ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

HONORIS UNITED UNIVERSITIES

Rapport de projet J2EE

5ème année

Ingénierie Informatique et Réseaux

Sous le thème

PLATEFORM DE GESTION EVENEMENTS EN MICROSERVICES SPRING BOOT

Réalisé par :
COSTO Mohamed Rayane
LEKFIFI Aymen

Encadré par :
M. Abdelmalek JEOUIT

Année universitaire 2025/2026

Remerciements

Nous tenons d’abord à remercier chaleureusement **M. Abdelmalek JEOUIT** pour son soutien et sa direction tout au long de notre projet de fin d'année. Sans son aide précieuse et ses conseils avisés, nous n'aurions pas pu mener à bien ce travail.

Nous voudrions adresser toute notre gratitude à notre responsable, dont la patience et les orientations ont été essentielles pour la réussite de ce projet. Sa disponibilité et son professionnalisme ont grandement facilité notre travail.

Nous tenons à remercier cordialement tout le corps professoral et administratif de l’École Marocaine des Sciences de l’Ingénieur de Rabat pour leur soutien tout au long de notre formation. Leur enseignement de qualité et leur encadrement ont été déterminants pour notre développement académique et professionnel.

Nos remerciements s'adressent également à nos camarades de promotion pour leur soutien moral et leur esprit de camaraderie, qui ont rendu cette expérience encore plus enrichissante et agréable.

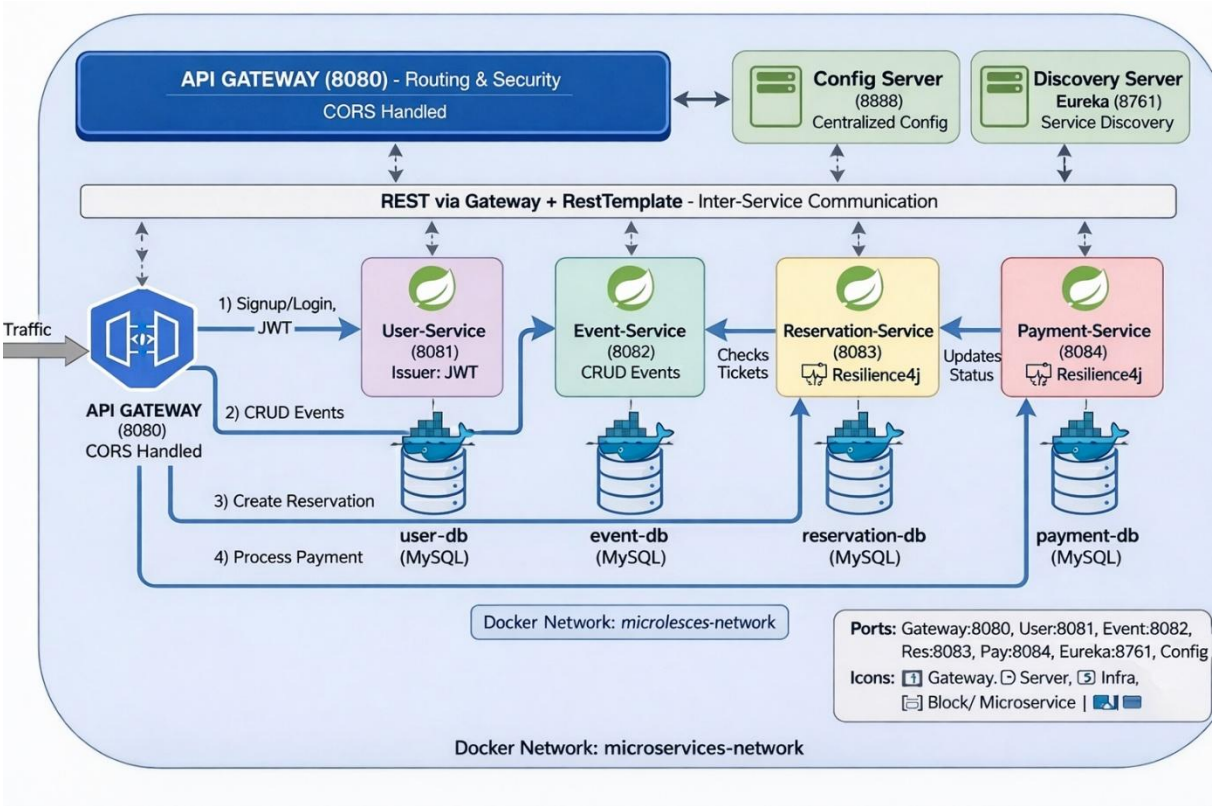
Enfin, merci à tous ceux qui ont participé de près ou de loin à l’élaboration de ce projet. Vos contributions, même les plus modestes, ont été d’une grande importance pour nous. À tous… merci.

Table des matières

1- Introduction

Ce rapport présente l'architecture et les choix techniques d'une plateforme de gestion d'événements (matchs, concerts, conférences) conçue selon une architecture **microservices** avec **Spring Boot** et **Spring Cloud**.
L'objectif principal est de mettre en œuvre une solution **scalable**, **résiliente** et **faiblement couplée**, permettant la création et la gestion d'événements, la réservation de tickets avec contraintes métier, ainsi que la simulation de paiements.
Le projet s'inscrit dans un cadre pédagogique visant à appliquer les concepts fondamentaux des architectures distribuées modernes, notamment la découverte de services, la configuration centralisée, la tolérance aux pannes et la conteneurisation.

2- Architecture globale



La plateforme repose sur une architecture microservices composée de quatre services métiers indépendants : **User-Service**, **Event-Service**, **Reservation-Service** et **Payment-Service**.
L'infrastructure **Spring Cloud** s'articule autour de :

- **API Gateway (8080)**, servant de point d'entrée unique pour toutes les requêtes REST. Elle assure le routage, la gestion du CORS et la réécriture des URLs.
- **Eureka Server (8761)**, utilisé pour la découverte dynamique des services.
- **Config Server (8888)**, permettant la centralisation des configurations via le profil *natif*.

Chaque microservice dispose de sa **base de données MySQL dédiée** (*user-db*, *event-db*, *reservation-db*, *payment-db*), garantissant l'isolation des données et le respect du découplage.

La communication interservices est **synchrone**, réalisée via des appels REST transitant par l'API Gateway, avec l'utilisation de **RestTemplate** côté services afin de conserver une stack simple et lisible.

La **résilience** est assurée par **Resilience4j**, appliqué sur les services critiques **Reservation-Service** et **Payment-Service** à l'aide de circuit breakers.

La sécurité repose sur des **JWT** émis par le User-Service après authentification, tandis que la gestion du **CORS** est centralisée au niveau du Gateway.

Le monitoring est assuré par **Spring Boot Actuator**, et l'ensemble de la stack est orchestré via **Docker Compose** sur un réseau commun nommé *microservices-network*.

Enfin, chaque microservice expose une documentation **Swagger/OpenAPI** facilitant les tests et la validation fonctionnelle.

3- Détails par microservice

3-1 User-Service (8081)

Le User-Service est responsable de la gestion des comptes utilisateurs. Il fournit les fonctionnalités d'inscription, d'authentification et de consultation des utilisateurs.

Il émet les **JWT** utilisés pour sécuriser les échanges.

La persistance est assurée via une base **MySQL user-db**.

Les technologies principales utilisées sont **Spring Boot**, **Spring Security**, **JWT** et **Swagger**.

3-2 Event-Service (8082)

Le Event-Service gère le **CRUD des événements** ainsi que l'inscription des participants à un événement donné.

Il s'appuie sur une base **MySQL event-db** et utilise **Spring Boot** et **Swagger** pour l'exposition et la documentation des APIs.

3-3 Reservation-Service (8083)

Le Reservation-Service permet la création de réservations avec des règles métier strictes, notamment une **limite de 4 tickets par utilisateur** et la vérification de la disponibilité des places via le Event-Service.
Toute réservation est initialisée avec le statut **PENDING_PAYMENT**.
La tolérance aux pannes est assurée par **Resilience4j** lors des appels vers le Event-Service.
Les données sont stockées dans **MySQL reservation-db**.

3-4 Payment-Service (8084)

Le Payment-Service simule le processus de paiement et met à jour le statut des réservations.
Il utilise **Resilience4j** pour protéger les appels vers le Reservation-Service et persiste les données dans **MySQL payment-db**.

3-5 API Gateway, Eureka et Config Server

L'API Gateway centralise le routage des requêtes vers les services métiers selon les chemins définis.
Eureka assure l'enregistrement et la découverte automatique des services, tandis que le Config Server permet l'externalisation des paramètres sensibles et des configurations communes.

4- Choix de l'architecture

Le choix d'une architecture **microservices avec bases de données dédiées** permet une isolation forte des services, une meilleure résilience et une évolution indépendante de chaque composant.
L'utilisation de **Spring Cloud Gateway** simplifie la gestion du routage, du CORS et de la sécurité.
Le couple **Eureka / Config Server** assure la découverte dynamique et la cohérence des configurations.
Les communications REST via Gateway et RestTemplate offrent une implémentation simple et facilement traçable.
Resilience4j est appliqué uniquement aux flux critiques afin d'éviter la propagation des pannes.
Enfin, **Docker Compose** garantit un déploiement local reproductible, et **Swagger** facilite la validation fonctionnelle.

5- Contraintes techniques respectées

Le projet respecte l'ensemble des contraintes imposées :

- Une base de données indépendante par microservice.

- Communication interservices synchrone via REST.
- Utilisation complète de l'écosystème **Spring Cloud**.
- Sécurité basée sur JWT et gestion centralisée du CORS.
- Tolérance aux pannes via Resilience4j.

6- Difficultés rencontrées et solutions

Plusieurs difficultés ont été rencontrées durant le développement.
Les problèmes de **CORS** et d'erreurs 403/500 en environnement Docker ont été résolus par une centralisation stricte du CORS au niveau du Gateway.
La découverte Eureka non fonctionnelle au démarrage a nécessité un contrôle du **bootstrap des services** et des configurations de ports.
Des incohérences de routage ont été corrigées en ajustant les règles de réécriture des URLs.
Des erreurs liées aux versions de **Resilience4j** ont été résolues par un alignement précis des dépendances.
Enfin, le frontend n'étant pas finalisé, il a été temporairement exclu tout en conservant un backend entièrement testable via Swagger et cURL.

7- Tests et validation

La validation fonctionnelle a été réalisée via **Swagger UI** pour chaque microservice.
Des tests ont également été effectués via **cURL** en passant par l'API Gateway.
Les endpoints **Actuator** ont permis de vérifier la disponibilité des services.
Le bon fonctionnement global a été confirmé grâce aux commandes Docker Compose (*ps*, *logs*).

8- Conclusion

L'architecture mise en place répond pleinement aux objectifs pédagogiques du projet J2EE.
Elle illustre une implémentation cohérente des microservices Spring Boot, appuyée par une infrastructure Spring Cloud complète, intégrant sécurité, résilience, monitoring et déploiement conteneurisé.
Cette base architecturale reste évolutive et permet l'ajout futur de fonctionnalités supplémentaires sans remise en cause du découplage existant.

9- Dépôt GitHub

https://github.com/mohamedrayane0104-dev/Event-Management-Platform