Alex Bulla – Michael Gumowski UniGE – Printemps 2011

Nouvelles Technologies de l'Information et de la Communication

Refonte de l'Hyperlivre

Sommaire

1 Introduction

Lors du premier semestre du cours de Nouvelles Technologies de l'Information et de la Communication (NTIC), nous avons été invités à participer à l'écriture d'articles pour l'Hyperlivre. Cet outil permet à des utilisateurs de contribuer à l'élaboration de textes via l'écriture de courtes notes. Ces notes représentant des sources de connaissances dont la nature peut varier. Ce n'est toutefois pas un blog, ni un CMS[1] et encore moins une application d'édition partagée. Quoiqu'il en soit, il emprunte des notions communes à ces plateformes, et peut être en un certain sens vu comme un blog collaboratif allégé ou un wiki.

Les sources de connaissances, appelées notes, doivent rester concises et précises. Elles peuvent contenir des informations variées (extrait de code sources, images, …), fournir des exemples réels et concrets ou explorer des concepts généraux. Les multiples contributions des utilisateurs doivent être modérées, afin de garantir leur pertinence et maintenir la cohérence de l'ensemble. Pour ce faire, les notes de l'hyperlivre peuvent être regroupées selon des concepts, créés à l'avance par l'administrateur.

L'Hyperlivre repose à l'heure actuelle sur la technologie Lazy, développée au sein de la Division Informatique de l'UNIGE. Initialement le système Lazy permet de gérer plusieurs projets, dont l'hyperlivre. Ce dernier utilise le système de gestion des utilisateurs de Lazy, ainsi qu'une base de données dédiée à l'application. Le code source de l'hyperlivre est piloté depuis Lazy. Actuellement, créer plusieurs instances d'hyperlivre revient à créer plusieurs projets dans Lazy.

Quoiqu'il en soit, l'intérêt de cet outil collaboratif doit résider dans sa simplicité d'utilisation. Pour ce projet de semestre, nous avons proposé une refonte l'outil Hyperlivre. Afin d'en simplifier son utilisation et de le rendre plus accessible, il est nécessaire de repenser certaines de ses fonctionnalités. En particulier, l'effort a été concentré sur l'élaboration d'une structure autonome (séparée de Lazy), dont certaines spécifications non fonctionnelles seront améliorées. La maintenance, la portabilité du code ainsi que sa modularité ont été au centre de nos préoccupations. De plus, l'interface doit être rendue plus conviviale et intuitive (meilleure accessibilité et ergonomie).

Le nouvel Hyperlivre cible deux catégories d'utilisateurs. Les premiers étant de simples utilisateurs du système, auteurs de notes. Les seconds sont les administrateurs. Pour ces derniers, la distribution et la mise en place de l'hyperlivre doit être simplifiée. Ils doivent également pouvoir gérer les utilisateurs et créer de nouvelles instances d'hyperlivres. Un type d'utilisateur intermédiaire, les modérateurs, possèdent des droits augmentés vis-à-vis d'utilisateurs classiques, sans pour autant avoir un contrôle complet sur l'hyperlivre.

[1] *Content management system*, un logiciel fournissant un système de gestion de contenu.

2 Cas d'utilisation

Pour ce projet, nous avons identifié les cas d'utilisations résumés dans le schéma UML de la Figure 2.1. Afin d'indiquer les fonctionnalités implémentées, nous utilisons trois couleurs. Ainsi les cas d'utilisations dont le fond est vert ont été intégralement implémentés. Ceux sur fond jaune ont été partiellement implémentés et intégrés. Ceux dont le fond est blanc ont été abandonnés.

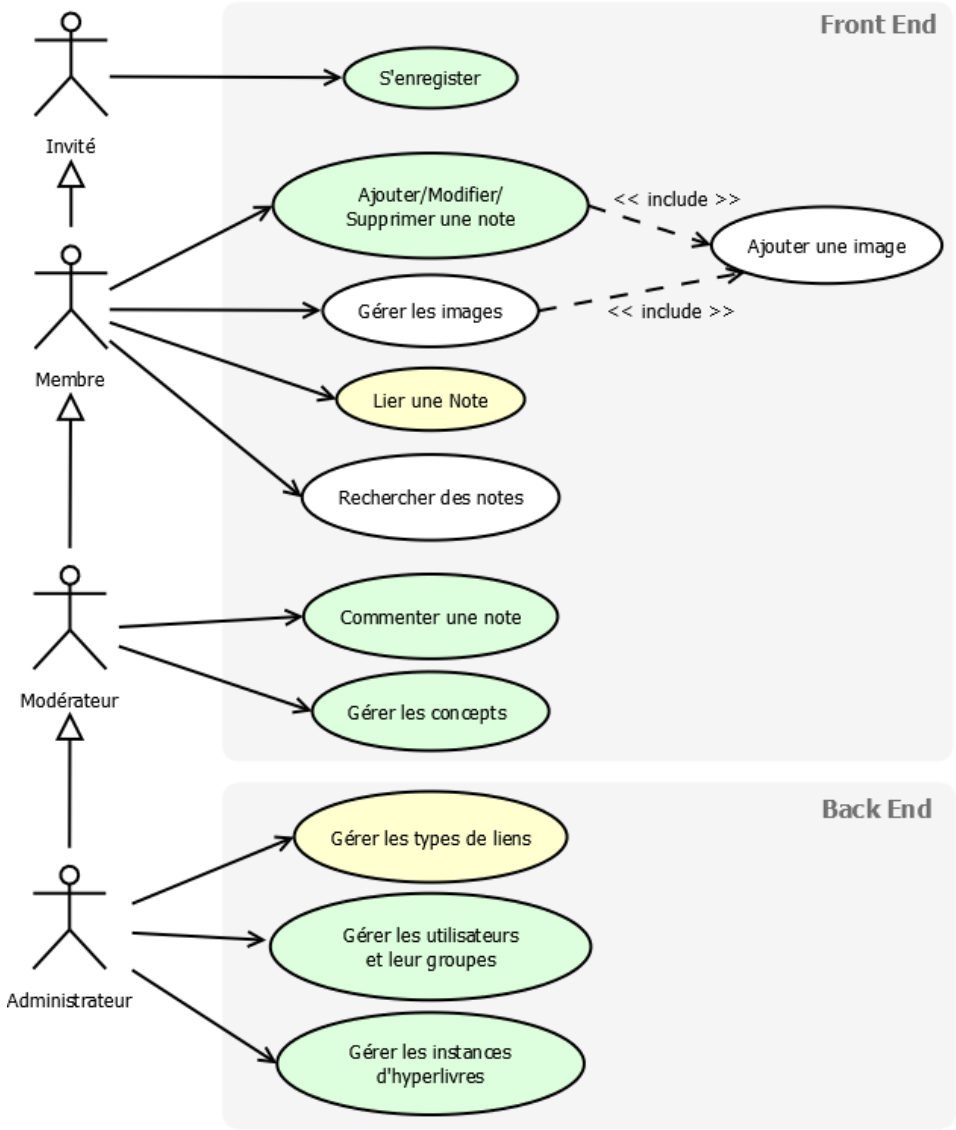

Figure 2.1 : Schéma des cas d'utilisation, décrivant les options possibles en fonctions des rôles des utilisateurs.

Pour des raisons de temps, nous n'avons implémenté qu'une partie de la gestion des liens conceptuels. La structure relationnelle (base de données) est entièrement fonctionnelle. Le contrôleur responsable de la création des liens a été réalisé. Cependant, l'intégration avec le système d'authentification n'est pas complète. Cette fonctionnalité a donc été partiellement fournie avec la dernière version de notre projet. A l'heure actuelle, il est donc possible pour tout utilisateur (authentifié et possédant les droits adéquats) de créer des liens. Toutefois ces liens peuvent s'exercer entre tout type d'objet (livre à livre, livre à concept, livre à notes, …), ce qui n'est pas souhaitable.

Cette fonctionnalité s'inspire des liens de type RDF. Un triplet est ainsi constitué d'un sujet, d'une propriété puis d'un objet. Pour une éventuelle future version du projet, il conviendra de limiter la portée et l'utilisation de ces entités par l'utilisateur basique. Par exemple en permettant uniquement la création de liens de type note-à-note et note-à-concept. Nous estimons que cette fonctionnalité est actuellement achevée à 80%.

En ce qui concerne les fonctionnalités non implémentées (gestions d'images et recherches de notes), nous avons décidé de suspendre leur réalisation, par manque de temps.

3 Plan du site web

La Figure 3.1 regroupe la plupart des cas de navigation classique possible sur le site. Le code de couleur sépare les fonctionnalités d'administration (Back-end) de celles offertes à l'utilisateur classique (Front-end).

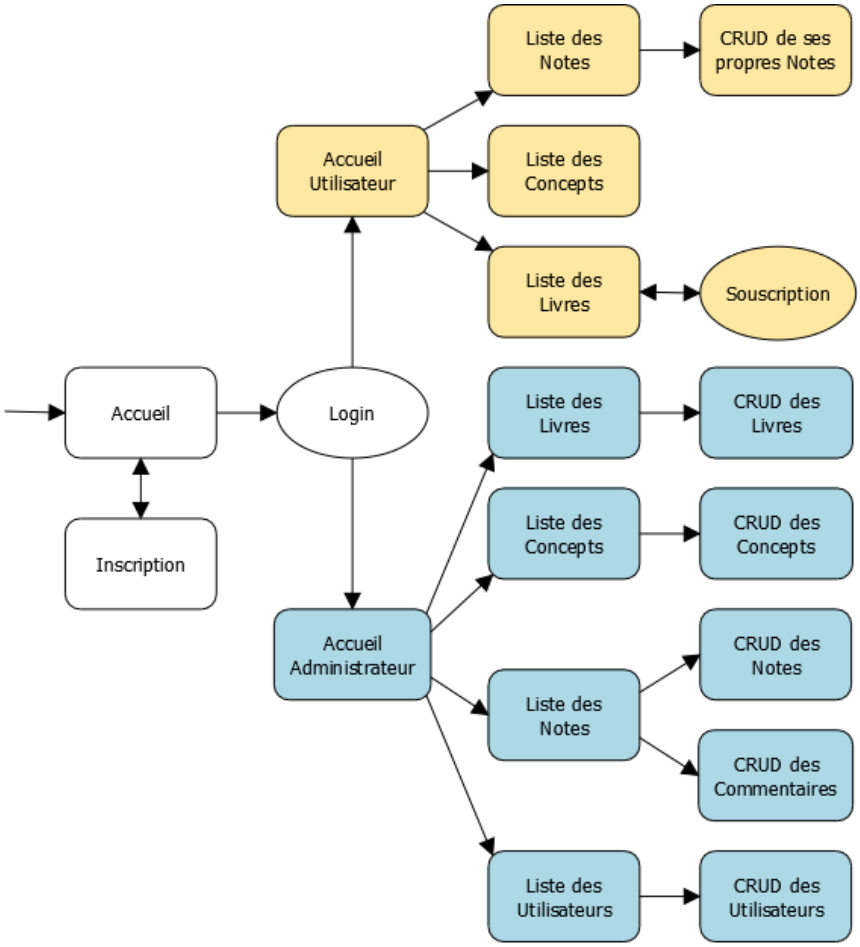

Figure 3.1 : Schéma des différentes options de navigations possibles sur le site web. Les options de l'administrateur sont indiquées en bleu, celles de l'utilisateur en saumon. Celles qui sont en blanc sont accessibles aux visiteurs non-authentifiés. Les rectangles représentent des pages web et les ellipsoïdes sont liés à des actions. Une page indiquée comme « CRUD » regroupe en réalité plusieurs vues et actions, relatives à la manipulation des objets.

4 Outils et Framework

Le choix des outils pour l'implémentation de l'hyperlivre a été réalisé suite à une recherche des meilleurs candidats possibles. Ainsi, pour implémenter la logique applicative, nous avons décidé d'employer le langage de script PHP. Celui-ci est largement employé et testés au sein des applications web.

Le schéma de la base de données est réalisé à l'aide de MySQL. Cette plateforme présente l'avantage d'être largement disponible sur les serveurs, notamment au sein de l'université.

L'utilisation d'autres outils a été envisagée à plusieurs reprises pendant la phase de planification du projet. Certains ont été abandonnés car ils se sont révélés inadapté à nos exigences. En particulier ceux-ci ont présenté des faiblesses en ce qui concerne des critères d'utilisabilité, de maintenance ou de facilité d'apprentissage.

4.1 PHP

PHP est un langage de script open-source, exécuté côté serveur. Sa syntaxe est empruntée aux langages C, Java et Perl, et est facile à apprendre. Le but de ce langage est entre-autres de permettre aux développeurs web d'écrire des pages dynamiques rapidement. Il est largement employé à travers le web. Il est donc facile d'obtenir des cas d'utilisations variés et concrets. D'autre part, ses fonctionnalités ont été testées et améliorées par une communauté grandissante de développeurs.

Dans le cadre de ce projet, nous avons utilisé PHP dans sa version 5.3.x. Pour des raisons de fonctionnalité, il n'a pas été possible de rester compatible avec les versions antérieures de ce langage. En contreparties, il nous est possible de profiter des dernières avancées technologiques du langage. Cette version est vivement recommandée par php.net à tous les hébergeurs de site web.

Pour des questions de robustesse et de facilité, il est d'usage courant d'employer un framework lorsqu'on travaille en PHP. Parmi la multitude de framework disponible, seuls certains se démarquent par leur efficacité. Nous avons opté pour un framework léger et rapide, qui nous permettait une grande flexibilité d'utilisation.

4.1.1 FuelPHP

FuelPHP[2] est un framework essentiellement écrit à l'aide de, et pour PHP. Il est simple d'utilisation, léger et rapide. De plus il est sous License libre (MIT), et bénéficie ainsi de l'expérience de la communauté de développeurs open source.

Les concepteurs du framework s'inspirent de leurs diverses expériences afin de développer un outil cohérent et efficace. Ils ne gardent que les meilleurs aspects des applications similaires et se débarrassent du superflu. La force de leur architecture réside dans la possibilité d'augmenter le framework à l'aide de modules indépendants. Ceux-ci sont produits par des indépendants et peuvent être chargés à la demande. Ainsi, l'intégrité du noyau de Fuel n'est pas remise en question.

[2] http://fuelphp.com/

En revanche, ce framework est très jeune (actuellement en ReleaseCandidate 2.1) et sa documentation reste incomplète. Notre brève expérience sur des frameworks similaires (*CodeIgniter*) nous a cependant permis une prise en mains relativement rapide. De plus, la jeunesse du framework apporte également une forte réactivité de la part de ses développeurs principaux. La plupart de nos questions ont donc trouvé une réponse adéquate dans de courts délais.

Un des atouts de Fuel est de permettre des appels statiques à travers l'application. Ainsi, les objets et les librairies sont chargés à la demande grâce au mécanisme de chargement automatique des nouvelles versions de php (*autoloading*). Contrairement à d'autres frameworks, le contexte applicatif peut être accédé depuis n'importe quel emplacement, ce qui augmente encore la flexibilité.

4.2 MySQL

La base de données est utilisée pour stocker les informations nécessaires au bon fonctionnement de l'Hyperlivre. Du à sa grande popularité, nous avons opté pour l'utilisation d'une base de donnée MySQL. FuelPHP peut cependant gérer d'autres types de bases.

Nous recommandons l'utilisation conjointe d'un gestionnaire de base de données, comme PHPMyAdmin. Ce dernier est open source et souvent déjà disponible sur les serveurs d'hébergement. L'ORM de Fuel nous autorise à créer des liens entre les objets PHP et les tables de la base (*object relational mapping*).

5 Principes et architecture du projet

A l'heure actuelle, il existe une tendance dans le développement des applications web dynamiques visant à les rendre plus modulables. En effet, on constate fréquemment que les mêmes portions de code peuvent être réutilisées dans différentes parties du programme. En ce qui concerne PHP, l'approche orientée objet qui se démocratise, permet entre-autres de palier à ce type d'exigences. Conjointement, il est souvent nécessaire d'appliquer une stratégie de développement formalisée et éprouvée (design pattern). Le génie logiciel permet alors d'apporter une réponse adaptée à la plupart des problèmes auxquels le développeur web est confronté.

Planifier à l'avance l'architecture d'un site, s'avère donc vital pour sa survie. En effet, on peut considérer qu'une application maintenable est potentiellement plus riche, et survivra plus longtemps dans un environnement logiciel sans cesse remanié. Le web est un univers tournoyant, dont les innovations, les standards et les us évoluent rapidement. Il est donc d'autant plus critique de doter les applications web d'une architecture robuste, qui à défaut d'être toujours d'actualité pourra cependant facilement s'adapter aux bouleversements divers.

Le modèle Modèle-Vue-Contrôleur (MVC) a pour but d'améliorer la portabilité, la modularité et la réutilisabilité du code. Le fait que FuelPHP implémente, respecte et met en pratique l'utilisation du pattern MVC, explique notre choix décisif en faveur de ce framework.

Il est parfois nécessaire de revoir et améliorer une fraction de l'application. Ces refontes peuvent malheureusement s'accompagner de difficultés notoires (et de maux de têtes abominables), lorsque les interdépendances au sein du code sont importantes notamment. C'est par ailleurs le problème

qu'il nous a été donné de résoudre sur l'Hyperlivre. Le design MVC rend cette tâche moins fastidieuse. En séparant les divers composants applicatifs en fonction de leurs rôles au sein de l'application, il est possible de réduire considérablement les interdépendances au sein du code.

L'adoption d'une telle architecture réduit le temps nécessaire aux futures mises à jour de l'Hyperlivre et l'autorise à disposer de fonctionnalités réutilisables. FuelPHP va au delà, en proposant le modèle HMVC, ou MVC hiérarchique, dans lequel chaque contrôleur est à la tête d'un appel particulier et coordonne une triade (triplet MVC). Ainsi, chacun de nos contrôleurs peuvent potentiellement appeler d'autres modules indépendants, récupérer leurs résultats puis reprendre l'exécution de leur programme.

La Figure 5.1 montre un exemple d'appel hiérarchique au sein d'une architecture HMVC. Des requêtes peuvent être envoyées aux différents contrôleurs de l'application, qui dirigent ensuite leur triade.

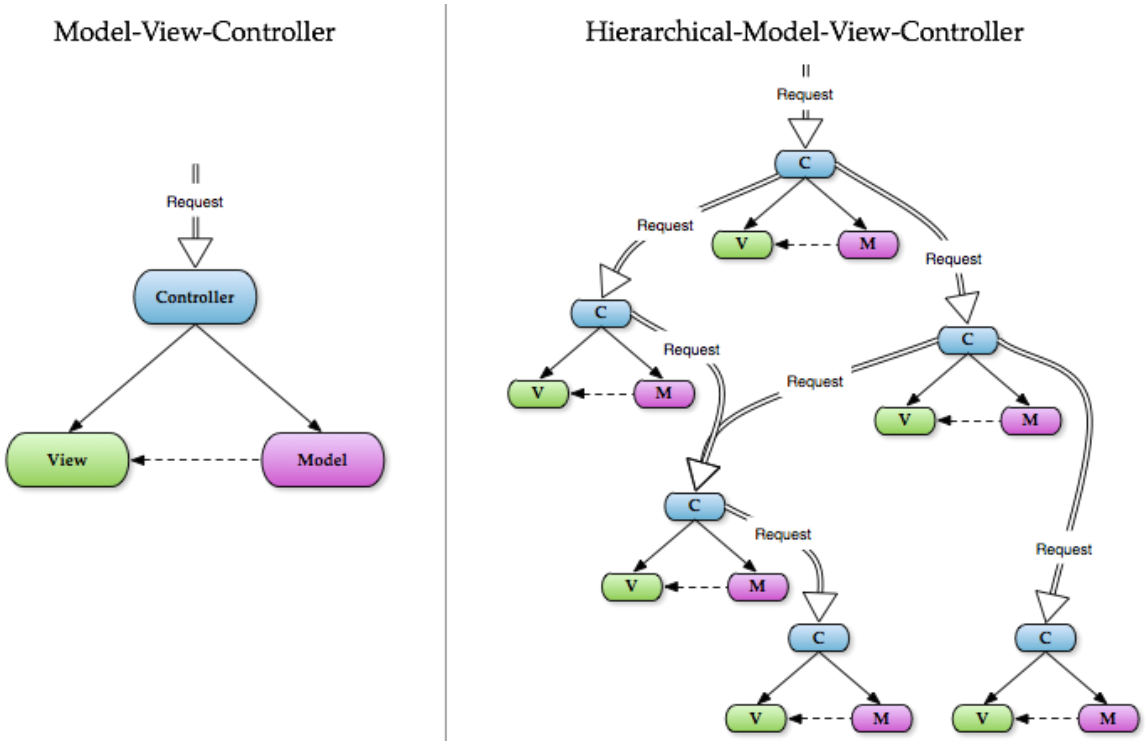

Figure 5.1 : Illustration[3] du modèle HMVC avec le parcours d'une requête dans un modèle MVC classique et dans un modèle MVC Hiérarchique.

On distique ainsi trois rôles au sein des objets de l'Hyperlivre. Les tâches assignées à chacun d'entre eux sont détaillées comme suit :

- Modèle : Il décrit ou contient l'ensemble des données propres à l'entité. Le modèle offre des méthodes pour accéder à ces données (insertion, suppression, mise à jour). Il offre aussi des méthodes pour récupérer ces données (DAO). Les résultats renvoyés par le modèle sont dénués de toute présentation.

[3] Source : https://github.com/WebDevilAZ/DesertCodeCampPHPMVC/wiki/Helpful-graphics

- Vue : La vue correspond à l'interface avec laquelle l'utilisateur interagit. Sa première tâche est de présenter les résultats renvoyés par le modèle. Sa seconde tâche est de recevoir toutes les actions de l'utilisateur (clic de souris, sélection d'une entrée, boutons, etc.). Ces différents événements sont envoyés au contrôleur. La vue n'effectue aucun traitement.
- Contrôleur : Le contrôleur est le point central d'une triade MVC. Il prend en charge la gestion des événements et traite les actions de l'utilisateur. La logique des services est implémentée dans ce type d'objet. Le contrôleur pilote la vue et provoque les modifications sur les modèles.

De plus, on relie une entité du domaine à chacune des triades ci-dessus. Les notes, les livres ou les utilisateurs sont donc tous représentés et fractionnés en triplets selon ce principe.

Pour s'assurer de l'intégrité des données, on sépare d'un modèle la logique qui permet sa validation depuis la vue. Il s'agit toutefois d'objet très fortement couplé à la représentation abstraite du modèle.

5.1 Présentation de l'architecture de l'Hyperlivre

Dans Fuel, les contrôleurs et les modèles sont implémentés en tant que classes PHP. On passe par une instance du contrôleur lorsque l'on désire accéder à une ressource particulière. L'URL est récupérée et routée selon le motif suivant :

```
http://nom.du.site.ch/public/index.php/controleur/methode/param1/param2/...
```

5.1.1 Contrôleurs

Le contrôleur **`index.php`** est le point d'entrée de l'application. Ce script est le seul disponible dans la partie publique. Il s'occupe ensuite d'instancier le contrôleur dont le code source se situe dans **`fuel/app/classes/controller/controleur.php`** et dont le nom est **`Controller_Controleur`**. Le répertoire contenant le framework et l'application sont dans la partie privée du site. Il est également possible d'utiliser un niveau de plus dans l'arborescence des contrôleurs.

Ainsi la classe **`Controller_Dir_Controller`** serait définie dans le fichier source **`fuel/app/classes/controller/dir/controleur.php`**. Tous les contrôleurs doivent étendre une classe du framework nommée **`Controller`** (du domaine de nom **`\Fuel\Core`**), ou étendre d'autres contrôleurs qui le font.

La classe doit ensuite posséder une méthode **`action_methode()`** publique, à laquelle les paramètres **`param1`**, **`parm2`**, ... (en nombre variable) sont passés comme des chaînes de caractères.

Au sein d'un contrôleur, il est possible d'effectuer des appels aux méthodes des modèles, diriger une pile d'appels (requête HMVC), ou encore demander à une vue de s'afficher. Une option consiste à faire correspondre à chaque contrôleur un modèle unique et plusieurs vues dont les noms se rapportent aux méthodes accessibles. Certaines méthodes comme **`before()`**, **`after()`** ou **`router()`** ont une sémantique particulière, et seront automatiquement exécutées à différents points du traitement de la requête. Ce mécanisme offre la possibilité de gérer le routage au sein de chaque

contrôleur de manière indépendante. Des fichiers de configuration permettent, quant à eux, de modifier de façon globale la manière dont les messages sont routés au niveau de l'application.

5.1.2 Modèles

Dans l'Hyperlivre, on emploie une classe statique représentant chaque modèle. Ce dernier est généralement fortement lié à sa représentation relationnelle. Il est possible d'interroger la base de données à l'aide d'un module de Fuel implémentant une ORM. Les modèles sont alors toutes des classes étendant **`Model`** (du domaine de nom **`Orm`**), et sont situées dans le répertoire **`fuel/app/classes/model`**.

En réalité, l'appel à une couche DAO (Data Access Object) se fait à travers ses classes qui implémentent le pattern Active Record. Similairement au cas des contrôleurs, la convention consiste à faire précéder le nom de la classe du modèle par un préfixe. Ainsi, la classe **`Model_Modele`** est implémentée dans le fichier source **`fuel/app/classes/model/modele.php`**.

5.1.3 Vues

Les vues peuvent être multiples et dépendent généralement de la classe **`View`**. Il s'agit cependant de scripts qui permettent l'emploi des balises HTML et/ou d'autres langages comme JavaScript. Toujours dans un souci de modularité, le framework permet l'utilisation de vues partielles. Ces dernières peuvent être compilées à divers endroits d'une vue plus globale (comme un template). Le framework insère ce contenu dans un l'objet **`Response`** qui modélise l'ensemble des éléments d'une réponse valide à envoyer au client (browser).

La totalité des scripts représentant des vues complètes ou partielles est disponible dans le répertoire **`fuel/app/views`**. En ce qui concerne l'Hyperlivre, nous avons décidé de faire correspondre le nom des scripts au nom des méthodes employés dans chaque contrôleur. Ainsi, par exemple, le répertoire **`fuel/app/views/books`** contient les vues **`index.php`**, **`view.php`**, **`add.php`** et **`edit.php`** qui correspondent respectivement au méthodes **`action_index()`**, **`action_view()`**, **`action_add()`** et **`action_edit()`** du contrôleur **`Controller_Books`**. De plus, chaque groupe d'utilisateurs, administrateurs ou utilisateurs classiques, se voit associer une vue globale différente. Les fichiers relatifs nommés **`template.php`** se trouvent respectivement dans les répertoires **`fuel/app/views/admin`** et **`fuel/app/views/user`**.

Le mécanisme d'internationalisation du site est implémenté complètement, cependant il reste un travail de traduction et d'intégration à effectuer dans la partie vue, et en particulier pour le français. Ceci se fait à l'aide de fichier de traduction, propre à chaque pages et répartis selon la même hiérarchie que celle des vues pour faciliter le travail du/des traducteurs. On trouvera l'ensemble des fichiers de traductions dans le répertoire **`fuel/app/lang`**. Au préalable il faudra cependant remplacer les champs de text « brut » des pages par des références. Le traducteur pourra s'inspirer de la page **`404.php`**.

Pour tous les autres détails, nous encourageons le lecteur à se référer à la documentation officielle du framework FuelPHP. L'ensemble de la documentation actuelle se trouve dans le répertoire **`docs`**.

5.1.4 Organisation

La Figure 5.2 représente le diagramme de classe des contrôleurs principaux de notre application. On remarque une structure hiérarchique avec comme racine le contrôleur du noyau de Fuel, **`Controller_Template`**. En effet, tous les contrôleurs sont liés à une vue globale (template), qui varie en fonction du groupe de l'utilisateur. Les contrôleurs héritant de **`Controlleur_Accessbook`** sont responsable d'effectuer des CRUD sur les entités concepts, notes et commentaires. Ils sont uniquement accessibles lorsque l'utilisateur a préalablement sélectionné un livre comme livre actif.

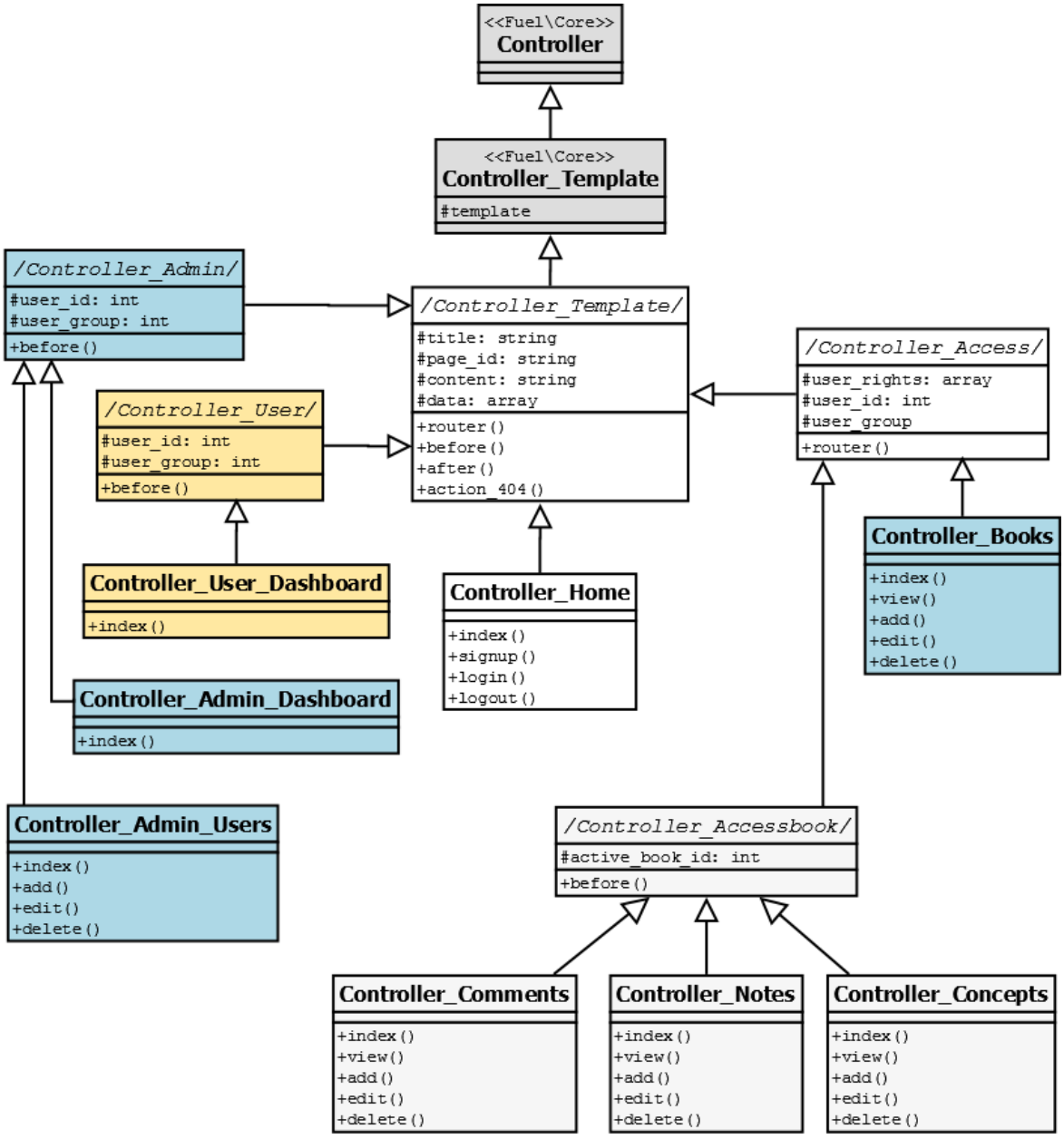

Figure 5.2 : Diagramme de classes des contrôleurs de l'Hyperlivre. Les classes affichées en bleu sont uniquement accessible par l'administrateur. Celles en saumon sont dédiées à l'utilisateur. Les classes grisées sont uniquement accessible lorsqu'un livre est sélectionné comme livre actif.

Le diagramme de classe concernant les modèles est présenté par la Figure 5.3. Les modèles sont très semblables entre eux. Ils présentent le moins de logique possible. A noter que l'utilisation de la propriété **`status`** permet d'indiquer l'état d'une note, d'un livre ou d'un commentaire. Par la suite, il est intéressant de filtrer ces entités par rapport à leurs statuts respectifs (*Archive*, *Caché*, *Public*, …).

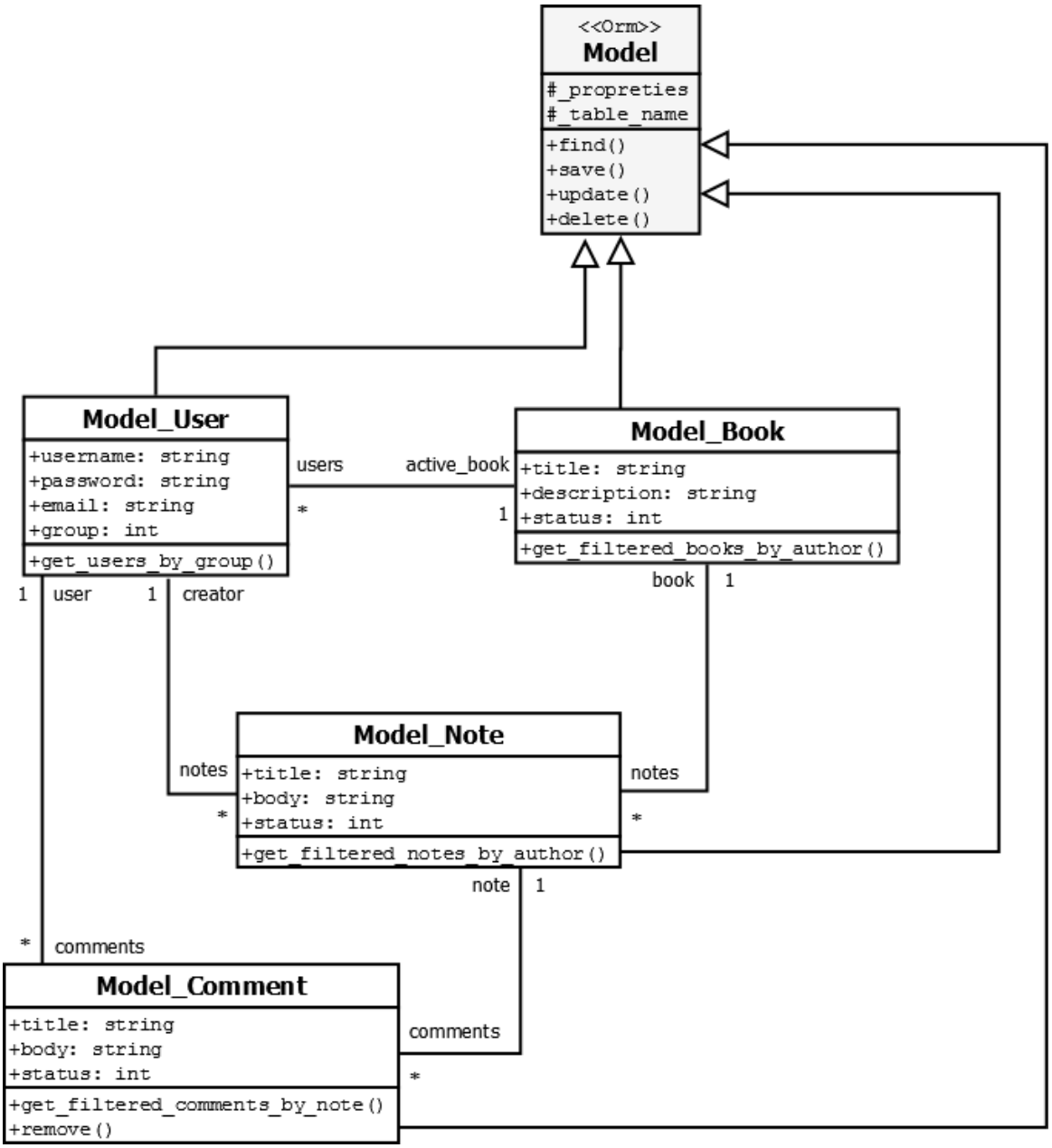

Figure 5.3 : Diagramme de classe des modèles fonctionnels.

Les modèles du diagramme illustré dans la Figure 5.4 dévoile la structure des liens logiques (inspiré de RDF), mais leurs utilisation n'est pas complètement aboutie, notamment dans les contrôleurs.

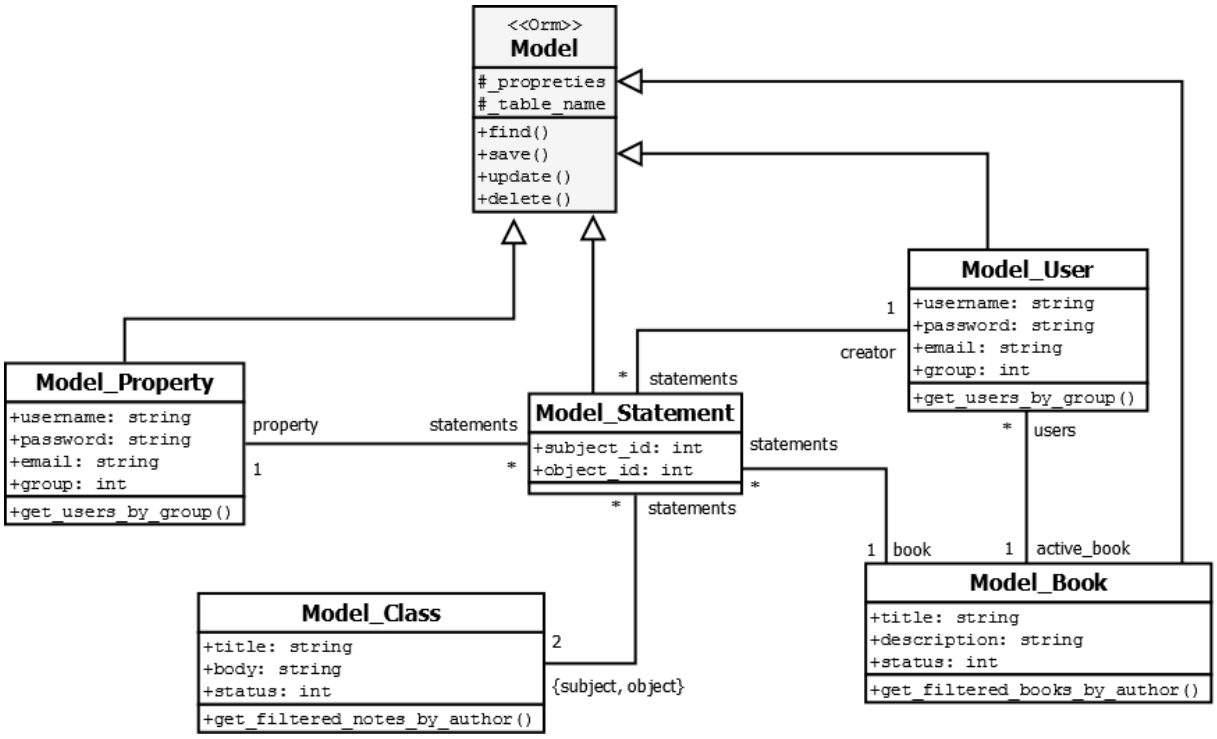

Figure 5.4 : Structure des liens logiques

6 Interface utilisateur

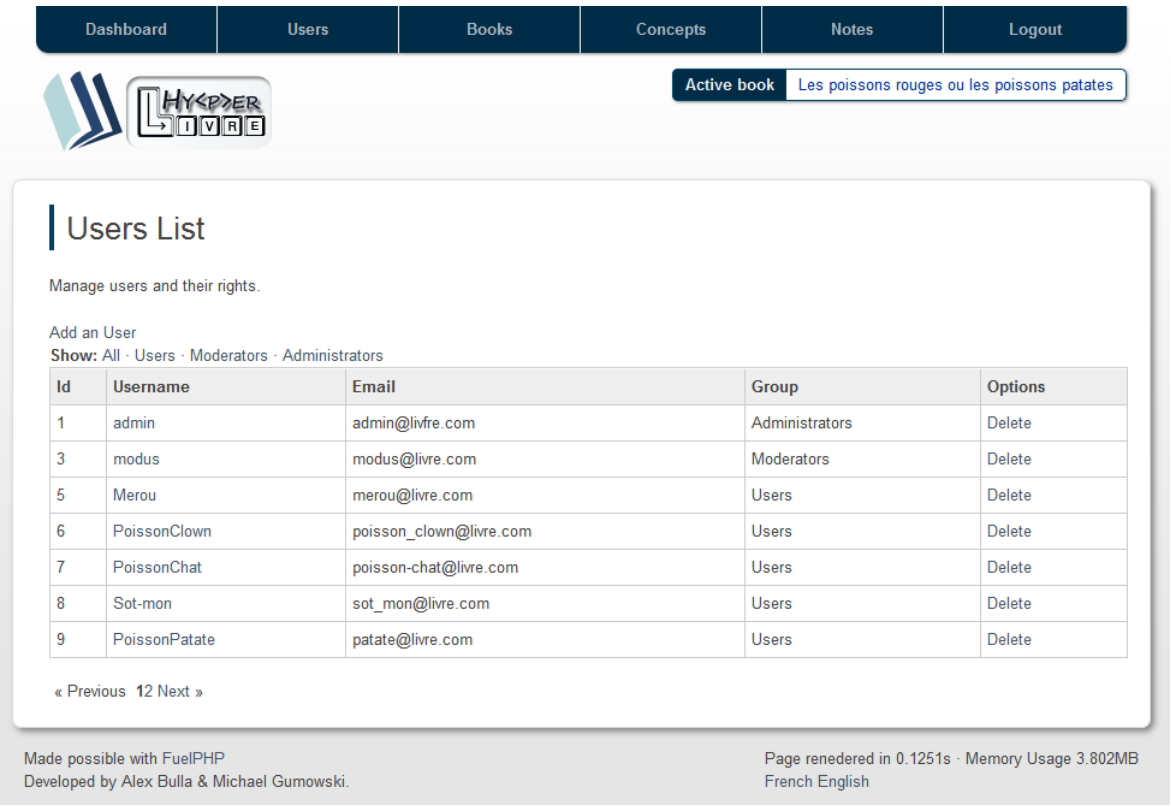

Figure 6.1 : Page d'administration des utilisateurs, accessible uniquement par l'administrateur du site.

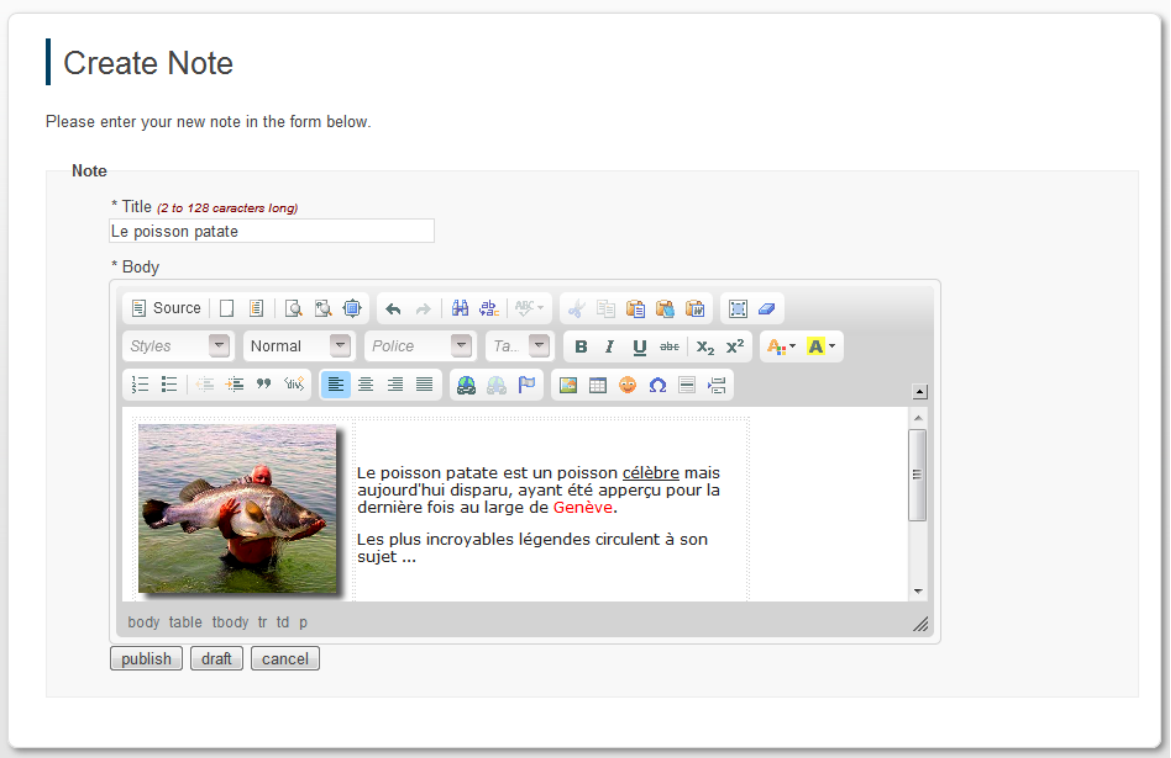

Figure 6.2 : Illustration de l'écran de création d'une note à l'aide de l'éditeur WYSIWYG.

Le menu supérieur, présent sur toutes les pages, permet de naviguer efficacement entre les différentes parties du site. Nous avons préféré l'emploi d'un style épuré et simple, où les éléments dynamique comme le javascript s'intègrent naturellement. Ces scripts se dégradent de manière respectueuse dans les navigateurs n'employant pas javascript ou les dernières techniques CSS3.

L'administrateur a désormais la possibilité de gérer de manière simplifiée la création d'instances d'hyperlivre et de calibrer leurs statuts (*masqué*, *ouvert pour l'inscription de participants*, *privé*, *archivé*). Ainsi, plusieurs livres partagent le site web et leurs accès par les utilisateurs sont paramétrables via l'interface d'administration.

Les utilisateurs peuvent désormais s'inscrire sur le site directement. Sans devoir passer par un administrateur. Ainsi, leur gestion est également simplifiée, via une interface dédiée, comme illustré dans la Figure 6.1 : **Page d'administration des utilisateurs, accessible uniquement par l'administrateur du site.**Figure 6.1.

La création de notes et concepts est également basée sur un éditeur visuel intégré (WYSIWYG), comme illustré dans la Figure 6.2.

7 Installation

Note : Afin de compléter les étapes suivantes, vous devez disposer des droits d'écriture adéquats.

Le déploiement de l'Hyperlivre s'effectue sur un serveur de type Apache[4]. Celui-ci doit être capable d'interpréter les scripts PHP 5.3. Il est préférable activer le module `mod_rewrite` dans la configuration du serveur et de compléter le fichier `.htaccess` en fonction de vos besoins.

Pour déployer le script SQL fourni avec l'application, il faudra posséder une connexion à une base de données MySQL. Nous conseillons l'installation supplémentaire du gestionnaire MySQL open-source phpMyAdmin[5] pour contrôler efficacement les connexions à la base. Celui-ci offre la possibilité d'importer le schéma SQL de l'Hyperlivre. Cependant, une interface de type terminal suffit amplement à la tâche (utilisez la commande `run` pour lancer le script SQL).

Si vous installez l'application en local, il existe des outils tout-en-un, comme Xampp[6], qui installent rapidement un environnement de développement complet sur votre machine.

L'archive fournie doit être décompressée dans un répertoire de votre site web, généralement à la racine. Il convient ensuite de restreindre l'accès à tous les répertoires ainsi crées, à l'exception du répertoire `public` qui doit être accessible aux internautes.

Tous les fichiers de configuration de l'Hyperlivre se trouvent dans le répertoire `fuel/app/config`. Il vous faudra notamment modifier certaines options dans le fichier `db.php`. Celui-ci regroupe les informations de connexion nécessaires pour dialoguer avec la base de données. Les autres fichiers de configuration permettent notamment de restreindre ou autoriser l'accès aux ressources pour chacun des groupes d'utilisateurs, d'indiquer les routes par défaut, et de piloter les détails du framework (cache, gestion des logs, ...).

Vous pouvez ensuite créer un utilisateur et lui fournir les droits administrateur (groupe 100) pour commencer à utiliser l'application.

L'archive fournies avec ce rapport est également disponible sur github, à l'adresse :

https://github.org/bullowski/hyperlivre

[4] http://www.apache.org/
[5] http://www.phpmyadmin.net/
[6] http://www.apachefriends.org/en/xampp.html

8 Perspectives

Bien que nous soyons arrivés à terme de ce projet, il existe un grand nombre de possibilité d'extension et de correction à apporter. En voici une liste non-exhaustive.

- Finaliser l'intégration des liens logique et conceptuels tels qu'ils ont été prévus dans l'implémentation originale.

- Actuellement, l'injection de code est possible à partir de différents formulaires. Ce problème était également présent dans l'hyperlivre. Un travail sur la sécurisation des champs est donc fortement recommandée si le projet devait s'étendre à une population d'utilisateurs potentiellement non-contrôlée. Nous proposons l'introduction unilatérale d'un langage de marquage comme `textile` ou `MarkDown`. Ainsi, l'injection de scripts offensifs serait évitée.

- Vérification d'identité à travers la validation d'adresse email.

- Le travail d'internationalisation pour intégrer le français et/ou d'autres langues.

- Le chargement d'image peut être facilement ajouté dans la classe `Controller_User_Dashboard`. Pour la vue il est possible d'intégrer un bouton permettant d'appeler cette action du contrôleur.

- Un installeur complet, initialement envisagé, permettrai un déploiement facilité sur la plateforme web. Ceci permettrait également une distribution à plus large échelle de l'hyperlivre.

- Des tests de régression à l'aide de PHPUnit seraient un plus pour contrôler l'évolution du projet.

9 Bilan

Dans ce projet, nous avons été confrontés à l'utilisation intensive de technologies PHP et Javascript. Ceci nous a permis de compléter notre expérience en la matière. L'utilisation du framwork FuelPHP a été enrichissante et nous a permis de découvrir le monde open-source orienté web.

Nous avons cependant sous-évalué la taille du projet et n'avons pas réussi à complètement atteindre les objectifs que nous nous étions fixés. Malgré tout, nous sommes satisfaits de ce que nous avons achevé, pour une première expérience avec Fuel. La prise en main est facile, mais nécessite tout de même une certaine expérience de PHP et plus particulièrement en technologie web.

Il a été particulièrement intéressant de constater les progrès accompli par les développeurs du framework, qui ont fourni un travail remarquable. Fuel s'inspire notamment des technologies similaires à « Ruby On Rails » pour les intégrer au monde PHP. La réactivité des développeurs de Fuel est vraiment un point fort. Ce framework s'avère vraiment très prometteur, d'autant plus qu'il n'est pas encore dans sa phase finale. Quoiqu'il en soit, nous avons apprécié de travailler avec cet outil.

10 Références

- FuelPHP : http://fuelphp.com/ et https://github.com/fuel
- Projet stationWagon : https://github.com/abdelm/stationwagon
- CKEditor : http://ckeditor.com/
- Nettuts+ : http://net.tutsplus.com/
- JQuery : http://jquery.com/
- JQueryForDesigners : http://jqueryfordesigners.com/
- Wikipedia : http://www.wikipedia.org/
- Les poissons rouges : http://www.lapagedupoissonrouge.net/