Sommaire

I- Compiler le projet

1. Configuration de l'environnement de développement :

La configuration de l'environnement a inclus l'installation de PostgreSQL et MongoDB en utilisant «brew services start postgresql@16» et «brew services start mongodb-community@7.0». Ces bases de données ont été choisies pour leurs fonctionnalités de stockage adaptées au projet.

2. Déploiement et lancement du projet :

Le projet a été déployé localement en exécutant l'application Spring Boot sur un serveur local. Le bon fonctionnement a été vérifié via un navigateur web.

3. Tests des API avec Postman :

Les fonctionnalités de l'API ont été testées avec Postman en envoyant des requêtes HTTP aux endpoints. Les principales fonctionnalités testées comprenaient la récupération des sites paralympiques, la recherche de sites par sport et l'obtention des sports par site.

4. Résultats des tests :

Les tests ont confirmé le bon fonctionnement des API, avec des résultats attendus et aucun problème majeur rencontré, démontrant la robustesse du projet.

5. Swaggers :

Le projet inclut trois fichiers Swagger pour documenter les API des trois services disponibles, facilitant leur utilisation et compréhension.

6. Utilisation de Docker :

Bien que Docker ait été mis en place, il n'est pas supporté sur les processeurs M1 d'Apple, empêchant son utilisation dans cet environnement.

II- Documentation technique

A- Schéma d'architecture

L'architecture de vos services `distances_services`, `sites_services`, et `calendar_services` adopte une approche de base de données polyglotte, utilisant Neo4j, PostgreSQL et MongoDB. Chaque service est spécialisé et utilise la base de données la mieux adaptée à ses besoins :

1. `distances_services` utilise **Neo4j** pour gérer des relations complexes entre les entités `Event`, `Site`, et `Distance`, tirant parti des capacités de graphes pour des requêtes rapides et efficaces.

2. `sites_services` utilise **PostgreSQL**, une base de données relationnelle, pour stocker et gérer des données structurées que sont les `Site` et `Sport`.

3. `calendar_services` utilise **MongoDB**, une base de données NoSQL, pour gérer des données semi-structurées comme le calendrier olympique, offrant une flexibilité dans la gestion des documents `Event` et permettant des ajustements faciles des schémas.

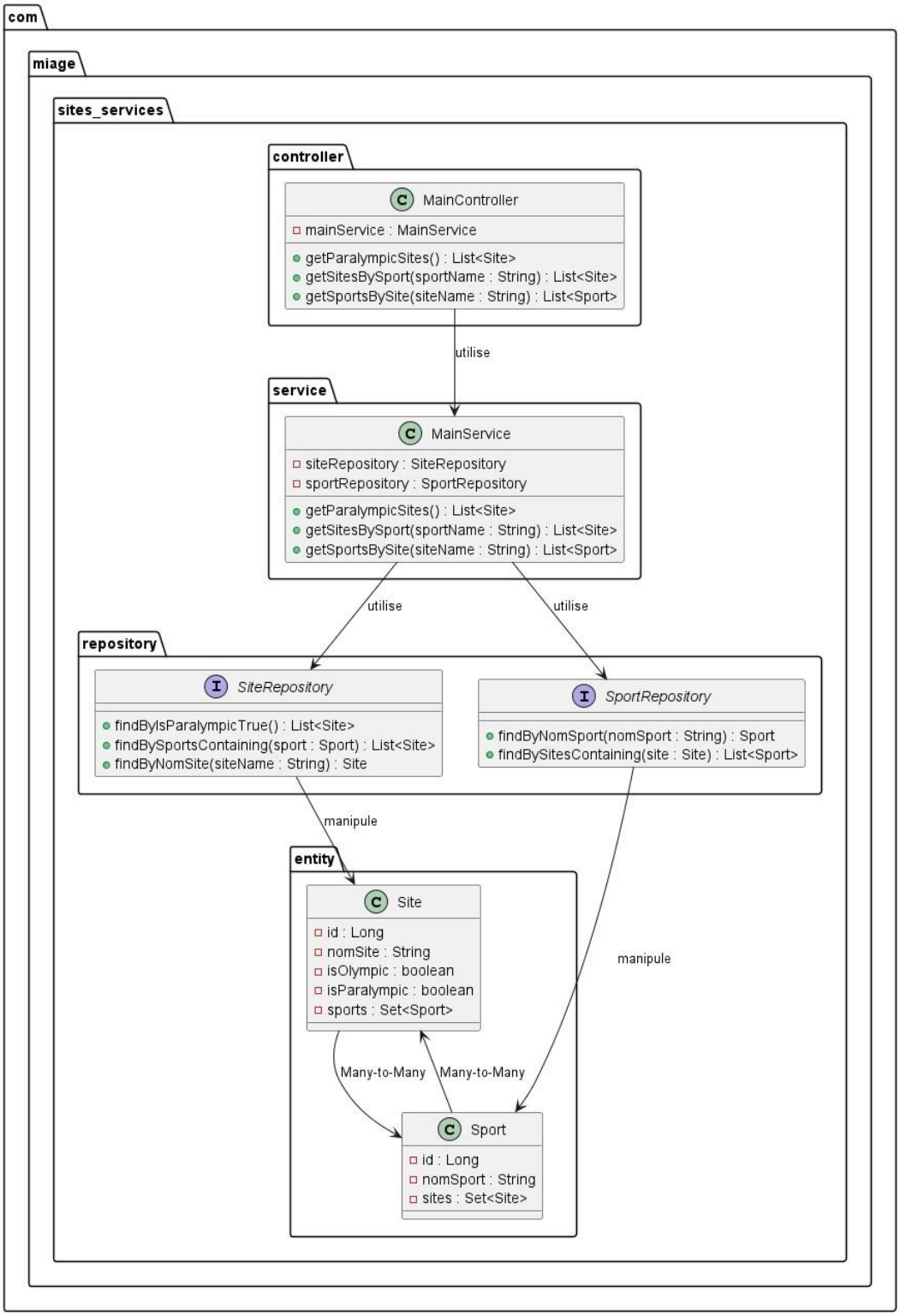
Chaque microservice utilise Spring Boot pour la configuration et l'infrastructure, et Spring Data pour interagir avec ses bases de données respectives, assurant une gestion cohérente des transactions et des opérations CRUD. Cette architecture polyglotte permet d'optimiser les performances et la scalabilité en utilisant la technologie de base de données la plus appropriée pour chaque type de données et cas d'utilisation. Les services sont autonomes et communiquent potentiellement via des API REST, assurant une flexibilité et une extensibilité de l'application globale.

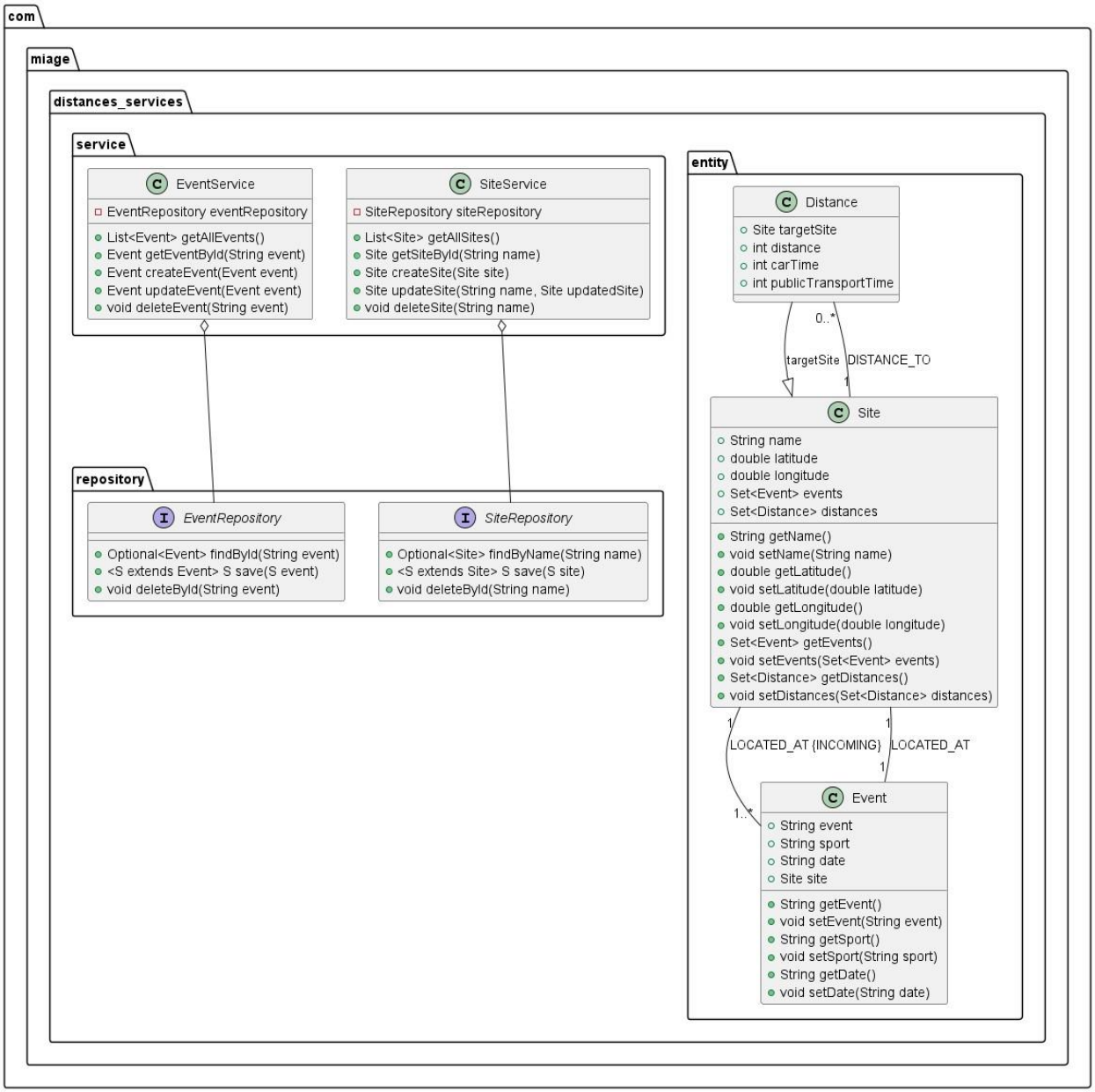
B- Choix techniques

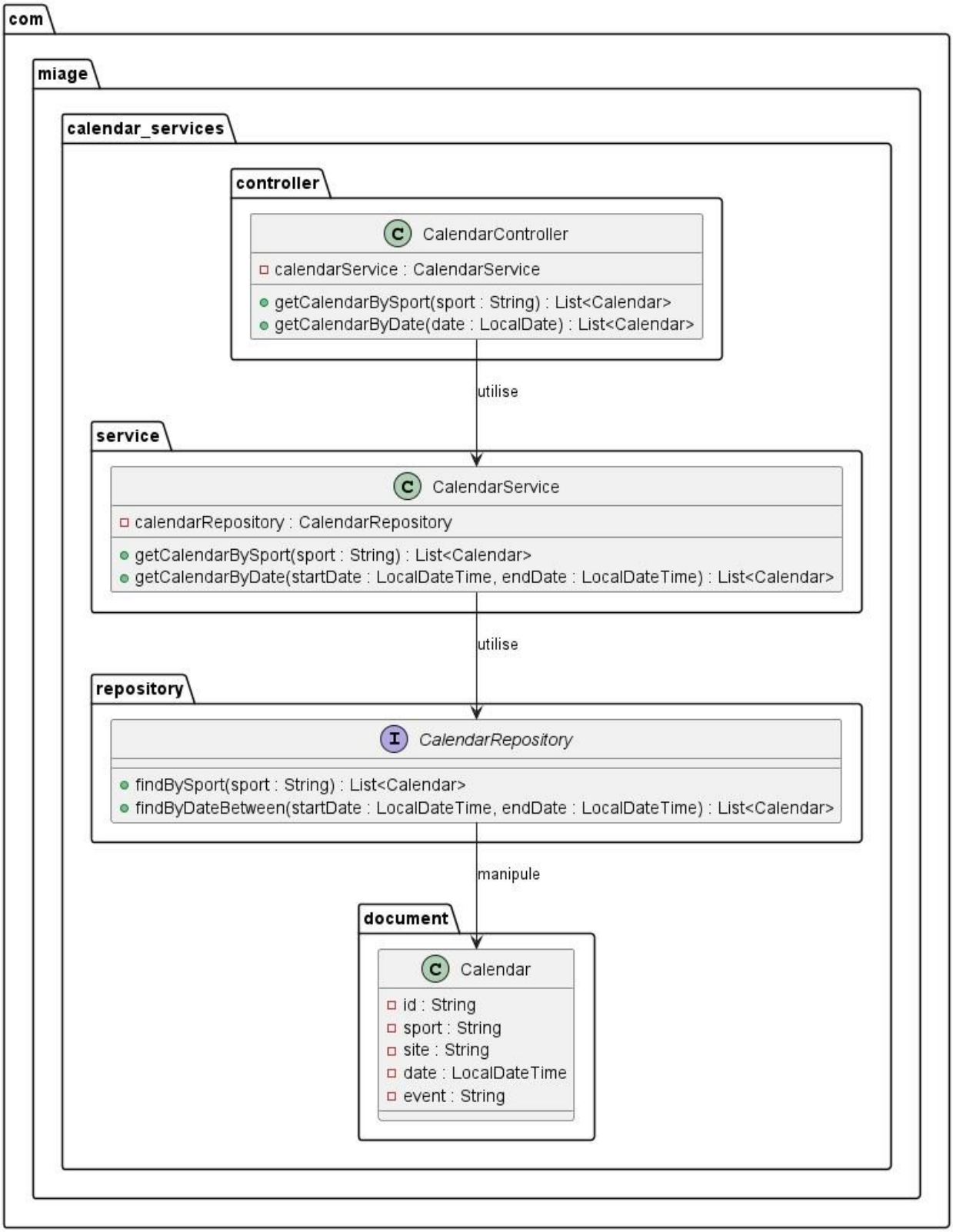
L'utilisation de Spring Boot, de Spring Data, des API REST et de l'architecture microservices a été évoquée plus haut. D'autres choix techniques ont également été mis en œuvre :

- Postman : Utilisé pour tester et documenter les API, assurant que le front-end puisse interagir efficacement avec les services backend.
- Docker : Pour la conteneurisation des services, facilitant le déploiement, la scalabilité et la gestion des dépendances.
- Minikube : Pour déployer et gérer des clusters Kubernetes locaux, offrant une vue d'ensemble sur le déploiement en production.

C- Diagrammes de classes


com
miage
sites_services
controller
MainController
mainService : MainService
getParalympicSites() : List<Site>
getSitesBySport(sportName : String) : List<Site>
getSportsBySite(siteName : String) : List<Sport>
utilise
service
MainService
siteRepository : SiteRepository
sportRepository : SportRepository
getParalympicSites() : List<Site>
getSitesBySport(sportName : String) : List<Site>
getSportsBySite(siteName : String) : List<Sport>
utilise
utilise
repository
SiteRepository
findByIsParalympicTrue() : List<Site>
findBySportsContaining(sport : Sport) : List<Site>
findByNomSite(siteName : String) : Site
SportRepository
findByNomSport(nomSport : String) : Sport
findBySitesContaining(site : Site) : List<Sport>
manipule
manipule
entity
Site
id : Long
nomSite : String
isOlympic : boolean
isParalympic : boolean
sports : Set<Sport>
Many-to-Many
Many-to-Many
Sport
id : Long
nomSport : String
sites : Set<Site>


com
miage
distances_services
service
EventService
EventRepository eventRepository
List<Event> getAllEvents()
Event getEventById(String event)
Event createEvent(Event event)
Event updateEvent(Event event)
void deleteEvent(String event)
SiteService
SiteRepository siteRepository
List<Site> getAllSites()
Site getSiteById(String name)
Site createSite(Site site)
Site updateSite(String name, Site updatedSite)
void deleteSite(String name)
repository
EventRepository
Optional<Event> findById(String event)
<S extends Event> S save(S event)
void deleteById(String event)
SiteRepository
Optional<Site> findByName(String name)
<S extends Site> S save(S site)
void deleteById(String name)
entity
Distance
Site targetSite
int distance
int carTime
int publicTransportTime
0..*
targetSite
DISTANCE_TO
1
Site
String name
double latitude
double longitude
Set<Event> events
Set<Distance> distances
String getName()
void setName(String name)
double getLatitude()
void setLatitude(double latitude)
double getLongitude()
void setLongitude(double longitude)
Set<Event> getEvents()
void setEvents(Set<Event> events)
Set<Distance> getDistances()
void setDistances(Set<Distance> distances)
1
LOCATED_AT {INCOMING}
1
1
LOCATED_AT
1
1..*
Event
String event
String sport
String date
Site site
String getEvent()
void setEvent(String event)
String getSport()
void setSport(String sport)
String getDate()
void setDate(String date)


com
miage
calendar_services
controller
CalendarController
calendarService : CalendarService
getCalendarBySport(sport : String) : List<Calendar>
getCalendarByDate(date : LocalDate) : List<Calendar>
utilise
service
CalendarService
calendarRepository : CalendarRepository
getCalendarBySport(sport : String) : List<Calendar>
getCalendarByDate(startDate : LocalDateTime, endDate : LocalDateTime) : List<Calendar>
utilise
repository
CalendarRepository
findBySport(sport : String) : List<Calendar>
findByDateBetween(startDate : LocalDateTime, endDate : LocalDateTime) : List<Calendar>
manipule
document
Calendar
id : String
sport : String
site : String
date : LocalDateTime
event : String

III- Bilan du projet

Nous avons beaucoup collaboré et apprécié le projet, surtout la modélisation des bases de données en Java via Spring Boot pour MongoDB et PostgreSQL. Travailler avec ces technologies nous a permis de renforcer nos compétences en gestion de données semi-structurées et structurées.

Cependant, malgré l'intérêt porté à Neo4j, nous avons rencontré une difficulté majeure liée à une erreur de configuration des beans dans Spring Boot, sur laquelle nous avons passé de nombreuses heures sans parvenir à la résoudre complètement.

La conteneurisation avec Docker a été bénéfique, simplifiant la gestion des environnements de développement et de production. Le déploiement avec Minikube nous a offert une meilleure compréhension des environnements conteneurisés.