Rapport Projet JEE

Mohamed HADDACHE

Quentin JEAN, Killian JONNEAUX, Farah MAHMOUD, Esteban VELA, Tom LACHENAUD ZIMMERMANN

ING2 GSI1

Sommaire

Introduction

L’objectif principal de ce projet de JEE a été de développer une application web de gestion de scolarité permettant aux administrateurs, enseignant et étudiants de gérer efficacement les informations académiques. Le projet respecte les contraintes d’architectures Modèle Vue Contrôleur et exploite les technologies suivantes :

- Java pour le backend, HTML, CSS et JavaScript pour l’interface utilisateur.
- Spring Boot, qui permet de faciliter la création de l'application en fournissant des outils puissants pour gérer les aspects comme les requêtes HTTP, la gestion des données et la sécurité.
- Hibernate, un outil de mappage objet-relationnel qui facilite la gestion des données en base de données.
- Apache Tomcat, qui est utilisé pour déployer l’application web.
- MySQL ou PostgreSQL, qui sont utilisés pour stocker les données relatives aux étudiants, enseignants, cours, inscriptions et résultats.

Le but de l'application est de centraliser la gestion des activités scolaires à travers une interface web, accessible aux différents types d'utilisateurs, avec des accès et des fonctionnalités différenciés selon le rôle de chaque utilisateur. En résumé, l'application doit permettre aux étudiants de suivre leurs inscriptions et résultats, aux enseignants de saisir les notes et gérer leurs cours, et aux administrateurs de gérer l'ensemble des informations et de superviser le bon fonctionnement du système.

Architecture du Projet

Le projet vient suivre une architecture structurée et modulaire grâce à l'utilisation du Modèle Vue Contrôleur.

Modèle :

- Etudiant : la classe Etudiant va venir stocker toutes les informations relatives à la création et la gestion d'un étudiant (nom, prénom, id, date de naissance, etc.).
- Enseignant : la classe Enseignant, comme Etudiant rassemble tout ce qui est nécessaire à la création et la gestion d'un enseignant.
- Cours : la classe Cours contient les informations sur les cours. Elle va permettre de créer les différents cours.
- Resultat : la classe Resultat va permettre de stocker les informations relatives aux notes et les résultats académiques des étudiants. Cette classe va notamment contenir la note, l'étudiant auquel on doit attribuer le résultat ou encore le cours associé.
- Admin : la classe Admin, tout comme les classes Etudiant et Enseignant, viens gérer le 3ème rôle disponible dans cette application.

Vue :

Les interfaces utilisateur sont réalisées en JSP et vont permettre l'affichages des différents éléments de l'application :

- Affichage des listes : les fichiers ayant « list » dans leur nom vont permettre d'afficher la liste des objets correspondants. Par exemple, le fichier *listEnseigant.jsp* va permettre, dans un tableau, d'afficher chaque enseignant ayant était enregistrer dans la base de données précédemment. Pour chaque enseignant, nous aurons d'afficher le nom, prénom et le contact.
- Détails : les fichiers ayant « detail » dans leur nom vont permettre, une fois sélectionner, d'afficher les détails d'un objet. Reprenons notre exemple sur les enseignants, lorsque nous avons la liste des enseignant, nous pouvons choisir d'afficher le détail d'un enseignant en particulier grâce au fichier *detailEnseigant.jsp*. Il sera donc possible d'accéder à toutes les informations relatives à un enseignant en particulier.
- Formulaires d'ajout/modification : les fichiers « ajouter » et « modifier » vont permettre de pouvoir ajouter ou modifier des informations. Par exemple, si l'on décide d'ajouter un nouvelle enseignant à la liste des enseignant, il faudra utiliser le fichier *ajouterEnseignant.jsp.* De la même façon, pour modifier un enseignant déjà créé, il faudra utiliser le fichier *modifierEnseignant.jsp.*
- Pages d'authentification : les fichier *login.jsp* et *auth.jsp* vont permettre la connexion à l'application via un compte utilisateur déjà existant.
- *mailForm* va permettre d'envoyer des mails. *Releve* permet d'afficher le relevé de notes. *Menu* permet d'afficher un menu présent sur chaque page facilitant la navigation sur l'application.

Contrôleur

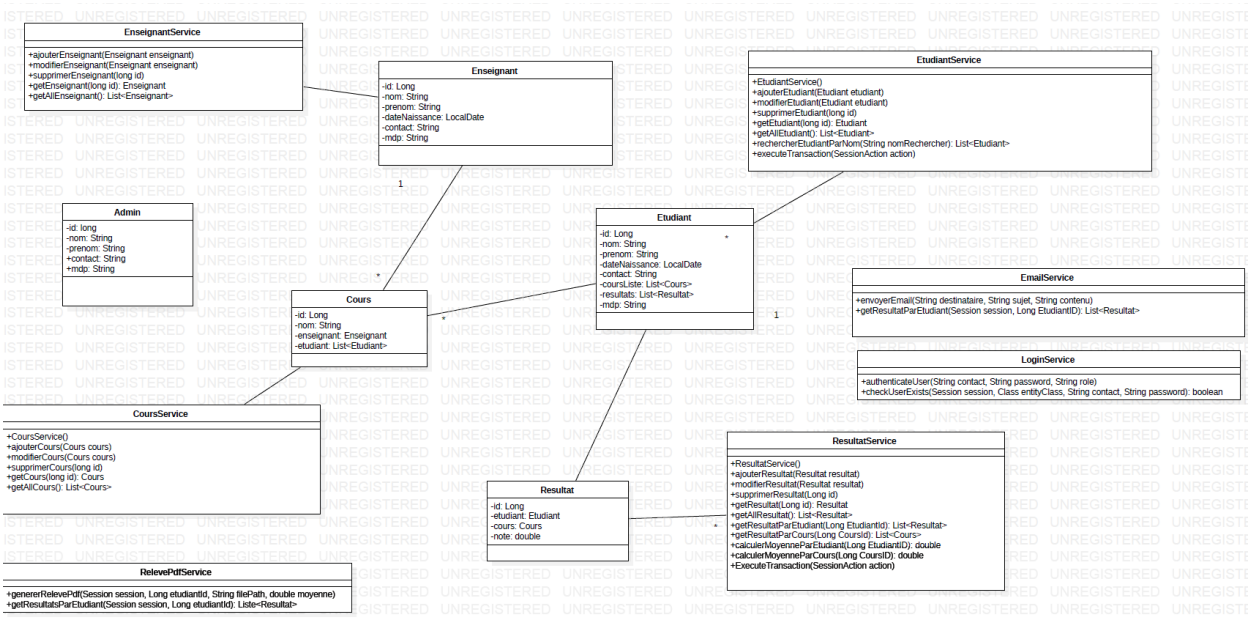
Les servlets vont venir gérer les requêtes http et coordonnent la logique métier.

- CoursServlet : la classe CoursServlet va rassembler toutes les méthodes permettant de gérer les différentes fonctionnalités relatives aux cours (ajouterCours, afficherCours, etc.) mais aussi de gérer toutes les requêtes HTML.
- EnseignantServlet : la classe EnseignantServlet, comme la classe CoursServlet, va venir rassembler les différentes méthodes nécessaires au bon fonctionnement de la logique métier de la classe Enseignant. Nous aurons ici des méthodes comme modifierEnseignant, supprimerEnseignant, etc.
- EtudiantServlet : cette classe, tout comme la classe précédente va reprendre le même fonctionnement dans les méthodes puisqu'il s'agit des méthodes classiques afin de gérer une personne.
- LoginServlet / LogoutServet : la classe LoginServlet/LogoutServlet vont permettre la redirection de l'utilisateur vers la page correspondant à son rôle après avoir rentré ses identifiants ou alors de se déconnecter.
- ResultatServlet : cette classe, comme les autres Servlet viens rassembler les méthodes nécessaires au bon fonctionnement de la gestion des résultats.
- RelevePdfServlet : cette classe va permettre de rassembler les méthodes permettant la création d'un relevé de note en pdf.

Schéma de Conception (MCD)

Afin de représenter les relations entre les entités principales, nous avons conçu un diagramme UML. Au début du projet, nous avions réaliser un schéma de ce à quoi pourrait ressembler le diagramme de classe afin de nous aider dans le lancement du développement de l'application. Au fur et à mesure de la création des différentes fonctionnalités, le diagramme s'est complété et nous avons finalement réussie à obtenir cette version finale.

Il est important de noter que sur le diagramme, par soucis de lisibilité, les méthodes GETTER et SETTER ainsi que les différents constructeurs n'apparaissent pas.



Fonctionnalités Développées

Gestion des étudiants :

- Ajout, modification, suppression et consultation des informations des étudiants.
- Recherche et filtrage selon différents critères.

Gestion des enseignants :

- Ajout et gestion des enseignants.
- Affectation des enseignants aux cours.

Gestion des cours :

- Création, mise à jour et suppression des cours.
- Attribution des étudiants et enseignants aux cours.

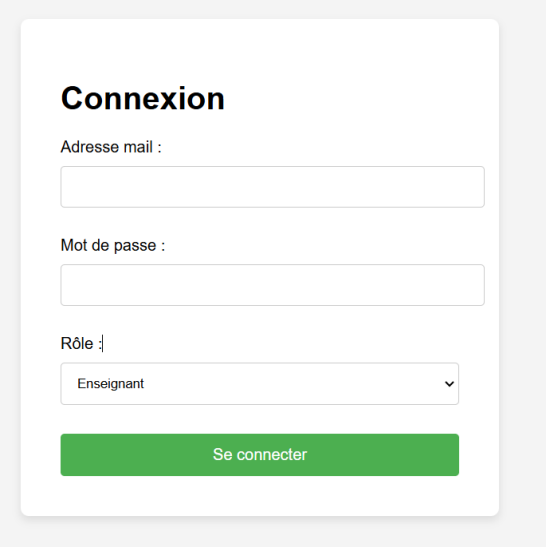
Saisie des Notes et Résultats :

- Les enseignants peuvent saisir les notes.
- Les étudiants consultent leurs relevés de notes.

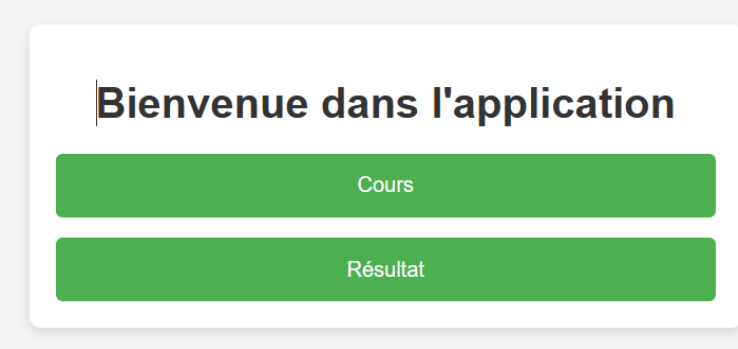
Authentification et Autorisation :

- Différents rôles : étudiants, enseignant, administrateur.
- Sécurisation des accès selon les rôles.

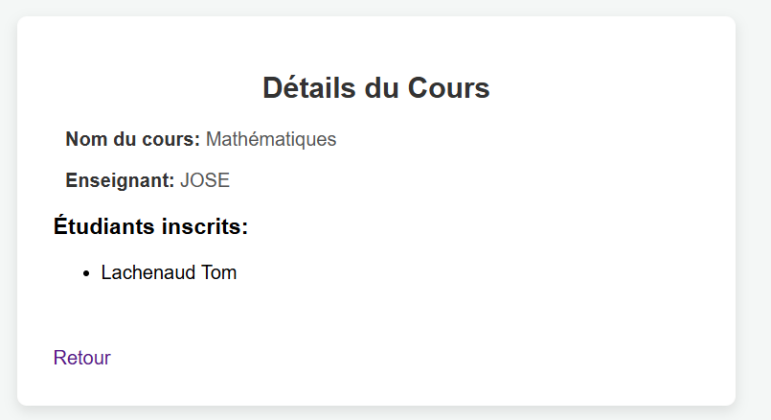
Fonctionnement de l’application



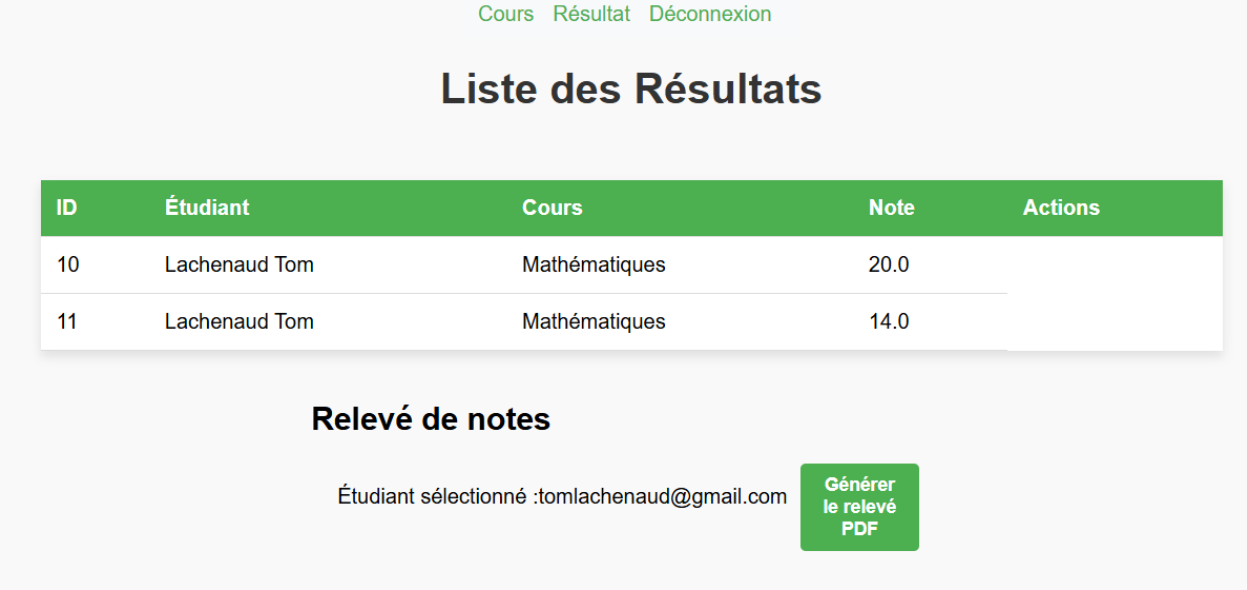
Menu accessible via le compte étudiant

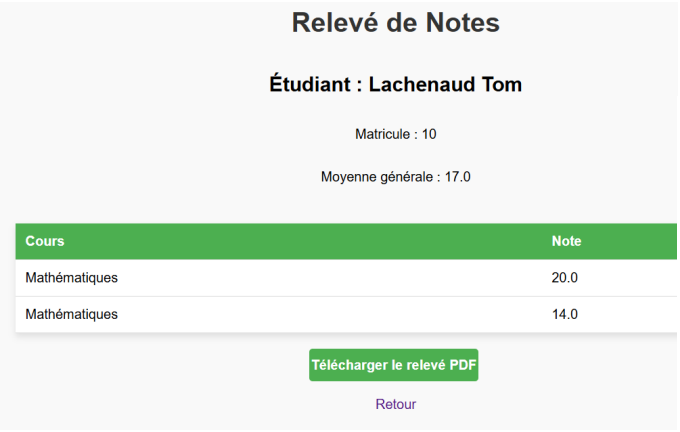


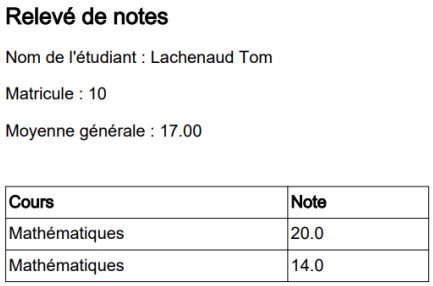
Interface permettant à l’étudiant de visualiser les différents cours auxquels il est inscrit ainsi que le professeur enseignant la matière et les détails de chaque cours



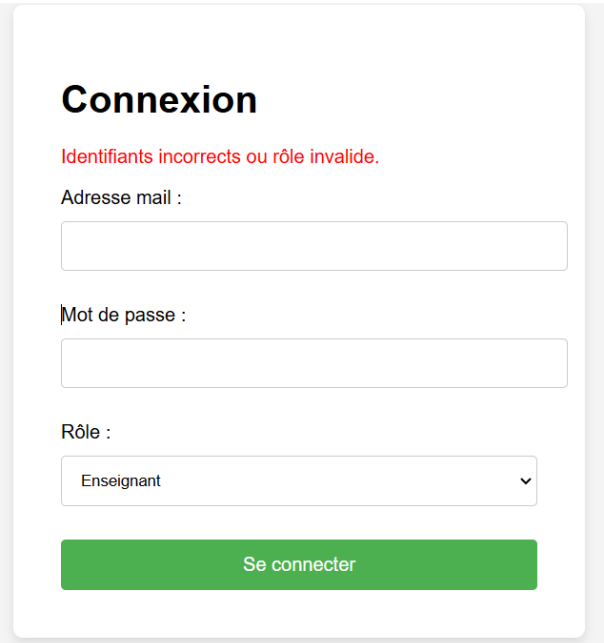
Interface permettant aux étudiants de voir les résultats des étudiants ainsi que d'accéder a leur relevé de notes sur l'application mais aussi via un PDF (3ème images) :

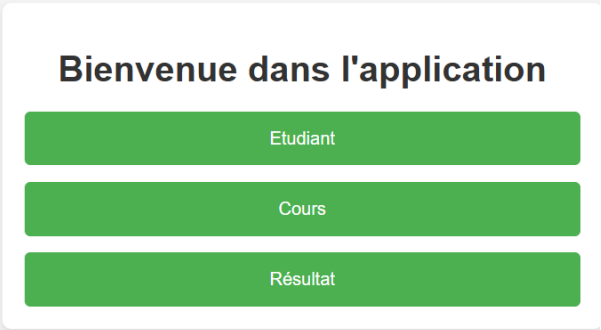




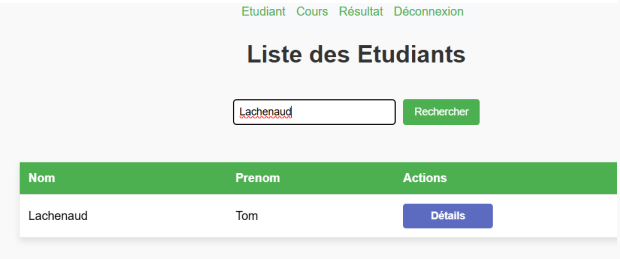


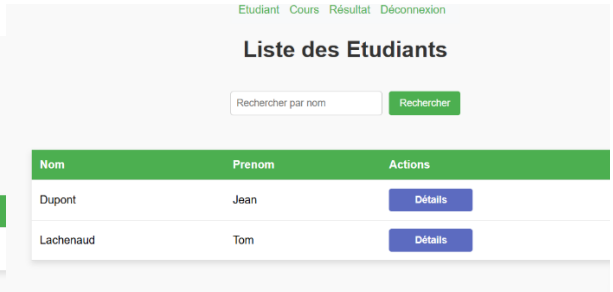
Interface de connexion lorsqu'il y a un problème d'authentification (à gauche) et interface du menu Enseignant (à droite) :

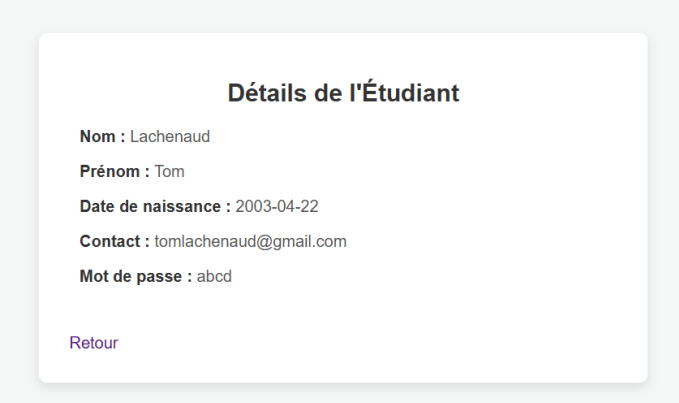




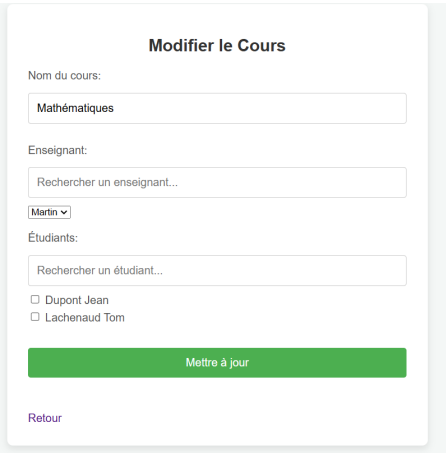
Interface permettant aux enseignant d'avoir un aperçut de tous les étudiants, de pouvoir rechercher un étudiant par son nom ou alors d'obtenir les détails d'un étudiant :

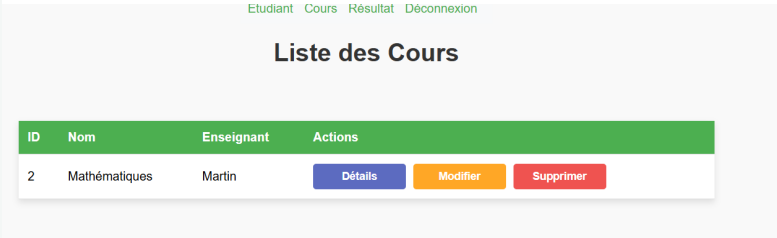




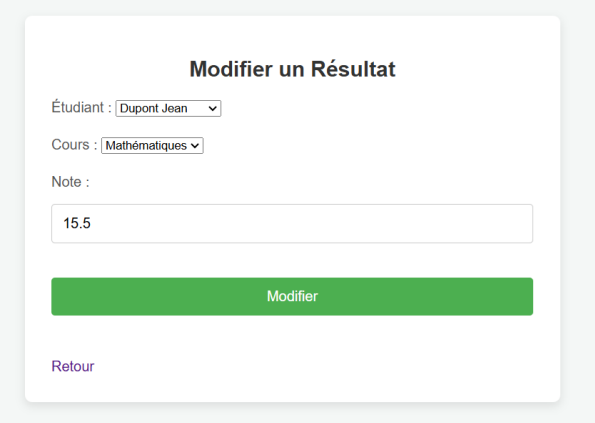


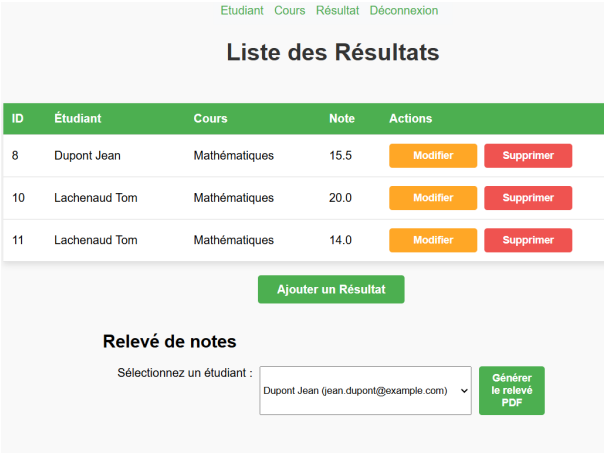
Interface permettant à l'enseignant de visualiser les cours ainsi que de savoir quel sont les étudiants inscrit à ce cours mais aussi de modifier, supprimer leurs cours :





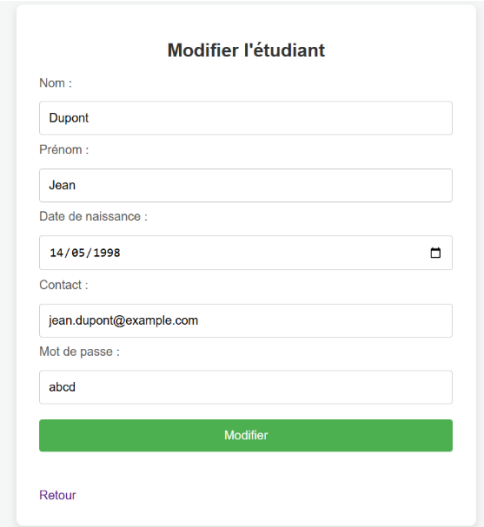
Interfaces permettant aux enseignant de modifier, d'ajouter et de visualiser les résultats de chaque étudiant :

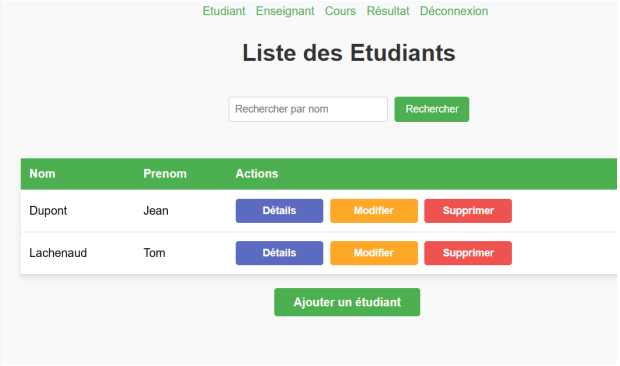


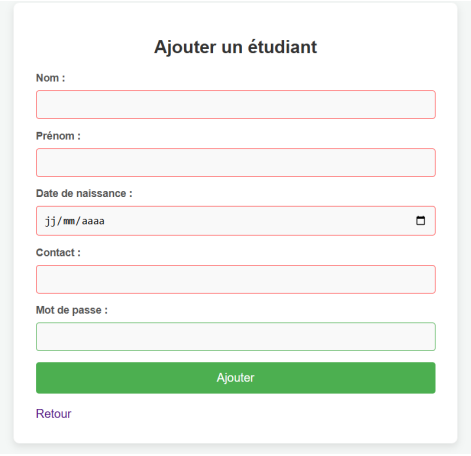


Menu de connexion de l'administrateur :

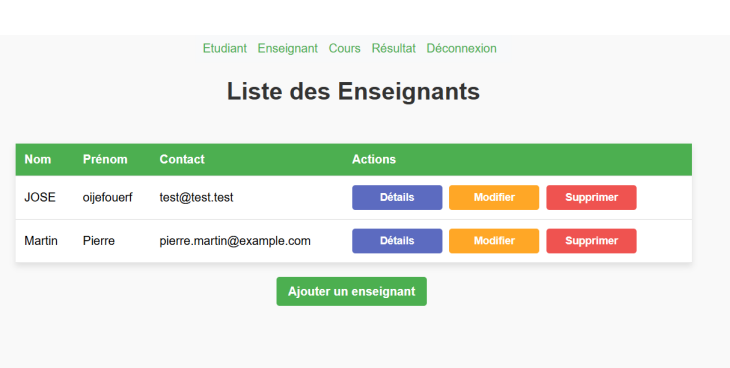
Interface à l'administrateur d'avoir une liste de chaque étudiant afin de pouvoir modifier leurs informations, de les supprimer ou alors d'ajouter des nouveaux étudiants :

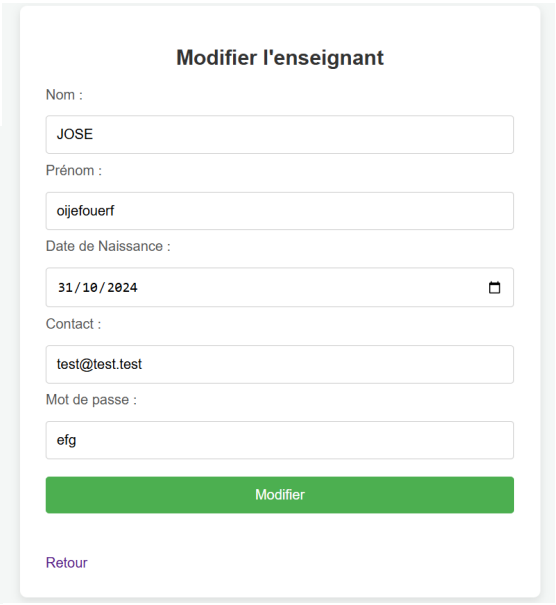




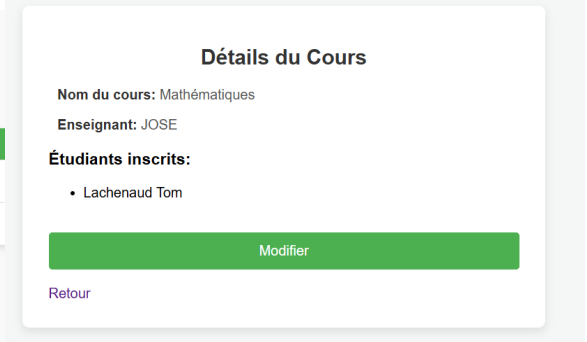


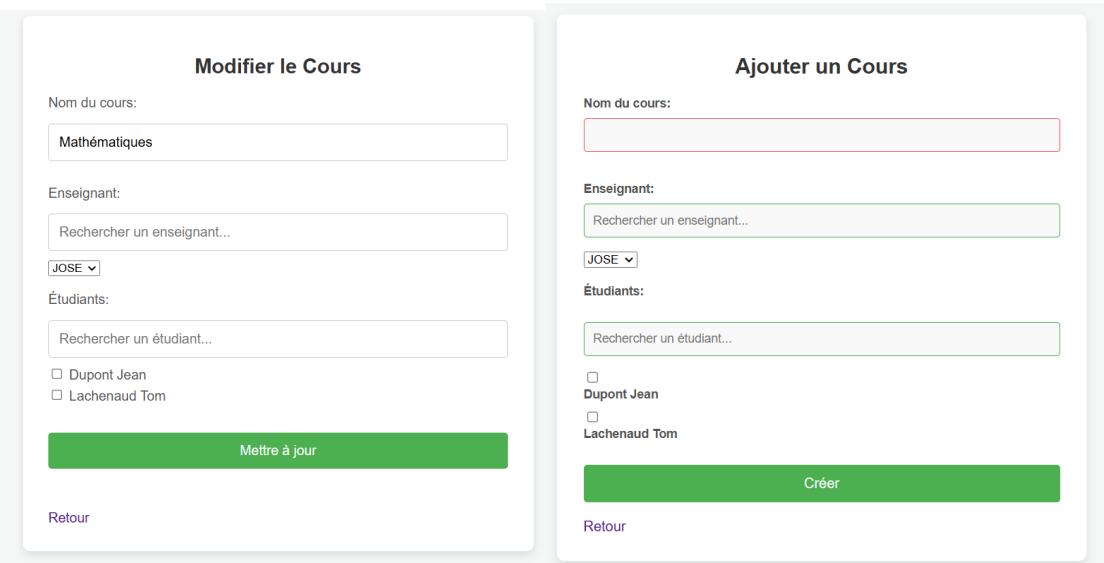
Interface permettant à l'administrateur de visualiser une liste d'enseignant afin d'obtenir leurs détails, de les modifier, de les supprimer ou alors d'ajouter des nouveaux enseignants :





<u>Interface permettant à l'administrateur de visualiser la liste des cours, les détails de chaque cours, modifier des cours, supprimer un cours ou alors d'ajouter un cours en rentrant l'enseignant et les étudiants suivant ce cours :</u>





Voici un mail type qu'un étudiant recevra lorsqu'il sera inscrit dans notre université :

Voici les mails type que les étudiants recevrons lorsqu'une modification aura lieu concernant leurs notes :

L'administrateur possède la même interface que l'enseignant concernant la liste des résultats.

Application Springboot

L'intégration de Spring Boot dans ce projet de gestion de scolarité avait pour objectif de simplifier le développement tout en assurant une structure robuste et maintenable. Contrairement à Jakarta EE, qui repose principalement sur des Servlets et JSP, Spring Boot offre une approche moderne basée sur la convention plutôt que la configuration, intégrant nativement les meilleures pratiques et un large écosystème de bibliothèques.

Le projet a été initialisé en utilisant Spring Initializr, permettant une configuration rapide et personnalisée des dépendances nécessaires, telles que Spring Web pour les API REST, Spring Data JPA pour la gestion de la persistance des données, et Thymeleaf comme moteur de templates pour les interfaces utilisateur dynamiques.

Cependant, malgré ces intentions, nous n'avons pas pu exécuter le projet en raison de problèmes de résolution des dépendances Maven, notamment liés à des incompatibilités entre certaines versions de bibliothèques utilisées dans le projet. Ces difficultés ont freiné la mise en place effective de Spring Boot et soulignent l'importance d'une gestion rigoureuse des dépendances. Cela constitue une piste d'amélioration majeure pour les futures itérations du projet, notamment par une vérification approfondie des versions des dépendances et leur compatibilité avec Spring Boot.

Néanmoins, l'architecture théorique du projet respecte le modèle MVC (Modèle-Vue-Contrôleur), tout en tirant parti des annotations puissantes offertes par Spring Boot :

- **Modèle :** Les entités (Étudiant, Enseignant, Cours, etc.) sont définies comme classes Java annotées avec @Entity. Les relations entre ces unités sont modélisées à l'aide d'annotations JPA telles que @OneToMany et @ManyToOne.
- **Contrôleur :** Les endpoints RESTful, responsables de gérer les requêtes HTTP, sont implémentés dans des classes annotées avec @RestController et organisés par fonctionnalités.
- **Vue :** Les pages utilisateur sont conçues avec Thymeleaf, offrant une meilleure intégration avec les données back-end par rapport aux fichiers JSP qui ont été transformés en fichiers HTML.

Spring Data JPA, malgré les problèmes rencontrés, reste un choix pertinent pour simplifier la persistance des données. Les interfaces telles que *JpaRepository* permettent d'effectuer des opérations CRUD de manière automatique, réduisant la quantité de code nécessaire. L'intégration avec une base de données relationnelle, comme MySQL, est configurée dans le fichier *application.properties*.

Malgré les obstacles techniques, l'utilisation de Spring Boot demeure prometteuse pour de futures améliorations, grâce à :

- Une configuration simplifiée grâce à l'approche « opinionated defaults ».
- Une meilleure gestion des dépendances et une compatibilité accrue avec des outils modernes.

Conclusion

Ce projet a été une opportunité enrichissante pour mettre en pratique les concepts et technologies enseignés au cours de la formation. En abordant à la fois Jakarta EE et Spring Boot, nous avons exploré deux approches différentes de développement d'applications web, chacune offrant des avantages spécifiques.

Le passage à Spring Boot avait pour objectif de moderniser et d'optimiser l'architecture de l'application grâce à ses outils intégrés et sa gestion simplifiée des dépendances. Cependant, des problèmes de résolution de dépendances Maven ont freiné la mise en œuvre complète de cette transition, mettant en lumière la nécessité d'une meilleure gestion des versions et de la compatibilité des bibliothèques. Malgré ces difficultés, l'approche convention-over-configuration et l'utilisation d'outils puissants comme Spring Data JPA et Spring Security restent des solutions prometteuses pour réduire la complexité du développement tout en garantissant la sécurité, la maintenabilité et la scalabilité du projet dans ses futures itérations.

L'application finale remplit les exigences fonctionnelles et techniques du cahier des charges : elle offre une interface utilisateur intuitive et des fonctionnalités robustes adaptées aux différents rôles (administrateurs, enseignants et étudiants). Les processus de gestion des étudiants, des enseignants, des cours et des résultats ont été intégrés efficacement, tandis que les mécanismes d'authentification et de notification garantissent une expérience utilisateur fluide et sécurisée.

Cependant il existe encore des pistes d'améliorations au projet puisque à aujourd'hui, les enseignants ont accès aux notes de tout les élèves même ceux n'étant pas dans leur cours. Ceci est dû à la construction de la base de données qui ne relie pas la table résultats et enseignants. Nous aurions donc dû créer un lien entre les deux pour que ceci soit résolu.

En conclusion, ce projet a non seulement permis de développer une solution fonctionnelle pour la gestion de scolarité, mais a également consolidé nos compétences en développement Java, en conception orientée objet, et en frameworks modernes. Cette expérience constitue une base solide pour aborder des projets encore plus complexes à l'avenir.