



# Rapport

Conception et implémentation de l'application

---

Projet JEE ING2

*Application de gestion de scolarité*

26 novembre 2024

**EL AASSAL** Rim

**NEHILI** Nadir

**VEROVE** Augustin

**DE CADENET** Julien

**HUYNH-QUAN-BINH** Melvin

# Sommaire

<b>Introduction.....</b>	<b>3</b>
<b>I - Conception.....</b>	<b>4</b>
Analyse des besoins.....	4
Administrateur.....	4
Enseignant.....	4
Étudiant.....	5
Modèle Conceptuel de Données (MCD).....	6
Justification de l'utilisation des UUID comme identifiants uniques.....	8
<b>II - Implémentation.....</b>	<b>9</b>
Version classique (sans Spring Boot).....	9
Étape 1 : Création des entités.....	9
Étape 2 : Création des DAO.....	9
Étape 3 : Création des services.....	9
Étape 4 : Création des contrôleurs/servlet.....	10
Étape 5 : Création des JSP et du style.....	10
Étape 6 : Améliorations et Bonus.....	10
Version avec Spring Boot.....	11
Étape 1 : Création des entités avec JPA.....	11
Étape 2 : Migration des DAO vers des repositories.....	11
Étape 3 : Création des services.....	11
Étape 4 : Remplacement des servlets par des controllers.....	11
Étape 5 : Configuration des vues JSP.....	11
Étape 6 : Gestion des fonctionnalités avancées.....	12



## Introduction

Le projet présenté dans ce rapport consiste en la création d'une application de gestion de scolarité. L'objectif principal de l'application est de fournir une interface pour chaque utilisateur (étudiant, enseignant, administrateur) pour gérer les différentes fonctionnalités.

Dans une première partie nous présenterons la conception de l'application avec la création de notre MCD et l'analyse des besoins des utilisateurs. Dans un second temps nous aborderons les étapes d'implémentations et les éventuelles difficultés rencontrées.

Le projet est accessible via ces deux versions :

Version Classique : <https://github.com/Nadirici/SchoolManagement/tree/without-spring-version>

Version Spring Boot : <https://github.com/Nadirici/SchoolManagement>

# I - Conception

## Analyse des besoins

Nous avons identifié trois types d'utilisateurs : Administrateur, Enseignant, Étudiant.

Chaque utilisateur doit se connecter via le même formulaire et être redirigé vers son "tableau de bord" en fonction de son rôle. De plus, chaque utilisateur peut consulter ses informations.

### Administrateur

L'administrateur a un accès complet à toutes les fonctionnalités du système. Il peut gérer toutes les entités (étudiants, enseignants, cours, inscriptions, résultats), avec des droits complets (CRUD) sur toutes les données.

Fonctionnalités	Besoin
Gestion des étudiants	CRUD sur les étudiants
Gestion des enseignants	CRUD sur les enseignants
Gestion des cours	CRUD sur les cours, Envoie d'un mail à l'enseignant pour le prévenir qu'il a été assigné à un cours
Gestion des devoirs	Consultation des devoirs, des notes et des stats,
Gestion des inscriptions aux cours	CRUD sur les inscriptions des étudiants aux cours
Gestion des demandes d'inscriptions	Consulter les demandes. Accepter ou Refuser, et envoi du mail à l'admin pour le prévenir d'une nouvelle demande

### Enseignant

L'enseignant a un accès limité aux fonctionnalités liées à la gestion de ses propres cours et étudiants. Il peut saisir des notes, consulter les résultats de ses étudiants et gérer ses devoirs.

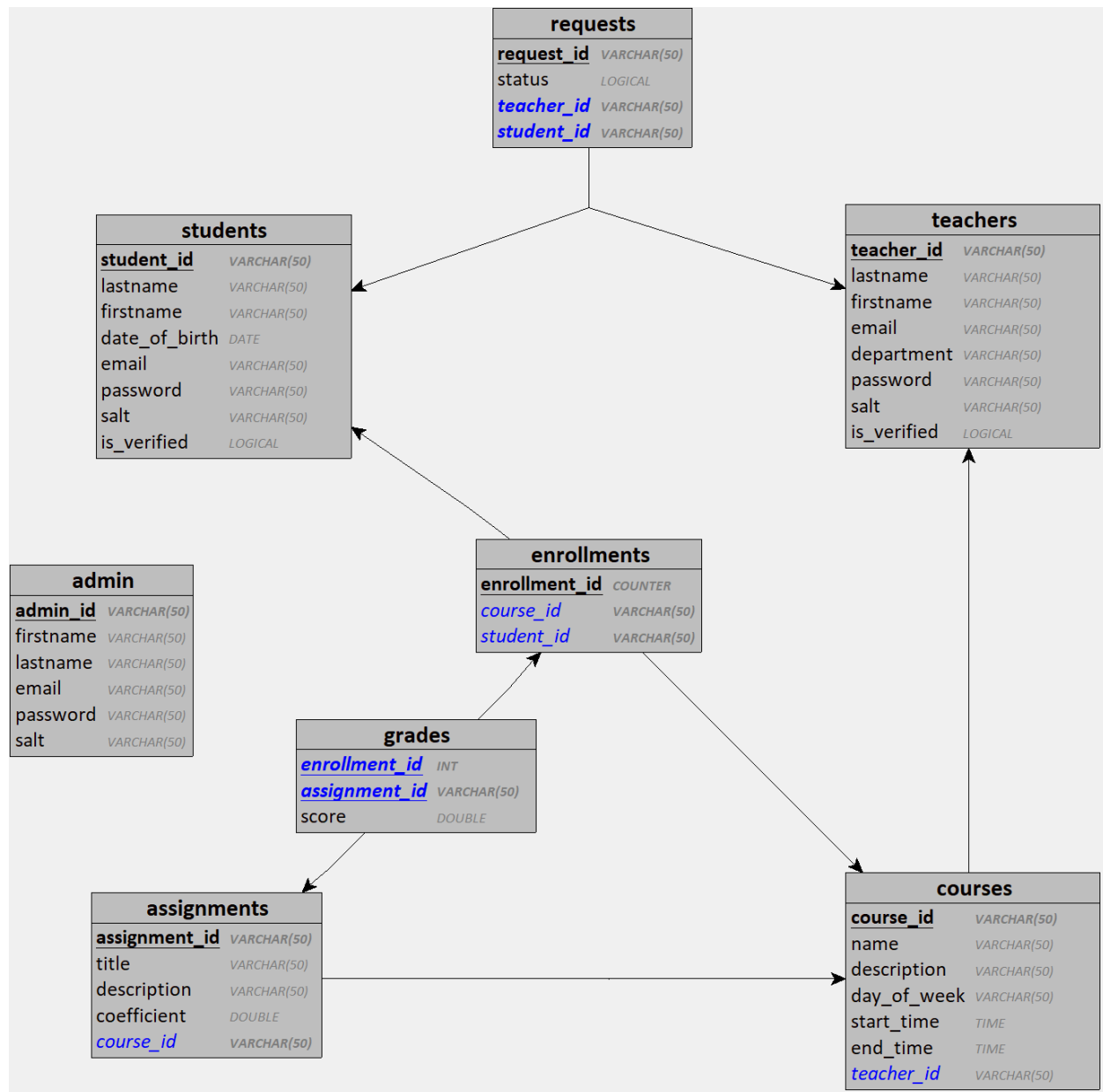
Fonctionnalités	Besoin
Gestion des cours	Consultation des détails de ses cours (Statistiques, étudiants, devoirs) Réception d'email quand il enseigne un nouveau cours Consultation de son emploi du temps
Gestion des devoirs	Création de devoirs Consultations des détails de ses devoirs (Statistiques, notes des étudiants) Saisie des notes


## Étudiant

L'étudiant a un accès limité à ses propres informations et résultats. Il peut s'inscrire aux cours, consulter ses résultats, et télécharger son bulletin de notes.

Fonctionnalités	Besoin
Gestion des cours	Consultation des cours disponibles Consultation des cours auxquels il est inscrit Consultation de son emploi du temps
Gestions des devoirs	Consultations des devoirs par cours Consultations de ses notes et des stats Téléchargement en PDF du bulletin Réception d'email quand il y a une nouvelle note

## Modèle Conceptuel de Données (MCD)





### 1. *students* (Étudiants)

- Cette entité permet de gérer les informations de base des étudiants, notamment l'enregistrement, l'authentification, et la vérification de leur statut dans l'application.
- Elle est liée aux autres entités comme les inscriptions et demande d'inscription (*enrollments*, *requests*).

### 2. *teachers* (Enseignants)

- Cette entité permet de gérer les informations de base des enseignants et de gérer leur connexion au système. Le champ *department* peut être utile pour filtrer ou organiser les cours.
- Elle est liée à la gestion des cours, avec une relation vers l'entité *courses*.

### 3. *courses* (Cours)

- Cette entité est au cœur de l'application, permettant de gérer tous les cours disponibles. La relation avec les enseignants (*teacher\_id*) permet de savoir quel enseignant enseigne chaque cours.
- Elle est liée à l'entité *assignments* pour gérer les devoirs associés à un cours et à l'entité *enrollments* pour suivre les inscriptions des étudiants.
- Les attributs *day\_of\_week*, *start\_time* et *end\_time* permettent l'affichage de l'emploi du temps.

### 4. *assignments* (Devoirs)

- Les devoirs sont associés à des cours spécifiques, et cette entité permet de gérer les devoirs de manière détaillée (titre, description, et coefficient).
- Elle est liée à l'entité *grades* pour permettre la saisie des notes par les enseignants pour chaque devoir.

### 5. *enrollments* (Inscriptions d'étudiants à des cours)

- Cette entité permet de lier les étudiants aux cours auxquels ils sont inscrits. Elle est essentielle pour le suivi des étudiants et de leurs cours.

- Elle est également liée à l'entité *grades* pour le suivi des notes des étudiants dans chaque cours.

#### 6. grades (Notes)

- Cette entité est utilisée pour enregistrer les notes des étudiants dans chaque devoir et cours. Elle est essentielle pour la gestion des évaluations.
- La relation avec *assignments* et *enrollments* permet de savoir à quel devoir et à quel étudiant chaque note correspond.

#### 7. requests (Demandes d'inscription à l'application)

- Cette entité gère les demandes des étudiants pour rejoindre l'application. Chaque demande est associée à un étudiant spécifique et à un enseignant qui pourrait valider ou non cette demande.
- Elle est liée à *students* et *teachers*, mais elle représente un état de demande et non d'inscription active dans un cours.

#### 8. admin (Administrateurs)

- Cette entité permet de gérer les administrateurs, qui sont responsables de l'ensemble du système (validation des inscriptions, gestion des cours, des étudiants, des enseignants, etc.).
- Les administrateurs ont un accès privilégié pour gérer les autres entités et superviser les processus de l'application.

### Justification de l'utilisation des UUID comme identifiants uniques

L'utilisation des UUID comme identifiants dans notre application garantit l'unicité des données, renforce la sécurité, et facilite la scalabilité de l'architecture. Ce choix permet de gérer efficacement des systèmes distribués tout en respectant les meilleures pratiques modernes de conception de bases de données.

Par exemple, si Melvin et Nadir ont chacun la même structure de base de données mais des données différentes, il est très facile de fusionner les deux bases de données car chaque UUID est unique.

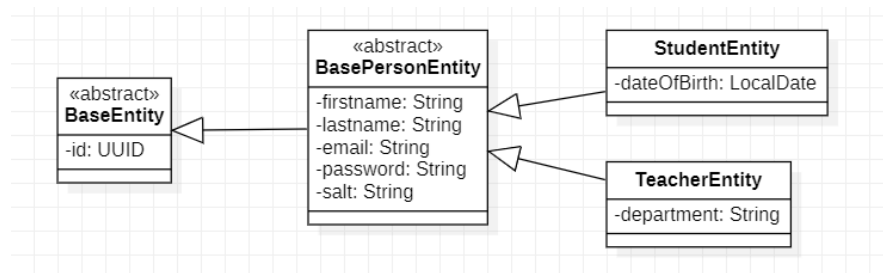


## II - Implémentation

### Version classique (sans Spring Boot)

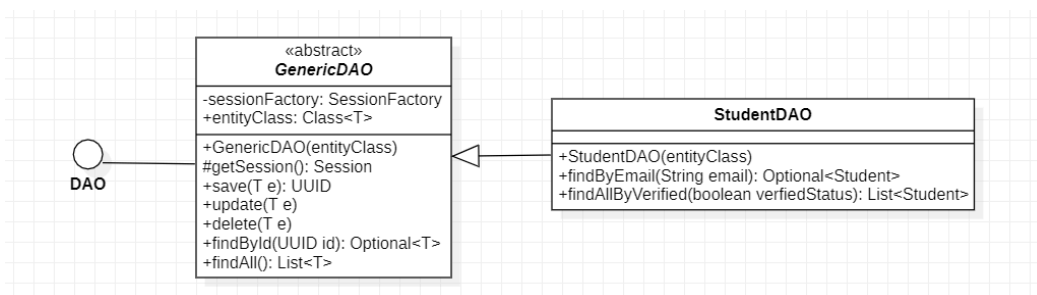
#### Étape 1 : Création des entités

Pour la réalisation des entités, nous avons créé une classe par entité dans notre base de données. Pour permettre une maintenance facile des entités, nous avons essayé de diviser le plus possible notre code. C'est à dire que pour les entités, nous avons une classe mère (BaseEntity.class) qui est commune à chaque entité avec le champ id qui se répète dans chaque entité.




#### Étape 2 : Création des DAO

Dans la continuité de l'implémentation, nous avons créé un DAO pour chaque entité. Ici aussi nous avons voulu limiter la redondance du code. Ainsi, nous avons une interface DAO.java qui définit les méthodes de base CRUD. De plus nous avons une classe GenericDAO.java qui implémente DAO.java.



#### Étape 3 : Création des services

Une fois les nos DAO créés, nous avons créé les différentes classes pour nos services. Les services doivent intégrer la logique qui permet de faire les différentes opérations. Ainsi nous avons des services qui permettent la création, la modification et la suppression d'entités, d'autres



qui permettent de récupérer des entités, des listes voir même des maps dans certains cas (exemple : `Map<Assignment;Grade>`).

#### **Étape 4 : Création des contrôleurs/servlet**

Quand les services sont finalisés (au moins les opérations CRUD) nous avons commencé à créer nos premier Servlet qui jouent le rôle de contrôleurs. Nous avons commencé par implémenter les services dans les servlets pour l'administration car cela nous a permis d'intégrer une grande partie des services. Dès que les servlet pour l'admin étaient opérationnels, nous avons pu faire les servlet pour les étudiants et les enseignants sans trop de difficultés car la plupart du temps la méthode existait dans un autre servlet.

Note : Nous avons utilisé le design pattern Composite pour calculer les statistiques (uniquement dans la version classique)

#### **Étape 5 : Création des JSP et du style**

Nous avons hiérarchisé les pages JSP de la même manière que nos contrôleurs, c'est-à-dire par type d'utilisateur (admin, student, teacher). Chaque page importe le même fichier CSS pour que nos pages aient toutes le même style

#### **Étape 6 : Améliorations et Bonus**

Quand nous avons considéré le projet à un stade avancé, nous avons amélioré l'application, notamment avec les fonctionnalités de mailing et de téléchargement PDF de bulletin qui été demandé mais que nous avons gardé pour la fin car ce ne sont pas des services nécessaires au bon fonctionnement de notre application.

## Version avec Spring Boot

Dans cette version, nous avons migré l'application classique vers une architecture basée sur Spring Boot, ce qui apporte une meilleure organisation du code, une maintenance simplifiée et une gestion plus fluide des dépendances grâce à Maven/Gradle.

### Étape 1 : Création des entités avec JPA

Les entités de la version classique ont été adaptées pour fonctionner avec JPA (Java Persistence API). Nous avons utilisé des annotations telles que `@Entity`, `@Table`, et `@Id` pour mapper les classes aux tables de la base de données.

Les relations (OneToMany, ManyToOne, etc.) entre entités ont été configurées pour refléter le MCD.

La classe mère `BaseEntity` ainsi que les entité de base ont été conservée pour centraliser les champs communs comme id, nom prénom etc

### Étape 2 : Migration des DAO vers des repositories

Les DAO ont été remplacés par des interfaces `JpaRepository` fournies par Spring Data JPA. Cette approche permet de réduire considérablement la quantité de code nécessaire pour les opérations CRUD.

### Étape 3 : Création des services

Les services contiennent la logique métier et font appel aux repositories pour interagir avec la base de données. Cette couche permet une séparation claire des responsabilités.

Nous avons annoté ces classes avec `@Service` pour les intégrer dans le cycle de vie de Spring.


### Étape 4 : Remplacement des servlets par des controllers

Les servlets ont été remplacés par des contrôleurs Spring MVC annotés avec `@Controller` ou `@RestController`. Chaque contrôleur gère les requêtes HTTP pour un rôle ou une fonctionnalité spécifique.

### Étape 5 : Configuration des vues JSP

Nous avons conservé l'utilisation des pages JSP, mais leur affichage est désormais géré via Spring MVC.

Les fichiers JSP sont placés dans le dossier `/WEB-INF/jsp/` pour des raisons de sécurité.



Le fichier application.properties configure le préfixe et le suffixe pour le rendu des vues JSP.

## **Étape 6 : Gestion des fonctionnalités avancées**

### **Envoi d'emails**

La fonctionnalité d'envoi d'emails a été intégrée à l'aide de Spring Mail. Cela simplifie l'envoi de notifications aux étudiants ou enseignants.

### **Téléchargement des bulletins en PDF**

Nous avons utilisé une bibliothèque comme iText ou Apache PDFBox pour générer les fichiers PDF dans les contrôleurs.

### **Cette migration vers Spring Boot a permis :**

Une meilleure organisation du code.

Une réduction de la redondance grâce aux annotations et mécanismes automatiques de Spring.

Une amélioration des performances et de la maintenabilité.

### **Focus sur la gestion des mots de passe dans la base de données.**

Dans l'application, la gestion des mots de passe repose sur des principes fondamentaux de sécurité pour protéger les données sensibles des utilisateurs, notamment les étudiants et les enseignants. Le processus utilise un algorithme de dérivation de clé sécurisé, PBKDF2WithHmacSHA1, pour générer un hachage robuste des mots de passe. Voici un résumé des étapes clés :

**Génération d'un sel unique :** Pour chaque utilisateur, un sel cryptographique aléatoire de 16 octets est généré à l'aide de la classe SecureRandom. Ce sel est essentiel pour protéger les mots de passe contre les attaques par table arc-en-ciel, en rendant chaque hachage unique, même si deux utilisateurs ont le même mot de passe.

**Hachage sécurisé :** Lorsqu'un mot de passe est fourni, il est combiné avec le sel correspondant, puis haché en utilisant l'algorithme PBKDF2 avec HMAC SHA-1. Ce processus itère **65 536 fois** pour ralentir les attaques par force brute.

**Stockage des informations :** Le hachage du mot de passe et le sel associé sont stockés dans la base de données, dans les entités Student et Teacher. Le sel est encodé en Base64 pour faciliter son stockage et sa récupération.



id	email	firstname	is_verified	lastname	password	salt	date_of_birth
----	-------	-----------	-------------	----------	----------	------	---------------

**Vérification des mots de passe :** Lorsqu'un utilisateur se connecte, son mot de passe fourni est haché à nouveau en utilisant le sel stocké. Le résultat est comparé avec le hachage enregistré. Si les deux correspondent, l'authentification est validée.

Ce mécanisme garantit que même en cas de fuite de la base de données, les mots de passe restent protégés, car les attaquants doivent déchiffrer chaque mot de passe individuellement, ce qui est coûteux en termes de temps et de ressources. Cette approche illustre une gestion sécurisée et conforme aux bonnes pratiques actuelles en matière de protection des mots de passe.

## Focus sur la gestion des statistiques

Dans le cadre de ce projet, les statistiques sont calculées à plusieurs niveaux (par étudiant, par cours, par devoir, globalement...), nécessitant une structure flexible et extensible pour agréger les données.

Pour répondre à ce besoin, nous avons utilisé le patron de conception Composite. Ce patron nous permet de représenter une hiérarchie d'objets de manière uniforme, qu'il s'agisse de composants simples (*Leaf*) ou de compositions (*CompositeStat*), tout en rendant le calcul des statistiques transparent pour les utilisateurs du système.