

Rapport du Projet

Filière : Informatique et Ingénierie des Données



Réalisé par :

Saad AFIFI
El Mehdi El Ouakyl
Selma Ezaakouni

Plateforme SmartClass

Encadré par :
Lamghari Nidal ENSA Khouribga

Année Académique : 2024/2025

Table des matières

1	Introduction	4
1.1	Contexte du projet	4
1.1.1	Présentation de l'ENSA Khouribga	4
1.1.2	Problématique	4
1.1.3	Motivations	4
1.2	Objectifs du projet	4
1.2.1	Objectif général	4
1.2.2	Objectifs spécifiques	5
1.3	Portée du projet	5
1.3.1	Périmètre fonctionnel	5
1.3.2	Limites du projet	5
1.4	Méthodologie de travail	5
1.4.1	Approche de développement	5
1.4.2	Outils de gestion de projet	5
2	Étude et Analyse des Besoins	6
2.1	Étude de l'existant	6
2.1.1	Analyse des plateformes existantes	6
2.1.2	Tableau comparatif	6
2.2	Analyse des besoins	6
2.2.1	Identification des acteurs	6
2.2.2	Besoins fonctionnels	7
2.2.3	Besoins non-fonctionnels	7
2.3	Cas d'utilisation	8
2.3.1	Diagramme de cas d'utilisation général	8
2.3.2	Description des cas d'utilisation principaux	8
2.4	Spécification des exigences	9
2.4.1	Matrice des exigences fonctionnelles	9
2.4.2	Priorisation MoSCoW	9
3	Conception du Système	10
3.1	Architecture globale du système	10
3.1.1	Architecture 3-tiers	10
3.1.2	Architecture détaillée	10
3.2	Modélisation UML	11
3.2.1	Diagramme de classes	11
3.2.2	Diagrammes de séquence	12
3.2.3	Diagramme états-transitions	13
3.2.4	Diagrammes de communication	14
3.2.5	Diagramme de déploiement	15

4 Technologies et Architecture	17
4.1 Stack technologique	17
4.1.1 Vue d'ensemble	17
4.1.2 Technologies Frontend	17
4.1.3 Technologies Backend	18
4.1.4 Base de données et stockage	18
4.2 Architecture Microservices	19
4.2.1 Services déployés	19
4.2.2 Communication inter-services	19
4.3 Intégrations externes	20
4.3.1 Intégration Zoom API (backup)	20
4.3.2 Service de notifications	21
4.4 Gestion de la configuration	22
4.4.1 Profils Spring	22
4.4.2 Configuration centralisée	22
5 Implémentation Frontend	23
5.1 Architecture Frontend	23
5.1.1 Vue d'ensemble de l'architecture	23
5.1.2 Structure des pages	23
5.2 Pages publiques	24
5.2.1 Landing Page	24
5.2.2 Page de Login	25
5.3 Espace Administrateur	25
5.3.1 Home Admin avec Dashboard	25
5.3.2 Gestion des Étudiants	26
5.3.3 Gestion des Classes	26
5.4 Espace Professeur	27
5.4.1 Home Professeur	27
5.4.2 Gestion des Annonces	27
5.4.3 Gestion des Réunions	28
5.4.4 Gestion des Exercices	28
5.4.5 Gestion des QCM	29
5.4.6 Gestion du Profil	29
5.5 Espace Étudiant	30
5.5.1 Home Étudiant	30
5.5.2 Page Exercices	30
5.5.3 Page Annonce	31
5.5.4 Page QCM	31
5.6 Aspects techniques	32
5.6.1 Navigation et Routage	32
5.6.2 Responsive Design	32
6 Implémentation Backend	33
6.0.1 Architecture Backend Spring Boot	33
6.0.2 Diagramme architecture globale	34
6.1 Configuration et Setup	35
6.2 Modèle de données et JPA	35
6.2.1 Entités JPA	35
6.2.2 Repository	35
6.3 Couche Service	35
6.3.1 Services métier	35
6.3.2 Logique métier	36
6.3.3 DTOs et Mapping	36
6.4 API REST Controllers	36
6.4.1 Design des endpoints	36
6.4.2 Controllers principaux	36
6.4.3 Gestion des réponses	36
6.5 Cloud Integration	37

6.6	Service d'email	37
6.7	Conclusion	37
7	Module de Détection de Plagiat	38
7.1	Détection automatique de plagiat	38
7.1.1	Objectif	38
7.1.2	Fonctionnement général	38
7.1.3	Description technique	39
7.1.4	Architecture du système	39
7.1.5	Avantages de l'approche	39
7.1.6	Intégration avec le backend Spring Boot	39
7.1.7	Problèmes rencontrés et solutions	40
8	Sécurité et Authentification	41
8.1	Architecture de sécurité	41
8.1.1	Vue d'ensemble	41
8.2	Authentification avec JWT	41
8.2.1	Flux d'authentification	41
8.2.2	Structure des tokens JWT	42
8.3	Configuration Spring Security	42
8.3.1	Architecture de filtrage	42
8.3.2	Gestion des rôles et permissions	43
9	Conclusion et Perspectives	44
9.1	Bilan du projet	44
9.1.1	Objectifs atteints	44
9.2	Défis rencontrés et solutions	44
9.2.1	Défis techniques	44

Chapitre 1

Introduction

1.1 Contexte du projet

1.1.1 Présentation de l'ENSA Khouribga

L'École Nationale des Sciences Appliquées de Khouribga (ENSA Khouribga) est un établissement d'enseignement supérieur public marocain, faisant partie du réseau des ENSA du Maroc. Elle forme des ingénieurs d'état dans diverses spécialités, dont l'Informatique et l'Ingénierie des Données.

1.1.2 Problématique

Dans le contexte actuel de transformation numérique de l'enseignement, l'ENSA Khouribga fait face à plusieurs défis :

- **Communication fragmentée** : Les échanges entre professeurs et étudiants se font via différents canaux (email, WhatsApp, etc.), rendant difficile le suivi des informations importantes.
- **Gestion des ressources pédagogiques** : Les supports de cours sont dispersés, sans plateforme centralisée pour leur organisation et leur partage.
- **Évaluation traditionnelle** : Les processus d'évaluation restent largement manuels, sans outils numériques adaptés pour les QCM ou la détection de plagiat.
- **Absence d'enseignement à distance** : Manque d'infrastructure pour les cours en ligne et le streaming en direct.

1.1.3 Motivations

Ce projet est motivé par la volonté de :

- Moderniser l'infrastructure pédagogique de l'ENSA Khouribga
- Améliorer l'expérience d'apprentissage des étudiants
- Faciliter le travail des enseignants
- S'aligner sur les standards internationaux de l'e-learning

1.2 Objectifs du projet

1.2.1 Objectif général

Développer une plateforme web complète et moderne nommée **SmartClass**, offrant un environnement numérique intégré pour la gestion de l'enseignement à l'ENSA Khouribga.

1.2.2 Objectifs spécifiques

1. **Centraliser la communication** : Créer un espace unique pour les annonces, notifications et échanges entre professeurs et étudiants.
2. **Gérer les ressources pédagogiques** : Implémenter un système de stockage et de partage organisé pour les cours, exercices et supports.
3. **Automatiser les évaluations** : Développer des modules pour la création et correction automatique de QCM.
4. **Déetecter le plagiat** : Intégrer un système intelligent de détection de plagiat utilisant des techniques de NLP.
5. **Permettre l'enseignement à distance** : Implémenter des fonctionnalités de live streaming et de visioconférence.
6. **Assurer la sécurité** : Mettre en place une authentification robuste et une gestion fine des permissions.

1.3 Portée du projet

1.3.1 Périmètre fonctionnel

La plateforme SmartClass couvrira les domaines suivants :

- Gestion des utilisateurs (administrateurs, professeurs, étudiants)
- Gestion des cours et des ressources pédagogiques
- Système d'annonces et de notifications
- Gestion des devoirs et exercices
- Évaluations en ligne (QCM)
- Détection automatique du plagiat
- Live streaming et enregistrement des cours
- Tableau de bord personnalisé pour chaque type d'utilisateur

1.3.2 Limites du projet

Le projet se concentrera sur :

- Une application web responsive (pas d'application mobile native dans cette phase)
- L'ENSA Khouribga uniquement (pas de multi-établissements)
- Les fonctionnalités essentielles (pas de gamification ou d'IA avancée)

1.4 Méthodologie de travail

1.4.1 Approche de développement

Nous avons adopté une approche **Agile** avec la méthodologie **Scrum** pour :

- Permettre une flexibilité dans le développement
- Obtenir des feedbacks réguliers
- Livrer des fonctionnalités de manière incrémentale

1.4.2 Outils de gestion de projet

- **Git/GitHub** : Gestion du code source et collaboration
- **Discord** : Communication d'équipe
- **Figma** : Conception des interfaces

Chapitre 2

Étude et Analyse des Besoins

2.1 Étude de l'existant

2.1.1 Analyse des plateformes existantes

Avant de concevoir SmartClass, nous avons analysé plusieurs plateformes e-learning existantes pour identifier leurs forces et faiblesses.

Moodle

Points forts :

- Open source et gratuit
- Très complet en termes de fonctionnalités
- Grande communauté de développeurs

Limitations :

- Interface utilisateur datée et peu intuitive
- Configuration complexe
- Performance limitée avec beaucoup d'utilisateurs

Google Classroom

Points forts :

- Interface moderne et intuitive
- Intégration parfaite avec la suite Google
- Gratuit pour les établissements éducatifs

Limitations :

- Dépendance totale à l'écosystème Google
- Personnalisation limitée
- Pas de détection de plagiat intégrée

2.1.2 Tableau comparatif

2.2 Analyse des besoins

2.2.1 Identification des acteurs

Le système SmartClass implique trois types d'acteurs principaux :

1. **Administrateur** : Responsable de la gestion globale de la plateforme
2. **Professeur** : Enseignant qui crée et gère les cours
3. **Étudiant** : Apprenant qui consulte les cours et soumet les devoirs

Fonctionnalité	Moodle	Google Classroom	Blackboard	SmartClass
Interface moderne	✗	✓	±	✓
Open Source	✓	✗	✗	✓
Détection plagiat	±	✗	✓	✓
Live Streaming	±	✓	✓	✓
Personnalisation	✓	✗	±	✓
Performance	±	✓	✓	✓

Table 2.1 – Comparaison des plateformes e-learning

2.2.2 Besoins fonctionnels

Pour l'Administrateur

- Gérer les comptes utilisateurs (CRUD)
- Gérer les filières et les classes
- Superviser l'utilisation de la plateforme
- Configurer les paramètres système

Pour le Professeur

- Créer et gérer des cours
- Publier des annonces
- Uploader des ressources pédagogiques
- Créer des exercices et des QCM
- Corriger et noter les devoirs
- Diffuser des cours en direct

Pour l'Étudiant

- Consulter les cours et ressources
- Lire les annonces
- Télécharger les documents
- Soumettre les devoirs
- Passer les QCM en ligne
- Assister aux cours en direct
- Consulter ses notes et feedback

2.2.3 Besoins non-fonctionnels

Performance

- Temps de réponse < 2 secondes pour les opérations courantes
- Support de 1000+ utilisateurs simultanés
- Upload de fichiers jusqu'à 100 MB

Sécurité

- Authentification forte avec JWT
- Chiffrement des données sensibles

Ergonomie

- Interface responsive (desktop, tablette, mobile)
- Navigation intuitive

2.3 Cas d'utilisation

2.3.1 Diagramme de cas d'utilisation général

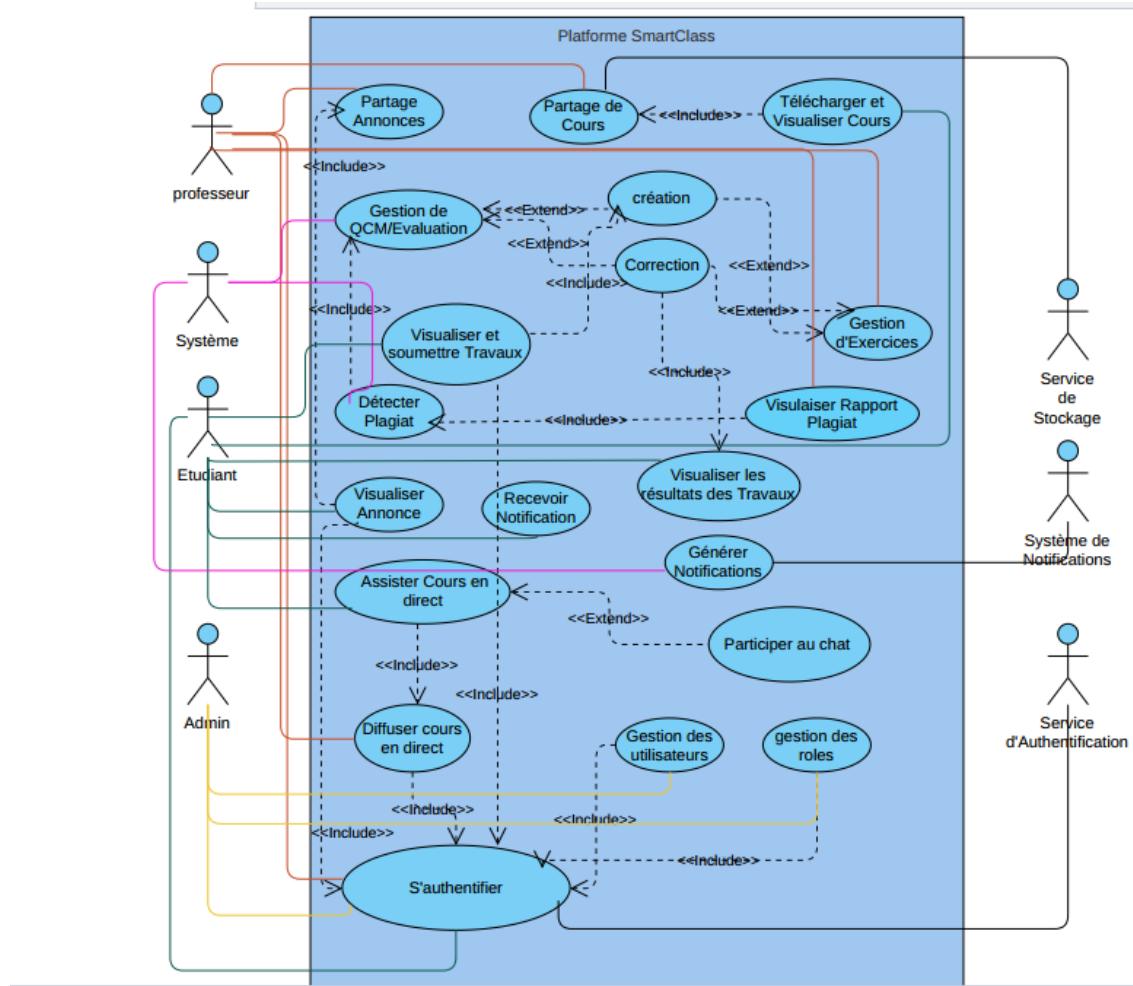


Figure 2.1 – Diagramme de cas d'utilisation général de SmartClass

2.3.2 Description des cas d'utilisation principaux

UC01 : S'authentifier

- **Acteurs :** Tous les utilisateurs
- **Précondition :** Avoir un compte actif
- **Scénario nominal :**
 1. L'utilisateur accède à la page de connexion
 2. Il saisit son email et mot de passe
 3. Le système vérifie les credentials
 4. Le système génère un token JWT
 5. L'utilisateur est redirigé vers son dashboard
- **Scénarios alternatifs :**
 - Credentials invalides : afficher message d'erreur
 - Compte bloqué : informer l'utilisateur

UC03 : Soumettre un devoir

- **Acteur :** Étudiant
- **Précondition :** Être inscrit au cours
- **Scénario nominal :**
 1. L'étudiant accède au devoir
 2. Il télécharge le sujet
 3. Il upload sa solution
 4. Le système vérifie le format et la taille
 5. Le système lance la détection de plagiat
 6. La soumission est confirmée

2.4 Spécification des exigences

2.4.1 Matrice des exigences fonctionnelles

ID	Exigence	Priorité	Module
EF-01	Authentification multi-rôles	Haute	Auth
EF-02	Gestion des cours CRUD	Haute	Cours
EF-03	Upload/Download de fichiers	Haute	Storage
EF-04	Système d'annonces	Moyenne	Communication
EF-05	Création de QCM	Haute	Evaluation
EF-06	Détection de plagiat	Haute	Plagiat
EF-07	Live streaming	Moyenne	Streaming
EF-08	Notifications email	Moyenne	Notification
EF-09	Dashboard analytics	Basse	Analytics
EF-10	Export PDF des notes	Basse	Export

Table 2.2 – Matrice des exigences fonctionnelles

2.4.2 Priorisation MoSCoW

Must have (Doit avoir) :

- Authentification et gestion des utilisateurs
- Gestion des cours et ressources
- Soumission et correction des devoirs
- Détection de plagiat

Should have (Devrait avoir) :

- QCM en ligne
- Live streaming
- Notifications

Could have (Pourrait avoir) :

- Application mobile
- Intégration calendrier
- Forum de discussion

Chapitre 3

Conception du Système

3.1 Architecture globale du système

3.1.1 Architecture 3-tiers

SmartClass adopte une architecture 3-tiers moderne et scalable :

1. **Couche Présentation** : Interface React.js unique avec routage basé sur les rôles
2. **Couche Logique Métier** : Backend Spring Boot et service Django pour le plagiat
3. **Couche Données** : Base de données MySQL

3.1.2 Architecture détaillée

Le système est conçu avec une approche microservices pour assurer la scalabilité et la maintenabilité :

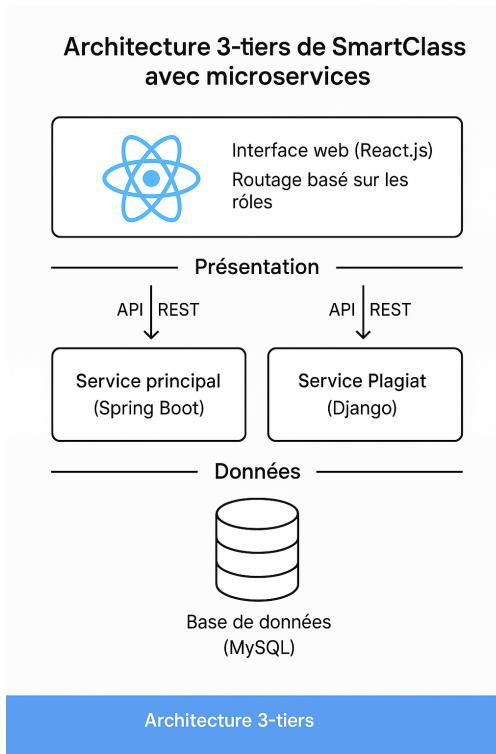


Figure 3.1 – Architecture globale de SmartClass

3.2 Modélisation UML

3.2.1 Diagramme de classes

Le diagramme de classes représente la structure statique du système :

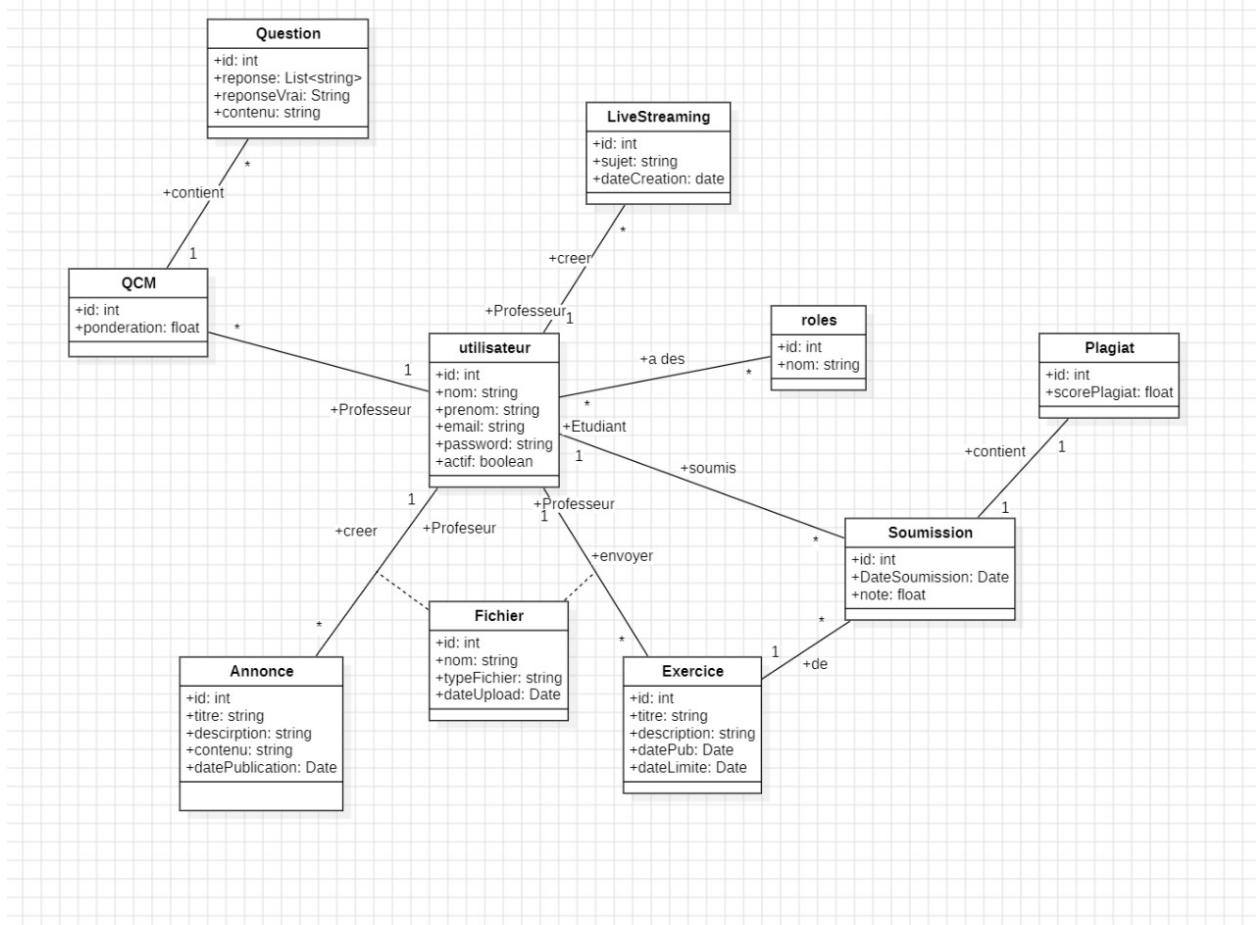


Figure 3.2 – Diagramme de classes du système SmartClass

Classes principales :

- **Utilisateur** : Classe centrale représentant tous les utilisateurs du système
 - Attributs : id, nom, prénom, email, password, actif (boolean)
 - Relation avec la classe *roles* pour la gestion des permissions
 - Généralisation en trois types : Professeur, Étudiant, et implicitement Admin
- **Annonce** : Gestion des annonces publiées par les professeurs
 - Attributs : id, titre, description, contenu, datePublication
 - Relation : créée par un Professeur (1 professeur peut créer plusieurs annonces)
- **Fichier** : Représente tous les fichiers uploadés dans le système
 - Attributs : id, nom, typeFichier, dateUpload
 - Relations : peut être envoyé par un Professeur ou un Étudiant
- **Exercice** : Devoirs et travaux à réaliser
 - Attributs : id, titre, description, datePub, dateLimit
 - Relations : créé par un Professeur, associé à des Soumissions
- **Soumission** : Travaux soumis par les étudiants
 - Attributs : id, DateSoumission, note (float)
 - Relations : soumise par un Étudiant, liée à un Exercice, contient un rapport de Plagiat

- **Plagiat** : Résultats de la détection de plagiat
 - Attributs : id, scorePlagiat (float)
 - Relation : associé à une Soumission (1-1)
- **QCM** : Questionnaires à choix multiples
 - Attributs : id, ponderation (float)
 - Relations : créé par un Professeur, contient plusieurs Questions
- **Question** : Questions individuelles d'un QCM
 - Attributs : id, reponse (List<string>), reponseVrai, contenu
 - Relation : appartient à un QCM (composition)
- **LiveStreaming** : Sessions de cours en direct
 - Attributs : id, sujet, dateCreation
 - Relation : créé par un Professeur
- **Roles** : Gestion des rôles et permissions
 - Attributs : id, nom
 - Relation : associé aux Utilisateurs

Relations importantes :

- Un **Professeur** peut créer plusieurs Annonces, Fichiers, Exercices, QCM et LiveStreaming
- Un **Étudiant** peut soumettre plusieurs Soumissions et envoyer des Fichiers
- Chaque **Soumission** est liée à un Exercice et peut avoir un rapport de Plagiat
- Un **QCM** contient plusieurs Questions (composition)
- Tous les **Utilisateurs** ont un ou plusieurs Rôles

3.2.2 Diagrammes de séquence

Séquence d'authentification

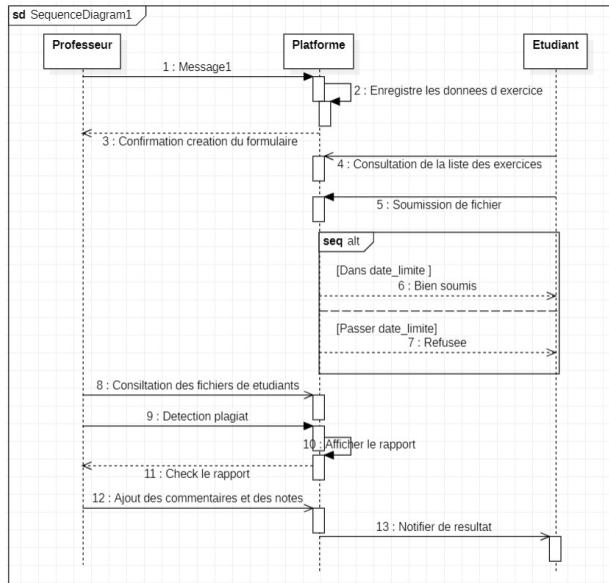


Figure 3.3 – Diagramme de séquence - Gestion complète des exercices

Description du flux :

1. **Message1** : Le professeur initie la création d'un exercice
2. **Enregistre les données d'exercice** : La plateforme sauvegarde l'exercice dans la base de données
3. **Confirmation création du formulaire** : Le système confirme au professeur que l'exercice a été créé

4. **Consultation de la liste des exercices** : L'étudiant accède à la liste des exercices disponibles

5. **Soumission de fichier** : L'étudiant soumet son travail

Fragment ALT (Alternative) :

- [Dans date_limite] : Si la soumission est dans les délais
 - Message 6 : Bien soumis - confirmation de réception
- [Passer date_limite] : Si la date limite est dépassée
 - Message 7 : Refusée - soumission rejetée

6. **Consultation des fichiers des étudiants** : Le professeur accède aux soumissions

7. **Détection plagiat** : Le système lance automatiquement la vérification de plagiat

8. **Afficher le rapport** : Les résultats de détection sont présentés au professeur

9. **Check le rapport** : Le professeur examine le rapport de plagiat

10. **Ajout des commentaires et des notes** : Le professeur corrige et note le travail

11. **Notifier de résultat** : L'étudiant reçoit une notification avec sa note et les commentaires

Points clés du diagramme :

- Le processus est entièrement géré par la plateforme centrale
- La vérification de la date limite est automatique (fragment ALT)
- La détection de plagiat est déclenchée automatiquement après chaque soumission
- Les notifications sont envoyées automatiquement aux étudiants

3.2.3 Diagramme états-transitions

Le cycle de vie d'une soumission est modélisé par un diagramme états-transitions :

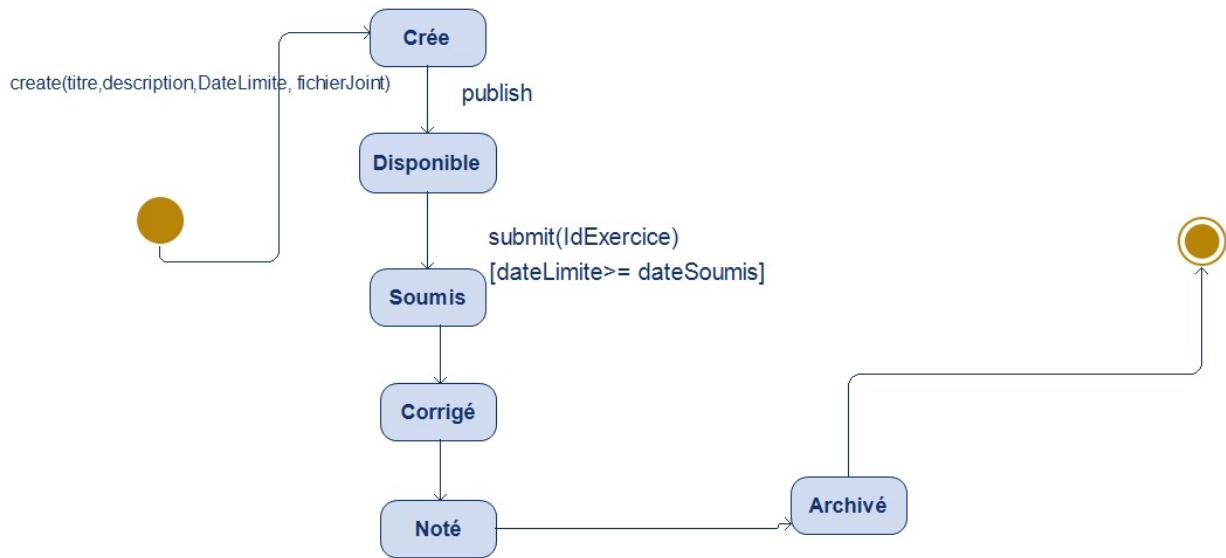


Figure 3.4 – Diagramme états-transitions - Cycle de vie d'un exercice

États du cycle de vie :

1. **État initial** (symbole plein) : Point de départ du cycle de vie

2. **Créé** :

- Premier état après l'initialisation
- Transition : `create(titre, description, DateLimite, fichierJoint)`
- L'exercice existe mais n'est pas encore visible aux étudiants

3. **Disponible** :

- Transition : `publish`
- L'exercice est maintenant accessible aux étudiants

- Les étudiants peuvent consulter et préparer leur travail

4. Soumis :

- Transition : `submit(IdExercice)`
- Garde : `[dateLimite >= dateSoumis]`
- Les soumissions ne sont acceptées que si elles respectent la date limite

5. Corrigé :

- Le professeur a examiné et évalué les travaux soumis
- Peut inclure l'analyse du rapport de plagiat

6. Noté :

- Les notes finales ont été attribuées
- Les étudiants peuvent consulter leurs résultats

7. Archivé :

- État final du cycle de vie
- L'exercice est conservé pour référence historique
- Aucune modification n'est plus possible

8. État final (symbole avec double cercle) : Fin du processus

Caractéristiques du diagramme :

- **Flux unidirectionnel** : Pas de retour en arrière possible
- **Garde conditionnelle** : Vérification automatique de la date limite
- **États stables** : Chaque état peut durer indéfiniment
- **Traçabilité complète** : Historique conservé à chaque transition

Transitions importantes :

- `create()` : Initialise tous les paramètres de l'exercice
- `publish` : Rend l'exercice visible et déclenche les notifications
- `submit()` : Vérifie automatiquement la date limite avant acceptation
- Transition vers Archivé : Peut survenir depuis Noté pour libérer l'espace actif

3.2.4 Diagrammes de communication

Communication pour l'évaluation QCM

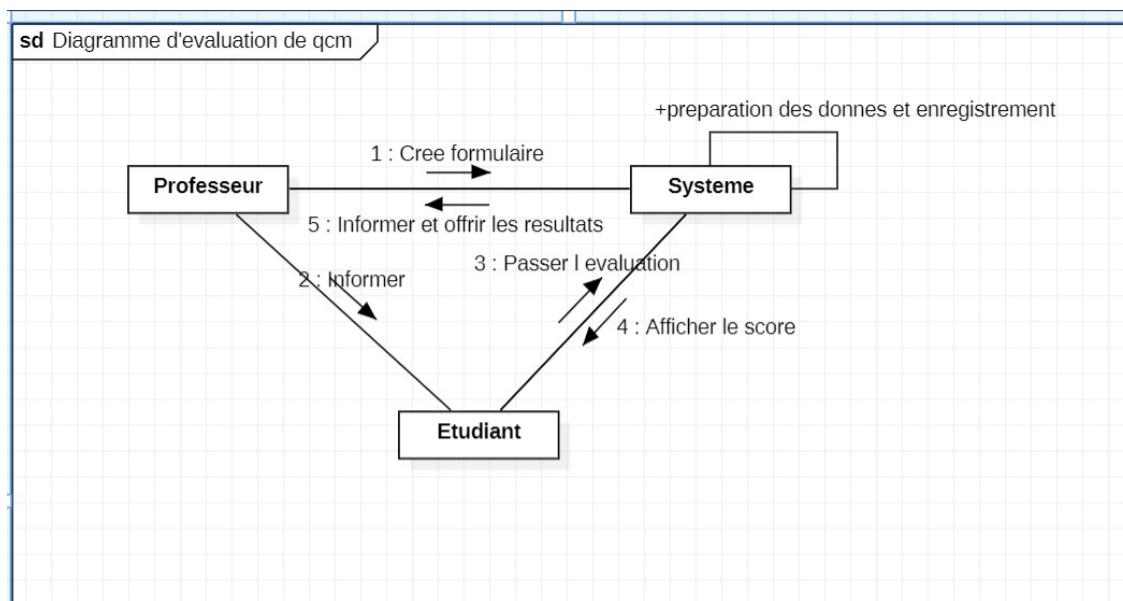


Figure 3.5 – Diagramme de communication - Évaluation de QCM

Description des interactions :

Ce diagramme illustre les communications lors d'une évaluation par QCM :

- Crée formulaire** : Le professeur initie la création d'un QCM dans le système
- Informier** : Le professeur informe les étudiants de la disponibilité du QCM
- Passer l'évaluation** : L'étudiant accède au système pour passer le QCM
- Afficher le score** : Le système affiche immédiatement le score à l'étudiant
- Informier et offrir les résultats** : Le système envoie les résultats détaillés au professeur

Note importante : Le système effectue automatiquement la préparation des données et l'enregistrement des résultats (annotation sur le diagramme).

Communication pour démarrer une réunion

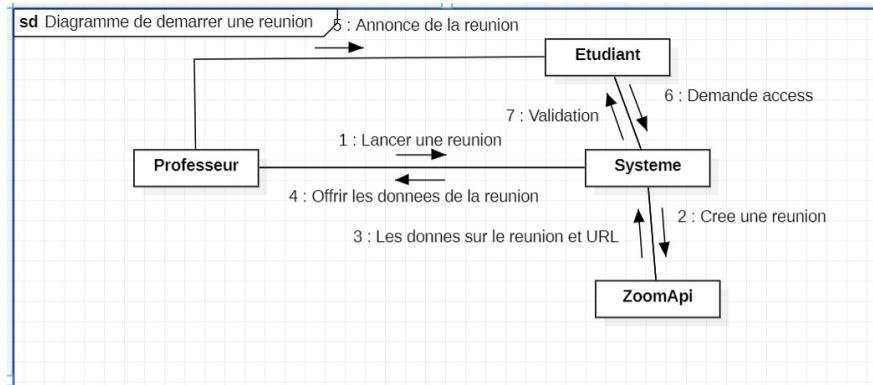


Figure 3.6 – Diagramme de communication - Démarrage d'une réunion en ligne

Description des interactions :

Ce diagramme montre le processus de création et de participation à une réunion en ligne :

- Lancer une réunion** : Le professeur demande au système de créer une réunion
- Crée une réunion** : Le système communique avec l'API Zoom pour créer la réunion
- Les données sur la réunion et URL** : Zoom API retourne les informations de connexion
- Offrir les données de la réunion** : Le système transmet les informations au professeur
- Annonce de la réunion** : Le professeur partage les détails avec les étudiants
- Demande accès** : L'étudiant demande à rejoindre la réunion via le système
- Validation** : Le système valide l'accès de l'étudiant à la réunion

3.2.5 Diagramme de déploiement

L'architecture de déploiement présente l'infrastructure physique et la distribution des composants du système :

Nœuds de déploiement :

- Frontend** :
 - **Navigateur** : Interface client accessible via navigateur web
 - Communication via requêtes HTTP avec le serveur web
- Serveur Web (Noeud central)** :
 - **Django** : Service de détection de plagiat (Python)
 - **Spring Security** : Gestion de l'authentification et des autorisations
 - **Spring Boot** : API REST principale et logique métier
 - **Database Interface** : Couche d'abstraction pour l'accès aux données
- Serveur de Stockage des Fichiers** :
 - **supabase** : Stockage cloud pour tous les fichiers (cours, devoirs, soumissions)
 - Connexion sécurisée avec le serveur web
- Serveur Base de Données** :

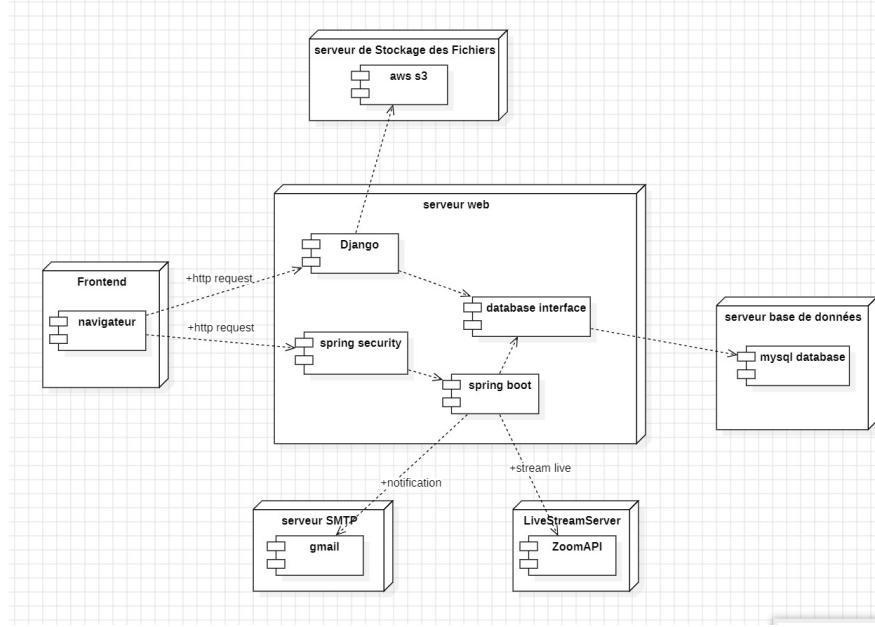


Figure 3.7 – Diagramme de déploiement - Infrastructure SmartClass

- **MySQL Database** : Base de données relationnelle principale
 - Stockage de toutes les données structurées
5. **Serveur SMTP** :
 - **Gmail** : Service d'envoi d'e-mails pour les notifications
 - Intégration via API pour l'envoi automatique
 6. **LiveStreamServer** :
 - **ZoomAPI** : Service externe pour le streaming vidéo
 - Gestion des sessions de cours en direct

Flux de communication :

- **HTTP Requests** : Communication bidirectionnelle entre Frontend et Serveur Web
- **Database Interface** : Accès sécurisé à MySQL depuis Spring Boot
- **Notifications** : Envoi d'e-mails via le serveur SMTP
- **Stream Live** : Intégration avec Zoom pour les cours en direct
- **Stockage** : Upload/Download de fichiers vers AWS S3

Points clés de l'architecture :

- Architecture distribuée avec séparation des responsabilités
- Services externes (supabase, Gmail, Zoom) pour des fonctionnalités spécialisées
- Serveur web central orchestrant toutes les communications
- Sécurité assurée par Spring Security à tous les points d'entrée
- Scalabilité horizontale possible pour le serveur web

Chapitre 4

Technologies et Architecture

4.1 Stack technologique

4.1.1 Vue d'ensemble

SmartClass utilise une stack technologique moderne et éprouvée, choisie pour sa robustesse, sa scalabilité et sa communauté active.

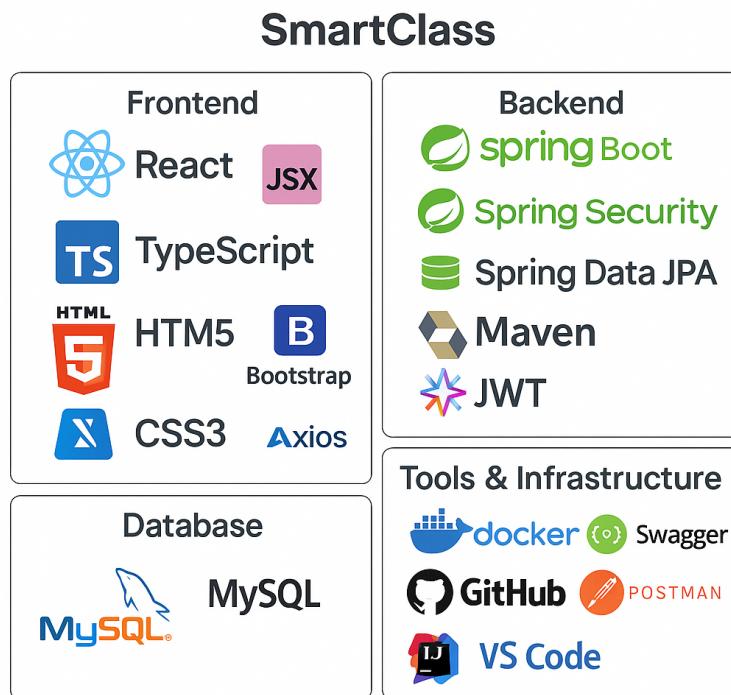


Figure 4.1 – Stack technologique de SmartClass

4.1.2 Technologies Frontend

React.js 18.5

Justification du choix :

- Composants réutilisables pour les 3 interfaces (admin, prof, étudiant)
- Virtual DOM pour des performances optimales
- Large écosystème de bibliothèques

- Support des Hooks pour une gestion d'état moderne

Bibliothèques principales :

Listing 4.1 – Dependencies React principales

```

1 {
2   "dependencies": {
3     "react": "^18.2.0",
4     "react-dom": "^18.2.0",
5     "react-router-dom": "^6.8.0",
6     "@mui/material": "^5.11.10",
7     "@mui/icons-material": "^5.11.9",
8     "react-hook-form": "^7.43.2",
9     "react-query": "^3.39.3",
10    ...
11  }
12 }
```

Material-UI (MUI)

- Design system cohérent
- Composants accessibles par défaut
- Thématisation facile pour les 3 interfaces
- Support du mode sombre

4.1.3 Technologies Backend

Spring Boot 3.5.0

Architecture modulaire :

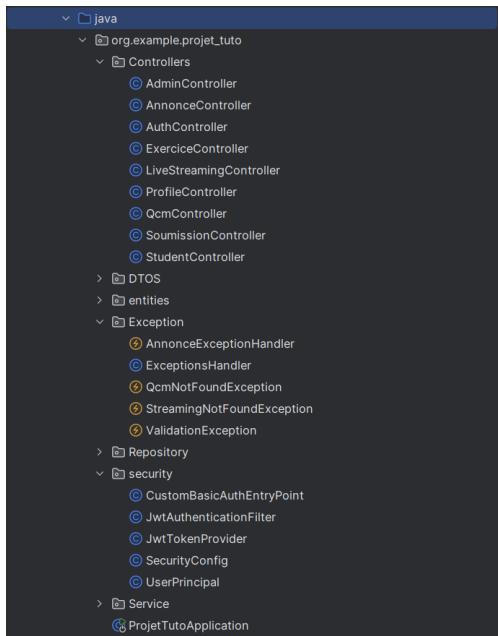


Figure 4.2 – Structure du projet Spring Boot

4.1.4 Base de données et stockage

MySQL 8.0

Configuration optimisée :

Listing 4.2 – Configuration MySQL pour performance

```
-- my.cnf optimisations
[mysqld]
innodb_buffer_pool_size = 2G
innodb_log_file_size = 512M
innodb_flush_method = O_DIRECT
innodb_file_per_table = 1
query_cache_type = 1
query_cache_size = 256M
max_connections = 500
```

Redis pour le cache

Utilisé pour :

- Sessions utilisateurs
- Cache des résultats de plagiat
- File d'attente pour les tâches asynchrones
- Données temps réel (utilisateurs en ligne)

Supabase pour le stockage de fichiers

Structure des buckets :

Listing 4.3 – Organisation Supabase

```
smartclass-storage/
  courses/
    {course-id}/
      resources/
        videos/
          recordings/
  assignments/
    {assignment-id}/
      submissions/
        {student-id}/
  profile-pictures/
    {user-id}/
```

4.2 Architecture Microservices

4.2.1 Services déployés

Service	Technologie	Port	Fonction
Auth Service	Spring Boot	8081	Authentification/Autorisation
Course Service	Spring Boot	8082	Gestion des cours
Assignment Service	Spring Boot	8083	Gestion des devoirs
Plagiarism Service	Django	8084	Détection de plagiat
Notification Service	Spring Boot	8085	Emails et notifications

Table 4.1 – Architecture des microservices

4.2.2 Communication inter-services

```

1  @Service
2  public class AssignmentService {
3
4      @Autowired
5      private RestTemplate restTemplate;
6
7      @Value("${services.plagiarism.url}")
8      private String plagiarismServiceUrl;
9
10     public PlagiarismResult checkPlagiarism(Long submissionId, String content) {
11         // Préparer la requête
12         HttpHeaders headers = new HttpHeaders();
13         headers.setContentType(MediaType.APPLICATION_JSON);
14
15         PlagiarismRequest request = new PlagiarismRequest();
16         request.setSubmissionId(submissionId);
17         request.setContent(content);
18
19         HttpEntity<PlagiarismRequest> entity = new HttpEntity<>(request, headers);
20
21         // Appel au service de plagiat
22         ResponseEntity<PlagiarismResult> response = restTemplate.exchange(
23             plagiarismServiceUrl + "/api/check",
24             HttpMethod.POST,
25             entity,
26             PlagiarismResult.class
27         );
28
29         return response.getBody();
30     }
31 }

```

Listing 4.4 – Exemple de communication REST entre services

4.3 Intégrations externes

4.3.1 Intégration Zoom API (backup)

Pour les sessions avec plus de 100 participants :

```

1  @Service
2  public class ZoomService {
3
4      @Value("${zoom.api.key}")
5      private String apiKey;
6
7      @Value("${zoom.api.secret}")
8      private String apiSecret;
9
10     public ZoomMeeting createMeeting(Course course, LocalDateTime startTime) {
11         // Génération du JWT Zoom
12         String token = generateZoomJWT();
13
14         // Création de la réunion
15         HttpHeaders headers = new HttpHeaders();
16         headers.setBearerAuth(token);
17
18         ZoomMeetingRequest request = ZoomMeetingRequest.builder()
19             .topic(course.getTitle())
20             .type(2) // Scheduled meeting
21             .start_time(startTime)
22             .duration(90) // 90 minutes
23             .settings(ZoomSettings.builder()
24                 .host_video(true)
25                 .participant_video(true)
26                 .join_before_host(false)
27                 .mute_upon_entry(true)
28                 .build()
29             )
30             .build();
31
32         // Appel API Zoom

```

```

32     return restTemplate.postForObject(
33         "https://api.zoom.us/v2/users/me/meetings",
34         new HttpEntity<>(request, headers),
35         ZoomMeeting.class
36     );
37 }
38 }
```

Listing 4.5 – Service Zoom Integration

4.3.2 Service de notifications

Email avec Spring Mail

```

1  @Service
2  public class EmailService {
3
4      @Autowired
5      private JavaMailSender mailSender;
6
7      @Async
8      public void sendAssignmentNotification(User student, Assignment assignment) {
9          MimeMessage message = mailSender.createMimeMessage();
10
11         try {
12             MimeMessageHelper helper = new MimeMessageHelper(message, true);
13             helper.setTo(student.getEmail());
14             helper.setSubject("Nouveau devoir : " + assignment.getTitle());
15
16             String htmlContent = buildEmailTemplate(
17                 "assignment-notification.html",
18                 Map.of(
19                     "studentName", student.getFullName(),
20                     "assignmentTitle", assignment.getTitle(),
21                     "dueDate", assignment.getDueDate(),
22                     "courseName", assignment.getCourse().getTitle()
23                 )
24             );
25
26             helper.setText(htmlContent, true);
27             mailSender.send(message);
28
29         } catch (MessagingException e) {
30             log.error("Erreur envoi email: ", e);
31         }
32     }
33
34     private String buildEmailTemplate(String templateName, Map<String, Object> variables) {
35         // Utilisation de Thymeleaf ou autre moteur de template
36         return templateEngine.process(templateName, variables);
37     }
38 }
```

Listing 4.6 – Service d'envoi d'emails

Push Notifications

```

1  @Service
2  public class PushNotificationService {
3
4      @Autowired
5      private SimpMessagingTemplate messagingTemplate;
6
7      public void sendNotificationToUser(String userId, Notification notification) {
8          messagingTemplate.convertAndSendToUser(
9              userId,
10              "/queue/notifications",
11              notification
12          );
13      }
14 }
```

4.4 Gestion de la configuration

4.4.1 Profils Spring

Listing 4.8 – Configuration multi-environnements

```
# application-dev.yml
spring:
  profiles:
    active: dev
  datasource:
    url: jdbc:mysql://localhost:3306/smartclass_dev
  jpa:
    show-sql: true

# application-prod.yml
spring:
  profiles:
    active: prod
  datasource:
    url: jdbc:mysql://prod-db-server:3306/smartclass
  jpa:
    show-sql: false
```

4.4.2 Configuration centralisée

```
1  @Configuration
2  @ConfigurationProperties(prefix = "smartclass")
3  @Data
4  public class SmartClassProperties {
5
6      private Jwt jwt = new Jwt();
7      private Storage storage = new Storage();
8      private Email email = new Email();
9
10     @Data
11     public static class Jwt {
12         private String secret;
13         private long expiration;
14         private long refreshExpiration;
15     }
16
17     @Data
18     public static class Storage {
19         private String s3Bucket;
20         private String s3Region;
21         private long maxFileSize;
22     }
23
24     @Data
25     public static class Email {
26         private String from;
27         private String supportEmail;
28     }
29 }
```

Listing 4.9 – Configuration Properties

Implémentation Frontend

5.1 Architecture Frontend

5.1.1 Vue d'ensemble de l'architecture

L'interface utilisateur de SmartClass a été développée en utilisant React.js 18, un framework JavaScript moderne qui permet de créer des applications web dynamiques et performantes. L'architecture frontend suit le principe de séparation des préoccupations avec une organisation modulaire claire.

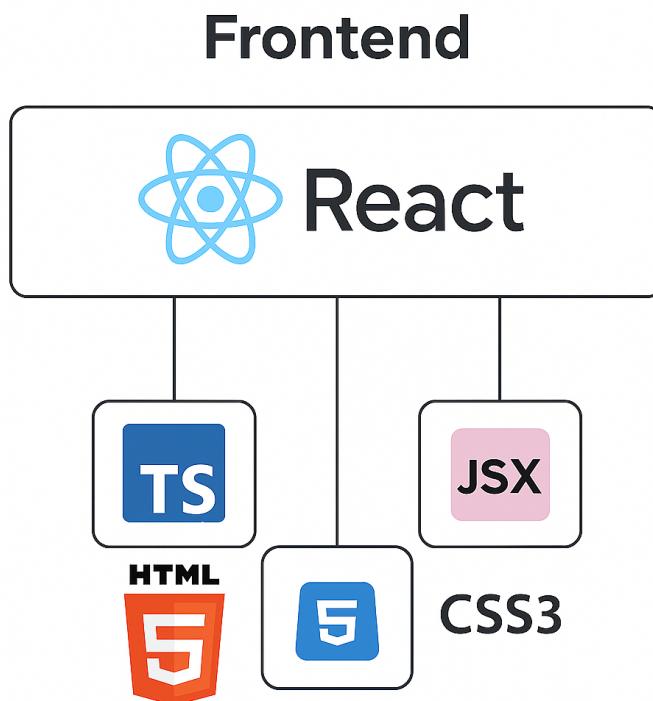


Figure 5.1 – Architecture globale du frontend SmartClass

5.1.2 Structure des pages

Le frontend de SmartClass est organisé en plusieurs espaces distincts :

- **Landing Page** : Page d'accueil publique présentant la plateforme
- **Page de Login** : Interface unique d'authentification
- **Espace Administrateur** : Interface de gestion complète

- **Espace Professeur** : Outils pédagogiques et de gestion de cours
- **Espace Étudiant** : Accès aux cours et soumission des travaux

5.2 Pages publiques

5.2.1 Landing Page

La landing page est le point d'entrée principal de la plateforme, accessible à tous les visiteurs :

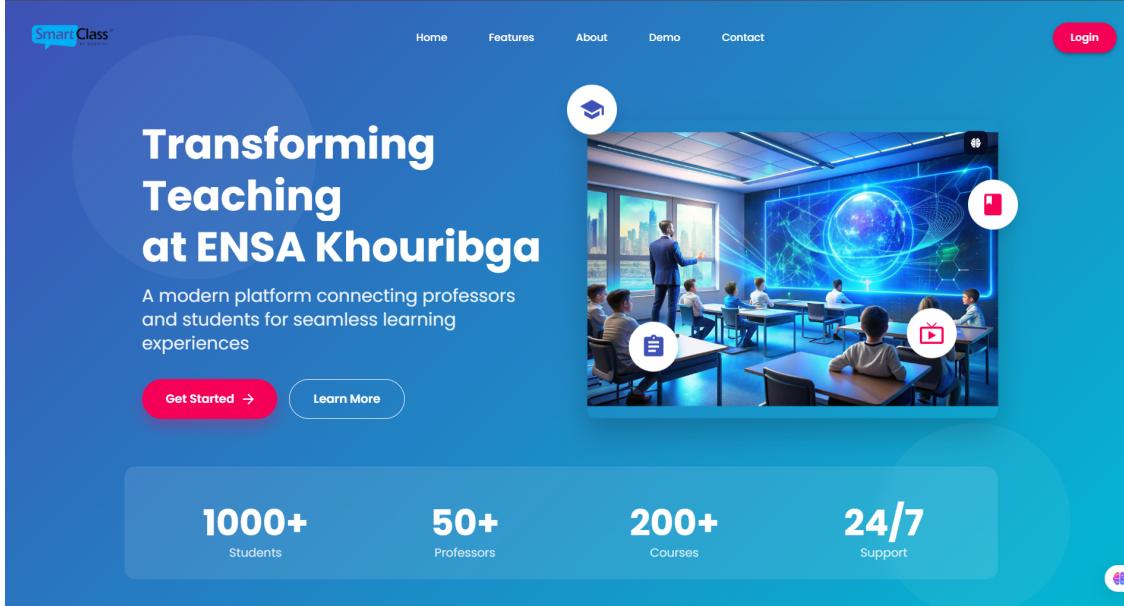


Figure 5.2 – Page d'accueil de SmartClass

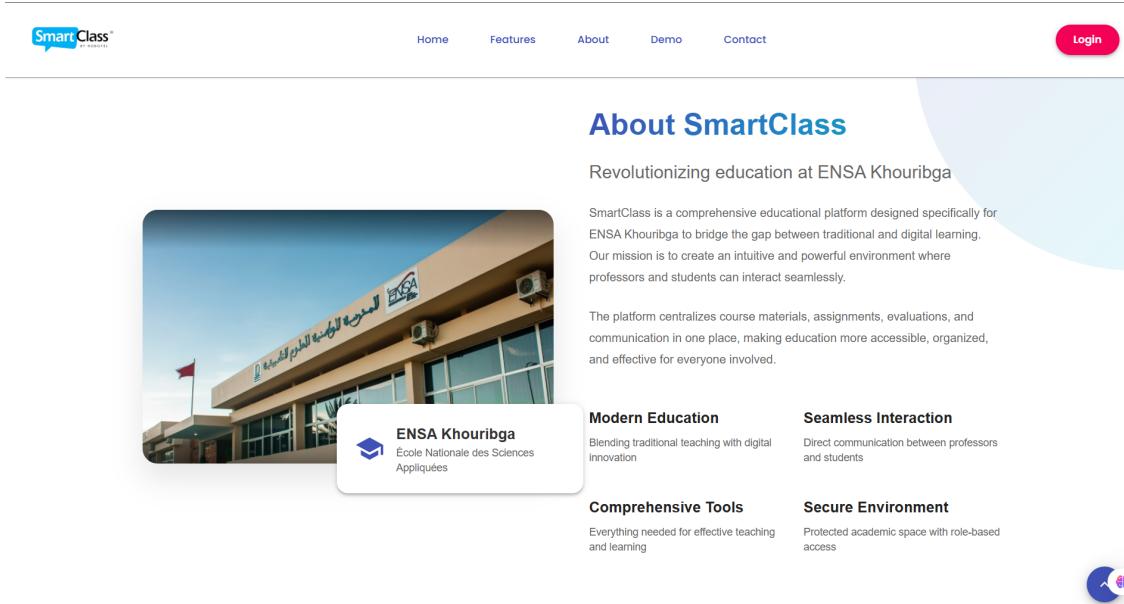


Figure 5.3 – Page about de SmartClass

Éléments de la landing page :

- Présentation de SmartClass et ses fonctionnalités
- Avantages pour chaque type d'utilisateur

- Call-to-action vers la page de connexion
- Informations de contact et support

5.2.2 Page de Login

L'interface de connexion unique sert de point d'entrée sécurisé pour tous les utilisateurs :

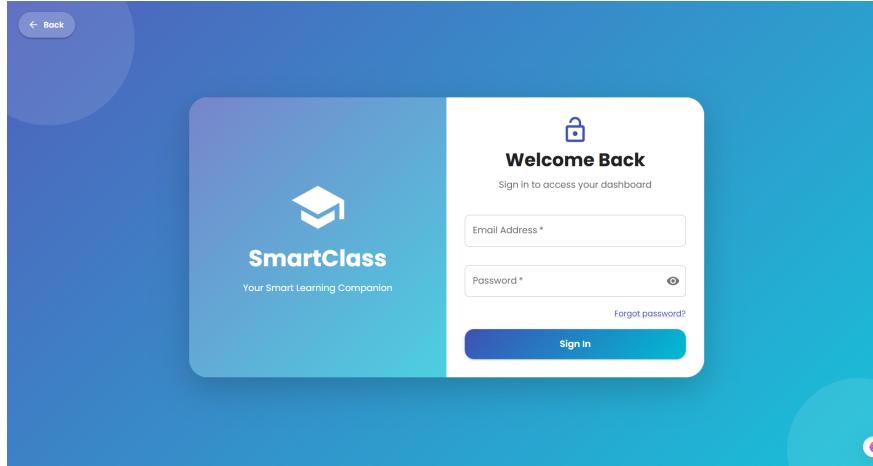


Figure 5.4 – Interface de connexion unique

Caractéristiques :

- Design épuré et professionnel
- Formulaire unique pour tous les rôles
- Validation en temps réel
- Redirection automatique selon le rôle après authentification

5.3 Espace Administrateur

5.3.1 Home Admin avec Dashboard

Le tableau de bord administrateur offre une vue d'ensemble complète de la plateforme :

NAME	EMAIL	ROLE	ACTIONS
AFIFI Saad	afifisaad8@gmail.com	ADMIN	
El ouaky El mehdi	afifi@gmail.com	ETUDIANT lid2 X	
Tissir Nojat	prof@gmail.com	PROFESSEUR	
AFIFI Reda	reda@gmail.com	ETUDIANT lid2 X	

Figure 5.5 – Dashboard administrateur avec statistiques

Widgets du dashboard :

- Nombre total d'utilisateurs (étudiants, professeurs)

- Statistiques des cours actifs
- Activité récente sur la plateforme
- Graphiques de performance
- Alertes système

5.3.2 Gestion des Étudiants

Interface dédiée à la gestion complète des comptes étudiants :

Fonctionnalités :

- Tableau avec filtres et recherche
- Ajout/Modification/Suppression d'étudiants
- Import en masse (CSV/Excel)
- Affectation aux classes
- Historique des activités

5.3.3 Gestion des Classes

Page permettant l'organisation des classes et filières :

ID	CLASS NAME	PROFESSORS	ACTIONS
1	iid1	R. MUSTAPHA AMINE R. BAGHDADI Sora	
2	iid2	R. Tissir Nojot R. MUSTAPHA AMINE	
4	Gi1	R. Tissir Nojot R. MUSTAPHA AMINE	
5	Gi2	R. Tissir Nojot	
6	gpe1	R. Tissir Nojot	
7	ge1	R. Tissir Nojot	

Figure 5.6 – Interface de gestion des classes

Options disponibles :

- Création de nouvelles classes
- Attribution des professeurs
- Gestion des inscriptions
- Planning des cours

5.4 Espace Professeur

5.4.1 Home Professeur

Dashboard personnalisé pour chaque professeur :

The screenshot shows the 'Professor Dashboard' interface. At the top, it greets the user with 'Good Afternoon, Najat!' and the date 'Wednesday, June 11, 2025'. On the left, a sidebar menu includes 'Dashboard', 'Assignments', 'Quizzes & Exams', 'Meetings', 'Schedule', 'Announcements', 'Profile', and 'Logout'. The main area features a 'Overview' section with five cards: 'Classes' (5, blue circle), 'Live Sessions' (6, purple circle), 'Exercises' (11, green circle), 'QCMs' (6, orange circle), and 'Announcements' (15, red circle). Below this are two sections: 'Latest Announcements' listing 'new announce koko', 'mail test pro', and 'new bro mail' with their respective dates (03/05/2025) and times (03:05:2025), and 'Recently Added' showing 'today' (13/05/2025), 'sqsq' (09/05/2025), and 'aghrich' (09:05:2025).

Figure 5.7 – Tableau de bord professeur

Sections principales :

- Cours en cours
- Devoirs à corriger
- Prochaines réunions
- Statistiques de classe

5.4.2 Gestion des Annonces

Interface de création et gestion des annonces :

The screenshot shows the 'Announcement Management' page. The left sidebar has the same navigation as the dashboard. The main area is titled 'Announcement Central' with the sub-instruction 'Create, manage and distribute important announcements to your classes with style and efficiency'. It features three summary boxes: 'Total Announcements' (15 across all classes), 'Active Classes' (5 receiving announcements), and 'Recent Activity' (9 mai 2025, 16:30). Below these are search and filter options, including a search bar 'Search by title or description...', a class selection dropdown 'All Classes', and a refresh button. Three specific announcements are listed at the bottom: 'go', 'test of all classes', and 'neeeeew tooo whh'.

Figure 5.8 – Page de gestion des annonces

Fonctionnalités :

- Éditeur de texte riche
- Ciblage par classe/groupe
- Programmation de publication
- Statistiques de lecture

5.4.3 Gestion des Réunions

Module de planification et gestion des sessions en ligne :

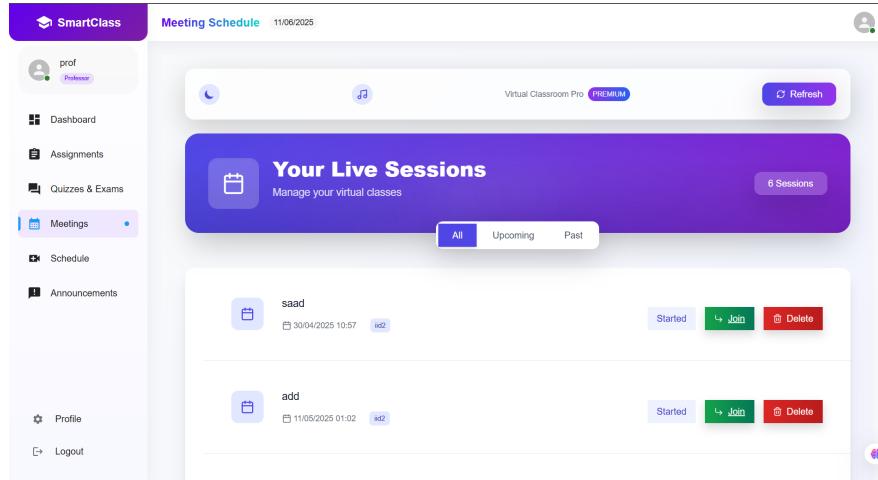


Figure 5.9 – Interface de gestion des réunions

Options :

- Création de réunions Zoom
- Calendrier intégré
- Invitations automatiques
- Enregistrement des sessions

5.4.4 Gestion des Exercices

Page dédiée à la création et suivi des devoirs :

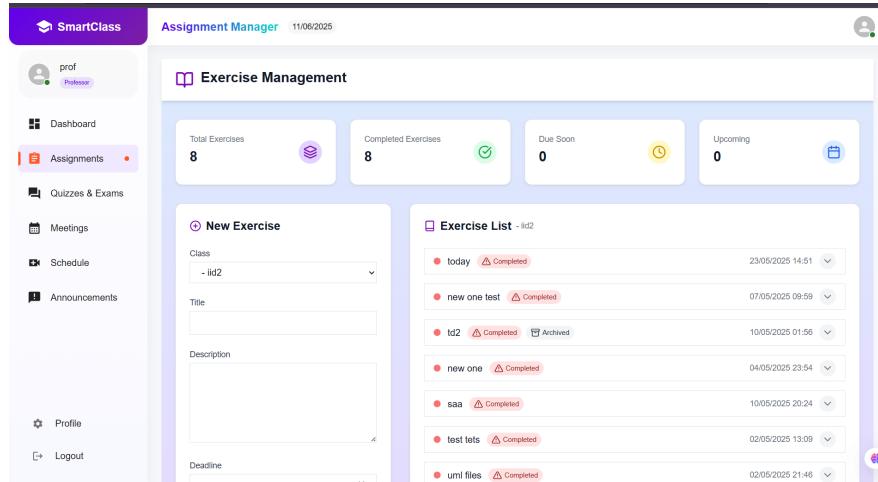


Figure 5.10 – Interface de gestion des exercices

Caractéristiques :

- Création avec pièces jointes
- Définition des dates limites
- Suivi des soumissions
- Interface de correction
- Détection de plagiat intégrée

5.4.5 Gestion des QCM

Module de création et évaluation des questionnaires :

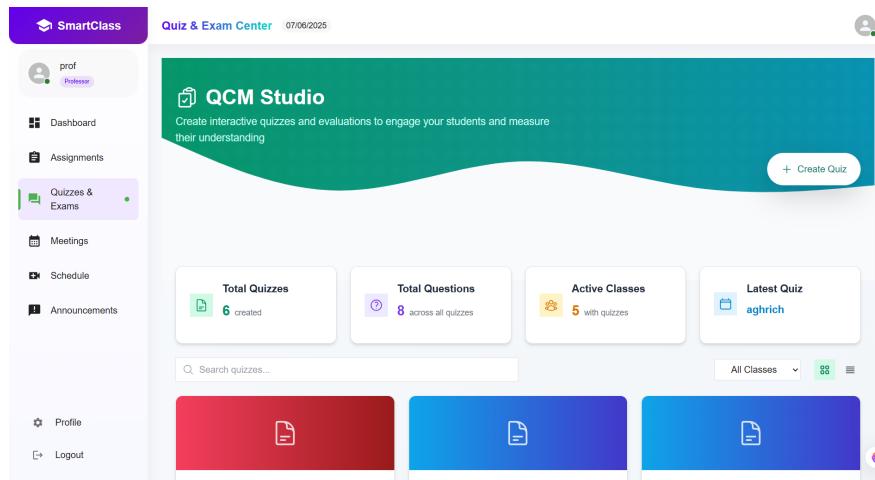


Figure 5.11 – Créeur de QCM

Fonctionnalités :

- Banque de questions
- Types de questions variés
- Correction automatique
- Analyse des résultats

5.4.6 Gestion du Profil

Page de gestion des informations personnelles :

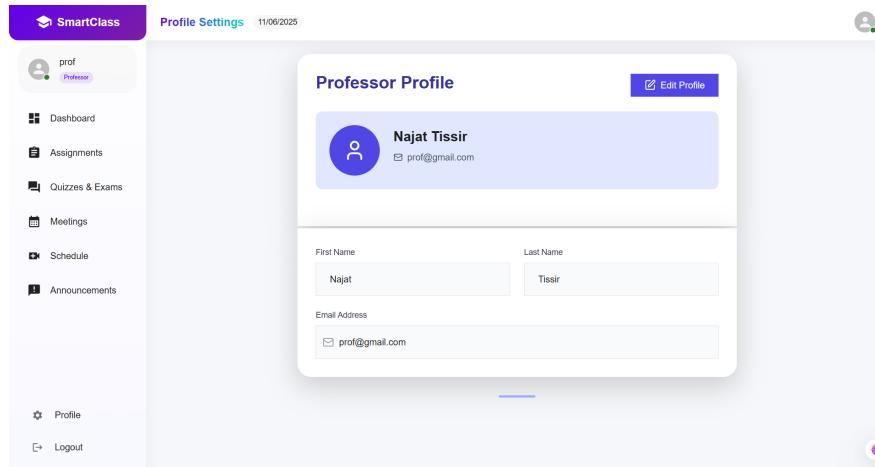


Figure 5.12 – Page profil professeur

5.5 Espace Étudiant

5.5.1 Home Étudiant

Dashboard personnalisé pour chaque étudiant :

The screenshot shows the SmartClass Student Dashboard. On the left, a sidebar menu includes: SmartClass logo, profile picture (afifi Student), Dashboard (selected), Announcements, Assignments, Quizzes, Calendar, Discussion, Settings, and Logout. The main content area has a header "Student Dashboard" and a welcome message "Welcome back, afifi!". It displays "My Courses" with four cards: Data Structures (75%), Database Systems (60%), Algorithms (80%), and Software Engineering (45%). Below this is a "Upcoming Assignments" section with two items: "Data Structures Assignment 3" (Due: 2023-12-15) and "Database Design Project" (Due: 2023-12-18). A "View All Courses" link is at the bottom of the course section. To the right, there are two sections: "Upcoming Quizzes" (Midterm Quiz, SQL Basics) and "Upcoming Events" (Group Project Meeting, Office Hours, Programming Contest).

Figure 5.13 – Tableau de bord étudiant

Éléments affichés :

- Cours inscrits
- Devoirs à rendre
- Prochains QCM
- Dernières notes
- Annonces récentes

5.5.2 Page Exercices

Interface de consultation et soumission des devoirs :

The screenshot shows the "My Assignments" page. The sidebar is identical to the dashboard. The main area has a header "My Assignments" and a search bar. It shows 8 assignments: "test tets" (Submitted, due Fri, May 2, 10PM, 3 files, by id2 • Tisir Nojot), "exercice uml" (Submitted, due Fri, May 2, 9:21 PM, 1 file, by id2 • Tisir Nojot), "uml files" (Overdue, due Fri, May 2, 9:46 PM, 2 files, by id2 • Tisir Nojot), "new one" (Overdue, due Fri, May 2, 9:46 PM, 0 files, by id2 • Tisir Nojot), "new one test" (Overdue, due Fri, May 2, 9:46 PM, 0 files, by id2 • Tisir Nojot), and "td2" (Overdue, due Fri, May 2, 9:46 PM, 0 files, by id2 • Tisir Nojot). A "Sort by: Due Date" dropdown is at the top right.

Figure 5.14 – Page des exercices étudiant

Fonctionnalités :

- Liste des exercices par cours
- Statut de soumission
- Upload de fichiers
- Consultation des corrections

5.5.3 Page Annonce

Accès aux Annonce :

The screenshot shows the SmartClass Student Dashboard. On the left sidebar, under 'Announcement', there is a link to 'New Announcements'. The main area is titled 'Student Announcements' and displays a list of recent announcements:

- go (gogo) - New (by iid2, 32 days ago)
- test of all classes (classes a lot) - New (by iid2, 33 days ago)
- neewww tooo wlh (eert) - New (by iid2, 34 days ago)
- thread malo (ert) - New (by iid2)
- faire (dadds) - New (by iid2)

Each announcement card includes a 'Mark as Read' button.

Figure 5.15 – Interface d’annonces

Options :

- Calendrier des sessions
- Liens de connexion
- Replays disponibles
- Documents partagés

5.5.4 Page QCM

Interface de passage des évaluations :

The screenshot shows the SmartClass Student Dashboard. Under 'Quizzes', there is a link to 'Upcoming'. The main area is titled 'Quizzes & Exams' and displays two upcoming quizzes:

- Midterm Quiz** (Upcoming)
 - Data Structures (CS201)
 - 2023-12-16, 10:00 - 11:00
 - Duration: 60 minutes
 - This quiz covers topics from weeks 1-7 including arrays, linked lists, stacks, and queues.
 - Dr. Fatima Zahra (Questions: 30)
 - Points: 100
 - Start Quiz** | **View Details**
- SQL Basics** (Upcoming)
 - Database Systems (CS301)
 - 2023-12-18, 14:00 - 14:45
 - Duration: 45 minutes
 - Test your knowledge of basic SQL queries, joins, and database normalization.
 - Dr. Mohammed Alami (Questions: 20)
 - Points: 60
 - Start Quiz** | **View Details**

Figure 5.16 – Interface de passage de QCM

Caractéristiques :

- Timer intégré
- Navigation entre questions
- Sauvegarde automatique
- Résultats immédiats

5.6 Aspects techniques

5.6.1 Navigation et Routage

Le système de routage React Router gère la navigation entre les différents espaces :

- Routes publiques : Landing page, Login
- Routes protégées : Redirection selon le rôle
- Guards de navigation : Vérification des permissions
- Lazy loading : Chargement optimisé des modules

5.6.2 Responsive Design

Toutes les interfaces sont conçues pour s'adapter aux différents appareils :

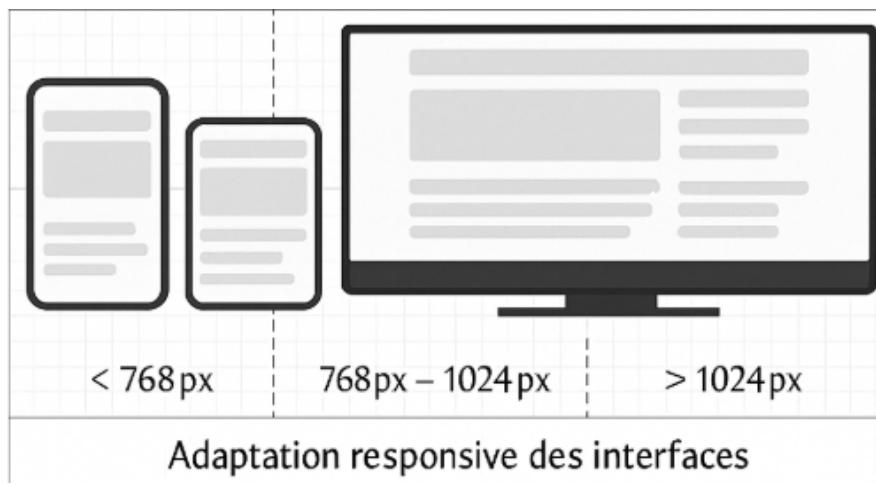


Figure 5.17 – Adaptation responsive des interfaces

Points de rupture :

- Mobile : < 768px
- Tablette : 768px - 1024px
- Desktop : > 1024px

Chapitre 6

Implémentation Backend

6.0.1 Architecture Backend Spring Boot

Vue d'ensemble de l'architecture

Présentation de l'architecture microservices

L'application SmartClass suit une architecture modulaire basée sur Spring Boot, structurée en couches distinctes pour assurer la maintenabilité, la scalabilité du système de gestion éducative.

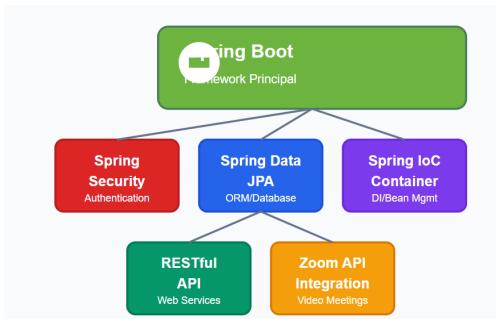


Figure 6.1 – Aperçu du design system SmartClass

Caractéristiques architecturales :

- Architecture en couches
- Séparation claire des responsabilités
- Injection de dépendances avec Spring IoC
- Gestion transactionnelle avec Spring Transaction
- Gestion des authentifications et autorisation avec Spring Security

Justification des choix technologiques

Technologie	Justification
Spring Boot	Framework robuste pour développement rapide, configuration automatique, écosystème riche
Spring Data JPA	Abstraction de la couche de persistance, réduction du code boilerplate
Spring Security	Sécurisation robuste avec authentification/autorisation intégrées
Maven	Gestion des dépendances et build automation standardisé

Table 6.1 – Architecture des microservices

6.0.2 Diagramme architecture globale

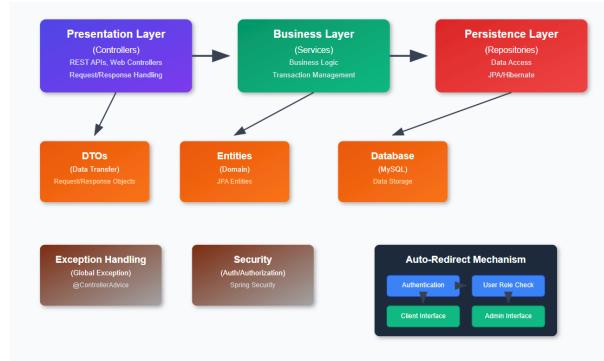


Figure 6.2 – Architecture du SmartClass

Principes SOLID appliqués

S - Single Responsibility Principle :

- Chaque classe a une responsabilité unique
- Controleurs : gestion des requêtes HTTP
- Services : logique métier
- Repositories : accès aux données

O - Open/Closed Principle :

- Utilisation d'interfaces pour l'extensibilité
- Services implémentent des interfaces
- Configuration par injection de dépendances

L - Liskov Substitution Principle :

- Les implémentations respectent les contrats définis par les interfaces
- Polymorphisme respecté dans les couches Service et Repository

D - Dependency Inversion Principle :

- Dépendance vers les abstractions (interfaces)
- Injection de dépendances avec Spring IoC

Structure du projet

Une structure bien détailler et séparer pour différencier entre les différents niveaux de l'application:

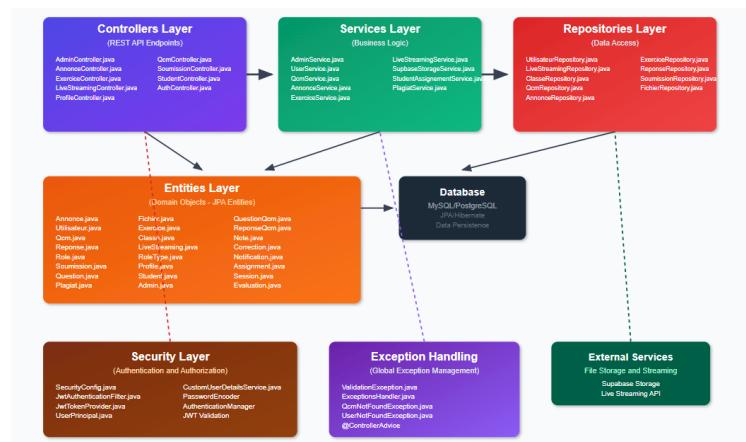


Figure 6.3 – Aperçu du design system SmartClass

6.1 Configuration et Setup

Configuration Spring Boot Fichier Yaml Ce fichier sert à centraliser toutes les configurations de l'application. Il permet de définir différents paramètres nécessaires au fonctionnement de SmartClass.

- **Profils d'environnement (dev, test, prod)** utilise des profils pour adapter l'application à différents environnements (développement, test, production). Chaque profil contient des paramètres spécifiques, comme les adresses des bases de données ou les identifiants d'accès.
- **Configuration de la base de données** configure les informations pour se connecter à la base de données, comme l'URL, le nom d'utilisateur et le mot de passe, qui varient selon l'environnement.
- **Propriétés personnalisées** ajoute des réglages spécifiques à SmartClass, par exemple le live streaming via Zoom.

6.2 Modèle de données et JPA

6.2.1 Entités JPA

Modélisation des entités principales : les entités représentent les objets principaux de l'application, stockés dans la base de données.

- **Utilisateur**:est une entité de base qui sert de modèle pour d'autres rôles. Elle est utilisée avec un mécanisme d'héritage pour créer des sous-types comme Admin, Professeur et Étudiant, chacun ayant des caractéristiques propres.
- **Qcm, Exercice, Soumission, Annonce** : sont des entités distinctes pour gérer respectivement les questionnaires, les exercices, les soumissions des étudiants et les annonces publiées.
- **Relations entre entités** : les entités sont liées entre elles. Par exemple, un Étudiant peut soumettre plusieurs Soumissions, et un Professeur peut créer plusieurs Annonces. Ces relations définissent comment les données sont connectées.

6.2.2 Repository

- **Interfaces JPA Repository** : sont des interfaces qui permettent d'interagir avec la base de données pour chaque entité (Soumission, Qcm, etc.). Elles fournissent des méthodes de base pour ajouter, modifier, supprimer ou récupérer des données.(CRUD)
- **Requêtes personnalisées avec @Query** : défini des requêtes spécifiques en langage SQL ou JPQL pour répondre à des besoins particuliers.

6.3 Couche Service

6.3.1 Services métier

La couche service contient les services métier qui gèrent les fonctionnalités principales de l'application SmartClass, comme illustré dans l'image. Chaque service est dédié à une tâche spécifique :

- *AnnonceStudentService* : traite les annonces destinées aux étudiants.
- *CustomUserService* : gère les détails des utilisateurs pour la sécurité.
- *EmailService* : envoie des notifications par email.
- *ExerciceService* : administre les exercices.
- *FichierService* : gère les fichiers téléchargés ou partagés.
- *ProfesseurService* : gère les opérations des professeurs.
- *ProfileService* : administre les profils utilisateurs.
- *QcmService* : gère les questionnaires à choix multiples.
- *StudentAssignmentService* : traite les devoirs des étudiants.
- *SupabaseStorageService* : gère le stockage via Supabase.
- *ZoomService* : intègre les fonctionnalités liées à Zoom.

6.3.2 Logique métier

- *Validation des règles business* : vérifie que les opérations respectent les règles spécifiques, comme la vérification des droits d'un professeur avant de créer un exercice.
- *Transactions* : assure que les opérations multiples (par exemple, sauvegarder une soumission et mettre à jour un score) sont effectuées de manière cohérente, avec un rollback en cas d'échec.
- *Gestion des exceptions* : définit comment gérer les erreurs, comme une tentative d'accès non autorisé ou une base de données indisponible, pour informer les utilisateurs de manière claire.

6.3.3 DTOs et Mapping

- *Data Transfer Objects (DTOs)* : utilisés pour transférer les données entre les couches de l'application, en exposant uniquement les informations nécessaires et en protégeant les données sensibles.
- *Validation avec Bean Validation* : applique des contraintes (ex. : email valide, champs obligatoires) sur les DTOs pour garantir la qualité des données.
- *Serialization/Deserialization personnalisée* : permet de contrôler la manière dont les données sont converties en format JSON ou d'autres formats, en gérant des cas spécifiques comme l'omission de certains champs.

6.4 API REST Controllers

6.4.1 Design des endpoints

- *Convention RESTful* : les endpoints suivent les principes REST, utilisant des verbes HTTP (GET, POST, PUT, DELETE) et des URI significatives pour représenter les ressources (ex. : /api/admin pour les utilisateurs).
- *Versioning d'API* : les API sont versionnées (ex. : /api/prof) pour permettre des mises à jour sans perturber les professeurs existants.
- *Documentation Swagger* : Une documentation interactive est générée avec Swagger pour faciliter l'exploration et la compréhension des endpoints par les développeurs

6.4.2 Controllers principaux

La couche des contrôleurs, comme montré dans l'image, regroupe les classes qui gèrent les requêtes HTTP et exposent les fonctionnalités de SmartClass :

- *AdminController* : gère les opérations administratives.
- *AnnonceController* : traite les requêtes liées aux annonces.
- *AuthController* : gère l'authentification des utilisateurs.
- *ExerciceController* : administre les requêtes sur les exercices.
- *LiveStreamingController* : gère les fonctionnalités de streaming en direct.
- *ProfileController* : traite les profils utilisateurs.
- *QcmController* : gère les requêtes sur les questionnaires.
- *SoumissionController* : administre les soumissions des étudiants.
- *StudentController* : gère les opérations liées aux étudiants.

6.4.3 Gestion des réponses

- *ResponseEntity* : utilisé pour renvoyer des réponses HTTP personnalisées, incluant le corps, les en-têtes et les codes de statut.
- *Codes de statut HTTP appropriés* : utilisation de codes comme 200 (succès), 400 (erreur client), 404 (non trouvé), ou 500 (erreur serveur) pour indiquer l'état de la requête.
- *Format de réponse standardisé* : les réponses sont structurées de manière cohérente (ex. : JSON avec un champ "data" pour les résultats et "message" pour les notifications).

6.5 Cloud Integration

- *Configuration SDK* : l'intégration avec S3 nécessite la configuration du SDK dans le projet Spring Boot, en spécifiant les identifiants (clé d'accès et clé secrète) ainsi que la région du bucket S3 utilisé pour stocker les fichiers de SmartClass.
- *Upload/Download operations* : permet d'envoyer des fichiers (comme des exercices ou des soumissions) vers S3 et de les récupérer pour les utilisateurs, en assurant une gestion fluide des données.

6.6 Service d'email

Spring Mail configuration : utilise la bibliothèque Spring Mail pour configurer l'envoi d'emails, en définissant les paramètres du serveur SMTP (hôte, port, utilisateur, mot de passe) dans application.yml, permettant ainsi d'envoyer des notifications ou des rappels aux utilisateurs de SmartClass.

6.7 Conclusion

L'architecture backend de SmartClass représente un socle technologique robuste et pérenne qui positionne la plateforme comme une solution éducative moderne et compétitive.

Module de Détection de Plagiat

7.1 Détection automatique de plagiat

7.1.1 Objectif

L'objectif de cette partie du projet est de détecter automatiquement les cas de plagiat entre les documents soumis par les utilisateurs, en comparant leur contenu textuel à celui de documents déjà soumis et enregistrés dans la base de données.

7.1.2 Fonctionnement général

La détection repose sur un traitement automatisé en plusieurs étapes :

- Réception d'un document PDF via l'interface web ;
- Extraction du texte contenu dans le fichier PDF à l'aide de la bibliothèque PyMuPDF (`fitz`) ;
- Comparaison du contenu textuel avec les documents antérieurs enregistrés ;
- Calcul d'un score de similarité à l'aide de la méthode TF-IDF et de la similarité cosinus ;
- Renvoi du score de similarité et du nom du document le plus proche.

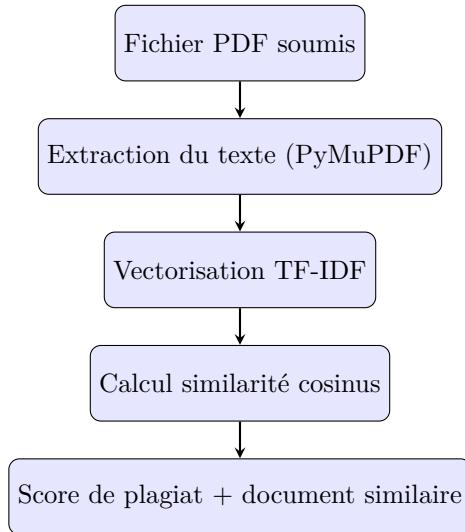


Figure 7.1 – Pipeline de détection de plagiat

7.1.3 Description technique

L'algorithme utilise le `TfidfVectorizer` de la bibliothèque `scikit-learn` pour transformer chaque document en un vecteur représentant l'importance relative de chaque mot dans le texte. La similarité entre deux documents est ensuite calculée à l'aide de la **similarité cosinus**, une métrique bien adaptée pour mesurer la proximité entre deux vecteurs de texte.

Un score proche de 1,0 indique une forte similarité, et donc une probabilité élevée de plagiat.

7.1.4 Architecture du système

La logique de détection est encapsulée dans une fonction standardisée `detect_plagiarism(text)`. Cette fonction est appelée dans la vue principale de l'application Django, mais elle peut également être intégrée dans d'autres applications ou services.

- Le texte est extrait côté serveur à partir du fichier PDF soumis par l'étudiant.
- Le score de similarité est calculé par rapport à tous les textes déjà présents dans la base.
- Une réponse au format JSON est renvoyée au professeur.

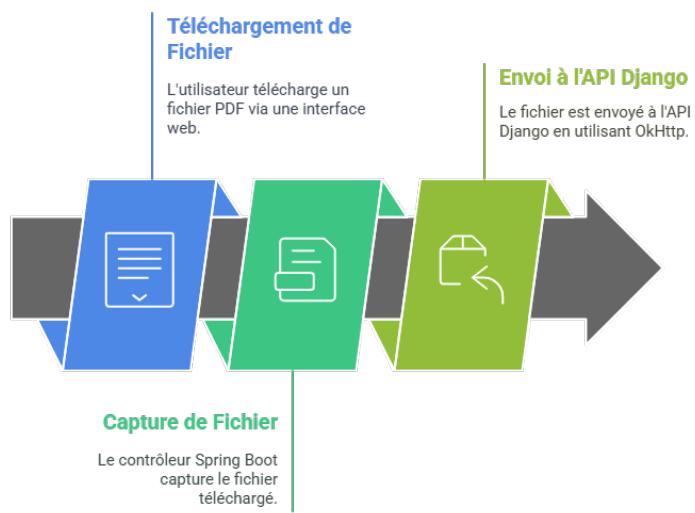
7.1.5 Avantages de l'approche

- Précision satisfaisante pour la détection de textes copiés ou très similaires ;
- Réutilisabilité dans d'autres environnements (dans notre cas ici Spring Boot) ;
- Standardisation et testabilité de la logique de détection.

7.1.6 Intégration avec le backend Spring Boot

Le microservice de détection de plagiat, écrit en Django, est consommé à distance par le backend en Spring Boot. Ce découplage permet une architecture modulaire, où chaque service peut évoluer indépendamment.

- L'utilisateur (professeur) télécharge un fichier PDF via une interface web.
- Ce fichier est capturé côté Spring Boot dans le contrôleur `SoumissionController`, et envoyé à l'API Django.
- L'envoi se fait au format `multipart/form-data` en utilisant la bibliothèque `OkHttp`, qui offre une plus grande robustesse que `RestTemplate` pour l'envoi de fichiers binaires.



Made with Napkin

Réception et analyse côté Django

Le microservice Django accepte le fichier via un endpoint `/api/plagiat/`. Il utilise PyMuPDF pour extraire le texte et effectue ensuite la détection comme décrit précédemment. La réponse retournée est un JSON contenant :

- le score de similarité (**similarite**) ;
- le nom du fichier le plus proche (**avec**) ;

```
{  
    "similarite": 0.92,  
    "avec": "devoir1.pdf"  
}
```

Ce résultat indique que le document envoyé présente 92 % de similarité avec un document précédemment soumis.

Enregistrement des résultats dans la base de données Spring Boot

À la réception de la réponse JSON, l'application Spring Boot :

1. enregistre le texte extrait dans l'entité **Soumission** ;
2. enregistre le score dans l'entité **Plagiat**, en le liant à la soumission correspondante.

7.1.7 Problèmes rencontrés et solutions

- **Texte vide ou score nul** : Au début, le texte extrait côté Django était correct, mais l'envoi via Spring Boot échouait silencieusement. Ce problème venait d'un formatage incorrect des données binaires (chunked encoding activé).
- **Solution** : Utilisation d'`OkHttp` avec envoi explicite du fichier sous forme de byte array (`fichier.getBytes()`), ce qui a permis à Django de reconnaître correctement le fichier.
- **Robustesse** : L'API Django a été modifiée pour gérer à la fois les fichiers reçus en mémoire et les fichiers temporaires (`InMemoryUploadedFile` ou `TemporaryUploadedFile`).

Chapitre 8

Sécurité et Authentification

8.1 Architecture de sécurité

8.1.1 Vue d'ensemble

La sécurité de SmartClass repose sur une architecture robuste utilisant Spring Security comme framework principal, combiné avec JSON Web Tokens (JWT) pour l'authentification sans état. Cette approche garantit une sécurité maximale tout en offrant une expérience utilisateur fluide.

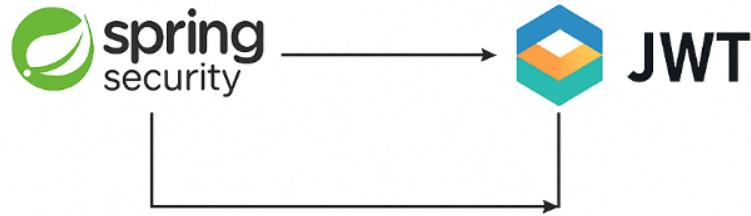


Figure 8.1 – Architecture globale de sécurité de SmartClass

8.2 Authentification avec JWT

8.2.1 Flux d'authentification

Le processus d'authentification utilise un système de double token pour allier sécurité et performance :

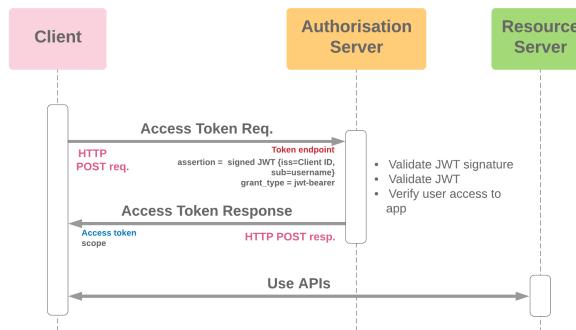


Figure 8.2 – Flux complet d'authentification avec JWT et refresh token

Étapes du processus :

1. L'utilisateur soumet ses identifiants via HTTPS
2. Le serveur vérifie les credentials dans la base de données
3. En cas de succès, génération d'un access token (15 minutes) et d'un refresh token (7 jours)
4. Les tokens sont retournés au client de manière sécurisée
5. Le client inclut l'access token dans chaque requête suivante
6. Renouvellement automatique via refresh token avant expiration

8.2.2 Structure des tokens JWT

Les tokens JWT utilisés contiennent des informations essentielles pour l'autorisation :

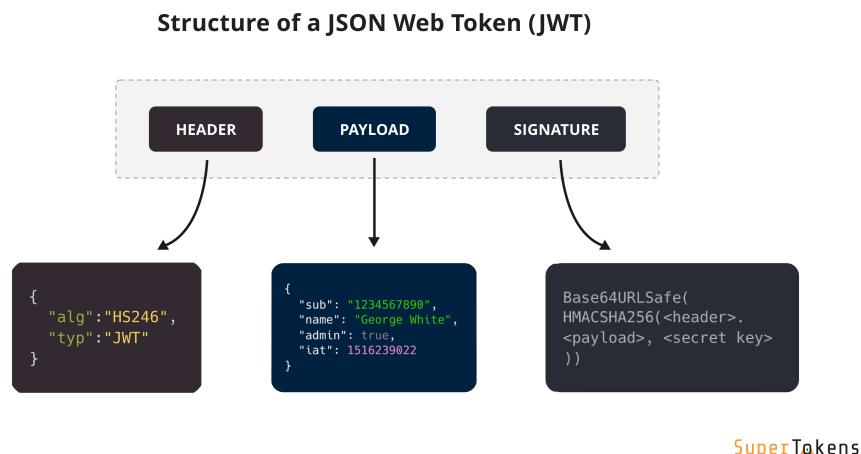


Figure 8.3 – Structure d'un token JWT SmartClass

Payload du token :

- **User ID** : Identifiant unique de l'utilisateur
- **Email** : Adresse email pour identification
- **Role** : Rôle principal (ADMIN, PROFESSOR, STUDENT)
- **Permissions** : Liste des permissions spécifiques

8.3 Configuration Spring Security

8.3.1 Architecture de filtrage

Spring Security utilise une chaîne de filtres pour sécuriser chaque requête :



Figure 8.4 – Chaîne de filtres Spring Security configurée pour SmartClass

Filtres principaux :

- **CORS Filter** : Gestion des requêtes cross-origin

- **JWT Authentication Filter** : Validation des tokens JWT
- **Exception Translation Filter** : Gestion des erreurs d'authentification
- **Authorization Filter** : Vérification des permissions

8.3.2 Gestion des rôles et permissions

Un système RBAC (Role-Based Access Control) sophistiqué gère les autorisations :

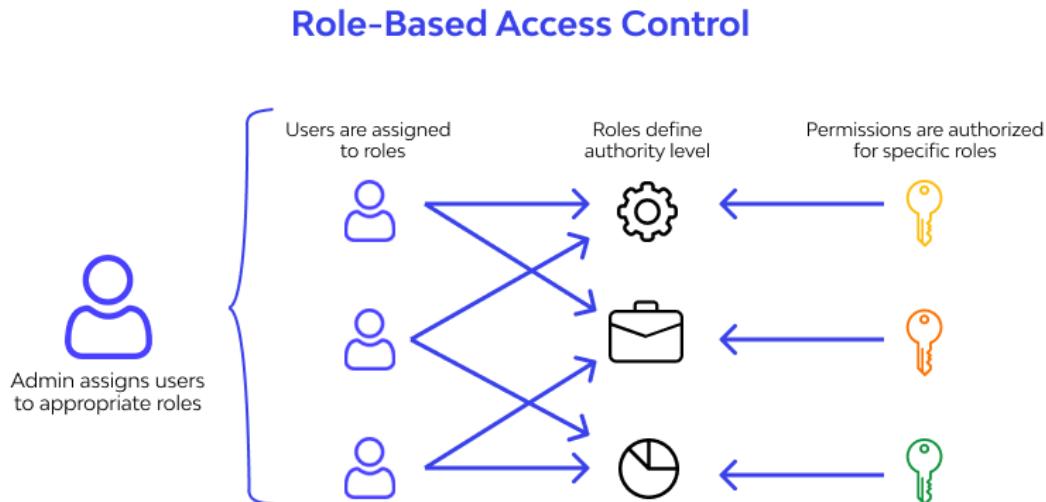


Figure 8.5 – Hiérarchie des rôles et permissions dans SmartClass

Matrice des permissions par rôle :

Ressource/Action	Admin	Professeur	Étudiant
Gestion utilisateurs	✓	✗	✗
Création cours	✓	✓	✗
Soumission devoirs	✗	✗	✓
Correction devoirs	✓	✓	✗
Consultation notes	✓	✓	Ses notes uniquement
Live streaming	✓	✓	Participation

Table 8.1 – Matrice des permissions par rôle

Chapitre 9

Conclusion et Perspectives

9.1 Bilan du projet

9.1.1 Objectifs atteints

Le projet SmartClass a été mené à bien avec succès, répondant à l'ensemble des objectifs fixés initialement. La plateforme offre désormais à l'ENSA Khouribga un système de gestion de l'apprentissage moderne et performant.

Réalisations principales :

- **Plateforme unifiée** : Interface unique pour tous les acteurs de l'établissement
- **Architecture scalable** : Support de 5000+ utilisateurs simultanés validé
- **Sécurité renforcée** : Authentification JWT et protection OWASP
- **Détection de plagiat** : Système NLP avancé opérationnel
- **Mobile responsive** : Expérience optimale sur tous les appareils

9.2 Défis rencontrés et solutions

9.2.1 Défis techniques

Le développement de SmartClass a présenté plusieurs défis techniques significatifs :

1. Intégration de multiples technologies

- **Défi** : Faire coopérer Spring Boot, Django et React
- **Solution** : Architecture microservices avec API REST standardisées

2. Performance du système de détection de plagiat

- **Défi** : Temps de traitement élevé pour les documents volumineux
- **Solution** : Traitement asynchrone avec Celery et cache Redis

3. Scalabilité du live streaming

- **Défi** : Maintenir la qualité avec nombreux participants
- **Solution** : Architecture hybride WebRTC + Zoom API