

JEE

Rapport Projet

Pauline Trepos
Marius Brosset
Alban Rouchonnat
Tatar Efe
Mathis Contassot

1. Introduction.....	3
2. Partie Conception.....	3
2.1 Conception base de données.....	3
2.1.1 Modèle Conceptuel des Données (MCD).....	3
2.1.2 Modèle Logique des Données (MLD).....	4
2.1.1 Flowchart.....	5
2.1.2 Affichage JSP avec Servlets.....	5
3. Présentation application.....	6
3.1 Interface générale.....	6
3.2 Interface étudiants.....	7
3.3 Interface professeurs.....	8
3.4 Interface Administrateurs.....	9
4. Backend Java.....	10
4.1 Dépendances.....	10
4.2 Gestion de la persistance - JPA.....	10
4.3 Servlets.....	10
4.4 Contrôleurs.....	10
4.5 Validateurs.....	11
4.6 Ssr.....	12
4.7 Connexion base de données.....	12
5. Conclusion.....	13

1. Introduction

Dans le cadre du cursus JEE (ING2, GSI) pour l'année académique 2024-2025, le projet consiste à développer une **application web** de gestion de scolarité. Cette application vise à centraliser et simplifier la gestion des informations, tout en offrant des fonctionnalités adaptées aux besoins des étudiants, enseignants et administrateurs.

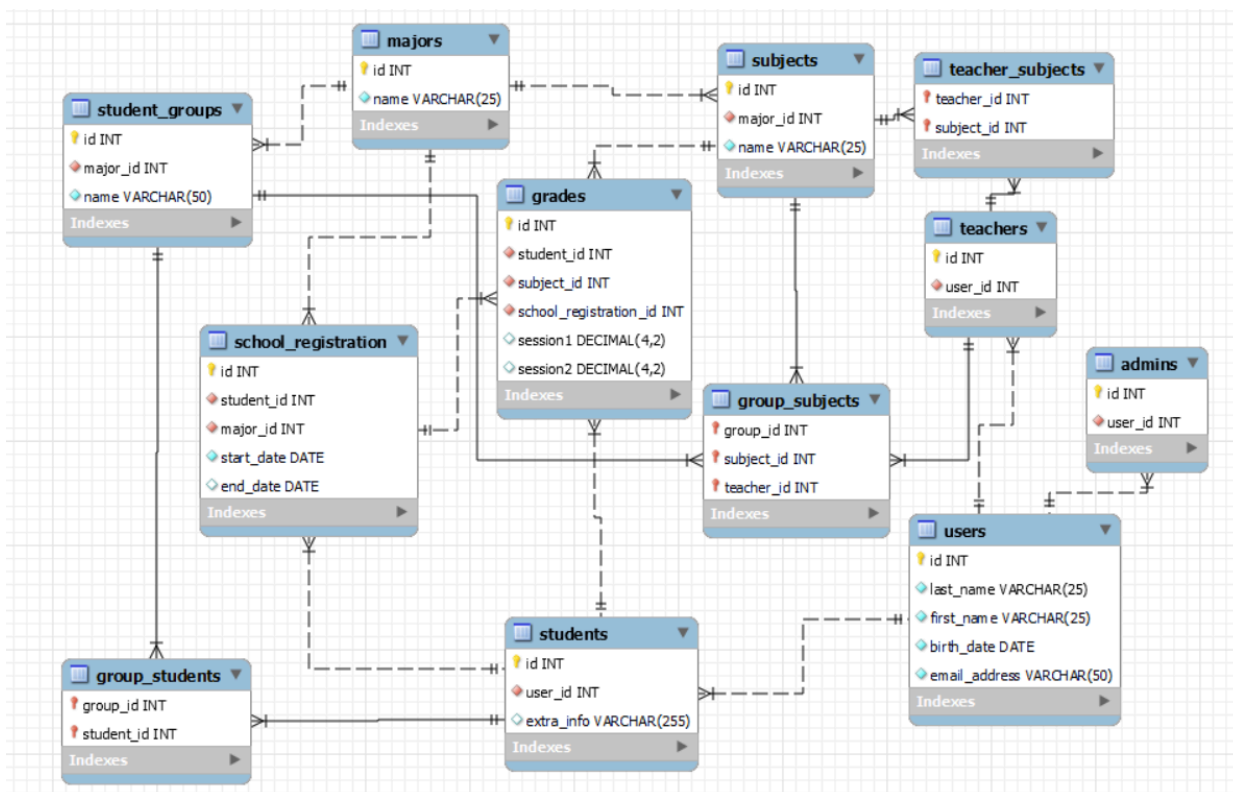
Pour ce projet, nous avons commencé par la conception de l'application, avec des outils de collaboration comme Canva pour améliorer l'efficacité du travail en groupe.

2. Partie Conception

2.1 Conception base de données

2.1.1 Modèle Conceptuel des Données (MCD)

La modélisation de la base de données est un point crucial du projet. Elle pourrait complètement nous bloquer et nous forcer à recommencer en cas d'oubli de données clés.



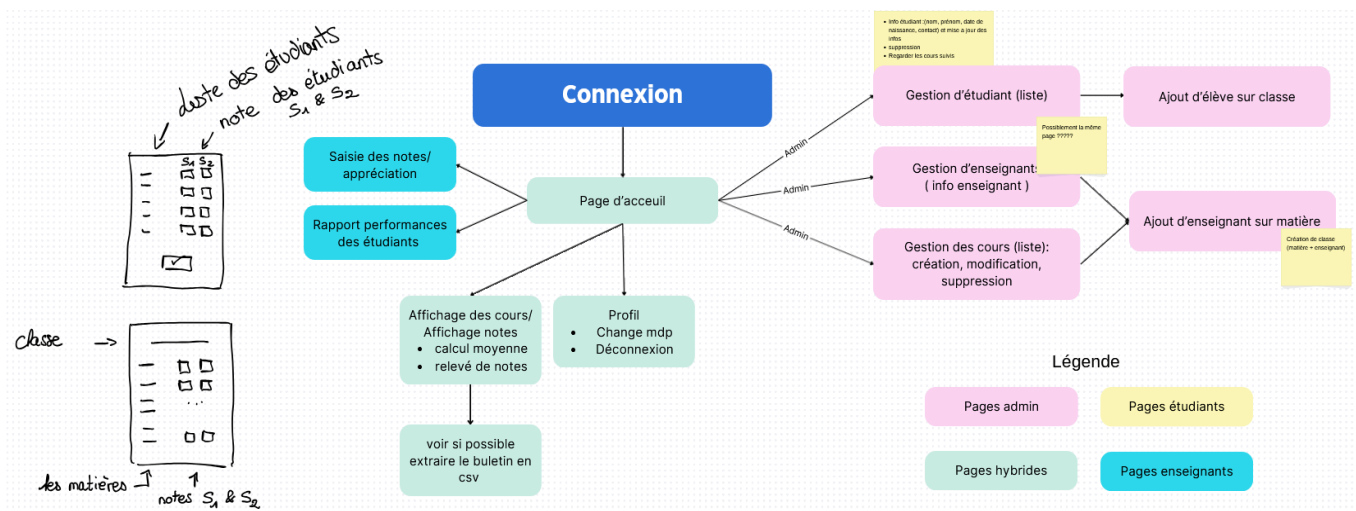
2.1.2 Modèle Logique des Données (MLD)

Ce diagramme représente un modèle conceptuel des données pour notre application de gestion de scolarité.

```
users(id, uuid, surname, first_name, date_of_birth, email, password, gender);
admins(id, uuid, #id_user);
students(id, uuid, #id_user);
courses(id, uuid, name);
enrollments(id, uuid, #id_student, #id_course, start_date, end_date);
subjects(id, uuid, name, #id_course);
teachers(id, uuid, #id_user);
groups(id, uuid, #id_course, name);
group_students(id, uuid, #id_group, #id_student);
group_subjects(id, uuid, #id_group, #id_subject, #id_teacher);
grade(id, uuid, #id_student, #id_subject, #id_enrollment, session1, session2);
```

On se basera donc sur cette base de données pour tout notre projet.

2.1 Conception frontend



2.1.1 Flowchart

Ce graphique illustre le fonctionnement de l'application. Le point de départ est une page de **connexion**, qui redirige les utilisateurs vers une page d'accueil centrale en fonction de leur rôle (administrateur, enseignant ou étudiant).

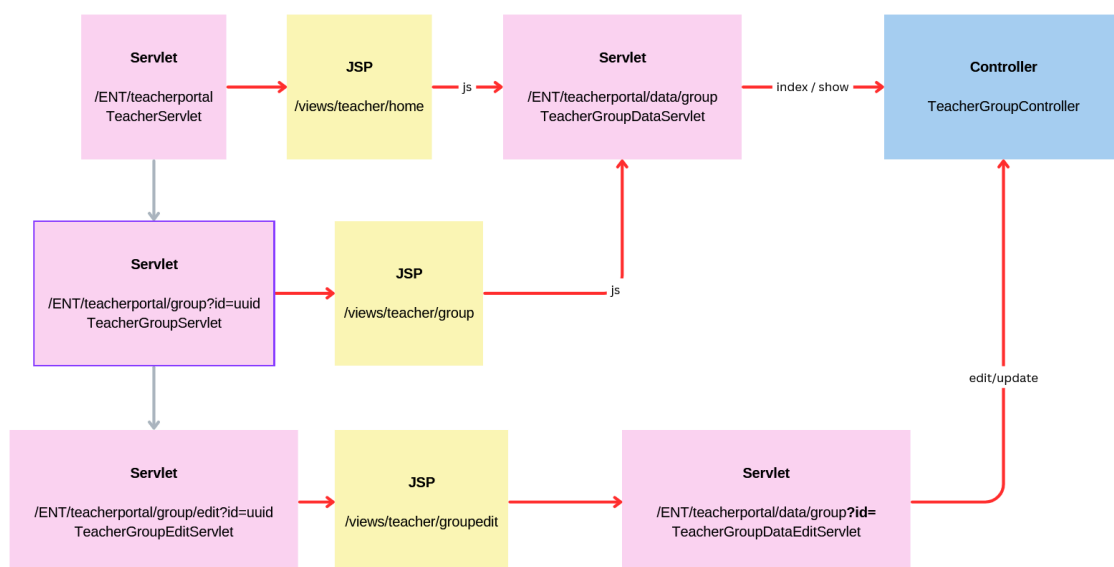
Depuis cette page d'accueil, plusieurs chemins sont possibles :

Un **administrateur** peut accéder aux outils de gestion, notamment la gestion des étudiants (affichage d'une liste d'étudiants), avec la possibilité d'ajouter un élève à une classe. De même, la gestion des enseignants est accessible pour consulter leurs informations et les associer à des matières spécifiques. L'administrateur peut également gérer les cours (création, modification, suppression).

Les **enseignants** ont accès aux fonctionnalités liées à l'évaluation, comme la saisie des notes et des appréciations pour les étudiants. Ils peuvent également consulter ou générer des rapports sur les performances des élèves. Ces fonctionnalités incluent l'affichage des cours et des notes, avec des calculs automatiques de moyennes et la possibilité d'extraire un bulletin en format CSV.

Les **étudiants** disposent d'un espace pour consulter leurs cours et leurs notes. Ils ont également accès à leur profil, où ils peuvent modifier leur mot de passe et se déconnecter.

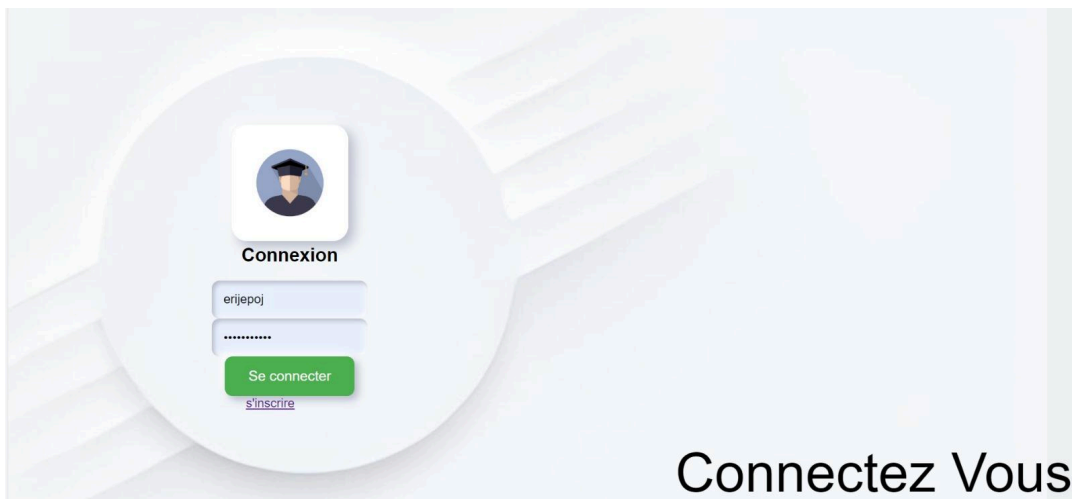
2.1.2 Affichage JSP avec Servlets



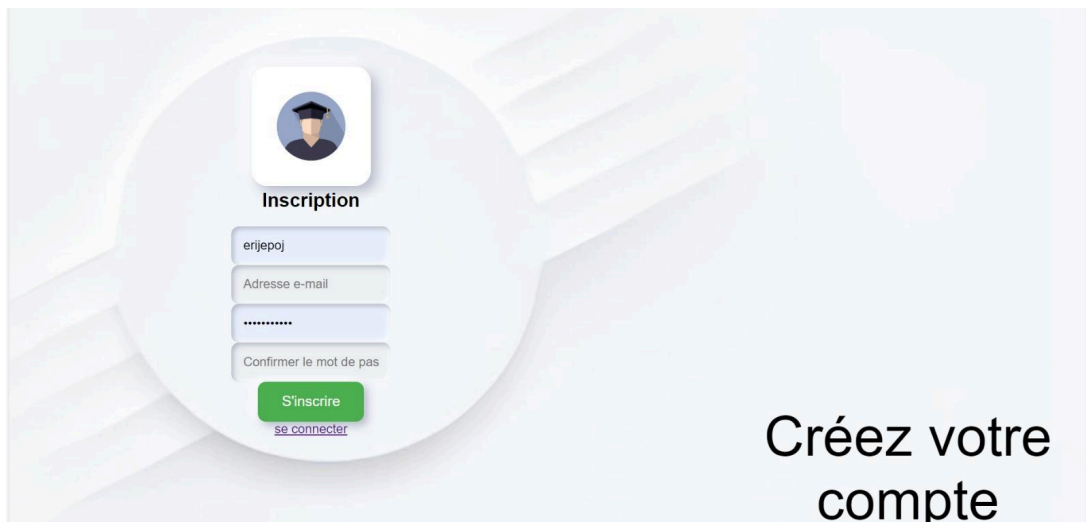
3. Présentation application

3.1 Interface générale

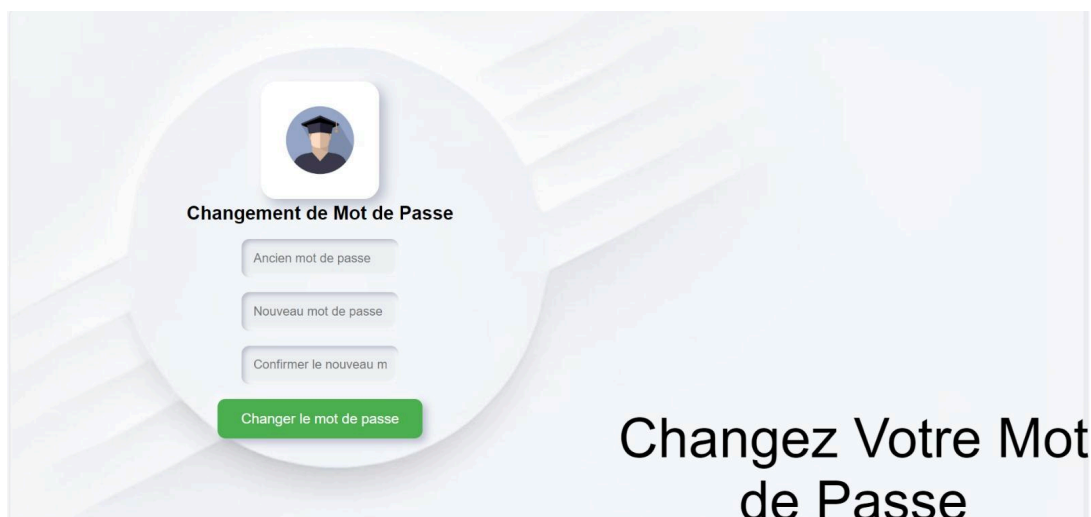
Page de Connexion : Permet aux utilisateurs de se connecter à l'application en entrant leur adresse email et leur mot de passe.



Page d'inscription : Permet à l'utilisateur de créer un compte en fournissant une adresse e-mail et un mot de passe et aussi une vérification via une confirmation de mot de passe.



Changement de mot de passe : Permet de changer son mot de passe.



3.2 Interface étudiants

Page de consultation des notes : Liste les matières avec les sessions principales et de rattrapage, proposant de générer et télécharger des rapports.

The screenshot shows a student dashboard with a dark header bar containing 'Dashboard' and 'Home' links, and a user profile for 'Tatar Efe'. Below the header, there is a link 'return to classes' and a section for 'group MI groupe 123' and 'subject Traitement de Signal'. A search bar 'filter by name or surname' is present. A table lists students with their names, surnames, and grades for Session 1 and Session 2.

Name	Surname	Session 1	Session 2
Charlie	Brown	1.0	
Avery	Cook		2.0
Riley	Bell	3.0	

total of 3 elements page number: 1 page size: 10

modify

Rapport de performance étudiant : Affiche les détails de performance d'un étudiant, incluant les résultats par session et son rang dans la classe. Les étudiants ont la possibilité d'exporter leurs notes au format .txt.

The screenshot shows a student dashboard with a dark header bar containing 'Dashboard' and 'Home' links, and a user profile for 'Smith John'. Below the header, there is a table listing courses with their start and end dates. A 'grades' dropdown menu is visible. Below the dropdown, a table shows session results for a student, including session 1, session 2, subject, and course. A 'total of 2 elements' message is displayed. A Notepad window titled 'rapport_etudiant(2).txt' is open, showing the student's name, course, and average grade.

Course	start	end
ING2 GSI 2024-2025	2024-01-01	2025-01-01
bouniboune	2024-11-06	2024-11-20
ING1 MIM 2023-2024	2024-11-11	2024-11-13

grades -

session 1	session 2	subject	course
20.0	1.0	Intelligence Artificielle	ING2 GSI 2024-2025
20.0	0.0	matiere de boune 2	bouniboune

total of 2 elements page number: 1 page size: 10


localhost:8082/EN1/data/student/review?course=5abf82a0-4a3e-4476-a450-57117451aff

rapport_etudiant(2).txt - Notepad

```
File Edit Format View Help
John Smith
ING2 GSI 2024-2025
Intelligence Artificielle: 20.0
average: 20.0
```

3.3 Interface professeurs

Saisie des notes : Permet aux enseignants d'entrer les notes des étudiants.

[Dashboard](#) [Home](#) Tatar Efe 

[return to class](#)

Surname	Name	Session 1	Session 2
Charlie	Brown	<input type="text" value="1.0"/>	<input type="text"/>
Avery	Cook	<input type="text"/>	<input type="text" value="2.0"/>
Riley	Bell	<input type="text" value="3.0"/>	<input type="text"/>

save

3.4 Interface Administrateurs

Page d'accueil : Cette page est l'accueil de l'administrateur, elle regroupe toutes les fonctionnalités auxquels il a accès

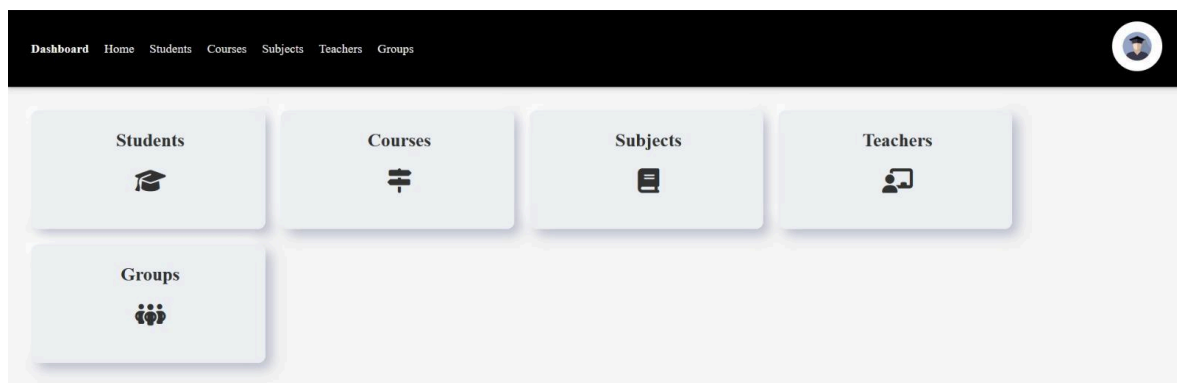


Tableau de bord - Liste des enseignants : Affiche la liste des enseignants, permet de modifier ou supprimer un enseignant, et offre la possibilité d'en ajouter un nouveau. Existe aussi pour les cours et les matières. L'administrateur a aussi une page similaire pour les étudiants.

Dashboard
Home
Students
Courses
Subjects
Teachers
Groups

home page

Filter Teachers
sort by
filter by gender

Name	Surname	Gender	Date of Birth	Email	
Charlie	Brown	Other	2000-03-03	charlie.brown3@example.com	
Efe	Tatar	Male	2024-11-28	test@test.com	
Hassan	Matouk	Other	1932-11-22	hmatouk@cy-tyech.fr	
Didier	Cransac	Male	1856-12-17	dcransac@cy-tech.fr	
Nathan	Signoud	Male	2003-10-05	signoudnat@cy-tech.fr	

total of 5 elements
page number:
page size:

Register New Teacher

4.Backend Java

4.1 Dépendances

Notre projet utilise Maven pour gérer les dépendances et assurer la cohérence du développement. Le fichier pom.xml centralise toutes les bibliothèques essentielles à notre application, notamment la Jakarta Servlet API pour la gestion des servlets et des requêtes HTTP, ainsi que Hibernate et JPA pour la gestion de la persistance des données. La connexion à la base de données MySQL est rendue possible grâce au connecteur MySQL, tandis que la base de données H2 a été utilisée pour faciliter les tests et le développement.

Les tests unitaires sont pris en charge par JUnit 5, garantissant une couverture adéquate du code. Le fichier pom.xml configure également l'encodage en UTF-8 et la compatibilité avec Java 11. Le packaging au format WAR permet un déploiement simple sur un serveur comme Tomcat. Enfin, grâce au plugin maven-war-plugin, la génération des fichiers déployables est automatisée, rendant le processus plus efficace.

Cette configuration Maven garantit une gestion centralisée et fiable des dépendances, facilitant l'évolution du projet tout en respectant les standards industriels.

4.2 Gestion de la persistance - JPA

Dans notre projet, nous avons utilisé JPA avec Hibernate pour gérer la persistance des données. La configuration principale se trouve dans le fichier persistence.xml, situé dans le dossier META-INF permet de connecter l'application à une base de données MySQL et de gérer les entités représentant les étudiants, enseignants, cours, inscriptions, et résultats.

Hibernate a été configuré pour garantir la compatibilité avec MySQL, assurer un contrôle manuel du schéma et afficher les requêtes SQL pour faciliter le débogage. Cette approche centralise la gestion des données, offre une flexibilité pour l'ajout de nouvelles entités, et maintient une cohérence dans les interactions entre le backend et la base de données.

Cette intégration a joué un rôle clé dans la fiabilité et l'évolutivité de notre application

4.3 Servlets

```
/**
 * Servlet implementation class AdminStudentsServlet
 */
@WebServlet("/adminportal/students")
```

Pour contrôler nos fichiers jsp, nous utilisons des **servlets**, avec les commandes `.setAttributes` et `.getRequestDispatcher` pour la redirection de pages tout en gardant des informations.

```
if( request.getParameter("id") != null ) {
    Student student = new StudentService(HibernateUtil.getEntityManager()).getStudentByUuid(request.getParameter("id"));
    request.setAttribute("student", student);
    request.getRequestDispatcher("/views/admin/student/student.jsp").forward(request, response);
    return;
}

request.getRequestDispatcher("/views/admin/student/students.jsp").forward(request, response);
```

4.4 Contrôleurs

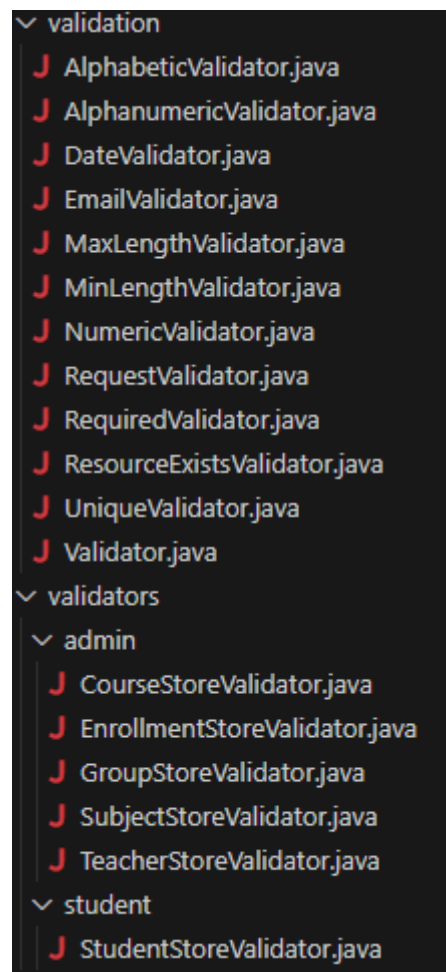
Les contrôleurs sont responsables de chaque filtres, par exemple ici dans **TeacherControllerAdmin** où on a l'initialisation de 3 filtres pour la page **teachers.jsp**.

```
String nameFilter = request.getParameter("name_filter");
String sortBy = request.getParameter("sort_order");
String genderFilter = request.getParameter("gender_filter");

String queryString = "SELECT t FROM Teacher t WHERE 1=1";

if( nameFilter != null && !nameFilter.isBlank() ) {
    queryString += " AND (t.user.firstName like :nameFilter OR t.user.surname like :nameFilter)";
}
if( genderFilter != null && !genderFilter.isBlank() ) {
    queryString += " AND t.user.gender = :genderFilter";
}
if( sortBy != null && !sortBy.isBlank() ) {
    if(sortBy.equals("sur_desc")) {
        queryString += " ORDER BY t.user.surname DESC";
    } else {
        queryString += " ORDER BY t.user.surname ASC";
    }
}
}
```

Ceci permet de gérer efficacement les interactions utilisateur sur la page tout en simplifiant le traitement des données.



4.5 Valideurs

Dans notre projet, nous avons mis en place un **système robuste** de validation des données pour garantir l'intégrité et la cohérence des informations saisies ou transmises.

Ce système repose sur les deux principaux répertoires validation et validators :

- Validators contient des validateurs spécifiques à des entités ou **cas d'utilisation précis**. Dans notre cas pour des pages admin et student.
- Validation contient des classes génériques permettant de valider des données selon des **critères spécifiques**. Ces classes sont **réutilisables** et servent de base pour vérifier des attributs communs.

Toutes ces classes vont utiliser la classe **validator**, une interface simple qui va donner la structure pour tous les autres validateurs :

```
public interface Validator {  
  
    public boolean validate(Object attribute, String rule);  
  
}
```

Prenons pour exemple **AlphabeticValidator** et **StudentStoreValidator**, ils sont indispensables pour assurer le bon fonctionnement de l'application, sans bugs, ni arrêts inattendus.

```

@Override
public boolean validate(Object attribute, String rule) {

    if( !(attribute instanceof String) )
    {
        return false;
    }

    String att = (String) attribute;

    for(int i = 0 ; i < att.length() ; i++)
        if( ! Character.isAlphabetic(att.charAt(i)) )
            return false;

    return true;
}

```

```

public StudentStoreValidator(HttpServletRequest request) {
    super(request);
}

@Override
public Map<String, String> rules() {
    Map<String, String> rules = new HashMap<>();
    rules.put("surname", "required,min:2,max:25");

    return rules;
}

```

4.6 Server side rendering (Ssr)

Le **Server-Side Rendering** est une technique où le rendu des pages web est effectué sur le serveur avant d'être envoyé au navigateur. Cela signifie que le serveur génère une page HTML complète avec les données intégrées, que le navigateur peut afficher immédiatement.

On va le décomposer en 3 étapes. Étape 1, **tDataTable.js** va faire les requêtes, ensuite, c'est le contrôleur qui va s'occuper de préparer les données au format **json** comme on peut le voir ici:

```

String jsonFormattedResponse = ServerSideRenderingHelper.getJsonFormattedTable(true, "");
response.getWriter().write(jsonFormattedResponse);

```

Et pour finir, **tDataTable.js** va récupérer ces données sous forme de tableau, et les insérer directement sur la page.

Pour aider avec le formatage des json, nous avons la classe **SideServerRenderingHelper** qui va gérer la vérification du json, et gérer les messages de validations ou d'erreurs.

4.7 Connexion base de données

Nos requêtes sont écrites et envoyées en java à l'aide de fonctions comme **.getResultList()** et **.getSingleResult()** qui vont envoyer la requête, et nous donner le résultat sous un certain format attendu.

```
// Retrieve all Subjects
public List<Subject> getAllSubjects() {
    TypedQuery<Subject> query = entityManager.createQuery("SELECT c FROM Subject c", Subject.class);
    return query.getResultList();
}
```

Dans cet exemple, on envoie une requête qui va chercher toutes les matières à l'aide de **.getResultList()**, cette fonction est utilisée quand on attend que la requête nous renvoie un ensemble de données sous forme de liste.

Ces données récupérées sont, pour certaines, injectées sur nos pages dans des tableaux d'autres serviront de différentes manières.

```
<%
    List<Subject> subjects = (List<Subject>)request.getAttribute("subjects");

    for(Subject subject : subjects) {
        %>
        <tr>
            <td><%= subject.getName() %></td>
            <td><%= subject.getCourse().getName() %></td>
        </tr>
    }
%>
```

Les tableaux qui reçoivent ces données sont gérés par des scripts javascript qui permettent de trier ces données ou de simplement recharger la table comme ici

```
document.getElementById('filter-form').addEventListener('input', function() {
    datatable.process();
});

export function processSubjectTable() {
    datatable.process();
}
window.processSubjectTable = processSubjectTable;
```

5. Conclusion

En réalisant ce projet, nous avons découvert les tâches à réaliser pour concevoir une base de données robuste, développer un frontend fonctionnel, et implémenter un backend fiable. Toutes ces étapes sont indispensables pour développer une application fonctionnelle.

Ce travail nous a donné une compréhension claire et globale du cycle de développement d'un projet, allant des premières idées jusqu'aux tests finaux. Nous avons appris à travailler en groupe, que ce soit pour la conception, l'implémentation, ou l'intégration des différentes fonctionnalités, il était très important d'être coordonnés. De plus, nous avons pris conscience de l'importance de la communication et des ajustements nécessaires au fur et à mesure de l'avancement pour répondre aux exigences.

Enfin, les délais imposés nous ont appris à travailler sous pression, tout en respectant les exigences de qualité et les contraintes technologiques.