Rapport projet JEE

Équipe :

Lyz PIAM

Assil MOALIKYAR

Théo Seng

Ilyas El Mouahssine

Organisation

Le front et le back ont été divisés en deux. Théo s'est occupé surtout de l'espace des admins et Assil s'est occupé principalement des professeurs. Lyz s'est occupée du back pour les deux versions.

L'application n'a pas été beaucoup avancée dus à des problèmes d'organisation et de communication. Les membres de l'équipe ont commencé à se synchroniser après la mi-novembre.

Documents de conception

Un MCD a été fait mais les plus à jour sont les diagrammes de classes des modèles et des servlets. Ils sont détaillés dans les fichiers mis sur github (dans le dossier conception). Les particularités suivantes en ressortent :

• la classe abstraite des utilisateurs n'a pas été implémentée mais elle nous a permis d'identifier les points commun entre les admins, les étudiants et les professeurs ;

• les professeurs ont une seule matière.

Une alternative possible pour le projet aurait été d'utiliser une entité SchoolYear représentant les promotions de la manière suivante :

• elle a un nom ;

• elle possède une liste de matières

• elle possède une liste d'étudiants et chaque étudiant est inscrit dans une promotion (pour une année donnée).

Cela aurait notamment permis de gérer plus efficacement le filtrage des cours par matière (car deux promos différentes peuvent avoir des matières ayant le même nom) et aussi de mieux répertorier les professeurs.

Backend du projet

Les étapes suivantes ont défini le cycle de développement de l'application Hibernate :

• création des tables avec la base de données par un fichier sql ;

• création ou modification des entités ;

• création des data access objects (DAO) pour manipuler les tables de la base de données (un par entité) avec l'entity manager pour hibernate ;

• création des repositories ;

• création des des services qui appellent les (DAO) et effectuent quelques vérifications quand nécessaire, un service est associé à un DAO ;

• pour la version hibernate, des tests réalisés avec JUnit tout au long du projet.

L'architecture de la partie backend est semblable à une imbrication de façades (design pattern) : les contrôleurs reposent sur les services qui s'appuient sur les DAO. Cela a permis de grandement simplifier et alléger le code des servlets.

Les conséquences du débogage de la gestion des entités par Entity Manager

Cela a pris beaucoup de temps car ce n'était pas évident de comprendre que pour les entités en relation, les champs de types entité récupérés par les getters ne sont plus dans l'état "managed" mais deviennent "transient". Dès lors, aucune modification apportée ne sera enregistrée dans la base de données. C'est crucial par exemple pour changer un étudiant de promo : son emploi du temps doit être mis à jour, or un moyen efficace de le faire est d'utiliser le getter dédié, il n'y a pas besoin de requête SQL. Par conséquent, pour faciliter le débogage, les retours des opérations de création et de mise à jour ont été changés en String. Les servlets ne peuvent pas recevoir les paramètres à enregistrer dans la session. La communication dans l'équipe ayant été difficile, nous avons compris le soucis trop tard.

Toutefois, le débogage a été un succès car il a permis de suivre le cycle de gestion des entités avec la méthode .contains() de l'EntityManager. Pour corriger complètement le problème du front, il suffit alors d'ajuster les types de retours des créations ou mises à jour d'entité selon le contexte (retour de type Course pour la création d'un compte étudiant) et de laisser la servlet affiché le message approprié selon le résultat de l'opération. Par exemple, en cas de retour valant null, la servlet pourrait afficher un message d'erreur.

Frontend de l'application :

On a décidé de séparer l'application en 3 parties : la partie étudiant, professeur et admin. On a attribué 1 étudiant par partie (Ilyas pour étudiant, Assil pour professeur et Théo pour Admin) avant de se concerter sur comment faire certains types de pages, ce qui fait qu'un temps conséquent à été utilisé pour uniformiser le code et le css (Le css final a été fait par Assil). Ce découpage a aussi mené à de la duplication de code puisqu'il y a des pages assez similaires entre chaque partie même si elles ont des fonctionnalités différentes.

Pour s'identifier, on demande à l'utilisateur de sélectionner s'il est étudiant, professeur ou admin. On le redirige sur la page de login adéquate. L'utilisateur s'identifie avec son mail et son mot de passe. La servlet va traiter l'information : elle vérifie si le mot de passe a minimum 8 caractères et elle vérifie que l'email est sous la forme "prenom.nom@cyu.fr" avec les classes du package java.util.regex.

La partie admin laisse l'utilisateur accéder aux listes d'étudiants, de professeurs, d'admin et de cours. Depuis ces pages, on a accès à formulaire pour créer chacune des entités respectives dans la base de données. Les formulaires utilisent pour ses listes déroulantes des énumérations pour ne pas se tromper lors de la saisie des données et pour permettre de la flexibilité dans le code si la base de données change. Une boucle for sur les Enum.values permet simplement de remplir les <option> d'un <select> sur la jsp.

Nous ne pouvons cependant pas modifier ou supprimer les entités citées dans la base de données même si ces fonctionnalités existent en backend. Le développement du frontend s'est effectué tard dû aux difficultés de configurer hibernate. Des boutons sont présents mais ne redirigent pas vers une jsp laisse le choix de supprimer ou de modifier l'entité choisie. Il aurait fallu que les boutons transmettent l'id de l'entité.

On ne peut aussi pas consulter ou modifier son profil et cela dans chaque partie. Pour faire cela, il faut que la session stocke dans un attribut "mail" l'email de l'utilisateur pour pouvoir l'identifier et récupérer ses données dans la base de données avec les services qui ont déjà été codés. Le transfert de cette information se fait bien entre les servlets de login et les pages sur lesquelles elles redirigent mais de, jsp à jsp, l'attribut "mail" ne persistait pas. Nous avons essayé "session.setAttribute('mail',mail)" et "request.setAttribute('mail',mail)" (avec mail une variable qui stocke l'attribut mail envoyée par la servlet) dans les jsp sources et "session.getAttribute('mail')" et "request.setAttribute('mail')" dans les jsp destinations mais cela n'a pas résolu le problème.

Une grande partie des fonctionnalités des étudiants et des professeurs dépendent de ces informations. Regarder son emploi du temps, regarder ses notes sont inaccessibles pour les étudiants par exemple. La navigation et l'interface de ces deux parties sont présentes mais relier les fonctions du backend avec le frontend n'a pas été réussi.