RAPPORT DU PROJET

ARCHITECTURE WEB ORIENTEES SERVICES

MIAGE IF 2023 – 2024

TRAN Ha Anh – HADERER Julien

Lien Github : https://github.com/haanh0811/microservices_java

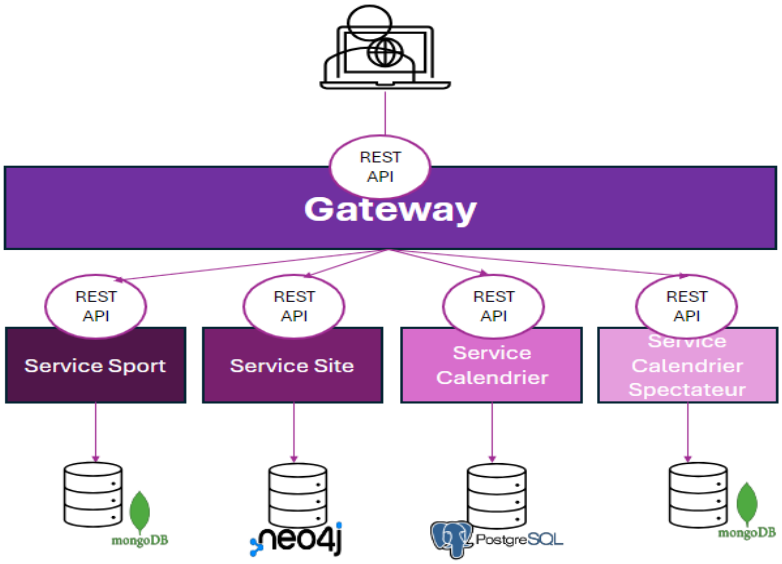
Instructions pour exécuter le projet

Pour exécuter le projet, les instructions sont disponibles dans le Readme du repository Github.

Documentation technique

1. Schéma d'architecture

1.1 Architecture générale



Dans ce projet, nous avons implémenté 5 microservices :

- Site
- Sport
- Calendrier
- Calendrier d'un spectateur
- Gateway

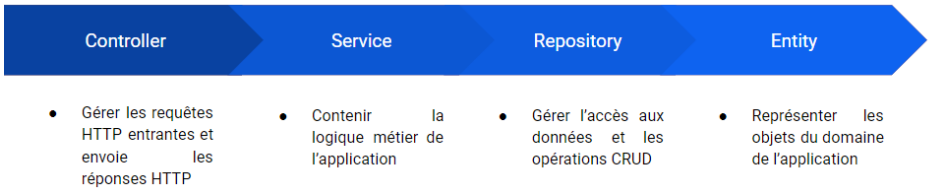
Nous avons décidé de séparer le code en 4 microservices afin que chacun fonctionne de façon indépendante des autres. C'est à dire que chacun possède une base de données lui étant propre. Nous avons choisi de compartimenter de la sorte pour les raisons suivantes.

Tout d'abord, nous avons jugé que les bases de données Site et Sport ne nécessitait pas d'être rassemblées dans un unique microservice. En effet, gérer les sites ne nécessite nullement de gérer des sports et inversement.

Nous avons ensuite décidé de créer un nouveau service pour la gestion du calendrier des épreuves. Cela car cette gestion nécessite une interaction avec les deux services précédemment créés.

Enfin, nous avons créé un ultime service dédié à la gestion du calendrier d'un spectateur. En effet, ce service contient une partie relative à la gestion d'utilisateur. Cela nous apparaissant comme trop loin de la responsabilité du service Calendrier, nous avons opté pour une séparation de ces deux services.

1.2 Architecture d'un microservice



Chacun des microservices Sport, Site, Calendrier et Calendrier Spectateur partagent une architecture commune visible dans le schéma ci-dessus. Les requêtes sont implémentées dans un Controller. Ce controller appelle un Service ayant pour mission de gérer la logique métier. Ce dernier fait appel au repository qui gère les interactions avec la base de données. Les objets stockés dans la base ont leur structure définie dans les classes du package entity.

2. Choix technique

2.1 Back end

a. Gestion de stockage

Nous avons choisi les modèles de données ci-dessous pour chaque microservice :

- Sport et CalendrierSpec sont de modèles Document dont la base de données est stockée en cloud avec mongoDB Cloud Atlas
- Le service Site contient une base de données Neo4j stocké dans le cloud via Neo4j Aura.
- Enfin, pour le service Calendrier, nous nous sommes basés sur la stack technique présentée en cours. A savoir, une base de données PostgresQL stocké via H2 Database.

b. Outil d'implémentation

Nous travaillons avec IntelliJ pour l'implémentation du code et Github pour le partage du code.

Dans un premier temps, nous testé les méthodes via le logiciel Postman, très utile pour envoyer des requêtes Post, Put et Get, avec des corps Json.

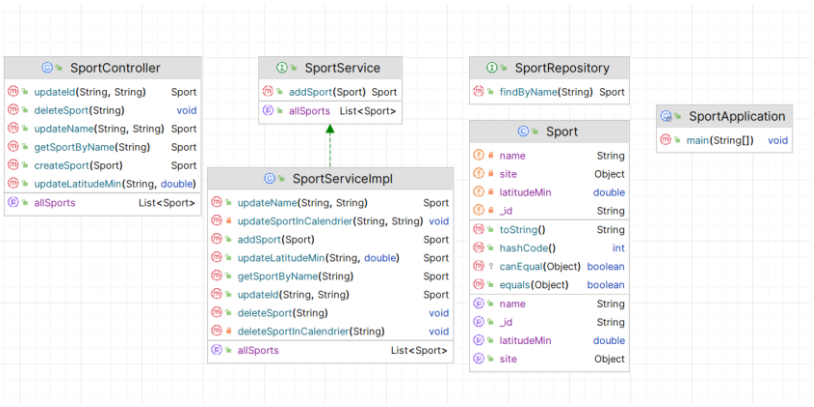
c. Front end

Afin de tester nos différentes API, deux solutions s'offrent à vous Tout d'abord, les endpoints sont accessibles via Postman.

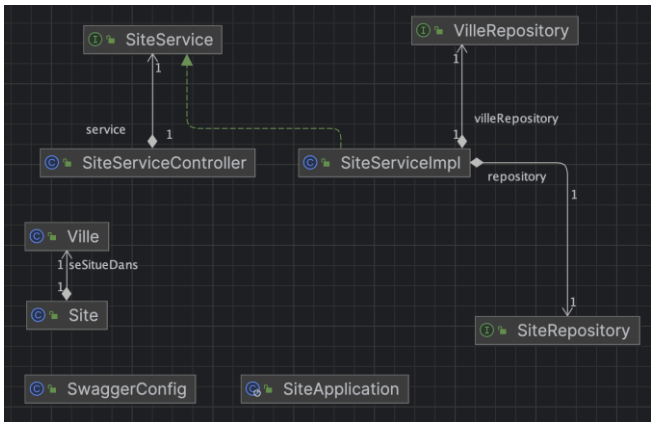
Afin de faciliter le travail de toute personne souhaitant tester nos requêtes, nous avons mis en place un swagger par projet. Ainsi, une description des inputs attendu est proposé à l'utilisateur.

3. Diagrammes

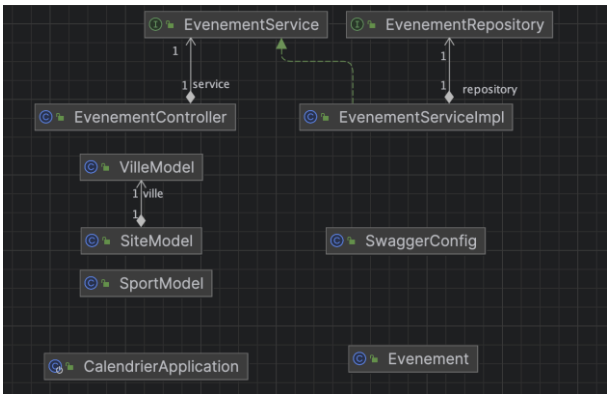
3.1 Sport



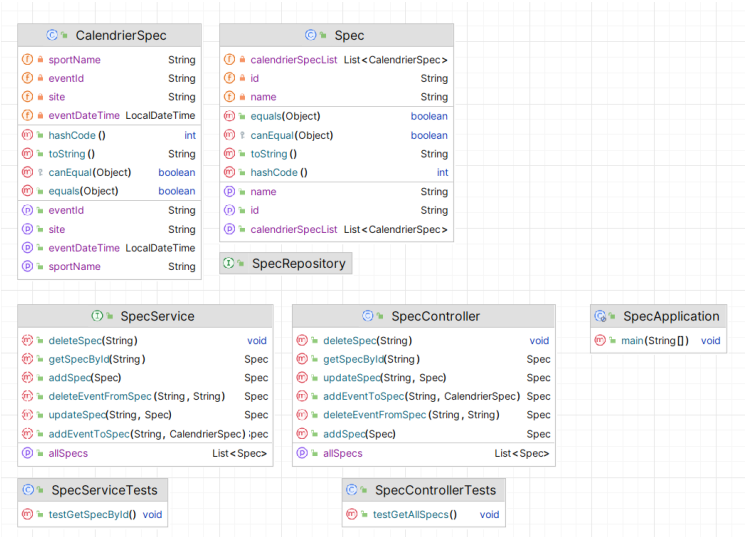
3.2 Site



3.3 Calendrier



3.4 Calendrier d'un spectateur



4. En plus

Pour aller un peu plus loin que la simple implémentation des services demandés, nous avons tenté de mettre en pratique certains points abordés durant le cours :

- Pour la conteneurisation nous avons fait avec Docker, ensuite nous avons utilisé MiniKube pour le déploiement le microservice CalendrierSpec

- Gestion des logs : Comme vu en cours, nous nous sommes dit qu'il serait intéressant de conserver des données relatives aux requêtes appelés. De plus, nous souhaitions être capable de garder la trace de certaines erreurs. Bref, nous souhaitions consigner toutes les informations illustrant le fonctionnement de notre application. Pour faire cela, nous avons implémenté sur 3 services la gestion des logs (services Site, Calendrier et Gateway). En pratique, les logs nous ont par moment aidé à détecter certaines erreurs. En effet, même lorsque l'on ne spécifie pas de logs, un service va inclure dans le fichier de logs bien plus d'informations que ce qui serait écrit sur la console. On gagne ainsi pas mal de détails sur ce qui se passe. Parfois même beaucoup trop. Un point négatif à l'utilisation des logs est que leur adoption a fait disparaître l'affichage standard sur la console. Ce qui n'est pas très plaisant lorsque l'on teste en même temps que l'on code. Les logs des 3 services ont été centralisés dans un même fichier. On sait que chaque service est disposé de son propre fichier de logs, qui vont être ensuite centralisés via des applications comme Kibana, Logstash ou Elasticsearch. Cependant, ne disposant pas des outils, nous les avons regroupés dans le même fichier.

- Gateway : Là encore, nous nous sommes appuyés sur une notion vue en cours. Afin de rendre l'architecture microservices invisible pour l'utilisateur, nous avons mis au point une gateway. Ainsi, à partir d'une même base d'url, l'ensemble des requêtes de chaque service est appelable. Pour faire cela, nous avons créé un cinquième service (Gateway). Cela fonctionne bien à un détail près. En effet, nous avons tenté de créer un unique swagger depuis lequel on pourrait utiliser les swaggers de chaque service. Dans les faits, le swagger affiche bien les autres swaggers. Cependant, les appels, bien que traité par chaque microservice, ne donnent aucun résulats visibles depuis ce swagger unique.

Bilan du projet

1.Difficultés rencontrées

Tout d'abord, la prise en main de la manipulation des données stockées a été un défi. Nous avons passé beaucoup de temps à lire des documents et à regarder des tutoriels en ligne pour comprendre comment

interagir avec les applications et leurs bases de données. Initialement, nous avions envisagé d'installer les bases de données en local, mais nous avons rapidement réalisé que cela était inefficace et coûteux en termes de temps et d'espace. Par conséquent, nous avons opté pour des bases de données dans le cloud, utilisant des services gratuits.

L'autre point bloquant concerne la communication entre les services. Nous avons dû réfléchir aux interactions ayant un vrai intérêt. La plupart des données (au moins les sites, sports et calendrier des épreuves) sont statiques. Nous avons donc décidé de ne pas les copier. Simplement, pour créer un évènement dans le calendrier, un contrôle est mis en place pour confirmer que le sport et le site existe bien. De la même façon, pour qu'un spectateur puisse s'inscrire pour suivre un évènement, on s'assure que l'épreuve existe bien (via le nom du sport et du site, la date et l'heure). Enfin, nous avons fait en sorte qu'un changement, bien qu'improbable, sur le nom d'un sport ou d'un site soit répercuté sur la base calendrier où l'on garde en mémoire simplement le nom du sport et du site. Cela afin de conserver une cohérence.

2. Ce que nous avons appris

Comme indiqué précédemment, nous avons réussi à appliquer les notions apprises en cours à travers ce projet. Tout d'abord, nous avons mis en pratique une architecture de microservices pour notre application, ce qui nous a permis de comprendre comment décomposer une application monolithique en services indépendants et interconnectés. Ensuite, nous avons implémenté des APIs REST, apprenant ainsi à établir une interaction client-serveur efficace et sécurisée en utilisant des endpoints bien définis. Enfin, nous avons développé notre projet en utilisant Spring Boot, ce qui nous a aidés à maîtriser ce framework puissant pour créer des applications Java robustes et évolutives.

De plus, nous avons dû nous informer sur la façon de mettre en pratique les autres points du cours que l'on souhaitait implémenter. En particulier, comment traiter la partie Docker ? Comment créer une Gateway ? Comment créer un fichier de logs ?

Enfin, nous avons passé du temps afin de comprendre comment interagir avec des bases de données MongoDb et Neo4j dans le cloud. Cela a nécessité de lire de la documentation mais également de rechercher des solutions pour stocker des données. Certains problèmes de configuration ont dû être résolus.