NOTIFICATION SERVICE

Documentation Technique

Portail de Suivi du Doctorat

Architecture Microservices

Information	Valeur
Microservice	notification-service
Port	8084
Base de données	MySQL - notification_db
Framework	Spring Boot 3.x + Spring Kafka
Email	SMTP Gmail + Thymeleaf Templates
Date	01/01/2026

TABLE DES MATIÈRES

1. PRÉSENTATION GÉNÉRALE DU MICROSERVICE

1.1 Rôle Fonctionnel

Le **notification-service** est le microservice responsable de l'envoi de toutes les notifications du portail doctoral. Il consomme les événements Kafka publiés par les autres services et envoie des emails HTML personnalisés aux utilisateurs concernés.

■ **En termes simples :** Ce service est le "facteur" du portail. Quand quelque chose d'important se passe (inscription créée, soutenance planifiée...), les autres services envoient un message à Kafka, et notification-service le reçoit et envoie un email au bon destinataire.

1.2 Responsabilités Principales

- **Consommation Kafka** : Écoute les topics d'événements des autres services
- **Génération d'Emails HTML** : Utilise Thymeleaf pour créer des emails professionnels
- **Envoi SMTP** : Envoie via Gmail SMTP avec authentification sécurisée
- **Traçabilité** : Sauvegarde chaque notification en base de données
- **Retry Automatique** : Réessaie les envois échoués (max 3 fois)
- **API de Consultation** : Permet de consulter l'historique des notifications

1.3 Types de Notifications

Catégorie	Types	Déclencheur
Inscriptions	CREATED, SUBMITTED, APPROVED, REJECTED	inscription-service
Soutenances	CREATED, PREREQUIS_VALIDATED, SCHEDULED, COMPLETED	soutenance-service
Jury	JURY_INVITATION	soutenance-service
Compte	ACCOUNT_CREATED, PASSWORD_RESET	user-service
Rappels	REINSCRIPTION_REMINDER, SOUTENANCE_REMINDER	Scheduler

2. ARCHITECTURE INTERNE

2.1 Pattern Event-Driven

Le service utilise une architecture événementielle (Event-Driven). Il ne reçoit pas de requêtes HTTP des autres services, mais consomme des événements Kafka de manière asynchrone.

```
■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■ ■
inscription-service■ → ■ KAFKA ■ → ■ notification-service ■ ■ soutenance-service
■ ■ BROKER ■ ■ ■ ■ user-service ■ ■ (Topics) ■ ■ Consumers → Email ■
■■■■■■■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■ ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
PRODUCTEURS MESSAGE CONSOMMATEUR
```

■ **Pourquoi Event-Driven ?** Les services sont découplés. inscription-service n'a pas besoin de connaître notification-service. Il publie juste un événement, et celui qui est intéressé le consomme. Si notification-service est en panne, les messages restent dans Kafka et seront traités plus tard.

2.2 Organisation du Code

Package	Rôle
config	Configuration Kafka (consumers), Async (thread pools), Thymeleaf
consumers	Listeners Kafka (@KafkaListener)
services	EmailService (SMTP), NotificationService (logique)
controllers	API REST pour consultation et tests
entities	Notification (entité JPA)
enums	NotificationType, NotificationStatus
events	Classes correspondant aux événements Kafka
repositories	NotificationRepository

3. FICHIERS DE CONFIGURATION

3.1 NotificationServiceApplication.java

Annotations clés :

- @SpringBootApplication - Configuration auto Spring Boot
- @EnableDiscoveryClient - Enregistrement Eureka
- @EnableKafka - Active les listeners Kafka
- @EnableAsync - Active l'exécution asynchrone (@Async)
- @EnableScheduling - Active les tâches planifiées (@Scheduled)

3.2 application.properties

Paramètre	Valeur	Description
server.port	8084	Port du service
spring.kafka.bootstrap-servers	localhost:9092	Broker Kafka
spring.kafka.consumer.group-id	notification-group	Groupe de consumers
spring.mail.host	smtp.gmail.com	Serveur SMTP Gmail
spring.mail.port	587	Port SMTP (TLS)
spring.mail.username	xxx@gmail.com	Compte Gmail
spring.mail.password	xxxx xxxx xxxx	App Password Gmail

■ **App Password Gmail :** Ce n'est pas le mot de passe Gmail normal ! Il faut générer un "mot de passe d'application" dans les paramètres de sécurité Google (Compte Google → Sécurité → Mots de passe des applications).

3.3 KafkaConfig.java

Rôle : Configure les ConsumerFactory spécifiques pour chaque type d'événement.

Pourquoi plusieurs factories ? Chaque événement a une structure différente. Une factory dédiée avec JsonDeserializer typé garantit une désérialisation correcte.

- inscriptionCreatedKafkaListenerContainerFactory
- inscriptionStatusKafkaListenerContainerFactory
- soutenanceCreatedKafkaListenerContainerFactory
- soutenanceStatusKafkaListenerContainerFactory

3.4 AsyncConfig.java

Rôle : Configure les pools de threads pour l'exécution asynchrone.

Pool	Core/Max Threads	Queue	Usage
emailExecutor	2 / 5	100	Envoi d'emails
notificationExecutor	3 / 10	200	Traitement notifications

■ **Pourquoi asynchrone ?** L'envoi d'email peut prendre 1-2 secondes. Sans @Async, le consumer Kafka serait bloqué. Avec un pool de threads dédié, plusieurs emails peuvent être envoyés en parallèle.

3.5 ThymeleafConfig.java

Rôle : Configure le moteur de templates pour générer les emails HTML.

Chemin des templates : classpath:/templates/emails/*.html

4. MODÈLE DE DONNÉES

4.1 NotificationType.java (Enum)

Rôle : Catégorise les 20+ types de notifications possibles.

Catégorie	Types
Inscriptions	INSCRIPTION_CREATED, SUBMITTED, APPROVED, REJECTED, PENDING_VALIDATION
Réinscriptions	REINSCRIPTION_REMINDER, REINSCRIPTION_APPROVED
Soutenances	SOUTENANCE_CREATED, PREREQUIS_VALIDATED, JURY_PROPOSED, AUTHORIZED, SCHEDULED
Rapports	RAPPORT_SUBMITTED, RAPPORT_FAVORABLE, RAPPORT_DEFAVORABLE
Comptes	ACCOUNT_CREATED, PASSWORD_RESET
Jury	JURY_INVITATION, VALIDATION_REQUIRED

4.2 NotificationStatus.java (Enum)

Statut	Description
PENDING	En attente d'envoi
SENT	Envoyée avec succès
FAILED	Échec d'envoi
RETRY	En attente de réessai

4.3 Notification.java (Entity)

Table : notifications

Champ	Type	Description
id	Long (PK)	Identifiant unique
type	NotificationType	Type de notification
status	NotificationStatus	Statut d'envoi
recipientEmail	String	Email du destinataire

recipientName	String	Nom du destinataire
subject	String	Sujet de l'email
content	TEXT	Contenu (optionnel)
referenceId	Long	ID de l'entité liée (inscription/soutenance)
referenceType	String	Type: INSCRIPTION, SOUTENANCE, USER
createdAt	LocalDateTime	Date de création
sentAt	LocalDateTime	Date d'envoi effectif
retryCount	int	Nombre de tentatives (max 3)
errorMessage	String	Message d'erreur si échec
metadata	TEXT (JSON)	Variables du template sérialisées

■ **Pourquoi stocker les notifications ?** Traçabilité complète : on sait quel email a été envoyé à qui, quand, et si ça a marché. Utile pour le debug et l'audit.

5. ÉVÉNEMENTS KAFKA CONSOMMÉS

Le service consomme les événements publiés par inscription-service et soutenance-service.

5.1 Topics Écoutés

Topic	Événement	Producteur
inscription-created	InscriptionCreatedEvent	inscription-service
inscription-status-changed	InscriptionStatusChangedEvent	inscription-service
soutenance-created	SoutenanceCreatedEvent	soutenance-service
soutenance-status-changed	SoutenanceStatusChangedEvent	soutenance-service
jury-invitation	JuryInvitationEvent	soutenance-service

5.2 Structure des Événements

Tous les événements héritent de **BaseEvent** (eventId, timestamp, source).

Événement	Champs Clés
InscriptionCreatedEvent	inscriptionId, doctorantEmail/Nom/Prenom, sujetThese, directeurTheseEmail
InscriptionStatusChangedEvent	oldStatus, newStatus, commentaire, validatedBy
SoutenanceCreatedEvent	soutenanceId, doctorantEmail, directeurTheseEmail
SoutenanceStatusChangedEvent	dateSoutenance, lieu, presidentJury, rapporteur1/2
JuryInvitationEvent	membreJuryEmail/Nom, roleJury, dateSoutenance, lieu

6. CONSUMERS KAFKA

6.1 InscriptionEventConsumer.java

Rôle : Écoute les événements liés aux inscriptions et déclenche les notifications.

Méthodes :

- **handleInscriptionCreated()** : Notifie le doctorant (confirmation) + le directeur (validation requise)
- **handleInscriptionStatusChanged()** : Notifie selon le nouveau statut (APPROVED → félicitations, REJECTED → information)

Logique de mapping statut → notification :

Nouveau Statut	NotificationType	Template
APPROVED / VALIDEE	INSCRIPTION_APPROVED	inscription-approved
REJECTED / REJETEE	INSCRIPTION_REJECTED	inscription-rejected
SUBMITTED / SOUMISE	INSCRIPTION_SUBMITTED	inscription-submitted
PENDING_VALIDATION	INSCRIPTION_PENDING_VALIDATION	inscription-pending

6.2 SoutenanceEventConsumer.java

Rôle : Écoute les événements liés aux soutenances et au jury.

Méthodes :

- **handleSoutenanceCreated()** : Notifie le doctorant + le directeur
- **handleSoutenanceStatusChanged()** : Notifie selon le statut (PLANIFIEE avec date, TERMINEE = Félicitations Docteur)
- **handleJuryInvitation()** : Envoie l'invitation au membre du jury avec son rôle

■ **Exemple SOUTENANCE_COMPLETED :** Quand le statut passe à TERMINEE, l'email envoyé a pour sujet "■ Félicitations Docteur !" et utilise le template soutenance-completed.html

7. SERVICES (LOGIQUE MÉTIER)

7.1 EmailService.java

Rôle : Envoie les emails via JavaMailSender (SMTP Gmail).

Méthode	Description
sendSimpleEmail()	Envoie un email texte simple
sendHtmlEmail()	Envoie un email HTML avec template Thymeleaf
sendEmailAsync()	Version asynchrone (@Async)
sendEmailWithAttachment()	Email avec pièce jointe
isMailServiceHealthy()	Health check du service mail

Processus sendHtmlEmail() :

- 1. Crée un MimeMessage avec MimeMessageHelper
- 2. Configure from (gmail), to (destinataire), subject
- 3. Charge le template Thymeleaf et injecte les variables (Context)
- 4. Génère le HTML avec templateEngine.process()
- 5. Envoie via mailSender.send()

7.2 NotificationService.java

Rôle : Orchestre la création de notification, l'envoi d'email et le suivi.

Méthode	Description
createAndSendNotification()	Crée en BDD + envoie email (méthode principale)
sendNotificationEmail()	Envoie l'email (@Async) et met à jour le statut
retryFailedNotifications()	Réessaie les échecs (@Scheduled toutes les 5 min)
getTemplateNameForType()	Mappe NotificationType → nom du template
getNotificationStats()	Statistiques (pending, sent, failed, today)

■ **Retry automatique :** Le @Scheduled(fixedRate=300000) exécute retryFailedNotifications() toutes les 5 minutes. Les notifications avec status=FAILED et retryCount < 3 sont réessayées.

8. CONTROLLERS (API REST)

8.1 NotificationController.java

Base URL : /api/notifications

Méthode	Endpoint	Description
GET	/	Liste toutes les notifications
GET	/{id}	Détails d'une notification
GET	/recipient/{email}	Notifications d'un destinataire
GET	/status/{status}	Notifications par statut
GET	/reference/{type}/{id}	Notifications d'une inscription/soutenance
GET	/stats	Statistiques (pending, sent, failed)
POST	/retry-failed	Force le réessai des échecs
GET	/health	Health check avec stats

8.2 TestController.java

Base URL : /api/test

Rôle : Endpoints de test pour vérifier le bon fonctionnement (à désactiver en production).

Endpoint	Usage
GET /ping	Vérifie que le service est up
POST /send-simple-email	Test envoi email texte
POST /send-template-email	Test envoi email HTML avec template
POST /create-notification	Test création notification complète
GET /templates	Liste tous les templates disponibles

9. FLUX COMPLET D'ENVOI D'EMAIL

```
┌──────────────────────────────────────────────────────────────────────────────┐
│ 1. PRODUCTEUR (inscription-service / soutenance-service)                     │
│ → Publie un événement sur le topic Kafka                                     │
│ Exemple: kafkaTemplate.send("soutenance-scheduled", event)                   │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│ 2. KAFKA BROKER                                                              │
│ → Stocke le message dans le topic                                            │
│ → Le garde jusqu'à consommation                                              │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│ 3. CONSUMER (@KafkaListener)                                                 │
│ → SoutenanceEventConsumer.handleSoutenanceStatusChanged()                    │
│ → Extrait les données de l'événement                                         │
│ → Prépare les variables pour le template                                     │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│ 4. NotificationService.createAndSendNotification()                           │
│ → Crée l'entité Notification (status=PENDING)                                │
│ → Sauvegarde en BDD                                                          │
│ → Appelle sendNotificationEmail() en @Async                                  │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│ 5. EmailService.sendHtmlEmail()                                              │
│ → Charge le template Thymeleaf (ex: soutenance-scheduled)                    │
│ → Injecte les variables dans le contexte                                     │
│ → Génère le HTML final                                                       │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
│ 6. JavaMailSender.send()                                                     │
│ → Connexion SMTP à smtp.gmail.com:587                                        │
│ → Authentification avec App Password                                         │
│ → Envoi de l'email                                                           │
└──────────────────────────────────────────────────────────────────────────────┘
                                    ▼
┌──────────────────────────────────────────────────────────────────────────────┐
```

```
■■
■ 7. MISE À JOUR STATUT ■
■ → Succès: status=SENT, sentAt=now() ■
■ → Échec: status=FAILED, retryCount++, errorMessage=... ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■■
```

10. TEMPLATES THYMELEAF

Les templates HTML sont stockés dans **src/main/resources/templates/emails/**

Template	Usage	Variables Principales
inscription-created	Confirmation inscription	doctorantNom, sujetThese, campagneNom
inscription-approved	Inscription validée	doctorantNom, sujetThese
inscription-rejected	Inscription refusée	doctorantNom, commentaire
soutenance-created	Demande soutenance	doctorantNom, sujetThese
soutenance-scheduled	Soutenance planifiée	dateSoutenance, lieu, salle
soutenance-completed	Félicitations Docteur	doctorantNom, mention
jury-invitation	Invitation jury	membreJuryNom, roleLabel, dateSoutenance
validation-required	Action requise	directeurNom, doctorantNom
generic-notification	Template par défaut	recipientName, message

■ **Thymeleaf, c'est quoi ?** Un moteur de template HTML. On écrit du HTML avec des placeholders comme th:text="${doctorantNom}", et Thymeleaf remplace par les vraies valeurs. Résultat : des emails HTML professionnels et dynamiques.

11. QUESTIONS / RÉPONSES POTENTIELLES

Q1 : Pourquoi utiliser Kafka plutôt que des appels REST directs ?

R : Découplage total : le producteur ne sait pas qui consomme. Si notification-service est en panne, les messages s'accumulent dans Kafka et seront traités au redémarrage. Avec REST, l'appel échouerait et l'inscription serait bloquée.

Q2 : Pourquoi plusieurs ConsumerFactory dans KafkaConfig ?

R : Chaque événement a une structure différente (InscriptionCreatedEvent vs SoutenanceCreatedEvent). Une factory dédiée avec un JsonDeserializer typé garantit une désérialisation correcte sans risque de ClassCastException.

Q3 : Que se passe-t-il si l'envoi d'email échoue ?

R : La notification passe en status=FAILED avec le message d'erreur. Le @Scheduled retryFailedNotifications() réessaie toutes les 5 minutes, jusqu'à 3 tentatives max. Après 3 échecs, la notification reste en FAILED pour investigation manuelle.

Q4 : Pourquoi @Async sur sendNotificationEmail() ?

R : L'envoi SMTP peut prendre 1-2 secondes. Sans @Async, le consumer Kafka serait bloqué et ne pourrait pas traiter d'autres événements. Avec un pool de threads dédié, plusieurs emails sont envoyés en parallèle.

Q5 : Comment Gmail autorise-t-il l'envoi ?

R : On utilise un 'App Password' (mot de passe d'application), pas le mot de passe Gmail normal. Il faut l'activer dans les paramètres de sécurité Google avec la validation en 2 étapes.

Q6 : Pourquoi stocker les notifications en base de données ?

R : Traçabilité complète : on peut voir tous les emails envoyés, leur statut, les erreurs. Utile pour le debug, l'audit, et pour répondre aux questions 'Avez-vous reçu l'email ?'

12. CONCLUSION

Le **notification-service** est le système nerveux de communication du portail doctoral, assurant que tous les acteurs (doctorants, directeurs, jury, admin) sont informés en temps réel des événements importants.

Points Forts de l'Architecture :

- **Event-Driven** : Découplage total via Kafka, résilience aux pannes
- **Asynchrone** : Pools de threads dédiés pour envoi parallèle
- **Templates HTML** : Emails professionnels via Thymeleaf
- **Traçabilité** : Toutes les notifications stockées en BDD
- **Retry Automatique** : Réessai des échecs jusqu'à 3 fois
- **20+ Types de Notifications** : Couverture complète des cas d'usage
- **API de Consultation** : Statistiques et historique accessibles

Valeur dans l'Architecture Globale :

Ce microservice centralise toute la communication du portail. Les autres services n'ont qu'à publier des événements Kafka - ils ne gèrent pas l'envoi d'emails. Cela respecte le principe de responsabilité unique (SRP) et facilite l'évolution (ajout de SMS, push notifications...).

■ **En résumé :** notification-service est le messager fiable du portail doctoral. Il garantit que personne ne manque une information importante, de l'inscription à la soutenance, avec des emails professionnels et un suivi complet.