façon structurée. Les données sont, grâce au format JSON, organisées dans un tableau et ainsi c'est plus facile d'y avoir accès. Le paramètre language=fr permet de traduire la page du film en français.

2) Exploration CLI

Voir le fichier correspondant au code php : phpcurl.php.

3) Page de détail (web)

Dans le fichier webinfo.php nous avons tout d'abord pris la page d'information du film dans la langue demandée (anglais par défaut) puis nous avons parcouru la page en regardant à chaque boucle si $key est égal au titre ou les autres paramètres demandés (tagline, description, …). Cela permet donc de lister les informations du film entré en paramètre. Enfin en dernier on renvoie un lien vers la page internet du film. Si aucun film n'est entré en paramètre, on a choisi comme film par défaut : Fight Club.

Sujet : Créer un agrégateur de flux RSS pour podcast

Le deuxième sujet proposé par le TP amenait à manipuler des objets XML traités par une librairie de feed tierce, afin de pouvoir proposer un affichage utilisable des données qu'ils contiennent. (voir le manuel d'utilisation à la fin)
Les points clefs sont:

- le traitement d'objets XML
- la gestion des données (POO fondamentale)
- la gestion des fichiers MP3
- la réflexion sur la manière d'implémenter un feed tel que celui de Twitter

En parallèle du sujet du TP ont été intéressants pour un apprentissage personnel:

- le développement d'une fonctionnalité de gestion des "feeds"
- la réflexion de solutions de sauvegarde courte sans base de données
- la réflexion d'un affichage efficace et semi-responsive (pas pour smartphone)
- la gestion des erreurs

I) Manipuler des fichiers RSS

La première étape consistait à afficher un tableau contenant les éléments fournis par un flux RSS. Pour cela il faut envisager les deux parties que seront le traitement de données en PHP côté serveur et la manipulation visuelle en HTML, CSS, JS… coté client.
Commençons par mettre en évidence les fonctions-clés du traitement des données:
Les informations à utiliser étaient:

- Le titre
- La date (de publication à priori)
- La description
- Le lien vers la page
- Le lien vers le fichier mp3
- La durée du fichier mp3

Afin de réaliser un traitement assez complet des données fournies par le flux RSS nous utilisons une première fonction *feed_class_podcast()* (comprendre feed comme nourrir et non pas fil ici) dont le rôle assez explicite est de veiller à la bonne construction des objets PHP à partir de l'objet XML fourni. Cette fonction est entièrement dédiée à la vérification des champs, des attributs de l'objet XML pour s'assurer qu'il convient au format requis pour un type podcast. Les champs demandés ne sont pas absolus, car comme nous avons pu le constater le format des objets fournis par les flux RSS de différents sites sont très peu normalisés.

Il semblerait qu'il y ait très peu de standards pour les nom des champs ainsi que pour leur contenu, ce qui peut rendre leur traitement assez difficile. Même en prenant en compte beaucoup de cas on ne peut jamais être totalement sûr que l'objet XML fournira les bonnes informations. Nous avons donc choisi de proposer plusieurs options pour notamment la durée, qui tantôt est dans la description tantôt dans un champs dédié, soit n'y est même pas du tout et il faut aller la chercher dans les attributs spécialisés de itunes. En revanche nous n'avons pas pu trouver de solution pour les liens vers la page qui ne correspondaient pas au format attendu, par exemple ceux fournis par le site "jeuxvideo.com" ne comprennent pas la racine de l'url et sont donc impossible à atteindre sans une correction complexe et hasardeuse avec le lien du mp3 (que nous n'avons pas mis en place).

II) Gestion des données et affichage

A l'issue de cette fonction nous obtenons des objets de la classe Podcast que nous avons créé. Cette classe contient les champs demandés par la question ainsi qu'un sous-objet Source pour attribuer une couleur aux Podcasts issus d'une même source, mais également pour effectuer des vérifications lors des opérations de gestion des sources justement. La classe Podcast propose ensuite des fonctions d'affichage: une pour l'affichage en ligne, une pour la ligne intercalaire des semaines, une autre pour les jours et enfin une pour le mode compact.

Finalement ce sont deux fonctions *display_row()* et *display_compact()* qui s'occupent d'afficher les deux tableaux en faisant appel aux fonctions des Podcasts. Ces fonctions s'occupent également de restreindre le nombre de podcasts à afficher selon la demande, de gérer l'affichage des intercalaires via des fonctions secondaires de calcul sur les dates (affichage ligne), et de gérer l'affichage dans le calendrier (compact). Ce dernier demandait un peu plus de réflexion car il n'est pas possible de parcourir un calendrier avec une boucle pour la simple raison qu'un calendrier classique se lit par ordre chronologique par semaine et anti chronologique entre les semaines. Plutôt que de chercher une solution algorithmique nous avons décidé d'utiliser d'autres objets: les Week et les Day. Un Day contient plusieurs évènements (Podcasts dans notre cas) et peut les afficher, une Week contient 7 Day et peut faire un affichage en ligne de ces Day. Il ne restait alors plus qu'à gérer la création de ces Week et à déterminer quand est-ce que l'on change de semaine.
Note: Nous aurions pu également créer la classe Calendar qui se serait occupée de tout cela pour alléger le code dans la fonction d'affichage.

Avec les éléments cités ci-dessus nous obtenons les affichages demandés pour une source de Podcast. Mais bien sûr pour créer un agrégateur de podcast il nous faut pouvoir afficher plusieurs sources à la fois. C'est pourquoi nous avons également la fonction *unify_resource()* qui utilise un tableau d'objets Sources pour créer un tableau d'objets Podcast triés selon leur date (via la fonction usort).

En combinant des podcasts de plusieurs sources on se confronte au problème précédemment évoqué de la normalisation des attributs des objets XML. En effet par exemple l'attribut "description" peut prendre diverses formes, certains sites mettent même du html dedans. De plus sa taille peut grandement varier, ce qui peut gêner l'affichage compact. C'est également dérangeant pour l'attribut html title qui a du mal à gérer ça, c'est pourquoi nous avons préféré utiliser notre propre tooltip.

III) Les fichiers MP3

Le format des mp3 des podcasts de La méthode scientifique ont un bitrate de 128 kbps et sont en stéréo. En réduisant avec Lame le bitrate à 32 kbps et en passant en mono on réduit le poids du mp3 originellement de 54 Mo à 14Mo.

Dans le html:
Nous avons remarqué que l'attribut download ne fonctionne pas comme attendu sous Firefox, qui se contente d'ouvrir le contenu dans un autre onglet. Malgré beaucoup de recherches nous ne sommes pas parvenus à résoudre ce problème.
Afin que la page ne mette pas plusieurs minutes (voire heures) à charger quand on affiche des milliers de podcasts nous avons décidé de lancer le chargement des players uniquement au lancement de ceux-ci.
Pour l'affichage compact, les balises audio n'ayant pas de contrôles, elles sont invisibles, mais bien existantes dans la page. On se sert d'un petit bouton play/pause et d'un peu de javascript pour les lancer (mais sans possibilité de manipulation du volume entre autre…).

IV) S'intéresser au fil Twitter

Au travers de la page de La méthode numérique nous avons pu retrouver le lien vers la page Twitter de leur compte mais pas ceux de chaque podcast. Automatiquement il faudrait chercher la balise de classe twitter-timeline et récupérer la valeur de son attribut href (voir la fonction *get_twitter()* qui n'est pas utilisée), mais ça n'est pas du tout générique.

Comme les liens des fils twitter spécifiques ne sont pas fournis par le flux RSS nous avions pensé que peut-être nous pourrions les retrouver avec les bons mot-clés, cependant Twitter semble générer ses url un peu aléatoirement, ce qui rend cette idée impossible.
L'API Twitter n'avait pas réellement sa place dans cet agrégateur de flux RSS. Cela dit nous l'avions déjà utilisé par le passé via un plugin wordpress pour afficher un fil d'actualité de notre choix. Il pourra être intéressant de la réutiliser pour un futur site (en écrivant soi-même le code de ce plugin).

A partir d'ici les fonctions dont il est fait mention ne sont pas directement liées au sujet initial du TP.

Bonus - Gérer les flux RSS

Pour pouvoir proposer une expérience un peu plus vraie d'un agrégateur de podcast il nous fallait pouvoir gérer les flux RSS directement dans la page. C'est là qu'intervient le petit formulaire discret et un peu brut en bas à gauche de l'écran. Ce formulaire fonctionne en méthode post, et les variables qu'ils renvoient sont captées par la page pour être évaluées. Cette évaluation détermine s'il faut créer une nouvelle source, en supprimer une ou encore la modifier. C'est sur la différenciation entres ces possibilités que le formulaire est "brut", car trois formulaires auraient été bien plus clairs. Cependant nous ne voulions pas non plus rendre la page trop complexe. Le fonctionnement est expliqué sur le site lui-même mais globalement le nom sert d'identifiant pour modifier, l'url d'identifiant pour supprimer et si rien ne correspond on se tourne vers la création.

Ensuite il nous fallait réfléchir à un moyen de conserver en mémoire les données des sources, afin qu'un utilisateur puisse conserver les flux qu'ils consulte sans avoir à les inscrire à chaque fois. Pour une solution rapide il y aurait eu les cookies, mais les cookies sont quelque chose qu'on a tendance à supprimer de temps en temps, or on ne veut jamais oublier les flux enregistrés, ce n'est pas une méthode de stockage idéale. On aurait pu penser également à une base de données bien sûr, mais pour une exécution sur un serveur local l'intérêt est discutable. Si l'on travaillait sur une application distribuée bien sûr, mais dans le cadre de travail actuel nous avons jugé la bdd inutile.
Finalement nous avons opté pour une solution employant des fichiers sur le disque. L'idée est d'avoir un fichier php contenant une variable locale que l'on pourra utiliser ailleurs. Cette variable contiendra un tableau de sources prêt à l'utilisation par nos fonctions de traitement. Ce fichier php est simplement intégré par un require_once, on peut donc discuter de la viabilité de cette manipulation: on melange code et données ce qui fonctionne ici mais n'est pas du tout idéal.

Lorsque le formulaire précédemment cité appelle nos fonctions de gestion de source c'est ce fichier qui va être réécrit pour modifier son contenu.
Une solution un peu plus élégante aurait été d'avoir un fichier texte brut (ou en xml même) et de le traiter pour obtenir un tableau que l'on mettrait dans notre variable ensuite, mais non n'avons pas eu le temps de mettre cela en place.
De plus ce mode de fonctionnement implique de prendre en compte beaucoup d'exceptions qui peuvent survenir à cause de la gestion des fichiers.

Bonus - Gestion des erreurs

En parlant de la gestion des fichiers, on peut également dire que cela amène beaucoup de cas à gérer autour de ce tableau de sources. Il y déjà bon nombre d'erreurs qui peuvent avoir lieu à cause de manque de permissions, et malheureusement (ou heureusement) le navigateur (www-data) n'a par défaut pas les droits requis pour écrire sur le disque dans un dossier lambda (ni se donner les droits de le faire forcément). Mais ce n'est pas tout, la lecture et l'écriture dans le fichier de sauvegarde (feeds.php) peut causer beaucoup d'erreurs si le contenu ne convient pas. Et il en reste d'ailleurs qui n'ont pas été prises en charge.

Dans le reste du code et notamment pour la vérification des objets xml nous avons essayé un maximum de prendre en charge les possibles erreurs et gérer les exceptions. Malgré tout il est difficile de dire que nous avons tout pris en compte. Celles prises en compte sont toutefois assez bien expliqué et affichées normalement.

Manuel d'utilisation

Bref manuel d'utilisation (complémentaire au README et à la page d'explication):
La page est composée de six éléments.
Deux sont les deux affichages demandés, ils prennent toute la place visible.
Le volet droit se déplie quand la souris passe dessus à la limite de la fenêtre (mais avant la barre de scroll).
Le bouton "switch display" permet d'afficher l'autre affichage. Le bouton avec la flèche en bas de ce volet remonte en haut de l'affichage actuel (pas de l'autre).
Le volet en haut à gauche permet de délimiter l'affichage des podcasts (modifier l'url marche aussi).
Le volet en bas à gauche permet de gérer les sources (voir la section Bonus correspondante).
Enfin le volet à gauche affiche toutes les sources avec leur couleur pour se repérer dans l'affichage compact (calendrier).

Dans l'affichage compact les podcasts sont encore plus compactes que demandé dans le TP, pour observer l'affichage demandé il faut passer la souris sur l'un des entêtes. Le bouton play/pause fonctionne pour lancer l'audio. Au survol du titre un tooltip s'affiche avec la description dedans. Le lien vers le Twitter étant lié à la source et non pas au podcast, (cf. discussion sur le fil Twitter) il est dans le volet gauche et non pas dans le calendrier. Les semaines au cours desquelles il n'y a aucun podcast sont représentées par une ligne pleine grisée.
Dans l'affichage en ligne tout est plus ou moins comme demandé. Il y a en plus des intercalaires pour les jours car la page s'est davantage développée pour recevoir beaucoup de flux RSS et donc beaucoup de podcasts.