Projet web - UnsapaIPW

Léo Unbekandt - Guillaume Paran - Lucas Saurel

Mai 2012

Table des matières

Introduction

Le 2^e sujet nous a été attribué. Il s'agissait de réaliser une plateforme de gestion d'examens sur laquelle des professeurs peuvent créer des examens et les étudiants doivent alors déposer un fichier qui soit accessible pour le professeur en guise de rendu. Le sujet nous imposait un certain nombre de cas d'utilisation auxquels nous avons répondu.

Un des points importants de l'application étaient la gestion des droits. Selon chaque rôle, les utilisateurs ne doivent pas voir la même chose pour une ressource donnée. Par exemple, pour les examens : un étudiant doit voir ceux auxquels il est inscrit tandis qu'un professeur doit voir ceux dont il est responsable.

0.1 Fonctionnalités de l'application

0.1.1 Gestion des rôles

L'application web que nous avons réalisée est un système de gestion d'examens. On distingue trois catégories d'utilisateurs :

- Les étudiants : ils consultent les examens auxquels leur promotion est inscrite et déposent leurs compositions.
- Les responsables d'examens : ils créent des examens et notent les étudiants.
- L'administrateur : il gère l'ensemble des utilisateurs, des promotions et des examens.

0.1.2 Les différentes *user stories*

L'utilisateur consulte la liste des examens auxquels il est inscrit

```
Route : /exams
Controller : ExamsController
Action : indexAction
```

L'utilisateur s'authentifie

Il peut soit cliquer manuellement sur *se connecter* ou il lui sera demandé de s'authentifier lorsqu'il tentera d'accéder à des pages où un utilisateur inconnu n'a pas les droits. Les mots de passe sont stockés en sha1 dans la base de données, personne à part l'utilisateur ne peut le récupérer.

```
Route : /login
Controller : FOSUserBundle:SecurityController
```

L'utilisateur s'inscrit

Lorsqu'un utilisateur s'inscrit son compte est immédiatement activé, mais nous pouvons configurer notre application pour qu'un mail soit envoyé avec une URL de validation.(Chemin : app/config/config.yml)

```
Route : /register
Controller : FOSUserBundle:RegistrationController
```

L'utilisateur dépose un examen

Parmis les examens auxquels il est inscrit un étudiant peut proposer un rendu pour l'un d'entre eux. Les fichiers DOC, DOCX, PDF et ZIP sont acceptés. Nous faisons la vérification par rapport aux mime-types et pas seulement par les extensions.

Lorsqu'il choisit l'examen pour lequel rendre un document, il sera averti s'il a déjà rendu quelque chose. Et l'interface lui proposera de télécharger l'ancienne version de son travail.

Page de soumission :

```
Route : /register/submit
Controller : ExamsController
Action : submitAction
```

Téléchargement de fichier :

```
Route : /download/{:examid}/{:userid}
Controller : AttendController
Action : downloadAction
```

L'utilisateur crée un examen

Si un utilisateur à le rôle de responsable de TD, il a la possibilité d'ajouter de créer des examens.

```
Route : /exams/add
Controller : ExamsController
Action : addAction
```

L'utilisateur associe des étudiants à un examen

Pour cela il y a 3 possibilités :
- Lors de l'ajout d'un examen, il choisit une promotion. Par défaut tous les étudiants de cette promotion seront concernés par cet examen.
- Lors de l'ajout d'un exam, après avoir choisi une promotion, il peut cliquer sur détails, et cela lui permettra de voir tous les étudiants de la promotion sélectionné, et il pourra choisir, étudiant par étudiant qui participe à l'examen.
- Sur la page listant les examens dont il est responsable, il peut cliquer sur "Gérer les étudiants", et il accèdera à une page où comme dans le point ce-dessus, il pourra gérer plus finement les étudiants qui sont concernés :

```
Route : /exams/{:examid}/students
Controller : AttendController
Action : examChoiceAction
```

Associer une note à un document déposé

Lorsqu'un examen est terminé, le responsable peut commencer à noter les rendus. Il ne peut pas le faire avant, car nous laissons aux étudiants la possibilité de modifier leur rendu jusqu'au dernier jour. Et noter un rendu qui n'est pas définitif ne sert à rien.

Donc sur la page qui liste les examens dont il est responsable (/exams), dans la partie "Examens terminés", un responsable peut avoir accès à la liste des étudiants. Voir ceux qui n'ont rien rendu, et ceux qui ont proposé un document. Il peut alors télécharger ce qui a été proposé. (/download/ :examid/ :userid)

Afin de faciliter l'ordre sur l'ordinateur du responsable, les documents téléchargés ont un nom sous la form "nomexamen_nométudiant.ext".

```
Route : /exams/{:examid}/marks
Controller : AttendController
Action : markStudentsAction
```

Un utilisateur accède aux moyennes par promo

Une page avec des statistiques simples est proposée pour tous les utilisateurs, même les non-identifiés.

```
Route : /stats
Controller : StatsController
Action : indexAction
```

Un utilisateur peut gérer les promos

Il existe un utilisateur ayant le rôle d'administrateur sur le site. Il a accès à un panneau d'administration, où il va pouvoir ajouter/modifier/supprimer des promotions, voir/modifier/(dé)sactiver/changer de rôle les comptes utilisateurs.

```
Route : /admin
Controller : AdminController
Action : indexAction
```

0.1.3 Ce que nous n'avons pas fait

Génération des mots de passe

Il est demandé dans le sujet de générer aléatoirement un mot de passe lors de l'inscription d'un nouvel utilisateur, or nous utilisons un Bundle qui gère toute la partie, identification, enregistrement, récupération de mots de passe, gestion du profil... Ainsi pour redéfinir la manière dont l'enregistrement est géré, il est nécessaire de redéfinir un certain nombre de classes (concrètement le controller, le formtype, le handler et le template). Nous avons considéré que c'était beaucoup de temps de travail pour pas grand chose donc nous nous sommes concentrés sur des tâches plus prioritaires.

3 états pour les examens

Le sujet demandait à ce qu'un examen puisse être :
- En création
- En cours
- Clos

Dans notre application, nous nous basons sur la date d'échéance de l'examen En l'occurrence, un examen dont la date n'est pas encore passée est "en cours" tandis qu'un examen dont la date est passée est "clos" et la notation a commencé.

0.2 Schéma de données

Le sujet nous imposait d'utiliser plusieurs entitées auxquelles nous avons ajouté un certain nombre d'attributs.

Voici le schéma relationnel de notre base de données, avec les relations entre les différents modèles :

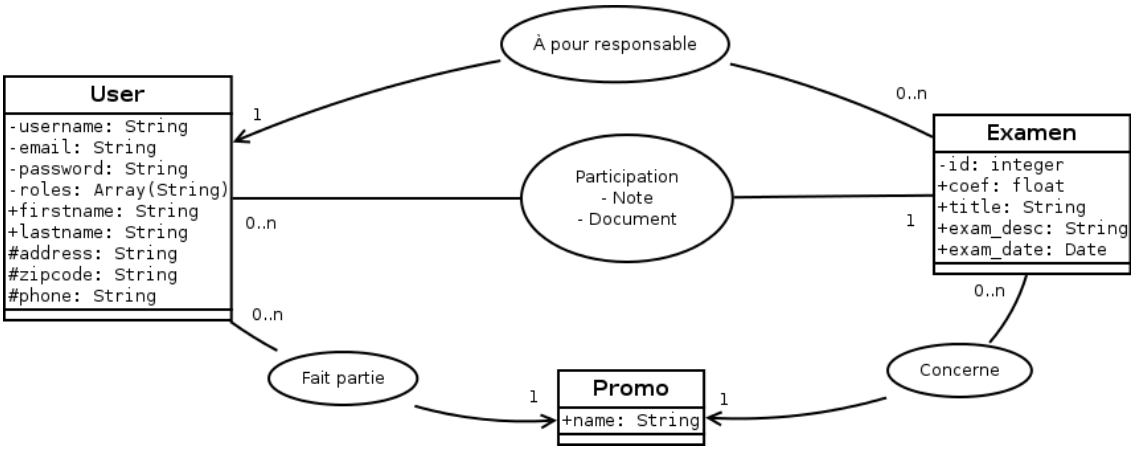

FIGURE 1 – Schéma relationnel

Puis voilà la modélisation exacte de nos données tel quelles sont stockés en base de données :

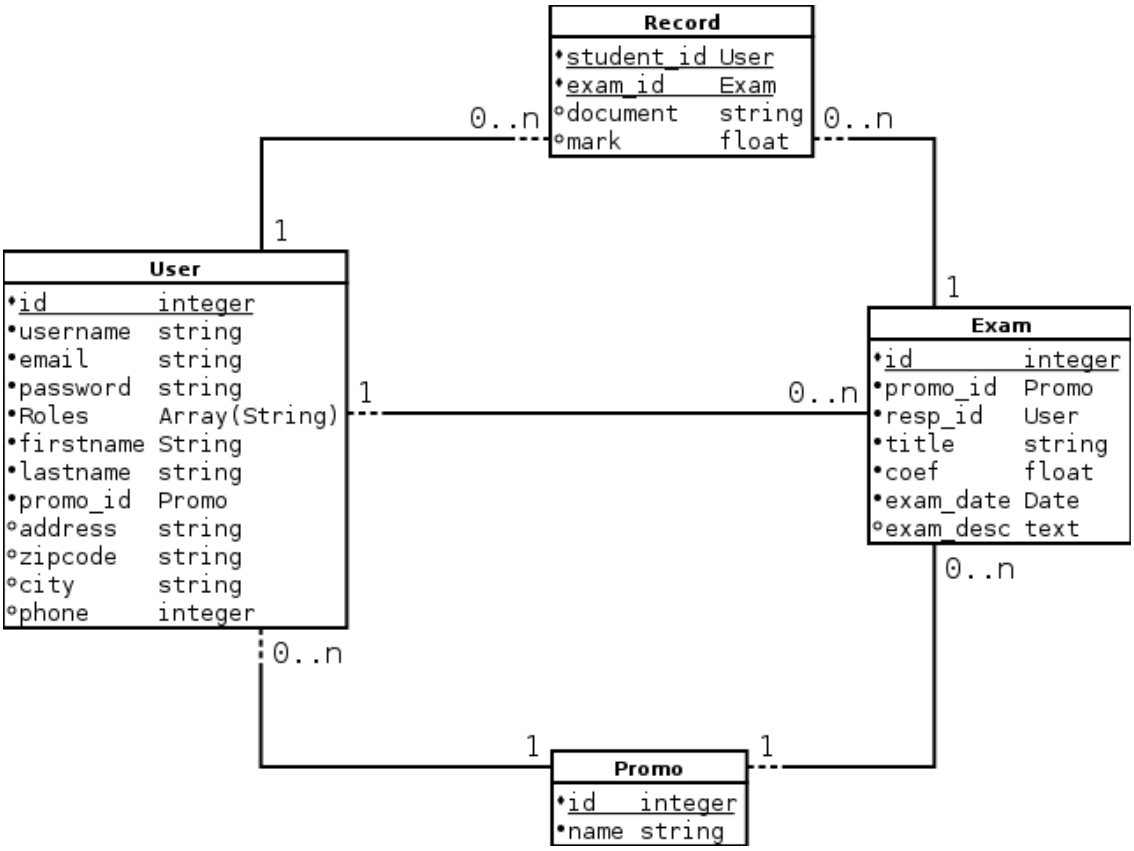

FIGURE 2 – Modèle physique de base de données

0.2.1 User

En plus des champs prérequis :
- Prénom
- Nom
- Adresse
- Code postal
- Ville
- Adresse e-mail
- Numéro de téléphone

Le UserBundle nous rajoute toute la partie nécessaire à la gestion de l'utilisateur côté serveur.
- Le mot de passe ⇒ chiffré en sha1 dans la BDD
- Les rôles ⇒ Pour la gestion des permissions
- Nom d'utilisateur
- Expiration du compte, confirmation par email etc.

Et enfin nous avons ajouté un champ pour notre besoin qui est une promotion, en effet, on considère qu'un étudiant appartient à une promotion précise.

0.2.2 Promo

Les promotions correspondent à un regroupement d'utilisateur, dans le cas d'utilisation présent, tous les étudiants d'une même année appartiennent à une promotion unique.

0.2.3 Exam

Un examen représente une épreuve définie par un enseignant. L'examen comprend les propriétés suivantes :
- Titre
- Descrption
- Promotion
- Coefficient
- Date limite
- Responsable : Un user responsable de TD

Par défaut, lors de la création de l'examen, tous les étudiants de la promotion sélectionnée sont concernés par l'examen, mais il est également possible de modifier au cas par cas si un étudiant est affecté par l'examen.

0.2.4 Record

L'entité Record fait le lien entre les étudiants et les examens. En effet dans un record on attribue pour un étudiant et un examen, la note, ainsi que le document rhttps ://rubygems.org/endu (Fichier pdf ou word).
On a donc :
- Étudiant
- Examen
- Note
- Document

Ainsi un Record avec la note et le document NULL, correspond au fait qu'un étudiant doit effectuer un rendu pour un tel examen. Quand il rend quelque chose, on peuple le champ 'Document'. Et ensuite quand l'examen est terminé le responsable de l'examen peut associer une note.

0.3 Réalisation Technique

0.3.1 Symfony2

Contrairement à ce qui a été vu en cours, nous avons choisi d'utiliser la version 2 du framework symfony. Pourquoi avons nous fait ce choix ? Nous trouvions intéressant de nous porter sur une technologie plus neuve que symfony 1.x, en pensant que d'ici quelques années, cette version sera bien plus utilisé dans des projets qu'une plus ancienne version.

Outre, cela nous permettait d'accumuler d'un côté les connaissances sur Symfony 1.x du cours, et en plus ce qui a été fait dans Symfony2. Du coup d'un point de vu pédagogique, nous avons bien plus appris de cette manière.

De plus la communauté Symfony s'est massivement adapté à la Symfony2, ainsi de nombreux plugins sont disponibles aisément, grâce au projet *Composer* [1]http ://getcomposer.org/ un gestionnaire de paquets pour PHP, et très utilisé dans cet écosystème, à la manière des gems en Ruby [2]https ://rubygems.org/.

0.3.2 UserBundle

Nous avons utilisé un Bundle d'extension nommé : UserBundle.

Ce bundle constitue la brique applicative qui permet d'effectuer toutes les actions nécessaires à presque tous les projets, c'est-à-dire :

- Inscription
- Authentification
- Connexion
- Déconnexion
- Gestion du profil
- Changement de mot de passe

Nous évitant du travail laborieux qui ne sert pas sur le plan métier de l'application.

0.3.3 Jeu de données

Afin de pouvoir effectuer des tests, nous avons mis en place un jeu de données comprenant plusieurs utilisateurs. Les mots de passe utilisés sont identiques aux noms d'utilisateur. Pour s'authentifier en tant qu'administrateur il suffit d'utiliser l'identifiant 'admin'. Les différents responsables de TD capables de gérer les examens et de noter les étudiants sont :

- 'tduser1'
- 'tduser2'
- 'tduser3'
- 'tduser4'
- 'tduser5'

Enfin, on peut s'authentifier en tant qu'étudiant en utilisant les identifiants suivants en choisissant 'x' entre 1 et 10 :

- Promo 2012 : 'user2012x'
- Promo 2013 : 'user2013x'
- Promo 2014 : 'user2014x'

Nous avons essayé de couvrir l'intégralité des possibilités de l'application avec notre jeu de données de test. C'est-à-dire 90 examens, qui se sont déjà passés où qui sont à venir, 30 étudiants de 3 promotions différentes, 5 responsables de TD, un administrateur, et 960 participations aux examens. Parmi ces participations, certaines ont déjà été notées, d'autres sont en attente de correction, ou encore n'ont pas de rendu, du coup ne peuvent être corrigées. Il y a aussi des participations où l'examen n'est pas terminé, et donc l'étudiant n'a pas forcément rendu de document.

0.3.4 Les tests avec PHPUnit

Nous avons principalement testé les différentes entités. Grâce à PHPUnit la couverture en tests de notre code a été modélisée :

Couverture des tests : http ://ares-ensiie.eu/ unbekandt2011/UnsapaIPW/cov

Pour le faire depuis la racine du projet :

```
phpunit --coverage-html=cov/ -c app/
```

0.3.5 La documentation développeur avec phpDocumentor

Tout notre code a été documenté en utilisant phpDocumentor. On peut trouver cette documentation à l'adresse suivante :

Documentation développeur : http ://ares-ensiie.eu/ unbekandt2011/UnsapaIPW/doc

Pour la générer depuis la racine du projet :

```
phpdoc -c app/phpdoc.dist.xml
```

0.4 Organisation de l'équipe

0.4.1 Gestion du code source

Afin de travailler le plus efficacement possible, nous avons décidé d'utiliser l'outil collaboratif Github. Ce service permet une méthode de travail complètement décentralisé, qui permet à chacun de travailler de son côté, en ensuite de notifier chacun par email lorsqu'il a des proposition à effectuer, afin de centrer la gestion du projet autour de la communication, sûrement l'élément le plus important dans la gestion de projet.

FIGURE 3 – Logo de Github

Techniquement, cette plateforme est basé sur le SCM **GIT**, dont l'efficacité n'est plus à prouver. Il est utilisé par des structures énormes (Développement du kernel Linux, Facebook, Twitter par exemple) et est reconnu pour être efficace.

0.4.2 Répartition des tâches

Concernant la répartition des tâches, nous avons fonctionné de manière relativement autonome, nous ne nous sommes pas attribué à chacun des tâches, mais nous avons listé tout ce que nous avions à faire dans un bug tracker. De cette manière chacun était libre de s'approprier les tâches selon sa motivation et les sujets sur lesquels il souhaitait travailler.

Conclusion