Applications Web Orientées Services

Rapport de projet

EZZAOUIA Fares & CYROT Victoire

Exécution du projet

- ➔ Se référer au Readme pour le lancement de l'application
- ➔ Pour peupler les bases de données, des scripts d'exemples sont fournis dans le dossier scripts_bd

Architecture & Implémentation

Choix des Bases de Données NoSQL

Modèle Document (MongoDB) : Pour stocker des documents JSON flexibles représentant les sites et les sports.
Modèle Graphe (Neo4j) : Pour gérer les relations complexes entre les sites, les sports, et les trajets entre les sites.
Modèle Relationnel (PostgreSQL) : Pour les données des utilisateurs et leurs itinéraires.
Pour en savoir plus sur les modèles choisis et leurs justifications, vous pouvez vous référer au rapport du projet NOSQL présent dans le repo Github sous le nom « Rapport Projet NOSQL »

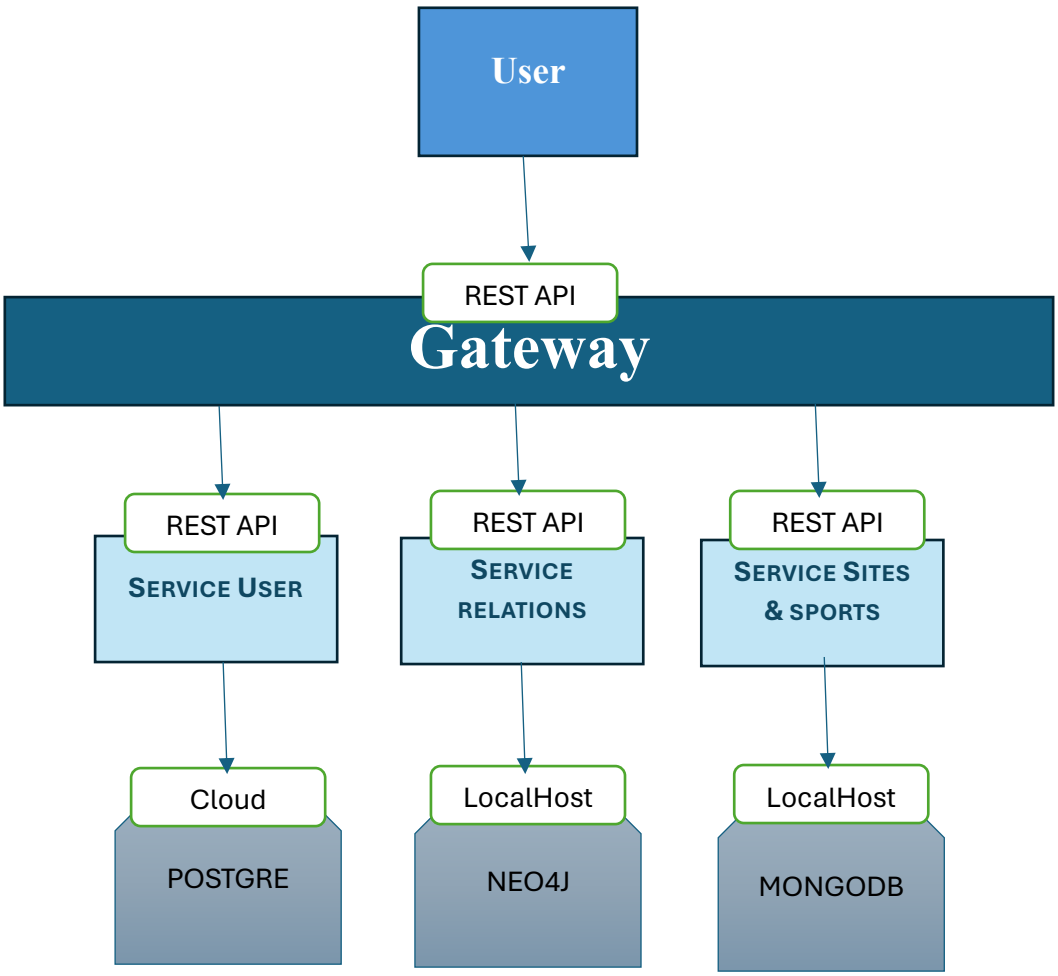
Création des Microservices

Pour implémenter ces bases de données et pour pouvoir les requêter, nous avons fait le choix de créer en Microservice sur Spring par type de base de données. Pour rendre invisible cette architecture aux yeux de l'utilisateur, nous avons aussi fait le choix de créer un Gateway pour pouvoir, à partir d'une même URL, requêter tous nos services.

Ainsi, nous avons créé :

- Un service pour gérer les relations entre les sites, les sports et les évènements, avec une base de données Neo4j
- Un service pour gérer les données utilisateurs et leurs itinéraires, avec une base de données PostgreSQL
- Un service pour gérer les informations détaillées des sports et sites olympiques, avec une base de données MongoDB
-

Ce qui nous donne l'architecture suivante :



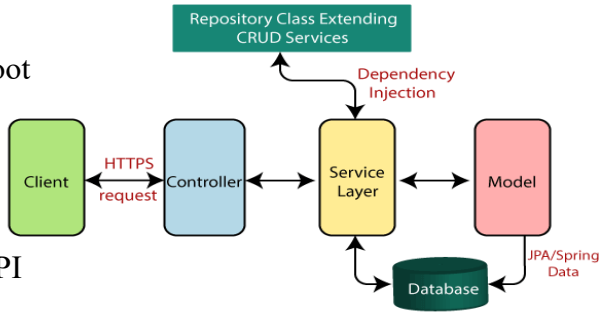
Service User

Pour la création du service User, nous avons créé une application Spring basique basé sur une base de données PostgreSQL. Le service est capable d'effectuer toutes les opérations CRUD classiques. Nous avons fait le choix de créer la base de données grâce à Docker, (tout comme pour la base de données Neo4j) pour 3 raisons simples :

1 - Portabilité et isolation : Docker permet de créer des conteneurs légers et portables qui peuvent être exécutés de manière cohérente sur n'importe quelle machine où Docker est installé, sans se soucier des dépendances logicielles ou de l'environnement, ce qui est idéal pour les machines qui n'ont pas PostgreSQL

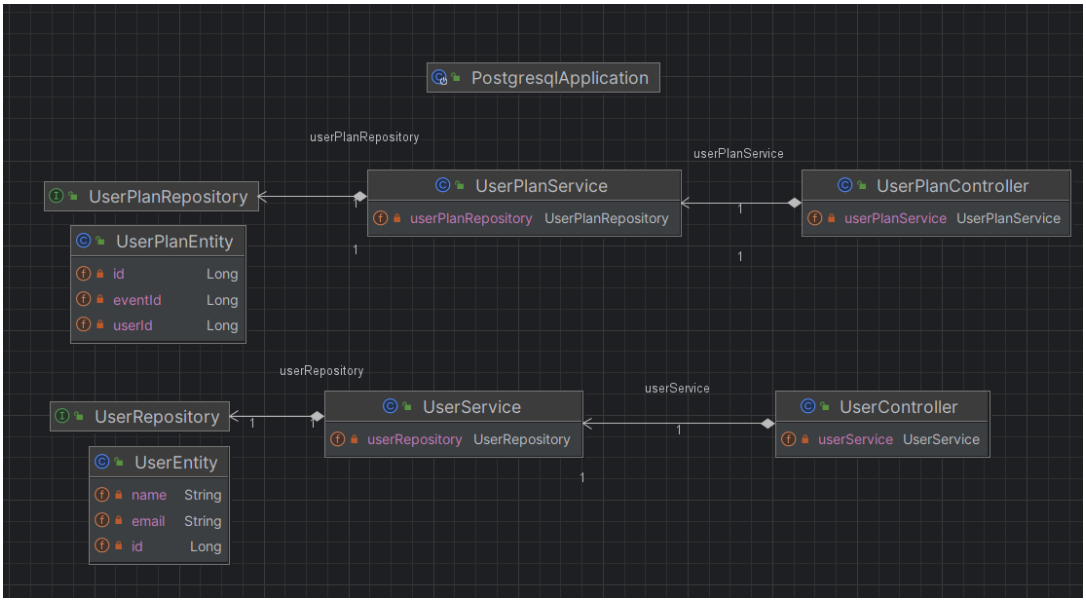
2 -Reproductibilité : Les conteneurs Docker sont définis par des fichiers Dockerfile, qui spécifient les étapes nécessaires pour créer l'image du conteneur. Cela permet de reproduire facilement l'environnement de la base de données sur différentes machines ou environnements ainsi que de la peupler plus facilement

3 – Pour apprendre à créer des applications en utilisant Docker, puisque l'applicationaujourd'hui est très utilisé et il est utile d'avoir cette compétence dans son arsenal.



Le micro-service repose sur une architecture Spring Boot classique. Pour nous faciliter la tâche, nous avons aussi utilisé les annotations Spring ainsi que Lombok ce qui nous as permis d'éviter d'écrire à chaque fois les méthodes classiques (constructeurs, getters et setters) ainsi que de créer les routes pour l'API REST.

Finalement, nous obtenons le diagramme de classe suivant :



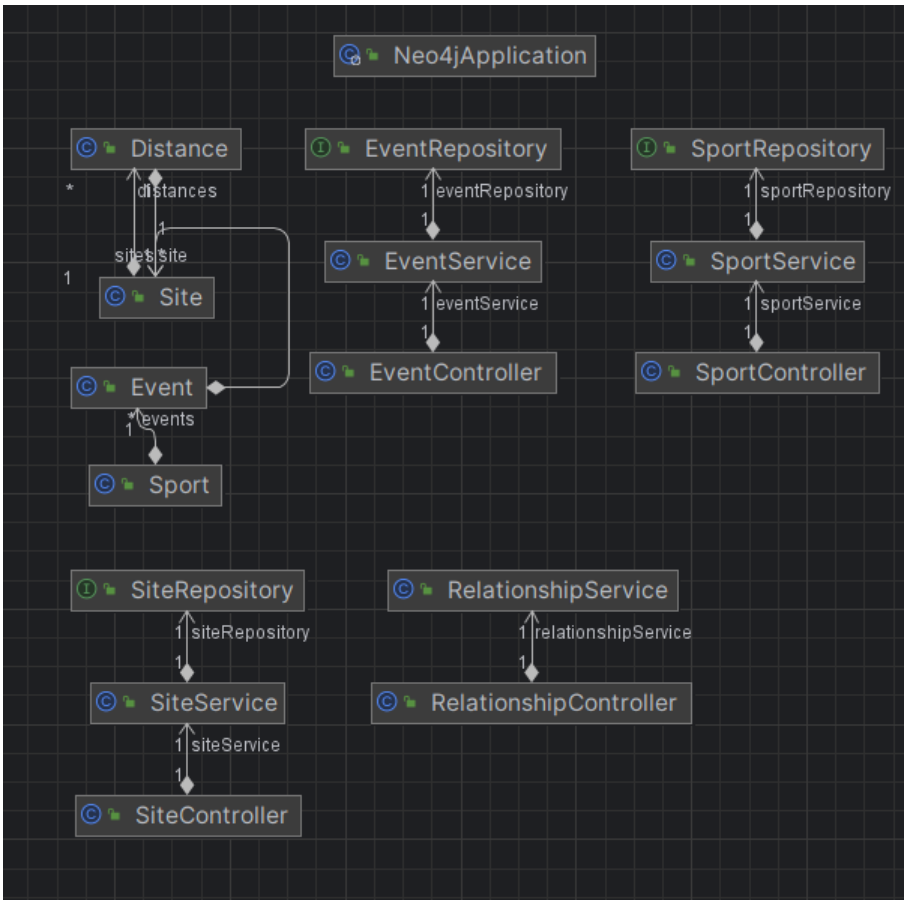
Nous n'avons pas eu de difficultés particulières à créer ce service, vu qu'il repose sur une architecture assez classique (Spring + Postgres) qui est très bien documenté en ligne.

Service Relations

Notre service Relations est basé sur une base de données Neo4j, géré grâce à Docker (comme pour la bdd PostgreSQL). L'application est capable d'effectuer toutes les opérations CRUD ainsi que d'effectuer des requêtes spécifiques en utilisant le langage Cypher. On peut donc par ailleurs, rechercher les sports avec des épreuves à une date donnée et/ou sur un site donné ou encore rechercher les sites accueillant un ou des sports donnés et/ou à une date donnée. C'est ce service qui nous a donné le plus de difficultés et qui a pris une grande partie du temps alloué au projet. En effet, ce type d'application manque de documentation en ligne et la gestion des relations entre les différents nœuds d'un graphe a était très complexe. Il fallait faire en sorte que nos requêtes POST créent toutes les relations nécessaires entre les différents types de nœuds (Sites, Sport et Event), tout en laissant la possibilité à l'utilisateur de créer cette relation plus tard. Il a donc fallu créer plusieurs requêtes POST qui prennent en compte

tous ces cas de figures. Nous avons donc passé beaucoup de temps à lire de la documentation et à s'informer sur des forums, pour pouvoir résoudre tous ces problèmes.

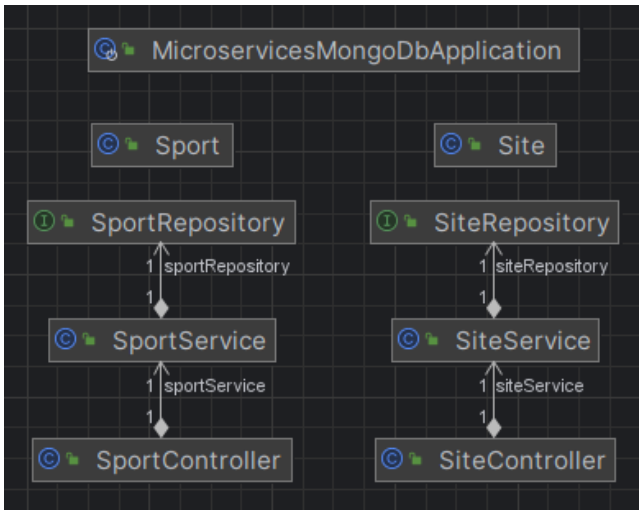
Au final, il nous reste quelques problèmes à régler concernant les relations, spécialement celles qui relient deux sites et qui contiennent les distances, que nous n'avons pas réussi à implémenter.



Service Sites & Sports

Tout comme le Service User, le service Sites & Sports a était assez facile à implémenter. En effet, les applications Spring avec MongoDB sont très bien documentées en ligne. Cela dit, pour ce service, nous avons décidé d'utiliser une base de données Cloud et non pas Docker, aussi dans le but d'apprendre à gérer ce type de base données.

Vu que nous l'avions déjà fait pour le projet Java de début d'année, nous n'avons pas eu de difficultés particulières à l'implémenter.



Service Gateway

Le service gateway a été un composant crucial dans notre architecture microservices, et ce pour plusieurs raisons :

1. Centralisation des points d'entrée : Un service gateway agit comme un point d'entrée central pour toutes les requêtes clients. Au lieu d'avoir plusieurs points d'entrée pour chaque service, le service gateway permet d'unifier l'accès aux services de notre application.
2. Routage intelligent : Le service gateway peut être configuré pour diriger les requêtes vers les services appropriés en fonction de différents critères tels que le chemin de l'URL, les en-têtes HTTP, les paramètres de requête, etc. Cela permet une gestion flexible du trafic et des versions des services.
3. Réduction de la complexité côté client : Les clients n'ont qu'à connaître l'URL du service gateway, ce qui réduit la complexité du développement et de la maintenance du code côté client.

Nous avons eu quelques problèmes à gérer les dépendances utiles à la création d'un gateway, mais une fois ces soucis réglés, nous n'avons pas eu de problèmes particuliers à l'implémenter dans notre application. Nous avons aussi renommé chaque routage de nos API Rest pour être sûr qu'il n'y ait pas de doublons

Bilan

Ce que nous avons aimé :

Résolution de problèmes complexes : Rencontrer et résoudre des défis techniques, tels que la modélisation des données, l'intégration de différentes composantes et la gestion des erreurs, a été enrichissant.

Ce que nous avons appris :

1. **Spring Boot et Spring Data Neo4j** : Nous avons approfondi nos connaissances sur l'utilisation de Spring Boot pour le développement d'applications web et Spring Data Neo4j pour interagir avec une base de données Neo4j.
2. **Gestion des entités et des relations** : Nous avons appris à modéliser des données complexes avec chaque base de données, en utilisant des entités, des relations et des annotations spécifiques à chacune (Postgres/Neo4j/MongoDB).
3. **Docker et mise en conteneur** : Nous avons acquis des compétences dans la mise en conteneur d'applications avec Docker, ce qui facilite le déploiement et la gestion des environnements de développement.

Ce que nous avons moins aimé :

1. **Débogage et résolution de problèmes** : Parfois, rencontrer des erreurs ou des problèmes techniques a été un défi, nécessitant du temps et des efforts pour les résoudre.
2. **Complexité de la modélisation des données** : La modélisation des données dans une base de données Neo4j peut être complexe, surtout lorsque des relations multiples et des requêtes complexes sont impliquées.

Globalement, le projet a été une expérience enrichissante qui nous a permis d'approfondir nos compétences en développement logiciel et d'explorer de nouvelles technologies comme les solutions NoSQL et Docker.