ING2 GSI3

Raphaël FROMENTIN

Lili-Rose GICQUEL

Alice POINSATTE

Audrey TRUONG

Maxime DUBIN–MASSE

Rapport projet JEE

Création d'un site de gestion de scolarité

30 novembre 2024

Matière : JEE

Professeur : Mohamed Haddache

Année scolaire : 2024-2025

Sommaire

Introduction

Dans le cadre du module de JEE, nous avons eu pour projet de réaliser une application web en utilisant deux méthodes, Spring Boot et Jakarta.

L'objectif de notre projet est la création d'une application web CRUD pour la gestion académique. Il permet de gérer des étudiants et professeurs et leurs cours et notes.

Les fonctionnalités principales que nous avons implémentées ont été l'authentification des utilisateurs, la gestion des utilisateurs (étudiants, professeurs et administrateurs), des cours (en permettant la création et affectation de cours ainsi que l'affichage de ces derniers dans les plannings), et enfin des notes (en permettant d'affecter des notes à des élèves et de générer des rapports de notes).

L'application web de gestion de scolarité vise à fournir une solution intuitive pour les tâches académiques donc pour les saisie des notes, inscriptions aux cours, consultations des résultats …

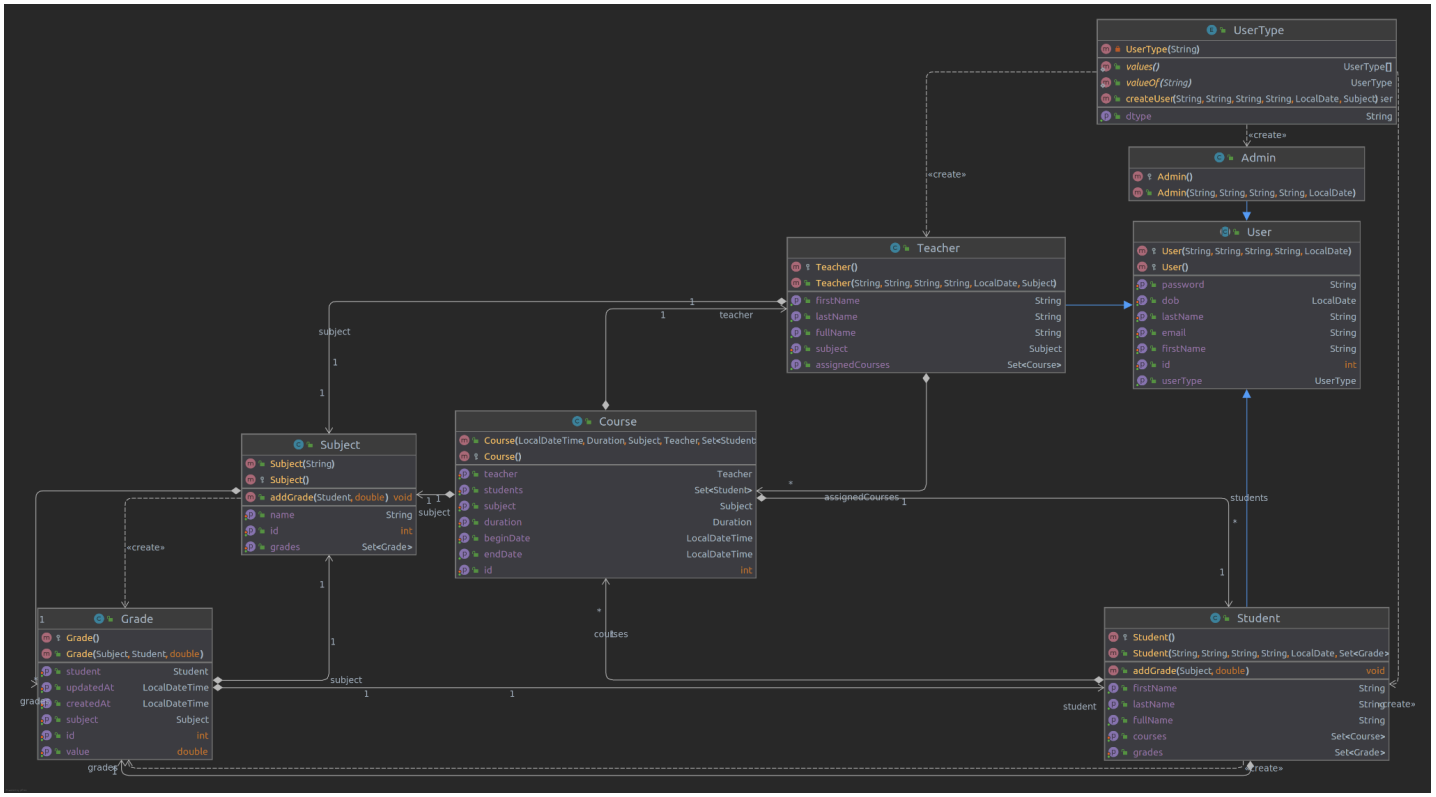
Cahier des charges

Voici alors ce qui nous été demandé de ce projet:

- **Description des fonctionnalités :**
    - Gestion des étudiants : Les étudiants ont un nom, prénom, une date de naissance, un mail et un mot de passe.
    - Gestion des enseignants : Ils ont un nom, prénom, une date de naissance, un mail et un mot de passe et une matière avec une affectation aux cours.
    - Gestion des cours : création, modification, suppression, et assignation des cours aux enseignants/étudiants.
    - Saisie des notes par les enseignants et les administrateurs si besoin.
    - Consultation des résultats par cours ou par étudiant.
    - Authentification avec gestion des rôles.

Chaque utilisateur a un rôle et des autorisations :

- Les professeurs et les administrateurs peuvent saisir, modifier et consulter les notes de tous les élèves.
- Les élèves peuvent quant à eux seulement consulter leurs notes et ne voient que les leurs. Ils ont plusieurs cours
- Un professeur est assigné à une seule matière.



Analyse du diagramme pour décrire la conception

- User :
  - Entité abstraite représentant un utilisateur générique.
  - Décrit les attributs communs (id, email, etc.).
  - Décrit ses sous-classes spécialisées (Étudiant, Enseignant, Administrateur).
- Student :
  - Stocke les notes des étudiants par matière.
- Teacher :
  - Associé à plusieurs cours qu'il enseigne.
- Admin :
  - Rôle spécifique pour la gestion globale.
- Course :
  - Définit les informations relatives à un cours (début, durée).
  - Relié à un enseignant et à plusieurs étudiants.
- Subject :
  - Représente une matière pouvant être associée à plusieurs cours.

Technologies utilisées

Ce module repose sur une architecture Model-View-Controller (MVC) intégrant les outils suivants :

- **Backend** : Spring Boot et Hibernate pour la logique métier et la persistance des données.
- **Frontend** : Pages JSP pour l'affichage et l'interaction utilisateur.
- **Base de données** : PostgreSQL pour stocker les données.

Pour la communication entre le modèle et sa vue, nous avons utilisé des DTO (data transfer object). Ces derniers permettent de lire les informations reçues d'un formulaire et de les encapsuler dans un objet qui sera ensuite utilisé dans le programme (registerDTO, permet d'enregistrer les informations lors de la création d'un utilisateur par exemple).

En utilisant ces DTOs, on évite que des données incorrectes ou incomplètes n'entrent dans le système, ce qui améliore la fiabilité de l'application.

Pour les bases de données, nous avons utilisé PostGreSQL. Ce dernier est plus simple à configurer, à moins de problèmes de persistances de données et permet également d'ajouter des contraintes à notre modèle (tel que la possibilité d'avoir des notes négatives).

Nous avons également utilisé des services pour encapsuler notre logique technique ce qui permet de la garder séparée de la logique métier et de rendre le code plus facilement testable.

Afin d'éviter de répéter la logique concernant la vérification de connexion et d'autorisations de l'utilisateur, nous avons mis en place un système de middlewares sur Jakarta (et utilisé celui intégré à Spring) pour intercepter les requêtes avant qu'elles n'atteignent les controllers pour éventuellement faire des actions comme rediriger vers la page de connexion si l'utilisateur n'est pas connecté.

Description des pages

Bannière:

Sur l'ensemble des pages il y a une bannière qui contient le titre de la page ainsi que des boutons de retour au menu principal et déconnexion lorsque l'utilisateur est connecté. Pour éviter la duplication, nous avons créé un fichier banner.jsp que nous appelons au début de chaque partie body du site. De cette manière, les changements à la bannière seraient appliqués à toutes les pages.

```
<div class="topdiv">
    <p><%= title %></p> <!-- Can add the user name here -->
    <div class="topMenuContainer">

        <%
            if(pageContext.getSession().getAttribute("user") != null){
        %>

            <img src="${pageContext.request.contextPath}/pictures/HomeDark.png"
alt="Home icon" class="icon" id="home"
onclick="window.location.href='${pageContext.request.contextPath}/'">
            <img
src="${pageContext.request.contextPath}/pictures/LogoutDark.png" alt="Logout
icon" class="icon" id="logout"
onclick="window.location.href='${pageContext.request.contextPath}/logout'">
        <%
            }
        %>
    </div>
</div>
```

On commence par récupérer l'attribut "user" de la session et s' il est trouvé on affiche les icônes du menu et de la déconnexion.

Au début de chaque partie <body>:

```
<body class="main_body">
<% String title = "Add a user"; %>
<%@ include file="banner.jsp" %>
<div class="centerdiv">
```

On peut choisir le titre qui serait affiché dans la bannière en utilisant du Java en modifiant l'attribut "title" et ensuite on inclut le fichier jsp de la bannière.

Admin:

Lorsqu'on se connecte au site depuis un compte administrateur, la plus grande différence visible sur le menu est la possibilité d'accéder à une page users. En cliquant sur cette page

il est possible de soit ajouter un utilisateur ou soit afficher la liste de l'ensemble des utilisateurs.

Dans la page d'ajout d'utilisateur, on peut y trouver un formulaire ou l'on peut y renseigner des informations telles que leur nom, prénom, email, le type d'utilisateur ainsi que leur sujet l'utilisateur créé est un professeur.

Dans la page d'affichage des utilisateurs, on y voit l'ensemble des utilisateurs et de leurs informations mais il est également possible de filtrer les utilisateurs selon leur email et type d'utilisateur. Pour le filtrage, nous avons créé un autre DTO qui possède une méthode check.

```
public boolean check(User user) {
        return getEmailSearch().map(email ->
user.getEmail().startsWith(email)).orElse(true)
                && getUserType().map(type -> user.getUserType() ==
type).orElse(true);
    }
```

Cette méthode reçoit un objet utilisateur et cherche à comparer l'email reçu par le DTO à celui de l'utilisateur et retourne true si cela est correct ou si le champ donné était vide. Cela est appelé par la méthode suivante dans notre controller.

```
@RequestMapping(value = "/search", method = RequestMethod.POST)
public ModelAndView getDisplayWithSearchPage(SearchDTO dto,
@RequestParam(required = false) String error, @RequestParam(required = false)
String message){
        return new ModelAndView("admin_display_users", Map.ofEntries(
                Map.entry("users", userRepository
                        .findAllByOrderByIdAsc()
                        .stream()
                        .filter(dto::check)
                        .toList()
                ),
                Map.entry("error", error == null ? "" : error),
                Map.entry("message", message == null ? "" : message)
        ));
    }
```

Lorsqu'on demande une recherche dans notre formulaire, l'action du formulaire exécute cette méthode. La liste des utilisateurs qui correspondent aux informations données dans le formulaire sont alors affichées dans la page d'affichage des utilisateurs.

Course

La gestion des cours est une fonctionnalité fondamentale de notre application de gestion de scolarité. Elle permet d'organiser et d'administrer les cours académiques en intégrant les rôles des enseignants, des matières, et des étudiants. Dans la partie gestion des cours, les admins peuvent créer, modifier, supprimer et consulter des cours. De plus, ils peuvent relier les enseignants, les étudiants et les matières aux cours, tout en respectant la cohérence des données. Les enseignants et les étudiants peuvent consulter leur emploi du temps de manière intuitive avec l'emploi du temps qui est affiché en fonction des horaires.

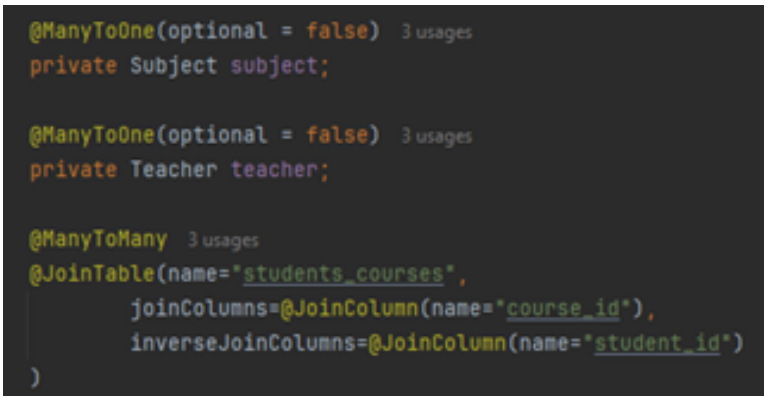
Partie conception du module :

L'entité Course est le cœur de la gestion des cours. Elle modélise les informations essentielles d'un cours (date, durée, enseignant, matière, étudiants inscrits) tout en établissant des relations avec d'autres entités comme Teacher, Student, et Subject.

Les attributs de Course sont :

- beginDate : Date et heure de début du cours.
- duration : Durée du cours.
- subject : Matière enseignée.
- teacher : Enseignant.
- students : Liste des étudiants inscrits au cours.

Pour la gestion des relations,



- Une relation @ManyToOne entre Course et Teacher pour représenter qu'un enseignant peut donner plusieurs cours.

- Une relation @ManyToOne entre Course et Subject pour indiquer qu'un cours concerne une matière spécifique.

- Une relation @ManyToMany entre Course et Student pour modéliser les inscriptions des étudiants à plusieurs cours.

Cette conception garantit une bonne organisation des données et les relations permettent de garantir la cohérence des données dans la base, tout en simplifiant leur manipulation via Hibernate.

Partie implémentation Backend :

Le contrôleur

Le contrôleur est l'élément central qui relie les utilisateurs à la logique de l'application. Son rôle est de recevoir les demandes (par exemple, afficher les cours d'un enseignant) et de renvoyer les bonnes données sous une forme compréhensible par le frontend.

Dans le contrôler CourseController, j'ai créé une méthode qui gère l'affichage des cours pour chaque type d'utilisateur :

- **Pour les administrateurs** : tous les cours sont affichés, avec la possibilité de les modifier ou de les supprimer.
- **Pour les enseignants** : seuls leurs cours de la semaine en cours sont visibles.
- **Pour les étudiants** : ils voient les cours auxquels ils sont inscrits.

Par exemple, pour afficher la page des cours, le contrôleur vérifie le rôle de l'utilisateur et récupère les données appropriées :

```
@RequestMapping(value = "", method = RequestMethod.GET)  Lili-Rose
public ModelAndView getCoursePage(HttpSession session, Optional<LocalDate> date) {
    // Récupère la date sélectionnée dans les paramètres de requête
    LocalDate selectedDate = date.orElse(LocalDate.now()); // Utilise la date du jour si aucun paramètre n'est fourni
    LocalDate selectedMonday = selectedDate.minusDays(selectedDate.getDayOfWeek().ordinal());
    LocalDate selectedSunday = selectedMonday.plusDays( daysToAdd: 7);

    return switch (session.getAttribute( s: "user")) {
        case Admin ignored -> {...}
        case Teacher teacher -> {...}
        case Student student -> {...}
        default -> throw new AssertionError( detailMessage: "Unexpected user");
    };
}
```

Cette approche permet de personnaliser l'expérience utilisateur et de garantir que chacun accède aux informations qui le concernent.

Optimisation des requêtes

D'autre part, pour optimiser les performances et éviter de surcharger la base de données avec des données inutiles, des requêtes personnalisées ont été mises en place dans le dépôt CourseRepository. Ces requêtes ciblent uniquement les informations nécessaires en fonction du contexte, ce qui permet de réduire la quantité de données récupérées et de fournir des résultats plus rapidement.

Voici deux exemples d'optimisation :

- **Cours d'un étudiant sur une période donnée**
  Lorsqu'il s'agit d'afficher les cours suivis par un étudiant (studentId) sur une plage de dates (from à to), une requête SQL native est utilisée :

```
SELECT *
FROM courses c
INNER JOIN students_courses sc ON c.id = sc.course_id
WHERE sc.student_id = :studentId
AND c.begin_date BETWEEN :from AND :to;
```

Ici, on filtre les cours uniquement pour l'étudiant concerné et la période indiquée.

Cette requête exploite la jointure directe entre les tables courses et students_courses (relation étudiant-cours), évitant de charger les cours qui ne sont pas liés à l'étudiant ou hors période. Cela améliore la performance lorsqu'il y a un grand volume de données.

- **Cours d'un enseignant sur une période donnée**
  Pour récupérer les cours dispensés par un enseignant (teacherId) sur une plage de dates (from à to), une requête similaire est utilisée :

```
SELECT *
FROM courses c
WHERE c.teacher_id = :teacherId
AND c.begin_date BETWEEN :from AND :to;
```

Ici, on filtre directement les cours affectés à l'enseignant concerné dans la période donnée.

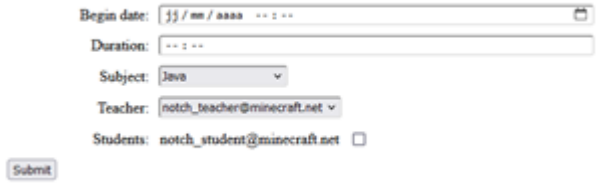
Cette approche évite de charger des informations superflues, comme les cours d'autres enseignants ou ceux en dehors de la période.

Partie Frontend (interface utilisateur)

La page d'administration

Pour les administrateurs, j'ai conçu une page qui permet de gérer tous les cours facilement. Elle combine deux parties :

- Un formulaire pour créer ou modifier des cours. Par exemple, l'administrateur peut choisir un enseignant ou inscrire des étudiants à un cours en cochant simplement des cases.



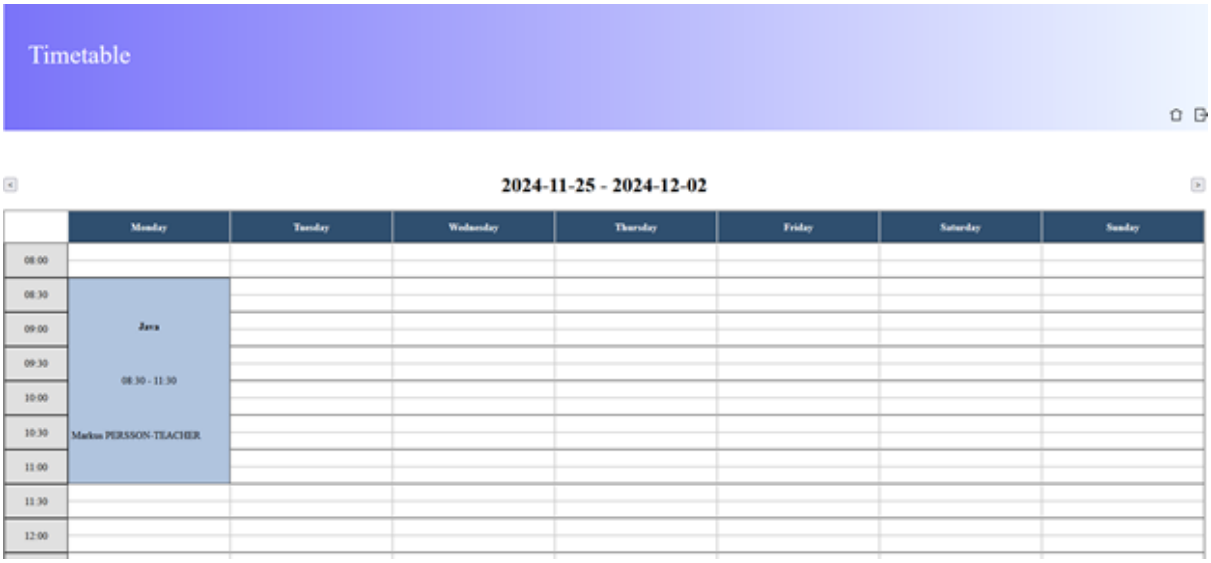
- Une table qui liste tous les cours existants avec des options pour les modifier ou les supprimer.

L'emploi du temps interactif

Les enseignants et les étudiants ont accès à une vue différente : un emploi du temps hebdomadaire. Chaque cours est placé sur une grille en fonction de son jour et de son heure. Par exemple :

- Les colonnes représentent les jours de la semaine.
- Les lignes correspondent à des créneaux horaires (par exemple, de 8h00 à 20h00).

Quand un cours est ajouté ou modifié, il apparaît automatiquement dans l'emploi du temps, avec son nom, son enseignant, et sa durée. C'est une façon claire et pratique pour les utilisateurs de voir leur planning.



Grades :

Objectif et fonctionnalité de la gestion des notes

La gestion des notes est un élément essentiel du système de gestion de scolarité. Elle permet aux administrateurs, enseignants, et étudiants de consulter, ajouter, modifier et supprimer des notes en fonction de leurs autorisations. Chaque note est associée à un étudiant, à une matière spécifique, et est enregistrée avec des informations telles que la date de création et de mise à jour. Ce module assure une gestion sécurisée et cohérente des notes, tout en offrant une interface web interactive pour les utilisateurs.

Conception du module :

L'entité Grade est au cœur de la gestion des notes. Elle représente une note attribuée à un étudiant pour une matière donnée. Voici les principaux attributs de cette entité :

- **subject** : Matière concernée.
- **student** : Étudiant concerné.
- **value** : Note obtenue.
- **createdAt** : Date de création.
- **updatedAt** : Date de mise à jour.

Il y a des relations entre les entités qui sont les suivantes :

- **@ManyToOne** entre **Grade** et **Subject** : une matière peut être associée à plusieurs notes.
- **@ManyToOne** entre **Grade** et **Student** : un étudiant peut avoir plusieurs notes.

Cette structure permet une gestion centralisée et cohérente des informations sur les notes.

Implémentation Backend :

Le contrôleur GradesController gère les demandes HTTP liées aux notes, en interagissant avec le GradeRepository, qui lui, permet d'interagir avec la base de données à l'aide de JPA et Hibernate, et inclut des méthodes personnalisées pour optimiser les requêtes. Le contrôleur récupère les notes d'un étudiant et permet de les classer par ordre alphabétique.

Voici un exemple de notre GradesController :

```
@RequestMapping(value = "/update", method = RequestMethod.POST)  gorosumo
public ModelAndView updateGrades(@Validated UpdateGradeDTO dto, HttpSession session) {
    return switch ((User) session.getAttribute( s: "user")) {
        case Admin ignored -> {...}

        case Teacher teacher -> {...}

        default -> throw new ResponseStatusException(HttpStatus.FORBIDDEN);
    };
}
```

Exemple de méthode pour récupérer les notes d'un étudiant :

```
public List<Grade> getAllByOptionalFilter(Optional<Integer> subjectId, Optional<Integer> studentId) {
    if (subjectId.isPresent() && studentId.isPresent()) {
        return getAllBySubjectAndStudentOrdered(subjectId.get(), studentId.get());
    } else if (subjectId.isPresent()) {
        return getAllBySubjectOrdered(subjectId.get());
    } else if (studentId.isPresent()) {
        return getAllByStudentOrdered(studentId.get());
    } else {
        return getAllOrdered();
    }
}
```

L'approche du GradeRepository permet de réduire la charge sur la base de données en récupérant uniquement les données nécessaires comme ceci :

```
public List<Grade> getAllBySubjectOrdered(int subjectId) {  2 usages
    String jpql = """
            SELECT g FROM Grade g
            JOIN g.student s
            JOIN g.subject sub
            WHERE sub.id = :subjectId
            ORDER BY s.lastName ASC, s.firstName ASC, sub.name ASC
            """;
```

Nous avons aussi la moyenne calculée dans le student_grades.jsp pour afficher la moyenne de l'étudiant comme ceci :

```
List<Grade> grades = (List<Grade>) pageContext.getRequest().getAttribute("grades");
double sum=0;
for(Grade grade : grades) {
  sum+=grade.getValue();
```

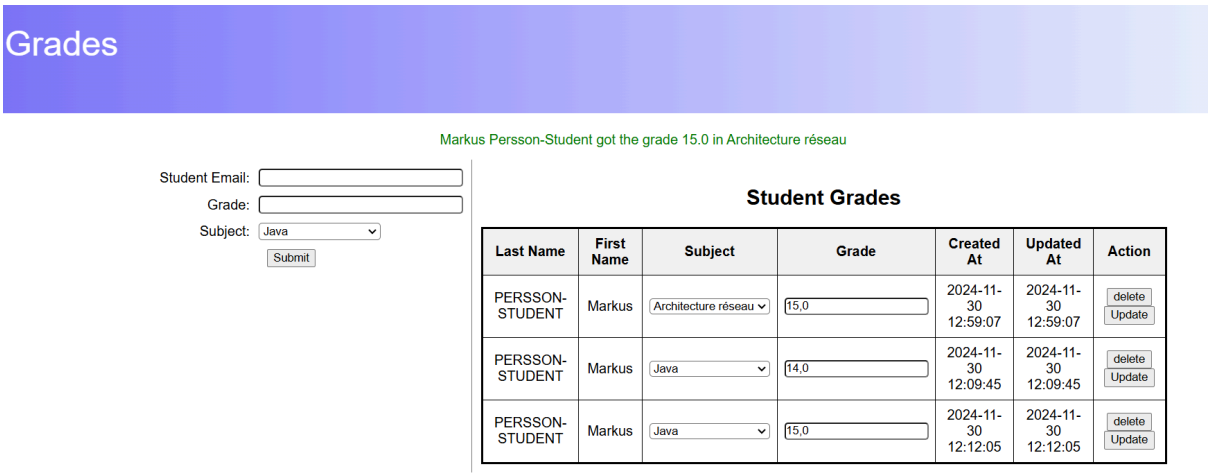
```
<label>Average: <%=sum/grades.size()%></label>
```

Partie Frontend (Interface Utilisateur) :

Le frontend permet aux utilisateurs de consulter et d'interagir avec les notes en fonction de leurs rôles.

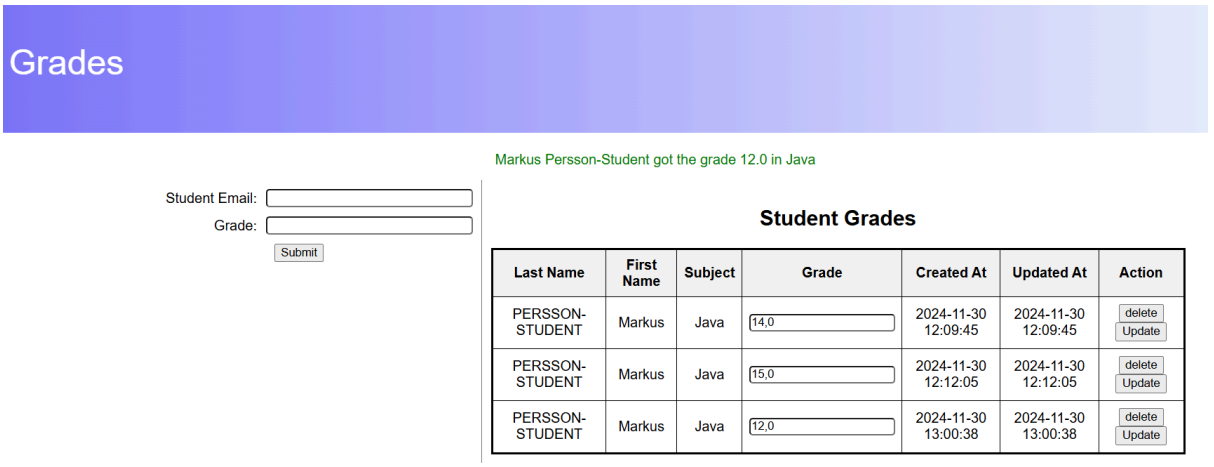
- **Page d'affichage des notes pour un étudiant (student_grades.jsp)** : L'étudiant peut consulter ses notes via cette page, qui présente une table dynamique avec les matières, les notes, et les dates de création et de mise à jour des notes.
- **Page d'administration des notes et pages des étudiants** : Les administrateurs et professeurs peuvent gérer toutes les notes à partir de cette page. Ils ont la possibilité de modifier ou supprimer des notes en accédant aux informations de tous les étudiants.

  Page admin (ajout de notes de matières différentes, suppression, modification) :



-

  Page teacher (ajout de notes d'une matière, ici java, modification et suppression) :



-

Page student (consultation de notes, moyenne générale et possibilité de voir le pdf)

Student Grades

Download PDF

Subject	Grade	Created At	Updated At
Architecture réseau	15.0	2024-11-30 12:59:07	2024-11-30 12:59:07
Java	12.0	2024-11-30 13:00:38	2024-11-30 13:00:38
Java	15.0	2024-11-30 12:12:05	2024-11-30 12:12:05
Java	14.0	2024-11-30 12:09:45	2024-11-30 12:09:45

Average: 14.0

PDF :

Grades: Markus PERSSON-STUDENT

Subject	Grade	Created At	Updated At
Architecture réseau	15.0	2024-11-30 12:59:07	2024-11-30 12:59:07
Java	12.0	2024-11-30 13:00:38	2024-11-30 13:00:38
Java	15.0	2024-11-30 12:12:05	2024-11-30 12:12:05
Java	14.0	2024-11-30 12:09:45	2024-11-30 12:09:45

Conclusion

Ce projet nous a offert l'opportunité de concevoir et de développer un site de gestion de scolarité, avec des fonctionnalités comme l'authentification, la gestion des données académiques et des emplois du temps, ainsi que l'attribution et la consultation des notes. Il nous a permis de mettre en œuvre des technologies comme Spring Boot, Hibernate, et PostgreSQL, tout en appliquant une architecture MVC qui garantit la clarté et la modularité de l'application.

Nous avons été confrontés à plusieurs défis techniques, tels que l'optimisation des requêtes pour éviter les surcharges de la base de données, notamment lors de la récupération des données liées aux utilisateurs et aux cours. La conception d'une interface utilisateur intuitive a également représenté un enjeu, en raison de la diversité des besoins des différents types d'utilisateurs. Nous avons notamment dû veiller à rendre l'interface accessible tout en respectant les contraintes liées aux rôles et aux autorisations.

Malgré ces défis, nous avons pu proposer une application fonctionnelle répondant au cahier des charges initial. Ce projet a été une expérience enrichissante qui nous a permis de développer nos compétences en développement web, en conception de bases de données et en gestion de projet.