MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR
DE LA RECHERCHE SCIENTIFIQUE ET DE L'INNOVATION

UNIVERSITÉ SULTAN MOULAY SLIMANE

ECOLE NATIONALE DES SCIENCES APPLIQUÉS DE KHOURIBGA

Rapport du Projet

Filière : Informatique et Ingénierie des Données

Réalisé par :
Oumaima **El Alami**
Salma **Adsaoui**

Développement d'une Application de Gestion des Notes pour un Établissement Scolaire :

Une Solution Intégrée pour Administrateurs et Enseignants

Encadré par :

M. Gharabi Noreddine ENSA Khouribga

Année Académique : **2025**

Sommaire

Remerciement

Nous souhaitons exprimer nos sincères remerciements à Monsieur Gharabi Noreddine pour ses précieux conseils et son accompagnement tout au long de ces deux années. En première année, grâce à ses explications claires et sa pédagogie, nous avons pu maîtriser les bases du langage Java, une compétence essentielle pour notre parcours. En deuxième année, son enseignement en Java EE (JEE) nous a permis de développer des compétences solides en développement d'applications web, qui ont été cruciales dans la réalisation de ce projet.

Nous remercions particulièrement Monsieur Gharabi Noreddine pour sa disponibilité, son soutien constant et son écoute attentive. Il a toujours pris le temps de répondre à nos questions, d'expliquer des concepts parfois complexes de manière simple et accessible. Ses conseils avisés ont grandement facilité notre progression et ont joué un rôle majeur dans la réussite de ce projet.

De plus, ses encouragements et son approche méthodique ont renforcé notre confiance en nos capacités et nous ont permis d'aborder ce projet avec sérénité et détermination. Sa passion pour l'enseignement et son engagement envers ses étudiants ont été une source constante d'inspiration.

Nous lui sommes profondément reconnaissants pour le temps et l'énergie qu'il a investis dans notre formation, ainsi que pour son rôle essentiel dans notre parcours académique.

Encore une fois, merci Monsieur Gharabi Noreddine pour votre précieuse aide, vos conseils éclairés et votre soutien tout au long de notre projet et de nos études.

Résumé

Ce projet vise à développer une application web permettant de gérer les notes des étudiants d'un établissement scolaire. L'application propose des fonctionnalités pour administrer les professeurs, les filières, les modules, les éléments de modules, les modalités d'évaluation et les notes. L'application repose sur deux profils principaux. Le premier profil, celui de l'Administrateur, permet de gérer les enseignants, les filières, les modules et leurs éléments, les modalités d'évaluation, ainsi que les comptes utilisateurs. L'administrateur a également la possibilité d'affecter des éléments de modules à des professeurs. Le second profil, celui du Professeur, lui permet de saisir, modifier et valider les notes des étudiants pour les éléments qu'il prend en charge. Il a également la possibilité d'exporter les données validées sous format PDF ou Excel.

Les règles de gestion de l'application sont les suivantes : les notes doivent être comprises entre 0 et 20, et les étudiants absents à une évaluation doivent être marqués distinctement. Un module ne peut être validé que si tous ses éléments sont validés. Les modalités d'évaluation incluent les CC (Contrôle Continu), TP (Travaux Pratiques) et projets, chacun ayant un coefficient spécifique. Les notes non validées restent modifiables par le professeur.

Concernant les technologies utilisées, le backend de l'application est développé avec le framework Spring boot, tandis que le frontend utilise React pour une interface dynamique. Le design de l'application est réalisé avec Bootstrap et material ui, ce qui garantit un style moderne et responsive pour s'adapter à différentes tailles d'écrans.

Mots clés : Gestion des notes, Administrateurs, Enseignants, Etudiants, Spring boot ,Material ui, React, Bootstrap .

Table des matières

Liste des figures

Introduction Générale

Dans le cadre du module de Programmation Web Java à l'École Nationale des Sciences Appliquées de Khouribga, nous avons été amenés à réaliser une application de gestion des notes avec deux interfaces distinctes : une pour les professeurs et une pour l'administrateur. Cette application vise à gérer efficacement l'ensemble du processus d'évaluation, depuis la configuration des modules jusqu'à l'export des résultats.

Le projet met l'accent sur une gestion rigoureuse des données académiques à travers plusieurs fonctionnalités clés. Pour l'administrateur, l'application offre des outils de gestion des professeurs, des filières, des modules et de leurs éléments, ainsi que la configuration des modalités d'évaluation. Pour les professeurs, elle propose un système intuitif de saisie et de validation des notes, avec des contrôles de cohérence intégrés pour assurer la fiabilité des données.

En s'appuyant sur des technologies modernes comme Spring boot pour le backend et React ViteJs pour le frontend, complétées par Material-UI pour l'interface utilisateur, l'application offre une solution robuste et évolutive. L'architecture choisie permet d'implémenter efficacement les différentes règles de gestion, notamment la gestion des coefficients, la validation des notes, et le calcul automatique des moyennes.

Ce document présente en détail l'analyse, la conception et la réalisation de cette application, en mettant en lumière les choix techniques effectués et les solutions apportées aux différentes contraintes du cahier des charges.

Chapitre 1

Analyse et Conception

Introduction

L'objectif principal de ce projet est le développement d'une application de gestion des notes pour une école, permettant de centraliser et d'automatiser les processus liés à la gestion des professeurs, des modules, des filières, des modalités d'évaluation et des notes des étudiants. Ce système doit offrir une interface utilisateur efficace et sécurisée pour répondre aux besoins variés des différents acteurs, à savoir les administrateurs et les professeurs. La réussite de cette application repose sur une compréhension claire des besoins métier et des règles de gestion spécifiques à l'environnement scolaire. L'analyse approfondie des exigences fonctionnelles et non fonctionnelles permet de définir les fonctionnalités attendues tout en garantissant une architecture logicielle optimisée et évolutive.

Dans cette phase, des diagrammes UML (Unified Modeling Language) sont utilisés pour modéliser les concepts métiers et les relations entre les entités du système, facilitant ainsi une meilleure communication entre les parties prenantes et une conception robuste de l'application. Ce chapitre détaille l'analyse des besoins fonctionnels et non fonctionnels pour identifier les fonctionnalités clés, ainsi que les règles de gestion assurant le respect des contraintes opérationnelles. La modélisation UML, notamment à travers les diagrammes de classes, permet de structurer et formaliser les éléments constitutifs du système, posant ainsi les bases solides d'une application performante, intuitive et conforme aux attentes des utilisateurs.

1.1 Besoins Fonctionnels

Les besoins fonctionnels définissent les fonctionnalités que l'application doit offrir pour répondre aux attentes des utilisateurs. Ils sont décrits comme suit :

- Gestion des Professeurs :
  - Ajouter, modifier, supprimer et consulter les informations des professeurs (nom, prénom, spécialité, code).
  - Gérer les comptes utilisateurs des professeurs avec des droits d'accès sécurisés.

  - Affecter les éléments de module à un professeur.
  - Visualiser les éléments déjà affectés à chaque professeur.

- Gestion des Filières :
  - Ajouter, modifier, supprimer et consulter les filières.
  - Afficher les modules par semestre dans chaque filière.
  - Afficher les étudiants selon leur niveau dans chaque filière.

- Gestion des Éléments de Module :
  - Ajouter, modifier et supprimer les informations des modules.
  - Ajouter des éléments dans un module avec leur coefficient respectif.
  - Afficher les éléments associés à chaque module avec la possibilité de supprimer ou modifier leurs informations.
  - Afficher les modalités d'évaluation de chaque module.
  - Associer des modalités d'évaluation à chaque élément avec leur coefficient (CC, TP, Projet, etc.).

- Gestion des Modalités d'Évaluation :
  - Ajouter, modifier et supprimer les modalités d'évaluation.
  - Afficher les modalités d'évaluation avec leurs coefficients respectifs.

- Saisie et Validation des Notes :
  - Permettre au professeur de :
    * Saisir les notes des éléments qu'il prend en charge.
    * Modifier les notes avant validation si le module n'est pas encore validé.
    * Valider les notes d'un élément, rendant ces notes non modifiables.
  - Afficher la liste des étudiants associés à chaque élément de module selon leur filière et leur niveau.
  - Vérifier la validité des données saisies :
    * Assurer que toutes les notes sont saisies.
    * Garantir que les notes sont comprises entre 0 et 20.
    * Demander une confirmation du professeur en cas de note 0 ou 20.
  - Marquer les étudiants absents avec une note 0, tout en les distinguant de ceux ayant réellement obtenu cette note.
  - Permettre au professeur de consulter la moyenne des notes pour chaque élément de module.

- Calculs Automatiques :
  - Calculer la moyenne d'un élément après validation des notes.
  - Calculer la note finale d'un élément pour un étudiant, en tenant compte des notes saisies pour chaque modalité d'évaluation avec leur coefficient respectif.

- Export des Notes :
  - Exporter les notes d'un élément validé au format Excel et PDF.

1.2 Besoins Non Fonctionnels

Les besoins non fonctionnels concernent les performances, la sécurité et l'ergonomie de l'application. Ils sont détaillés comme suit :

- Sécurité :
  - Authentification des utilisateurs (administrateurs et professeurs) via email et mot de passe.
- Performances :
  - Temps de réponse rapide pour les actions de saisie, consultation et validation des notes.
- Convivialité et Ergonomie :
  - Interface utilisateur intuitive et facile à naviguer pour les administrateurs et les professeurs.
  - Affichage clair et structuré des informations pour une gestion efficace des données.
- Maintenabilité :
  - Code source structuré et modulaire pour faciliter les futures évolutions et la maintenance de l'application.

Ces exigences permettront de garantir une application robuste, performante et conforme aux attentes des utilisateurs finaux.

1.3 Règles de gestion

Les règles de gestion définissent le comportement et les contraintes que l'application doit respecter afin de garantir la cohérence des données et le bon fonctionnement de l'application.

- **Professeurs** :
  - Un professeur peut être affecté à plusieurs éléments de module, appartenant à différentes filières.
- **Filières et Modules** :
  - Un module appartient à une **filière** et à un **semestre** donné (de S1 à S5).
- **Éléments de Module** :
  - Chaque **élément de module** appartient à un **module** et dispose d'un **coefficient**.
  - Le coefficient détermine l'importance de chaque élément dans la note finale du module.

- **Modalités d'Évaluation** :
  - Chaque **élément de module** dispose d'une ou plusieurs **modalités d'évaluation**.
  - Chaque modalité d'évaluation est associée à un **coefficient** qui détermine son poids dans le calcul de la note finale de l'élément.
- **Professeurs et Affectation des Éléments de Module** :
  - Un professeur peut être affecté à un ou plusieurs éléments de module dans différentes filières.
  - Les professeurs sont responsables de la gestion des notes des éléments de module auxquels ils sont affectés.
- **Comptes Utilisateurs** :
  - Les utilisateurs de l'application se connectent via un **login** et un **mot de passe**.
  - Il existe deux types de profils utilisateurs dans l'application :
    - **Administrateur** : Responsable de la gestion des professeurs, des filières, des modules, des éléments de module, des modalités d'évaluation, et des comptes utilisateurs.
    - **Professeur** : Peut saisir, modifier, et valider les notes des éléments de module qu'il prend en charge. Peut aussi exporter les notes dans un fichier **Excel** ou **PDF**.
- **Gestion des Notes** :
  - **Saisie des Notes** :
    - Le professeur doit saisir les notes pour chaque élément de module qu'il prend en charge.
    - Les notes doivent être comprises entre 0 et 20.
    - En cas de note 0 ou 20, l'application doit demander une confirmation au professeur avant la validation des notes.
    - Un étudiant absent doit être marqué comme absent avec une note 0. L'application doit différencier un étudiant absent d'un étudiant ayant réellement obtenu la note 0.
  - **Validation des Notes** :
    - Une fois les notes validées pour un élément de module, celles-ci ne peuvent plus être modifiées.
  - **Calcul de la Moyenne** :
    - L'application doit calculer la moyenne d'un élément de module après la saisie et la validation des notes.
- **Export des Notes** :
  - Les notes d'un élément validé peuvent être exportées dans un fichier **Excel** ou **PDF**.
- **Règles de Validation** :
  - Avant de valider les notes d'un élément, l'application doit vérifier que :
    - Toutes les notes doivent être comprises entre 0 et 20.

* Si des notes de 0 ou de 20 sont présentes, une confirmation doit être demandée au professeur.
* Les étudiants absents doivent être correctement différenciés des étudiants ayant réellement obtenu une note de 0.

1.4 Conception UML

- Diagramme de classe :

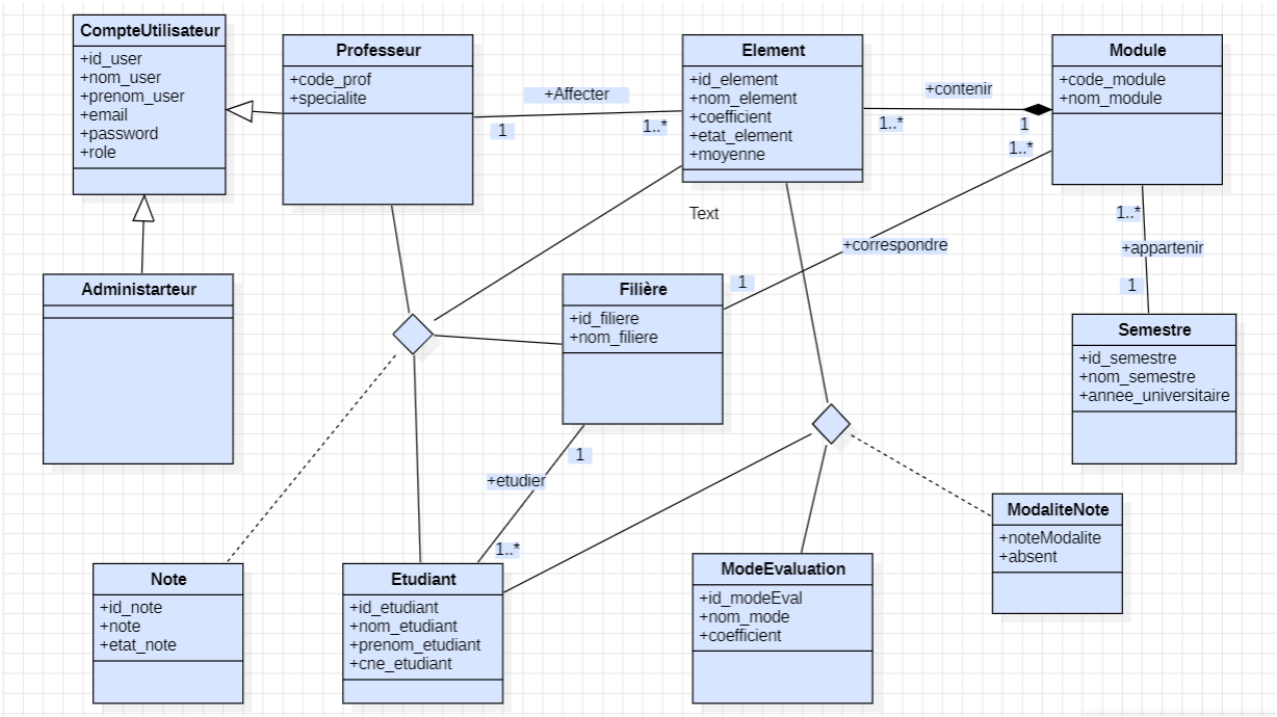

FIGURE 1.1 : Diagramme de classe

Explication du diagramme de classes

- **CompteUtilisateur** : Classe représentant les utilisateurs de l'application, tels que les administrateurs et les professeurs. Elle contient les informations de base sur chaque utilisateur, notamment son nom, prénom, email, mot de passe, et son rôle.
- **Administrateur** : Hérite de `CompteUtilisateur`. Représente les utilisateurs ayant des privilèges d'administration.
- **Professeur** : Représente les professeurs assignés à divers éléments d'enseignement. Cette classe inclut les informations spécifiques telles que le code et la spécialité du professeur.
- **Filière** : Représente les différentes filières d'études disponibles au sein de l'institution, chaque filière ayant un identifiant unique et un nom.
- **Element** : Classe modélisant un cours ou une matière dans un module donné. Elle contient des informations telles que le nom, le coefficient, le statut, et la moyenne obtenue dans l'élément.

- **Module** : Un regroupement d'éléments d'enseignement. Chaque module est identifié de manière unique et porte un nom.
- **Semestre** : Période d'enseignement dans une année universitaire donnée. Chaque semestre est identifié par un nom et lié à une année universitaire spécifique.
- **Etudiant** : Classe représentant les étudiants inscrits. Elle contient des informations telles que le code national étudiant (CNE), le nom et prénom de l'étudiant.
- **Note** : Classe modélisant les notes obtenues par les étudiants. Chaque note a un état (validée ou non) et une valeur.
- **ModeEvaluation** : Représente les différentes méthodes d'évaluation, telles que les examens et les contrôles continus, avec leur coefficient associé.
- **ModaliteNote** : Décrit les modalités spécifiques d'une note, comme l'absence de l'étudiant ou une note attribuée selon une modalité spécifique.

Associations principales

- **Affecter (entre Professeur et Element)** : Un professeur peut être responsable de plusieurs éléments d'enseignement, et chaque élément peut être pris en charge par un professeur.
- **Etudier (entre Filière et Etudiant)** : Une filière peut inclure plusieurs étudiants inscrits.
- **Contenir (entre Module et Element)** : Chaque module regroupe plusieurs éléments d'enseignement.
- **Correspondre (entre Element et Filière)** : Un élément d'enseignement est associé à une filière particulière.
- **Appartenir (entre Semestre et Module)** : Un semestre regroupe plusieurs modules.
- **Héritage (entre CompteUtilisateur, Administrateur et Professeur)** : Les classes `Administrateur` et `Professeur` héritent de la classe `CompteUtilisateur`.

Conclusion

Ce chapitre a permis d'établir une base solide pour le développement de l'application de gestion des notes en analysant minutieusement les besoins fonctionnels et non fonctionnels, ainsi que les règles de gestion spécifiques à l'environnement scolaire. Grâce à la modélisation UML, notamment le diagramme de classes, nous avons structuré et visualisé les entités et leurs relations, facilitant ainsi une compréhension commune du système par toutes les parties prenantes. Ces travaux préliminaires garantissent une conception cohérente et robuste, essentielle pour la réussite des prochaines phases du projet.

Chapitre 2

Implémentation

2.1 Structure du Projet

2.1.1 Organisation du Code Source

Le projet est divisé en deux modules principaux, chacun ayant son rôle distinct :

- **Backend :** L'implémentation se fait avec Spring Boot, qui gère les logiques métier. Les contrôleurs, services et entités sont tous définis dans ce module. Le backend est responsable de la gestion des données et des requêtes HTTP.
- **Frontend :** Le développement frontend se fait avec React, un framework JavaScript qui permet de créer des interfaces utilisateur dynamiques. Le frontend gère la présentation des données et interagit avec le backend via des appels API.

2.1.2 Structure des Dossiers Backend (Spring Boot)

Le backend utilise une structure de projet typique de Spring Boot. Les fichiers sources sont organisés comme suit :

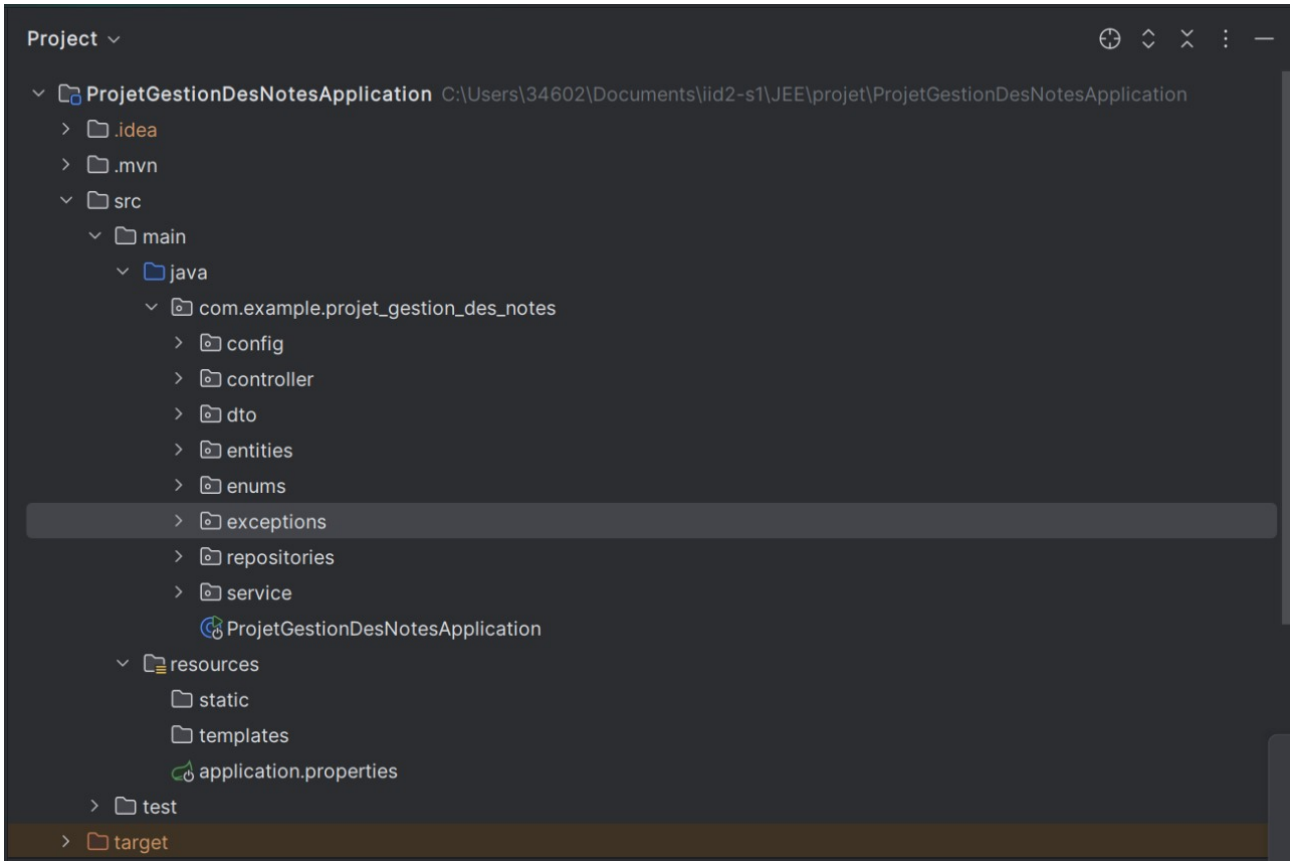

FIGURE 2.1 : Structure des dossiers du backend

La structure du projet suit une architecture en couches typique d'une application Spring Boot, organisée comme suit :

- Le dossier **controller** contient les classes responsables de la gestion des requêtes HTTP.
- Le dossier **dto** (**Data Transfer Objects**) contient les objets de transfert de données.
- Le dossier **entities** contient les classes représentant les entités de l'application.
- Le dossier **exceptions** contient les classes de gestion des exceptions.
- Le dossier **repositories** contient les interfaces d'accès aux données.
- Le dossier **service** contient la logique métier de l'application.
- **application.properties** pour la configurations essentielles pour l'application Spring Boot

Cette organisation suit une architecture en couches avec une séparation claire des responsabilités :

- Une couche présentation (`controllers`)
- Une couche métier (`services`)
- Une couche d'accès aux données (`repositories` et `entities`)

2.1.3 Structure des Dossiers Frontend (React)

Le frontend, développé en React, est structuré de la manière suivante :

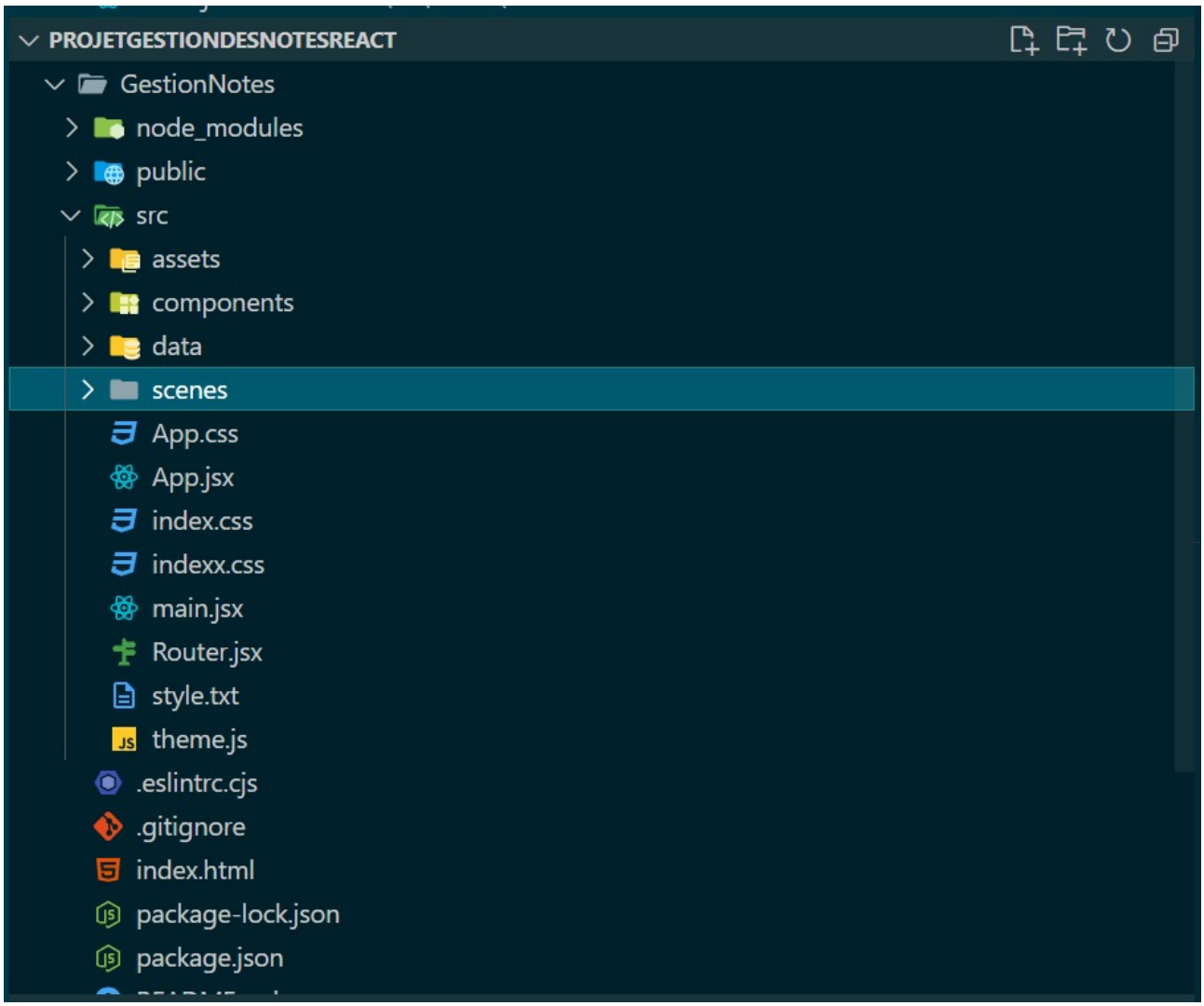

FIGURE 2.2 : Structure des dossiers du frontend

Le projet est structuré de manière organisée afin de faciliter la gestion du code source et des ressources. Voici la description de chaque répertoire important dans la structure du projet.

- **node_modules/** :
  Ce répertoire contient tous les packages npm installés pour le projet. Il est automatiquement généré lors de l'exécution de `npm install` et contient les dépendances nécessaires au projet, telles que React, React-DOM et d'autres bibliothèques externes. Il ne doit pas être modifié manuellement.

- **src/** :
  Le dossier `src` est l'endroit où réside tout le code source de l'application React. Il contient les fichiers JavaScript, CSS et autres qui composent l'application. C'est ici que le développement principal a lieu, et le code à l'intérieur de ce dossier est transformé par des outils comme Webpack lors de la création de la version de production.

- **assets/** :
  Ce répertoire contient probablement des fichiers statiques tels que des images, des polices ou

d'autres ressources multimédia utilisées dans l'application. Ces fichiers sont référencés dans les composants ou les scènes du projet, mais sont stockés séparément pour une meilleure organisation.

- **components/** :
  Le dossier `components` contient des composants réutilisables dans React. Ces composants sont des éléments modulaires de l'interface utilisateur, tels que des boutons, des champs de saisie, des en-têtes et des pieds de page, qui peuvent être utilisés sur plusieurs pages ou scènes de l'application.

- **scenes/** :
  Le dossier `scenes` contient les scènes principales ou les pages de l'application. Chaque scène représente une vue ou une section différente de l'application, telle qu'un tableau de bord, un profil utilisateur ou la page d'accueil. Ces scènes sont composées de plusieurs composants qui sont assemblés pour créer la page complète.

2.1.4 Configuration des Dépendances

2.1.4.1 Configuration de l'Application

Le fichier **application.properties** contient des configurations essentielles pour l'application Spring Boot, telles que les paramètres de la base de données, le port du serveur, et d'autres configurations spécifiques à l'application.

- **Nom de l'application :** Le nom de l'application est défini avec `spring.application.name`.
- **Configuration de la base de données :** La configuration de la base de données est définie sous `spring.datasource`. Elle comprend l'URL de la base de données, le nom d'utilisateur, le mot de passe, ainsi que le pilote JDBC utilisé.
- **Hibernate :** La gestion des entités est configurée avec `spring.jpa`, où `hibernate.ddl-auto` spécifie l'action à effectuer sur le schéma de la base de données.
- **Logs :** La configuration des logs permet de suivre les actions réalisées dans le cadre des requêtes HTTP et des actions Hibernate pour faciliter le débogage.

Voici le contenu du fichier `application.properties` :

```
spring.application.name=ProjetGestionDesNotesApplication

# Configuration de la base de données
spring.datasource.url=jdbc:mysql://localhost:4306/gestionnotess?
   createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
server.port=8080
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
```

```
12 logging.level.org.springframework.web=DEBUG
13 logging.level.org.springframework.web.servlet=DEBUG
14 logging.level.org.springframework.http=DEBUG
15 spring.jpa.properties.hibernate.format_sql=true
16
17 spring.main.allow-circular-references=true
18 logging.level.org.hibernate.type.descriptor.sql=TRACE
```

=> Explications :

- **spring.datasource.url :** Spécifie l'URL de connexion à la base de données MySQL, incluant le nom de la base de données `gestionnotess` et l'option `createDatabaseIfNotExist=true` pour créer la base si elle n'existe pas.
- **spring.datasource.username et spring.datasource.password :** Ces propriétés sont utilisées pour définir les informations d'authentification à la base de données.
- **spring.jpa.hibernate.ddl-auto :** Cette propriété contrôle la façon dont Hibernate gère les changements de schéma. Ici, `update` indique que Hibernate mettra automatiquement à jour la base de données en fonction des entités définies.
- **logging.level.org.springframework.web et autres :** Ces propriétés permettent de configurer le niveau de journalisation des actions effectuées par Spring. Le niveau `DEBUG` permet de voir des détails supplémentaires dans les logs.
- **spring.jpa.properties.hibernate.FormatSql** :Cette propriété permet de formater le SQL généré par Hibernate pour une meilleure lisibilité dans les logs.
- **spring.main.allow-circular-references :** Cette option permet à Spring de gérer les références circulaires dans les beans.
- **logging.level.org.hibernate.type.descriptor.sql :** Définit le niveau de trace des requêtes SQL pour plus de détails lors du débogage.

2.1.4.2 Configuration du frontend (React)

Le fichier de configuration du frontend, tel que **package.json**, joue un rôle clé dans la gestion des dépendances et la configuration de l'application React. Ce fichier répertorie toutes les bibliothèques et les outils nécessaires pour faire fonctionner l'application, tels que les frameworks de routage, les bibliothèques de gestion des requêtes API, et les outils de développement comme Webpack et Babel. Il permet également de définir des scripts personnalisés, tels que ceux pour démarrer le serveur de développement ou pour générer des builds de production. Le fichier **package.json** inclut également des informations importantes sur le projet, telles que son nom, sa version et ses points d'entrée. En parallèle, le fichier **package-lock.json** est utilisé pour verrouiller les versions des dépendances, garantissant ainsi que l'installation des modules reste cohérente sur toutes les machines de développement, ce qui est essentiel pour éviter des conflits ou des erreurs liées aux versions.

2.2 Frontend (React ViteJs)

2.2.1 Composants Principaux

Dans une application React, les composants principaux sont responsables de l'affichage des différentes sections et de la gestion de l'état global de l'application. Chaque composant peut utiliser des hooks comme `useState` pour gérer l'état local, ainsi que `useEffect` pour effectuer des appels API ou des actions lors du montage du composant.

Le composant `AppRouter` utilise `React Router` pour définir les différentes routes et permettre la navigation entre les pages.

Le composant principal `Professors` est utilisé pour afficher la liste des professeurs dans l'application. Ce composant exploite des hooks React, comme `useState` et `useEffect`, pour gérer l'état local et effectuer des appels API afin de récupérer les données des professeurs. Il fait également partie de la gestion de la navigation dans l'application, étant intégré dans le composant `AppRouter`, qui configure toutes les routes de l'application.

Voici un exemple de code pour le composant `Professors` :

Listing 2.1 : Composant Professors

```
import { useState, useEffect } from 'react';
import axios from 'axios';

const Professors = () => {
    const [professorsList, setProfessorsList] = useState([]);
    const [loading, setLoading] = useState(true);
    const [error, setError] = useState(null);

    const fetchProfessors = () => {
        setLoading(true);
        setError(null);
        axios.get('http://localhost:8080/api/professors')
        .then(response => {
            setProfessorsList(response.data);
            setLoading(false);
        })
        .catch(error => {
            setError('Error fetching professors');
            setLoading(false);
        });
    };

    useEffect(() => {
        fetchProfessors();
    }, []);

```

```
    if (loading) return <p>Loading...</p>;
    if (error) return <p>{error}</p>;

    return (
        <div>
            <h1>Professors List</h1>
            <ul>
                {professorsList.map(professor => (
                    <li key={professor.id}>{professor.name}</li>
                ))}
            </ul>
        </div>
    );
};

export default Professors;
```

Ce composant effectue une requête GET pour récupérer la liste des professeurs depuis le backend. L'état `professorsList` est mis à jour avec les données reçues. Il gère également l'état de chargement (`loading`) et d'erreur (`error`) pour améliorer l'expérience utilisateur.

2.2.2 Tables et Affichage des Données

L'affichage des données dans un tableau est une pratique courante pour organiser les informations de manière claire et structurée. Dans cette section, nous allons illustrer comment utiliser la bibliothèque `DataGrid` de `Material-UI` pour afficher une liste d'étudiants dans un tableau interactif. Cette approche inclut des fonctionnalités telles que la sélection de lignes, un en-tête de colonne dynamique, et un style conditionnel basé sur l'état des éléments.

Voici un exemple de code React qui utilise le `DataGrid` de `Material-UI` pour afficher les étudiants :

Listing 2.2 : Exemple d'Affichage des Données avec DataGrid

```
<Box display="flex" justifyContent="center" alignItems="center" height
   ="75vh">
  <Box flex={1} sx={{ maxWidth: '90%' }} height="100%">
    <DataGrid
      rows={students}
      columns={columns}
      getRowId={(row) => row.idEtudiant}
      components={{ Toolbar: GridToolbar }}
      checkboxSelection
      sx={{
        '& .MuiDataGrid-root': { border: 'none' },
        '& .MuiDataGrid-cell': { border: 'none' },
        '& .MuiDataGrid-columnHeaders': {
          backgroundColor: isElementValidated ? '#e8f5e9' : '#d8eaf4',
```

```
14            borderBottom: 'none',
15          },
16          '& .MuiDataGrid-footerContainer': {
17            borderTop: 'none',
18            backgroundColor: isElementValidated ? '#e8f5e9' : '#d8eaf4',
19          },
20          '& .MuiDataGrid-virtualScroller': {
21            backgroundColor: colors.primary[400],
22          },
23          '& .MuiCheckbox-root': {
24            color: `${colors.greenAccent[200]} !important`,
25          },
26          '& .MuiDataGrid-toolbarContainer .MuiButton-text': {
27            color: `${colors.gray[100]} !important`,
28          },
29        }}
30      />
31    </Box>
32  </Box>
```

Dans cet exemple :

- Le tableau utilise `DataGrid` pour afficher les lignes (`rows`) et les colonnes (`columns`).
- La sélection de cases à cocher (`checkboxSelection`) permet de sélectionner plusieurs lignes dans le tableau.
- La couleur d'arrière-plan des en-têtes de colonnes et du pied de page est dynamique, en fonction de l'état de validation de l'élément (`isElementValidated`).
- Le tableau a un style personnalisé pour les cellules, les en-têtes, le pied de page et les boutons de la barre d'outils, ce qui améliore l'expérience utilisateur.
- Les données sont affichées de manière responsive grâce à l'utilisation de la propriété

Cette approche garantit que les utilisateurs peuvent facilement naviguer et interagir avec les données dans le tableau tout en ayant une interface esthétique et fonctionnelle.

L'utilisation de `DataGrid` permet également une gestion avancée des tableaux, notamment le filtrage, la pagination et l'édition des cellules. Ces fonctionnalités peuvent être facilement intégrées en fonction des besoins de l'application.

2.2.3 Gestion d'État et Utilisation des Hooks (useState, useEffect)

La gestion d'état dans une application React fait référence à la façon dont les données sont stockées, mises à jour et partagées à travers les différents composants d'une application. Cela permet à un composant de répondre aux actions des utilisateurs ou à d'autres événements, et d'afficher des informations dynamiquement en fonction de l'état actuel de l'application.

Dans le composant `Professors`, l'état est géré à l'aide de `useState`. Cela permet de suivre les modifications des données et de rendre le composant interactif.

Exemple de gestion de l'état :

Listing 2.3 : Gestion de l'État dans le Composant Professors

```
const [professorsList, setProfessorsList] = useState([]);
const [selectedProfessorCode, setSelectedProfessorCode] = useState(null);
```

Ici, `professorsList` contient les données des professeurs, et `selectedProfessorCode` est utilisé pour suivre le professeur sélectionné.

En plus de `useState`, nous utilisons également `useEffect` pour effectuer des appels API et synchroniser l'état avec les données externes. Par exemple, lorsqu'un composant est monté, nous effectuons une requête pour récupérer la liste des professeurs.

Listing 2.4 : Utilisation de useState et useEffect pour la gestion des données

```
const [professorsList, setProfessorsList] = useState([]);
const [loading, setLoading] = useState(true);

useEffect(() => {
    fetchProfessors();
}, []);
```

Les hooks en React sont des fonctions qui permettent d'ajouter des fonctionnalités d'état et d'effets secondaires aux composants fonctionnels.

`useState` gère l'état local du composant, tandis que `useEffect` permet d'exécuter des effets secondaires comme les appels API lors du montage du composant, en garantissant que les données externes sont récupérées et utilisées dans le composant.

2.2.4 Intégration API

L'intégration avec l'API se fait principalement via `axios`, qui envoie des requêtes HTTP au backend. Voici un exemple de requête pour récupérer la liste des professeurs :

Listing 2.5 : Requête API pour récupérer les Professeurs

```
const fetchProfessors = () => {
    axios.get('http://localhost:8080/api/professors')
    .then(response => {
        setProfessorsList(response.data);
    })
    .catch(error => {
        setError('Error fetching professors');
    });
};
```

2.2.5 Gestion des Requêtes axios

Les requêtes avec `axios` sont courantes dans les applications React pour récupérer des données depuis un serveur. Voici comment effectuer une requête GET avec gestion d'erreurs :

Listing 2.6 : Requête avec axios

```
axios.get('http://localhost:8080/api/professors')
.then(response => {
    setProfessorsList(response.data);
})
.catch(error => {
    console.error('Error fetching professors:', error);
});
```

2.2.6 Gestion des Erreurs

Les erreurs peuvent survenir lors des appels API. Il est important de les capturer pour informer l'utilisateur et faciliter le débogage. Voici comment gérer les erreurs dans une requête API :

Listing 2.7 : Gestion des Erreurs dans la Requête API

```
axios.get('http://localhost:8080/api/professors')
.then(response => {
    setProfessorsList(response.data);
})
.catch(error => {
    setError('Error fetching professors');
});
```

Dans cet exemple, si une erreur survient pendant la requête, elle sera affichée dans la console, ce qui aidera les développeurs à diagnostiquer et résoudre les problèmes.

2.2.7 Loading State

Le `loading state` est important pour afficher un indicateur de chargement à l'utilisateur pendant que les données sont récupérées.

Exemple de gestion de l'état de chargement :

Listing 2.8 : Gestion du Loading State

```
const [loading, setLoading] = useState(true);

useEffect(() => {
    fetchProfessors();
}, []);

if (loading) return <p>Loading...</p>;
```

Cela permet d'afficher un message ou un indicateur de chargement tant que les données ne sont pas prêtes.

2.2.8 Utilisation de LocalStorage

L'application utilise le `localStorage` pour stocker et récupérer les informations essentielles sur l'utilisateur après une connexion réussie. En particulier, l'ID de l'utilisateur (qui dans ce cas est

celui du professeur) est récupéré depuis `localStorage` et utilisé pour charger dynamiquement les données spécifiques, telles que les semestres, les filières, et les éléments affectés à un professeur.

Voici un exemple de code permettant de récupérer l'ID du professeur :

Listing 2.9 : Gestion du LocalStorage

```
const professorId = localStorage.getItem("userId");
```

2.3 Backend (Spring Boot)

2.3.1 Implémentation des entités JPA

Les entités JPA (Java Persistence API) jouent un rôle central dans le développement d'applications utilisant Spring Boot. Elles représentent les objets métier mappés à des tables dans une base de données relationnelle, facilitant ainsi les opérations de persistance et d'interaction avec les données. Chaque entité est une classe annotée qui définit les attributs correspondant aux colonnes de la table, ainsi que les relations avec d'autres entités.

Dans cette section, nous illustrons l'implémentation des entités avec un exemple concret : l'entité Element, qui représente un concept clé du système. Cette entité intègre des annotations JPA pour le mappage et des relations permettant de structurer les données de manière cohérente.

Listing 2.10 : Entité Element

```
package com.example.projet_gestion_des_notes.entities;

import jakarta.persistence.*;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;

@Entity
public class Element {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idElement;

    private String nomElement;
    private Double coefficient;
    private String etatElement;
    private Double moyenne;
    @ManyToOne
    @JoinColumn(name="idProf")
    private Professeur professeur;

    @OneToMany(mappedBy = "element", cascade = CascadeType.ALL)
```

```
24      private List<ModaliteNotee> modaliteNotes;
25
26      @ManyToMany(fetch=FetchType.EAGER)
27      @JoinTable(
28              name = "avoir",
29              joinColumns = @JoinColumn(name = "idElement"),
30              inverseJoinColumns = @JoinColumn(name = "idModeEval")
31      )
32      private Collection <ModeEvaluation> modesEvaluation = new ArrayList
           <>();
33
34
35      @ManyToOne
36      @JoinColumn(name = "idModule")
37      private Module module;
38
39
40      @OneToMany(mappedBy = "element",cascade = CascadeType.REMOVE)
41      private Collection<Note> notes;
42  }
```

L'entité `Element` représente un élément pédagogique dans le contexte de gestion des notes. Voici les points clés de cette implémentation :

- **Annotations JPA** :
  - `@Entity` : Indique que cette classe est une entité JPA et sera mappée à une table dans la base de données.
  - `@Id` : Spécifie que l'attribut `idElement` est la clé primaire.
  - `@GeneratedValue(strategy = GenerationType.IDENTITY)` : Configure la stratégie de génération automatique des valeurs pour la clé primaire.
- **Relations entre entités** :
  - `@ManyToOne` : Utilisé pour les relations plusieurs-à-un. Par exemple, un `Element` est lié à un seul `Professeur` et à un seul `Module`.
  - `@OneToMany(mappedBy = "element")` : Définit une relation un-à-plusieurs, comme la relation entre `Element` et `ModaliteNotee`, ou `Element` et `Note`.
  - `@ManyToMany(fetch = FetchType.EAGER)` : Utilisé pour la relation plusieurs-à-plusieurs entre `Element` et `ModeEvaluation`, avec une table de jonction appelée `avoir`.
- **Attributs principaux** :
  - `nomElement` : Le nom de l'élément pédagogique.
  - `coefficient` : Le coefficient associé à l'élément.
  - `etatElement` : Indique l'état de l'élément (par exemple, actif ou inactif).

- `moyenne` : La moyenne calculée pour cet élément.

- **Constructeurs et méthodes** :
  - Le constructeur par défaut est défini pour JPA.
  - Un constructeur paramétré est inclus pour faciliter la création d'objets `Element`.
  - Des méthodes `getter` et `setter` sont définies pour gérer les attributs et assurer la cohérence des relations, comme dans `setProfesseur()`, où l'association bidirectionnelle est gérée.

Cette entité intègre également des annotations comme `@JoinColumn` et `@JoinTable` pour configurer précisément les relations entre les tables dans la base de données.

2.3.2 Couche DTO

La couche DTO (Data Transfer Object) joue un rôle crucial dans l'architecture de l'application en facilitant le transfert de données entre les différentes couches, tout en maintenant une séparation claire des préoccupations. Contrairement aux entités JPA, qui sont directement liées à la base de données, les DTO sont des objets conçus spécifiquement pour transporter des données nécessaires à une vue ou une fonctionnalité donnée.

L'utilisation des DTO permet de limiter l'exposition des entités JPA, de réduire la surcharge des données transférées et de personnaliser les structures selon les besoins de l'interface utilisateur ou des services.

Dans cette section, nous présentons l'implémentation d'un DTO pour l'entité `Element`. Ce DTO contient des attributs spécifiques, comme le nom et le prénom du professeur associé, ainsi que des détails sur les modes d'évaluation, pour fournir une vue simplifiée et adaptée de l'objet métier.

Listing 2.11 : Element DTO

```
import java.util.List;

public class ElementDTO {

    private Long elementId;
    private String elementName;
    private Double coefficient;
    private String etatElement;
    private String professeurNom;
    private String professeurPrenom;
    private List<String> modesNoms;
    private List<Double> modesCoefficients;
    private Long professeurId;

    public Long getProfesseurId() {
        return professeurId;
    }
    public void setProfesseurId(Long professeurId) {
        this.professeurId = professeurId;
```

```
20      }
21
22      public List<String> getModesNoms() {
23          return modesNoms;
24      }
25      public void setModesNoms(List<String> modesNoms) {
26          this.modesNoms = modesNoms;
27      }
```

2.3.3 Couche Repository

La couche Repository dans une application Spring Boot est responsable de l'accès aux données et de leur gestion dans la base de données. En étendant l'interface `JpaRepository`, les repositories offrent des méthodes intégrées pour effectuer des opérations CRUD.

Dans cette section, nous présentons le repository de l'entité `Element`, qui inclut des méthodes personnalisées pour répondre aux besoins spécifiques du système.

Listing 2.12 : Element Repository

```
1  import java.util.List;
2
3  @Repository
4  public interface ElementRepository extends JpaRepository<Element, Long> {
5      @Query("SELECT e.nomElement, p.codeProf, e.idElement FROM Element e
           JOIN e.professeur p")
6      List<Object[]> findElementNamesAndProfessors();
7
8      @Query("SELECT e FROM Element e WHERE e.professeur IS NULL")
9      List<Element> findUnassignedElements();
10     List<Element> findByModule(Module module);
```

Voici une description des points principaux de l'implémentation du `ElementRepository` :

- **`findElementNamesAndProfessors()`** : Une requête JPQL qui retourne une liste d'objets contenant les noms des éléments, les codes des professeurs associés, et les identifiants des éléments. Cette méthode est utile pour des affichages spécifiques ou des rapports.
- **`findUnassignedElements()`** : Cette méthode retourne les éléments qui ne sont pas encore assignés à un professeur (`professeur IS NULL`).
- **`findByModule(Module module)`** : Une méthode personnalisée générée automatiquement par Spring Data JPA, qui récupère tous les éléments associés à un module donné.

Ces méthodes illustrent comment combiner la puissance de Spring Data JPA et des requêtes JPQL pour répondre aux besoins complexes d'accès aux données.

2.3.4 Couche Service

La couche Service dans une application Spring Boot représente la logique métier. Elle agit comme un intermédiaire entre les couches Controller et Repository, en orchestrant les opérations nécessaires et en appliquant les règles métier. Dans cette section, nous présentons un exemple de service pour l'entité `Element`, qui utilise les repositories pour récupérer et transformer les données en objets `DTO`.

Listing 2.13 : Element Service

```
@Service
@Transactional
public class ElementService {

    @Autowired
    private ElementRepository elementRepository;
    @Autowired
    private CompteUtilisateurRepository compteUtilisateurRepository;

    public List<ElementDTO> getElementsByModuleId(Long moduleId) {
        List<Element> elements = elementRepository.findElementsByModuleId
           (moduleId);

        return elements.stream().map(element -> {
            String professeurNom = element.getProfesseur() != null ?
               element.getProfesseur().getNomUser() : null;
            String professeurPrenom = element.getProfesseur() != null ?
               element.getProfesseur().getPrenomUser() : null;

            return new ElementDTO(
                    element.getIdElement(),
                    element.getNomElement(),
                    element.getCoefficient(),
                    element.getEtatElement(),
                    professeurNom,
                    professeurPrenom
            );
        }).collect(Collectors.toList());
    }
```

Les points clés du code `ElementService` sont les suivants :

- **`@Service`** : Indique que cette classe est un service Spring, permettant son injection dans d'autres composants.
- **`@Transactional`** : Garantit que les opérations de la méthode se déroulent dans un contexte transactionnel, assurant ainsi la cohérence des données.
- **`getElementsByModuleId(Long moduleId)`** : Méthode qui récupère les éléments liés à un

module donné et les transforme en objets `ElementDTO`, tout en gérant des cas spécifiques comme l'absence d'un professeur assigné.

2.3.5 Couche Controller

La couche Controller dans une application Spring Boot gère les requêtes HTTP et sert de point d'entrée pour les interactions avec le système. En utilisant les annotations Spring, cette couche mappe les routes, valide les données d'entrée, et invoque les services nécessaires pour répondre aux requêtes. Ci-dessous, un exemple de controller pour l'entité `Element`.

Listing 2.14 : Element Controller

```
@RestController
@RequestMapping("/api/element")
@CrossOrigin(origins = "http://localhost:5173")
public class ElementControler {

    @Autowired
    private ElementService elementService;
    @Autowired
    private ElementRepository elementRepository;

    @GetMapping("/average/{idElement}")
    public ResponseEntity<Double> getElementAverage(@PathVariable Long
       idElement) {
        Double average = elementService.calculateElementAverage(idElement
           );
        return ResponseEntity.ok(average);
    }
```

Les points clés du code `ElementController` sont les suivants :

- **`@RestController`** : Marque la classe comme un contrôleur Spring, permettant de gérer les requêtes REST.
- **`@RequestMapping("/api/element")`** : Définit le préfixe de l'URL pour toutes les routes de ce contrôleur.
- **`@CrossOrigin(origins = "http://localhost:5173")`** : Autorise les requêtes cross-origin depuis une application frontend spécifique (souvent utilisée pour le développement local).
- **`@GetMapping("/average/idElement")`** : Associe une requête HTTP GET à la méthode `getElementAverage`, qui retourne la moyenne d'un élément donné en réponse à l'URL correspondante.
- **`ResponseEntity`** : Fournit une manière standard d'encapsuler les réponses HTTP, ici pour retourner un statut `200 OK` avec la moyenne calculée.

2.4 Conclusion

Dans ce chapitre, nous avons exploré en profondeur la structure du projet, la gestion des dépendances et l'implémentation des principales fonctionnalités dans le frontend avec React, ainsi que l'intégration avec le backend via des requêtes API. Nous avons mis l'accent sur la gestion des erreurs, l'état avec les hooks, et les meilleures pratiques comme l'utilisation du Context API ou de Redux pour la gestion de l'état global. Cette approche modulaire et flexible permet de maintenir et d'étendre facilement le projet.

Nous avons également détaillé l'implémentation du backend avec Spring Boot, en commençant par les entités JPA, comme l'entité **Element**, qui représente des concepts clés du système et utilise des annotations pour gérer les relations entre les tables. La couche DTO a été introduite pour séparer les préoccupations entre les entités JPA et les données transférées, offrant une vue simplifiée et adaptée. La couche Repository, qui gère l'accès aux données à travers des méthodes personnalisées, et la couche Service, qui orchestre la logique métier, ont également été explorées pour illustrer comment les données sont manipulées. Enfin, la couche Controller gère les requêtes HTTP et permet de répondre aux actions demandées via des points d'entrée API.

Grâce à cette architecture bien structurée, le projet offre une solution robuste, évolutive et facile à maintenir, tout en garantissant une interaction fluide entre le frontend et le backend.

Chapitre 3

Interfaces et Ergonomie

3.1 Introduction

Ce chapitre se concentre sur les interfaces utilisateurs de l'application, en mettant en avant leur conception, leur ergonomie, et leur rôle dans la navigation et l'interaction. Une interface utilisateur bien pensée est essentielle pour garantir une expérience fluide et intuitive, en particulier dans un système où les utilisateurs ont des rôles distincts, tels que les administrateurs et les professeurs.

Nous commencerons par présenter les pages générales accessibles à tous les utilisateurs, à savoir la page d'accueil et la page de connexion. Ensuite, nous détaillerons les interfaces spécifiques à chaque type d'utilisateur, accompagnées de captures d'écran, de descriptions et d'un aperçu des flux de navigation. Cette structure vise à démontrer comment l'application répond aux besoins des utilisateurs tout en maintenant une interface cohérente et ergonomique.

3.2 Page d'accueil (HomePage)

La page d'accueil constitue l'entrée principale de l'application et offre une vue globale de ses fonctionnalités. Elle accueille les utilisateurs avec un message chaleureux et un design attrayant. Cette interface inclut une barre de navigation permettant d'accéder aux sections essentielles telles que les événements, les clubs, les questions fréquemment posées (FAQ) et les contacts. De plus, un bouton `Login` est disponible pour diriger les utilisateurs vers la page de connexion. La page met également en avant des éléments comme les événements à venir, les clubs disponibles et les ressources de soutien pour les étudiants, grâce à des cartes informatives et des boutons interactifs. Le tout est conçu pour offrir une expérience intuitive et engageante.

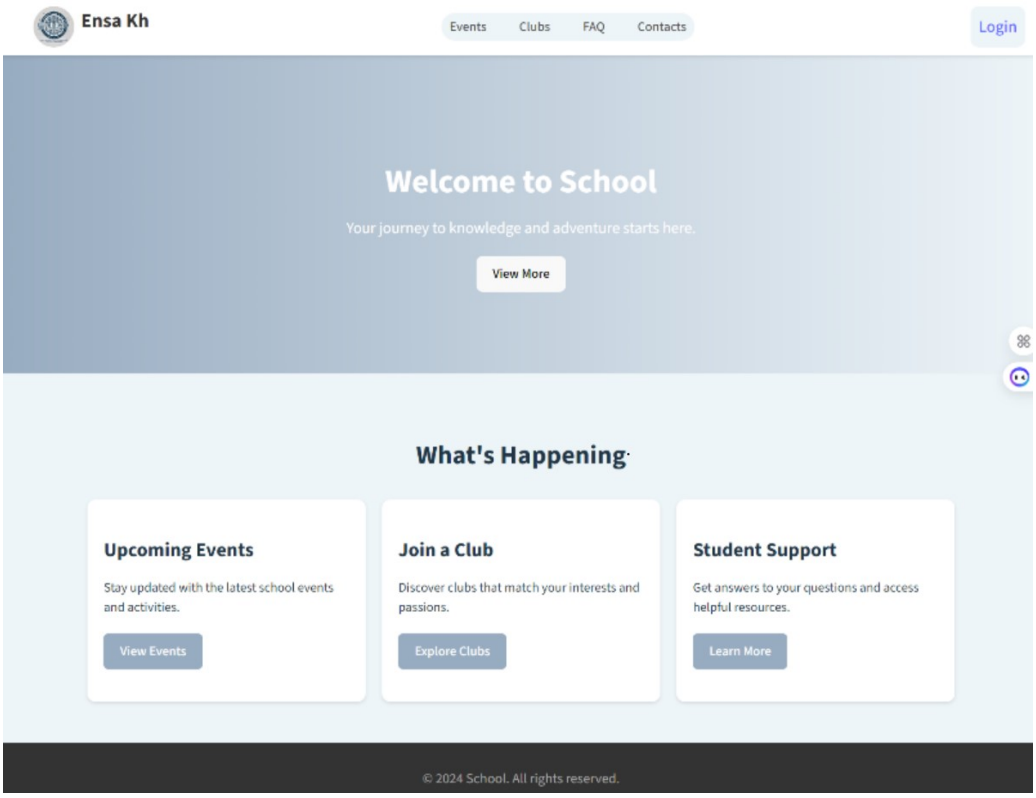

FIGURE 3.1 : Capture d'écran de la page d'accueil

3.3 Page de connexion (Login)

La page de connexion constitue une étape cruciale pour garantir un accès sécurisé aux fonctionnalités spécifiques selon le rôle de l'utilisateur. Elle permet aux utilisateurs de saisir leurs identifiants, à savoir leur adresse e-mail et leur mot de passe, pour s'authentifier. En cas de succès, les utilisateurs sont redirigés vers leur interface respective, qu'il s'agisse de celle de l'administrateur ou du professeur. La page de connexion inclut également des messages d'erreur en cas de problème, comme des identifiants incorrects ou des erreurs de réseau, afin de guider l'utilisateur. Enfin, un lien pour réinitialiser le mot de passe est proposé, renforçant ainsi l'aspect pratique et sécurisé de cette interface.

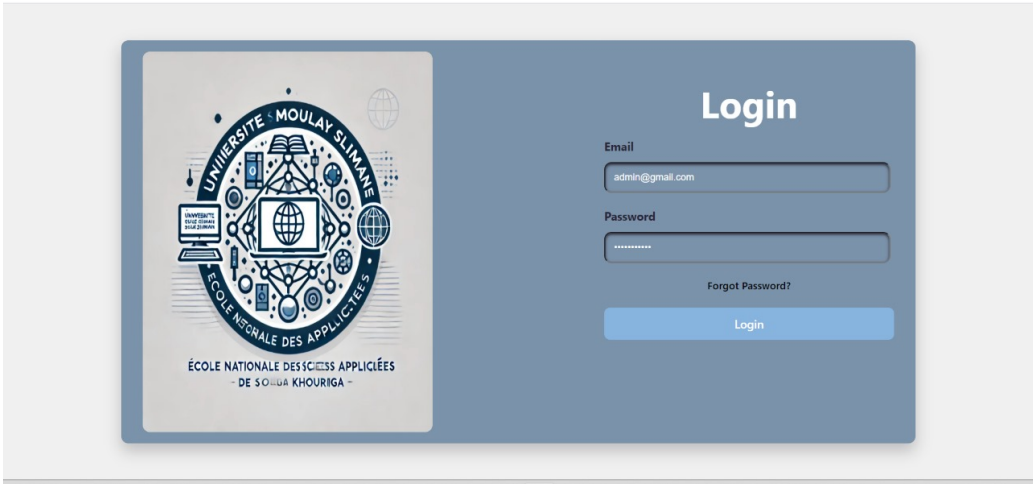

FIGURE 3.2 : Capture d'écran de la page de connexion

3.4 Interface Administrateur

La section administrateur est dédiée à la gestion et à l'administration de la plateforme, permettant à l'administrateur de gérer plusieurs aspects essentiels du système. Il doit pouvoir gérer les professeurs (ajout, modification, suppression), les filières, les modules et leurs éléments (cours, examens, etc.), ainsi que les modalités d'évaluation. L'administrateur est également responsable de l'affectation des éléments aux professeurs et de la gestion des comptes utilisateurs, y compris la création, la modification et la suppression des comptes utilisateurs.

3.4.1 Icônes et Menu Utilisateur

La section des boutons d'icônes et du menu utilisateur permet une gestion fluide des interactions dans l'interface utilisateur. Trois icônes principales sont utilisées : une pour basculer entre le mode sombre et le mode clair (en fonction du thème actuel), une autre pour accéder aux paramètres, et enfin une icône de profil utilisateur. Lorsque l'utilisateur clique sur l'icône de profil, un menu déroulant s'ouvre, affichant des informations telles que le nom de l'utilisateur et son adresse e-mail. Ce menu permet également à l'utilisateur de se déconnecter de la plateforme via un bouton dédié. Cette fonctionnalité améliore l'expérience utilisateur en offrant un accès rapide aux paramètres de l'interface et à la gestion du compte personnel.

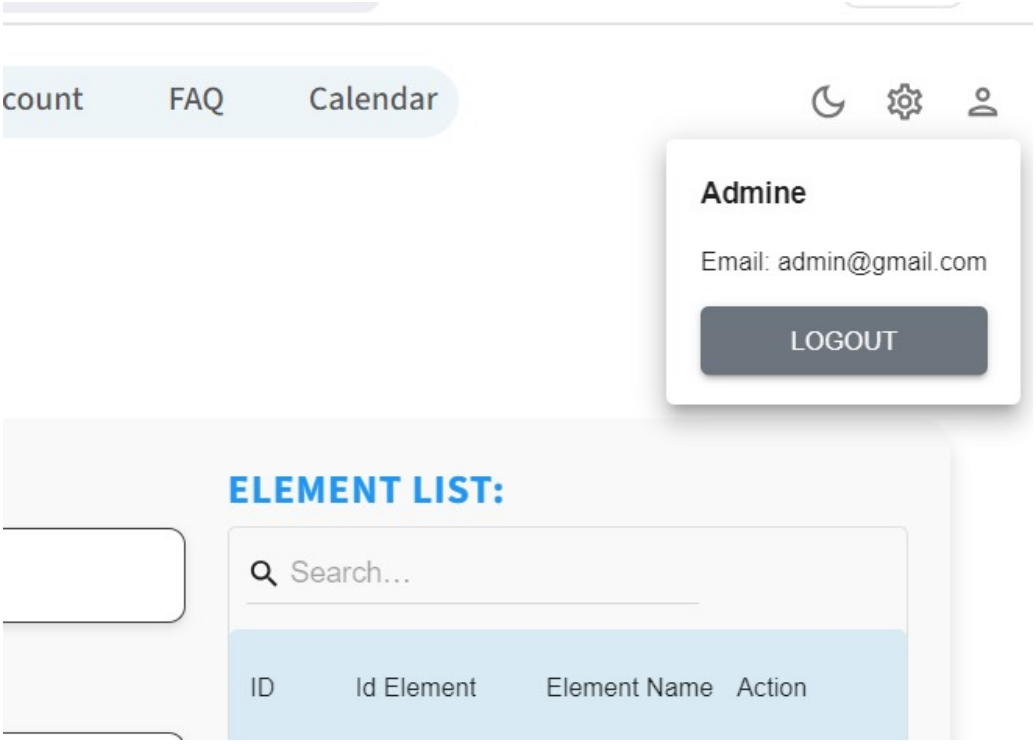

FIGURE 3.3 : Capture d'écran du button Profil

3.4.2 Gestion des professeurs

L'administrateur a la possibilité de gérer les informations des professeurs, telles que leurs noms, e-mails, spécialité et un code professeur unique. Il peut également créer, modifier et supprimer des comptes utilisateurs, ainsi qu'affecter des éléments à chaque professeur. L'interface a été divisée en quatre parties principales

3.4.2.1 Ajout et modification de professeurs

Cette partie permet à l'administrateur d'ajouter de nouveaux professeurs ou de modifier les informations des professeurs existants.

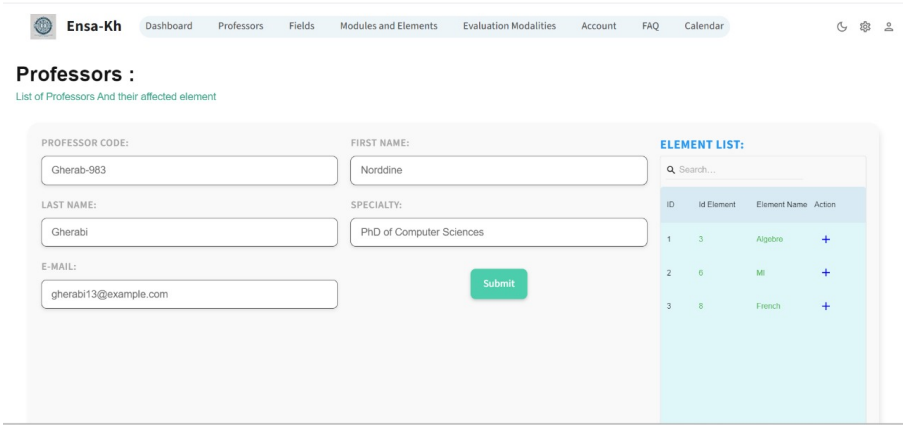

FIGURE 3.4 : Écran d'ajout et modification des professeurs

L'administrateur peut ajouter, modifier et supprimer des professeurs. Voici des exemples d'alertes visuelles correspondant à chaque action :

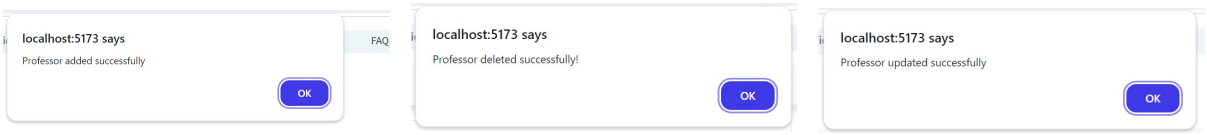

FIGURE 3.5 : Alertes d'ajout, suppression et modification des professeurs

- **Ajout** : L'alerte montre que l'ajout d'un professeur a été effectué avec succès.
- **Suppression** : L'alerte indique que le professeur a été supprimé.
- **Modification** : L'alerte confirme que les informations du professeur ont été mises à jour.

3.4.2.2 Éléments non affectés

Cette section affiche tous les éléments qui n'ont pas encore été affectés à aucune professeur, permettant à l'administrateur de les attribuer facilement.

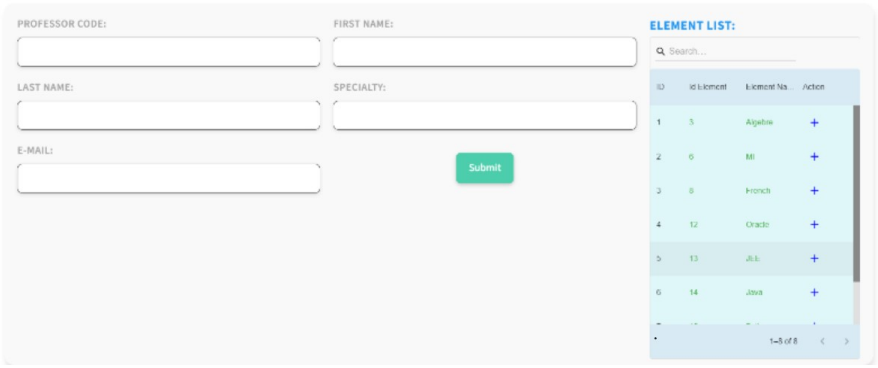

FIGURE 3.6 : Éléments non affectés à un professeur

3.4.2.3 Affichage de la Liste des professeurs

Dans cette section, l'application affiche une liste de tous les professeurs inscrits, accompagnée d'une colonne d'actions qui permet à l'administrateur de supprimer ou de modifier les informations des professeurs. L'affichage des professeurs ainsi que l'affectation des éléments à un professeur sont gérés via le composant `DataGrid`, qui offre plusieurs fonctionnalités puissantes.

L'une des fonctionnalités importantes est la possibilité de filtrer les données rapidement grâce à la barre d'outils de filtrage `GridToolbarQuickFilter`. Ce composant permet aux utilisateurs d'appliquer des filtres sur les colonnes du tableau, facilitant ainsi la recherche de données spécifiques. En outre, le 'DataGrid' permet de télécharger les données sous différents formats. Ces fonctionnalités sont accessibles via l'interface de la barre d'outils, qui inclut des boutons dédiés au filtrage rapide et à l'exportation des données.

Le tableau utilise également la pagination pour gérer les grandes quantités de données et affiche des informations de manière claire grâce à une interface bien conçue. Les colonnes sont stylisées avec des couleurs distinctes pour faciliter la lecture, et un système de sélection de lignes est mis en place pour des actions rapides sur les données sélectionnées.

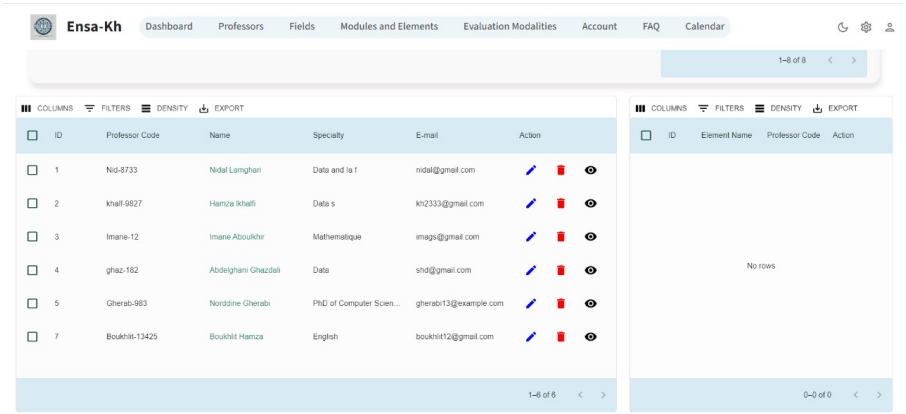

FIGURE 3.7 : Liste des professeurs avec options d'actions

3.4.2.4 Éléments affectés à un professeur

Lorsqu'un professeur est sélectionné, cette partie affiche les éléments qui lui ont été attribués et on peut supprimer un element affecter a un professeurs. Cela permet à l'administrateur de visualiser et de gérer facilement les responsabilités d'enseignement.

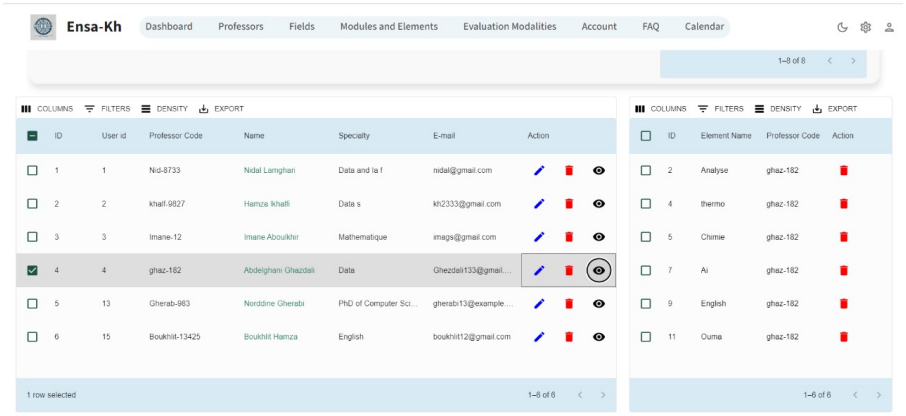

FIGURE 3.8 : Éléments affectés à un professeur sélectionné

3.4.3 Gestion des Comptes utilisateurs

L'administrateur peut gérer les informations des professuers, tels que leurs noms, e-mails ,spécialité et . Il est également possible de créer, modifier et supprimer des comptes utilisateurs.

- Ajout de nouveaux utilisateurs avec des informations de connexion.
- Modification des informations existantes des utilisateurs.
- Suppression des utilisateurs inactifs ou non nécessaires.

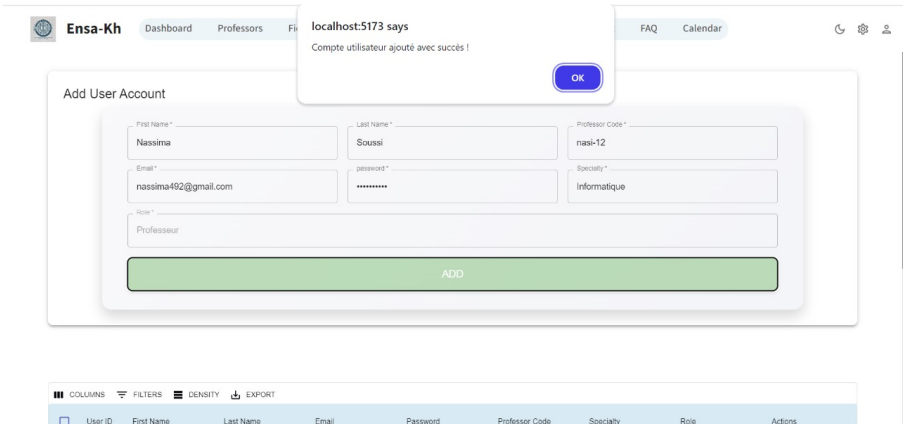

FIGURE 3.9 : Ajourt d'un compte utilisateur

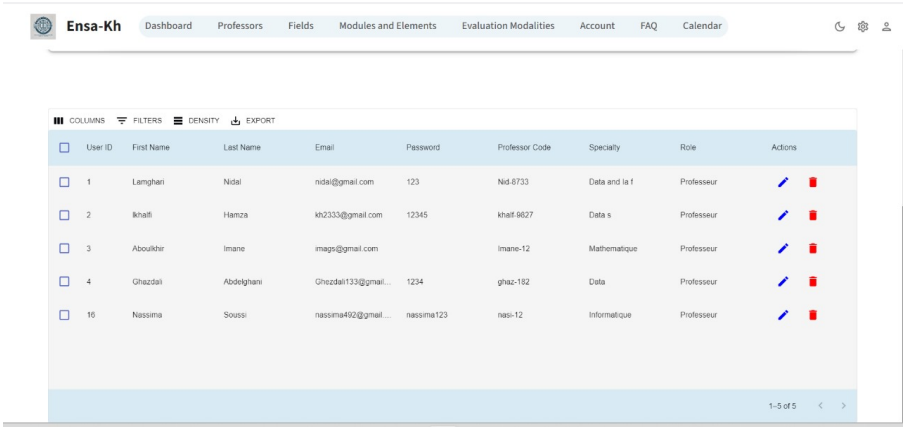

FIGURE 3.10 : Affichage des comptes utilisateurs

3.4.4 Gestion des Filières

Ce système permet à un administrateur de gérer les filières d'un établissement d'enseignement, en lui offrant la possibilité d'ajouter, de supprimer et de modifier des filières. Il peut également voir la liste des modules disponibles dans chaque filière par semestre et consulter la liste des étudiants inscrits dans chaque filière, filtrée par niveau.

3.4.4.1 Ajout ,Affichage et modification des filieres

L'administrateur peut créer, mettre à jour et supprimer des filières. Les filières sont définies par un nom et une description. Lorsqu'une filière est ajoutée ou modifiée, elle est envoyée au serveur via une requête API qui gère les données de manière persistante.

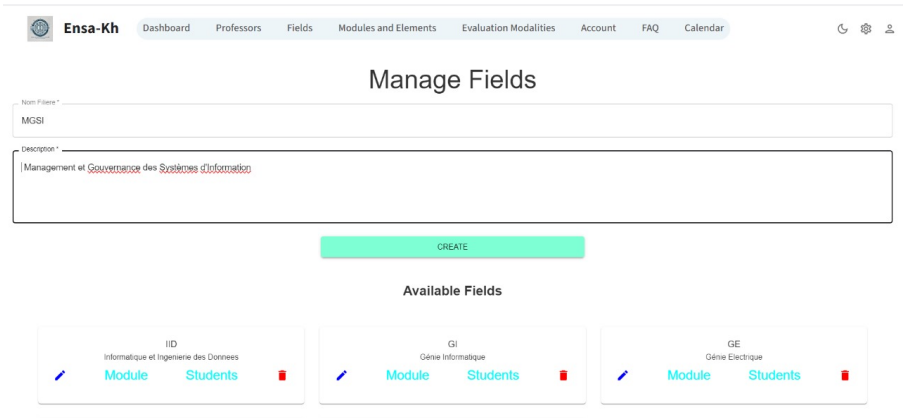

FIGURE 3.11 : Écran d'ajout et modification des filieres

3.4.4.2 Consulter les Filières

Le système permet également aux administrateurs de consulter la liste des filières existantes. Cette fonctionnalité fournit un aperçu rapide des filières disponibles, avec des options pour afficher les détails ou modifier chaque filière.

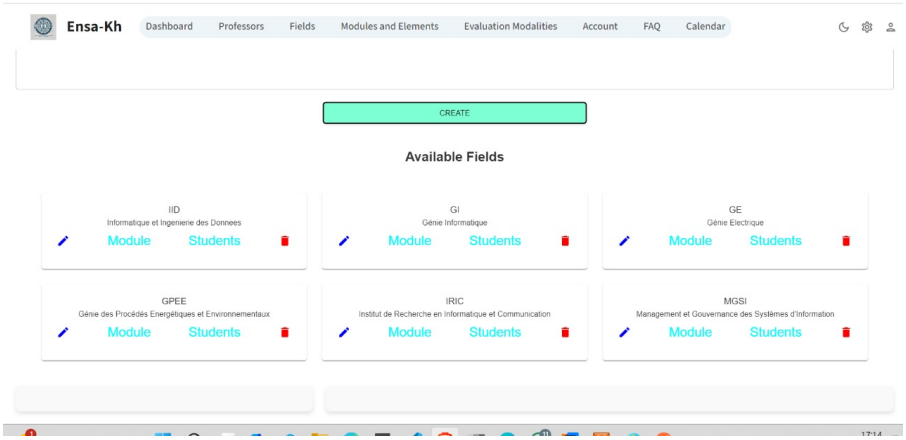

FIGURE 3.12 : Affichage des filieres

3.4.4.3 Gestion des Semestres et Modules

Pour chaque filière, l'administrateur peut afficher les semestres associés. Lorsqu'un semestre est sélectionné, le système récupère et affiche les modules disponibles pour ce semestre spécifique. Les modules sont affichés dans un tableau avec leur nom et code.

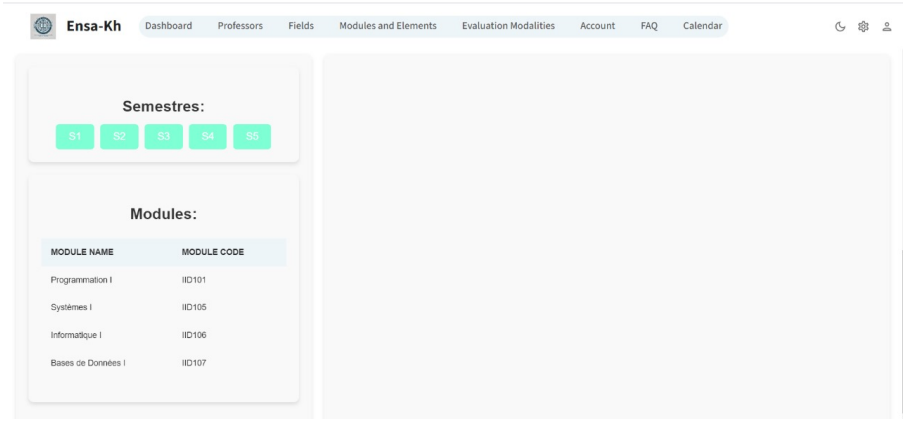

FIGURE 3.13 : Affichage des modules pour chaque semestre selectioner dans une filiere specifique

3.4.4.4 Affichage des Étudiants

Une fois une filière sélectionnée, l'administrateur peut choisir un niveau d'étudiants (par exemple, 1ère année, 2ème année, etc.). En fonction du niveau sélectionné, une liste d'étudiants inscrits dans la filière et à ce niveau est affichée. Les informations affichées pour chaque étudiant incluent son identifiant, son nom, son prénom, son numéro CNE (Code National Étudiant), et son niveau d'études.

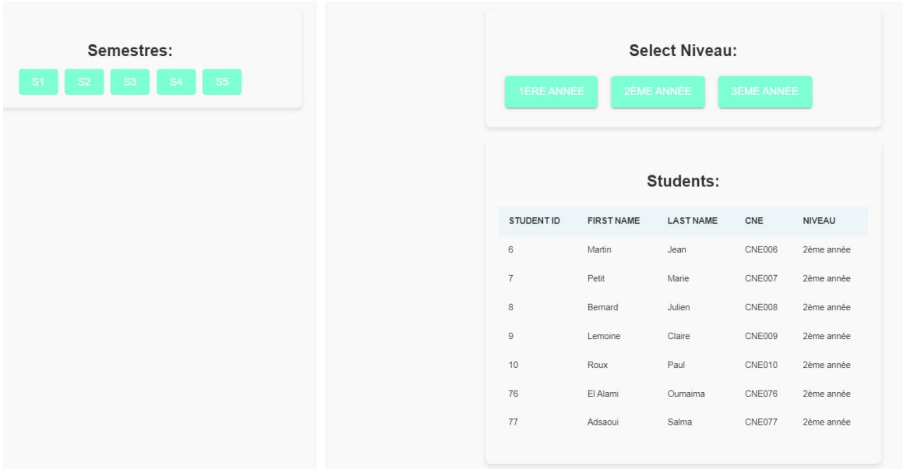

FIGURE 3.14 : Affichage des etudiants selon leurs niveau dans une filiere

3.4.5 Gestion des Modalités d'Évaluation

Cette section présente la gestion des modalités d'évaluation dans une application web. L'application permet à l'utilisateur de gérer des modalités, en ajoutant, modifiant ou supprimant des informations liées aux modalités d'évaluation, telles que leur nom et coefficient.

3.4.5.1 Ajout et modification d'une modalités d'évaluation

L'application permet à l'utilisateur de gérer les modalités d'évaluation de manière intuitive et dynamique. L'ajout de nouvelles modalités se fait à travers un formulaire où l'utilisateur peut renseigner le nom et le coefficient de la modalité. Pour modifier une modalité existante, l'utilisateur sélectionne celle-ci, et ses données sont automatiquement chargées dans le formulaire pour une mise à jour facile. Enfin, une icône de suppression permet de retirer définitivement une modalité de la liste et du serveur.

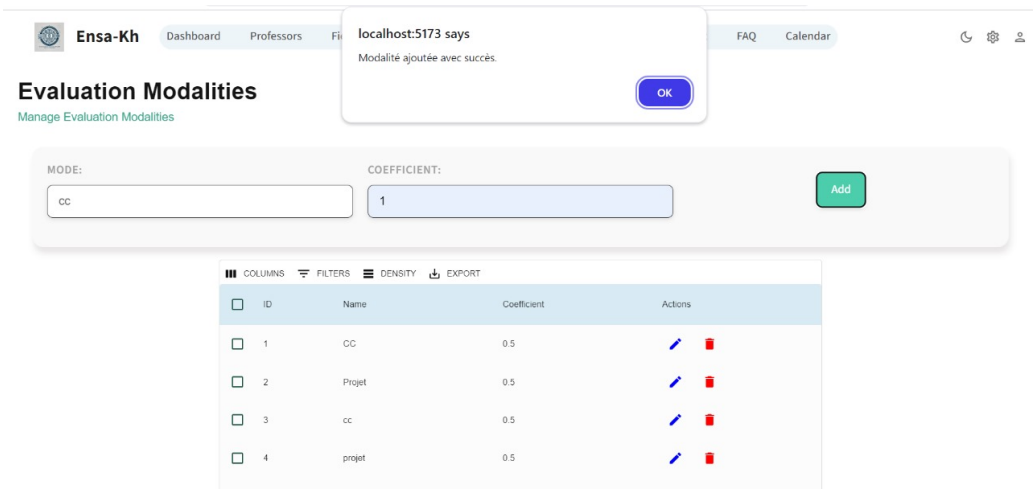

FIGURE 3.15 : Exemple d'ajout d'une modalités.

3.4.5.2 Affichage des Modalités

Les modalités sont affichées dans une grille, permettant une consultation rapide des informations de chaque modalité.

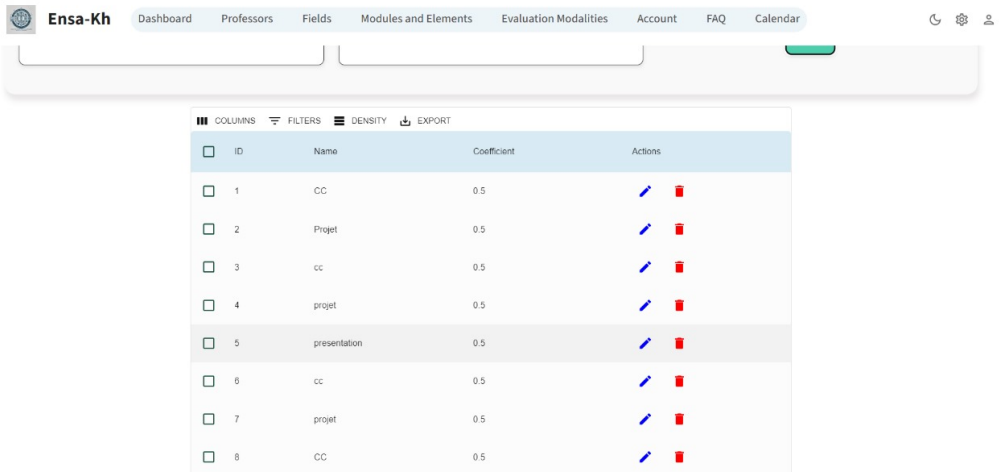

FIGURE 3.16 : La liste des modalités utiliser dans l'application.

3.4.6 Gestion des Modules et leurs Elements

Cette interface permet de gérer efficacement les modules et leurs éléments associés. Elle offre des fonctionnalités d'ajout, de modification, de suppression et de recherche pour maintenir une organisation claire et à jour. Chaque module est présenté avec ses détails essentiels, et les éléments associés sont hiérarchisés pour une navigation fluide. Cette gestion centralisée simplifie le suivi et l'administration des données.

3.4.6.1 Ajout d'un module

Cette interface permet d'ajouter et de gérer des modules ainsi que leurs éléments associés de manière intuitive et précise. L'utilisateur peut saisir les informations principales du module, telles que le code unique, le nom, le semestre et la filière. En complément, il est possible d'ajouter les éléments du module en listant leurs noms, ainsi que les coefficients correspondants, tous séparés par des virgules.

Une gestion rigoureuse des cas d'exception a été intégrée pour garantir la validité des données saisies :

- La somme des coefficients des éléments doit obligatoirement être égale à 1.
- Le nombre d'éléments saisis doit correspondre au nombre de coefficients renseignés.
- Les coefficients doivent être des nombres valides.

En cas d'erreur, des messages d'alerte dynamiques sont affichés pour informer l'utilisateur et l'aider à corriger les saisies. Cette interface assure ainsi une saisie fiable et efficace, tout en offrant une expérience utilisateur fluide.

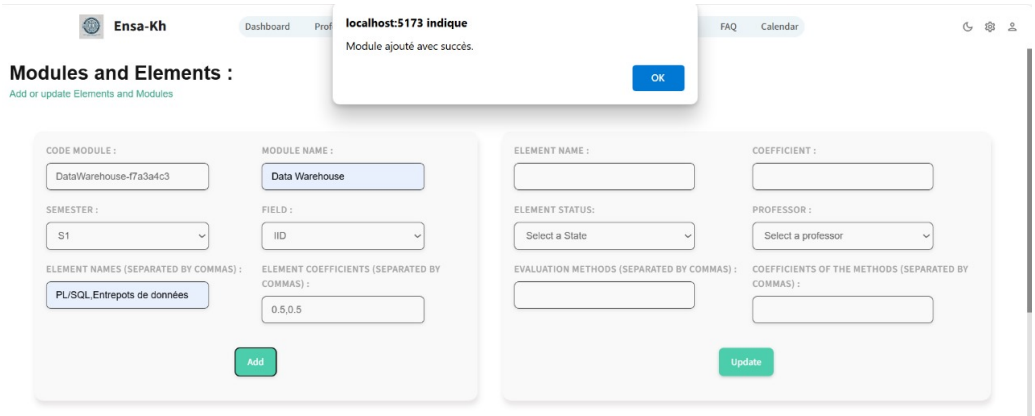

FIGURE 3.17 : Ajout d'un module.

3.4.6.2 Modification d'un module

Cette interface est conçue pour permettre la modification des informations liées à un module de manière claire et structurée. Elle offre des champs dédiés pour saisir ou mettre à jour le code unique du module, son nom, le semestre auquel il appartient, ainsi que la filière correspondante. En complément, l'interface permet d'ajouter ou de modifier les éléments associés au module, en indiquant leurs noms et leurs coefficients respectifs. Ces coefficients sont saisis sous forme de valeurs numériques, séparées par des virgules. Cette organisation garantit une gestion intuitive et précise des données

tout en facilitant leur mise à jour.

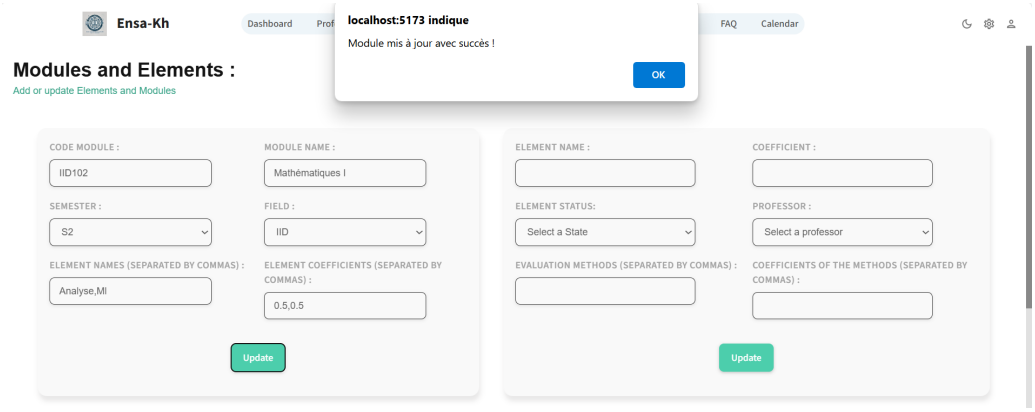

FIGURE 3.18 : Modification d'un module.

3.4.6.3 Modification d'un element

L'interface de modification d'un élément de module permet une gestion complète et intuitive des informations clés telles que le nom de l'élément, son coefficient, le nom du professeur responsable, l'état de l'élément, ainsi que les modalités d'évaluation et leurs coefficients associés. Chaque champ est soigneusement conçu pour garantir une saisie précise et cohérente des données. Une validation en temps réel vérifie que la somme des coefficients des modalités est égale à 1, assurant ainsi une pondération correcte. Des messages d'erreur clairs et détaillés s'affichent en cas d'incompatibilité entre le nombre de modalités et les coefficients fournis, ou si un coefficient saisi n'est pas un nombre valide. Cette interface garantit ainsi une gestion rigoureuse et fiable des informations tout en offrant une expérience utilisateur fluide et sécurisée.

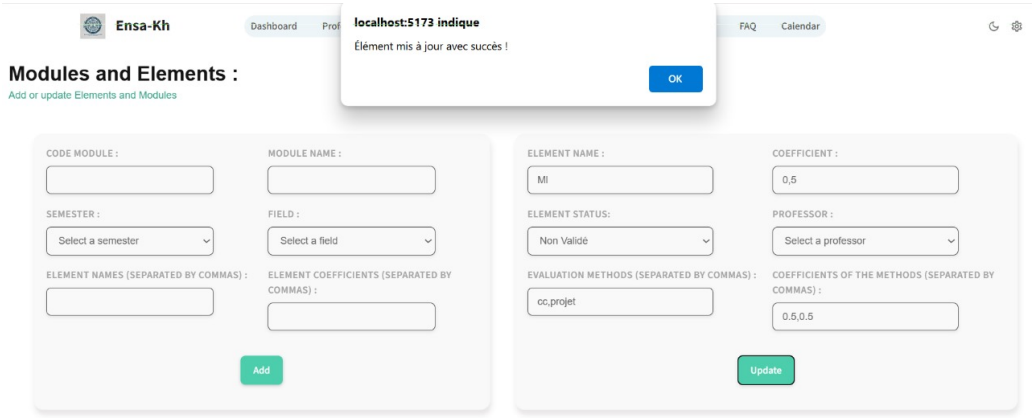

FIGURE 3.19 : Modification d'un element.

3.4.6.4 Affichage des modules ,leur elements et les modalités d'evaluation de chaque element

Cette interface permet la gestion des modules et leurs détails sous forme de tableaux interactifs :

[label=–]**Modules** : Affiche les modules avec code, nom, semestre et programme. Des actions permettent de modifier, supprimer ou visualiser les éléments associés. **Éléments** : Présente les

détails des éléments (nom, coefficient, professeur, état) avec options de modification, suppression et visualisation des modalités d'évaluation. **Modalités d'évaluation** : Liste les modalités avec leurs coefficients, et offre la possibilité de les supprimer.

Les tableaux comportent des options avancées telles que le filtrage selon toutes les colonnes, le téléchargement des données sous forme de fichiers ODF ou CSV, ainsi que l'adaptation de la densité d'affichage pour une gestion fluide et cohérente des informations.

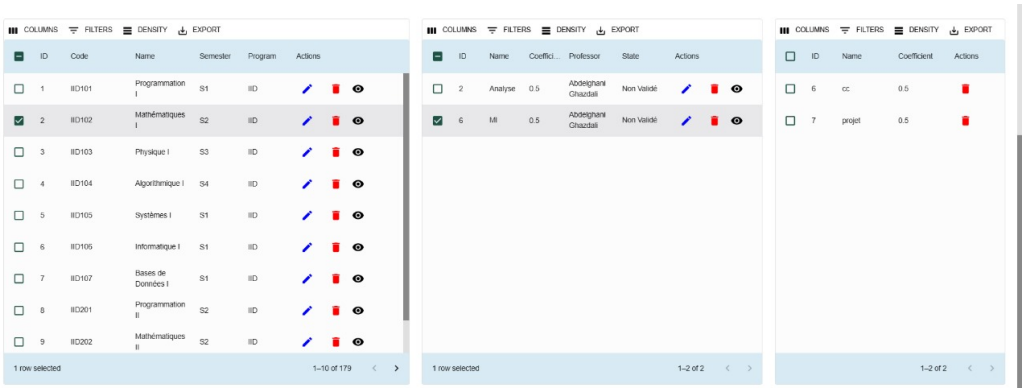

FIGURE 3.20 : La liste des modules,des éléments et des modalités d'évaluation.

3.4.7 FAQ : Consultation des Réponses aux Questions Fréquentes

L'interface FAQ (Frequently Asked Questions) de l'application est conçue pour fournir à l'administrateur une vue d'ensemble des questions fréquemment posées, ainsi que leurs réponses détaillées. Cette interface facilite l'accès rapide aux informations et explications liées aux fonctionnalités principales du système.

L'interface est structurée sous la forme d'un composant interactif de type accordéon, où chaque question peut être déployée pour afficher les détails associés. Les questions abordées couvrent divers aspects, tels que la gestion des professeurs, des filières, des éléments de modules, et des notes. Voici un exemple des questions disponibles :

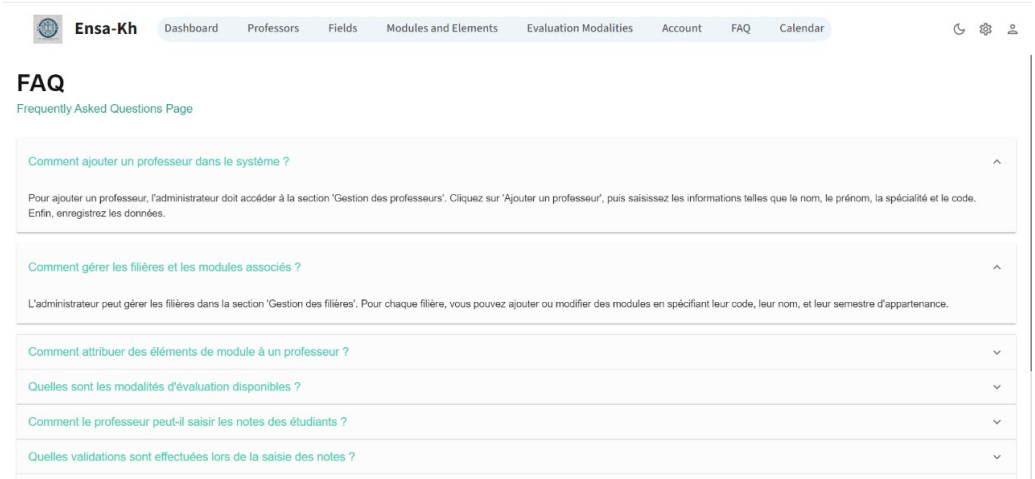

FIGURE 3.21 : Aperçu de l'interface FAQ dans l'application

3.4.8 Interface Calendrier : Gestion des Événements

L'interface calendrier de l'application offre une vue interactive et complète permettant à l'administrateur de gérer efficacement les événements. Grâce à l'intégration de la bibliothèque `FullCalendar`, cette interface assure une expérience utilisateur fluide et intuitive pour l'ajout, la modification et la suppression d'événements.

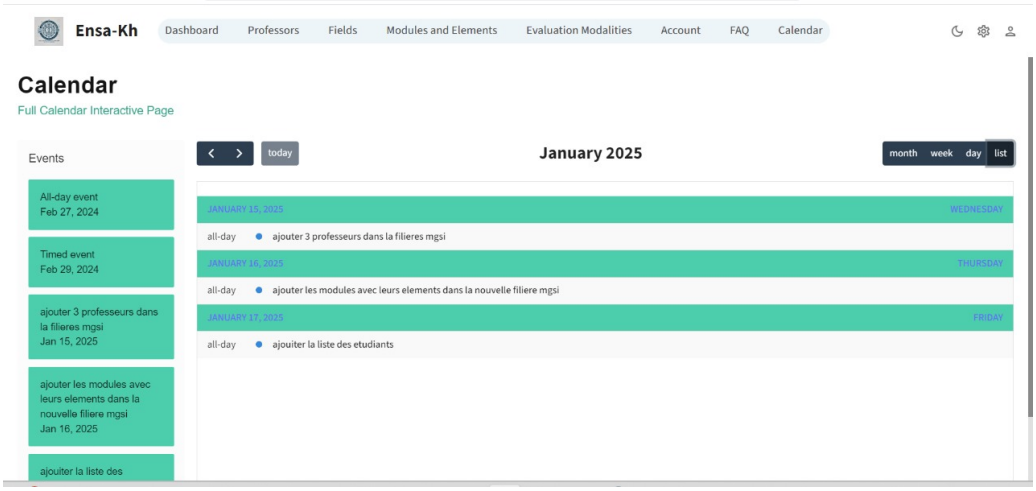

FIGURE 3.22 : Aperçu de l'interface calendrier dans l'application

Fonctionnalités principales :

- **Affichage interactif du calendrier :** L'interface permet de visualiser les événements sous différents formats, tels que vue mensuelle, hebdomadaire, journalière, ou liste.
- **Ajout et suppression d'événements :** Les utilisateurs peuvent ajouter de nouveaux événements en cliquant sur une date, saisir un titre, et enregistrer. Les événements peuvent être supprimés à l'aide d'une confirmation.
- **Synchronisation en temps réel :** Les événements ajoutés ou modifiés sont mis à jour instantanément, et leur affichage est optimisé pour différents appareils (responsive design).
- **Personnalisation du contenu :** Les couleurs et les éléments d'affichage s'adaptent au thème général de l'application.

3.5 Interface Professeur

L'interface destinée aux professeurs permet de gérer efficacement leurs cours, étudiants et ressources. La barre de navigation permet aux utilisateurs de naviguer rapidement entre différentes sections telles que la gestion des cours, la saisie et la validation des notes, ainsi que l'accès aux paramètres de l'utilisateur. Elle comprend également des icônes pour changer le mode d'affichage (sombre ou clair), ainsi qu'un menu déroulant affichant les informations personnelles du professeur (prénom, nom, email) et un bouton pour se déconnecter de l'application. L'aperçu des informations offre une vue d'ensemble rapide sur les cours et les étudiants. Il affiche le nombre de cours actifs et à venir, ainsi que les statistiques concernant les étudiants, comme le nombre total d'étudiants, le nombre d'étudiants actifs et diplômés.

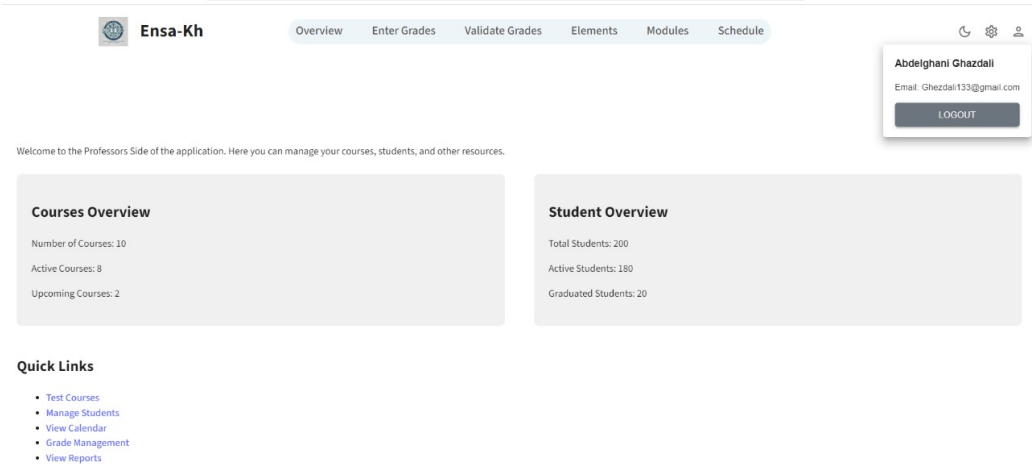

FIGURE 3.23 : Aperçu de l'interface Professeur dans l'application

3.5.1 Ajout des notes aux élèves

Le processus d'ajout des notes aux élèves commence par un simple clic sur l'option "Notes" dans l'interface dédiée aux professeurs. Une fois cette option sélectionnée, les semestres associés au professeur seront récupérés depuis la base de données. À l'étape suivante, lorsque le professeur choisit un semestre particulier, une liste des filières qu'il enseigne sera affichée. Après avoir sélectionné une filière, la liste des éléments (modules, matières, etc.) associés à ce professeur pour le semestre et la filière choisis sera également récupérée. Ce processus est divisé en trois étapes principales : la récupération des semestres, la récupération des filières et, enfin, la récupération des éléments.

Les images ci-dessous illustrent chacune de ces étapes dans le processus :

La première étape consiste à récupérer les semestres associés au professeur. Lorsque le professeur clique sur "Notes", le système interroge la base de données pour obtenir la liste des semestres affectés à ce professeur.

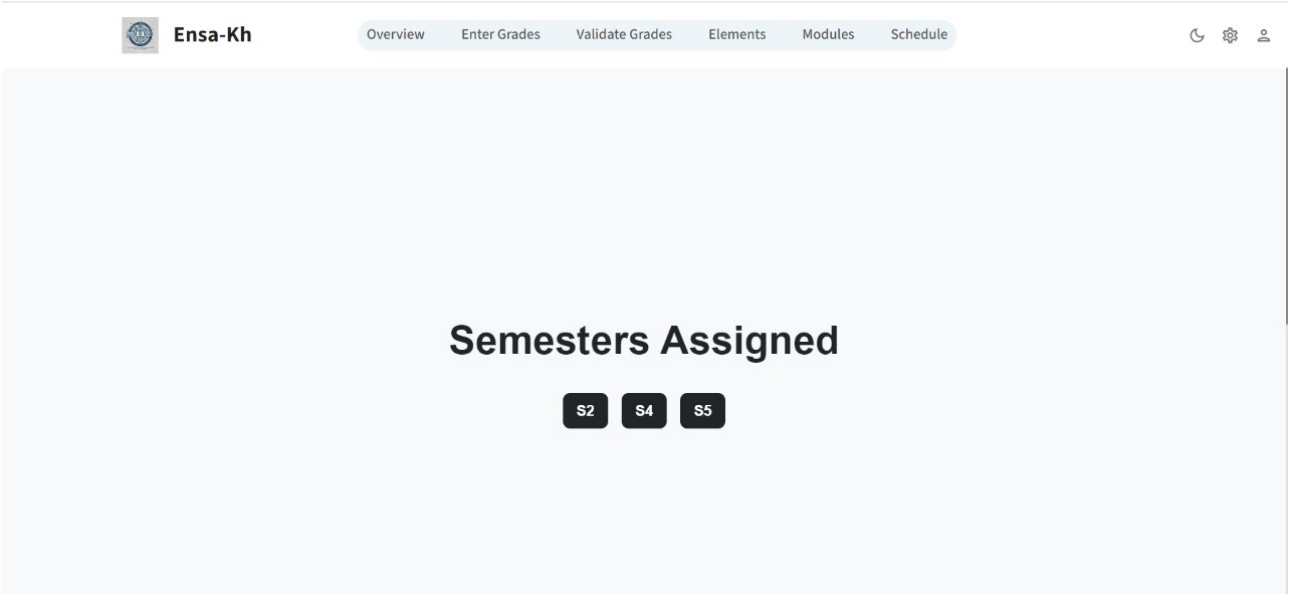

FIGURE 3.24 : Récupération des semestres associés au professeur

Après avoir choisi un semestre spécifique, la liste des filières dans lesquelles le professeur enseigne est récupérée et affichée. Cette étape permet au professeur de sélectionner la filière à laquelle il souhaite attribuer des notes.

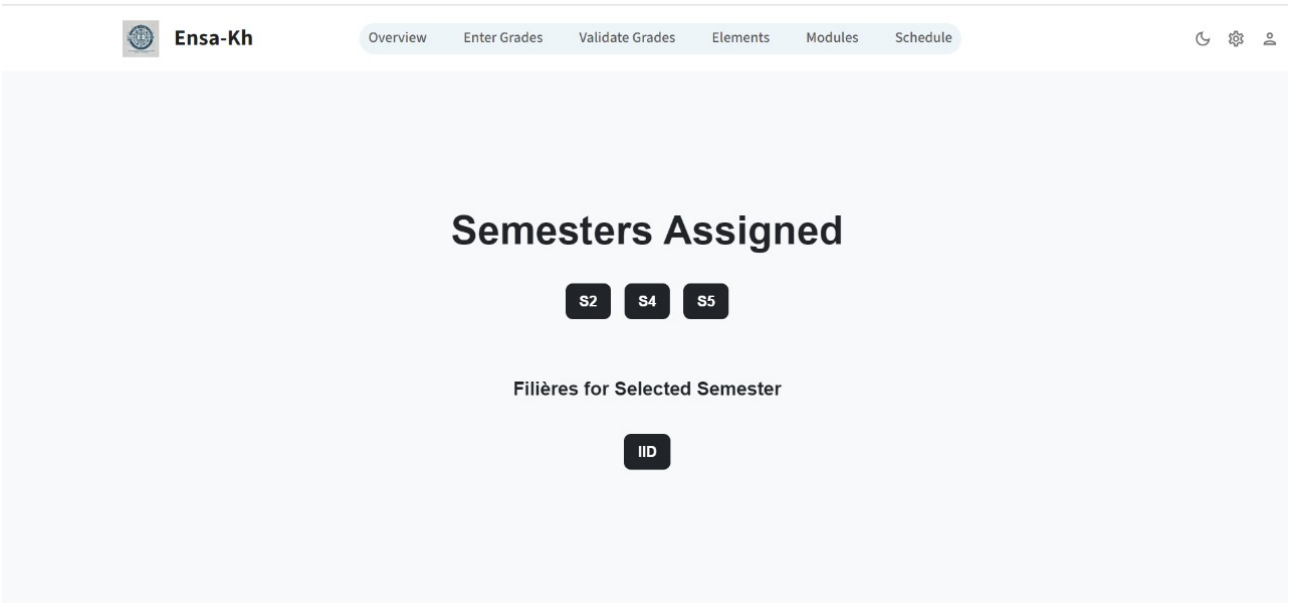

FIGURE 3.25 : Récupération des filières associées au semestre sélectionné

Enfin, une fois la filière sélectionnée, la liste des éléments associés à ce professeur pour cette filière et ce semestre est récupérée. Cela permet au professeur d'entrer les notes correspondantes pour chaque élément.

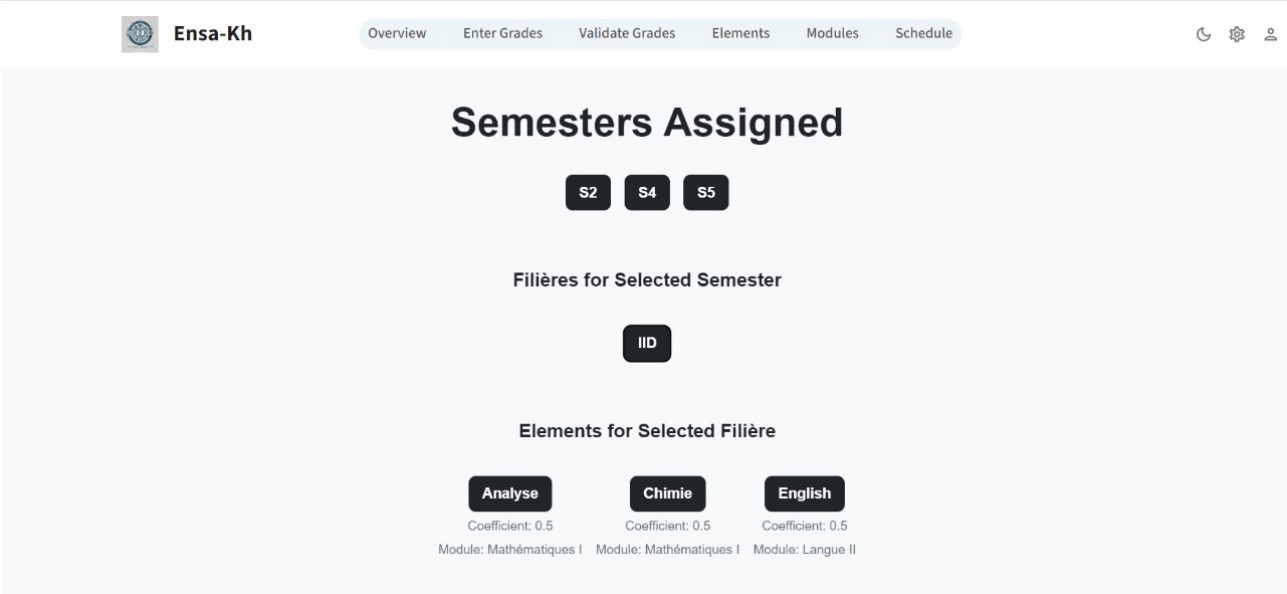

FIGURE 3.26 : Récupération des éléments associés à la filière et au semestre sélectionnés

Le système permet aux professeurs d'afficher les notes des étudiants pour un élément spécifique et de saisir les notes des étudiants dans un formulaire interactif.

Lorsqu'un professeur sélectionne un élément, les notes des étudiants associés à cet élément sont affichées dynamiquement dans une interface conviviale. La moyenne des notes est calculée et présentée en temps réel. L'image suivante montre comment les notes des étudiants sont affichées après la sélection de l'élément.

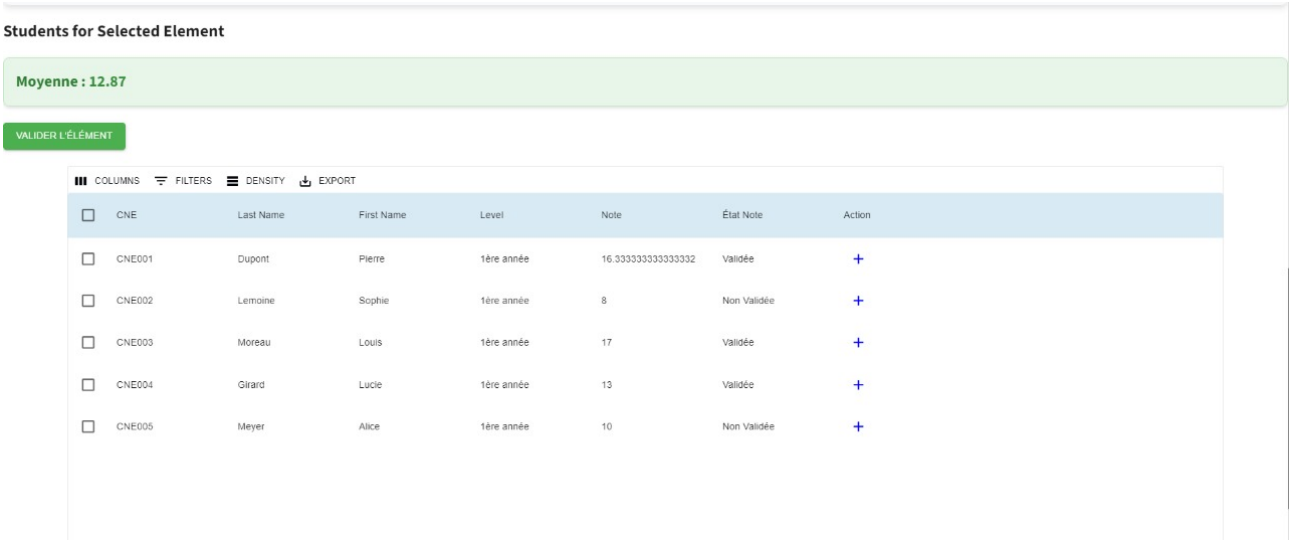

FIGURE 3.27 : Affichage des notes des étudiants pour un élément sélectionné.

Ensuite, lorsqu'un professeur clique sur le bouton `Ajouter` dans le tableau, une fenêtre modale s'ouvre. Cette fenêtre permet au professeur de saisir les notes des étudiants pour différentes modalités d'évaluation. Chaque modalité dispose d'un champ de saisie où le professeur peut entrer les notes. En cas d'absence d'un étudiant, une case à cocher permet de marquer cette absence.

L'image suivante illustre le formulaire modale permettant la saisie des notes pour un étudiant.

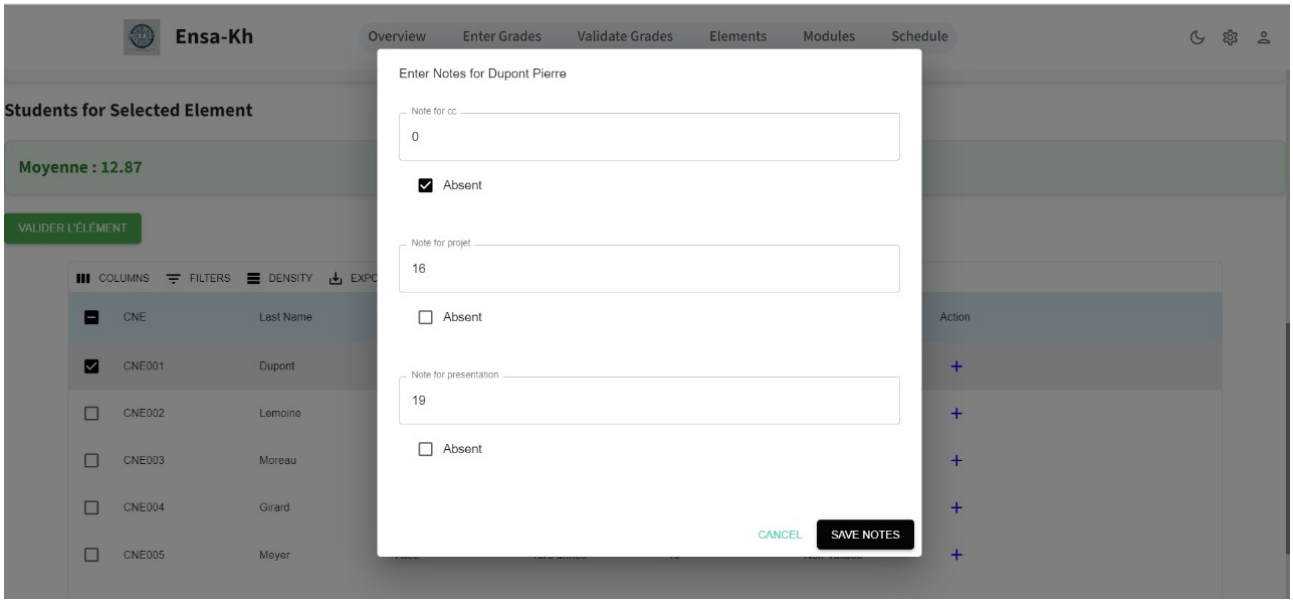

FIGURE 3.28 : Fenêtre modale pour la saisie des notes des étudiants.

Une fois que le professeur a saisi les notes, un message de confirmation apparaît. Ce message avertit le professeur s'il tente de saisir une note extrême, comme 0 ou 20, et lui demande de confirmer cette action.
L'image suivante montre la fenêtre de confirmation affichée avant de sauvegarder les notes.

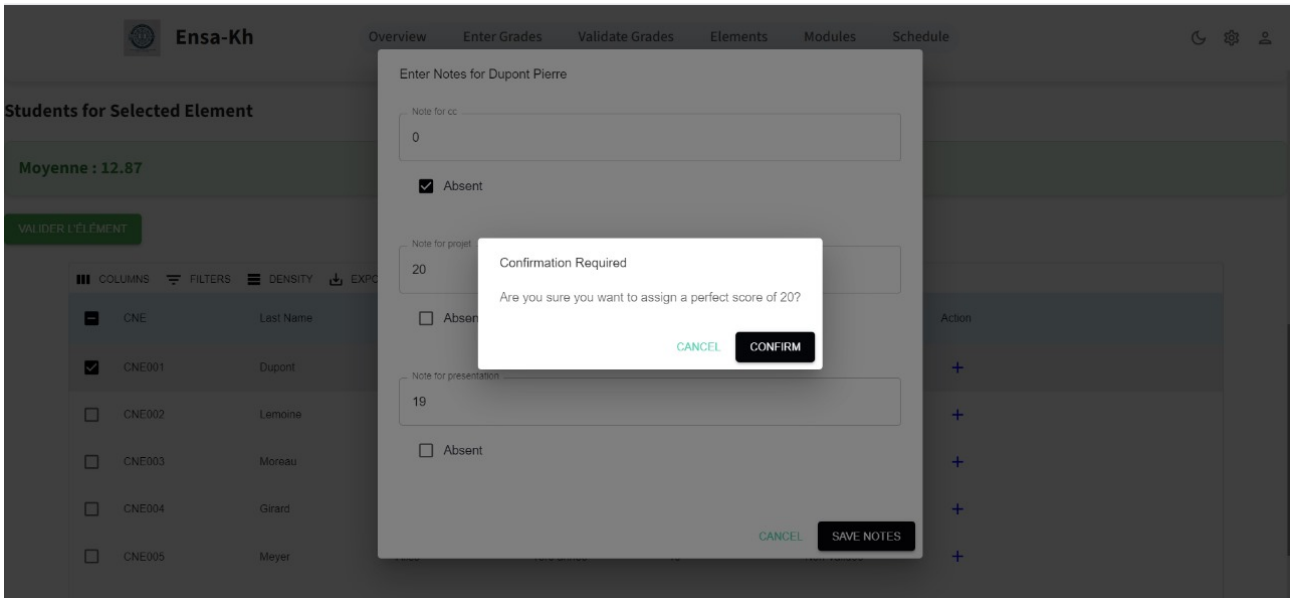

FIGURE 3.29 : Fenêtre de confirmation avant la saisie finale des notes.

Enfin, une fois que le professeur a confirmé la saisie des notes, le système passe à l'étape de calcul automatique. Après la validation de la note.

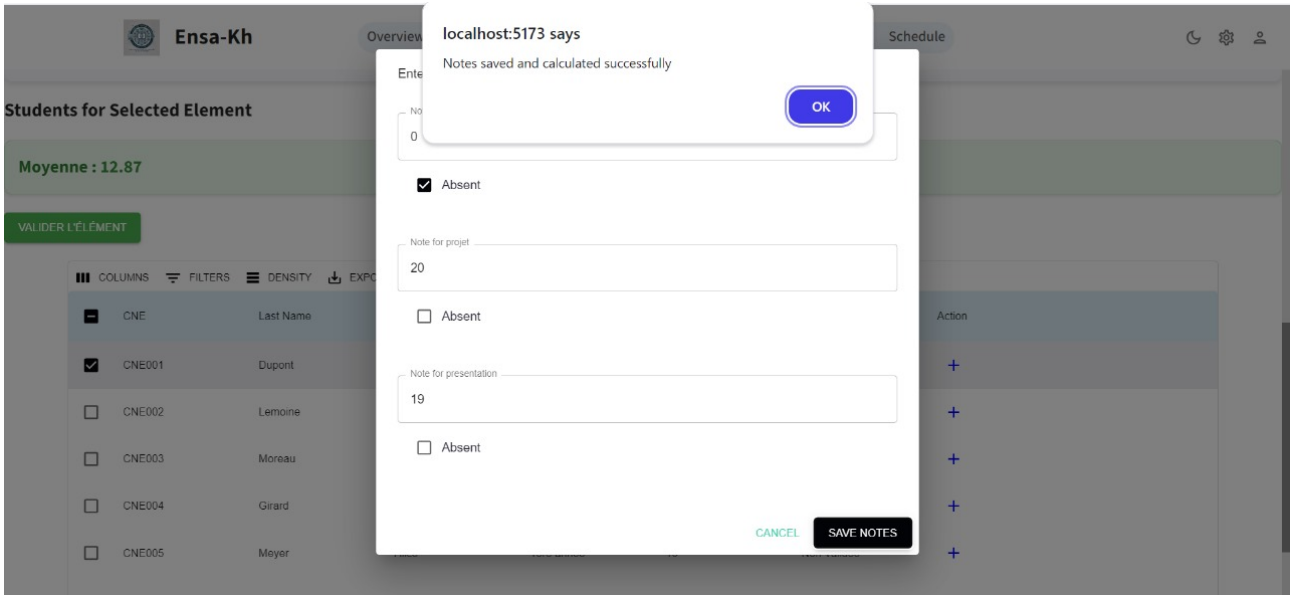

FIGURE 3.30 : sauvegarde saisie des notes.

Une fois les notes saisies et confirmées, le système procède à une mise à jour dynamique des résultats. Non seulement les notes des étudiants sont recalculées et affichées instantanément, mais la moyenne de chaque élément est également mise à jour automatiquement. Cette fonctionnalité assure que les données reflètent les modifications immédiatement après la validation des notes.

L'image ci-dessous illustre l'affichage de la moyenne de l'élément et des résultats des étudiants, qui sont mis à jour en temps réel, dès que les notes sont saisies et validées par le professeur.

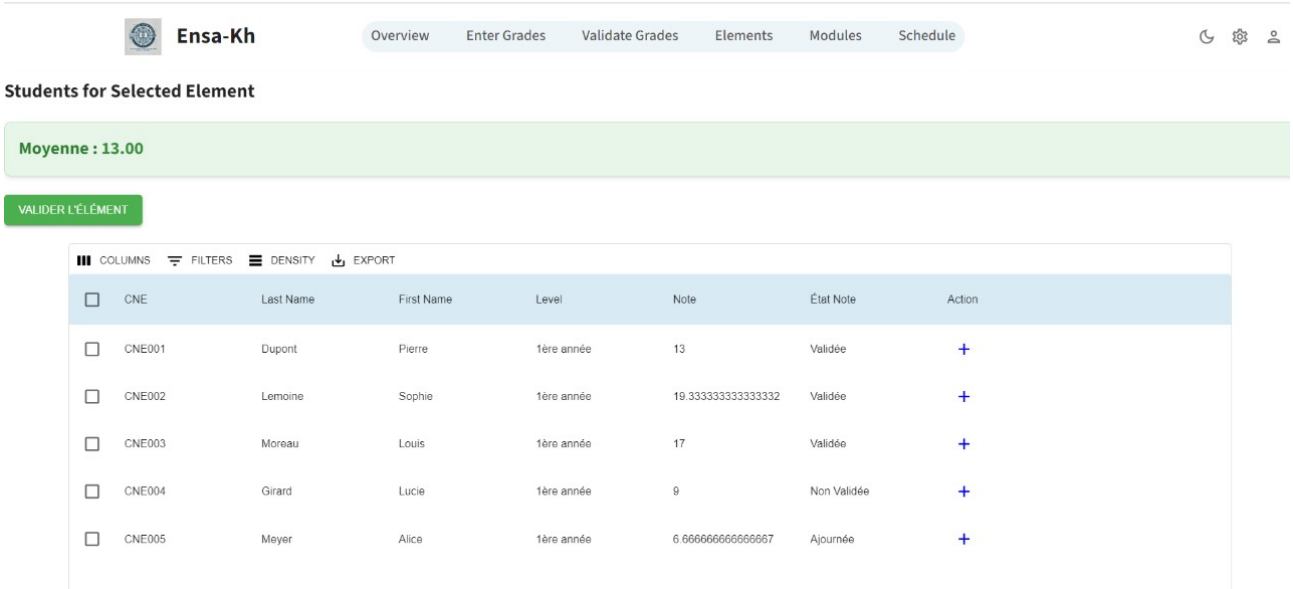

FIGURE 3.31 : Mise à jour dynamique des notes après saisie et sauvegarde.

3.5.2 Validation d'un element

Une fois les notes saisies et confirmées, un bouton de validation permet au professeur de finaliser la saisie des notes pour l'élément. Lorsque ce bouton est cliqué, l'élément est marqué comme validé, ce qui signifie que les notes ne peuvent plus être modifiées pour cet élément. Cela permet d'assurer que les résultats sont définitivement enregistrés et que les notes ne sont pas sujettes à des modifications ultérieures.

Une fois l'élément validé, le système met à jour dynamiquement les informations dans le tableau des résultats. La moyenne des étudiants pour cet élément est également recalculée et affichée immédiatement, comme illustré dans l'image ci-dessous.

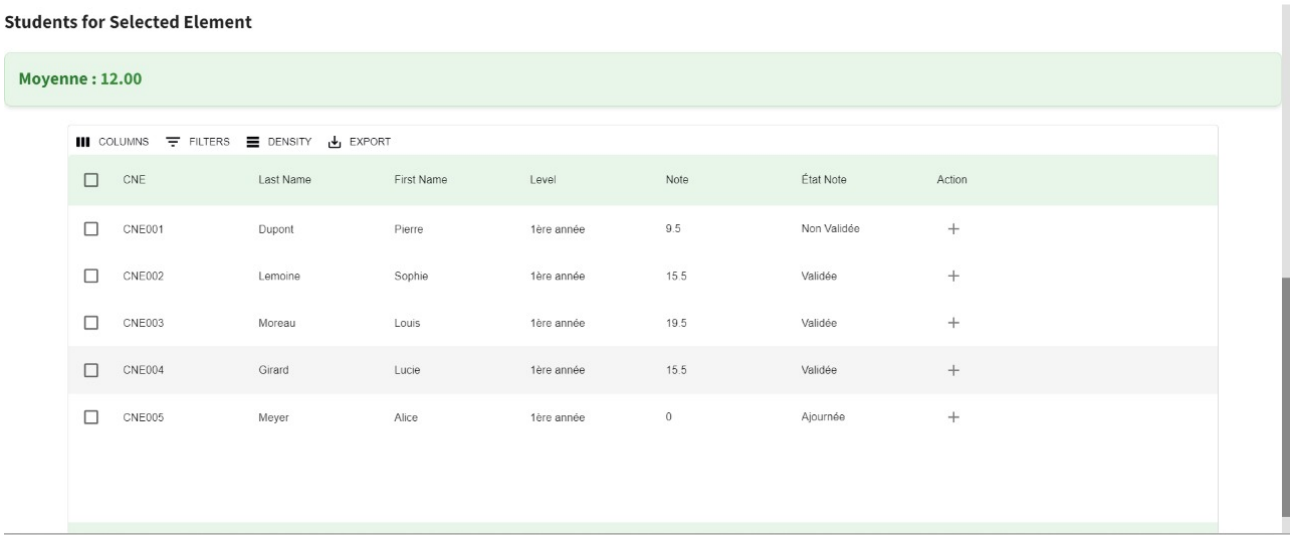

FIGURE 3.32 : Validation pour l'élément après saisie des notes.

3.5.3 Exportation des Notes d'un Élément Validé

Une fois que les notes d'un élément sont validées, il est possible d'exporter ces notes dans un fichier au format Excel ou PDF, selon les préférences du professeur. Cette fonctionnalité permet de conserver une copie des résultats dans un format accessible et facilement partageable.

Le processus d'exportation se déclenche une fois l'élément marqué comme validé. Le système propose alors deux options : exporter les notes dans un fichier Excel ou dans un fichier PDF. Les images ci-dessous montrent respectivement les options d'exportation pour chaque format.

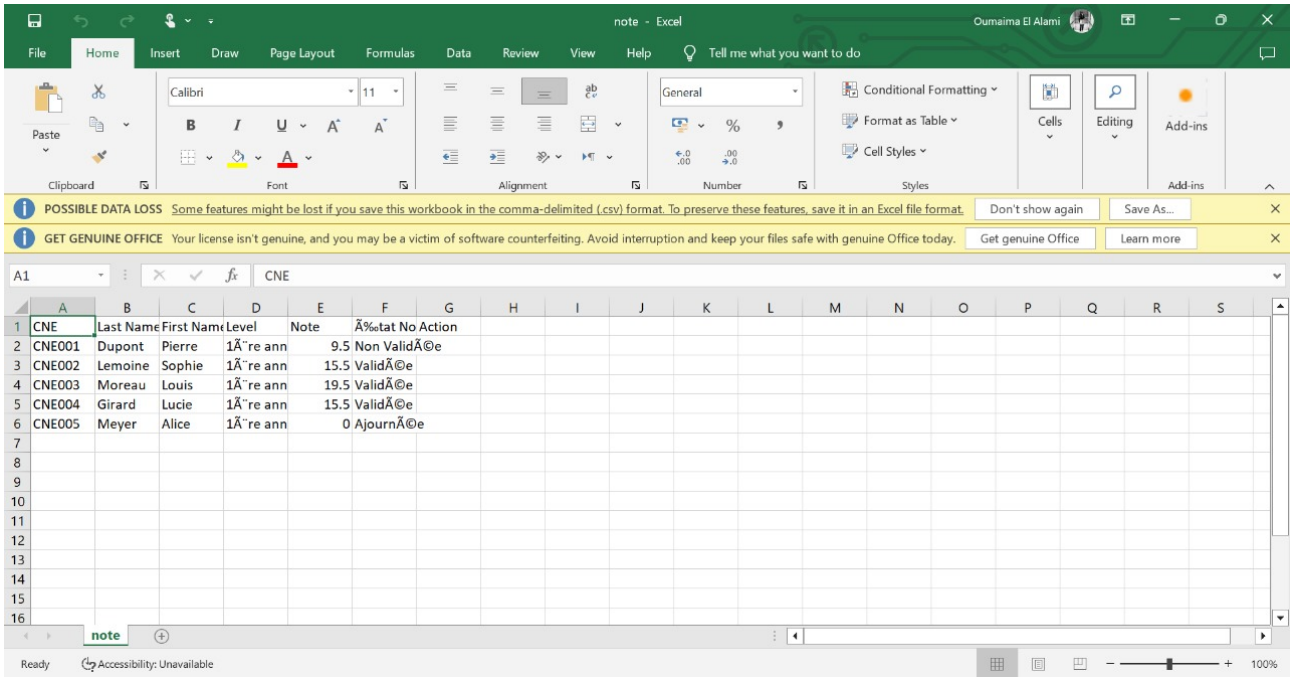

FIGURE 3.33 : Option d'exportation des notes dans un fichier Excel.

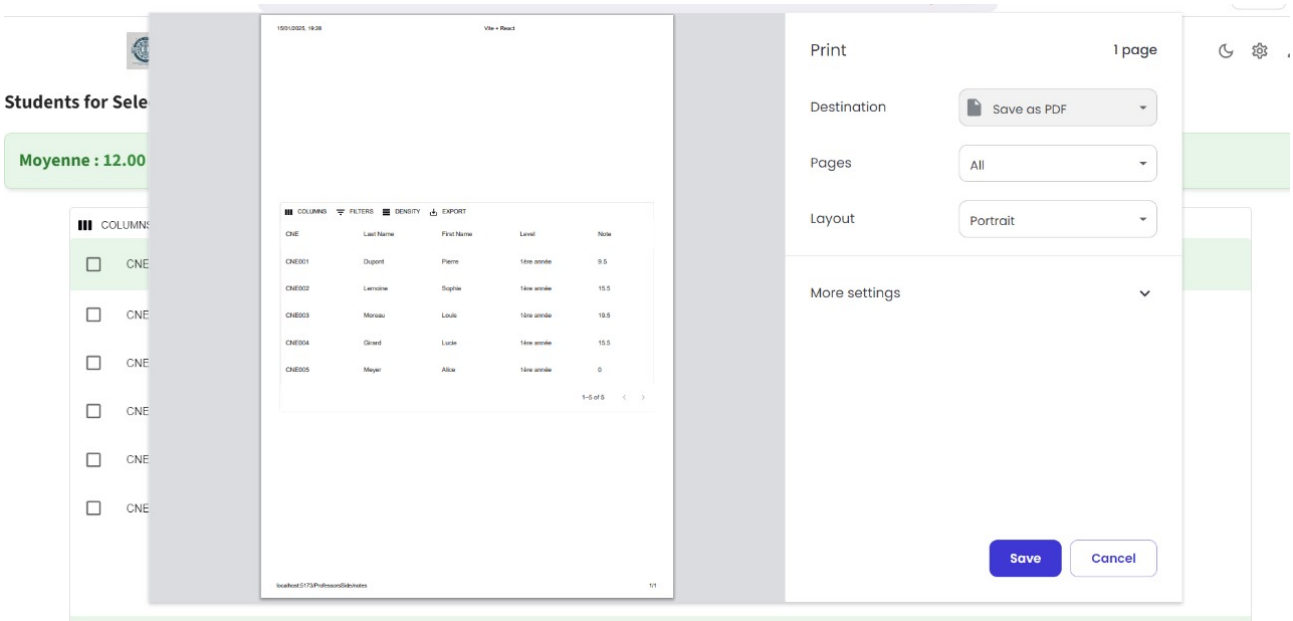

FIGURE 3.34 : Option d'exportation des notes dans un fichier PDF.

Ces options permettent de sauvegarder les notes dans un format pratique pour la gestion ou l'archivage des résultats, tout en assurant que les données restent intègres et facilement accessibles.

Conclusion

L'interface développée pour le système de gestion des notes permet une utilisation fluide et claire aussi bien pour les administrateurs que pour les professeurs. Pour les administrateurs, la gestion des professeurs, des modules et des éléments de module est centralisée, facilitant ainsi le suivi et l'organisation des activités académiques. De plus, la possibilité d'affecter des éléments à des professeurs et de gérer les comptes utilisateurs garantit une administration efficace et bien structurée.

Pour les professeurs, l'interface permet de saisir, modifier et valider les notes des étudiants de manière simple et intuitive. L'option de validation des notes assure un contrôle rigoureux des résultats avant leur enregistrement définitif, tout en garantissant la sécurité et la précision des données. La possibilité d'exporter les résultats dans des formats comme Excel ou PDF ajoute une dimension pratique à l'outil, facilitant le partage des résultats et leur archivage.

En somme, cette application offre un environnement numérique performant et sécurisé pour la gestion des évaluations, répondant ainsi aux besoins spécifiques des deux types d'utilisateurs, tout en garantissant une gestion des notes fluide, fiable et facilement exportable.

Chapitre 4

Difficultés et Solutions

Introduction

Ce chapitre met en lumière les principales difficultés techniques rencontrées lors du développement du système de gestion, ainsi que les solutions mises en œuvre pour les surmonter. Les défis abordés incluent la conception de la base de données et la validation des données. Ces aspects ont nécessité des ajustements constants et des optimisations techniques pour garantir la fiabilité, la sécurité et la performance du système.

4.1 Challenges techniques rencontrés

- **Conception de la base de données :**
  - Difficulté à modéliser correctement les relations entre les modules, les éléments de modules et les modalités d'évaluation.
- **Saisie et validation des notes :**
  - Validation stricte des données saisies (notes entre 0 et 20, différenciation entre absence et note zéro).
  - Gestion de la confirmation par le professeur en cas de notes extrêmes (0 ou 20).

4.2 Solutions mises en œuvre

- **Base de données relationnelle optimisée :**
  - Élaboration d'un modèle Entité/Association précis avec des contraintes d'intégrité rigoureuses pour éviter les incohérences.
  - Gestion centralisée des coefficients pour chaque modalité avec des contrôles stricts lors de l'ajout ou de la modification.

- **Validation des données :**
  - Implémentation de contrôles en temps réel pour garantir la validité des notes saisies.
  - Ajout de messages de confirmation pour les cas où des notes extrêmes (0 ou 20) sont présentes.

4.3 Retour d'expérience

- **Optimisation continue :** La gestion initiale des coefficients et modalités d'évaluation a nécessité plusieurs ajustements pour éviter les erreurs de saisie et garantir la cohérence des données.
- **Flexibilité du système :** Le choix de technologies modernes telles que `Spring Boot` pour le back-end et `React` pour le front-end a offert une grande flexibilité et facilité de maintenance.

Conclusion

En surmontant les divers défis techniques, le projet a permis de mettre en place un système robuste et performant, répondant aux besoins fonctionnels et utilisateurs. Les solutions apportées ont non seulement assuré une meilleure gestion des données et une expérience utilisateur fluide, mais elles ont également jeté les bases pour une évolutivité future. Ces apprentissages et améliorations continueront de guider les développements ultérieurs.

Conclusion Générale

Ce projet de gestion des notes pour une école a permis de mettre en œuvre une solution complète pour répondre aux besoins spécifiques des professeurs et des administrateurs dans la gestion des évaluations académiques. Grâce à une architecture solide basée sur le framework Spring Boot pour le backend et React pour le frontend, nous avons développé une application robuste et performante qui facilite la gestion des modules, des éléments, des notes, ainsi que des modalités d'évaluation.

L'application offre une interface intuitive et ergonomique, permettant à l'administrateur de gérer les professeurs, les filières et les évaluations, tandis que le professeur peut saisir, modifier et valider les notes des étudiants de manière fluide et efficace. La fonctionnalité de validation des données avant la saisie finale des notes garantit l'intégrité des informations, et les options d'exportation des résultats en PDF ou Excel offrent un niveau de flexibilité appréciable.

Ce projet a également permis de renforcer nos compétences techniques en Spring Boot, React, et dans la gestion des interactions entre le backend et le frontend. La gestion des transactions, des validations des données, ainsi que la conception et la mise en œuvre d'une interface utilisateur moderne ont été des défis stimulants et enrichissants.

Enfin, ce projet a non seulement contribué à l'acquisition de compétences techniques, mais aussi à une meilleure compréhension des processus métier et des besoins des utilisateurs finaux. Grâce à une approche agile et à une collaboration étroite entre les différentes parties prenantes, nous avons pu offrir une solution répondant aux attentes des utilisateurs tout en respectant les délais et les exigences du cahier des charges.