

Dossier de Projet : KundApp

La Plateforme des Pratiques Énergétiques

Partie 1 : Présentation Générale du Projet

1. Résumé du Projet (Management Summary)

1.1. Contexte et Genèse

Le projet KundApp est né du constat que les professionnels des pratiques énergétiques (kundalini, soins énergétiques, yoga, etc.) sont contraints d'utiliser une multitude d'outils numériques fragmentés pour gérer leur activité. Cette dépendance engendre une complexité significative dans la gestion des séances et des participants. Actuellement, la communication s'effectue majoritairement via des applications de messagerie comme WhatsApp, et la planification via des SMS ou e-mails.

De plus, les organisateurs doivent s'adapter aux systèmes de paiement et de planification des salles où ils exercent, tout en gérant la collecte des paiements via des plateformes tierces telles que Stripe. Cette prolifération d'outils n'est pas seulement chronophage, elle génère également une surcharge administrative superflue, source d'erreurs et de perte d'efficacité.

1.2. Pitch du Projet

KundApp est une plateforme web centralisée, dont l'ambition est de devenir le "Doctolib de l'énergétique". Elle vise à faciliter la mise en relation entre professionnels des pratiques énergétiques et participants, en simplifiant considérablement la gestion des séances.

Pour les professeurs, KundApp permet une création et une gestion aisées de leurs événements (dates, lieux, nombre de places). Les utilisateurs, quant à eux, bénéficient d'une interface unique pour consulter les séances disponibles, s'y inscrire via un système de crédits pré-achetés, et suivre leur historique de participation. L'objectif principal est d'offrir une expérience fluide et intégrée, afin de libérer du temps pour l'essentiel : la pratique elle-même.

1.3. Public Cible

L'application s'adresse à deux profils d'utilisateurs principaux :

- Les professionnels indépendants du secteur de l'énergétique : professeurs de yoga, facilitateurs de kundalini, praticiens de soins énergétiques, etc.
- Leurs clients : des participants réguliers ou occasionnels à ces séances.

2. Expression des Besoins (Cahier des Charges)

2.1. Problématique Détaillée

L'analyse du besoin a mis en lumière plusieurs points de friction majeurs dans le processus de gestion actuel :

- Fragmentation des outils : la dépendance à des outils multiples (WhatsApp, e-mails, SMS, plateformes de paiement) complique considérablement le suivi des interactions, augmentant ainsi les risques d'oublis et de doubles réservations.
- Gestion manuelle des inscriptions : le processus de suivi des participants, effectué manuellement, est chronophage et manque de fiabilité, particulièrement pour les sessions très demandées.
- Difficulté à créer une communauté : les clients peinent à trouver des facilitateurs locaux ou en ligne, et les facilitateurs rencontrent des difficultés à attirer de nouveaux clients. La plateforme doit servir de catalyseur pour rassembler cette communauté.

2.2. Solution Apportée par KundApp

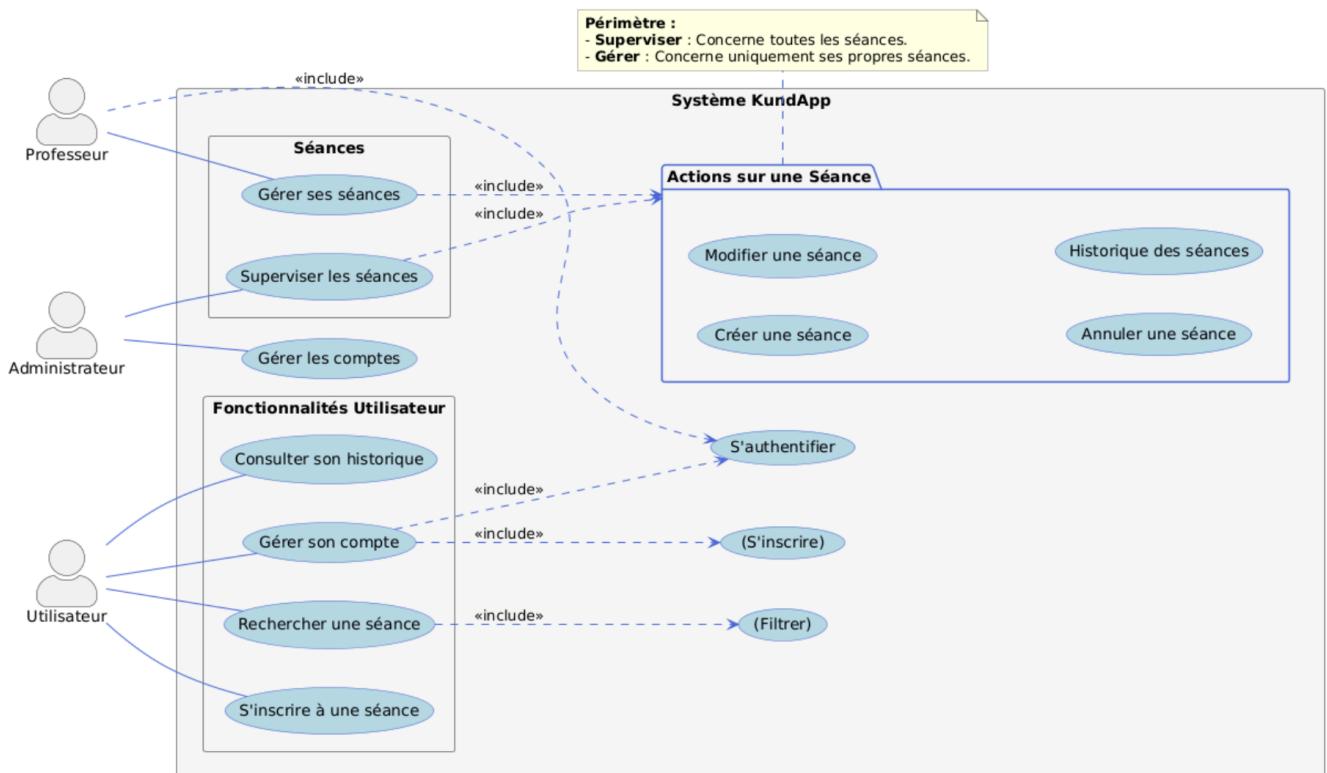
KundApp répond à ces problématiques en proposant :

- Centralisation : toutes les informations (créneaux, lieux, descriptifs, participants) sont regroupées sur une seule et même plateforme.
- Automatisation : le processus d'inscription et de suivi est entièrement automatisé. Cela inclut des fonctionnalités telles que les liens Google Maps, les liens Zoom, la gestion des paiements via Stripe et la gestion automatisée des inscriptions/annulations.
- Optimisation : l'intégration future d'une liste d'attente permettra de maximiser le taux de remplissage des cours en comblant instantanément les places devenues disponibles.

2.3. Spécifications Fonctionnelles (User Stories)

- En tant qu'administrateur (Admin), je veux :
 - Gérer les comptes utilisateurs (clients et professeurs).
 - Superviser l'ensemble des séances créées.
- En tant que professeur (Teacher), je veux :
 - Me connecter à mon compte.
 - Créer, modifier et annuler mes propres séances.
 - Consulter l'historique de mes cours et la liste des participants.
- En tant qu'utilisateur (Client), j'ai besoin de :
 - M'inscrire et me connecter à la plateforme.
 - Consulter la liste des séances disponibles et les filtrer.
 - M'inscrire à une séance en utilisant mes crédits.
 - Consulter l'historique de mes participations.

2.4. Diagramme de Cas d'Utilisation



3. Maquettes et Parcours Utilisateur

3.1. Maquettes des Écrans Clés

Connexion

Email

Mot de passe

SE CONNECTER

Nouveau sur KundApp? Inscrivez-vous

This is a wireframe of the login form. It features a header with the Kundalini logo, followed by a title 'Connexion'. Below the title are two input fields: 'Email' and 'Mot de passe'. A large red button labeled 'SE CONNECTER' contains a yellow arrow icon. At the bottom of the form is a link 'Nouveau sur KundApp? Inscrivez-vous'.

Formulaire d'authentification

Créer votre compte

Rejoignez notre communauté d'énergie positive

Prénom

Nom

Email

Mot de passe

CRÉER MON COMPTE

Déjà membre? Connectez-vous.

This is a wireframe of the sign-up form. It starts with a header featuring the Kundalini logo and the title 'Créer votre compte'. Below this is a sub-instruction 'Rejoignez notre communauté d'énergie positive'. The form includes four input fields: 'Prénom' and 'Nom' side-by-side, followed by 'Email' and 'Mot de passe'. A large red button labeled 'CRÉER MON COMPTE' with a yellow arrow icon is at the bottom. A link 'Déjà membre? Connectez-vous.' is located at the very bottom.

Formulaire d'inscription

Projet : KundApp

KundApp Admin Sessions

4 crédits user@gmail.com DÉCONNEXION

Sessions
Découvrez et gérez vos séances

Toutes les catégories EN LIGNE EN PRÉSENTIEL

RECHERCHER PAR PROFESSEUR

SESSIONS DISPONIBLES MES PROCHAINES SÉANCES HISTORIQUE

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

S'INSCRIRE ⓘ

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

S'INSCRIRE ⓘ

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

S'INSCRIRE ⓘ

© 2025 KundApp Admin. Tous droits réservés.

Vue User - séances disponibles

KundApp Admin Sessions

4 crédits user@gmail.com DÉCONNEXION

Sessions
Découvrez et gérez vos séances

Toutes les catégories EN LIGNE EN PRÉSENTIEL

RECHERCHER PAR PROFESSEUR

SESSIONS DISPONIBLES MES PROCHAINES SÉANCES HISTORIQUE

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

SE DÉSINSCRIRE ⓘ Inscrit ✓

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

SE DÉSINSCRIRE ⓘ Inscrit ✓

MEDITATION
Jeudi 21 aout 2025
14:00 - 15h30

Prof Jenna
En ligne
0 participants (5 places)

SE DÉSINSCRIRE ⓘ Inscrit ✓

© 2025 KundApp Admin. Tous droits réservés.

Vue User - prochaines séances

Projet : KundApp

KundApp Admin Sessions

4 crédits user@gmail.com DÉCONNEXION

Sessions

Découvrez et gérez vos séances

Toutes les catégories EN LIGNE EN PRÉSENTIEL RECHERCHER PAR PROFESSEUR

SESSIONS DISPONIBLES MES PROCHAINES SÉANCES HISTORIQUE

MEDITATION	Prof Jenna	Annuler	Informations
Jeu 21 aout 2025 14:00 - 15h30	■ En ligne 0 participants (5 places)		
MEDITATION	Prof Jenna	Annuler	Informations
Jeu 21 aout 2025 14:00 - 15h30	■ En ligne 0 participants (5 places)		
MEDITATION	Prof Jenna	Annuler	Informations
Jeu 21 aout 2025 14:00 - 15h30	■ En ligne 0 participants (5 places)		

© 2025 KundApp Admin. Tous droits réservés.

Vue User - historique

KundApp Admin

admin@gmail.com DÉCONNEXION

Administration KundApp

Gestion des utilisateurs et du système

Recherche

TEACHER CLIENTS SESSIONS

Liste des clients

NOUVEAU CLIENT

Prénom	Nom	Email	Téléphone	Statut	Crédits	Crée le	Action
John	Doe	user@gmail.com	0629712167	Actif	4	21/11/1990	MODIFIER
John	Doe	user@gmail.com	0629712167	Actif	4	21/11/1990	MODIFIER

© 2025 KundApp Admin. Tous droits réservés.

Vue Admin - liste des clients

Projet : KundApp

The screenshot shows the 'Administration KundApp' page. At the top, there's a navigation bar with the logo 'KundApp Admin', the email 'admin@gmail.com', and a 'Déconnexion' button. Below the header, a search bar with a play icon and the word 'Recherche' is visible. A breadcrumb menu at the top left includes 'TEACHER', 'CLIENTS', and 'SESSIONS'. The main content area is titled 'Liste des teachers' and contains a table with two rows of teacher data. Each row includes columns for 'Prénom', 'Nom', 'Email', 'Téléphone', 'Statut', 'Crée le', and 'Action'. The first teacher has an 'Actif' status and was created on '21/11/1990'. The second teacher also has an 'Actif' status and was created on '21/11/1990'. Both rows have a 'MODIFIER' button. A red 'NOUVEAU TEACHER' button is located at the top right of the table area. At the bottom of the page, a copyright notice reads '© 2025 KundApp Admin. Tous droits réservés.'

Vue Admin - liste des professeurs

The screenshot shows the 'Administration KundApp' page. The layout is identical to the teacher list, with the 'SESSIONS' tab selected in the breadcrumb menu. The main content area is titled 'Liste des sessions' and contains a table with five rows of session data. Each row includes columns for 'Date/Heure', 'Lieu', 'Teacher', 'Sujet', 'Statut', 'Participant', and 'Action'. The sessions listed are: 1) Date: 11/09/2025, Time: 12:00, Lieu: En ligne, Teacher: Jenna Watkins, Sujet: YOGA, Statut: Annulé, Participants: 2 participants (with a cursor icon over the number), Action: VOIR. 2) Date: 11/09/2025, Time: 12:00, Lieu: Omvida 75015, Teacher: Jenna Watkins, Sujet: YOGA, Statut: Programmée, Participants: 2 participants, Action: MODIFIER. 3) Date: 11/09/2025, Time: 12:00, Lieu: En ligne, Teacher: Jenna Watkins, Sujet: YOGA, Statut: Annulé, Participants: 2 participants, Action: VOIR. 4) Date: 11/09/2025, Time: 12:00, Lieu: Omvida 75015, Teacher: Jenna Watkins, Sujet: YOGA, Statut: Programmée, Participants: 2 participants, Action: MODIFIER. 5) Date: 11/09/2025, Time: 12:00, Lieu: En ligne, Teacher: Jenna Watkins, Sujet: YOGA, Statut: Annulé, Participants: 2 participants, Action: VOIR. A copyright notice '© 2025 KundApp Admin. Tous droits réservés.' is at the bottom.

Vue Admin - liste des séances

Projet : KundApp

The screenshot shows the KundApp Admin dashboard. At the top, there is a header with the logo 'KundApp Admin' and the word 'Accueil'. On the right side of the header, there are links for 'teacher@gmail.com' and 'DÉCONNEXION'. Below the header, a pink banner says 'Bienvenue sur votre espace KundApp' and 'Consultez et gérez vos prochaines séances'. The main content area has three tabs: 'MES PROCHAINES SÉANCES' (which is selected), 'HISTORIQUE', and 'CRÉER UNE SÉANCE'. Under 'MES PROCHAINES SÉANCES', there are three entries for 'MEDITATION' sessions. Each entry includes the date ('Jeudi 21 aout 2025'), time ('14:00 - 15h30'), location ('En ligne'), number of participants ('0 participants (5 places)'), and a 'MODIFIER' button.

© 2025 KundApp Admin. Tous droits réservés.

Vue teacher

A modal window titled 'Confirmer votre inscription' is displayed. At the top, it says 'MEDITATION'. The details of the session are listed: Date: Jeudi 21 aout 2015, Professeur: Jenna; Horaire: 14:00 - 15:55, Type: En ligne. A blue box indicates a 'Cout : 1 crédit'. A yellow box contains the note: 'Toute annulation doit avoir lieu 48h minimum avant le début de la séance.' At the bottom, there are two buttons: 'ANNULER' and 'CONFIRMER'.

Vue User - modale - inscription

Projet : KundApp

Modifier le User

Prénom *	Nom *
John	John
Email *	Téléphone *
John@gmail.com	0629712167
Date de naissance *	
16/12/1991	
Crédits	
4	
Adresse	
Rue	Ville
567 chemin du Vieux Reynier	Houston
Code Postal	Pays
83500	USA

[SUPPRIMER CE CLIENT](#) [DÉSACTIVER CE CLIENT](#) [ANNULER](#) [CONFIRMER](#)

Vue Admin - modale - Créer/modifier le user

Modifier le Teacher

Prénom *	Nom *
John	John
Email *	Téléphone *
John@gmail.com	0629712167
Date de naissance *	
16/12/1991	
Biographie	
<input type="text"/>	
Adresse	
Rue	Ville
567 chemin du Vieux Reynier	Houston
Code Postal	Pays
83500	USA

[SUPPRIMER CE TEACHER](#) [DÉSACTIVER CE TEACHER](#) [ANNULER](#) [CONFIRMER](#)

Vue Admin - modale - Créer/modifier le teacher

Projet : KundApp

Créer une séance

Type de cours *
Pilates ...

Description *

Type de séance
 En présentiel En ligne
Vous pouvez changer de type de séances. Les champs spécifiques à l'ancien type seront effacés.

Lieu de la séance

Nom de la salle/ Lieu *
Omvida

Code Postal *
83500

Lien Google Map *
<https://www.google.com/maps>

Dates et horaires

Date et heure de début*
22/08/2024 14h00

Durée *
1h30

Participants et crédits

Nombre de places *
12

Crédits requis *
1

Entre 1 et 50 participants

Matériel

Les participants doivent apporter leur tapis
Cochez cette case si les participants doivent venir avec leur propre équipement

Participants inscrits à la séance

Aucun participant inscrit ...

ANNULER **ENREGISTRER**

Vue teacher - modale - Créer une séance

Projet : KundApp

Consultation de session annulée

Type de cours *
Yoga

Description *

Type de séance
 En présentiel En ligne

Lien Zoom *
<https://zoom.us/fr/signin#/login>

Lien de la réunion Zoom pour se connecter à la session en ligne

Dates et horaires

Date et heure de début*
22/08/2024 14h00  Durée *
1h30

Participants et crédits

Nombre de places *
12 Crédits requis *
1

Entre 1 et 50 participants

Impossible de modifier : des participants se sont déjà inscrits à la séance. Annulez cette session et créez en une autre pour modifier le nombre de crédits.

Participants inscrits à la séance

Prénom	Nom	Email
John	Doe	user@gmail.com
John	Doe	user@gmail.com
John	Doe	user@gmail.com



Vue teacher/admin - modale - consulter séance (cas annulée)

Projet : KundApp

Modifier une séance

Type de cours *
Pilates ...

Description *

Type de séance
 En présentiel En ligne
Vous pouvez changer de type de séances. Les champs spécifiques à l'ancien type seront effacés.

Lieu de la séance

Nom de la salle/ Lieu * Code Postal *
Omvida 83500

Lien Google Map *
https://www.google.com/maps

Dates et horaires

Date et heure de début* Durée *
22/08/2024 14h00 1h30

Participants et crédits

Nombre de places * Crédits requis *
12 1
Entre 1 et 50 participants Impossible de modifier : des participants se sont déjà inscrits à la séance. Annulez cette session et créez en une autre pour modifier le nombre de crédits.

Matériel

Les participants doivent apporter leur tapis
Cochez cette case si les participants doivent venir avec leur propre équipement

Participants inscrits à la séance

1 participant(s) inscrit(s)

[ANNULER CETTE SÉANCE](#) [FERMER](#) [SAUVEGARDER](#)

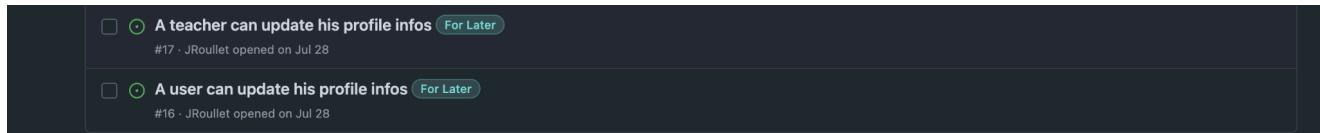
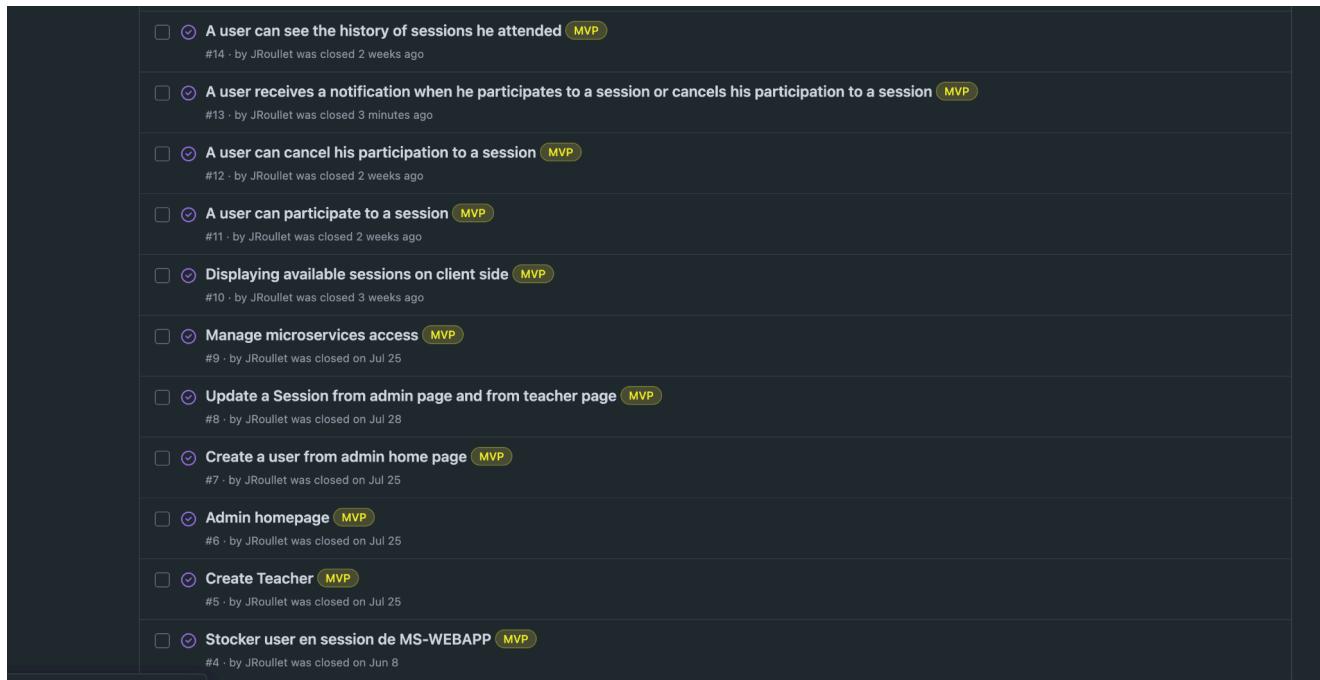
Vue teacher/admin - modale - modifier la séance

Partie 2 : Conception et Architecture Technique

4. Gestion de Projet

4.1. Méthodologie et Organisation

Ce projet a été mené individuellement en adoptant une approche inspirée des principes Agile. L'organisation du développement a reposé sur la création d'un backlog de User Stories, essentielles pour structurer les tâches et atteindre efficacement le Minimum Viable Product (MVP). La gestion de ce backlog et le suivi de l'avancement ont été réalisés à l'aide des Issues GitHub.



4.2. Gestion de Versions avec Git et GitHub

Ayant œuvré en solitaire, la gestion des versions a été simplifiée. L'utilisation de Git, avec un maintien constant sur la branche main, a permis un suivi rigoureux des modifications et des sauvegardes régulières. Le dépôt centralisé sur GitHub a assuré la sécurité et la pérennité du code.

En ce qui concerne la gestion des fichiers, le projet global est organisé autour d'un dossier KundApp, qui dispose de son propre répertoire GitHub. De plus, un dossier distinct, kundapp_config, gère les configurations de chaque microservice. Ces configurations sont centralisées et récupérées via GitHub par le Config Server.

5. Architecture Logicielle

5.1. Choix d'une Architecture Microservices

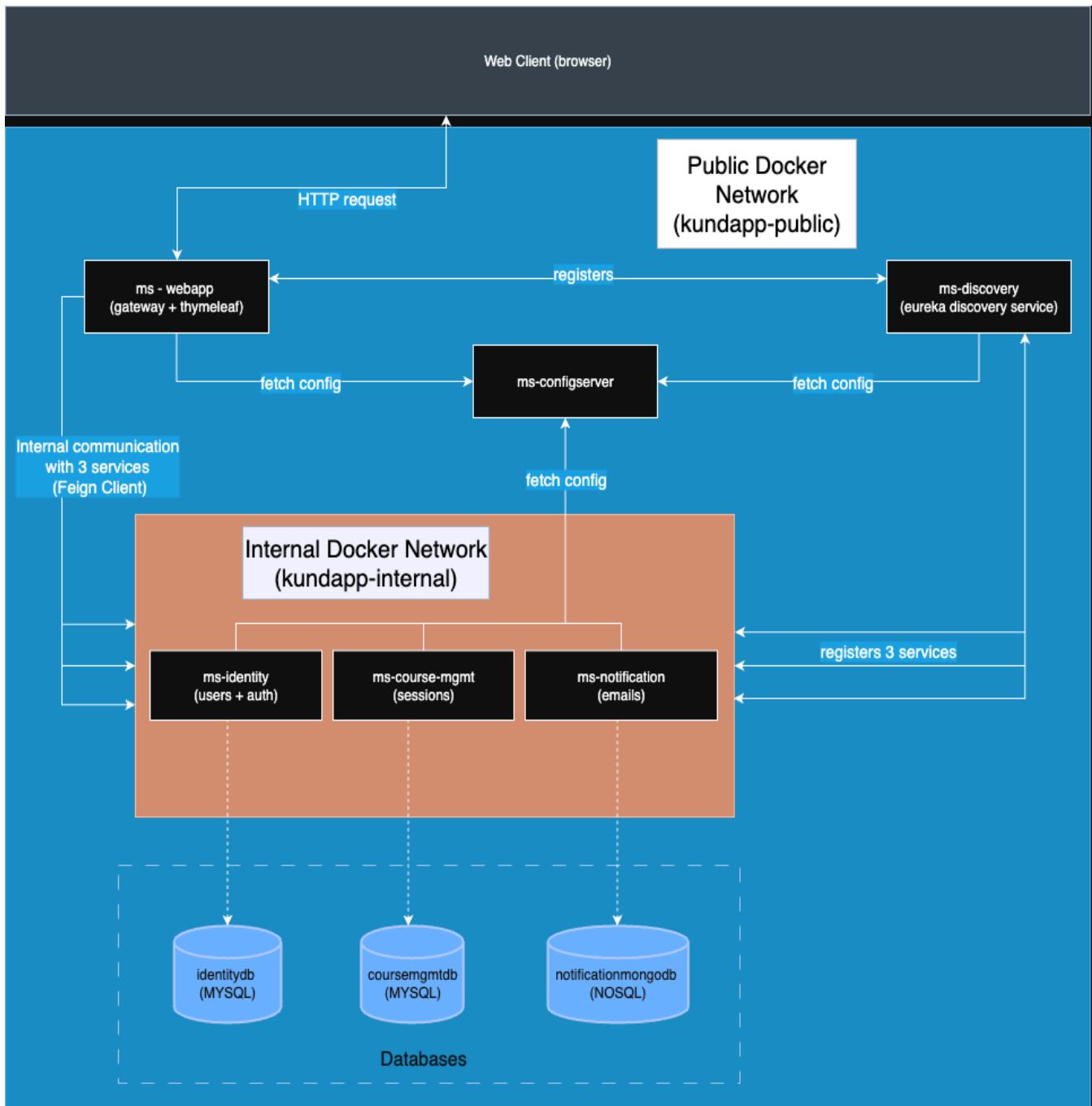
L'application KundApp est construite sur une architecture microservices. Ce choix a été motivé par la nécessité de découpler les différentes responsabilités fonctionnelles, favorisant ainsi la maintenabilité, la scalabilité et la résilience du système. Chaque service possède sa propre base de données et peut être développé, déployé et mis à l'échelle indépendamment des autres.

Les principaux microservices sont :

1. ms-configserver : centralise la gestion des configurations (via Spring Cloud Config Server).
2. ms-discovery : facilite la découverte des microservices (utilisant Netflix Eureka).
3. ms-webapp : agit comme une passerelle (“gateway”), il est responsable de l'orchestration et de la gestion du service d'interface utilisateur (via Thymeleaf).
4. ms-identity : gère les identités, l'authentification et les autorisations des utilisateurs (stockage MySQL).
5. ms-coursemgmt : prend en charge la gestion des cours, des inscriptions et des participants (stockage MySQL).
6. ms-notification : s'occupe de la génération de notifications (stockage MongoDB).

Cette architecture a été délibérément conçue pour être évolutive. Elle pourra ainsi accueillir de manière agile de nouveaux services critiques à l'avenir, comme un microservice de paiement ou de gestion des abonnements, sans impacter le fonctionnement des modules existants, démontrant ainsi toute la flexibilité et la résilience de notre approche.

5.2. Schéma d'Architecture Détaillé



Architecture microservice de l'application

5.3. Les Piliers de l'Infrastructure

5.3.1. ms-config-server : Centralisation de la Configuration

Le Spring Cloud Config Server est un composant stratégique qui externalise la configuration de tous les microservices. Au démarrage, chaque service contacte le Config Server pour récupérer ses propriétés (URL de la base de données, adresses des autres services, etc.). Cette approche offre des avantages majeurs :

- Gestion centralisée : elle élimine la nécessité de modifier les fichiers application.yml pour chaque service, établissant ainsi une source unique de vérité (généralement un dépôt Git) qui centralise l'intégralité de la configuration.
- Sécurité améliorée : les informations sensibles (secrets, mots de passe) sont stockées et gérées en un seul endroit sécurisé.
- Dynamisme : il est possible de rafraîchir la configuration des services à chaud, sans avoir besoin de les redémarrer, ce qui est crucial pour la maintenance en production.

5.3.2. ms-discovery (Eureka) : Communication et Résilience

Dans une architecture distribuée, les services nécessitent un mécanisme pour se localiser mutuellement. Netflix Eureka assure cette fonction d'annuaire dynamique (Service Registry).

- Enregistrement et découverte : Chaque microservice s'enregistre auprès d'Eureka lors de son démarrage. Pour établir une communication, un service ne procède pas à un codage en dur de l'adresse d'un autre ; il interroge Eureka, par exemple : « Où se situe ms-identity ? », et Eureka lui communique l'adresse IP et le port actuels.
- Contrôles de santé et résilience : les services transmettent un « heartbeat » (signal de vie) à Eureka à intervalles réguliers. Si un service devient indisponible (panne, problème réseau), Eureka cesse de recevoir son heartbeat et le retire de son registre après un certain délai. Les autres services ne tenteront plus de le contacter, ce qui prévient les erreurs en cascade et confère une résilience accrue au système global.
- Équilibrage de charge côté client : si plusieurs instances d'un même service sont déployées pour gérer la charge, Eureka en a connaissance. Un client (tel que ms-webapp) peut alors répartir intelligemment ses requêtes entre les différentes instances disponibles, garantissant un équilibrage de charge efficace.

5.4. Stack Technique

- *Frontend : Thymeleaf, HTML5/CSS3, Bootstrap (MDB), JavaScript*
- *Backend : Java 17 & Spring Boot 3, Spring Cloud, Maven, Architecture 3-tiers*
- *Bases de Données : Approche Polyglotte (MySQL, MongoDB), JPA (Hibernate).*
- *DevOps : Docker & Docker Compose, Jib.*



6. Conception des Bases de Données

La conception des données est un pilier de l'architecture microservices de KundApp. Chaque service métier dispose de sa propre base de données, assurant une autonomie et un découplage rigoureux. Cette approche, connue sous le nom de "Base de données par Service" (Database per Service), prévient les goulots d'étranglement et permet à chaque service d'évoluer indépendamment.

6.1. Maîtrise du Schéma de Données : L'approche "Code-First" contrôlée

Bien que le projet utilise un ORM (JPA/Hibernate) pour la couche d'accès aux données, une décision d'architecture clé a été de ne pas déléguer la création du schéma de base de données à Hibernate (spring.jpa.hibernate.ddl-auto=none) et de le reprendre à notre compte.

Cette approche a été choisie pour garantir une maîtrise totale et une prévisibilité parfaite de la structure des données. Les tables sont définies explicitement via des scripts 01-schema.sql versionnés avec le code source de chaque microservice. Cette méthode présente plusieurs avantages majeurs :

- Stabilité : elle évite les effets de bord potentiels des mises à jour automatiques d'Hibernate, qui peuvent être imprévisibles en production.
- Clarté : le schéma de la base de données est le reflet exact et unique du Modèle Physique de Données (MPD) validé.
- Séparation des responsabilités : l'ORM est utilisé pour ce qu'il fait de mieux, le mapping objet-relationnel, tandis que la structure de la base de données reste sous un contrôle strict, comme il se doit.

6.2. Modélisation des Données Relationnelles (MySQL)

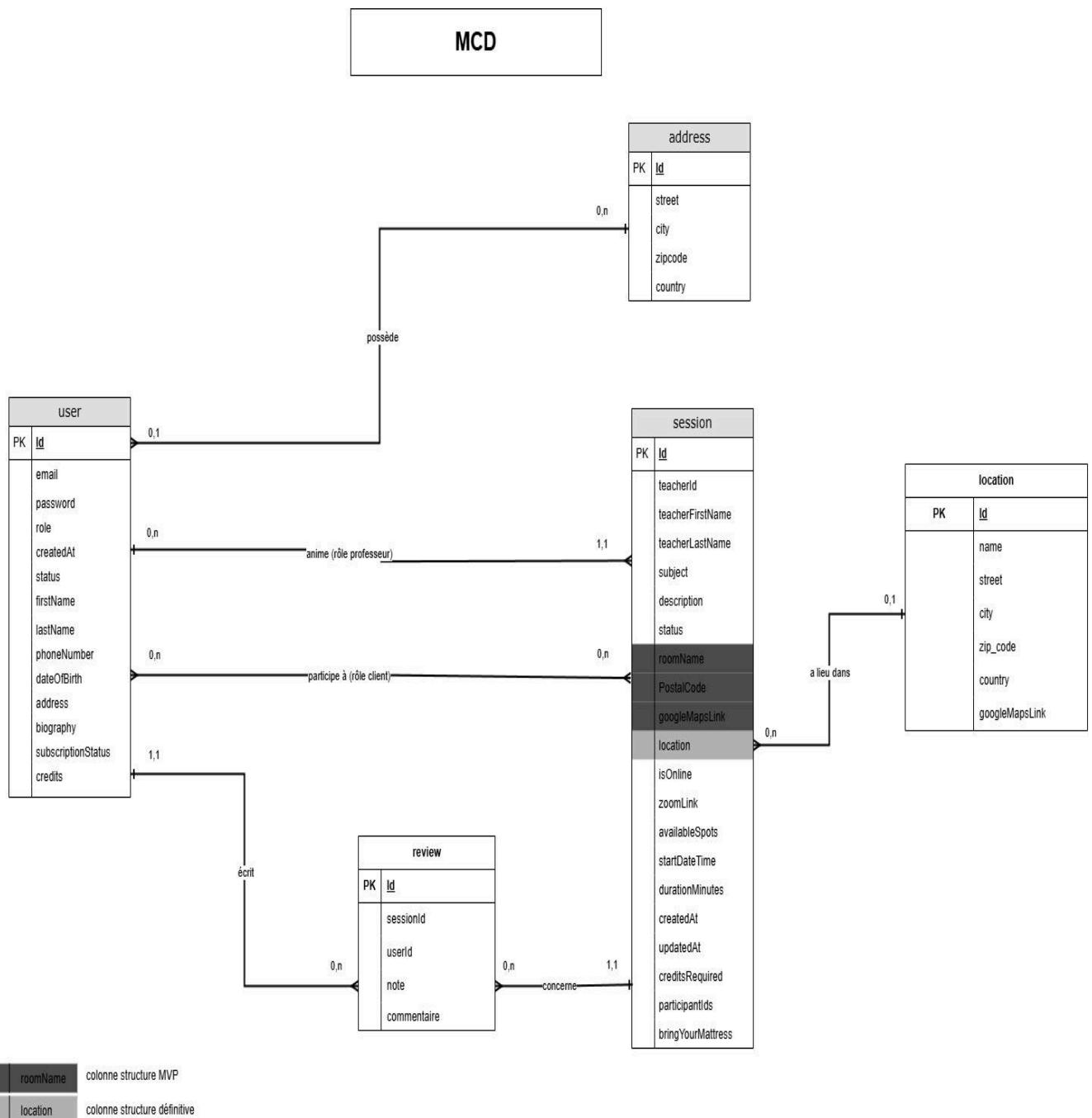
Pour les services ms-identity (garant de l'identité des utilisateurs, de leurs rôles et profils) et ms-coursemgmt (qui gère le cœur de l'application : cours, inscriptions, participants), le choix d'une base de données relationnelle comme MySQL s'est imposé.

Cette décision est motivée par la nature même des données gérées :

- Intégrité et Cohérence : la nécessité de garantir une intégrité référentielle forte est primordiale. Une inscription, par exemple, doit impérativement correspondre à un utilisateur et à une session valides.
- Nature Transactionnelle : les opérations sur les utilisateurs et les cours sont transactionnelles et nécessitent un modèle de données structuré et fiable.

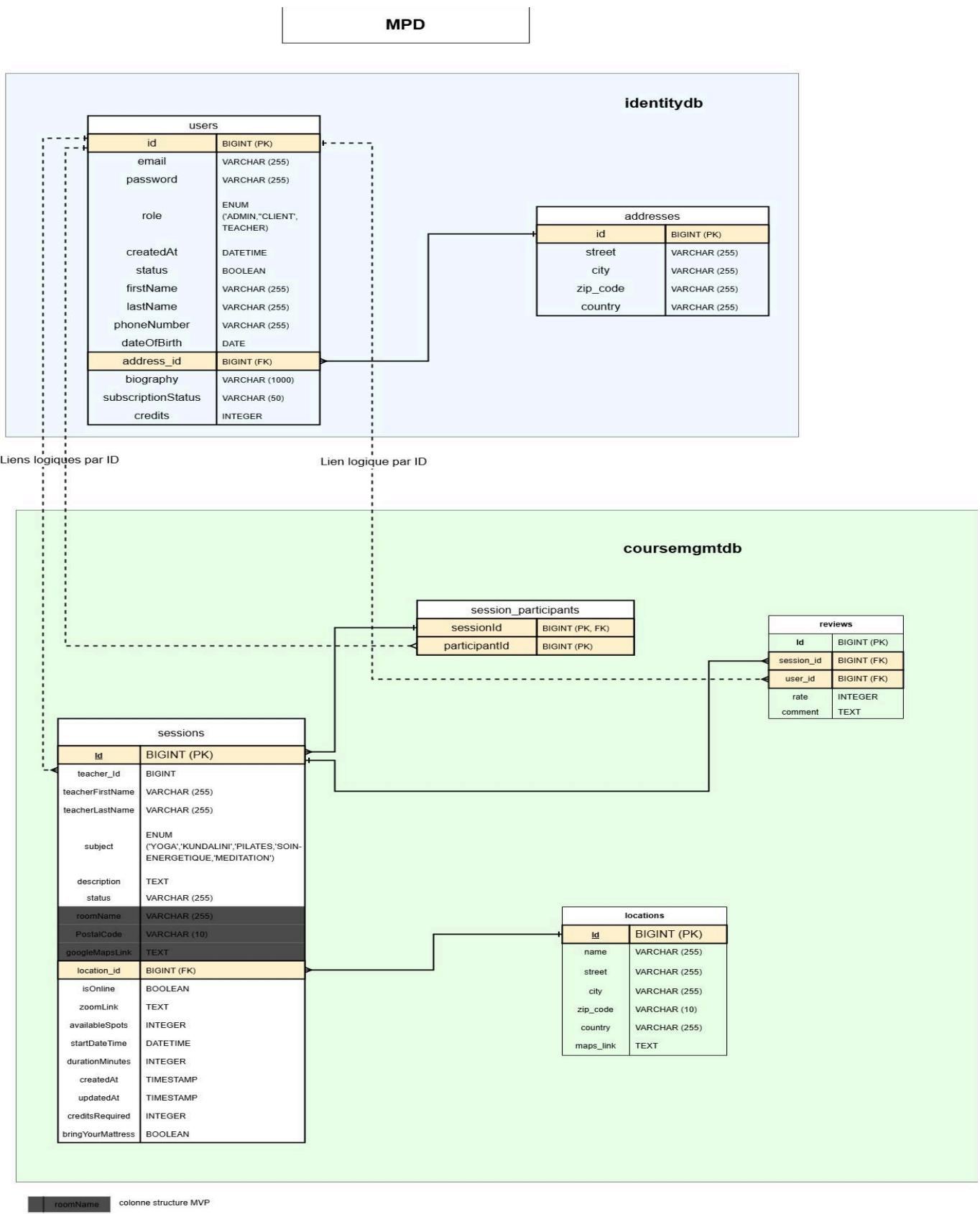
Les diagrammes ci-dessous représentent la vision cible de l'architecture des données, conçue pour l'évolution future du projet. Dans le cadre de ce MVP, certaines entités prévues (comme Review ou Location) ont été modélisées mais leur implémentation a été volontairement reportée à une prochaine itération.

Projet : KundApp



Modèle Conceptuel des Données

Projet : KundApp



Modèle Physique des Données

6.3. Service ms-notification (Base de données NoSQL - MongoDB)

Pour le service de notifications, une approche radicalement différente a été adoptée avec le choix de MongoDB, une base de données NoSQL orientée document.

6.3.1. Justification et Structure du Document

Le choix de MongoDB repose sur deux critères principaux :

- **Flexibilité (Schéma Dynamique)** : Les notifications sont par nature événementielles et hétérogènes. Une notification d'inscription, d'annulation ou de modification de cours n'emporte pas les mêmes données. Un schéma non rigide permet de stocker ces différents types d'événements sans avoir à altérer la structure de la base, ce qui serait lourd et coûteux avec un modèle relationnel.
- **Performance en Écriture** : Le service de notification est destiné à recevoir de nombreux événements en rafale. MongoDB excelle dans les opérations d'écriture rapides, ce qui garantit que le système peut absorber des pics de charge sans dégradation.

La structure d'une notification est modélisée sous la forme d'un document BSON (format JSON), ce qui permet de stocker toutes les informations relatives à un événement de manière atomique (voir exemple à la page suivante).

6.3.2. Un Service Volontairement Découplé

Un point essentiel de l'architecture de ce service est son découplage total des autres services métier. ms-notification n'a aucune connaissance directe des entités User de ms-identity ou Session de ms-coursemgmt.

Il expose une API simple qui attend des données brutes (un userId, un email, des détails de session, etc.). C'est le ms-webapp qui agit en tant qu'orchestrateur : c'est lui qui a la responsabilité de récupérer les informations nécessaires auprès des services concernés (ms-identity, ms-coursemgmt) avant de forger la requête et d'appeler ms-notification.

Cette absence de couplage fort est un avantage stratégique : si demain le service ms-identity change, l'impact sur ms-notification sera nul tant que le contrat d'API est respecté. C'est une illustration parfaite de la résilience et de la maintenabilité offertes par une architecture microservices bien conçue.

Exemple de document de notification :

JSON

```
{  
  "_id": "64c9a8b1...",  
  "recipient": {  
    "userId": "13",  
    "firstName": "Remy",  
    "lastName": "Martin",  
    "email": "remy.martin@gmail.com"  
  },  
  "eventType": "SESSION_ENROLLED_STUDENT",  
  "status": "SENT",  
  "session": {  
    "id": 42,  
    "name": "Kundalini",  
    "startTime": "2024-09-15T18:00:00Z",  
    "teacherName": "Jenna Watkins"  
  },  
  "createdAt": "2024-08-15T10:00:00Z",  
  "updatedAt": "2024-08-15T10:00:05Z",  
  "attempts": 1  
}
```

7. Catalogue des API et Communication Inter-Services

La communication au sein de l'écosystème KundApp repose sur des contrats d'API clairs. Le ms-webapp, agissant en tant que Gateway, expose une série d'endpoints RESTful qui sont consommés par le client web. En interne, les services communiquent de manière synchrone via des clients Feign, assurant une intégration fluide et résiliente.

7.1. Contrat d'API (Endpoints de ms-webapp)

Verbe HTTP	Route	Rôle Requis	Description
Authentification			
POST	/signup	Public	Crée un nouveau compte pour un CLIENT.
POST	/signin	Public	Authentifie un utilisateur et ouvre une session.
POST	/logout	Authentifié	Déconnecte l'utilisateur.
Gestion par les Administrateurs			
GET	/admin/sessions	ADMIN	Affiche toutes les sessions de la plateforme.
POST	/admin/teachers	ADMIN	Crée un nouveau compte TEACHER.
GET	/admin/teachers	ADMIN	Récupère la liste de tous les enseignants.
GET	/admin/users	ADMIN	Récupère la liste de tous les clients.
POST	/admin/users/{id}/credits	ADMIN	Ajoute des crédits à un utilisateur spécifique.
Gestion par les Enseignants			
GET	/teacher/sessions	TEACHER	Affiche les sessions créées par le professeur connecté.
POST	/teacher/sessions	TEACHER	Crée une nouvelle session pour le professeur.
PUT	/teacher/sessions/{id}	TEACHER	Met à jour une session appartenant au professeur.
POST	/teacher/sessions/{id}/cancel	TEACHER	Annule une session et rembourse les participants.

Actions des Clients			
GET	/client/sessions	CLIENT	Affiche les sessions auxquelles le client est inscrit.
GET	/client/sessions/available	CLIENT	Affiche toutes les sessions disponibles pour l'inscription.
POST	/client/sessions/{id}/register	CLIENT	Inscrit le client connecté à une session.
POST	/client/sessions/{id}/unregister	CLIENT	Désinscrit le client d'une session.

7.2. Communication Inter-Services (Clients Feign)

En arrière-plan, ms-webapp orchestre les appels vers les autres microservices au moyen de clients Feign. Cette approche déclarative simplifie considérablement la communication HTTP interne.

- **IdentityFeignClient** : ce client est employé pour toutes les opérations relatives aux utilisateurs (création, récupération, gestion des crédits) en assurant la communication avec ms-identity.
- **CourseManagementFeignClient** : ce client est utilisé pour interagir avec ms-course-mgmt afin de gérer les sessions (création, mise à jour, inscription des participants).
- **NotificationFeignClient** : ce client est invoqué pour acheminer des requêtes de notification asynchrones à ms-notification après des actions significatives (inscription, annulation, etc.).

7.3. Documentation d'API avec Swagger/OpenAPI

Pour garantir la clarté des contrats d'API et faciliter la collaboration entre le développement frontend et les différents microservices, le projet intègre une documentation interactive.

Cette documentation est générée automatiquement en s'appuyant sur la bibliothèque (dépendance) `springdoc-openapi-starter-webmvc-ui`. Cet outil puissant s'intègre à Spring Boot et analyse les contrôleurs de l'application pour produire une spécification OpenAPI 3 normalisée.

Les avantages de cette approche sont multiples :

- Source de vérité unique : la documentation est toujours synchronisée avec le code. Toute modification d'un endpoint ou d'un DTO est immédiatement reflétée.
- Clarté et interactivité : chaque microservice expose une interface web *Swagger UI*. Elle permet à tout développeur de visualiser les endpoints disponibles, leurs verbes *HTTP*, les paramètres requis, les DTO attendus en entrée/sortie et les codes de réponse possibles.
- Facilité de test et de débogage : l'interface *Swagger UI* offre la possibilité de tester chaque route directement depuis le navigateur, en envoyant des requêtes et en inspectant les réponses.

Cet outil est un élément essentiel de l'usine logicielle du projet. Il favorise un développement plus rapide et une maintenance simplifiée en réduisant les frictions inter-équipes. Le code est destiné à être lu et compris par tous, et non pas uniquement par son auteur. De plus, la documentation sert de référence sur les évolutions et l'état des endpoints, même en cas de changement au sein de l'équipe projet.

Projet : KundApp

The screenshot shows the Swagger UI interface for the `admin-user-controller`. A GET request is defined for the endpoint `/admin/users/{id}`. The parameter `id` is set to `5`. The "Responses" section displays the JSON response body, which includes fields like `name`, `email`, `firstName`, `lastName`, `phoneNumber`, `dateOfBirth`, `address`, `city`, `zipcode`, `country`, `role`, `status`, `createdAt`, and `credit`. Below the response body, the "Response headers" section lists standard HTTP headers such as `cache-control`, `connection`, `date`, `expires`, `pragma`, `x-content-type-options`, `x-frame-options`, and `x-xss-protection`.

The screenshot shows the Swagger UI interface for the `teacher-session-management-controller`. The main title is **KundApp WebApp API** (GAS 2.0). The "Schemas" section lists several data types: `UserParticipantDTO`, `SessionWithParticipantsDTO`, `SessionNoParticipantsDTO`, `Address`, `UserDTO`, and `TeacherDTO`. The "Operations" section lists two GET methods for the `teacher-session-management-controller`: `/teacher/sessions/{sessionId}/participants` and `/teacher/sessions/{sessionId}/details`. Other controllers listed include `user-session-management-controller`, `admin-user-controller`, `admin-teacher-controller`, and `admin-session-management-controller`.

Partie 3 : Réalisations Techniques Approfondies

8. Réalisations Personnelles Significatives

8.1. Gestion robuste des inscriptions et annulations

Le processus d'inscription est critique et doit être atomique pour garantir la cohérence des données. De même, l'annulation d'une séance par un professeur doit garantir la restitution des crédits à tous les participants. J'ai donc conçu un "saga pattern" orchestré par la ms-webapp.

Lors de l'inscription : des validateurs en amont vérifient les prérequis. Le service orchestre ensuite l'appel pour déduire le crédit (ms-identity) puis celui pour inscrire le participant (ms-coursemgmt). En cas d'échec de la deuxième étape, une opération de compensation est appelée pour restituer le crédit.

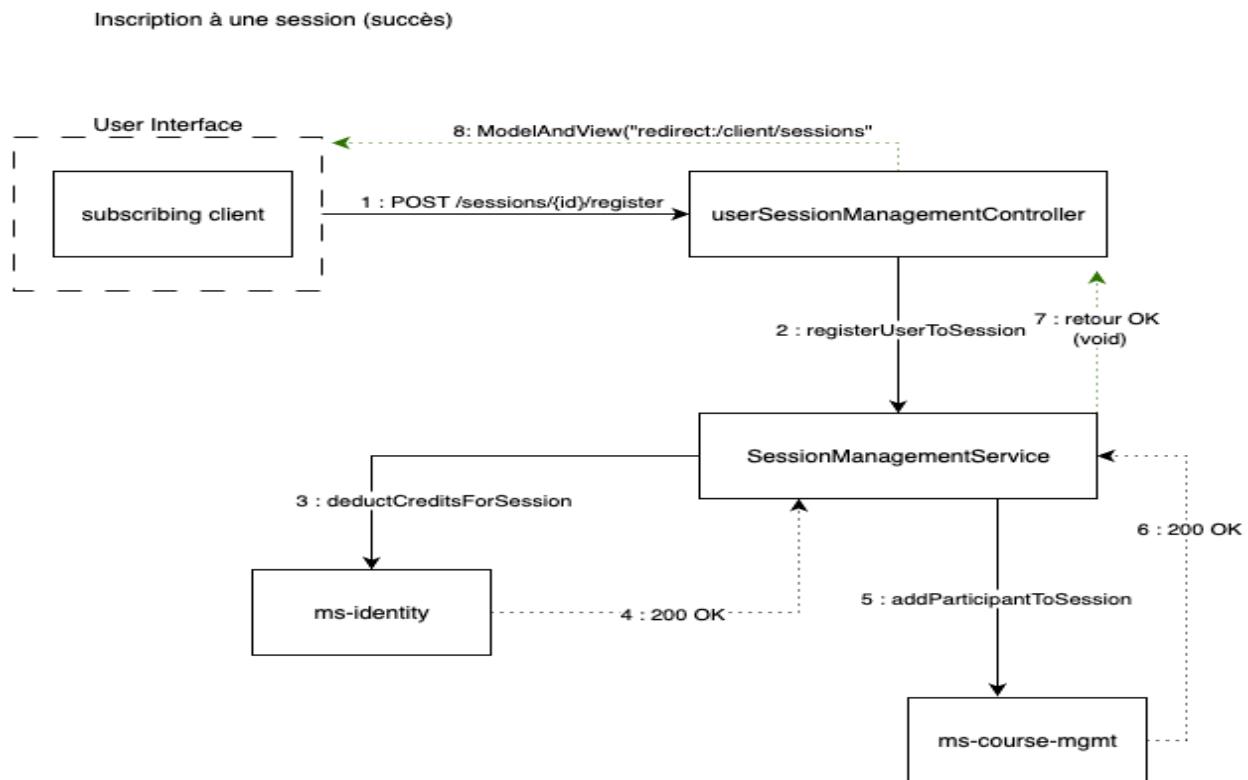
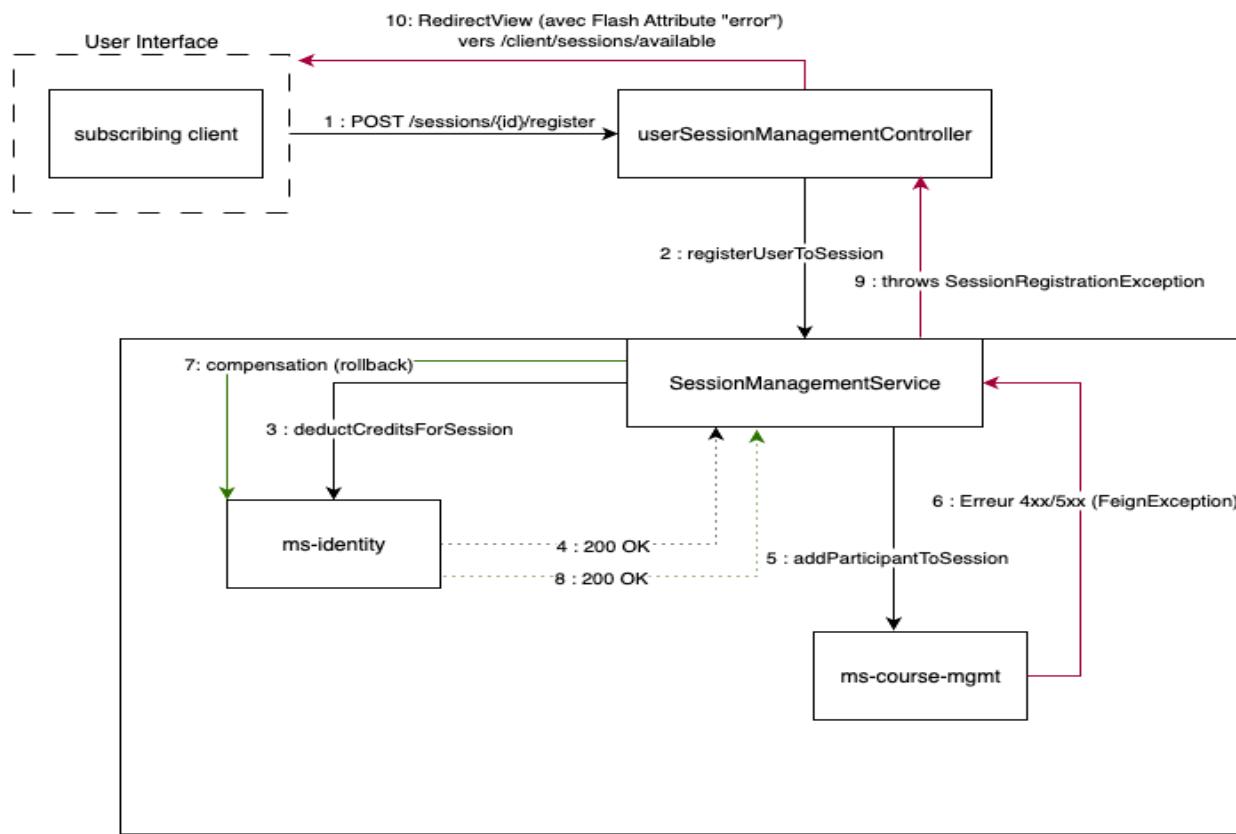


Diagramme de séquence d'inscription à une session (succès)

Projet : KundApp

Inscription à une session (error)



ALT

Diagramme de séquence d'inscription à une session (erreur)

Projet : KundApp

```
public class SessionManagementService {

    @Transactional 1 usage  ± Jacques
    public int registerToSession(Long sessionId) {
        UserDTO user = sessionService.getCurrentUser();
        Long userId = user.getId();

        log.info("Starting session registration for user {} to session {}", userId, sessionId);

        SessionWithParticipantsDTO session = getSessionDetails(sessionId);

        // Validations
        validationService.validateUserHasSufficientCredits(user, session.getCreditsRequired());
        validationService.validateSessionAvailability(session);
        validationService.validateUserNotAlreadyRegistered(session, userId);

        log.info("All validations passed for user {} and session {}", userId, sessionId);

        // Credit deduction
        CreditOperationResponse creditResponse = creditService.deductCredits(userId, sessionId, session.getCreditsRequired());
        log.info("Credits deducted successfully for user {}. Previous: {}, New: {}",
            userId, creditResponse.previousCredits(), creditResponse.newCredits());

        try {
            // Participant registration
            ParticipantOperationResponse participantResponse = addUserToSession(sessionId, userId);
            log.info("User {} successfully registered to session {}. Participants: {}/",
                userId, sessionId, participantResponse.currentParticipantCount(),
                participantResponse.availableSpots());

            // Send notification at successful registration
            try {
                notificationService.sendUserEnrolledNotifications(user.getId(), session);
                log.debug("User enrollment notifications sent successfully");
            } catch (Exception notificationException) {
                log.warn("Failed to send enrollment notifications for user {} and session {}: {}",
                    userId, sessionId, notificationException.getMessage());
            }

            // Returns the new credits after successful registration
            return creditResponse.newCredits();
        } catch (Exception participantException) {
            // Error handling
            log.error("Unexpected failure adding participant despite validations. Rolling back credits for user {} and session {}",
                userId, sessionId, participantException);

            try {
                creditService.refundCredits(userId, sessionId, session.getCreditsRequired());
                log.info("Credits successfully rolled back for user {} after unexpected failure", userId);
            } catch (Exception rollbackException) {
                throw new CreditRollbackFailedException(userId, sessionId, session.getCreditsRequired(), rollbackException);
            }

            throw new SessionRegistrationException("Unexpected failure during registration", participantException);
        }
    }
}
```

Méthode pour l'inscription à une session, dans ms-course-mgmt, visualisation du saga pattern
(java)

Projet : KundApp

```
public class ValidationService {

    public void validateUserHasSufficientCredits(UserDTO user, Integer creditsRequired) { 3 usages ± Jacques
        if (user.getCredits() < creditsRequired) {
            log.warn("User {} has insufficient credits. Available: {}, Required: {}", user.getId(), user.getCredits(), creditsRequired);
            throw new InsufficientCreditsException(user.getId(), user.getCredits(), creditsRequired);
        }
    }

    public void validateSessionAvailability(SessionWithParticipantsDTO session) { 3 usages ± Jacques
        int currentParticipants = session.getParticipantIds() != null ? session.getParticipantIds().size() : 0;
        int maxCapacity = session.getAvailableSpots();

        if (currentParticipants >= maxCapacity) {
            throw new SessionNotAvailableException(session.getId(), currentParticipants, maxCapacity);
        }
    }

    public void validateUserNotAlreadyRegistered(SessionWithParticipantsDTO session, Long userId) { 3 usages ± Jacques
        if (session.getParticipantIds() != null && session.getParticipantIds().contains(userId)) {
            log.warn("User {} is already registered for session {}", userId, session.getId());
            throw new UserAlreadyRegisteredException(userId, session.getId());
        }
    }

    public void validateSessionOwnership(SessionWithParticipantsDTO session, Long teacherId) {...}
    public boolean hasSignificantChanges(SessionWithParticipantsDTO original, SessionWithParticipantsDTO updated) {...}
}
```

Méthodes de validation

```
@Controller ± Jacques
@RequestMapping(@"/client/sessions")
@RequiredArgsConstructor
@Slf4j
public class UserSessionManagementController {

    private final SessionManagementService sessionManagementService;

    @...
    public ResponseEntity<List<SessionNoParticipantsDTO>> getAvailableSessions() {...}

    @...
    public ResponseEntity<List<SessionNoParticipantsDTO>> getUpcomingSessions() {...}

    @...
    public ResponseEntity<List<SessionNoParticipantsDTO>> getHistorySessions() {...}

    | Register current user to a session
    @PostMapping(@"/{sessionId}/register") ± Jacques
    public ModelAndView registerToSession(@PathVariable Long sessionId,
                                           RedirectAttributes redirectAttributes) {
        try {
            int newCredits = sessionManagementService.registerToSession(sessionId);
            redirectAttributes.addFlashAttribute(attributeName: "success", attributeValue: "Inscription confirmée ! Votre place est réservée.");
            redirectAttributes.addFlashAttribute(attributeName: "creditsOperation", attributeValue: "registration");
            redirectAttributes.addFlashAttribute(attributeName: "newCredits", newCredits);

            //Catch local exceptions for user feedback
        } catch (SessionValidationException e) {
            log.error("Session validation error: {}", e.getMessage());
            redirectAttributes.addFlashAttribute(attributeName: "error", e.getUserMessage());
            // Catch Feign Exceptions
        } catch (FeignException e) {
            String errorMessage = switch (e.status()) {...};
            redirectAttributes.addFlashAttribute(attributeName: "error", errorMessage);
        } catch (Exception e) {
            redirectAttributes.addFlashAttribute(attributeName: "error", attributeValue: "Erreur technique lors de l'inscription");
        }

        return new ModelAndView(viewName: "redirect:/client?tab=upcoming");
    }
}
```

Endpoint associé (Controller)

Projet : KundApp

```
@FeignClient(name = "ms-course-mgmt", path = "/api/sessions")  8 usages  ± Jacques *
public interface CourseManagementFeignClient {

    Client session methods

    @GetMapping(@RequestMapping("/client/available"))  new *
    List<SessionNoParticipantsDTO> getAvailableSessionsForClient();

    @GetMapping(@RequestMapping("/client/upcoming/{participantId}))  new *
    List<SessionNoParticipantsDTO> getUpcomingSessionsForClient(@PathVariable("participantId") Long participantId);

    @GetMapping(@RequestMapping("/client/past/{participantId}))  new *
    List<SessionNoParticipantsDTO> getPastSessionsForClient(@PathVariable("participantId") Long participantId);

    @PostMapping(@RequestMapping("/client/{sessionId}/participants"))
    ParticipantOperationResponse addParticipantToSession(
        @PathVariable Long sessionId,
        @RequestBody AddParticipantRequest request);

    @PostMapping(@RequestMapping("/client/{sessionId}/participants/remove/{userId}"))
    ParticipantOperationResponse removeParticipantFromSession(
        @PathVariable Long sessionId,
        @PathVariable Long userId);

    @GetMapping(@RequestMapping("/{sessionId}"))
    SessionWithParticipantsDTO getSessionById (@PathVariable("sessionId") Long sessionId);

    Teacher session methods

    @PostMapping(RequestMapping("/teacher"))
    SessionCreationResponseDTO createSession(@Valid @RequestBody SessionCreationWithTeacherDTO dto);

    // Upcoming sessions
    @GetMapping(@RequestMapping("/teacher/{teacherId}/upcoming"))
    List<SessionWithParticipantsDTO> getUpcomingSessionsByTeacher(@PathVariable("teacherId") Long teacherId);

    // History of sessions
    @GetMapping(@RequestMapping("/teacher/{teacherId}/past"))
    List<SessionWithParticipantsDTO> getPastSessionsByTeacher(@PathVariable("teacherId") Long teacherId);

    @PutMapping(@RequestMapping("/{sessionId}/teacher/{teacherId}"))
    SessionWithParticipantsDTO updateSessionByTeacher(@PathVariable Long sessionId,
```

Communication FeignClient, liste des endpoints (appel vers ms-coursemgmt)

Projet : KundApp

```
@Service * Jacques*
@Slf4j
@RequiredArgsConstructor
public class SessionParticipantImplService implements SessionParticipantService {

    private final SessionRepository sessionRepository;
    private static final int CANCELLATION_CUTOFF_HOURS = 48; 1 usage

    Adds participant to session with business validations

    @Transactional 1 usage * Jacques*
    public ParticipantOperationResponse addParticipantToSession(Long sessionId, Long userId) {
        log.info("Adding participant {} to session {}", userId, sessionId);

        Session session = sessionRepository.findById(sessionId)
            .orElseThrow(() -> new SessionNotFoundException("Session not found with ID: " + sessionId));

        validateSessionNotFull(session);
        validateUserNotAlreadyRegistered(session, userId);

        // Initialize participant list if null
        if (session.getParticipantIds() == null) {
            session.setParticipantIds(new ArrayList<>());
        }

        session.getParticipantIds().add(userId);
        sessionRepository.save(session);

        log.info("Participant {} successfully added to session {}. New count: {}/{}",
            userId, sessionId, session.getParticipantIds().size(), session.getAvailableSpots());

        return new ParticipantOperationResponse(
            sessionId,
            userId,
            operation: "ADD_PARTICIPANT",
            LocalDateTime.now(),
            session.getParticipantIds().size(),
            session.getAvailableSpots(),
            new ArrayList<>(session.getParticipantIds())
        );
    }
}
```

Méthode d'ajout de participant à la session (ms-coursemgmt)

Projet : KundApp

```
CREDITS SECTION

@PostMapping(@RequestMapping("/internal/credits/session-registration-deduct")) ▲ Jacques
💡 CreditOperationResponse deductCreditsForSessionRegistration(
    @RequestBody SessionRegistrationDeductRequest request);

@PostMapping(@RequestMapping("/internal/credits/session-rollback-refund")) ▲ Jacques
CreditOperationResponse refundCreditsForSessionRollback(
    @RequestBody SessionRollbackRefundRequest request);

@PostMapping(@RequestMapping("/internal/credits/batch-deduct")) ▲ Jacques
void batchDeductCreditsForRollback(@RequestBody BatchCreditOperationRequest request);

@PostMapping(RequestMapping("/internal/credits/batch-refund")) ▲ Jacques
void batchRefundCreditsForCancellation(@RequestBody BatchCreditOperationRequest request);
```

Communication Feign Client depuis ms-webApp vers ms-identity (gestion users et crédits) appelées lors de la tentative d'enregistrement à une session

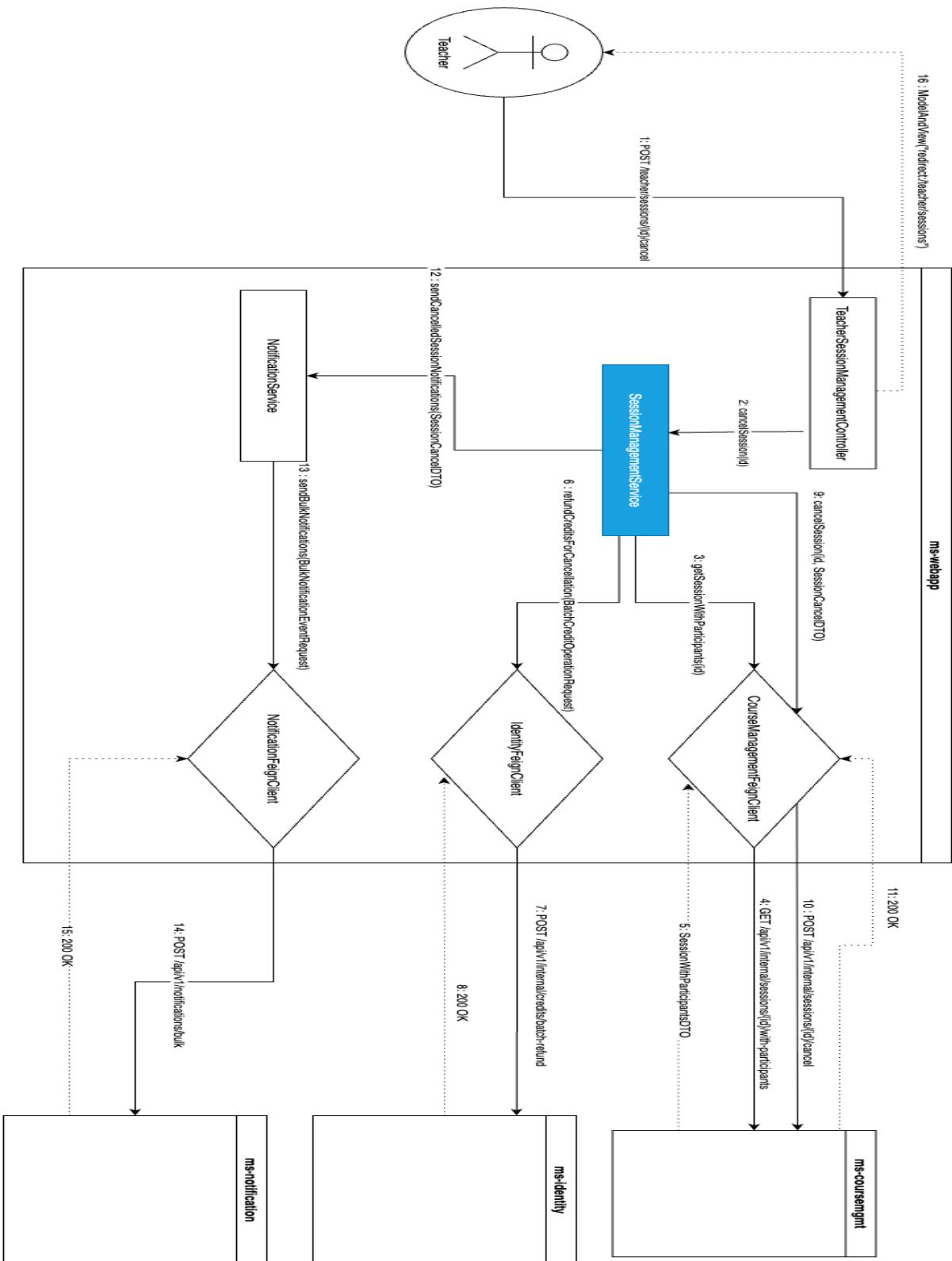
8.2. Système de notifications flexible et découpé

On cherche à concevoir un service capable de gérer une variété croissante de notifications (inscription, annulation, modification de cours, etc.) sans nécessiter de modification de son schéma de base de données et en restant totalement indépendant des autres services métier.

J'ai développé le microservice ms-notification en effectuant des choix technologiques rigoureux afin de garantir sa flexibilité et sa robustesse :

- Base de données NoSQL (MongoDB) : idéale pour ce cas d'usage. La nature "schemaless" de MongoDB permet de stocker des documents de notification avec des structures variées. Une notification d'inscription ne possède pas les mêmes champs qu'une notification de bienvenue, et MongoDB gère cela nativement.
- Découplage total : le service ms-notification est "aveugle". Il ne connaît ni l'entité User, ni l'entité Session. Il expose une API simple qui reçoit des données brutes et un type d'événement. Il incombe à l'orchestrateur (ms-webapp) de collecter ces informations avant d'appeler le service.

Cette architecture rend le système de notification extrêmement résilient aux changements et facile à étendre. L'orchestration de l'envoi des notifications d'annulation d'une session est un parfait exemple de ce découplage.



Annulation d'une séance (renvoi des credits aux participants et notifications)

Projet : KundApp

```
public void sendSessionCancelledNotifications(SessionWithParticipantsDTO session) { 7 usages ± Jacques*
    //Notify participants
    if (session.getParticipantIds() != null && !session.getParticipantIds().isEmpty()) {
        try {
            List<UserParticipantDTO> participants = identityFeignClient.getUsersBasicInfo(session.getParticipantIds());
            List<NotificationUserDto> participantDtos = participants.stream() Stream<UserParticipantDTO>
                .map(this::buildNotificationUserDto) Stream<NotificationUserDto>
                .toList();

            // Notify users
            BulkNotificationEventRequest request = new BulkNotificationEventRequest(
                NotificationEventType.SESSION_CANCELLED_TO_USER_NOTIFICATION,
                buildNotificationSessionDto(session),
                participantDtos,
                internalSecret
            );
            notificationFeignClient.processBulkNotificationEvent(request);
        } catch (Exception e) {...}
    }
    // Notify teacher
    try {
        UserParticipantDTO teacher = identityFeignClient.getUserBasicInfo(session.getTeacherId());

        List<NotificationUserDto> participantDtos = Collections.emptyList();
        if (session.getParticipantIds() != null && !session.getParticipantIds().isEmpty()) {
            List<UserParticipantDTO> participants = identityFeignClient.getUsersBasicInfo(session.getParticipantIds());
            participantDtos = participants.stream() Stream<UserParticipantDTO>
                .map(this::buildNotificationUserDto) Stream<NotificationUserDto>
                .toList();
        }
        NotificationEventRequest teacherRequest = new NotificationEventRequest(
            NotificationEventType.SESSION_CANCELLED_TO_TEACHER_NOTIFICATION,
            buildNotificationUserDto(teacher),
            buildNotificationSessionDto(session),
            participantDtos,
            internalSecret
        );
        notificationFeignClient.processNotificationEvent(teacherRequest);

    } catch (Exception e) {...}
}
```

Méthode d'orchestration d'envoi de notifications (ms-webapp)

Le code ci-dessus illustre parfaitement le rôle de l'orchestrateur. La méthode sendSessionCancelledNotifications dans ms-webapp ne se contente pas d'appeler ms-notification. Elle effectue une logique métier à plus haute valeur ajoutée :

1. Elle reçoit un SessionCancelDTO contenant la liste des participants.
2. Elle transforme cette liste en un BulkNotificationEventRequest, un DTO spécifiquement conçu pour l'API de ms-notification.
3. Elle invoque ensuite le NotificationFeignClient avec cette requête unique.

Ce modèle est très efficace : il délègue la tâche "bête" (envoyer une notification) au service spécialisé, tout en conservant la logique métier "intelligente" (qui notifier et avec quelles données) au sein de l'orchestrateur.

Projet : KundApp

```
// Single Notification Event Processing
@Override 2 usages • Jacques
@Transactional
public NotificationEventResponse processNotificationEvent(NotificationEventRequest request) {
    log.info("Processing notification event - Type: {}, User: {}, Session: {}",
        request.eventType(),
        request.user().email(),
        request.session().id()
    );
    try {
        // 1. Convert DTO to Entity
        Notification notification = notificationMapper.toEntity(request);

        // 2. Save (status is PENDING)
        notification = notificationRepository.save(notification);
        log.debug("Notification saved with ID: {}", notification.getId());

        // 3. Send email with retry logic
        boolean emailSent = emailUtilityService.sendEmailWithRetry(notification);

        // 4. Update notification status to SENT or FAILED
        notification.setStatus(emailSent ? NotificationStatus.SENT : NotificationStatus.FAILED);
        if (emailSent) {
            notification.setSentAt(LocalDateTime.now());
        }
        notification = notificationRepository.save(notification);

        // 5. Return response
        NotificationEventResponse response = notificationMapper.toResponseDto(notification);

        log.info("Notification processed successfully - ID: {}, Status: {}",
            notification.getId(), notification.getStatus());

        return response;
    } catch (Exception e) {
        log.error("Failed to process notification event: {}", e.getMessage(), e);
        throw new NotificationProcessingException("Failed to process notification event");
    }
}
```

Méthode de génération de notification unique (ms-notification)

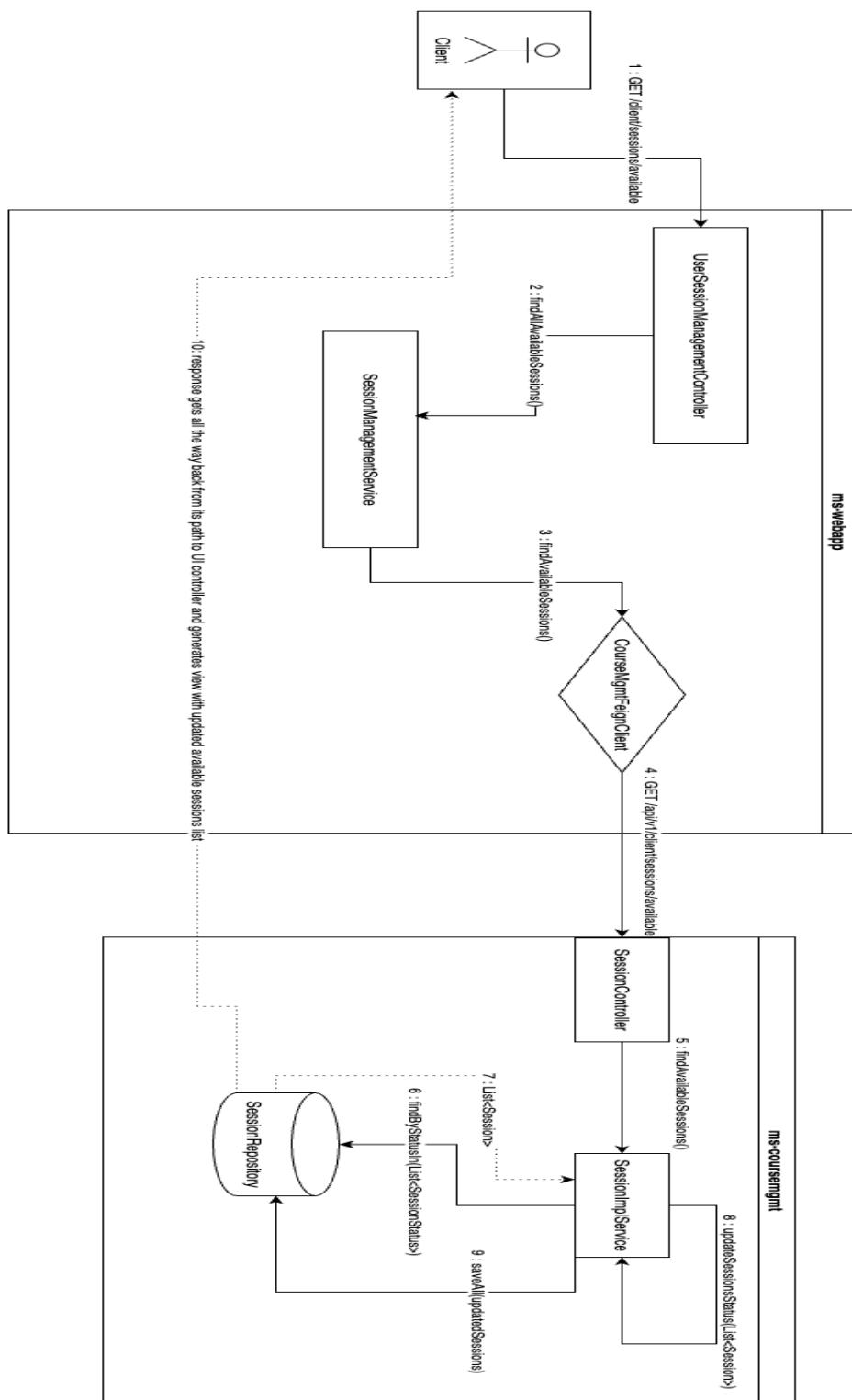
8.3. Mise à Jour "Just-in-Time" du Statut des Sessions

Le statut d'une session (SCHEDULED, COMPLETED,CANCELED) doit toujours être à jour pour refléter la réalité. Une solution classique aurait été de mettre en place des tâches planifiées (cron jobs) pour vérifier et mettre à jour les statuts en arrière-plan. Cependant, cette approche ajoute de la complexité (gestion des threads, charge potentiellement inutile sur la base de données). J'ai opté pour une solution plus légère et tout aussi efficace.

J'ai implémenté une stratégie de mise à jour "Just-in-Time" (juste à temps). Le statut d'une session n'est pas mis à jour par un processus en arrière-plan, mais au moment précis où les données sont lues de la base de données pour être affichées à l'utilisateur.

1. Centralisation de la logique : la logique de mise à jour est centralisée dans une méthode privée `updateSessionsStatus` au sein du service `SessionImplService.java`.
2. Déclenchement à la lecture : cette méthode est systématiquement appelée par toutes les méthodes de service qui récupèrent des listes de sessions (ex: `getAvailableSessionsForClient`, etc...) avant de retourner les données au controller.
3. Performance optimisée : La méthode parcourt la liste des sessions récupérées et ne met à jour en base de données que celles dont le statut a changé (une session dont l'heure de fin est passée, si la session est programmée et que l'heure de début est passée, cette dernière n'est juste plus affichée). Cela limite les écritures en base de données.

4.



Update des séances terminées en direct de l'affichage de l'interface utilisateur

Projet : KundApp

```
@Service * Jacques*
@RequiredArgsConstructor
@Transactional
public class SessionImplService implements SessionService {

    private final SessionRepository sessionRepository;
    private final SessionMapper sessionMapper;
    private final SessionJobManagement sessionJobManagement;

    Client part

    @Override 1 usage new*
    public List<SessionNoParticipantsDTO> getAvailableSessionsForClient() {
        LocalDateTime now = LocalDateTime.now();

        sessionJobManagement.updateCompletedSessions();

        List<Session> sessions = sessionRepository
            .findByStatusOrderByStartTimeAsc(SessionStatus.SCHEDULED) List<Session>
            .stream() Stream<Session>
            .filter( Session session -> session.getStartTime().isAfter(now))
            .toList();

        return sessions.stream() Stream<Session>
            .map(sessionMapper::toSessionGetClientResponseDTO) Stream<SessionNoParticipantsDTO>
            .toList();
    }
}
```

```
> @{...}
public class SessionJobManagement {

    private final SessionRepository sessionRepository;
    private final SessionMapper sessionMapper;
    ⚡

    Updates the status of sessions that have ended to 'COMPLETED'. This method could be called
    periodically (e.g., via a scheduled cron job for improvement).

    public void updateCompletedSessions() { 3 usages * Jacques
        LocalDateTime now = LocalDateTime.now();
        List<Session> sessionsToComplete = sessionRepository.findAll().stream()
            .filter( Session s -> s.getStatus() == SessionStatus.SCHEDULED)
            .filter( Session s -> s.getStartTime().plusMinutes(s.getDurationMinutes()).isBefore(now))
            .collect(toList());

        sessionsToComplete.forEach( Session s -> s.setStatus(SessionStatus.COMPLETED));
        if (!sessionsToComplete.isEmpty()) {
            sessionRepository.saveAll(sessionsToComplete);
        }
    }
}
```

Méthode de mise à jour des sessions (ms-course-mgmt)

appelée depuis la webApp lors de l'affichage du dashboard

9. Sécurité de la Plateforme

La sécurité n'a pas été une réflexion après coup, mais un pilier de la conception de l'architecture de KundApp. Une approche de "défense en profondeur" a été mise en place, combinant plusieurs couches de protection pour garantir la confidentialité, l'intégrité et la disponibilité des données.

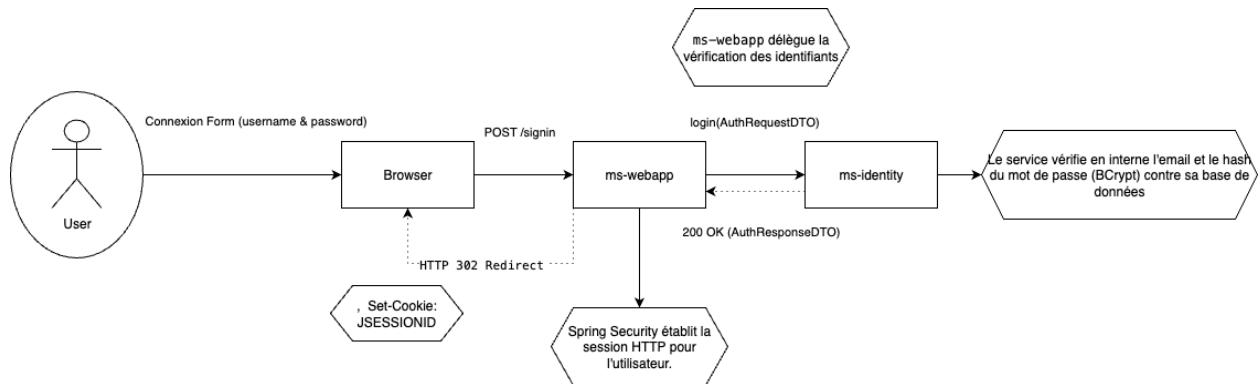
9.1. Authentification, Autorisations et Sécurité

J'ai mis en place une sécurité à plusieurs niveaux, gérée par Spring Security, pour répondre à une séquence de questions logiques : Qui est l'utilisateur ? Que peut-il faire ? Et comment s'assurer que ses actions sont légitimes ?

Étape 1 : Authentification par Session

La base de la sécurité est une authentification "stateful" robuste, qui s'appuie sur les sessions et les cookies.

Flux d'Authentification : le processus est orchestré par ms-webapp mais validé par ms-identity. Lors de la soumission du formulaire de connexion, ms-webapp délègue la vérification des identifiants à ms-identity via un appel Feign. Si la validation réussit, ms-webapp crée une session sécurisée sur le serveur et renvoie un cookie JSESSIONID au navigateur. Ce cookie, configuré en HttpOnly, agit comme un "laissez-passer" qui identifie l'utilisateur lors de toutes ses requêtes ultérieures.



Processus d'authentification

Étape 2 : Contrôle d'Accès par Rôles

Une fois authentifié, l'utilisateur bénéficie d'un accès contrôlé et granulaire aux différentes routes de l'application, en fonction de son rôle.

Chaque point d'accès est sécurisé et requiert un rôle spécifique (CLIENT, TEACHER ou ADMIN). Cette configuration est centralisée afin d'optimiser la lisibilité et la maintenabilité.

Extrait de la configuration de sécurité des routes : la meilleure preuve de l'implémentation réside dans le code. Cet extrait de la classe SpringSecurityConfig illustre la protection explicite des routes de l'application et leur association à des rôles spécifiques.

```
public class SpringSecurityConfig {
    * Handles route access rules, login success handlers, CSRF, and session management.
    */
    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(CsrfConfigurer<HttpSecurity> csrf -> csrf
                .ignoringRequestMatchers(patterns: "/signup", "/signin"))
            // public routes
            .authorizeHttpRequests((AuthorizationManagerRequestMatcher... requests) -> requests
                .requestMatchers(@"/signin",
                    @"/signup",
                    @"/static/**",
                    @"/error",
                    @"/css/**",
                    @"/images/**",
                    @"/js/signin_signup.js",
                    @"/v3/api-docs/**", //SWAGGER
                    @"/swagger-ui.html",
                    @"/swagger-ui/**")
                .authorizedUrl()
                .permitAll() AuthorizationManagerRequestMatcher...)
                .requestMatchers(@"/admin/**").hasRole("ADMIN")
                .requestMatchers(@"/client/**").hasAnyRole(...roles: "CLIENT")
                .requestMatchers(@"/teacher/**").hasAnyRole(...roles: "TEACHER")
                .requestMatchers(@"/api/sessions/**").hasAnyRole(...roles: "TEACHER", "ADMIN", "CLIENT")
                .anyRequest() AuthorizedUrl
                .authenticated())
            .formLogin((FormLoginConfigurer<HttpSecurity> form) -> form
                // customized login form
                .loginPage("/signin")
                .loginProcessingUrl("/authenticate") // To go through postmapping /authentication, using our own AuthenticationProvider (to retrieve info)
                .usernameParameter("email")
                .passwordParameter("password")
                .successHandler(customAuthenticationSuccessHandler) // custom routes by authenticated role
                .failureHandler(customAuthenticationFailureHandler) // custom errors management
                .permitAll())
            ..)
            // logout management
            .logout((LogoutConfigurer<HttpSecurity> logout) -> logout
                .logoutUrl("/logout")
                .logoutSuccessUrl("/signin?logout=true")
                .permitAll());
        return http.build();
    }
}
```

Configuration spring security (routes séparées pour chaque rôle)

```

@Component ✎ Jacques
public class CustomAuthenticationSuccessHandler implements AuthenticationSuccessHandler {

    private final Logger logger = LoggerFactory.getLogger(this.getClass()); 1 usage

    private static final Map<String, String> ROLE_REDIRECT_MAP = Map.of( 1 usage
        Role.ADMIN.getAuthority(), v1: "/admin",
        Role.CLIENT.getAuthority(), v2: "/client",
        Role.TEACHER.getAuthority(), v3: "/teacher"
    );

    // Manages each possible route when connecting as a user
    @Override no usages ✎ Jacques
    public void onAuthenticationSuccess(HttpServletRequest request,
                                         HttpServletResponse response,
                                         Authentication authentication) throws IOException {

        String authority = authentication.getAuthorities().stream() Stream<capture of extends GrantedAuthority>
            .map(GrantedAuthority::getAuthority) Stream<String>
            .findFirst() Optional<String>
            .orElse( other: "ROLE_UNKNOWN");

        String redirectUrl = ROLE_REDIRECT_MAP.getOrDefault(authority, defaultValue: "/signin");

        logger.info("Authority: {}, Redirect to: {}", authority, redirectUrl);
        response.sendRedirect(redirectUrl);
    }
}

```

Processus d'authentification et routes dédiées

Étape 3 : Protection des Actions

Enfin, il faut s'assurer que les actions qui modifient des données (créer une session, s'inscrire...) sont bien initiées par l'utilisateur depuis l'application elle-même. C'est le rôle de la protection contre les attaques de type Cross-Site Request Forgery (CSRF).

- Fonctionnement :
 1. Côté serveur : pour chaque session, Spring Security génère un jeton CSRF secret et unique. Thymeleaf l'insère automatiquement dans une balise <meta> de la page HTML envoyée au client.
 2. Côté client : mon script common.js est conçu pour lire ce jeton depuis la page.
 3. Vérification : pour chaque requête sensible (type: POST, PUT, etc.), le JavaScript ajoute ce jeton dans les en-têtes. Spring Security compare alors le jeton reçu avec celui qu'il attendait. Si les jetons correspondent, l'action est validée ; sinon, elle est rejetée.

Projet : KundApp

Cette approche garantit que les actions de l'utilisateur sont non seulement autorisées par son rôle, mais aussi sécurisées dans leur exécution.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org" lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta name="_csrf" th:content="${_csrf.token}" />
    <meta name="_csrf_header" th:content="${_csrf.headerName}" />
    <meta name="app-success" th:content="${success}" th:if="${success}" />
    <meta name="app-error" th:content="${error}" th:if="${error}" />
    <meta name="credits-operation" th:content="${creditsOperation}" th:if="${creditsOperation}" />
    <meta name="new-credits" th:content="${newCredits}" th:if="${newCredits}" />
    <title>Sessions | KundApp</title>
    <link th:href="@{/css/mdb.min.css}" rel="stylesheet"/>
    <link th:href="@{/css/index.css}" rel="stylesheet"/>
</head>
```

Gestion du token csrf côté front

```
// Submit form action with CSRF token
function submitAction(actionUrl) : void { Show usages  ± Jacques
    const form : HTMLFormElement = document.createElement( tagName: 'form' );
    form.method = 'POST';
    form.action = actionUrl;
    form.style.display = 'none';

    // Add CSRF token
    const csrfToken = CommonUtils.getCsrfToken();
    const csrfInput : HTMLInputElement = document.createElement( tagName: 'input' );
    csrfInput.type = 'hidden';
    csrfInput.name = '_csrf';
    csrfInput.value = csrfToken;
    form.appendChild(csrfInput);

    document.body.appendChild(form);
    form.submit();
    document.body.removeChild(form);
}
```

Injection du token csrf lors des requêtes Post

9.2. Sécurité de la Communication Inter-Services

Dans une architecture microservices, tous les endpoints ne sont pas destinés à être publics. Certains, particulièrement ceux qui effectuent des opérations critiques comme la modification des crédits d'un utilisateur, ne doivent jamais être accessibles depuis l'extérieur. Il était donc impératif de mettre en place une couche de sécurité supplémentaire pour la communication interne entre les services.

J'ai implémenté un mécanisme de secret partagé (shared secret) qui est directement intégré au contrat de données (DTO) des requêtes internes. Cette approche garantit que seuls les services de confiance peuvent exécuter des opérations sensibles.

1. Transport du secret : le token secret (internalSecret) est transmis directement comme un attribut dans le corps de la requête (le DTO). Chaque DTO destiné à un endpoint interne sensible possède un champ internalSecret.

```
public record SessionRegistrationDeductRequest( 5 usages  ▲ Jacques
    @NotNull(message = "User ID is required")  7 usages
    Long userId,
    @NotNull(message = "Session ID is required")  2 usages
    Long sessionId,
    @Min(value = 1, message = "Credits required must be at least 1")  4 usages
    @NotNull(message = "Credits required is mandatory")
    Integer creditsRequired,
    @NotNull(message = "Internal secret is required")  1 usage
    String internalSecret
) {}
```

InternalSecret contenu dans l'objet lié à la déduction des crédits

- Injection du secret par l'appelant : lorsqu'un service (ex: ms-webapp) prépare son appel Feign, il a la responsabilité de peupler ce champ internalSecret dans l'objet DTO avec la valeur du secret qu'il possède dans sa configuration.

```

mswebapp:
  image: jrouillet/kundapp-ms-webapp:latest
  container_name: webapp-ms
  ports:
    - "8080:8080"
  networks:
    - kundapp-public
    - kundapp-internal
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

msidentity:
  image: jrouillet/kundapp-ms-identity:latest
  container_name: identity-ms
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

```

InternalSecret configuration

- Validation systématique par le récepteur : dès la réception de la requête, la toute première action du controller interne est d'extraire le secret du DTO et de le passer à un composant SecurityValidator. Ce validateur compare le secret reçu avec celui attendu. Si le secret est manquant ou invalide, une exception UnauthorizedInternalAccessException est levée, et la requête est immédiatement rejetée avec une erreur 401 Unauthorized, bien avant que la moindre logique métier ne soit exécutée.

```

@Component 2 usages ± Jacques
@Slf4j
public class SecurityValidator {

    @Value("${app.internal.secret}")
    private String internalSecret;

    public void validateInternalSecret(String providedSecret) { 4 usages ± Jacques
        if (!internalSecret.equals(providedSecret)) {
            log.warn("Invalid internal secret provided");
            throw new SecurityException("Invalid internal secret");
        }
    }
}

```

InternalSecret validateur

Cette méthode est particulièrement robuste car la validation de sécurité fait partie intégrante du traitement de la requête, et non d'un filtre ou d'un intercepteur séparé.

Extrait de la validation du secret dans un controller interne : ce code, présent au début de chaque endpoint sensible, est le gardien de la sécurité inter-services. Il illustre comment la validation est la première étape non négociable.

```
| Deducts credits for session registration with security validation
@Transactional 1 usage  ± Jacques
public CreditOperationResponse deductCreditsForSessionRegistration(SessionRegistrationDeductRequest request) {
    log.info("Processing credit deduction for user {} and session {}", request.userId(), request.sessionId());
    securityValidator.validateInternalSecret(request.internalSecret());
```

Appel du validateur de l'InternalSecret dans la méthode deductCreditsForSessionRegistration

9.3. Segmentation du Réseau avec Docker

La sécurité de l'application ne se limite pas au code, elle a également été pensée au niveau de l'infrastructure de déploiement. Une segmentation réseau stricte a été mise en place pour isoler les services et contrôler les flux de communication.

Le fichier docker-compose.yml définit deux réseaux virtuels distincts pour créer des périmètres de sécurité.

- kundapp-public : un réseau externe conçu pour exposer les seuls services qui doivent être joignables depuis l'extérieur. Il s'agit principalement de la gateway ms-webapp, qui sert de point d'entrée unique, et des services d'infrastructure comme Eureka.
- kundapp-internal : un réseau privé, marqué avec l'option internal: true dans la configuration Docker. Les services métier critiques qui manipulent les données sensibles (ms-identity, ms-coursemgmt, ms-notification) sont exclusivement connectés à ce réseau. Ils sont ainsi totalement inaccessibles depuis l'extérieur, même en cas de mauvaise configuration d'un pare-feu.

Toute communication à destination des services métier doit obligatoirement transiter par la gateway ms-webapp, qui agit comme un bastion et un point de contrôle unique pour toutes les requêtes entrantes.

```

mswebapp:
  image: jrouillet/kundapp-ms-webapp:latest
  container_name: webapp-ms
  ports:
    - "8080:8080"
  networks:
    - kundapp-public
    - kundapp-internal
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config


networks:
  kundapp-public:
    driver: bridge
  kundapp-internal:
    driver: bridge
    internal: true

msidentity:
  image: jrouillet/kundapp-ms-identity:latest
  container_name: identity-ms
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

```

Réseaux public et interne Docker

9.4. Bonnes Pratiques Complémentaires

Enfin, plusieurs bonnes pratiques de sécurité standard ont été appliquées de manière systématique à travers tout le projet pour se prémunir contre les vulnérabilités les plus courantes.

- Hachage des mots de passe : la sécurité des mots de passe est non négociable. J'utilise l'algorithme robuste et adaptatif BCrypt, qui est le standard recommandé par Spring Security. Le mot de passe n'est jamais stocké en clair ; seul son hash est conservé en base de données, rendant toute fuite de données beaucoup moins critique.

```
@Component ▲ Jacques
public class SecurityConfig {

    /**
     * Password encoder bean using BCrypt algorithm.
     */
    @Bean ▲ Jacques
    PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

```
@RequiredArgsConstructor 2 usages ▲ Jacques
@Service
public class AuthService {
    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final Logger logger = LoggerFactory.getLogger(AuthService.class); 5 usages

    public AuthResponseDTO authenticate(AuthRequestDTO request) { 1 usage ▲ Jacques
        Optional<User> userOpt = userRepository.findByEmail(request.getEmail());

        if (userOpt.isPresent()) {
            User user = userOpt.get();
            logger.info("User found: {}", user.getEmail());

            if (passwordEncoder.matches(request.getPassword(), user.getPassword())) {
                logger.info("Authentication successful for: {}", user.getEmail());
                return new AuthResponseDTO(
                    authenticated: true,
                    user.getId(),
                    user.getEmail(),
                    user.getRole()
                );
            }
        }
    }
}
```

PasswordEncoder & BCrypt

```
@Override 1 usage  ▲ Jacques
public UserDTO createUser(UserCreationDTO dto){
    if (userRepository.findByEmail(dto.getEmail()).isPresent()) {
        throw new EmailAlreadyExistsException("Email already exists");
    }
    User user = new User();
    user.setEmail(dto.getEmail());
    user.setPassword(passwordEncoder.encode(dto.getPassword()));
    user.setRole(Role.CLIENT);
    user.setFirstName(dto.getFirstName());
    user.setLastName(dto.getLastName());
    user.setPhoneNumber(dto.getPhoneNumber());
    user.setDateOfBirth(dto.getDateOfBirth());
    user.setAddress(dto.getAddress());
    user.setStatus(true);
    user.setCredits(dto.getCredits());

    User savedUser = userRepository.save(user);
    return userMapper.toUserDto(savedUser);
}
```

Encodage du mot de passe lors de la création de l'utilisateur

- Prévention des injections sql : L'intégralité des accès à la base de données s'effectue via l'ORM JPA/Hibernate et ses Repositories. En utilisant les méthodes fournies par JpaRepository ou en construisant des requêtes avec JPQL, les paramètres sont automatiquement "échappés" (prepared statements), ce qui prévient nativement et efficacement les attaques par injection SQL.

```
@Repository 8 usages  ▲ Jacques
public interface UserRepository extends JpaRepository<User, Long> {

    @Query(value = "SELECT * FROM users WHERE email= ?", nativeQuery = true) 7 usages  ▲ Jacques
    Optional<User> findByEmail(@Email String email);
```

Requête native SpringJPA protégée contre les injections SQL

- Prévention XSS (Cross-Site Scripting) : Pour se prémunir contre l'injection de scripts malveillants côté client, Thymeleaf est configuré par défaut pour échapper toutes les variables dynamiques affichées dans les templates HTML. Cela garantit que toute donnée provenant de la base ou d'un utilisateur est traitée comme du texte et non comme du code exécutable.

```
<div class="col-4 session-col-center">
    <div class="session-teacher-line">
        <span class="fw-bold teacher-name">
            Prof. <span th:text="${upcomingSession.teacherFirstName + ' ' + upcomingSession.teacherLastName}">Marie Dupont</span>
        </span>
    </div>
    <div class="session-room-line">
        <small>
            <th:block th:if="${upcomingSession.isOnline}">
                <span class="session-online-badge">█ En ligne</span>
            </th:block>
            <th:block th:unless="${upcomingSession.isOnline}">
                <span th:text="${upcomingSession.roomName ?: 'Lieu non défini'}">Salle A</span>
            </th:block>
        </small>
    </div>
    <div class="session-participants-line">
        <small>
            <span th:text="${upcomingSession.registeredParticipants}>5</span> participants
            (<span th:text="${upcomingSession.availableSpots - upcomingSession.registeredParticipants}>3</span> places)
        </small>
    </div>
</div>
```

Utilisation des mécanismes de protection natif inhérents aux variables thymeleaf

- Sécurité des cookies : le cookie de session (JSESSIONID) est configuré par Spring Security en mode HttpOnly. Cet attribut critique interdit à tout script JavaScript exécuté dans le navigateur d'accéder au cookie, le protégeant ainsi efficacement contre le vol de session via une faille XSS.

10. DevOps : Usine Logicielle et Déploiement

Afin de garantir un accès aisément, une reproductibilité optimale et une maintenance facilitée du projet, le déploiement a été effectué via Docker. Cette approche DevOps s'articule autour de la conteneurisation des services, d'une configuration centralisée et d'une stratégie de tests multiniveaux.

10.1. Conteneurisation et Orchestration avec Docker

L'architecture microservices a été intégralement conçue pour une conteneurisation avec Docker, assurant ainsi une identité parfaite entre les environnements de développement et de production, éliminant de fait le célèbre "ça marche sur ma machine".

- Conteneurisation de chaque service : chaque microservice (Java/Spring) est encapsulé au sein de sa propre image Docker. Pour optimiser et accélérer la création de ces images, le plugin Maven Jib a été employé, permettant la construction directe des images sans Dockerfile et une séparation intelligente des dépendances et des classes applicatives pour des builds plus rapides.
- Orchestration avec Docker Compose : un fichier docker-compose.yml assure l'orchestration du lancement de l'ensemble de la stack applicative, incluant les microservices, les bases de données (MySQL, MongoDB), le serveur de configuration et le service discovery.
- Gestion des Dépendances au Démarrage : le défi majeur dans une architecture microservices réside dans la gestion de l'ordre de démarrage. Une combinaison de depends_on et de healthcheck a été utilisée dans le docker-compose.yml pour garantir qu'un service ne démarre qu'après que ses dépendances critiques (telles que le serveur de configuration ou sa base de données) soient pleinement opérationnelles.

Veuillez trouver ci-dessous le “docker-compose” et la “common-config” nécessaires au déploiement de l'application KundApp.

Projet : KundApp

```
services:

msconfigserver:
  image: jrouurret/kundapp-ms-configserver:latest
  container_name: configserver-ms
  networks:
    - kundapp-internal
    - kundapp-public
  ports:
    - "8888:8888"
  healthcheck:
    test: "curl --fail --silent localhost:8888/actuator/health/readiness | grep UP || exit 1"
    interval: 10s
    timeout: 5s
    retries: 10
    start_period: 10s
  secrets:
    - git_password
  environment:
    GIT_PASSWORD_FILE: /run/secrets/git_password
  extends:
    file: common-config.yml
    service: microservice-base-config

msdiscovery:
  image: jrouurret/kundapp-ms-discovery:latest
  container_name: discovery-ms
  networks:
    - kundapp-internal
    - kundapp-public
  ports:
    - "8761:8761"
  depends_on:
    msconfigserver:
      condition: service_healthy
  healthcheck:
    test: "curl --fail --silent localhost:8761/actuator/health/readiness | grep UP || exit 1"
    interval: 10s
    timeout: 5s
    retries: 10
    start_period: 10s
  extends:
    file: common-config.yml
    service: microservice-configserver-config

msidentitydb:
  container_name: identitydb
  networks:
    - kundapp-public
    - kundapp-internal
  ports:
    - "3308:3306"
  environment:
    MYSQL_DATABASE: identitydb
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
  volumes:
    - ./docker/mysql-init:/docker-entrypoint-initdb.d
  extends:
    file: common-config.yml
    service: microservice-db-config
```

Projet : KundApp

```
mscoursegmtdb:
  container_name: coursegmtdb
  networks:
    - kundapp-public
    - kundapp-internal
  ports:
    - "3309:3306"
  environment:
    MYSQL_DATABASE: coursegmtdb
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
  volumes:
    - ./docker/mysql-course-init:/docker-entrypoint-initdb.d
  extends:
    file: common-config.yml
    service: microservice-db-config

msnotificationmongodb:
  container_name: notificationmongodb
  networks:
    - kundapp-public
    - kundapp-internal
  ports:
    - "27018:27017"
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${MONGODB_USERNAME}
    MONGO_INITDB_ROOT_PASSWORD: ${MONGODB_PASSWORD}
    MONGO_INITDB_DATABASE: ${MONGODB_DATABASE}
  extends:
    file: common-config.yml
    service: microservice-mongo-config

mswebapp:
  image: jroulet/kundapp-ms-webapp:latest
  container_name: webapp-ms
  ports:
    - "8080:8080"
  networks:
    - kundapp-public
    - kundapp-internal
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

msidentity:
  image: jroulet/kundapp-ms-identity:latest
  container_name: identity-ms
  depends_on:
    msidentitydb:
      condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE_URL}
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config
```

Projet : KundApp

```
mscoursemgmt:
  image: jrouillet/kundapp-ms-course-mgmt:latest
  container_name: course-mgmt-ms
  depends_on:
    - mscoursemgmtdb:
        condition: service_healthy
  environment:
    SPRING_DATASOURCE_URL: ${DATABASE.Course_URL}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

msnotification:
  image: jrouillet/kundapp-ms-notification:latest
  container_name: notification-ms
  ports:
    - *9010:9010*
  depends_on:
    - msnotificationmongodb:
        condition: service_healthy
  environment:
    SPRING_PROFILES_ACTIVE: docker,mock
    MONGODB_USERNAME: ${MONGODB_USERNAME}
    MONGODB_PASSWORD: ${MONGODB_PASSWORD}
    MONGODB_DATABASE: ${MONGODB_DATABASE}
    MONGODB_URI: mongodb://${MONGODB_USERNAME}:${MONGODB_PASSWORD}@msnotificationmongodb:27017/${MONGODB_DATABASE}?authSource=admin
    APP_INTERNAL_SECRET: ${INTERNAL_SECRET}
  extends:
    file: common-config.yml
    service: microservice-discovery-config

data-initializer:
  image: mysql:9.3
  container_name: data-initializer
  depends_on:
    - msidentity:
        condition: service_started
    - msidentitydb:
        condition: service_healthy
    - mscoursemgmt:
        condition: service_started
    - mscoursemgmtdb:
        condition: service_healthy
  volumes:
    - ./docker/mysql-init/02-data.sql:/tmp/insert-identity-data.sql
    - ./docker/mysql-course-init/02-data.sql:/tmp/insert-course-data.sql
  command: >
    sh -c "
      echo 'Waiting for services...' &&
      sleep 45 &&
      echo 'Inserting identity data' &&
      mysql -h msidentitydb -u root -p${MYSQL_ROOT_PASSWORD} msidentitydb < /tmp/insert-data.sql &&
      echo 'Identity data inserted!' &&
      echo 'Inserting course data...' &&
      mysql -h mscoursemgmtdb -u root -p${MYSQL_ROOT_PASSWORD} mscoursemgmtdb < /tmp/insert-course-data.sql &&
      echo 'Course data inserted!' &&
      echo 'Verification:' &&
      mysql -h msidentitydb -u root -p${MYSQL_ROOT_PASSWORD} msidentitydb -e 'SELECT email, role FROM users;' &&
      mysql -h mscoursemgmtdb -u root -p${MYSQL_ROOT_PASSWORD} mscoursemgmtdb -e 'SELECT id, subject, room_name FROM session;'

  restart: "no"
  environment:
    MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
  extends:
    file: common-config.yml
    service: microservice-base-config
```

Docker-compose (depends on, ordre de démarrage des services distincts)

Projet : KundApp

```
services:
  network-deploy-service:
    networks:
      - kundapp-internal

  microservice-base-config:
    extends:
      service: network-deploy-service
    deploy:
      resources:
        limits:
          memory: 700m
    environment:
      TZ: Europe/Paris
  microservice-configserver-config:
    extends:
      service: microservice-base-config
    environment:
      SPRING_PROFILES_ACTIVE: docker
      SPRING_CONFIG_IMPORT: configserver:http://msconfigserver:8888/
      SPRING_DATASOURCE_USERNAME: ${SPRING_DATASOURCE_USERNAME}
      SPRING_DATASOURCE_PASSWORD: ${SPRING_DATASOURCE_PASSWORD}

  microservice-discovery-config:
    extends:
      service:
        microservice-configserver-config
    depends_on:
      msconfigserver:
        condition: service_healthy
      msdiscovery:
        condition: service_healthy
    environment:
      EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://msdiscovery:8761/eureka/

  microservice-db-config:
    extends:
      service: network-deploy-service
    image: mysql
    healthcheck:
      test: [ "CMD", "mysqladmin", "ping", "-h", "localhost" ]
```

Docker-compose - Common-config (depends on, ordre de démarrage des services distincts)

Projet : KundApp

```
microservice-discovery-config:
  extends:
    service:
      microservice-configserver-config
  depends_on:
    msconfigserver:
      condition: service_healthy
    msdiscovery:
      condition: service_healthy
  environment:
    EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://msdiscovery:8761/eureka/

microservice-db-config:
  extends:
    service: network-deploy-service
  image: mysql
  healthcheck:
    test: [ "CMD", "mysqladmin" , "ping", "-h", "localhost" ]
    timeout: 10s
    retries: 10
    interval: 10s
    start_period: 10s
  environment:
    MYSQL_CHARSET: utf8mb4
    MYSQL_COLLATION: utf8mb4_unicode_ci
  volumes:
    - ./docker/mysql-config/my.cnf:/etc/mysql/conf.d/my.cnf

microservice-mongo-config:
  extends:
    service: network-deploy-service
  image: mongo
  healthcheck:
    test: ["CMD", "mongosh", "admin", "--username=${MONGODB_USERNAME}", "--password=${MONGODB_PASSWORD}", "--eval", "db.runCommand({ ping: 1 })"]
    interval: 10s
    timeout: 5s
    retries: 10
    start_period: 10s
```

Docker-compose - Common-config (depends on, ordre de démarrage des services distincts)

10.2. Stratégie de Tests Unitaires

Afin de garantir la fiabilité de la logique métier de chaque microservice, des tests unitaires ont été mis en place avec JUnit 5 et Mockito. Ces tests ont pour objectif de valider le comportement de chaque composant de la couche "Service" de manière totalement isolée.

Pour ce faire, les dépendances externes (comme les Repositories ou les Feign Clients) sont "mockées" (simulées) avec Mockito. Cela permet de tester un scénario métier spécifique (par exemple, "un utilisateur sans crédit ne peut pas s'inscrire") sans avoir besoin de démarrer une base de données ou les autres microservices.

```
class SessionManagementServiceTeacherTest {
    @Mock 8 usages
    private ValidationService validationService;

    @InjectMocks 11 usages
    private SessionManagementService sessionManagementService;

    private static final Long TEACHER_ID = TestDataBuilders.DEFAULT_TEACHER_ID; 19 usages
    private static final Long SESSION_ID = TestDataBuilders.DEFAULT_SESSION_ID; 21 usages
    private static UserDTO testTeacher; 9 usages
    private static SessionWithParticipantsDTO testSession; 16 usages
    private static SessionWithParticipantsDTO testSessionWithParticipants; 11 usages

    @BeforeAll new *
    static void setUpAll() {
        testTeacher = TestDataBuilders.createUserDTOForUser(TEACHER_ID, credits: 10);
        testSession = TestDataBuilders.createSessionWithNoParticipantsDTOForBaseSession();
        testSessionWithParticipants = TestDataBuilders.createSessionWithParticipantsDTOForSpecificParticipantsNoArguments();
    }

    @Test new *
    void createSessionForCurrentTeacherSuccess_shouldCreateSessionAndSendNotificationTest() {
        // Given
        SessionCreationDTO creationDto = createSessionCreationDTO();
        SessionCreationResponseDTO responseDto = new SessionCreationResponseDTO();
        responseDto.setSessionId(SESSION_ID);

        when(sessionService.getCurrentUser()).thenReturn(testTeacher);
        when(courseFeignClient.createSession(any(SessionCreationWithTeacherDTO.class))).thenReturn(responseDto);
        when(courseFeignClient.getSessionById(SESSION_ID)).thenReturn(testSession);
        doNothing().when(notificationService).sendSessionCreatedNotification(anyLong(), any());

        // When
        SessionCreationResponseDTO result = sessionManagementService.createSessionForCurrentTeacher(creationDto);

        // Then
        verify(sessionService).getCurrentUser();
        verify(courseFeignClient).createSession(any(SessionCreationWithTeacherDTO.class));
        verify(courseFeignClient).getSessionById(SESSION_ID);
        verify(notificationService).sendSessionCreatedNotification(TEACHER_ID, testSession);
        assertEquals(SESSION_ID, result.getSessionId());
    }
}
```

Tests Unitaires

Projet : KundApp

Ce test unitaire, situé dans le microservice ms-webapp, a pour objectif de valider un scénario métier critique : la création d'une nouvelle session par un professeur. Il s'assure que non seulement la session est correctement transmise au microservice de gestion de cours, mais aussi qu'une notification de confirmation est bien envoyée au professeur.

Objectif du test : Vérifier que le SessionManagementService orchestre correctement les appels à d'autres microservices (via Feign) lorsqu'un professeur crée une session avec succès.

Le test se décompose en trois phases classiques (Arrange, Act, Assert ou Given, When, Then) :

1. Initialisation (Arrange) :

- On simule avec des mocks (@Mock) les dépendances externes du SessionManagementService : CourseManagementFeignClient (pour créer la session) et NotificationService (pour envoyer l'email).
- On prépare les objets nécessaires : un SessionCreationDTO contenant les informations de la session à créer et un UserDTO représentant le professeur connecté.
- On définit le comportement des mocks avec Mockito.when(). On paramètre que lorsque la méthode createSession du courseFeignClient est appelée, elle doit retourner une réponse de succès simulée (SessionCreationResponseDTO).

2. Action (Act) :

- On exécute la méthode à tester :
`sessionManagementService.createSessionForCurrentTeacher(sessionCreationDTO)`.

3. Vérification (Assert) :

- On s'assure avec Mockito.verify() que la méthode createSession du courseFeignClient a bien été appelée une seule fois avec les bonnes données.
- Plus important encore, on vérifie également que la méthode sendSessionCreatedNotification du notificationService a également été appelée. Cela garantit que la logique de notification, qui fait partie intégrante du processus métier, n'a pas été oubliée.
- Enfin, on s'assure que le résultat retourné par la méthode est bien celui attendu (non nul et conforme à la réponse simulée).

Partie 4 : Veille et Conclusion

11. Veille Technologique et Sécurité

11.1. Veille Technologique

Pour ce projet, une veille régulière a été nécessaire pour faire des choix techniques pertinents. Les apprentissages principaux concernent la maîtrise de l'écosystème Docker (réseaux, volumes), l'utilisation de Jib pour optimiser la création d'images Java, et l'approfondissement de JavaScript natif pour dynamiser l'interface. Les sources principales pour cette recherche ont été les documentations officielles, des blogs techniques comme Baeldung et des plateformes comme Medium et Stack Overflow.

Architecture & Patterns

- **Microservices Guide (Martin Fowler)** : <https://martinfowler.com/microservices/>
- **Saga Design Pattern (Microsoft Azure)**:
<https://learn.microsoft.com/fr-fr/azure/architecture/patterns/saga>
- **HTML Over The Wire (Hotwire)** : <https://hotwired.dev/>
- **Data Consistency in a Microservices Architecture (Chris Richardson)** :
<https://www.google.com/search?q=https://microservices.io/patterns/data/data-consistency.html>

Écosystème Spring

- **Introduction to Spring Cloud Netflix - Eureka** :
<https://www.baeldung.com/spring-cloud-netflix-eureka>
- **Introduction to Spring Cloud OpenFeign** :
<https://www.baeldung.com/spring-cloud-openfeign>
- **Centralized Exception Handling in Spring Boot** :
<https://www.igneek.com/blog/centralized-exception-handling-in-spring-boot/>
- **BCrypt Password Hashing in Spring Security** :
<https://medium.com/@javacharter/bcrypt-the-secret-to-keeping-your-passwords-safe-6e96cb27adb1>
- **Spring Boot Testing with Mockito** :
<https://symflower.com/en/company/blog/2024/how-to-do-mocking-spring-boot/>

Gestion des Données (JPA & Bases de Données)

- **Optimising Spring Data & JPA queries** :
https://blog.touret.info/2024/03/25/jpa_spring_data_query_optimisations/
- **MongoDB vs SQL : différences et points communs** :
<https://www.ionos.fr/digitalguide/serveur/know-how/mongodb-vs-sql/>

DevOps & Déploiement

- **Control startup and shutdown order in Compose** :
<https://docs.docker.com/compose/how-tos/startup-order/>
- **Jib vs. Docker: A Comparative Analysis** :
<https://medium.com/@agamkakkar/jib-vs-docker-a-comparative-analysis-eccc74ea80e3>

Frontend (HTML, CSS, JavaScript)

- **A Complete Guide to Flexbox** : <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>
- **The Fetch API** :
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch
- **Understanding Event Delegation** : <https://javascript.info/event-delegation>
- **HTML sémantique : C'est quoi et comment l'utiliser ?**:
<https://www.google.com/search?q=https://fr.semrush.com/blog/html-semantique/>
- **Toaster (A simple toast notification library)** :
<https://www.google.com/search?q=https://github.com/script-tag/Toaster>
- **Thymeleaf: Improving templates** :<https://www.thymeleaf.org/doc/articles/layouts.html>

11.2. Vulnérabilités et Mesures de Sécurité

La veille menée a permis d'identifier les risques de sécurité courants et d'y apporter des réponses techniques standards et robustes, directement intégrées dans le framework Spring.

- Hachage des Mots de Passe : Les mots de passe ne sont jamais stockés en clair. Conformément aux recommandations de sécurité actuelles, l'algorithme BCrypt est utilisé via Spring Security.
- Prévention des Injections SQL : L'application utilise l'ORM Spring Data JPA pour toutes les interactions avec la base de données. En utilisant des requêtes typées et des Repositories, le framework prépare les requêtes de manière sécurisée (PreparedStatement), ce qui élimine nativement le risque d'injection de code SQL malveillant.
- Protection contre le Cross-Site Scripting (XSS) : Le moteur de template Thymeleaf est configuré par défaut pour échapper automatiquement toutes les données dynamiques qui sont affichées dans les pages HTML. Cette mesure empêche un utilisateur malveillant d'injecter du code JavaScript qui pourrait être exécuté par le navigateur d'un autre utilisateur.
- Gestion des Secrets : Les informations sensibles (identifiants de base de données, clés API, etc.) sont externalisées. Elles ne sont pas écrites en dur dans le code ou dans les fichiers de configuration versionnés sur Git. Pour l'environnement de production, elles sont gérées en tant que secrets Docker, et injectées dans les conteneurs au moment de leur démarrage.

12. Conclusion et Rétrospective

12.1. Bilan du Projet : Un MVP Réussi

Le projet KundApp a atteint avec succès son objectif principal : la livraison d'un Produit Minimum Viable (MVP) fonctionnel, validant le concept et l'architecture technique. Ce parcours a permis de mettre en pratique et de consolider un large éventail de compétences clés indispensables au métier de Concepteur Développeur d'Applications :

- Architecture Logicielle : Conception et mise en œuvre d'un écosystème microservices cohérent et découplé.
- Développement Backend : Maîtrise de l'écosystème Java/Spring (Spring Boot, Spring Cloud, Spring Security) pour créer des services robustes.
- Gestion des Données : Utilisation de bases de données relationnelles (MySQL avec JPA/Hibernate) et NoSQL (MongoDB) adaptées aux besoins de chaque service.
- DevOps : Conteneurisation de l'ensemble de l'application avec Docker et orchestration via Docker Compose, en incluant des stratégies de démarrage contrôlé (healthchecks).
- Sécurité : Implémentation de la sécurité applicative à plusieurs niveaux (authentification, autorisations par rôle).

12.2. Leçons Apprises : Des Défis Techniques à la Montée en Compétence

Ce projet a été un parcours riche en défis techniques, qui ont été autant d'opportunités d'apprentissage concrètes.

Maîtrise de l'Écosystème Microservices

Le défi technique le plus formateur a été la gestion de la cohérence des données dans une architecture distribuée. La mise en œuvre d'une transaction distribuée pour les inscriptions (inspirée du Pattern Saga) a été particulièrement complexe. Assurer la cohérence entre le débit des crédits dans ms-identity et la validation de l'inscription dans ms-coursemgmt, notamment la gestion du rollback, a nécessité une conception rigoureuse. Cette expérience m'a permis de comprendre en profondeur les problématiques de communication et de consistance inhérentes à ce type d'architecture.

Rétrospective sur l'Architecture des Rôles (La Dette Technique Identifiée)

La plus grande leçon de ce projet réside dans l'analyse critique d'un choix de conception initial. L'architecture a été conçue autour d'un système où chaque utilisateur possède un rôle unique et exclusif (CLIENT, TEACHER ou ADMIN).

Si cette approche a simplifié le développement initial, elle a rapidement révélé ses limites :

Projet : KundApp

- Duplication du Code : Elle a engendré une répétition importante de la logique dans les controllers, les templates Thymeleaf et les fichiers JavaScript, complexifiant la maintenance.
- Manque de Flexibilité Métier : Plus important encore, elle interdit des cas d'usage essentiels, comme la possibilité pour un enseignant de s'inscrire au cours d'un autre, car il ne peut pas cumuler les rôles.

Rendre le projet en l'état est un choix délibéré de gestion de projet, privilégiant la stabilité de la livraison à un refactoring précipité. Cette analyse critique et la définition d'un plan de correction clair constituent une part essentielle des compétences d'un Concepteur.

12.3. Perspectives et Axes d'Amélioration Futurs

KundApp est une base solide destinée à évoluer. La feuille de route des prochaines itérations est claire et s'articule autour de deux axes principaux.

Résorption de la Dette Technique

La priorité absolue est de corriger la gestion des rôles pour assainir l'architecture :

1. Fondations : Refactoriser ms-identity pour permettre à un utilisateur de posséder plusieurs rôles (via @ElementCollection ou une relation ManyToMany).
2. Backend : Fusionner les controllers dupliqués en services unifiés, avec une sécurité fine appliquée sur chaque méthode.
3. Frontend : Unifier les vues en un tableau de bord unique (dashboard.html) utilisant les fragments Thymeleaf et sec:authorize pour un affichage dynamique.

Évolutions Fonctionnelles et Techniques

Une fois l'architecture assainie, les évolutions suivantes seront envisagées :

- Authentification Robuste : Migrer vers une solution standardisée comme Keycloak pour gérer l'authentification et autoriser des applications mobiles via des tokens JWT.
- Monétisation : Intégrer une solution de paiement telle que Stripe pour l'achat de crédits.
- Fonctionnalités Métier : Mettre en place une liste d'attente automatisée pour les cours complets et permettre un accès public (non connecté) au catalogue des séances.