

# Spring Boot GraphQL



initialisation d'un projet spring Boot

<https://start.spring.io/>

pour aller plus loin aller voir je vous conseil architecte

<https://www.jhipster.tech/>

autre avantage exemple sur le gitHub de la lib officiel.

<https://www.graphql-java.com/tutorials/getting-started-with-spring-boot/>

<https://github.com/graphql-java/graphql-java>

- **Mise en place du service graphQL**

Mise en place facilement avec la dépendance

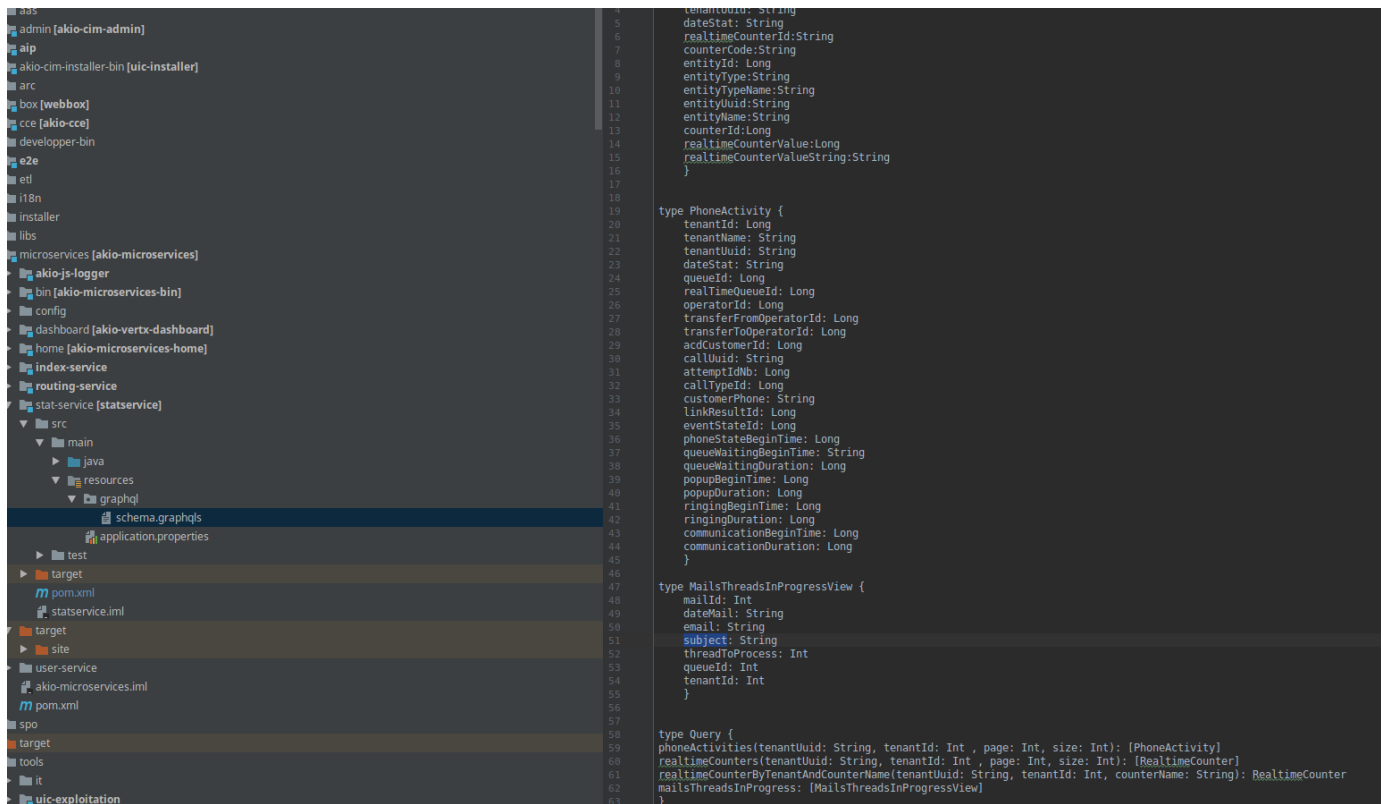
"graphql-spring-boot-starter"

```
<!-- GraphQL -->
<dependency>
  <groupId>com.graphql-java</groupId>
  <artifactId>graphql-spring-boot-starter</artifactId>
  <version>5.0.2</version>
</dependency>
<dependency>
  <groupId>com.graphql-java</groupId>
  <artifactId>graphql-java-tools</artifactId>
  <version>5.2.4</version>
</dependency>
```

Par défaut, cela exposera le service GraphQL sur le noeud final */graphql* de notre application

- **Écrire le schéma**

Ces fichiers doivent être enregistrés avec l'extension « *.graphqls* » et peuvent être présents n'importe où sur le classpath. Nous pouvons également avoir autant de fichiers que vous le souhaitez, de manière à pouvoir scinder le schéma en modules à votre guise.



## • Résolveur de requête racine

Les seules exigences sont que les beans implémentent *GraphQLQueryResolver* et que chaque champ de la requête racine du schéma comporte une méthode dans l'une de ces classes portant le même nom.

```

@Component
public class RealtimeCounterQuery implements GraphQLQueryResolver {

    @Autowired
    RealtimeCountersService realtimeCountersService;

    public List<RealtimeCounter> getRealtimeCounters(
        String tenantUuid,
        int tenantId,
        int page,
        int size) {

        return realtimeCountersService.getRealtimeCounterByIndex(
            getElasticSearchPath(REALTIME_COUNTER_PATH, tenantUuid,
tenantId),
            page,
            size);
    }

    public RealtimeCounter realtimeCounterByTenantAndCounterName(
        String tenantUuid,
        int tenantId,
        String counterName) {

        return realtimeCountersService.
realtimeCounterByTenantAndCounterName(
            getElasticSearchPath(REALTIME_COUNTER_PATH, tenantUuid,
tenantId),
            counterName);
    }
}

```

dans le fichier graphqls nous devons avoir bien défini les types et la query:

```

type RealtimeCounter {
    tenantId: Long
    tenantName: String
    tenantUuid: String
    dateStat: String
    realtimeCounterId:String
    counterCode:String
    entityId: Long
    entityType:String
    entityTypeNames:String
    entityUuid:String
    entityName:String
    counterId:Long
    realtimeCounterValue:Long
    realtimeCounterValueString:String
}

type Query {
    realtimeCounters(tenantUuid: String, tenantId: Int , page: Int, size:
    Int): [RealtimeCounter]
    realtimeCounterByTenantAndCounterName(tenantUuid: String, tenantId: Int,
    counterName: String): RealtimeCounter
}

```

## • Mutations

GraphQL a également la possibilité de mettre à jour les données stockées sur le serveur, au moyen de mutations.

Les mutations sont définies dans le code Java à l'aide de classes implémentant *GraphQLMutationResolver* au lieu de *GraphQLQueryResolver*

```

@Component
public class RealtimeCounterMutation implements GraphQLMutationResolver {

    .....

}

```

```

# The Root Mutation for the application
type Mutation {
    ...
}

```

## • Mise en place du service Elasticsearch

<https://github.com/VanRoy/spring-data-jest>

Spring Data implementation for Elasticsearch based on Jest Rest client

Useful to use Spring Data with Elasticsearch cluster accessible only by HTTP ( for example on AWS).

```
<dependency>
  <groupId>com.github.vanroy</groupId>
  <artifactId>spring-boot-starter-data-jest</artifactId>
  <version>3.2.0.RELEASE</version>
</dependency>
```

#### Configuration

```
## ELASTICSEARCH
spring.data.jest.uri=http://127.0.0.1:9200
```

#### exemple code implémentation

```
@Autowired
private ElasticsearchOperations elasticsearchTemplate;

private static final int LIMIT = 100;

public List<RealTimeCounter> getRealTimeCounterByIndex(String indexName,
int page, int size) {
    //Query
    SearchQuery searchQuery = new NativeSearchQueryBuilder().
        withQuery(matchAllQuery()).
        withIndices(indexName).
        withPageable(PageRequest.of(page, size)).build();

    //Result
    return elasticsearchTemplate.queryForList(
        searchQuery,
        RealTimeCounter.class);
}
```

- **Mise en place du service PostgreSQL**

dépendances

```

<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
  <version>${spring-boot.version}</version>
</dependency>

```

Config :

```

## Spring DATASOURCE
spring.jpa.database=POSTGRESQL
spring.datasource.platform=postgres
spring.datasource.url=jdbc:postgresql://localhost:5433/user6
spring.datasource.username=postgres
spring.datasource.password=postgres
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.jdbc.lob.non_contextual_creation=tru

```

exemple de service appelle d'une vue :

```

public interface MailsThreadsInProgressViewRepository extends
JpaRepository<MailsThreadsInProgressView, Integer>{

    @Transactional(readOnly = true)
    List<MailsThreadsInProgressView> findAll();

}



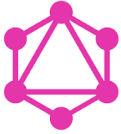
```

Conclusion

Facile à prendre en main et bonne documentation

Spring Boot



<div>ElasticSearch</div> <div></div>	<div>OK</div> <div>en plus via transport http 9200 + spring data Elastic</div>
<div>Postgresql</div> <div></div>	<div>OK</div> <div>Spring Data</div>
<div>API Graphql</div> <div></div>	<div>Ok</div> <div>graphql-spring</div>