TECHNOLOGIES WEB - PROJET PHP
RAPPORT DE PROJET

GALISSON ANAIS
REGNAULT ANTOINE
NANTEL MAELIG

Rendu le 24/06/2015

Introduction

Durant le cours de programmation web dispensé en première année de formation Informatique Multimédia et Réseaux (IMR) de l'ENSSAT, il nous a été demandé de réaliser une application web devant permettre aux enseignants de l'ENSSAT de choisir les cours qu'ils souhaitent enseigner pour l'année universitaire. L'objectif principal de l'application est de simplifier la réalisation des emplois du temps en début d'année. Ce projet sera encadré par M. CHEVELU qui tiendra le rôle de client.

L'objectif pédagogique de ce projet est d'une part de nous donner une première expérience en gestion de projet (sur l'ensemble de son cycle de vie) et d'autre part, de nous permettre d'appliquer le modèle de conception Modèle Vue Contrôleur (MVC) à l'aide du langage de programmation PHP et du framework CodeIgniter.

I) Étude des besoins

Lors du lancement d'un projet, la première étape est de comprendre correctement l'attente du client. Pour cela il est nécessaire d'établir une liste des exigences logicielles, c'est-à-dire les fonctionnalités générales et métiers à réaliser. Ensuite, il est important d'établir une maquette des interfaces de l'application afin que le client puisse la valider.

A) Cahier des charges : spécification des exigences logicielles

Nous avons étudié le sujet donné au lancement du projet afin d'établir une liste complète des fonctionnalités attendues. Pour cela, nous avons déroulé des cas d'utilisations en nous plaçant à la place des utilisateurs finaux.
Nous avons eu un certain nombre de questions à éclaircir, ce qui a été fait lors d'une séance de projet avec M.CHEVELU.

Une version très détaillée et formelle du cahier des charges et disponible en annexe de ce rapport.

B) Maquettage de l'interface graphique

Après avoir défini l'ensemble des fonctionnalités, nous avons réalisé des maquettes de l'interface graphique dans le but d'avoir une vision globale de notre application. Nous avons voulu réaliser une application avec le moins de pages possible, où les informations sont rapides d'accès pour rendre l'utilisation simple et rapide. L'objectif étant de disposer d'un site avec la navigation la plus fluide et intuitive possible, afin de simplifier la prise en main pour nos futurs utilisateurs.

Pour avoir accès aux données facilement, une barre de navigation est présente sur toutes les pages et permet d'avoir accès aux informations à tout moment. Les différents onglets de la barre de navigation seront :

- Mes cours : Contient l'ensemble des cours auquel l'enseignant connecté est affecté ainsi qu'un bilan des heures effectuées et à faire.

- Vœux : Affiche l'ensemble des cours, c'est dans cet onglet que l'enseignant peut s'inscrire à un cours, ou devenir responsable d'un module.

- Enseignants : Permets l'accès à liste des enseignants. Une barre de recherche en fonction du nom et du prénom est prévue. L'enseignant accède au mail, au statut et aux cours de l'enseignant qu'il choisit.

- Mon compte : Cet onglet contiendra les informations du compte de l'enseignant connecté. C'est ici qu'il pourra modifier ses informations (ex. : mot de passe).

Pour l'administrateur, qui a plus de droits (on rappelle qu'un administrateur est en fait un enseignant particulier), un onglet supplémentaire "Administration » sera ajouté.
De plus certains onglets déjà présents permettent l'accès à des pages avec plus d'options :

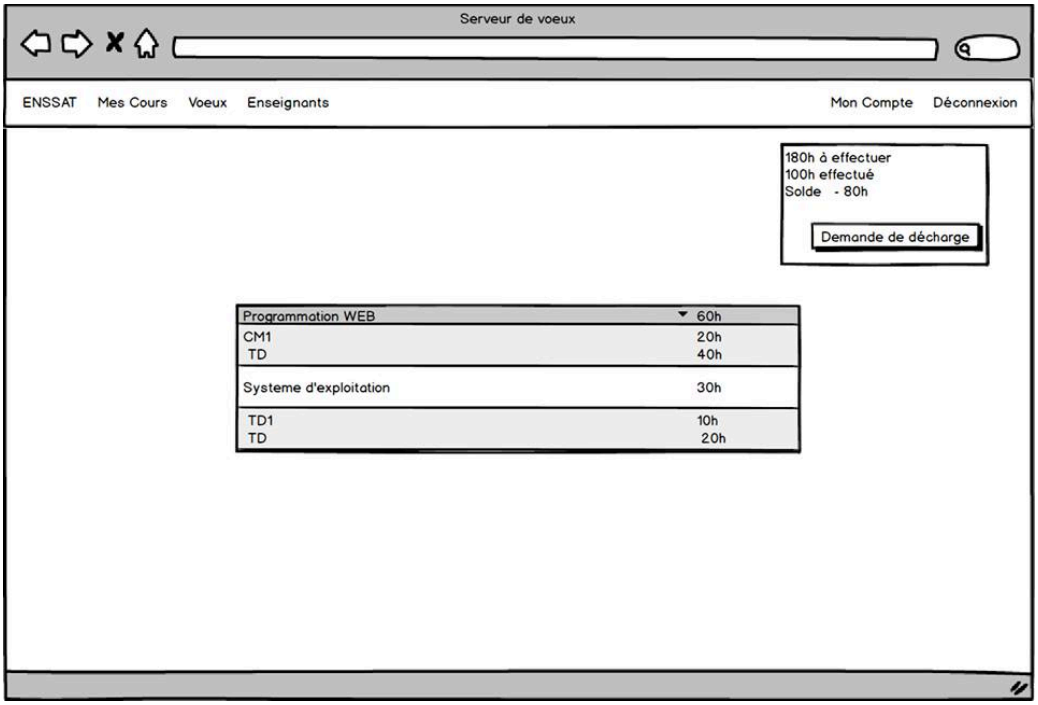
- L'onglet « Vœux » est beaucoup plus complet, l'administrateur peut, en plus des fonctionnalités déjà existantes, ajouter/supprimer un enseignant à un cours. Il peut supprimer/ajouter un responsable de module parmi les enseignants existants. L'ajout et la suppression de modules et de cours sont aussi des fonctionnalités présentes.

- La page "Enseignant" est aussi plus complète, l'administrateur a la possibilité de changer les informations d'un enseignant, de le supprimer et d'en créer un.

Nous allons maintenant présenter quelques pages de la maquette réalisée (l'ensemble plus complet se trouve en annexe de ce rapport).

Page "Mes cours" (aussi page d'accueil) :

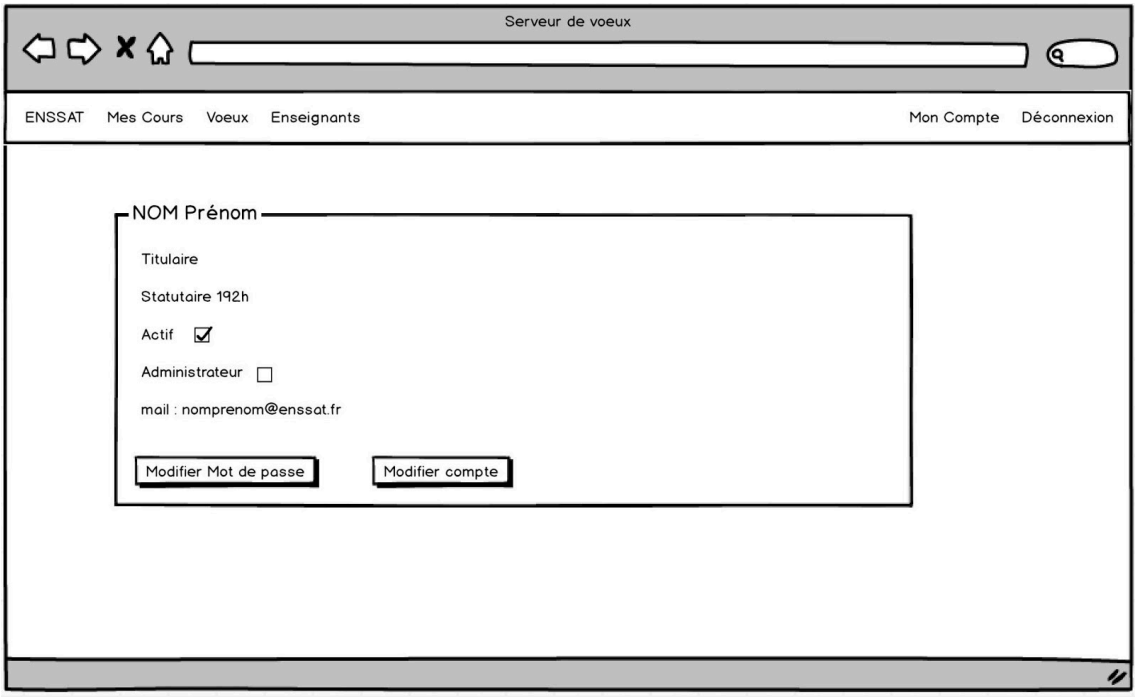
Le but de cette page est d'afficher à l'enseignant connecté les cours qui lui sont assignés.



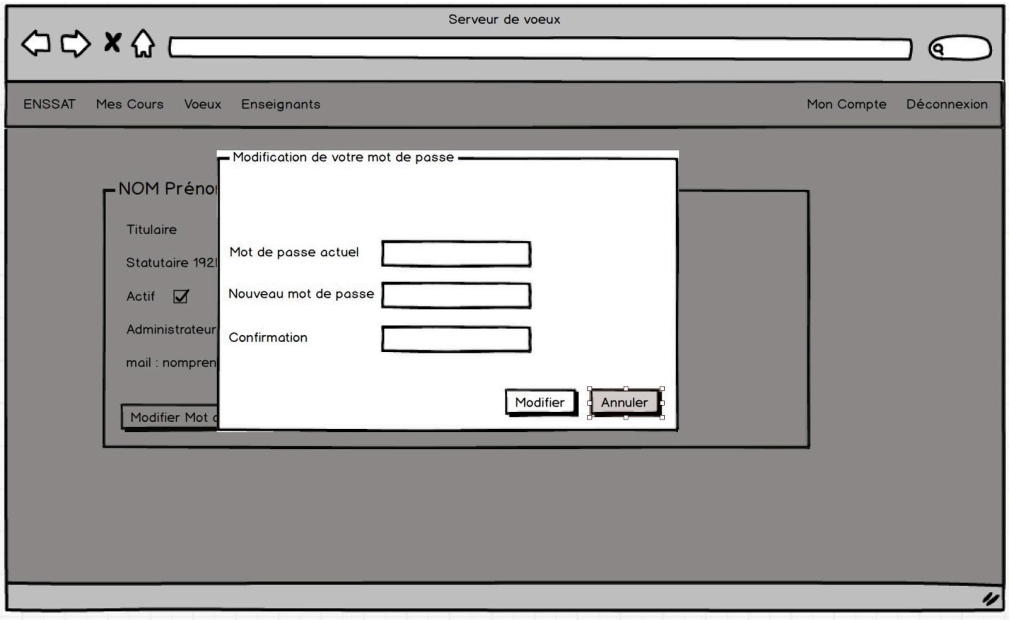
On y voit apparaître aussi en haut à droite de l'écran l'affichage du nombre d'heures réalisées et celles qui lui restent à faire. Le solde est de couleur rouge quand il est négatif et de couleur verte quand l'enseignant a effectué le nombre d'heures minimales de son statutaire ou plus (les heures supplémentaires sont possibles).

Pour le tableau affichant les cours de l'enseignant, nous avons décidé de pouvoir afficher à la fois les modules et les cours qui les composent. Cela représente potentiellement un très grand nombre de lignes et il y aurait eu une perte de visibilité considérable. Pour remédier à ce problème, nous avons pensé à faire un tableau "accordéon". Au départ, le tableau n'affiche que les modules et lorsque la personne clique sur celui qui l'intéresse, les parties de cours apparaissent en dessous et disparaissent lors du second clic. Nous avons ensuite décidé d'ajouter le nombre d'heures auquel s'est inscrit l'enseignant pour un cours pour qu'il ait une vision globale des heures qu'il a à effectuer.

Page "Mon Compte" :



Cette page contient tous les renseignements d'un enseignant. Il a la possibilité de modifier ses différents attributs (mail, statut, nom, prénom…). Pour la modification, nous avons décidé d'utiliser des pop-up pour éviter le surplus de pages (une fenêtre apparaît dynamiquement sur la page actuelle, sans avoir besoin de refaire un chargement). Par exemple sur l'image du dessous, un modal s'affiche lorsque l'enseignant veut modifier son mot de passe. Ce type de composant est très bien adapté à des affichages de taille fixe et raisonnable (pour un grand nombre d'informations, il est préférable de charger une nouvelle page).



Cette proposition de maquette a été validée par M.CHEVELU durant une de nos séances de projet. Nous verrons plus tard dans ce rapport quels sont les choix technologiques que nous avons retenus pour permettre la mise en œuvre de cette maquette.

Nous avons fait le choix d'utiliser la couleur orange comme dominante dans notre application, étant donné que c'est la couleur principale de la charte graphique de l'ENSSAT.

II) Gestion de projet

A) Organisation de groupe

Ce projet est à réaliser en trinôme. Afin de simplifier la collaboration, nous avons tous décidé de mettre en place un environnement de travail similaire et d'utiliser un outil de gestion de versions.

B) Outil de gestion de versions : Git

Aujourd'hui, dans tous les projets de développement informatique, la gestion de versions est devenue totalement indispensable. Les objectifs principaux sont de permettre le maintien de plusieurs versions viables d'un logiciel dans le temps, ainsi que de faciliter le travail de plusieurs personnes sur un même projet. Pour cela, l'appui d'un logiciel est très souvent nécessaire. Nous avons choisi d'utiliser l'outil Git et d'utiliser les serveurs de GitHub pour le partage.

C) Environnement de développement

Afin de simplifier le travail de groupe, nous avons également choisi de tout travailler avec le même environnement de développement. Pour cela, nous avons décidé d'utiliser l'IDE Éclipse accompagné du plug-in PDT (PHP Development Tools). Afin d'homogénéiser notre code, nous avons réalisé un formateur de code (permets une indentation automatique du code source lors de la sauvegarde). Ce formateur ainsi que les préférences utilisées dans Eclipse sont fournis dans les sources du projet afin de permettre à un éventuel développeur de nous rejoindre en s'adaptant rapidement à nos conventions de codage.

Concernant la base de données, nous avons tous travaillé avec un serveur MySQL local à notre machine (nous n'avons donc pas de base de donnée partagée). Nous avons juste eu à nous mettre d'accord au lancement du projet sur le nom de la base, ainsi que les informations d'identifications.

D) Planification prévisionnelle

Pour nous permettre de mener à bien le projet, nous avons dès le départ décidé de planifier les différentes étapes et de nous tenir du mieux possible au planning. Le diagramme de Gantt initial est fourni en annexe de ce rapport.

Nous avons choisi de nous inspirer des méthodes agiles pour la gestion de ce projet. Pour cela, nous avons défini des itérations courtes dans notre développement et de faire régulièrement le point sur l'avancé du groupe. Cela nous a permis de cibler rapidement les points forts/points faibles de chacun et ainsi de nous entraider pour gagner en efficacité.

- Itération 1 : mise en place des environnements de travail et de la structure générale de l'application

- Itération 2 : créations des opérations de base sur les différentes entités (CRUD)

- Itération 3 : gestion de la table 'contenue' permettant d'afficher et d'affecter les cours d'un utilisateur

- Itération 4 : regrouper l'ensemble des pages, gestion de la navigation

Le diagramme de Gantt prévisionnel est fourni en annexe de ce rapport.

Étant donné que c'est notre premier projet de groupe cette année, sur des technologies pas forcément maitrisées au début du projet, il a été très difficile de chiffrer les temps de développement. Nous n'avons donc pas réussi à suivre notre planning prévisionnel. Un diagramme réel est également fourni en annexe.

III) Technologies mises en œuvre

A) Technologies imposées

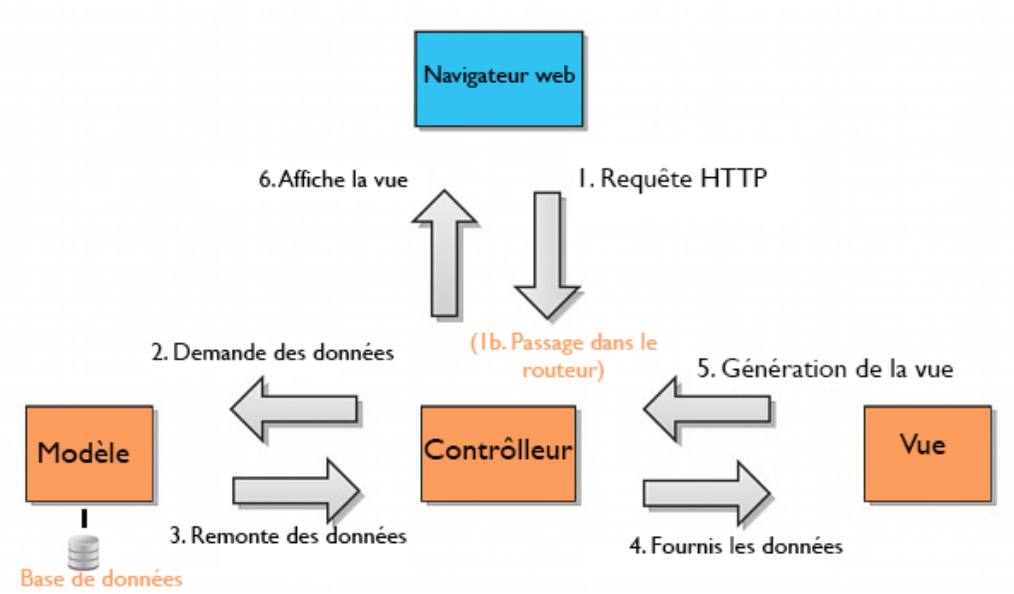
Pour ce projet, l'ENSSAT nous a imposé certains critères technologiques. Nous devons réaliser l'application en utilisant le framework CodeIgniter, et donc en utilisant le langage PHP et en respectant l'architecture MVC.
La structure de la base de données ainsi que le SGBD MySQL nous a également été imposé.

Architecture MVC

- Modèle : cette partie gère les données de l'application. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. Le modèle permet aussi de vérifier la validité des données de l'application (par exemple, un login d'enseignant doit être unique…).

- Contrôleur : cette partie gère la logique métier de l'application et qui prend des décisions. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue (et idem dans le sens inverse, il récupère les données de la vue dans le cas de formulaires.). C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

- Vue : cette partie se concentre sur l'affichage. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML, mais aussi quelques vérifications ou boucles dans un langage dynamique (dans notre cas, des injections PHP). Par souci d'ergonomie, elle peut également s'occuper de faire un premier contrôle de validité sur les données à l'aide de scripts (par exemple JavaScript). Les erreurs de saisies peuvent ainsi être détectées au plus vite et ne nécessitent pas forcément un rechargement de page.



Un grand nombre de frameworks (et plus généralement d'applications) utilise ce patron de conception. Il est donc important de bien comprendre son fonctionnement indépendamment de l'implémentation utilisée afin de pouvoir l'adapter dans d'autres situations.

PHP et framework CodeIgniter

Un framework est un ensemble de composants logiciels permettant de définir les grandes lignes du développement d'une application. On peut considérer un framework comme "*une boite à outils*". L'avantage de CodeIgniter pour un premier projet est sa légèreté. En effet, il a été conçu pour fournir le juste minimum de fonctionnalités et d'outils pour le développement d'applications. L'ensemble des composants qu'il fournit est en plus optionnel. Son temps d'apprentissage est donc très limité comparé à d'autres frameworks.

Parmi les fonctionnalités que CodeIgniter propose, on retrouve notamment :

- Une librairie pour gérer les bases de données : ActiveRecord. Cela permet de ne pas écrire de code SQL à la main, mais d'utiliser des appels de fonctions PHP pour construire notre requête. Le grand avantage est ainsi de s'abstraire du SGBD utilisé et donc de pouvoir migrer plus facilement l'application si besoin. C'est une librairie dite ORM (Object Relational Mapping) permettant de gérer la correspondance entre les entités PHP et leur stockage en base de données relationnelle.

- Une librairie pour les sessions. CodeIgniter n'utilise pas le mécanisme de sessions proposé par PHP. Le fonctionnement pour l'utilisateur est très similaire.
- Des helpers, fournissant une aide dans la réalisation de tâche récurrente pour les développeurs (gestion des URL, création de formulaires).

- …

La documentation de CodeIgniter est très complète et il est important de s'y référer dès que l'on a une fonctionnalité à réaliser, afin de savoir si des outils pouvant nous simplifier la tâche existent déjà.

Nous avons utilisé la dernière version stable de CodeIgniter (3.0). Cette version impose donc l'utilisation d'une version de PHP supérieur à 5.4.

B) Choix technologiques

Bootstrap

Pour la réalisation de nos interfaces graphiques, nous avons choisi d'utiliser le framework Bootstrap. C'est un ensemble de composants HTML, CSS et JS nous permettant de développer

rapidement des pages web élégantes. Un autre avantage de Bootstrap est d'être responsive, c'est-à-dire qu'il s'adapte correctement à la taille de l'appareil servant à visualiser la page. Ce framework nous a permis de développer rapidement des interfaces graphiques fluides et ergonomiques suivant le modèle de notre maquette sans avoir à réécrire a les composants.
La documentation officielle étant très complète et claire, la prise en main de cet outil a donc été très rapide.

Ce framework est utilisé par un très grand nombre de sites web et a initialement été développé par Twitter, avant d'être rendu open source.

Autres librairies

AmCharts : des graphiques pour mieux visualiser

Dans notre application, nous avons inclus un certain nombre de graphiques permettant aux utilisateurs d'avoir une représentation visuelle de leurs données. Pour cela, nous nous sommes appuyés sur la librairie JavaScript « AmCharts ». L'intérêt de cette libraire est principalement la simplicité d'utilisation, en effet, il est possible de concevoir l'apparence du graphique sur un éditeur en ligne et il ne reste plus qu'à intégrer les données au format JSON. AmCharts propose également un grand nombre de graphiques différents.

DataTables

DataTable est une librairie JavaScript permettant de mettre en place très rapidement des tables HTML paginables, triables et chercheables. L'ensemble des données est chargé au premier accès, puis les traitements sont réalisés côté client. Cette librairie a été mise en place sur la page « Enseignants » de l'application. Lorsque nous avons souhaité la mettre en place sur les autres pages, nous avons rencontré une limitation fonctionnelle. En effet, toutes nos tables sur les autres pages utilisent la fonctionnalité JavaScript de Bootstrap « collapse » (permet d'afficher et de masquer le contenu d'une ligne). Ces deux scripts ne sont pas compatibles et compte tenu du temps disponible pour le projet, nous n'avons pas pris le temps de chercher à proposer une amélioration de DataTable. Nous n'avons pas trouvé d'autre librairie JavaScript compatible avec cette fonctionnalité de Bootstrap.

IV) Développement

Au début de notre phase de développement, nous nous sommes concertés afin de mettre en place une architecture générale pour notre application. Cette structure permet d'homogénéiser notre code et d'éviter de dupliquer des actions pouvant être regroupées ou automatisées.

A) Structure générale de l'application

Authentification et héritage de contrôleurs

L'ensemble des pages de notre application nécessite d'être authentifié afin de pouvoir y accéder. Pour cela, nous avons créé un contrôleur général à notre application (Application_controller) héritant de la classe de base de CodeIgniter (CI_Controller). Dans le constructeur de notre contrôleur, nous plaçons le code permettant de vérifier que l'utilisateur tentant d'accéder à la page est bien connecté.

```
class Application_controller extends CI_Controller
{

    public function __construct ()
    {
        parent::__construct();
        $this -> require_login();
    }
```

Chaque contrôleur de notre application doit donc désormais étendre la classe Application_controller et non directement celle fournie par CodeIgniter.

Nous avons appliqué le même principe pour les pages administrateurs.

```
abstract class Admin_controller extends Application_controller
{

    public function __construct ()
    {
        parent::__construct(); // S'occupe de vérifier que l'on est déjà connecté.
        $this -> require_admin_privilege();
    }
```

Les contrôleurs d'administration héritent donc désormais de la classe Admin_controller, qui, héritant elle même de Application_controlleur, se charge en plus de vérifier que l'utilisateur dispose bien des droits d'administration.

B) Headers et barre de navigation

Nous avons rapidement constaté que pour chaque vue à afficher, nous devions charger l'header (chargements des fichiers de styles et de scripts), la barre de navigation (présente sur toute les pages) et le footer. Afin de simplifier le chargement d'une vue, nous avons donc surchargé la classe Loader de CodeIgniter afin d'y rajouter la fonction « template ». Cette fonction permet de demander le chargement d'une vue, sans avoir à se préoccuper des autres fichiers.

```php
class MY_Loader extends CI_Loader
{
    public function template ( $view, $data = array() )
    {
        $this -> view( 'template/header' ); // styles et js
        if ( $this -> session -> userdata( 'me' ) !== NULL ) {
            $me = $this -> session -> userdata( 'me' );
            $this -> view( 'template/navbar' );
        }
        $this -> view( 'template/flash', get_flashs() );
        $this -> view( $view, $data );
        $this -> view( 'template/footer' );
    }

}
```

Messages d'information : les flashs

Nous avons constaté que dans une application web, nous sommes souvent amenés à afficher des messages d'erreur ou de confirmation à l'internaute et que le déroulement est toujours le même

- Ajouter un message et un type (succès, erreur, information, alerte)
- L'afficher dans la vue

Nous avons donc décidé d'écrire un helper permettant d'automatiser ces tâches. Ce nouveau helper est chargé automatiquement par CodeIgniter dans tous nos fichiers. Il permet ainsi d'ajouter un des quatre types de messages, qui seront valables uniquement pour la prochaine requête HTTP (une fois la requête effectuée, le message n'a plus d'existant).

```php
flash_error( "Login et/ou mot de passe incorrect !" );
redirect( 'login', 'refresh' );
```

Nous avons donc ensuite créé une vue qui sera chargée automatiquement sur chaque page, affichant chaque message s'il y en a un. Voici le rendu une fois stylisé avec Bootstrap.

C) Modifications sur la base de données

Comme expliqué précédemment, la structure de la base de données nous a été fournie en début de projet. Nous avions cependant la possibilité d'y effectuer de légères modifications nous permettant de prendre en compte nos fonctionnalités. Nous avons donc modifié :

- La taille du champ 'pwd' dans la table Enseignant. À l'origine les mots de passe de l'application étaient stockés en clair. Nous avons remédié à ce problème en utilisant les fonctions de hashage/salage directement proposées par PHP. De ce fait, il a été nécessaire d'augmenter la taille du champ dans la base, pour permettre un stockage complet de la chaine sécurisée.
- Nous avons modifié la structure de la table 'décharge' afin d'inclure un motif. L'ajout de cette colonne a ainsi permis à un enseignant de disposer d'une ou plusieurs décharges avec des motifs différents. C'est un gain en lisibilité pour les utilisateurs.

- Les modules et les cours sont désormais identifiés en base par un ID auto-incrémenté. Nous avons fait ce choix afin de permettre de modifier le nom, qui servait précédemment d'identifiant. Il devient alors une information d'affichage. La mécanique interne de l'ID est totalement transparente pour l'internaute.

VI) Tests et validation

Une fois l'application avancée, nous avons décidé de prendre du temps pour effectuer des tests afin de vérifier la fiabilité de notre application. Pour cela nous avons utilisé Gremlins et Selenium IDE, deux façons différentes de tester.

A) Gremlins.js

En premier lieu nous avons utilisé un script nommé Gremlin.js. Si le script est activé, ce dernier réalise des actions (clics souris, saisie clavier) totalement aléatoires à une fréquence élevée. Nous l'avons utilisé plusieurs fois, mais un problème revenait de façon récurrente, le script cliquait au bout d'un temps très court sur le bouton "Déconnecté". Donc le test n'était plus intéressant. De plus, aucun contrôle n'est possible.

Le principal avantage de ce script est sa rapidité de mise en œuvre. En effet il ne nécessite aucune intervention humaine pour la mise en place et son exécution (pas de scénarios à prévoir). De ce fait, il ne permet pas de tester efficacement la logique métier de l'application, mais peut cependant permettre de remonter rapidement des erreurs de développement.

B) Sélénium

Sélénium IDE est un outil qui permet de tester l'interface utilisateur des applications web. Sous forme d'extension pour Firefox, il permet d'enregistrer une suite d'actions sur une page web par exemple :

- Ouvrir une page précise,
- Effectuer un click sur un élément de la page,
- Inséré du texte dans un élément...

Les tests créés peuvent être joués individuellement ou à la suite, regroupés sous forme de scénarios qui les enchaînent dans un ordre prédéfini. On peut donc simuler une activité quelconque sur une interface WEB et vérifier le fonctionnement complet de l'application.

Grâce à cet outil, nous avons réalisé des tests, notamment tous les ajouts, les suppressions et les modifications possibles dans notre application web. Dès qu'un ajout est fait, on vérifie si l'élément créé est présent, et inversement si un élément est supprimé (Selenium permet vérifier qu'un élément n'existe pas dans la page). Ces tests nous ont permis de vérifier lors de chaque commis sur l'outil de gestion de version que nous n'avions pas introduit de régressions. Nous avons notamment découvert quelques jours après avoir réalisé la suppression des décharges qu'elle ne fonctionnait finalement pas correctement.

Compte tenu du temps imparti pour ce projet, nous n'avons pas eu le temps de tester en profondeur notre application. Nous nous sommes contentés des tests d'opérations basiques. Développer cet aspect serait une bonne piste d'évolution de l'application. Une autre bonne pratique à adopter serait de réaliser des tests unitaires en PHP afin de vérifier de manière atomique que chaque fonctionnalité se comporte bien comme attendu. Nous n'avons pas eu le temps de monter en compétence sur cet aspect durant le projet.

VI) Bilan de nos réalisations

A) État d'avancement

Points forts

Nous avons réussi à aboutir sur des interfaces graphiques très proches de notre maquette. Nous sommes donc satisfaits d'une part par le rendu obtenu et d'autre part par la fluidité de navigation au sein de notre application.

Le principal point faible de notre application est qu'elle ne dispose pas de solution permettant d'exporter les données au format CSV. En effet, le manque de temps se faisant ressentir (délais imposés et plusieurs autres projets académiques à travailler en parallèle), nous avons fait l'impasse sur cette fonctionnalité.

B) Améliorations possibles

Les améliorations envisageables sont toujours nombreuses :

- Tester plus en profondeur notre application
- Sur les tables « accordéons » d'effectuer un chargement des données à la volée, uniquement lorsque l'on souhaite un affichage. Pour le moment l'intégralité des données (modules et cours) est chargée à la création de la page. C'est un chargement qui peut s'avérer lourd.
- Gérer l'envoi des emails lors de la création de comptes utilisateurs. Nous avons laissé cette fonctionnalité de côté afin d'empêcher l'envoi de spam aux enseignants. PHP inclus une librairie permettant de gérer cela très simplement. Il faut néanmoins disposer d'un serveur mail sur le serveur d'hébergement (ou passer par un service externe).
- Développer des fonctions de recherches plus avancées sur les cours et pourquoi pas de nouvelles statistiques sur un écran réservé au reporting.

Conclusion

Ce projet de technologies web nous a permis de gagner en compétences dans plusieurs domaines. Tout d'abord, nous avons réalisé une étude des besoins client et avons donc constaté que ces besoins peuvent évoluer et ne sont pas forcément toujours très clairs. Il est donc important d'arriver à la rédaction d'un cahier des charges clair et complet, validé par le client, avant de se lancer dans le développement. Nous avons également eu l'occasion d'appréhender le principe des méthodes agiles en gestion de projet. Les itérations rapides permettent un retour client à chaque étape importante du projet afin d'adapter la réalisation au plus tôt en cas de besoin.

D'un point de vue technique, nous avons pu mettre en pratique le modèle MVC à l'aide du langage PHP et du framework CodeIgniter. Il a été important de s'abstraite de l'implémentation proposée afin de bien comprendre le principe du MVC et d'un framework web. Cette montée en compétence nous permettra donc d'être plus rapidement autonomes sur d'autres projets académiques ou professionnels pouvant se dérouler sur d'autres technologies.

Ce projet nous a également permis d'approfondir nos connaissances en programmation web côté client (JavaScript). Nous avons surtout appris à utiliser des librairies déjà existantes et à les intégrer à notre projet.