USER SERVICE

Documentation Technique

Portail de Suivi du Doctorat

Architecture Microservices

Information	Valeur
Microservice	user-service
Port	8081
Base de données	MySQL - user_db
Framework	Spring Boot 3.x + Spring Security
Authentification	JWT (JSON Web Token)
Date	01/01/2026

TABLE DES MATIÈRES

1. PRÉSENTATION GÉNÉRALE DU MICROSERVICE

1.1 Rôle Fonctionnel

Le **user-service** est le microservice central du portail doctoral, responsable de la gestion complète des utilisateurs : authentification, autorisation, gestion des profils et orchestration du workflow de candidature.

■ **En termes simples :** Ce service est le "gardien" du portail. Il vérifie qui vous êtes (authentification) et ce que vous avez le droit de faire (autorisation). C'est comme le vigile à l'entrée d'un bâtiment qui vérifie votre badge et vous donne accès aux étages autorisés.

1.2 Responsabilités Principales

- **Authentification JWT** : Connexion sécurisée avec tokens (comme un badge temporaire)
- **Gestion des Utilisateurs** : CRUD complet sur les profils (Candidats, Doctorants, Directeurs, Admins)
- **Workflow de Candidature** : Orchestration du processus Admin → Directeur → Doctorant
- **Gestion des Rôles** : Contrôle d'accès basé sur 4 rôles distincts
- **Stockage de Fichiers** : Upload et téléchargement des documents (CV, diplômes)
- **Exposition d'API** : Endpoints REST pour les autres microservices (via Feign)

1.3 Les 4 Rôles du Système

Rôle	Description	Permissions
CANDIDAT	Personne qui vient de s'inscrire	Soumettre candidature, voir son profil
DOCTORANT	Candidat accepté en thèse	Gérer sa thèse, demander dérogations
DIRECTEUR_THESE	Encadrant de doctorants	Valider candidatures, superviser
ADMIN	Administrateur du portail	Tout gérer, valider en premier

■ **Analogie :** Pensez à une université : le CANDIDAT est comme un étudiant qui postule, le DOCTORANT est inscrit officiellement, le DIRECTEUR_THESE est le professeur encadrant, et l'ADMIN est le service de scolarité.

2. ARCHITECTURE INTERNE

2.1 Organisation du Code

Le microservice suit une architecture en couches classique Spring Boot :

Couche	Package	Rôle
Controllers	controllers	Reçoit les requêtes HTTP et renvoie les réponses
Services	services	Contient la logique métier (les règles)
Repositories	repositories	Accède à la base de données
Security	security	Gère JWT et l'authentification
Entities	entities	Représente les tables de la BDD
DTOs	dto	Objets pour transférer les données
Mappers	mappers	Convertit Entity ↔ DTO
Config	config	Configuration Spring

■■ **Pourquoi cette architecture ?** Chaque couche a une responsabilité unique. C'est comme une chaîne de production : le Controller reçoit la commande, le Service la traite, et le Repository stocke/récupère les données. Cette séparation facilite la maintenance et les tests.

2.2 Flux d'une Requête Authentifiée

```
1. Client envoie requête avec token JWT dans le header "Authorization: Bearer xxx" 2. JwtAuthenticationFilter intercepte et extrait le token 3. JwtService vérifie la validité du token (signature, expiration) 4. CustomUserDetailsService charge l'utilisateur depuis la BDD 5. SecurityContext est mis à jour avec l'utilisateur authentifié 6. La requête atteint le Controller approprié 7. Le Service exécute la logique métier 8. La réponse est renvoyée au client
```

3. FICHIERS DE CONFIGURATION

3.1 UserServiceApplication.java

Rôle : Point d'entrée principal du microservice Spring Boot.

Annotations utilisées :

- **@SpringBootApplication** : Active l'auto-configuration, le scan de composants et la configuration Spring
- **@EnableFeignClients** : Permet d'utiliser OpenFeign pour appeler d'autres microservices

■ **Explication simple :** C'est comme le bouton "ON" de votre application. Quand vous lancez main(), Spring Boot démarre, scanne tous les composants (@Service, @Controller, etc.) et configure automatiquement tout ce qui est nécessaire.

3.2 application.properties

Rôle : Fichier de configuration centralisé contenant tous les paramètres du service.

Paramètre	Valeur	Description
server.port	8081	Port d'écoute du service
spring.datasource.url	jdbc:mysql://localhost:3306/user_db	Connexion MySQL
spring.jpa.hibernate.ddl-auto	update	Mise à jour auto du schéma BDD
eureka.client.service-url	http://localhost:8761/eureka/	Inscription à Eureka
jwt.secret	clé encodée base64	Clé secrète pour signer les tokens
jwt.expiration	86400000	Durée du token (24h en ms)

■ **Pourquoi 86400000 ms ?** C'est 24 heures en millisecondes (24 × 60 × 60 × 1000). Après 24h, l'utilisateur devra se reconnecter ou utiliser son refresh token.

3.3 SecurityConfig.java

Rôle : Configuration centrale de Spring Security - définit qui peut accéder à quoi.

Fonctionnalités clés :

- Désactivation CSRF (inutile pour API REST stateless)
- Configuration CORS pour Angular (localhost:4200)
- Définition des endpoints publics (/api/auth/**, /api/files/**)
- Protection des endpoints par rôle (ADMIN, DIRECTEUR_THESE)
- Ajout du filtre JWT avant le filtre standard de Spring
- Mode stateless (pas de session serveur)

■ **Stateless, c'est quoi ?** Le serveur ne garde aucune information de session. Chaque requête doit contenir le token JWT. C'est comme montrer son badge à chaque porte, plutôt que de se faire reconnaître par le gardien.

4. MODÈLE DE DONNÉES

4.1 Role.java (Énumération)

Rôle : Définit les 4 rôles possibles dans le système.

Valeur	Description
ADMIN	Administrateur avec tous les droits
CANDIDAT	Utilisateur qui vient de s'inscrire (rôle initial)
DOCTORANT	Candidat accepté et inscrit en thèse
DIRECTEUR_THESE	Professeur encadrant des doctorants

■ **Cycle de vie d'un rôle :** Un utilisateur s'inscrit comme CANDIDAT → s'il est accepté, il devient DOCTORANT. Les DIRECTEUR_THESE et ADMIN sont créés par un admin.

4.2 User.java (Entité JPA)

Rôle : Représente la table 'users' en base de données. Implémente UserDetails pour Spring Security.

Champ	Type	Description
id	Long (PK)	Identifiant unique auto-généré
matricule	String (unique)	Identifiant de connexion
password	String	Mot de passe hashé (BCrypt)
email	String (unique)	Adresse email
nom, prenom	String	Identité de l'utilisateur
role	Role (enum)	Rôle dans le système
etat	String	État du workflow (EN_ATTENTE_ADMIN, VALIDE...)
directeurId	Long	ID du directeur assigné
titreThese	String	Sujet de thèse (assigné par directeur)
nbPublications	Integer	Nombre de publications du doctorant
nbConferences	Integer	Nombre de conférences

heuresFormation	Integer	Heures de formation effectuées

■ **UserDetails, c'est quoi ?** C'est une interface de Spring Security. En l'implémentant, notre entité User devient directement utilisable pour l'authentification. Spring sait comment récupérer le username, le password et les rôles.

5. COUCHE REPOSITORY

5.1 UserRepository.java

Rôle : Interface d'accès aux données qui étend JpaRepository. Spring Data JPA génère automatiquement les implémentations.

Méthode	Description
findByMatricule()	Trouve un utilisateur par son matricule
findByEmail()	Trouve un utilisateur par son email
existsByMatricule()	Vérifie si un matricule existe déjà
existsByEmail()	Vérifie si un email existe déjà
findByRole()	Liste tous les utilisateurs d'un rôle
findByEtat()	Liste les utilisateurs par état de workflow
findByDirecteurId()	Liste les doctorants d'un directeur

■ **La magie de Spring Data JPA :** Vous déclarez juste le nom de la méthode (findByMatricule), et Spring génère automatiquement le SQL ! Le nom de la méthode est analysé : find + By + Matricule → SELECT * FROM users WHERE matricule = ?

6. OBJETS DE TRANSFERT (DTOs)

Les DTOs (Data Transfer Objects) sont des objets utilisés pour transférer des données entre les couches de l'application, notamment entre le backend et le frontend.

■ **Pourquoi des DTOs ?** L'entité User contient le mot de passe hashé - on ne veut pas l'envoyer au frontend ! Le DTO permet de choisir exactement quels champs exposer. C'est comme un filtre entre vos données internes et le monde extérieur.

DTO	Utilisation	Champs Clés
LoginRequest	Requête de connexion	username, password
RegisterRequest	Requête d'inscription	matricule, email, password, nom, prenom, telephone
AuthResponse	Réponse après login/register	accessToken, refreshToken, userId, role, etat
UserDTO	Représentation utilisateur	Tous les champs sauf password
ChangePasswordRequest	Changement de mot de passe	oldPassword, newPassword, confirmPassword

6.1 UserMapper.java

Rôle : Convertit les entités User en DTOs et vice-versa.

Méthodes : toDTO(User) → UserDTO, toEntity(UserDTO) → User, updateEntityFromDTO()

7. SÉCURITÉ ET AUTHENTIFICATION JWT

7.1 Qu'est-ce que JWT ?

JWT (JSON Web Token) est un standard ouvert pour transmettre des informations de manière sécurisée. Un token JWT est composé de 3 parties séparées par des points :

```
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJNQVQwMDEiLCJpYXQiOjE3MDk4MjQwMDB9.signature |___
Header    |        Payload         |   Signature   |
```

■ **Analogie du ticket de cinéma :** Le token JWT est comme un ticket. Le Header dit quel type de ticket c'est, le Payload contient vos informations (place, séance), et la Signature prouve que c'est un vrai ticket émis par le cinéma (pas une copie).

7.2 JwtService.java

Rôle : Service central pour toutes les opérations liées aux tokens JWT.

Méthode	Description
generateToken(UserDetails)	Crée un nouveau token JWT (24h)
generateRefreshToken()	Crée un refresh token (7 jours)
extractUsername()	Extrait le matricule du token
isTokenValid()	Vérifie signature + expiration
isTokenExpired()	Vérifie si le token a expiré
getSigningKey()	Récupère la clé de signature (HMAC-SHA256)

7.3 JwtAuthenticationFilter.java

Rôle : Filtre qui intercepte TOUTES les requêtes HTTP pour vérifier le token JWT.

Processus :

- 1. Récupère le header 'Authorization'
- 2. Vérifie qu'il commence par 'Bearer '
- 3. Extrait le token (après 'Bearer ')
- 4. Extrait le username du token via JwtService
- 5. Charge l'utilisateur depuis la BDD
- 6. Valide le token (signature + expiration)
- 7. Met à jour le SecurityContext avec l'utilisateur authentifié
- 8. Passe la main au Controller

7.4 CustomUserDetailsService.java

Rôle : Implémente UserDetailsService de Spring Security pour charger les utilisateurs depuis la BDD.

Méthode principale : loadUserByUsername(matricule) - Cherche l'utilisateur par matricule et retourne un objet UserDetails avec ses autorités (rôles).

8. COUCHE SERVICE (LOGIQUE MÉTIER)

8.1 AuthService.java

Rôle : Gère toutes les opérations d'authentification (inscription, connexion, tokens).

Méthode	Description
register(RegisterRequest)	Inscription simple (JSON)
registerWithFiles()	Inscription avec upload de fichiers (CV, diplômes)
login(LoginRequest)	Connexion (matricule OU email)
refreshToken()	Renouvelle le token d'accès
changePassword()	Change le mot de passe
getCurrentUser()	Retourne le profil de l'utilisateur connecté

■ **Login intelligent :** L'utilisateur peut se connecter avec son matricule OU son email. Le service essaie d'abord de trouver par matricule, puis par email si non trouvé.

8.2 UserService / UserServiceImpl.java

Rôle : Gère la logique métier des utilisateurs et le workflow de candidature.

Catégorie	Méthodes
CRUD	createUser(), updateUser(), deleteUser(), getUserById()
Recherche	getUserByUsername(), getUserByEmail(), getUsersByRole()
Workflow Admin	validerCandidature(), validerCandidatureAvecDirecteur(), refuserCandidature()
Workflow Directeur	validerCandidatureDirecteur(), validerCandidatureDirecteurAvecSujet()
Gestion Directeur	getDoctorantsByDirecteur()

8.3 FileStorageService.java

Rôle : Gère le stockage des fichiers uploadés (CV, diplômes, lettres de motivation).

Fonctionnement : Génère un nom unique (UUID), stocke le fichier dans le dossier 'uploads/', et retourne le nom du fichier pour stockage en BDD.

9. COUCHE CONTROLLER (API REST)

9.1 AuthController.java

Base URL : /api/auth

Méthode	Endpoint	Description
POST	/register	Inscription simple (JSON)
POST	/register-with-files	Inscription avec fichiers (multipart)
POST	/login	Connexion et génération des tokens
GET	/me	Profil de l'utilisateur connecté
POST	/change-password	Changer son mot de passe
GET	/validate	Valider si le token est encore valide
POST	/logout	Déconnexion (côté client)

9.2 UserController.java

Base URL : /api/users

Méthode	Endpoint	Rôle Requis
POST	/	Public (temporaire)
GET	/	Authentifié
GET	/{id}	Public (inter-services)
GET	/{id}/dto	Public (inter-services)
PUT	/{id}	ADMIN
DELETE	/{id}	ADMIN
PUT	/{id}/validate-admin	ADMIN
PUT	/{id}/refuse	ADMIN
PUT	/{id}/validate-directeur	ADMIN, DIRECTEUR_THESE
PUT	/{id}/refuse-directeur	ADMIN, DIRECTEUR_THESE

GET	/directeur/{id}/doctorants	Authentifié

9.3 FileController.java

Base URL : /api/files

Endpoint : GET /{filename} - Télécharge un fichier par son nom (détection MIME automatique)

10. WORKFLOW DE CANDIDATURE

Le processus de candidature passe par plusieurs étapes de validation :

```
1. INSCRIPTION (Candidat) ■■→ Rôle: CANDIDAT, État: EN_ATTENTE_ADMIN 2. VALIDATION ADMIN ■■→ Accepte + Assigne directeur → État: EN_ATTENTE_DIRECTEUR ■■→ Refuse avec motif → État: REFUSE 3. VALIDATION DIRECTEUR ■■→ Accepte + Assigne sujet thèse → Rôle: DOCTORANT, État: VALIDE ■■→ Refuse avec motif → État: REFUSE
```

■ **Pourquoi ce workflow ?** L'admin vérifie d'abord les conditions administratives (documents, éligibilité) et assigne un directeur. Ensuite, le directeur valide l'adéquation du candidat avec sa thématique de recherche et définit le sujet.

État	Signification	Prochaine Action
EN_ATTENTE_ADMIN	Candidature soumise	Admin doit valider/refuser
EN_ATTENTE_DIRECTEUR	Validée par admin	Directeur doit valider/refuser
VALIDE	Accepté comme doctorant	Fin du processus
REFUSE	Candidature refusée	Fin du processus

11. QUESTIONS / RÉPONSES POTENTIELLES

Q1 : Pourquoi utiliser JWT plutôt que des sessions classiques ?

R : JWT est stateless : le serveur n'a pas besoin de stocker l'état de session. C'est idéal pour les microservices car chaque service peut valider le token indépendamment. De plus, ça facilite le scaling horizontal.

Q2 : Comment le mot de passe est-il sécurisé ?

R : On utilise BCrypt, un algorithme de hachage adaptatif. Il est lent volontairement (pour résister aux attaques brute-force) et inclut un salt unique pour chaque hash. Même deux mots de passe identiques auront des hash différents.

Q3 : Pourquoi l'entité User implémente UserDetails ?

R : C'est une intégration native avec Spring Security. En implémentant cette interface, notre User devient directement utilisable pour l'authentification sans adaptateur. Spring Security sait comment récupérer username, password et authorities.

Q4 : Comment les autres microservices accèdent-ils aux données utilisateur ?

R : Via les endpoints /api/users/{id}/dto qui sont publics (permitAll). Les autres services utilisent OpenFeign pour appeler ces endpoints. Le fallback retourne des données par défaut si user-service est indisponible.

Q5 : Que se passe-t-il si le token JWT expire ?

R : Le client reçoit une erreur 401. Il doit alors utiliser son refresh token (valide 7 jours) pour obtenir un nouveau access token. Si le refresh token est aussi expiré, l'utilisateur doit se reconnecter.

Q6 : Pourquoi avoir 2 workflows de validation (Admin puis Directeur) ?

R : L'admin vérifie les aspects administratifs (documents, éligibilité) et affecte le candidat à un directeur. Le directeur évalue ensuite l'adéquation scientifique et définit le sujet de thèse. C'est une séparation des responsabilités.

12. CONCLUSION

Le **user-service** est le pilier central du portail doctoral, assurant la sécurité et la gestion des identités pour l'ensemble de l'écosystème microservices.

Points Forts de l'Architecture :

- **Sécurité JWT Robuste** : Tokens signés avec HMAC-SHA256, expiration configurée, refresh tokens
- **Authentification Flexible** : Connexion par matricule OU email
- **Workflow Multi-Étapes** : Validation Admin → Directeur avec traçabilité complète
- **API Bien Documentée** : Endpoints REST cohérents et sécurisés par rôle
- **Résilience** : Endpoints publics pour communication inter-services (Feign)
- **Gestion de Fichiers** : Upload sécurisé avec UUID pour éviter les conflits

Valeur dans l'Architecture Globale :

Ce microservice est le point d'entrée sécurisé pour tous les utilisateurs du portail. Il fournit l'authentification centralisée (JWT) utilisée par tous les autres services, gère le cycle de vie complet des utilisateurs (de candidat à doctorant), et expose des endpoints pour que les autres microservices puissent récupérer les informations utilisateur via OpenFeign.

■ **En résumé :** Sans user-service, aucun utilisateur ne pourrait s'authentifier, aucun rôle ne serait vérifié, et les autres microservices ne sauraient pas qui fait les requêtes. C'est le gardien et l'annuaire du portail doctoral.