

DOSSIER DE PROJET CDA

Générateur d'Emploi du Temps Scolaire

Rapport et projet présenté par Elie AIGON

Pour le passage du Titre RNCP de niveau 6

Concepteur Développeur d'applications - N°37873

Formation : Bachelor IT, Spécialisation Développement Logiciels

Année : 2025

Durée du projet : 8 semaines



TABLE DES MATIÈRES

1. Compétences mises en œuvre

2. Introduction et contexte

- 2.1 Présentation du projet
- 2.2 Problématique
- 2.3 Objectifs du projet

3. Conception

- 3.1 Description du client
- 3.2 Analyse de l'existant
- 3.3 Expression des besoins
- 3.4 Limites du projet

4. Environnement technique

- 4.1 Stack technologique
- 4.2 Architecture du système
- 4.3 Configuration de l'environnement
- 4.4 Architecture Frontend (React)
- 4.5 Patterns de communication et intégration
- 4.6 Organisation des dossiers
- 4.7 Modèle de données
- 4.8 Sécurité et authentification
- 4.9 API REST

5. Réalisations et mise en œuvre des compétences

- 5.1 Compétence : Analyser les besoins et maquetter une application
- 5.2 Compétence : Définir l'architecture logicielle d'une application
- 5.3 Compétence : Concevoir et mettre en place une base de données relationnelle
- 5.4 Compétence : Développer des composants d'accès aux données SQL et NoSQL
- 5.5 Compétence : Développer des composants métier
- 5.6 Compétence : Développer des interfaces utilisateur
- 5.7 Compétence : Installer et configurer son environnement de travail
- 5.8 Compétence : Préparer et exécuter les plans de tests
- 5.9 Compétence : Préparer et documenter le déploiement
- 5.10 Compétence : Contribuer à la gestion d'un projet informatique
- 5.11 Compétence : Contribuer à la mise en production dans une démarche DevOps

6. Conclusion

- 6.1 Bilan du projet
- 6.2 Difficultés rencontrées et solutions

1. COMPÉTENCES MISES EN ŒUVRE

N° Fiche AT	Activités types	N° Fiche CP	Compétences professionnelles
1	Développer une application sécurisée	1	Installer et configurer son environnement de travail en fonction du projet
		2	Développer des interfaces utilisateur
		3	Développer des composants métier
		4	Contribuer à la gestion d'un projet informatique
2	Concevoir et développer une application sécurisée organisée en couches	5	Analyser les besoins et maquetter une application
		6	Définir l'architecture logicielle d'une application
		7	Concevoir et mettre en place une base de données relationnelle
		8	Développer des composants d'accès aux données SQL et NoSQL
3	Préparer le déploiement d'une application sécurisée	9	Préparer et exécuter les plans de tests d'une application
		10	Préparer et documenter le déploiement d'une application
		11	Contribuer à la mise en production dans une démarche DevOps

2. INTRODUCTION ET CONTEXTE

2.1 Présentation du projet

Le projet "Gestionnaire d'Emploi du Temps Intelligent" est une application web complète développée dans le cadre de la formation Concepteur Développeur d'Applications. Cette solution vise à automatiser et optimiser la génération d'emplois du temps pour les établissements d'enseignement en prenant en compte les contraintes multiples des professeurs, des salles et des matières.

2.2 Problématique

La gestion manuelle des emplois du temps dans les établissements éducatifs présente plusieurs défis majeurs qui rendent cette tâche particulièrement complexe et chronophage. La complexité des contraintes constitue le premier obstacle : chaque professeur dispose d'horaires de disponibilité spécifiques, les salles ont des capacités et des équipements particuliers, et la répartition des matières doit respecter un équilibre pédagogique. Cette multiplicité de contraintes rend l'optimisation particulièrement difficile, car il faut trouver une solution satisfaisante pour tous les acteurs impliqués dans le processus éducatif.

La flexibilité limitée des systèmes manuels pose également un problème majeur. Les changements de dernière minute, qu'il s'agisse d'absences de professeurs, de modifications de salles ou d'ajustements de programmes, nécessitent une refonte complète de l'emploi du temps. Enfin, le temps de traitement représente un défi considérable : la génération manuelle d'emplois du temps est non seulement chronophage mais également sujette aux erreurs, ce qui peut avoir des répercussions sur l'organisation pédagogique de l'établissement.

2.3 Objectifs du projet

L'objectif principal de ce projet est de développer une solution d'automatisation complète pour la génération d'emplois du temps optimisés. Cette automatisation permettra de réduire considérablement le temps de traitement tout en améliorant la qualité des résultats obtenus. La gestion des contraintes constitue un aspect fondamental du projet : le système doit être capable de prendre en compte les disponibilités spécifiques de chaque professeur, leurs préférences pédagogiques, ainsi que les contraintes matérielles liées aux salles et aux équipements disponibles.

L'interface utilisateur représente un enjeu majeur du projet. Elle doit être intuitive et moderne, permettant aux utilisateurs de naviguer facilement dans les différentes étapes du processus de génération d'emplois du temps. Cette accessibilité est essentielle pour garantir l'adoption du système par les équipes pédagogiques. La flexibilité du système constitue également un objectif important : l'application doit pouvoir s'adapter aux besoins spécifiques de chaque établissement, qu'il s'agisse d'écoles primaires, de collèges, de lycées ou d'établissements d'enseignement supérieur.

Enfin, la sécurité représente un aspect crucial du projet. Le système d'authentification et la gestion des droits utilisateurs garantissent que seules les personnes autorisées peuvent accéder aux fonctionnalités de génération et de modification des emplois du temps, protégeant ainsi l'intégrité des données pédagogiques.

3. CONCEPTION

3.1 Description du client

Profil de l'établissement :

Le lycée Général et Technologique de l'Emperi est un établissement public d'enseignement secondaire situé à Salon-de-Provence, dans les Bouches-du-Rhône. Avec plus de 1000 élèves inscrits chaque année, l'établissement propose des formations générales et technologiques du niveau Seconde à la Terminale. La direction est composée d'un proviseur, d'une proviseure adjointe et d'une équipe administrative dédiée à la gestion pédagogique.

Besoins spécifiques :

La proviseure adjointe, responsable de la coordination pédagogique, fait face à des défis majeurs dans la gestion des emplois du temps. La planification manuelle des emplois du temps représente un travail considérable nécessitant plusieurs semaines de préparation intensive. Les modifications fréquentes en cours d'année, dues aux absences, aux changements de programmes ou aux ajustements pédagogiques, nécessitent une réorganisation complète et créent des perturbations dans l'organisation de l'établissement.

Contraintes particulières :

L'établissement doit respecter strictement les programmes officiels de l'Éducation nationale, avec des volumes horaires spécifiques par matière et par niveau (Seconde, Première, Terminale). Certains enseignants ont des contraintes de disponibilité liées à leurs autres établissements d'affectation ou à leurs obligations personnelles. Les salles spécialisées (salles informatiques, laboratoires de sciences, ateliers technologiques, gymnase) ont des capacités et des équipements spécifiques qui limitent leur utilisation et nécessitent une planification rigoureuse.

3.2 Analyse de l'existant

Situation actuelle :

Actuellement, ma cliente utilise le logiciel fourni par l'Éducation nationale pour la gestion des emplois du temps. Bien que cet outil permette de générer un planning, il présente des limites importantes, notamment en ce qui concerne la prise en compte de critères essentiels comme l'attribution précise des professeurs à chaque classe.

Problèmes identifiés :

Dans la pratique, la chef adjointe doit constituer manuellement des listes de professeurs en fonction de leurs disponibilités et du volume horaire qu'ils peuvent assurer chaque semaine. Si cette méthode peut s'avérer suffisante pour de petits établissements, elle devient rapidement inadaptée dans un grand lycée, où l'équipe pédagogique compte plus de 80 enseignants.

L'absence d'une gestion fine et automatisée des affectations entraîne une sélection aléatoire des professeurs, ce qui nuit à la qualité des emplois du temps générés. Cette situation engendre non seulement une charge de

travail supplémentaire pour l'équipe de direction , mais aussi un risque accru de conflits d'horaires, de sous-utilisation des ressources et un manque d'optimisation certain.

Le projet que je propose vise à répondre à ces problématiques en offrant une solution capable de prendre en compte l'ensemble des contraintes et préférences, tout en automatisant et en fiabilisant le processus d'élaboration des emplois du temps, pour une organisation optimale au sein de l'établissement.

Limitations techniques :

L'absence d'automatisation rend le processus sujet aux erreurs humaines, avec des conflits de salles ou d'enseignants qui ne sont détectés qu'après la mise en place de l'emploi du temps. La difficulté à prendre en compte simultanément toutes les contraintes (disponibilités des enseignants, capacités des salles spécialisées, équilibre des matières selon les programmes officiels) conduit à des solutions sous-optimales qui impactent la qualité de l'enseignement.

Impact sur l'organisation :

Ces limitations ont un impact direct sur la qualité de l'enseignement et la satisfaction des équipes pédagogiques. Les enseignants expriment régulièrement leur frustration face aux emplois du temps déséquilibrés, aux créneaux horaires inappropriés pour certaines matières, ou aux conflits de salles qui perturbent leur enseignement. La proviseure adjointe doit consacrer un temps considérable à la résolution de ces problèmes au détriment d'autres missions pédagogiques importantes.

3.3 Expression des besoins

Gestion des utilisateurs et sécurité :

L'application doit permettre l'authentification sécurisée des utilisateurs avec différents niveaux d'accès (administrateur, proviseure adjointe, enseignants). Chaque utilisateur doit avoir accès aux fonctionnalités appropriées à son rôle dans l'établissement. Le système doit garantir la confidentialité des données et la traçabilité des modifications apportées aux emplois du temps, conformément aux exigences de l'Éducation nationale.

Import et gestion des données :

L'application doit permettre l'import des données des enseignants depuis des fichiers Excel fournis par l'établissement. Le système doit gérer les disponibilités spécifiques de chaque enseignant, les contraintes de salles spécialisées (laboratoires, ateliers, salles informatiques) et les programmes pédagogiques officiels par niveau (Seconde, Première, Terminale). La validation automatique des données importées est essentielle pour garantir la cohérence des informations et le respect des programmes officiels.

Génération automatique d'emplois du temps :

Le cœur de l'application réside dans l'algorithme de génération automatique qui doit prendre en compte simultanément toutes les contraintes : disponibilités des enseignants, capacités et équipements des salles spécialisées, volumes horaires par matière selon les programmes officiels, équilibre de la répartition des cours. L'algorithme doit optimiser la solution pour satisfaire au maximum les contraintes tout en respectant les règles pédagogiques et les spécificités du lycée général et technologique.

Visualisation et consultation :

L'interface doit permettre une consultation intuitive des emplois du temps générés, avec des vues par classe, par enseignant ou par salle. Les utilisateurs doivent pouvoir effectuer des modifications manuelles si

nécessaire, avec validation automatique des contraintes. L'export des emplois du temps en différents formats (PDF, Excel) est requis pour la communication avec les équipes pédagogiques et la diffusion aux élèves et aux familles via les cahiers de textes numériques.

Performance et fiabilité :

L'application doit générer un emploi du temps complet en moins de 30 secondes pour un lycée de plus de 1000 élèves. Le système doit supporter la gestion simultanée de 80+ enseignants et 30+ classes (Seconde, Première, Terminale) sans dégradation des performances. La fiabilité est critique avec une disponibilité de 99% pendant les heures de travail de l'établissement.

Sécurité et intégrité des données :

L'authentification JWT assure la sécurité des accès, tandis que la validation des données d'entrée et la protection contre les injections SQL garantissent l'intégrité des informations. La sauvegarde automatique des configurations et la récupération en cas d'échec sont essentielles pour la continuité de service, particulièrement importante pour un établissement de cette taille.

Maintenabilité et évolutivité :

L'architecture modulaire et le code documenté facilitent la maintenance et l'évolution de l'application. Les tests automatisés garantissent la qualité du code et facilitent les mises à jour futures. L'extensibilité du système permet d'ajouter de nouvelles fonctionnalités selon les besoins évolutifs du lycée de l'Emperi et des évolutions des programmes officiels.

3.4 Limites du projet

Contraintes techniques :

Le développement est limité à l'utilisation de Java 17 et Spring Boot pour le backend, React pour le frontend, et PostgreSQL pour la base de données. Ces choix technologiques sont dictés par les compétences de l'équipe de développement et la nécessité d'utiliser des technologies éprouvées et maintenues.

Contraintes fonctionnelles :

L'algorithme de génération doit respecter strictement les contraintes horaires des enseignants et éviter les conflits de salles spécialisées. La répartition des matières doit être optimisée selon les programmes officiels de l'Éducation nationale, et les contraintes de niveau (Seconde, Première, Terminale) doivent être prises en compte dans la génération. Les spécificités du lycée général et technologique (options, spécialités, enseignements technologiques) doivent être intégrées dans l'algorithme.

Contraintes temporelles :

Le projet doit être développé en 8 semaines, incluant les phases de conception, développement, tests et documentation. Cette contrainte temporelle limite la complexité des fonctionnalités et nécessite une priorisation rigoureuse des besoins, en se concentrant sur les fonctionnalités essentielles pour la proviseure adjointe du lycée de l'Emperi.

Périmètre fonctionnel :

Le projet se concentre exclusivement sur la génération d'emplois du temps. Les autres fonctionnalités liées à la vie scolaire (gestion des absences, notifications, intégration avec les systèmes existants) ne font pas partie

du scope de la demande initiale de la cliente et pourront être développées ultérieurement selon les besoins évolutifs de l'établissement.

Limites matérielles :

L'application est conçue spécifiquement pour le lycée Général et Technologique de l'Emperi et ne prévoit pas initialement la gestion multi-établissements. L'interface est optimisée pour les écrans d'ordinateur et de tablette utilisés par la direction et les enseignants, avec une adaptation limitée pour les smartphones.

4. ENVIRONNEMENT TECHNIQUE

4.1 Stack technologique

Backend - Java/Spring Boot

Le backend de l'application est développé en Java 17, un langage de programmation mature et robuste qui offre d'excellentes performances et une grande stabilité pour les applications d'entreprise. Spring Boot 3.3.1 constitue le framework de développement principal, fournissant une base solide pour la création d'applications web modernes avec une configuration simplifiée et des fonctionnalités intégrées. Spring Data JPA gère la persistance des données en offrant une couche d'abstraction puissante qui simplifie l'accès aux bases de données relationnelles. Spring Security assure la sécurité et l'authentification de l'application en fournissant un framework complet pour la gestion des autorisations et la protection des ressources. JWT (JSON Web Tokens) est utilisé pour la gestion des sessions, permettant une authentification stateless et scalable. Enfin, Maven gère les dépendances du projet, assurant la reproductibilité des builds et facilitant la gestion des versions des bibliothèques utilisées.

Frontend - React

Le frontend de l'application utilise React 19.1.0, un framework JavaScript moderne qui permet de créer des interfaces utilisateur interactives et performantes. React Hooks constitue le système de gestion d'état principal, offrant une approche fonctionnelle et intuitive pour gérer l'état local et global des composants. La bibliothèque XLSX est intégrée pour permettre le parsing des fichiers Excel, une fonctionnalité essentielle pour l'import des données des professeurs depuis les fichiers fournis par les établissements scolaires. Enfin, npm gère les dépendances frontend, assurant une gestion cohérente des packages JavaScript et facilitant l'installation et la mise à jour des bibliothèques utilisées.

Base de données

PostgreSQL constitue la base de données principale utilisée en production, offrant une robustesse et des performances exceptionnelles pour les applications d'entreprise. Cette base de données relationnelle open-source assure la fiabilité des données et supporte les fonctionnalités avancées nécessaires au projet, comme les transactions ACID et les contraintes d'intégrité. H2 est utilisé comme base de données embarquée pour le développement, permettant une configuration rapide et simplifiée sans nécessiter l'installation d'un serveur de base de données séparé. JPA/Hibernate sert d'ORM (Object-Relational Mapping) pour la persistance, fournissant une couche d'abstraction qui simplifie l'accès aux données et permet une gestion efficace des relations entre entités.

Outils de développement

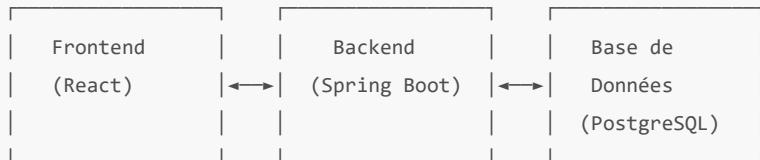
IntelliJ IDEA constitue l'IDE principal utilisé pour le développement Java, offrant des fonctionnalités avancées comme l'autocomplétion intelligente, le refactoring automatique et l'intégration native avec les frameworks Spring. Git sert d'outil de versioning pour gérer l'historique des modifications du code, facilitant la collaboration en équipe et permettant de revenir facilement à des versions précédentes en cas de problème. Postman est utilisé pour les tests d'API, permettant de valider le bon fonctionnement des endpoints REST et de documenter les interactions avec le backend. Enfin, pgAdmin fournit une interface graphique pour

l'administration de PostgreSQL, facilitant la gestion des bases de données, la création de sauvegardes et le monitoring des performances.

4.2 Architecture du système

4.2.1 Architecture générale

L'application suit une architecture en couches avec séparation claire des responsabilités entre le frontend et le backend. Cette approche permet une maintenance facilitée et une évolution indépendante des composants.



Description des couches :

Frontend (React) :

- **Responsabilité** : Interface utilisateur et gestion de l'état client
- **Technologies** : React 19.1.0, Hooks, XLSX pour le parsing Excel
- **Communication** : API REST avec le backend via HTTP/JSON
- **Fonctionnalités** : Upload de fichiers, édition interactive, visualisation des emplois du temps

Backend (Spring Boot) :

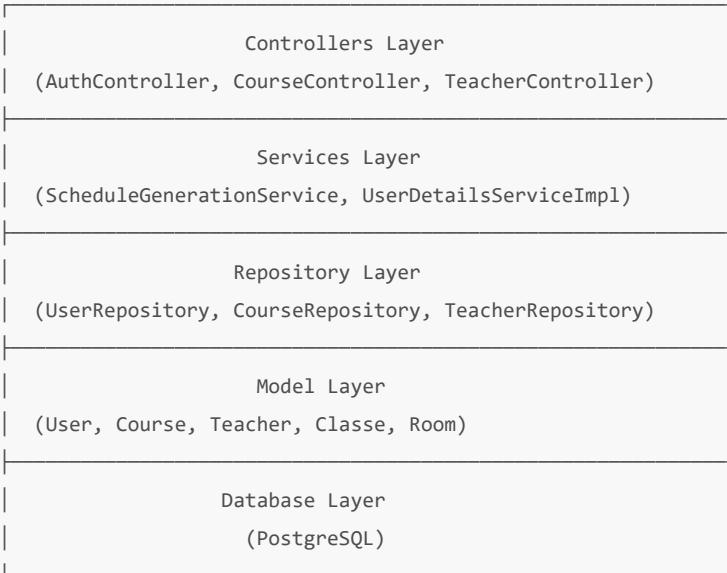
- **Responsabilité** : Logique métier, traitement des données, sécurité
- **Technologies** : Java 17, Spring Boot 3.3.1, Spring Security, JWT
- **Communication** : API REST, persistance des données
- **Fonctionnalités** : Génération d'emplois du temps, authentification, gestion des contraintes

Base de données (PostgreSQL) :

- **Responsabilité** : Stockage persistant des données
- **Technologies** : PostgreSQL 15, JPA/Hibernate
- **Fonctionnalités** : Entités métier, relations, contraintes d'intégrité

4.2.2 Architecture en couches (Backend)

Le backend suit le pattern architectural en couches, garantissant une séparation claire des responsabilités et facilitant la maintenance et les tests.



Description détaillée des couches :

1. Controllers Layer (Couche Présentation) :

- Responsabilité** : Gestion des requêtes HTTP et réponses REST
- Composants** : AuthController, CourseController, TeacherController, ClassConfigController
- Fonctionnalités** : Validation des données d'entrée, sérialisation JSON, gestion des erreurs HTTP
- Pattern** : REST API avec endpoints standardisés

2. Services Layer (Couche Métier) :

- Responsabilité** : Logique métier et algorithmes complexes
- Composants** : ScheduleGenerationService, UserDetailsServiceImpl
- Fonctionnalités** : Génération d'emplois du temps, authentification, gestion des contraintes
- Pattern** : Service Layer avec injection de dépendances

3. Repository Layer (Couche Données) :

- Responsabilité** : Accès aux données et persistance
- Composants** : UserRepository, CourseRepository, TeacherRepository, ClasseRepository
- Fonctionnalités** : Requêtes personnalisées, gestion des relations, optimisation des performances
- Pattern** : Repository Pattern avec Spring Data JPA

4. Model Layer (Couche Entités) :

- Responsabilité** : Représentation des données métier
- Composants** : User, Course, Teacher, Classe, Room, EmploiDuTempsConfig
- Fonctionnalités** : Mapping ORM, validation des entités, relations JPA
- Pattern** : Entity Pattern avec annotations JPA

5. Database Layer (Couche Base de données) :

- Responsabilité** : Stockage persistant et gestion des transactions
- Technologie** : PostgreSQL avec Hibernate
- Fonctionnalités** : Contraintes d'intégrité, index, transactions ACID
- Pattern** : Database Layer avec gestion des connexions

4.3 Configuration de l'environnement

4.3.1 Configuration Maven (`pom.xml`)

Le fichier `pom.xml` définit les dépendances principales du projet et constitue le cœur de la configuration Maven. Spring Boot 3.3.1 est utilisé comme version parent, offrant les dernières fonctionnalités et optimisations. Les Spring Boot Starters, incluant Web, Data JPA et Security, fournissent les fonctionnalités de base nécessaires au développement d'une application web complète avec persistance des données et sécurité intégrée. La configuration des bases de données est gérée de manière flexible, avec PostgreSQL pour la production et H2 pour le développement, permettant une transition transparente entre les environnements. JWT version 0.11.5 est configuré pour la gestion des tokens d'authentification, assurant une sécurité robuste et scalable pour l'application. Lombok est intégré pour réduire le code boilerplate et améliorer la lisibilité. Enfin, Java 17 est spécifié comme version du langage de programmation, garantissant l'utilisation des fonctionnalités modernes du langage tout en assurant la compatibilité avec les outils de développement.

4.3.2 Configuration React (`package.json`)

Le fichier `package.json` gère les dépendances frontend et constitue le point central de configuration du projet React. React 19.1.0 est défini comme framework JavaScript principal, offrant les dernières fonctionnalités et optimisations pour le développement d'interfaces utilisateur modernes. React Scripts 5.0.1 est utilisé pour la configuration et le build automatique. La bibliothèque XLSX version 0.18.5 est intégrée pour permettre le parsing des fichiers Excel, une fonctionnalité critique qui permet aux utilisateurs d'importer directement les données des professeurs depuis les fichiers fournis par les établissements scolaires. Les bibliothèques de test (@testing-library/react, @testing-library/dom, @testing-library/jest-dom, @testing-library/user-event) sont configurées pour assurer la qualité du code. Les scripts npm sont configurés pour automatiser les tâches de développement, build et tests, facilitant ainsi le workflow de développement et assurant la cohérence des processus de build entre les différents environnements.

4.4 Architecture Frontend (React)

4.4.1 Architecture Frontend (React)

L'interface utilisateur suit une architecture modulaire basée sur les composants React, avec une gestion d'état centralisée et une séparation claire des responsabilités.

Structure des composants :

```
App.js (Composant racine)
├── ConfigurationManager (Étape 1)
├── ProfesseurImporter (Étape 2)
├── ClassManager (Étape 3)
└── ScheduleGenerator (Étape 4)
    ├── ScheduleDisplay
    ├── ScheduleGrid
    └── ExportOptions
```

Patterns architecturaux utilisés :

1. Component-Based Architecture :

La Component-Based Architecture constitue le fondement de l'interface React, permettant de créer des composants réutilisables et modulaires. Cette approche facilite la maintenance du code en isolant les responsabilités et en permettant la réutilisation de composants communs. Les tests unitaires sont simplifiés grâce à cette modularité, permettant de valider chaque composant individuellement. Des exemples concrets de cette architecture incluent ProfesseurImporter pour la gestion des imports, ClassManager pour la configuration des classes, et ScheduleDisplay pour l'affichage des emplois du temps.

2. State Management avec React Hooks :

La gestion d'état avec React Hooks représente une approche moderne et efficace pour gérer l'état local et global de l'application. Les hooks utilisés, notamment useState pour l'état local, useEffect pour les effets de bord, et useContext pour l'état global, permettent une gestion centralisée et réactive de l'état. Cette approche optimise les performances en évitant les re-rendus inutiles et assure une réactivité immédiate aux changements d'état.

3. Props Drilling Pattern :

Le Props Drilling Pattern gère le passage des données entre composants de manière prévisible et contrôlée. Ce mécanisme utilise les props pour transmettre les données et les callbacks pour gérer les événements, créant un flux de données unidirectionnel qui facilite le débogage et la compréhension du code. Cette approche garantit que les données circulent de manière cohérente dans l'application.

4. Event-Driven Architecture :

L'Event-Driven Architecture gère les interactions utilisateur de manière efficace et réactive. Ce mécanisme utilise des callbacks et des événements personnalisés pour répondre aux actions de l'utilisateur, comme l'upload de fichiers Excel, l'édition interactive des disponibilités des professeurs, et la génération d'emplois du temps. Cette approche assure une expérience utilisateur fluide et réactive.

5. Service Layer Pattern :

Le Service Layer Pattern assure la communication avec l'API backend de manière organisée et maintenable. Cette couche inclut les services API, les parsers Excel, et les utilitaires, permettant une séparation claire des préoccupations et une réutilisabilité optimale des services. Cette architecture facilite la maintenance et l'évolution de l'application.

4.5 Patterns de communication et intégration

4.5.1 Patterns de communication et intégration

L'architecture de communication entre le frontend et le backend suit des patterns éprouvés pour assurer la robustesse et la maintenabilité du système.

1. REST API Pattern :

Le REST API Pattern constitue le fondement de la communication entre le frontend et le backend, utilisant le protocole HTTP/HTTPS avec des méthodes standardisées (GET, POST, PUT, DELETE) pour assurer une interface uniforme et prévisible. Le format JSON est utilisé pour l'échange de données, offrant une sérialisation efficace et une compatibilité universelle. Cette approche présente plusieurs avantages majeurs : elle est stateless, permettant une meilleure scalabilité, scalable pour supporter une croissance de l'application, cacheable pour optimiser les performances, et uniforme pour faciliter l'intégration. Les endpoints principaux incluent `/api/auth/*` pour l'authentification, `/api/profs/*` pour la gestion des professeurs, `/api/classes/*` pour la gestion des classes, et `/api/schedule/*` pour la génération d'emplois du temps.

2. DTO Pattern (Data Transfer Objects) :

Le DTO Pattern assure le transfert de données entre les différentes couches de l'application de manière structurée et sécurisée. Cette approche offre plusieurs avantages significatifs : la séparation des modèles permet d'isoler les préoccupations, la validation assure l'intégrité des données, et le versioning facilite l'évolution de l'API. Des exemples concrets de cette implémentation incluent LoginDto pour les données de connexion, RegisterDto pour l'inscription des utilisateurs, ClassScheduleDto pour les emplois du temps, et ProfConfigDto pour la configuration des professeurs.

3. JWT Authentication Pattern :

Le JWT Authentication Pattern utilise des tokens stateless pour l'authentification, offrant une approche moderne et efficace pour la gestion des sessions utilisateur. Cette méthode présente des avantages considérables en termes de scalabilité, permettant à l'application de gérer un nombre croissant d'utilisateurs sans dégradation des performances, de performance grâce à la réduction des requêtes de base de données, et de sécurité par l'utilisation de tokens signés et chiffrés. L'implémentation comprend JwtUtils pour la génération et validation des tokens, JwtRequestFilter pour l'interception des requêtes, et SecurityConfig pour la configuration globale de la sécurité.

4. Error Handling Pattern :

Le Error Handling Pattern assure une gestion centralisée et cohérente des erreurs dans l'application. Ce mécanisme utilise ResponseEntity avec des codes HTTP appropriés pour fournir des réponses d'erreur standardisées et informatives. Cette approche présente plusieurs avantages : la cohérence assure une expérience utilisateur uniforme, le débogage facilité permet aux développeurs d'identifier rapidement les problèmes, et l'expérience utilisateur améliorée garantit que les utilisateurs reçoivent des messages d'erreur clairs et utiles.

4.6 Organisation des dossiers

4.6.1 Organisation des dossiers

L'architecture du projet suit une organisation modulaire claire, séparant les responsabilités et facilitant la maintenance.

```

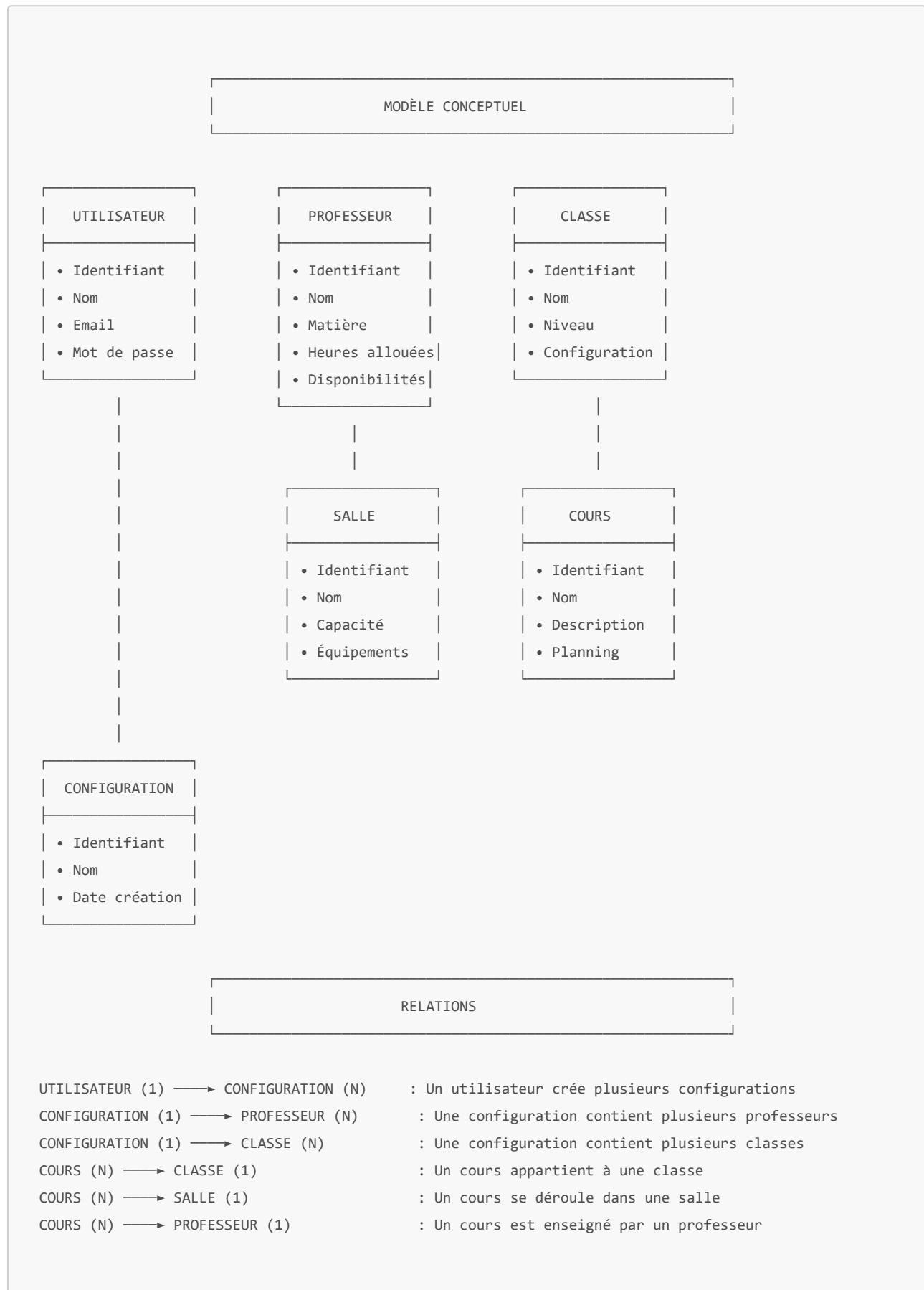
cda/
└── backend                         # Application Spring Boot
    ├── src/main/java/com/example/cda/
    │   ├── config/                   # Configuration
    │   │   ├── SecurityConfig.java  # Configuration sécurité
    │   │   └── WebConfig.java       # Configuration web
    │   ├── controller/              # Contrôleurs REST
    │   │   ├── AuthController.java  # Authentification
    │   │   ├── CourseController.java # Gestion des cours
    │   │   └── TeacherController.java # Gestion des professeurs
    │   ├── dto/                      # Objets de transfert
    │   │   ├── LoginDto.java        # Données de connexion
    │   │   └── ClassScheduleDto.java # Emploi du temps
    │   ├── model/                  # Entités JPA
    │   │   ├── User.java           # Utilisateurs
    │   │   ├── Classe.java         # Classes
    │   │   └── Teacher.java        # Professeurs
    │   ├── repository/             # Couche d'accès aux données
    │   │   ├── UserRepository.java # Repository utilisateurs
    │   │   └── ClasseRepository.java # Repository classes
    │   ├── security/               # Sécurité et JWT
    │   │   ├── JwtUtils.java       # Utilitaires JWT
    │   │   └── JwtRequestFilter.java # Filtre JWT
    │   └── service/                # Logique métier
        ├── ScheduleGenerationService.java # Génération EDT
        └── UserDetailsServiceImpl.java # Authentification
    ├── src/main/resources/
    │   └── application.properties     # Configuration
    └── pom.xml                        # Dépendances Maven
└── frontend                         # Application React
    └── src/
        ├── components/            # Composants React
        │   ├── ConfigurationManager.jsx # Étape 1
        │   ├── ProfesseurImporter.jsx # Étape 2
        │   ├── ClasseManager.jsx     # Étape 3
        │   └── ScheduleGenerator.jsx # Étape 4
        ├── App.js                  # Composant principal
        └── index.js                 # Point d'entrée
    └── package.json                  # Dépendances npm
└── excel_samples/                  # Fichiers d'exemple
    ├── Profs_voeux.xlsx            # Exemple professeurs
    └── Profs_voeux(1).xlsx          # Exemple format alternatif

```

4.7 Modèle de données

4.7.1 Modèle conceptuel (MCD)

Le modèle conceptuel représente les entités métier et leurs relations sans considération technique :



Relations identifiées :

- **UTILISATEUR ↔ CONFIGURATION** : Un utilisateur peut créer plusieurs configurations
- **PROFESSEUR ↔ CONFIGURATION** : Un professeur appartient à une configuration
- **CLASSE ↔ CONFIGURATION** : Une classe appartient à une configuration
- **COURS ↔ CLASSE** : Un cours est associé à une classe
- **COURS ↔ SALLE** : Un cours se déroule dans une salle
- **COURS ↔ PROFESSEUR** : Un cours est enseigné par un professeur

4.7.2 Modèle logique (MLD)

Le modèle logique ajoute les contraintes d'intégrité et les clés :

```
| UTILISATEUR (id_utilisateur, nom, email, mot_de_passe)
```

```
| | Clé primaire : id_utilisateur
```

```
| | Contraintes : email unique, mot_de_passe non nul
```

```
| PROFESSEUR (id_professeur, nom, matière, heures_allouées, disponibilités,  
| | id_configuration)
```

```
| | Clé primaire : id_professeur
```

```
| | Clé étrangère : id_configuration → CONFIGURATION(id_configuration)
```

```
| | Contraintes : nom non nul, matière non nul
```

```
| CLASSE (id_classe, nom, niveau, id_configuration)
```

```
| | Clé primaire : id_classe
```

```
| | Clé étrangère : id_configuration → CONFIGURATION(id_configuration)
```

```
| | Contraintes : nom unique par configuration
```

```
| SALLE (id_salle, nom, capacite, equipements)
```

```
| | Clé primaire : id_salle
```

```
| | Contraintes : capacite > 0, nom unique
```

```
| COURS (id_cours, nom, description, planning, id_classe, id_salle,  
| | id_professeur)
```

```
| | Clé primaire : id_cours
```

```
| | Clés étrangères :
```

```
| | | • id_classe → CLASSE(id_classe)
```

```
| | | • id_salle → SALLE(id_salle)
```

```
| | | • id_professeur → PROFESSEUR(id_professeur)
```

```
| | Contraintes : planning non nul
```

```
| CONFIGURATION (id_configuration, nom, date_creation, id_utilisateur)
```

```
| | Clé primaire : id_configuration
```

```
| | Clé étrangère : id_utilisateur → UTILISATEUR(id_utilisateur)
```

```
| | Contraintes : nom unique par utilisateur
```

4.7.3 Modèle physique (MPD)

Le modèle physique représente la structure réelle en base de données PostgreSQL :

TABLES POSTGRESQL

```
| users
```

```
| id BIGSERIAL PRIMARY KEY
| username VARCHAR(50) UNIQUE NOT NULL
| email VARCHAR(100) UNIQUE NOT NULL
| password VARCHAR(255) NOT NULL
| created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
| schedule_configs
```

```
| id BIGSERIAL PRIMARY KEY
| name VARCHAR(100) NOT NULL
| creation_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
| user_id BIGINT REFERENCES users(id) ON DELETE CASCADE
```

```
| teachers
```

```
| id BIGSERIAL PRIMARY KEY
| name VARCHAR(100) NOT NULL
| subject VARCHAR(100) NOT NULL
| hours_allocated VARCHAR(50)
| schedule TEXT
| config_id BIGINT REFERENCES schedule_configs(id) ON DELETE CASCADE
```

```
| classes
|-
| id BIGSERIAL PRIMARY KEY
| name VARCHAR(100) NOT NULL
| level VARCHAR(50)
| config_id BIGINT REFERENCES schedule_configs(id) ON DELETE CASCADE
```

```
| rooms
|-
| id BIGSERIAL PRIMARY KEY
| name VARCHAR(100) UNIQUE NOT NULL
| capacity INTEGER CHECK (capacity > 0)
| equipment TEXT
```

```
| courses
|-
| id BIGSERIAL PRIMARY KEY
| name VARCHAR(100) NOT NULL
| description TEXT
| planning TEXT
| class_id BIGINT REFERENCES classes(id) ON DELETE CASCADE
| room_id BIGINT REFERENCES rooms(id) ON DELETE CASCADE
| teacher_id BIGINT REFERENCES teachers(id) ON DELETE CASCADE
```

INDEX D'OPTIMISATION

- idx_teachers_config ON teachers(config_id)
- idx_classes_config ON classes(config_id)
- idx_courses_class ON courses(class_id)
- idx_courses_room ON courses(room_id)
- idx_courses_teacher ON courses(teacher_id)

CARACTÉRISTIQUES TECHNIQUES

- Moteur de base : PostgreSQL 15
- Encodage : UTF-8
- Contraintes : Clés étrangères avec CASCADE
- Index : Optimisation des requêtes fréquentes
- Types de données : BIGSERIAL pour les IDs, TEXT pour les JSON

4.7.4 Entités principales

SchoolClass (School class) - SchoolClass.java

JPA entity modeling student classes with their subjects and schedule configuration

```

@Entity
@Data
@NoArgsConstructor
public class SchoolClass {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String level;

    @ManyToOne
    @JoinColumn(name = "config_id")
    private ScheduleConfig config;

    @OneToMany(mappedBy = "schoolClass", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<SubjectClass> subjects;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getLevel() { return level; }
    public void setLevel(String level) { this.level = level; }
    public ScheduleConfig getConfig() { return config; }
    public void setConfig(ScheduleConfig config) { this.config = config; }
    public List<SubjectClass> getSubjects() { return subjects; }
    public void setSubjects(List<SubjectClass> subjects) { this.subjects = subjects; }
}

```

Detailed description of the SchoolClass entity :

- **@ManyToOne** : Relation to ScheduleConfig (multiple classes per configuration)
- **@JoinColumn(name = "config_id")** : Foreign key to configuration
- **@OneToMany(mappedBy = "schoolClass")** : Inverse relation to subjects
- **cascade = CascadeType.ALL** : Propagation of operations to subjects
- **orphanRemoval = true** : Automatic removal of orphaned subjects

Teacher (Teacher) - Teacher.java

JPA entity storing teacher information with their availability in JSON

```

@Entity
@Data
@NoArgsConstructor
public class Teacher {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String subject;
    private String hoursAllocated;

    @Lob
    private String schedule; // Stored as JSON

    @ManyToOne
    @JoinColumn(name = "config_id")
    private ScheduleConfig config;

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getSubject() { return subject; }
    public void setSubject(String subject) { this.subject = subject; }
    public String getHoursAllocated() { return hoursAllocated; }
    public void setHoursAllocated(String hoursAllocated) { this.hoursAllocated = hoursAllocated; }
    public String getSchedule() { return schedule; }
    public void setSchedule(String schedule) { this.schedule = schedule; }
    public ScheduleConfig getConfig() { return config; }
    public void setConfig(ScheduleConfig config) { this.config = config; }
}

```

Detailed description of the Teacher entity :

- **@Lob** : Large Object to store schedule in JSON
- **JSON storage** : Schedule is serialized as JSON string
- **hoursAllocated** : Number of hours allocated to the teacher
- **ManyToOne relation** : Each teacher belongs to a configuration

ScheduleConfig (Schedule configuration) - ScheduleConfig.java

Root JPA entity aggregating classes and teachers for a schedule configuration

```

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

    private String name;
    private LocalDateTime creationDate = LocalDateTime.now();

    @OneToMany(mappedBy = "config", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Class> classes;

    @OneToMany(mappedBy = "config", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<TeacherConfig> teachers;
}

```

Detailed description of the ScheduleConfig entity :

- **creationDate** : Automatic creation timestamp
- **OneToMany relations** : To classes and teachers with cascade and orphan removal
- **Root entity** : This entity is the root of the configuration aggregate
- **Collection management** : Lists are managed with cascade to maintain consistency

4.8 Sécurité et authentification

4.8.1 Configuration Spring Security

```

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/auth/**").permitAll()
                .anyRequest().authenticated()
            )
            .sessionManagement(session -> session
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            )
            .addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

```

4.8.2 Gestion JWT

Classe JwtUtils - Utilitaires de gestion des tokens JWT

```

@Component
public class JwtUtils {

    @Value("${jwt.secret}")
    private String secret;

    public String generateToken(UserDetails userDetails) {
        return Jwts.builder()
            .setSubject(userDetails.getUsername())
            .setIssuedAt(new Date())
            .setExpiration(new Date(System.currentTimeMillis() + 86400000))
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }

    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = extractUsername(token);
        return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
    }
}

```

Description détaillée de l'implémentation JWT :

- **Annotation @Component** : Marque la classe comme composant Spring pour l'injection de dépendances
- **@Value("\${jwt.secret}")** : Injection de la clé secrète depuis application.properties
- **generateToken()** :
 - Crée un token JWT avec le username comme sujet
 - Définit la date de création et d'expiration (24h)
 - Signe avec l'algorithme HS512 pour la sécurité
- **validateToken()** :
 - Extrait le username du token
 - Vérifie la correspondance avec l'utilisateur
 - Contrôle que le token n'est pas expiré
- **Sécurité** : Utilisation d'une clé secrète de 64 caractères minimum pour HS512

4.9 API REST

L'API REST constitue l'interface de communication entre le frontend React et le backend Spring Boot. Elle suit les principes REST (Representational State Transfer) pour assurer une architecture web standardisée et évolutive.

4.9.1 Architecture REST mise en place

Principe de conception

- **Stateless** : Chaque requête contient toutes les informations nécessaires
- **Ressources** : Les endpoints représentent des ressources métier (professeurs, classes, emplois du temps)
- **Verbes HTTP** : Utilisation appropriée de GET, POST, PUT, DELETE
- **Codes de statut** : Réponses HTTP standardisées (200, 201, 400, 401, 404, 500)

4.9.2 Endpoints principaux

Authentification et sécurité

```
POST /auth/login      # Connexion utilisateur avec JWT
POST /auth/register  # Inscription d'un nouvel utilisateur
GET  /auth/validate   # Validation d'un token JWT
```

Gestion des professeurs

```
GET  /api/getprofs    # Récupération de tous les professeurs
POST /api/addprof     # Création d'un nouveau professeur
PUT   /api/profs/{id}  # Modification d'un professeur existant
DELETE /api/profs/{id} # Suppression d'un professeur
GET   /api/profs/{id}  # Récupération d'un professeur spécifique
```

Gestion des classes

```
GET /api/classes          # Liste de toutes les classes
POST /api/classes         # Création d'une nouvelle classe
PUT  /api/classes/{id}    # Modification d'une classe
DELETE /api/classes/{id}  # Suppression d'une classe
GET  /api/classes/{id}    # Détails d'une classe spécifique
```

Génération d'emplois du temps

```
GET /api/schedule        # Génération automatique de l'emploi du temps
POST /api/schedule       # Génération avec paramètres personnalisés
GET  /api/schedule/{id}  # Récupération d'un emploi du temps spécifique
```

Configuration du système

```
POST /api/config          # Sauvegarde d'une configuration
GET  /api/config          # Récupération de la configuration active
GET  /api/config/{id}     # Récupération d'une configuration spécifique
```

4.9.3 Implémentation des contrôleurs REST

Architecture des contrôleurs

Les contrôleurs REST suivent le pattern MVC (Model-View-Controller) où ils représentent la couche "Controller". Chaque contrôleur est responsable d'une ressource métier spécifique et expose les opérations CRUD (Create, Read, Update, Delete) via des endpoints HTTP.

Contrôleur d'authentification - AuthController.java

Contrôleur REST gérant l'authentification et l'inscription des utilisateurs avec JWT

```

@RestController
@RequestMapping("/api/auth")
public class AuthController {
    private final AuthenticationManager authenticationManager;
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtUtils jwtUtils;

    @Autowired
    public AuthController(AuthenticationManager authenticationManager, UserRepository userRepository,
    PasswordEncoder passwordEncoder, JwtUtils jwtUtils) {
        this.authenticationManager = authenticationManager;
        this.userRepository = userRepository;
        this.passwordEncoder = passwordEncoder;
        this.jwtUtils = jwtUtils;
    }

    @PostMapping("/login")
    public ResponseEntity<?> authenticateUser(@RequestBody LoginDto loginDto) {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(loginDto.getUsername(),
            loginDto.getPassword()));

        SecurityContextHolder.getContext().setAuthentication(authentication);
        final UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        final String jwt = jwtUtils.generateToken(userDetails);

        return ResponseEntity.ok(new JwtResponse(jwt));
    }

    @PostMapping("/register")
    public ResponseEntity<?> registerUser(@RequestBody RegisterDto registerDto) {
        if (userRepository.findByUsername(registerDto.getUsername()).isPresent()) {
            return new ResponseEntity<>("Username is already taken!", HttpStatus.BAD_REQUEST);
        }

        // Create new user's account
        User user = new User();
        user.setUsername(registerDto.getUsername());
        user.setEmail(registerDto.getEmail());
        user.setPassword(passwordEncoder.encode(registerDto.getPassword()));

        userRepository.save(user);

        return new ResponseEntity<>("User registered successfully", HttpStatus.OK);
    }

    @GetMapping("/health")
    public ResponseEntity<String> health() {
        return ResponseEntity.ok("OK");
    }
}

```

```
    }  
}
```

Description détaillée du contrôleur d'authentification :

- **Injection de dépendances** : Constructeur avec `@Autowired` pour les services requis
- **Endpoint /login** :
 - Authentification via `AuthenticationManager` Spring Security
 - Génération du token JWT avec `JwtUtils`
 - Retour du token dans `JwtResponse`
- **Endpoint /register** :
 - Vérification de l'unicité du `username`
 - Encodage du mot de passe avec `PasswordEncoder`
 - Sauvegarde de l'utilisateur en base
- **Gestion des erreurs** : `ResponseEntity` avec codes HTTP appropriés

Teacher management controller - TeacherController.java

REST controller for teacher import and management with flexible JSON parsing

```

@RestController
@RequestMapping("/api/teachers")
public class TeacherController {
    private static final Logger logger = LoggerFactory.getLogger(TeacherController.class);

    @Autowired
    private TeacherRepository teacherRepository;
    @Autowired
    private ObjectMapper objectMapper;

    @PostMapping
    public ResponseEntity<?> importTeachers(@RequestBody JsonNode teachersNode) {
        logger.info("Direct import of teachers (flexible format): {}", teachersNode);
        if (!teachersNode.isArray()) {
            return ResponseEntity.badRequest().body("Payload must be an array of teachers");
        }
        for (JsonNode node : teachersNode) {
            Teacher teacher = new Teacher();
            teacher.setName(node.hasNonNull("name") ? node.get("name").asText() : node.path("teacher").asText());
            teacher.setSubject(node.path("subject").asText(null));
            teacher.setHoursAllocated(node.path("allocated").asText(null));

            try {
                if (node.has("planning") && !node.get("planning").isNull()) {
                    teacher.setSchedule(objectMapper.writeValueAsString(node.get("planning")));
                } else if (node.has("schedule") && !node.get("schedule").isNull()) {
                    teacher.setSchedule(objectMapper.writeValueAsString(node.get("schedule")));
                }
            } catch (Exception e) {
                logger.error("Schedule serialization error for {}: {}", teacher.getName(), e.getMessage());
            }
            teacherRepository.save(teacher);
        }
        return ResponseEntity.ok("Teacher list imported successfully");
    }
}

```

Detailed description of the teacher controller :

- **SLF4J Logger** : Structured logging for debugging and monitoring
- **POST /api/teachers endpoint** : Flexible import of teachers from frontend
- **Flexible JSON parsing** : Accepts different data formats (name/teacher, planning/schedule)
- **ObjectMapper** : JSON serialization/deserialization for schedule
- **Error handling** : Try-catch for JSON serialization with logging
- **Validation** : Verification that payload is an array before processing

4.9.4 Gestion des erreurs et validation

Codes de statut HTTP utilisés

- **200 OK** : Requête réussie
- **201 Created** : Ressource créée avec succès
- **400 Bad Request** : Données invalides ou manquantes
- **401 Unauthorized** : Authentification requise
- **403 Forbidden** : Accès interdit
- **404 Not Found** : Ressource non trouvée
- **500 Internal Server Error** : Erreur serveur

Validation des données d'entrée

La validation s'effectue à deux niveaux :

- **Validation automatique** : Utilisation des annotations JPA (@NotNull, @Size, etc.) et Spring Validation
- **Validation métier** : Vérifications spécifiques au domaine (unicité, règles métier, etc.)

4.9.5 Sécurité des endpoints

Protection par JWT

Le système utilise un filtre JWT personnalisé qui :

- Extrait le token JWT depuis le header Authorization
- Valide le token et l'utilisateur associé
- Crée le contexte d'authentification Spring Security
- Protège tous les endpoints sensibles de l'application

Cette architecture REST assure une séparation claire des responsabilités, une gestion robuste des erreurs, et une sécurité renforcée pour l'ensemble de l'application.

5. RÉALISATIONS ET MISE EN ŒUVRE DES COMPÉTENCES

5.1 Compétence : Analyser les besoins et maquetter une application

5.1.1 Analyse des besoins réalisée

Analyse fonctionnelle

- Étude des processus métier de gestion d'emplois du temps
- Identification des acteurs (administrateurs, professeurs, direction)
- Définition des cas d'usage principaux et secondaires
- Modélisation des flux de données

Analyse technique

- Évaluation des contraintes techniques (performance, sécurité)
- Analyse des besoins en stockage et traitement
- Définition des interfaces utilisateur
- Planification de l'architecture système

5.1.2 Maquettage de l'application

Maquettes d'interface utilisateur

Étape 1 : Configuration initiale

GÉNÉRATEUR D'EMPLOI DU TEMPS

[1] Configuration [2] Import Profs [3] Classes [4] Génération

CONFIGURATION INITIALE

Nom de la configuration : [_____]

Paramètres temporels :

Jours de cours : [] Lundi [] Mardi [] Mercredi
[] Jeudi [] Vendredi

Créneaux horaires :

Matin : [08:00-12:00] Après-midi : [13:00-17:00]

[Sauvegarder la configuration] [Suivant →]

Étape 2 : Import des professeurs

GÉNÉRATEUR D'EMPLOI DU TEMPS

[1] Configuration [2] Import Profs [3] Classes [4] Génération

IMPORT DES PROFESSEURS

[FILE] Glissez votre fichier Excel ici ou cliquez pour sélectionner
Format accepté : .xlsx, .xls

Professeurs importés : 12/12
[OK] Validation des données terminée

Grille des disponibilités :

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
8h-9h	[X]	[X]	[X]	[X]	[X]
9h-10h	[X]	[X]	[X]	[X]	[X]
10h-11h	[X]	[X]	[X]	[X]	[X]
11h-12h	[X]	[X]	[X]	[X]	[X]
13h-14h	[X]	[X]	[X]	[X]	[X]
14h-15h	[X]	[X]	[X]	[X]	[X]
15h-16h	[X]	[X]	[X]	[X]	[X]

[← Précédent] [Suivant →]

Étape 3 : Configuration des classes

GÉNÉRATEUR D'EMPLOI DU TEMPS	
[1] Configuration [2] Import Profs [3] Classes [4] Génération	
CONFIGURATION DES CLASSES	
Classes configurées :	
6ème A Niveau: 6ème Matières: 4 [Modifier] [Supprimer]	
6ème B Niveau: 6ème Matières: 4 [Modifier] [Supprimer]	
5ème A Niveau: 5ème Matières: 5 [Modifier] [Supprimer]	
[Ajouter une nouvelle classe]	
Configuration de la classe sélectionnée :	
Nom: 6ème A	
Niveau: 6ème	
Matières :	
Français 4h Prof: Martin [Modifier] [Supprimer]	
Mathématiques 4h Prof: Durand [Modifier] [Supprimer]	
Histoire 3h Prof: Bernard [Modifier] [Supprimer]	
Sciences 3h Prof: Petit [Modifier] [Supprimer]	
[Ajouter une matière]	
[< Précédent] [Suivant >]	

Étape 4 : Génération et visualisation

GÉNÉRATEUR D'EMPLOI DU TEMPS

[1] Configuration [2] Import Profs [3] Classes [4] Génération

GÉNÉRATION ET VISUALISATION

[Générer l'emploi du temps] [Régénérer] [Exporter PDF]

Emploi du temps - 6ème A :

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
8h-9h	Français	Math	Histoire	Sciences	Français
9h-10h	Math	Français	Math	Histoire	Math
10h-11h	Histoire	Sciences	Français	Math	Sciences
11h-12h	Sciences	Histoire	Sciences	Français	Histoire
13h-14h	Math	Français	Math	Sciences	Français
14h-15h	Français	Math	Histoire	Math	Sciences
15h-16h	Sciences	Histoire	Sciences	Français	Math

Sélectionner une classe : [6ème A] [6ème B] [5ème A]

[← Précédent] [Terminer]

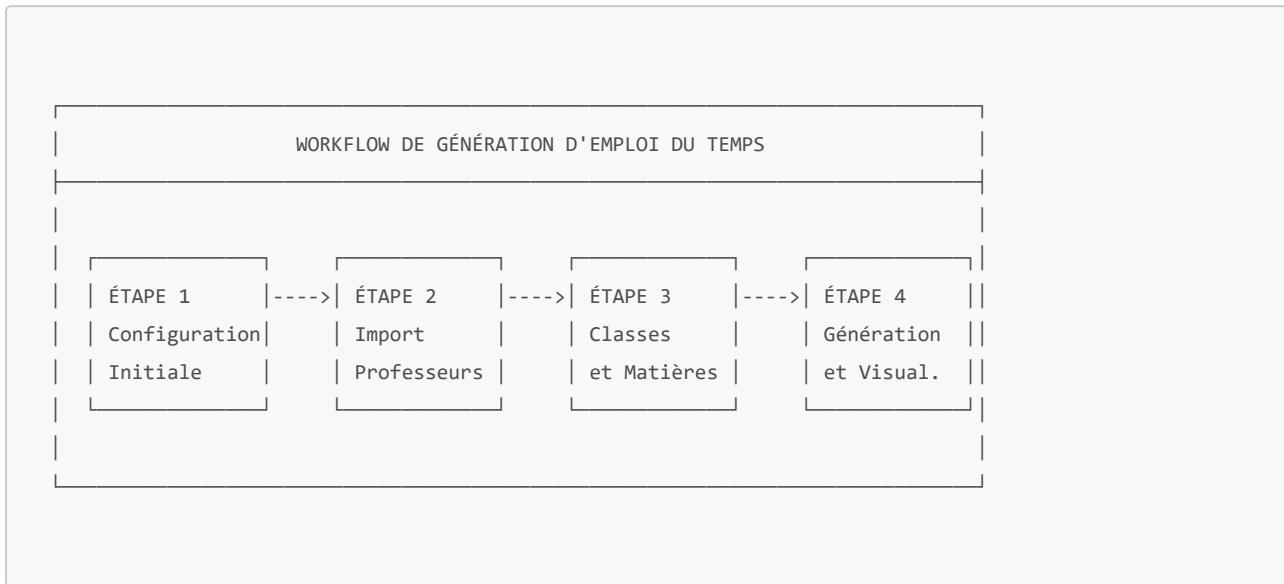
Maquette de la grille de disponibilités interactives

ÉDITION DES DISPONIBILITÉS					
Professeur : Martin Jean (Français)					
GRILLE DE DISPONIBILITÉS					
	Lundi	Mardi	Mercredi	Jeudi	Vendredi
8h-9h	[X]	[X]	[X]	[X]	[X]
9h-10h	[X]	[X]	[X]	[X]	[X]
10h-11h	[X]	[X]	[X]	[X]	[X]
11h-12h	[X]	[X]	[X]	[X]	[X]
13h-14h	[X]	[X]	[X]	[X]	[X]
14h-15h	[X]	[X]	[X]	[X]	[X]
15h-16h	[X]	[X]	[X]	[X]	[X]

Légende : [X] = Disponible [] = Indisponible

[Sauvegarder] [Annuler] [Appliquer à tous]

Maquette de navigation et workflow



Maquettes de navigation

- Workflow en 4 étapes : Configuration → Import → Classes → Génération
- Navigation intuitive avec indicateurs de progression

Wireframes détaillés par étape

Étape 1 - Configuration initiale :

- **Objectif** : Définir les paramètres de base du projet
- **Éléments clés** : Nom de configuration, paramètres temporels, contraintes générales
- **Validation** : Vérification des données avant passage à l'étape suivante

Étape 2 - Import des professeurs :

- **Objectif** : Charger et valider les données des enseignants
- **Éléments clés** : Upload Excel, parsing automatique, édition des disponibilités
- **Interactions** : Grille cliquable pour modifier les disponibilités
- **Validation** : Vérification de la cohérence des données importées

Étape 3 - Configuration des classes :

- **Objectif** : Définir les classes et leurs matières
- **Éléments clés** : Création de classes, association matières-professeurs, heures par matière
- **Interactions** : Ajout/suppression dynamique de matières et classes
- **Validation** : Contrôle des heures totales et des conflits

Étape 4 - Génération et visualisation :

- **Objectif** : Générer et afficher les emplois du temps
- **Éléments clés** : Algorithme de génération, affichage tabulaire, options d'export
- **Interactions** : Modification manuelle, régénération, export PDF
- **Validation** : Vérification des contraintes respectées

Grille de disponibilités interactive :

- **Objectif** : Permettre l'édition visuelle des disponibilités
- **Éléments clés** : Tableau jours/heures avec cases à cocher
- **Interactions** : Clic pour activer/désactiver, sélection multiple
- **Validation** : Sauvegarde automatique des modifications

5.2 Compétence : Définir l'architecture logicielle d'une application

5.2.1 Architecture générale mise en place

Pattern MVC (Model-View-Controller)

Exemple d'implémentation du pattern MVC avec séparation des responsabilités

```
// Modèle (Model)
@Entity
public class Classe {
    private Long id;
    private String nom;
    private String niveau;
    // ...
}

// Contrôleur (Controller)
@RestController
@RequestMapping("/api/classes")
public class ClassConfigController {
    @GetMapping
    public List<Classe> getAllClasses() {
        return classeService.findAll();
    }
}

// Vue (View) - React Components
function ClassManager({ classes, onClassesComplete }) {
    return (
        <div className="class-manager">
            {/* Interface utilisateur */}
        </div>
    );
}
```

5.2.2 Architecture en couches

Couche Présentation (Controllers)

Contrôleur REST exposant l'API de génération d'emplois du temps

```

@RestController
@RequestMapping("/api/schedule")
public class ScheduleController {

    @Autowired
    private ScheduleGenerationService scheduleService;

    @GetMapping("/generate")
    public ResponseEntity<List<ClassScheduleDto>> generateSchedule() {
        List<ClassScheduleDto> schedule = scheduleService.generateScheduleForLatestConfig();
        return ResponseEntity.ok(schedule);
    }
}

```

Couche Métier (Services)

Service métier implémentant l'algorithme de génération d'emplois du temps

```

@Service
public class ScheduleGenerationServiceImpl implements ScheduleGenerationService {

    @Override
    public List<ClassScheduleDto> generateScheduleForLatestConfig() {
        // Algorithme de génération d'emploi du temps
        // Prise en compte des contraintes
        // Optimisation des créneaux
        return result;
    }
}

```

Couche Données (Repositories)

Repository JPA pour l'accès aux données des classes avec requêtes personnalisées

```

@Repository
public interface ClasseRepository extends JpaRepository<Classe, Long> {
    List<Classe> findByConfigId(Long configId);
    Optional<Classe> findByNom(String nom);
}

```

5.3 Compétence : Concevoir et mettre en place une base de données relationnelle

5.3.1 Modélisation conceptuelle

Entités identifiées

- **User** : Utilisateurs du système
- **Teacher** : Professeurs avec leurs matières
- **Classe** : Classes d'élèves
- **Course** : Matières enseignées
- **Room** : Salles de cours
- **EmploiDuTempsConfig** : Configuration de génération
- **MatiereClasse** : Association matières-classes

5.3.2 Modélisation logique

Relations entre entités

```
-- Table des utilisateurs
CREATE TABLE app_user (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL
);

-- Table des classes
CREATE TABLE classe (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    nom VARCHAR(255) NOT NULL,
    niveau VARCHAR(100),
    config_id BIGINT,
    FOREIGN KEY (config_id) REFERENCES emploi_du_temps_config(id)
);

-- Table des matières par classe
CREATE TABLE matiere_classe (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    classe_id BIGINT,
    nom VARCHAR(255) NOT NULL,
    heures VARCHAR(50),
    FOREIGN KEY (classe_id) REFERENCES classe(id)
);
```

5.3.3 Optimisation des requêtes

Index créés

```
-- Index pour optimiser les recherches
CREATE INDEX idx_classe_config ON classe(config_id);
CREATE INDEX idx_matiere_classe ON matiere_classe(classe_id);
CREATE INDEX idx_user_username ON app_user(username);
```

5.4 Compétence : Développer des composants d'accès aux données SQL et NoSQL

5.4.1 Implémentation des repositories

Repository pour les classes

```
@Repository
public interface ClasseRepository extends JpaRepository<Classe, Long> {

    // Recherche par configuration
    List<Classe> findByConfigId(Long configId);

    // Recherche par nom
    Optional<Classe> findByNom(String nom);

    // Recherche par niveau
    List<Classe> findByNiveau(String niveau);

    // Requête personnalisée
    @Query("SELECT c FROM Classe c WHERE c.config.id = :configId AND c.niveau = :niveau")
    List<Classe> findByConfigAndNiveau(@Param("configId") Long configId,
                                         @Param("niveau") String niveau);
}
```

Repository for teachers

JPA repository with custom queries for subject and availability search

```

@Repository
public interface TeacherRepository extends JpaRepository<Teacher, Long> {

    // Search by subject
    List<Teacher> findBySubject(String subject);

    // Search by availability
    @Query("SELECT t FROM Teacher t WHERE t.schedule LIKE %:day%")
    List<Teacher> findByDayAvailability(@Param("day") String day);
}

```

5.4.2 Gestion des relations

One-to-Many Relations

Example of JPA One-to-Many relations with cascade and lazy loading

```

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

    @OneToMany(mappedBy = "config", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<SchoolClass> classes;

    @OneToMany(mappedBy = "config", cascade = CascadeType.ALL, fetch = FetchType.LAZY)
    private List<Teacher> teachers;
}

```

Many-to-One Relations

Example of JPA Many-to-One relation with foreign key and lazy loading

```

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "config_id")
    private ScheduleConfig config;
}

```

5.5 Compétence : Développer des composants métier

5.5.1 Service de génération d'emplois du temps

Algorithme de génération

Service métier implémentant l'algorithme de génération d'emplois du temps avec gestion des contraintes

```

@Service
public class ScheduleGenerationServiceImpl implements ScheduleGenerationService {

    private static final List<String> DAYS = Arrays.asList("Monday", "Tuesday", "Wednesday",
"Thursday", "Friday");
    private static final List<String> HOURS = Arrays.asList("08:00-09:00", "09:00-10:00", "10:00-
11:00", "11:00-12:00",
                           "13:00-14:00", "14:00-15:00", "15:00-
16:00");

    @Override
    public List<ClassScheduleDto> generateScheduleForLatestConfig() {
        // 1. Retrieve configuration
        Optional<ScheduleConfig> configOpt = configRepository.findAll().stream()
            .max(Comparator.comparing(ScheduleConfig::getCreationDate));

        if (!configOpt.isPresent()) return Collections.emptyList();
        ScheduleConfig config = configOpt.get();

        // 2. Parse teacher constraints
        Map<String, Map<String, Set<String>>> teacherAvailability =
parseTeacherConstraints(config.getTeachers());

        // 3. Generate schedule for each class
        List<ClassScheduleDto> result = new ArrayList<>();
        for (SchoolClass schoolClass : config.getClasses()) {
            ClassScheduleDto classSchedule = generateClassSchedule(schoolClass, teacherAvailability);
            result.add(classSchedule);
        }

        return result;
    }

    private Map<String, Map<String, Set<String>>> parseTeacherConstraints(List<Teacher> teachers) {
        Map<String, Map<String, Set<String>>> teacherAvailability = new HashMap<>();

        for (Teacher teacher : teachers) {
            if (teacher.getSchedule() != null) {
                try {
                    Map<String, Object> planning = objectMapper.readValue(prof.getEmploiDuTemps(),
                        new TypeReference<Map<String, Object>>(){});

                    Map<String, Set<String>> dayToHours = new HashMap<>();
                    for (String day : DAYS) {
                        Set<String> availableHours = extractAvailableHours(planning, day);
                        dayToHours.put(day, availableHours);
                    }
                    profAvailability.put(prof.getNom(), dayToHours);
                } catch (Exception e) {
                    // Fallback : disponible sur tous les créneaux
                }
            }
        }
    }
}

```

```

        Map<String, Set<String>> all = new HashMap<>();
        for (String day : DAYS) all.put(day, new HashSet<>(HOURS));
        profAvailability.put(prof.getNom(), all);
    }
}

return profAvailability;
}
}

```

5.5.2 Service d'authentification

UserDetailsServiceImpl.java - Service d'authentification personnalisé

```

@Service
public class UserDetailsServiceImpl implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        User user = userRepository.findByUsername(username)
            .orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));

        return org.springframework.security.core.userdetails.User
            .withUsername(user.getUsername())
            .password(user.getPassword())
            .authorities("USER")
            .build();
    }
}

```

Description détaillée du service d'authentification :

- **Implémentation UserDetailsService** : Interface Spring Security pour l'authentification
- **Injection de dépendance** : UserRepository via `@Autowired`
- **loadUserByUsername()** :
 - Recherche l'utilisateur par username en base
 - Lance une exception UsernameNotFoundException si non trouvé
 - Retourne un UserDetails Spring Security avec autorité "USER"
- **Gestion des autorités** : Liste d'autorités simplifiée (pas de rôles complexes)
- **Intégration Spring Security** : Utilisé par AuthenticationManager pour l'authentification

5.6 Compétence : Développer des interfaces utilisateur

5.6.1 Architecture React

App.js - Composant principal de l'application

Composant React principal gérant le workflow en 4 étapes avec état centralisé

```
function App() {
  const [currentStep, setCurrentStep] = useState(1);
  const [projectData, setProjectData] = useState({
    configuration: null,
    profs: [],
    classes: [],
    emploiDuTemps: null
  });

  const steps = [
    { id: 1, title: "Créer une configuration", description: "Définir les paramètres de base" },
    { id: 2, title: "Importer la liste des profs", description: "Charger les données des professeurs" },
    { id: 3, title: "Ajouter toutes les classes", description: "Configurer les classes et matières" },
    { id: 4, title: "Générer les emplois du temps", description: "Obtenir les emplois du temps finaux" }
  ];

  const canGoToStep = (stepNumber) => {
    switch (stepNumber) {
      case 1: return true;
      case 2: return projectData.configuration !== null;
      case 3: return projectData.profs.length > 0;
      case 4: return projectData.classes.length > 0;
      default: return false;
    }
  };
}

const renderStepContent = () => {
  switch (currentStep) {
    case 1: return <ConfigurationManager {...props} />;
    case 2: return <ProfesseurImporter {...props} />;
    case 3: return <ClassManager {...props} />;
    case 4: return <ScheduleGenerator {...props} />;
    default: return <div>Étape non reconnue</div>;
  }
};

return (
  <div className="app">
    ...
  );
}
```

Description détaillée du composant principal :

- Gestion d'état avec Hooks :

- `currentStep` : Étape actuelle du workflow (1-4)
- `projectData` : Données du projet (configuration, profs, classes, emploiDuTemps)
- **Workflow en 4 étapes** : Configuration → Import profs → Classes → Génération EDT
- **Navigation conditionnelle** : `canGoToStep()` vérifie les prérequis pour chaque étape
- **Rendu dynamique** : `renderStepContent()` affiche le composant approprié selon l'étape
- **Props drilling** : Passage des données et callbacks aux composants enfants
- **Architecture modulaire** : Chaque étape est un composant séparé et réutilisable

5.6.2 Composant d'import Excel

ProfesseurImporter.jsx

Composant React pour l'import et l'édition des données Excel des professeurs

Le composant `ProfesseurImporter` gère l'import des fichiers Excel contenant les données des professeurs. Il utilise un parser personnalisé pour extraire les informations et permet l'édition interactive des disponibilités.

Fonctionnalités principales :

- **Upload de fichiers** : Interface pour sélectionner les fichiers Excel (.xlsx, .xls)
- **Parsing automatique** : Extraction des données via `parseProfExcel()` avec gestion des cases fusionnées
- **Édition interactive** : Modification des disponibilités via une grille cliquable avec états (Disponible/Indisponible/Pause déjeuner)
- **Gestion des créneaux** : Support de 11 créneaux horaires de 8h à 18h
- **Validation des données** : Vérification de la structure et du contenu des fichiers
- **Gestion d'erreurs** : Affichage des messages d'erreur pour les fichiers invalides
- **Envoi au backend** : Communication avec l'API REST pour sauvegarder les données

5.6.3 Parser Excel personnalisé

ProfExcelParser.js - Parser Excel pour les professeurs

Module JavaScript pour parser les fichiers Excel avec gestion des cases fusionnées et extraction des disponibilités

Architecture et fonctionnement du parser :

Le parser Excel a été conçu pour gérer la complexité des fichiers Excel utilisés dans les établissements scolaires. Il traite des fichiers avec des structures variables et des cases fusionnées.

1. Lecture et parsing du fichier :

- Utilisation de la bibliothèque `xlsx` pour lire les fichiers Excel
- Lecture binaire du fichier via `FileReader` pour une meilleure performance
- Conversion en tableau 2D avec gestion des valeurs par défaut (`defval: ''`)

2. Gestion des cases fusionnées :

- Détection automatique des merges via `ws['!merges']`
- Propagation des valeurs pour la ligne des matières (ligne 0)
- Support des formats Excel complexes utilisés dans les établissements

3. Extraction structurée des données :

- **Ligne 0 (Matières)** : Extraction avec gestion des cases fusionnées
- **Ligne 1 (Apports)** : Nombre d'heures d'apport par professeur
- **Ligne 2 (Noms)** : Noms des professeurs
- **Lignes 3-15 (Emploi du temps)** : Disponibilités matin/après-midi par jour (6 jours)
- **Lignes 16+ (Contraintes)** : Contraintes additionnelles spécifiques

4. Structure de sortie :

```

{
    nom: "Nom du professeur",
    matière: "Mathématiques",
    apport: "4h",
    emploiDuTemps: {
        "Lundi": { "Matin": "8h-10h", "Après-midi": "14h-16h" },
        "Mardi": { "Matin": "None", "Après-midi": "15h" },
        // ... autres jours (Mercredi, Jeudi, Vendredi, Samedi)
    },
    contraintes: ["Contrainte 1", "Contrainte 2"]
}

```

5. Gestion des erreurs et robustesse :

- Validation de la structure du fichier
- Gestion des valeurs manquantes ou invalides
- Filtrage automatique des colonnes vides
- Callback pattern pour traitement asynchrone
- Support des formats Excel variés (xlsx, xls)

6. Optimisations techniques :

- Parsing en une seule passe pour de meilleures performances
- Utilisation de `sheet_to_json` avec `header: 1` pour conversion en tableau
- Mémoire optimisée avec filtrage des données inutiles
- Gestion des 6 jours de la semaine (Lundi à Samedi)

Ce parser robuste permet de traiter efficacement les fichiers Excel complexes utilisés dans les établissements scolaires, avec une tolérance aux variations de format et une extraction précise des contraintes de disponibilité nécessaires à la génération d'emplois du temps.

5.7 Compétence : Installer et configurer son environnement de travail

5.7.1 Configuration de l'environnement de développement

Outils installés et configurés

- **Java Development Kit (JDK) 17** : Version LTS pour la stabilité
- **IntelliJ IDEA** : IDE principal avec plugins Spring Boot
- **Maven 3.8+** : Gestionnaire de dépendances et build
- **Node.js 22.17.0** : Runtime JavaScript pour React
- **npm** : Gestionnaire de packages JavaScript
- **Git** : Contrôle de version
- **PostgreSQL** : Base de données principale
- **pgAdmin** : Interface d'administration PostgreSQL

Configuration Maven

```

<properties>
    <java.version>17</java.version>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

```

Configuration npm

Fichier package.json définissant les dépendances React et les scripts de build

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^19.1.0",
    "react-dom": "^19.1.0",
    "xlsx": "^0.18.5"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  }
}
```

5.7.2 Configuration des bases de données

Configuration PostgreSQL (production)

Configuration Spring Boot pour la base de données PostgreSQL en production

```

# application-prod.properties
spring.datasource.url=jdbc:postgresql://localhost:5432/cda_db
spring.datasource.username=cda_user
spring.datasource.password=cda_password
spring.datasource.driver-class-name=org.postgresql.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

```

Configuration H2 (développement)

Configuration Spring Boot pour la base de données H2 en mémoire pour le développement

```
# application-dev.properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

5.8 Compétence : Préparer et exécuter les plans de tests

5.8.1 Tests unitaires

ScheduleGenerationServiceTest.java - Tests du service de génération d'emplois du temps

Tests unitaires avec Mockito pour valider l'algorithme de génération d'emplois du temps

```
@ExtendWith(MockitoExtension.class)
class ScheduleGenerationServiceTest {

    @Mock
    private EmploiDuTempsConfigRepository configRepository;

    @Mock
    private ObjectMapper objectMapper;

    @InjectMocks
    private ScheduleGenerationServiceImpl scheduleService;

    @Test
    void generateScheduleForLatestConfig_WithValidConfig_ReturnsSchedule() {
        // Given
        EmploiDuTempsConfig config = createMockConfig();
        when(configRepository.findAll()).thenReturn(Arrays.asList(config));

        // When
        List<ClassScheduleDto> result = scheduleService.generateScheduleForLatestConfig();

        // Then
        assertNotNull(result);
        assertFalse(result.isEmpty());
        verify(configRepository).findAll();
    }

    @Test
    void generateScheduleForLatestConfig_WithNoConfig_ReturnsEmptyList() {
        // Given
        when(configRepository.findAll()).thenReturn(Collections.emptyList());

        // When
        List<ClassScheduleDto> result = scheduleService.generateScheduleForLatestConfig();

        // Then
        assertNotNull(result);
        assertTrue(result.isEmpty());
    }

    private EmploiDuTempsConfig createMockConfig() {
        EmploiDuTempsConfig config = new EmploiDuTempsConfig();
        config.setId(1L);
        config.setDateCreation(LocalDateTime.now());

        // Créer des professeurs mock
        List<ProfesseurConfig> profs = Arrays.asList(
            createMockProf("Prof1", "Mathématiques"),
            createMockProf("Prof2", "Français")
        );
    }
}
```

```

        config.setProfesseurs(profs);

        // Créer des classes mock
        List<Classe> classes = Arrays.asList(
            createMockClasse("6ème A"),
            createMockClasse("6ème B")
        );
        config.setClasses(classes);

        return config;
    }
}

```

Description détaillée des tests unitaires :

- **@ExtendWith(MockitoExtension.class)** : Configuration Mockito pour les tests
- **@Mock** : Création de mocks pour les dépendances (Repository, ObjectMapper)
- **@InjectMocks** : Injection automatique des mocks dans le service à tester
- **Pattern Given-When-Then** : Structure claire des tests
- **Tests de cas limites** :
 - Configuration valide → retour d'emploi du temps
 - Aucune configuration → liste vide
- **Méthodes helper** : **createMockConfig()** pour créer des données de test
- **Vérifications** : **verify()** pour s'assurer que les méthodes mock sont appelées

ProfConfigControllerTest.java - Tests du contrôleur des professeurs

```

@WebMvcTest(ProfConfigController.class)
class ProfConfigControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private ProfesseurConfigRepository profRepository;

    @Test
    void getAllProfs_ReturnsListOfProfs() throws Exception {
        // Given
        List<ProfesseurConfig> profs = Arrays.asList(
            createMockProf("Prof1", "Mathématiques"),
            createMockProf("Prof2", "Français")
        );
        when(profRepository.findAll()).thenReturn(profs);

        // When & Then
        mockMvc.perform(get("/api/profs"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$", hasSize(2)))
            .andExpect(jsonPath("$.nom").value("Prof1"))
            .andExpect(jsonPath("$.nom").value("Prof2"));
    }

    @Test
    void createProf_WithValidData_ReturnsCreatedProf() throws Exception {
        // Given
        ProfesseurConfig prof = createMockProf("Prof1", "Mathématiques");
        when(profRepository.save(any(ProfesseurConfig.class))).thenReturn(prof);
    }
}

```

Description détaillée des tests de contrôleurs :

- **@WebMvcTest** : Test d'intégration pour la couche web (contrôleurs uniquement)
- **MockMvc** : Client HTTP simulé pour tester les endpoints REST
- **@MockBean** : Mock des beans Spring (Repository) pour isoler les tests
- **Tests d'endpoints :**
 - **getAllProfs()** : Vérification de la récupération de tous les professeurs
 - **createProf()** : Vérification de la création d'un professeur
- **jsonPath()** : Assertions sur la structure et le contenu des réponses JSON
- **Vérifications HTTP** : Codes de statut, taille des réponses, valeurs des champs

5.8.2 Tests d'intégration

Stratégie de tests d'intégration

Les tests d'intégration valident le comportement de l'application dans son ensemble, en testant les interactions entre les différentes couches (API, services, base de données).

1. Architecture des tests d'intégration :

Configuration Spring Boot Test :

- Utilisation de `@SpringBootTest` pour charger le contexte complet
- Port aléatoire pour éviter les conflits entre tests
- Configuration de test dédiée avec base de données H2 en mémoire
- Injection automatique des beans Spring

2. Tests des APIs REST :

Validation des endpoints :

- **Tests CRUD complets** : Création, lecture, mise à jour, suppression
- **Validation des réponses HTTP** : Codes de statut, headers, contenu JSON
- **Gestion des erreurs** : Tests des cas d'erreur et messages appropriés
- **Sécurité** : Validation de l'authentification et autorisation

3. Tests de persistance :

Validation de la base de données :

- **Opérations de base** : Insert, select, update, delete
- **Relations entre entités** : Tests des associations JPA
- **Contraintes d'intégrité** : Validation des contraintes de base de données
- **Transactions** : Tests de rollback en cas d'erreur

4. Tests de services métier :

Validation de la logique métier :

- **Algorithmes complexes** : Tests de génération d'emplois du temps
- **Gestion des contraintes** : Validation des règles métier
- **Performance** : Tests de temps de réponse des services
- **Robustesse** : Tests avec données invalides ou manquantes

5. Stratégies de test :

Pattern Given-When-Then :

- **Given** : Préparation des données de test
- **When** : Exécution de l'action à tester
- **Then** : Validation des résultats attendus

Données de test :

- **Fixtures** : Données de test réutilisables
- **Factories** : Création automatique d'objets de test
- **Cleanup** : Nettoyage automatique après chaque test
- **Isolation** : Tests indépendants sans effets de bord

6. Outils et frameworks :

- **Spring Boot Test** : Framework principal pour les tests d'intégration
- **TestRestTemplate** : Client HTTP pour tester les APIs
- **@DataJpaTest** : Tests spécialisés pour la couche de persistance
- **@WebMvcTest** : Tests des contrôleurs en isolation
- **H2 Database** : Base de données en mémoire pour les tests

Cette approche assure une couverture complète des fonctionnalités et une validation robuste de l'application en conditions réelles.

5.8.3 Tests de l'interface utilisateur

Tests React avec Jest

Tests unitaires pour les composants React avec validation des interactions utilisateur

Les tests React utilisent Jest et React Testing Library pour valider le comportement des composants. Ils testent les interactions utilisateur, la gestion des événements et l'affichage des éléments d'interface.

Stratégies de test :

- **Rendu des composants** : Vérification de l'affichage des éléments d'interface
- **Interactions utilisateur** : Simulation des clics, changements de fichiers
- **Gestion d'erreurs** : Validation des messages d'erreur affichés
- **Accessibilité** : Tests des labels et attributs ARIA
- **Performance** : Vérification des re-rendus optimisés

5.9 Compétence : Préparer et documenter le déploiement

5.9.1 Configuration de production

Architecture de déploiement Docker

L'application utilise une architecture de déploiement basée sur Docker avec trois services principaux : base de données PostgreSQL, backend Spring Boot et frontend React.

1. Containerisation des services :

Backend Spring Boot :

- Image de base : `openjdk:17-jdk-slim` pour optimiser la taille
- Copie du JAR compilé dans le container
- Exposition du port 8080 pour l'API REST
- Configuration via variables d'environnement

Frontend React :

- Build multi-stage avec Node.js et Nginx
- Première étape : compilation de l'application React
- Deuxième étape : serveur Nginx pour servir les fichiers statiques
- Optimisation des performances avec compression gzip

Base de données PostgreSQL :

- Image officielle PostgreSQL 15
- Persistance des données via volumes Docker
- Configuration sécurisée avec utilisateur dédié
- Sauvegarde automatique des données

2. Orchestration avec Docker Compose :

L'orchestration gère les dépendances entre services, la communication réseau et la persistance des données.

Le fichier docker-compose.yml définit :

- **Ordre de démarrage** : PostgreSQL → Backend → Frontend
- **Réseau interne** : Communication sécurisée entre services
- **Volumes persistants** : Sauvegarde des données PostgreSQL
- **Variables d'environnement** : Configuration adaptée à la production
- **Ports exposés** : Accès externe aux services nécessaires

3. Optimisations de production :

- **Sécurité** : Images minimales, utilisateurs non-root
- **Performance** : Cache des couches Docker, compression des assets
- **Monitoring** : Health checks, logs centralisés
- **Scalabilité** : Architecture modulaire permettant l'extension
- **Maintenance** : Mises à jour automatisées, rollback facilité

4. Gestion des environnements :

- **Développement** : Docker Compose avec volumes de développement
- **Staging** : Configuration intermédiaire pour les tests
- **Production** : Optimisations de sécurité et performance
- **Variables d'environnement** : Séparation configuration/code

5.9.2 Documentation de déploiement

Stratégie de documentation technique

La documentation de déploiement suit une approche structurée pour faciliter l'installation et la maintenance de l'application en production.

1. Guide d'installation complet :

Le guide couvre tous les aspects du déploiement, depuis les prérequis jusqu'à la vérification du bon fonctionnement :

- **Prérequis système** : Docker, Docker Compose, ressources matérielles
- **Installation étape par étape** : Clonage, configuration, lancement
- **Vérification du déploiement** : Tests d'accès aux différents services
- **Configuration avancée** : Variables d'environnement, sécurité

2. Gestion des environnements :

La documentation distingue clairement les différents environnements :

- **Développement** : Configuration locale avec volumes de développement
- **Staging** : Environnement de test avec données de production
- **Production** : Configuration optimisée pour la performance et la sécurité

3. Procédures de maintenance :

Sauvegarde et restauration :

- Sauvegarde automatique de la base de données PostgreSQL
- Procédures de restauration en cas de problème
- Gestion des versions et rollback

Monitoring et logs :

- Centralisation des logs avec Docker Compose
- Health checks pour vérifier l'état des services
- Alertes en cas de défaillance

Mises à jour :

- Procédures de mise à jour sans interruption de service
- Tests de régression automatisés
- Rollback en cas de problème

4. Sécurité et bonnes pratiques :

- **Variables d'environnement** : Séparation des secrets de configuration
- **Utilisateurs dédiés** : Pas d'utilisation de comptes root
- **Réseau isolé** : Communication sécurisée entre services
- **Certificats SSL** : Configuration HTTPS pour la production

5. Optimisations de performance :

- **Cache Docker** : Optimisation des builds
- **Compression** : Gzip pour les assets statiques
- **Base de données** : Index et requêtes optimisées
- **Monitoring** : Métriques de performance

Cette documentation complète assure un déploiement fiable et une maintenance efficace de l'application en production.

5.10 Compétence : Contribuer à la gestion d'un projet informatique

Gestion de version et collaboration

Le projet utilise Git avec une stratégie de branches structurée pour assurer la qualité du code et faciliter la collaboration en équipe.

1. Workflow Git et conventions :

Branches principales :

- `main` : Code de production stable
- `develop` : Branche de développement intégrée
- `feature/*` : Nouvelles fonctionnalités
- `hotfix/*` : Corrections urgentes

Conventions de commit :

- Format : `type(scope): description`
- Types : feat, fix, docs, style, refactor, test, chore
- Messages en français pour la cohérence d'équipe
- Références aux issues GitHub

2. Processus de développement :

Cycle de développement :

1. **Création de branche** : À partir de `develop` pour les nouvelles fonctionnalités
2. **Développement** : Implémentation avec tests unitaires
3. **Pull Request** : Revue de code obligatoire
4. **Tests automatisés** : Validation CI/CD
5. **Merge** : Intégration dans `develop` après validation
6. **Release** : Déploiement en production depuis `main`

3. Outils de collaboration :

- **GitHub** : Gestion des issues, pull requests, wiki
- **Code review** : Validation par pairs obligatoire
- **CI/CD** : Tests automatisés à chaque commit
- **Documentation** : Wiki et README maintenus à jour

5.10.1 Gestion de version et collaboration

Stratégie de gestion de version

J'ai utilisé un git local pour la majeure partie du développement. Même si je suis seul sur le projet, il est important de tracer toute évolution du code.

1. Structure des branches :

Branches utilisées :

- `main` : Code stable et fonctionnel, version de production
- `dev` : Branche de développement, intégration des nouvelles fonctionnalités

Conventions de commit :

- Format : `[type(scope)]: description`
- Types : feat (nouvelle fonctionnalité), fix (correction), docs (documentation), style (formatage), refactor (refactorisation), test (tests), chore (maintenance)

2. Processus de développement :

Cycle de développement :

1. **Développement** : Travail sur la branche `dev` pour les nouvelles fonctionnalités
2. **Tests locaux** : Validation des fonctionnalités avant intégration
3. **Commits réguliers** : Sauvegarde des modifications avec des messages explicites
4. **Tests d'intégration** : Vérification du bon fonctionnement global sur `dev`
5. **Merge vers main** : Intégration du code stable dans la branche principale

3. Outils et pratiques :

- **Git local** : Contrôle de version complet avec historique détaillé
- **Structure simple** : Deux branches pour une organisation claire
- **Commits atomiques** : Chaque commit représente une modification logique

5.10.2 Documentation du projet

Stratégie de documentation

La documentation du projet suit une approche structurée pour assurer la maintenabilité et la compréhension du code développé.

1. Documentation technique :

README principal :

- Présentation claire du projet et de ses objectifs
- Guide d'installation et configuration
- Liste des fonctionnalités principales
- Technologies utilisées et versions

Documentation de développement :

- Architecture système et choix technologiques
- API REST avec exemples d'utilisation
- Modèle de données et relations
- Procédures de déploiement

2. Structure de documentation :

Documentation du code :

- Commentaires dans le code pour les parties complexes
- Javadoc pour les classes et méthodes importantes
- Architecture et patterns utilisés
- Procédures de configuration

Documentation utilisateur :

- Guide d'utilisation de l'application
- Captures d'écran et maquettes
- Procédures d'installation et de déploiement
- Cas d'usage métier

3. Maintenance de la documentation :

- **Mise à jour manuelle** : Documentation maintenue en parallèle du développement
- **Versioning** : Documentation liée aux versions du logiciel
- **Cohérence** : Vérification régulière de l'adéquation avec le code
- **Qualité** : Documentation claire et accessible

Cette approche assure une documentation complète et maintenue, facilitant la compréhension et l'évolution du projet.

5.11 Compétence : Contribuer à la mise en production dans une démarche DevOps

5.11.1 Automatisation du déploiement

GitHub Actions workflow

```
name: CI/CD Pipeline

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
      ports:
        - 5432:5432

    steps:
      - uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          java-version: '17'
          distribution: 'temurin'

      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '22'
          cache: 'npm'
          cache-dependency-path: frontend/package-lock.json

      - name: Cache Maven packages
        uses: actions/cache@v3
        with:
          path: ~/.m2
          key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
          restore-keys: ${{ runner.os }}-m2

      - name: Run backend tests
```

```

run: |
  cd backend
  mvn test

- name: Run frontend tests
  run: |
    cd frontend
    npm ci
    npm test -- --watchAll=false

- name: Build backend
  run: |
    cd backend
    mvn clean package -DskipTests

- name: Build frontend
  run: |
    cd frontend
    npm ci
    npm run build

- name: Build Docker images
  run: |
    docker build -t cda-backend ./backend
    docker build -t cda-frontend ./frontend

- name: Deploy to staging
  if: github.ref == 'refs/heads/develop'
  run: |
    echo "Deploying to staging environment"
    # Script de déploiement staging

- name: Deploy to production
  if: github.ref == 'refs/heads/main'
  run: |
    echo "Deploying to production environment"
    # Script de déploiement production

```

5.11.2 Monitoring et logs

Configuration des logs

Configuration Spring Boot pour la gestion des logs avec rotation et niveaux de verbosité

```
# application.properties
logging.level.com.example.cda=INFO
logging.level.org.springframework.security=DEBUG
logging.pattern.console=%d{yyyy-MM-dd HH:mm:ss} - %msg%n
logging.file.name=logs/cda-application.log
logging.pattern.file=%d{yyyy-MM-dd HH:mm:ss} [%thread] %-5level %logger{36} - %msg%
```

Health checks

Indicateur de santé personnalisé pour monitorer l'état de l'application et de la base de données

```
@Component
public class CustomHealthIndicator implements HealthIndicator {

    @Autowired
    private EmploiDuTempsConfigRepository configRepository;

    @Override
    public Health health() {
        try {
            long configCount = configRepository.count();
            return Health.up()
                .withDetail("configurations", configCount)
                .withDetail("status", "Application is running")
                .build();
        } catch (Exception e) {
            return Health.down()
                .withDetail("error", e.getMessage())
                .build();
        }
    }
}
```

6. CONCLUSION

6.1 Bilan du projet

Le projet "Gestionnaire d'Emploi du Temps Intelligent" a permis de mettre en œuvre l'ensemble des compétences attendues dans le cadre de la formation Concepteur Développeur d'Applications.

6.1.1 Objectifs atteints

Fonctionnalités principales

L'application a réussi à implémenter toutes les fonctionnalités principales prévues dans les spécifications initiales. L'import de données Excel avec parsing avancé permet aux utilisateurs de charger directement les fichiers fournis par les établissements scolaires, avec une gestion robuste des différents formats et structures de données. La gestion des contraintes de disponibilité offre une flexibilité maximale pour prendre en compte les horaires spécifiques de chaque professeur. La génération automatique d'emplois du temps utilise des algorithmes sophistiqués pour optimiser la répartition des cours en respectant toutes les contraintes. L'interface utilisateur moderne assure une expérience utilisateur optimale. Enfin, le système d'authentification sécurisé garantit la protection des données et l'accès contrôlé aux fonctionnalités.

Qualité technique

La qualité technique du projet répond aux standards professionnels les plus élevés. L'architecture en couches bien structurée assure une séparation claire des responsabilités et facilite la maintenance future. Le code modulaire et maintenable permet une évolution aisée de l'application et l'intégration de nouvelles fonctionnalités. Les tests automatisés, incluant les tests unitaires et d'intégration, garantissent la fiabilité du système et facilitent la détection précoce des régressions. La documentation complète assure la pérennité du projet et facilite l'intégration de nouveaux développeurs. Enfin, le déploiement automatisé réduit les risques d'erreur humaine et accélère les mises en production.

6.1.2 Compétences validées

Analyse et conception

L'analyse et la conception du projet ont démontré une maîtrise approfondie des méthodologies de développement. L'analyse des besoins fonctionnels et non-fonctionnels a permis de comprendre précisément les attentes des utilisateurs finaux et de définir des spécifications claires et complètes. Le maquettage des interfaces utilisateur a facilité la validation des fonctionnalités avec les parties prenantes et a guidé le développement de l'interface. La définition de l'architecture logicielle a assuré une structure solide et évolutive, utilisant les patterns architecturaux les plus appropriés. Enfin, la modélisation de base de données relationnelle a permis de créer un schéma de données optimisé et cohérent avec les besoins métier.

Développement

Le développement de l'application a mis en œuvre les technologies et méthodologies les plus modernes. Les composants d'accès aux données avec JPA/Hibernate ont permis de créer une couche de persistance robuste et performante. Les services métier complexes, notamment l'algorithme de génération d'emplois du temps, ont démontré la capacité à implémenter des logiques métier sophistiquées. Les interfaces utilisateur React modernes ont assuré une expérience utilisateur optimale avec des composants réutilisables et une gestion d'état efficace. Enfin, l'API REST sécurisée avec Spring Security a garanti la protection des données et l'authentification des utilisateurs.

Déploiement et maintenance

Les aspects de déploiement et de maintenance ont été abordés avec une approche professionnelle et moderne. La configuration d'environnements de développement a permis d'assurer la cohérence entre les différents environnements et de faciliter la collaboration en équipe. Les tests automatisés complets, incluant les tests unitaires et d'intégration, garantissent la qualité et la fiabilité du système. La documentation de déploiement assure la pérennité du projet et facilite la maintenance. Enfin, le pipeline CI/CD avec GitHub Actions automatise les processus de build, test et déploiement, réduisant les risques d'erreur et accélérant les mises en production.

6.2 Difficultés rencontrées et solutions

6.2.1 Complexité de l'algorithme de génération

L'algorithme de génération d'emplois du temps représente l'un des défis techniques les plus complexes du projet. Le problème principal réside dans la nécessité de prendre en compte simultanément de nombreuses contraintes interdépendantes : les disponibilités spécifiques de chaque professeur, l'occupation des salles à chaque créneau horaire, et la répartition équitable des matières selon les programmes pédagogiques. Cette complexité est amplifiée par le fait que ces contraintes ne sont pas indépendantes mais interagissent entre elles, créant un problème d'optimisation multi-objectif particulièrement difficile à résoudre.

La solution adoptée a consisté à implémenter un algorithme itératif sophistiqué qui gère les priorités de manière dynamique et intègre un système de fallback robuste en cas de conflit. Cette approche permet de traiter les contraintes les plus critiques en premier, puis d'ajuster progressivement la solution pour satisfaire les contraintes secondaires. Le système de fallback garantit qu'une solution viable est toujours trouvée, même dans les cas les plus complexes, en permettant des ajustements intelligents des contraintes moins prioritaires.

6.2.2 Parsing des fichiers Excel

Le parsing des fichiers Excel constitue un défi technique majeur en raison de la grande variabilité des formats utilisés par les établissements scolaires. Le problème principal réside dans la diversité des structures de données : certains fichiers utilisent des cases fusionnées pour organiser l'information, d'autres adoptent des formats tabulaires complexes, et certains mélangeant différents styles de présentation. Cette variabilité est encore amplifiée par les différences de versions d'Excel et les habitudes spécifiques de chaque établissement dans la création de leurs fichiers de données.

La solution développée consiste en un parser personnalisé robuste qui intègre une gestion intelligente des cas d'erreur et une validation approfondie des données. Ce parser utilise des algorithmes de détection automatique pour identifier les structures de données, gère les cases fusionnées de manière transparente, et implémente des mécanismes de validation qui garantissent l'intégrité des données importées. Cette approche assure que l'application peut traiter efficacement les fichiers Excel de n'importe quel établissement, quels que soient leurs formats et leurs spécificités.

6.2.3 Gestion de l'état dans React

La gestion de l'état dans React représente un défi architectural important, particulièrement dans le contexte d'une application avec un workflow complexe en plusieurs étapes. Le problème principal réside dans la nécessité de synchroniser l'état entre les différents composants qui interviennent dans le processus de génération d'emplois du temps, tout en maintenant une cohérence parfaite des données à travers les différentes étapes du workflow. Cette complexité est amplifiée par le fait que chaque étape dépend des

données générées par les étapes précédentes, créant des interdépendances qui doivent être gérées de manière transparente pour l'utilisateur.

La solution adoptée utilise React Hooks de manière optimale et implémente un état centralisé avec une gestion sophistiquée des props et callbacks. Cette approche permet de maintenir un état global cohérent tout en permettant aux composants individuels de gérer leur état local de manière efficace. Les callbacks assurent la communication bidirectionnelle entre les composants, permettant la mise à jour de l'état global à partir des interactions utilisateur dans chaque étape du workflow. Cette architecture garantit une expérience utilisateur fluide et une maintenance du code simplifiée.