DELAMOTTE Clément, COMPAGNON-MINARRO Olivier, FRANCHINI Matthias, BELMAHI Zakarya, MCHICH Nada

RAPPORT PROJET JEE

ING2 GSI - Mohammed Haddache

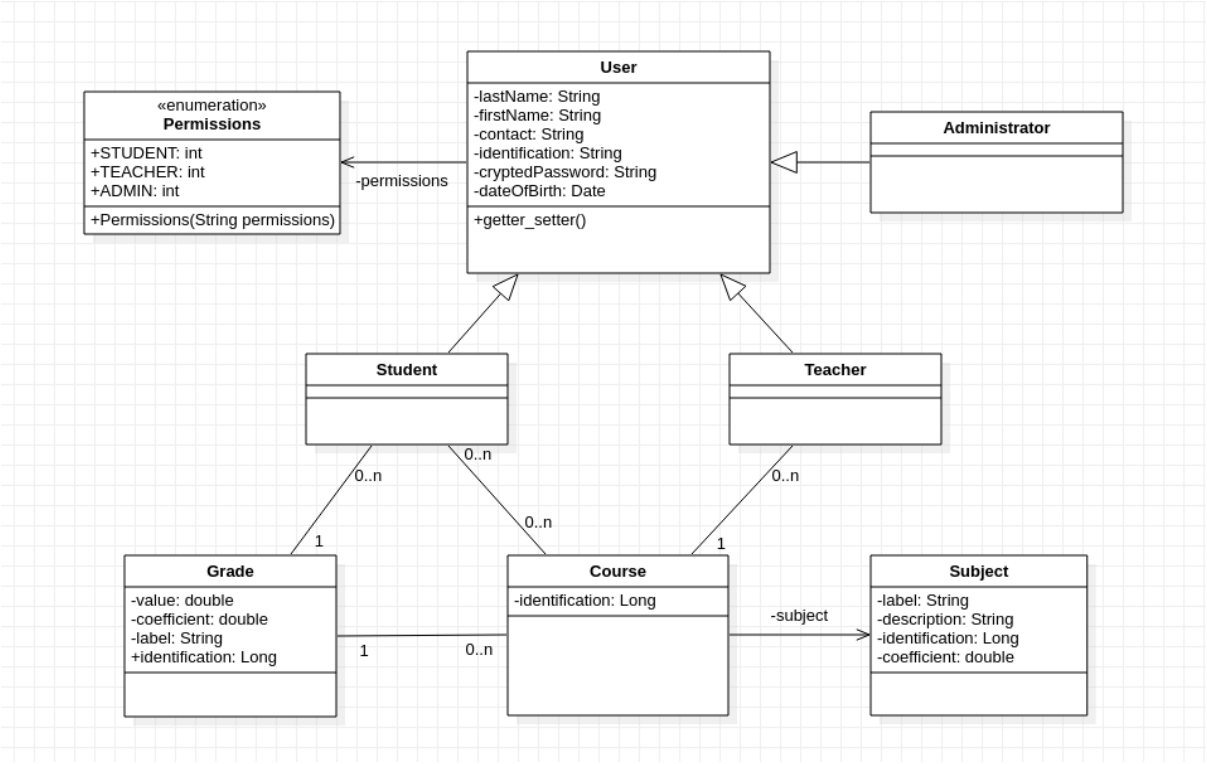
2024 - 2025

Table des matières

I. Conception - MCD

Au commencement du projet, nous avons interprété chacune des contraintes qui étaient imposées afin de construire un diagramme de classe nous permettant de créer un site complet.

Le diagramme de classe :



Explication des classes:

- **User**: Cette classe représente un utilisateur, sa particularité étant qu'elle possède un attribut permissions qui définit et contrôle l'accès d'un utilisateur sur le site. Ici l'attribut est simple puisque les rôles ne sont pas nombreux mais on avait aussi pensé à en faire une collection en fonction de ce qu'un utilisateur a le droit de faire ou non.
- **Permissions**: L'énumération qui définit les actions disponibles aux utilisateurs dont chacune des valeurs est associée à une chaîne de caractères pour faciliter l'affichage sur les différentes pages.
- **Administrator**: Il n'y a pas grand chose à dire si ce n'est que cette classe est présente pour faciliter les requêtes SQL/HQL puisque l'on peut directement faire appel à la classe pour récupérer les administrateurs. De même pour Student et Teacher mais ces classes ont elles d'autres caractéristiques et étaient donc nécessaires.
- **Student**: Un étudiant a ici plusieurs notes et cours en plus de ces attributs hérités de la classe User ce qui est pratique lorsqu'un étudiant est connecté pour l'affichage de ces cours et notes.

- **Teacher**: Un enseignant peut avoir plusieurs cours ce qui est aussi pratique pour afficher les cours qui lui sont associés sur les différentes pages jsp.
- **Subject**: Cette classe représente une matière similaire à celle que l'on retrouve dans nos UE pour chaque semestre. Elle a un libellé, une description, un coefficient qui est utile pour générer la moyenne.
- **Grade**: Une note est associée à un cours et un étudiant mais pour une combinaison des deux, plusieurs notes peuvent exister ce que nous avons pris en compte au fur et à mesure de l'implémentation.
  Originellement, l'identifiant d'un cours et d'un étudiant représentaient la clé primaire dans la base de donnée mais on s'est dit que l'on pouvait avoir plusieurs notes pour un même cours comme avec des partiels couplé à du contrôle continu.
  Evidemment, elle possède un libellé, un coefficient ainsi qu'une valeur pour gérer les différents calculs et l'affichage.
- **Course**: Un cours peut être associé à un unique professeur mais avoir plusieurs étudiants. Il représente aussi concrètement une matière pour les étudiants et c'est les notes associées à ce cours qui comptent pour le calcul de la moyenne.

II. Implementation

Hibernate

Le modèle et la BDD:

On a commencé l'implémentation en faisant les différentes classes pour représenter le modèle en accord avec le diagramme (qui a eu plusieurs versions au fil du développement).

Pour ce qui est de la base de donnée, nous avons choisi de rajouter sur chacune des classes du modèle des annotations Hibernate afin de pouvoir générer la base de données de manière automatique ce qui nous a fait gagner un peu de temps supplémentaire pour travailler sur les vues et contrôleurs.

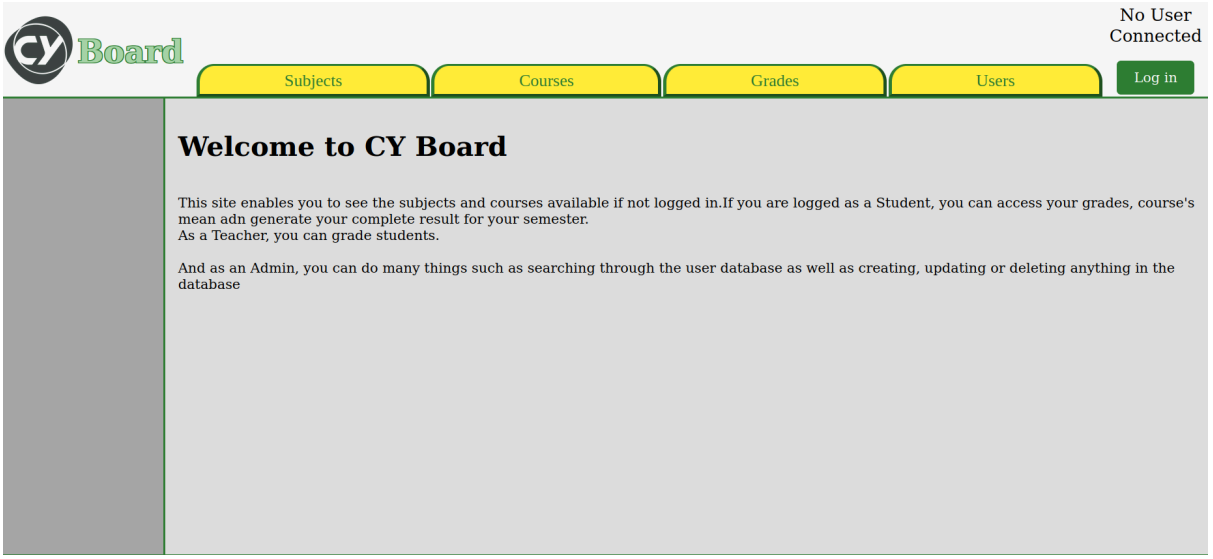
Bien sûr, nous avons quand même regardé la manière dont avait été générée la base de données afin de reproduire un fichier sql pour la création de celle-ci et un autre pour y insérer des valeurs par défaut pour effectuer des tests.

Maintenant que le travail de base a été effectué, nous pouvons nous concentrer sur la construction des vues et des contrôleurs.

Les vues:

L'architecture du site a été la première étape puisqu'il s'agissait de savoir comment afficher les enregistrements de la base de données et effectuer les opérations spécifiques comme la création, la modification et la suppression d'enregistrements.

Voici la disposition que nous avons choisi pour la page d'accueil
Index.jsp:



Pour rester bref:

- En haut à gauche se trouve le lien pour revenir vers la page principale
- La barre de gauche contient les options
- La barre du haut représente les différentes catégories accessibles aux utilisateurs
- Le bouton en haut à droite permet de se connecter / déconnecter. On peut aussi voir son identifiant juste au dessus si l'on est connecté
- L'élément central va permettre l'affichage pour les opérations que l'on veut effectuer

Parmi toutes les catégories, voici ce qu'il est possible de faire en prenant en compte vos permissions et données personnelles:

- **Subjects**: Voir toutes les matières proposées par l'école, voir les cours associés pour une matière, les opérations CRUD (admin seulement)
- **Courses**: Voir tous les cours accessibles (ou ceux auxquels l'utilisateur n'est pas inscrit), s'inscrire à un cours (étudiant seulement), les opérations CRUD (admin seulement)
- **Grades**: Voir toutes les notes (admin seulement), voir ces notes et générer son relevé de notes (étudiant seulement), noter les étudiants (enseignant seulement)
- **Users** (catégorie admin): Voir les utilisateurs, les opérations CRUD ainsi que le filtrage avec recherche

Les options disponibles pour ces différentes catégories seront affichées dynamiquement dans la barre de gauche et ceci en fonction de vos permissions.

Avant de passer à l'explication des contrôleurs, il nous faut vous expliquer la manière qu'on a utilisé pour transmettre les informations correctement sans conflit entre certaines opérations.

Tout d'abord, pour la majorité des affichages, chaque enregistrement de la base de donnée est représenté par un bouton radio caché pour permettre de ne sélectionner qu'un seul

élément pour effectuer des opérations sur celui-ci comme les opérations CRUD pour les administrateurs. Nous utilisons aussi des mot-clés spécifiques dans les formulaires pour que le traitement se fasse correctement au niveau des servlets.

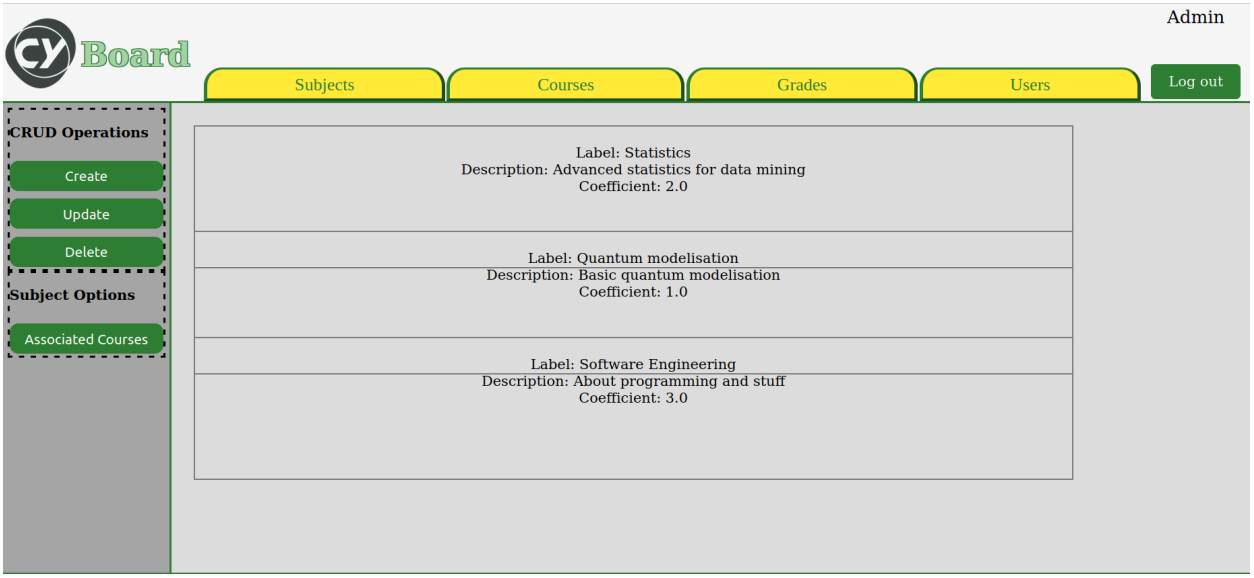
Voici la liste des mot-clés:

- **action**: Type d'opération à effectuer sur les données ou la vue.
- **option**: Indique un choix menant à une potentielle action. Dans certains cas, l'option modifie la vue directement pour permettre certaines actions.
- **categorie**: Représente la page mais aussi le contrôleur qui va recevoir les requêtes.
- **selected[*]** : L'objet à récupérer pour effectuer des actions, comme *selectedSubject* par exemple.
- **[*]s**: Récupération de tous les enregistrements pour une classe / catégorie, comme *subjects* par exemple.
- **loggedUser**: Représente la session de l'utilisateur et ses données personnelles.
- **error**: Indique une erreur dans l'exécution d'une requête.

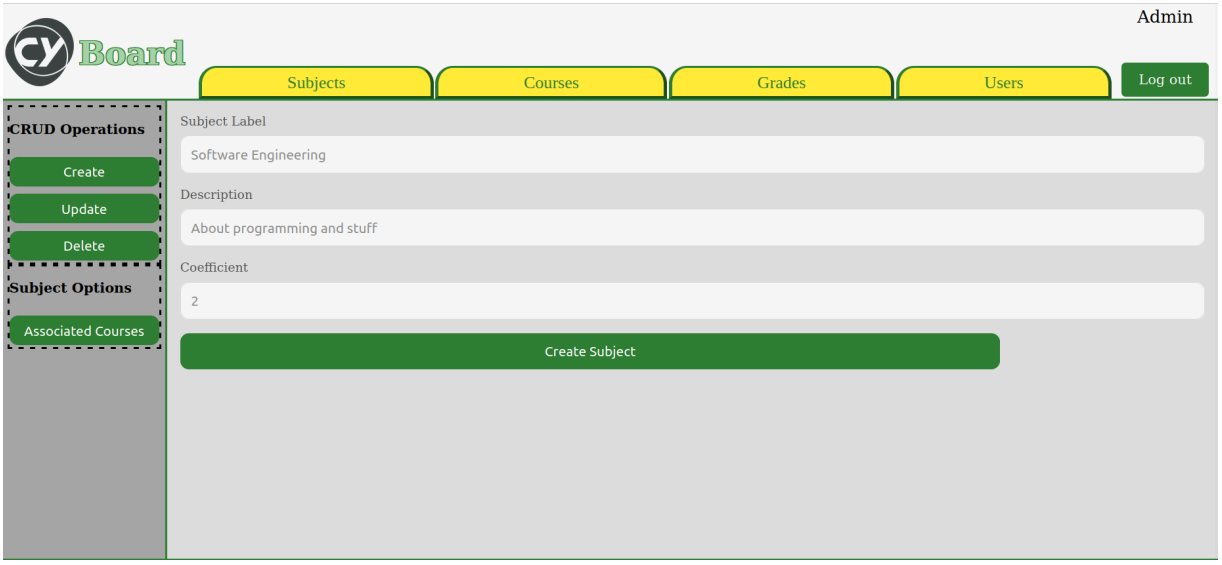
Il existe d'autres mot-clés mais ceux-ci sont beaucoup moins prévalant dans le code (comme search pour les recherches ou les noms d'attributs des classes dans les opérations CRUD)

Avant de passer aux contrôleurs, voici un aperçu de ce qui a été expliqué jusqu'à présent avec la catégorie Subjects.

<u>subject.jsp:</u>



subject.jsp avec l'option "create" :



Avant de passer aux contrôleurs, il nous semble important de mentionner aussi le fait que chacune des pages en jsp sont factorisées / fragmentées et font donc appel à des balises de type jsp:include. L'objectif est d'éviter la duplication de code ce qui simplifie les modifications, la lecture et pour vous la notation du code des pages.

De plus, la plupart des fragments sont affichés en fonction de ce qui est renvoyé par les différents contrôleurs grâce à l'usage des paramètres et attributs des requêtes dont les mot-clés sont ceux cités plus haut pour pouvoir y avoir accès facilement sans surcharger les requêtes de mot-clés.

Les contrôleurs:

Pour parler des contrôleurs, il faut évidemment commencer par décrire la manière dont ils communiquent entre eux. Tout d'abord, nous avons un *FrontController* dont le rôle est de recevoir chacune des requêtes envoyées par l'utilisateur et de les renvoyer au contrôleur le plus cohérent et adéquat pour répondre à celles-ci.

Listes des contrôleurs:

- **LoginController**: Gère les requêtes liées à la connexion d'un utilisateur.
- **UserController**: Gère les requêtes liées aux opérations sur les utilisateurs comme la connexion pour vérifier le couple identifiant et mot de passe, la récupération d'un utilisateur avec ses cours associés (si étudiant ou enseignant) et pour faire une recherche avec des filtres par exemple.
- **SubjectController**: Gère les requêtes liées aux matières et leurs cours associés.
- **CourseController**: Gère les requêtes liées aux différents cours, leurs notes associées ainsi que la liste des étudiants associés à un cours.
- **GradeController**: Gère les requêtes liées aux notes comme pour celles d'un étudiant, lorsqu'un enseignant veut noter ces étudiants pour un cours et d'autres.

Evidemment, il ne s'agit ici que d'une description basique de ce que font les contrôleurs puisque ceux-ci peuvent dans certains cas faire appel à d'autres méthodes de contrôleurs. En effet, chaque contrôleur gère les méthodes reliées à la classe qu'il représente toujours dans cette optique de cohérence vis-à-vis des opérations effectuables par chacune de nos classes.

Ainsi si une nouvelle composante devait s'ajouter à notre modèle alors nous aurions que peu de mal à l'implémenter. Il suffirait de rajouter la classe dans le modèle, ajouter un contrôleur pour celle-ci dans le package correspondant et de définir des méthodes pour manipuler cette nouvelle classe.

Une étape cruciale pour effectuer ces opérations est de bien vérifier si chacune des composantes que l'on reçoit dans nos requêtes pour des méthodes de création, modification ou de suppression sont bien valides et ne viennent pas causer de problèmes dans les appels à la base de données.

Pour valider les différents paramètres des formulaires et requêtes, nous avons créé un singleton nommé ***ModelValidator*** qui nous permet de vérifier que chaque String que l'on récupère suit un schéma particulier et est non nul (ou composé uniquement de caractères vides, notamment pour des String représentant des caractères non numériques).

Spring Boot

Ayant presque compléter le projet *Hibernate*, nous sommes donc passés sur *Spring Boot* afin de gagner du temps sur l'implémentation de celui-ci.

Spring Boot est un framework cherchant à automatiser et simplifier les principales fonctions d'Hibernate. Le principe est de pouvoir facilement créer, lier et utiliser les méthodes CRUD sur les différentes composantes d'une base de données d'un projet.

De plus, les différentes pages et méthodes peuvent être facilement reliés grâce à leur assignation à un url. Les attributs sont également récupérables comme variable directement dans les pages web. Ainsi, *Spring Boot* facilite l'échange de données et les appels de méthodes et variables.

Pour adapter notre projet d'*Hibernate* à *Spring Boot*, la théorie est plutôt simple :

Il suffit de démarrer un projet *Spring Boot*, de le paramétrer correctement (de la même manière qu'*Hibernate* : on précise par exemple quelle est la base de données à utiliser).

Fichier de configuration de spring boot (appelé application.properties) :

```
spring.application.name=SpringBootMarche
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
spring.datasource.url=jdbc:mysql://localhost:3306/bdd
spring.datasource.username=root
spring.datasource.password=mdp
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.transaction.jta.platform=null
spring.jpa.properties.hibernate.transaction.coordinator_class=org.hibernate.transaction.JDBCTransactionFactory
```

Ici, les requêtes SQL ont été paramétrées afin d'être affichées en détail dans le terminal. Les options correspondantes ne sont donc pas nécessaires. Le dossier des pages jsp ainsi que l'extension .jsp sont précisés à *Spring Boot*. Le dossier utilisé est donc /WEB-INF/jsp/ contenu dans webapp.

On importe ensuite le projet créé sous Hibernate et on peut directement récupérer toutes les jsp, controllers etc. On crée pour chaque Bean que l'on à besoin d'appeler dans notre base de donnée un « Repository » de JpaRepository qui sert à appeler les différentes méthodes de gestion de BDD que nous voulons.

Pour cela, il suffit d'y écrire comme nom de méthode la requête à effectuer. Par exemple, dans userRepository, la méthode:

```
User findUserByIdentification(String identification);
```

Elle permet d'obtenir un objet User avec son identifiant sans écrire une requête MySQL ou HQL complète.

On adapte ensuite les controllers créés au préalable sous Hibernate avec les nouvelles méthodes offertes par les repository. Il faut donc associer aux méthodes répondant aux pages jsp un url ou mapping.

Le *FrontController* du projet hibernate n'est plus utile car les liens des pages jsp emmènent directement à la méthode associée. Pour charger une page ou une redirection, le nom du fichier à charger doit être renvoyé par la méthode appelée. Ainsi, il est plus facile avec *Spring Boot* de dédier des url à une action.

Les paramètres sont alors définis pour chaque méthode de mappage différent et les attributs nécessaires dans les pages jsp sont à attribuer dans les controllers correspondants. La méthode de gestion de page dans les controllers est similaire à hibernate.

III. Conclusion

Ce projet à été l'occasion de grandement améliorer nos compétences en ce qui concerne la maîtrise du JEE nécessaire à la création d'un projet web. En effet, nous avons été confrontés à de nombreux problèmes techniques que nous ne savions pas résoudre et nous avons dû chercher ensemble des solutions à ceux-ci. De plus, nous avons aussi gagné en autonomie et nous saisissons mieux l'organisation d'un projet informatique de long terme.