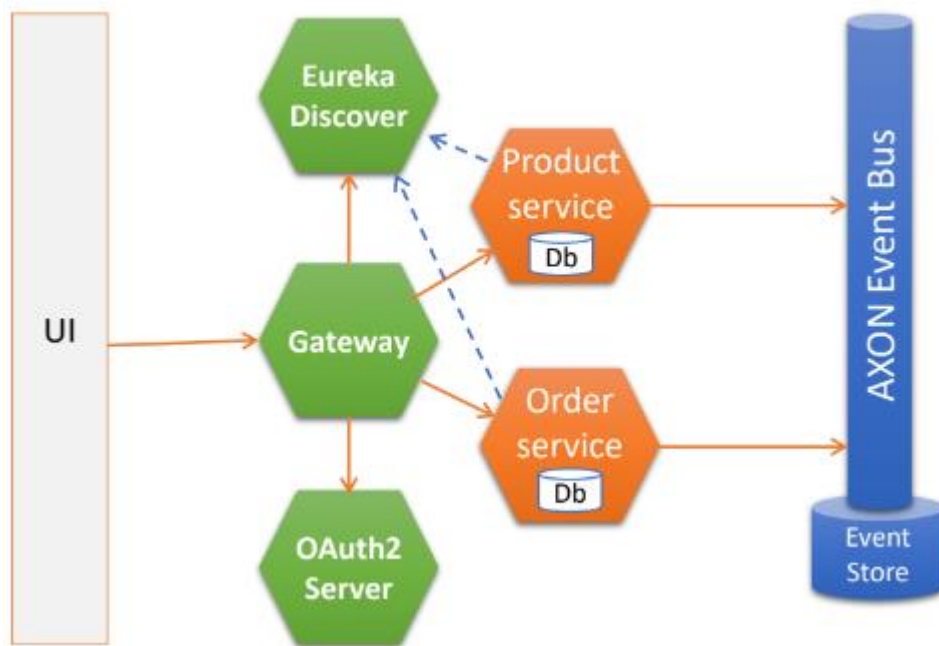


## COMPTE-RENDU : EXAMEN BLANC DE SYSTEMES DISTRIBUES

Filière : « Ingénierie Informatique : Big Data et Cloud  
Computing » II-BDCC



Réalisé par :

Khadija BENJILALI

Encadré par :

Pr. Mohamed YOUSSEFI

Année Universitaire : 2022-2023

# Sommaire

Travail à faire.....	3
Travail à faire : .....	3
<b>PARTIE 1 : Structure et Architecture du projet .....</b>	<b>4</b>
1. Architecture technique du projet .....	4
2. Diagramme de classe global du projet .....	4
3. Maven Dependencies de radar-service : .....	4
4. Common-api : .....	5
5. BaseCommand .....	5
6. BaseEvent .....	5
<b>PARTIE 2 : Création de radar-service.....</b>	<b>6</b>
1. Structure de radar-service : .....	6
2. Partie Ecriture: .....	6
3. Partie Lecture: .....	8

# Travail à faire

On souhaite créer un système distribué basé sur les micro-services en utilisant une architecture pilotée par les événements respectant les deux patterns Event Sourcing et CQRS. Cette application devrait permettre de gérer les infractions concernant des véhicules suites à des dépassement de vitesses détectés par des radars automatiques. Le système se compose de trois micro-services :

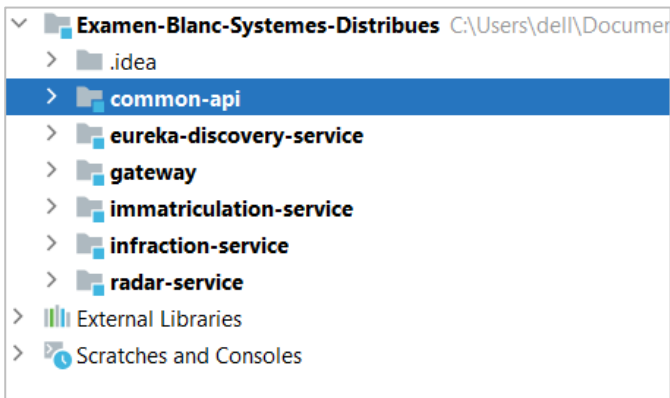
- ✚ Le micro-service qui permet de gérer les radars. Chaque radar est défini par son id, sa vitesse maximale, des coordonnées : Longitude et Latitude.
- ✚ Le micro-service d'immatriculation qui permet de gérer des véhicules appartenant des propriétaires. Chaque véhicule appartient à un seul propriétaire. Un propriétaire est défini par son id, son nom, sa date de naissance, son email et son email. Un véhicule est défini par son id, son numéro de matricule, sa marque, sa puissance fiscale et son modèle.
- ✚ Le micro-service qui permet de gérer les infractions. Chaque infraction est définie par son id, sa date, le numéro du radar qui a détecté le dépassement, le matricule du véhicule, la vitesse du véhicule, la vitesse maximale du radar et le montant de l'infraction.

## Travail à faire :

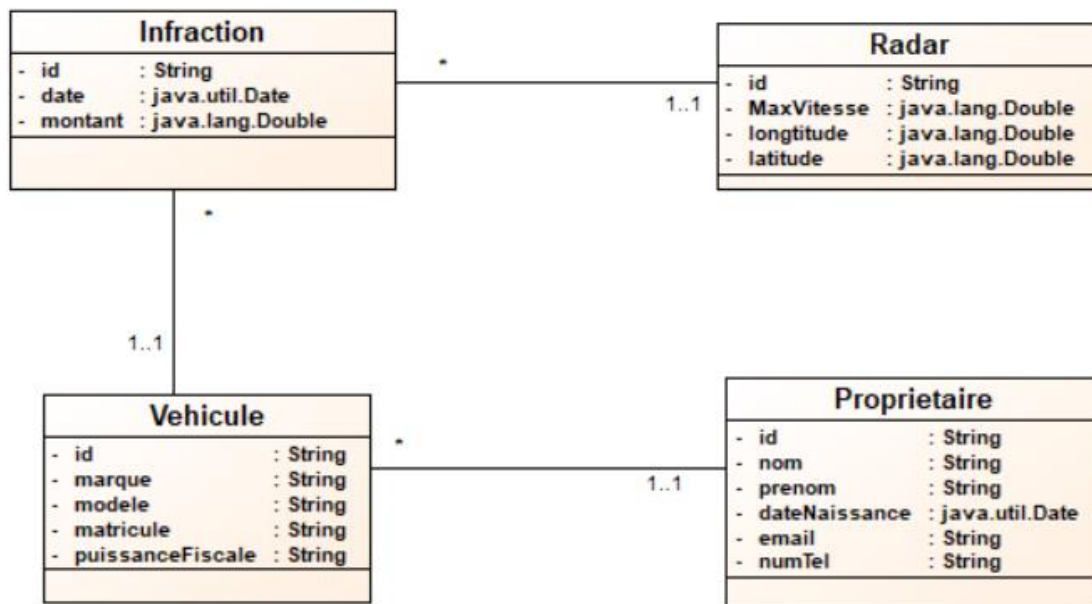
- **Etablir une architecture technique du projet**
- **Etablir un diagramme de classe global du projet**
- **Développer le micro-service Radar**
- **Développer le micro-service Immatriculation**
- **Développer le micro-service Infractions**
- **Mettre en place les services techniques de l'architecture micro-service (Gateway, Eureka**
- **Discovery service)**

# PARTIE 1 : Structure et Architecture du projet

## 1. Architecture technique du projet



## 2. Diagramme de classe global du projet



## 3. Maven Dependencies de radar-service :

Au démarrage on ne va pas utiliser le serveur AXON

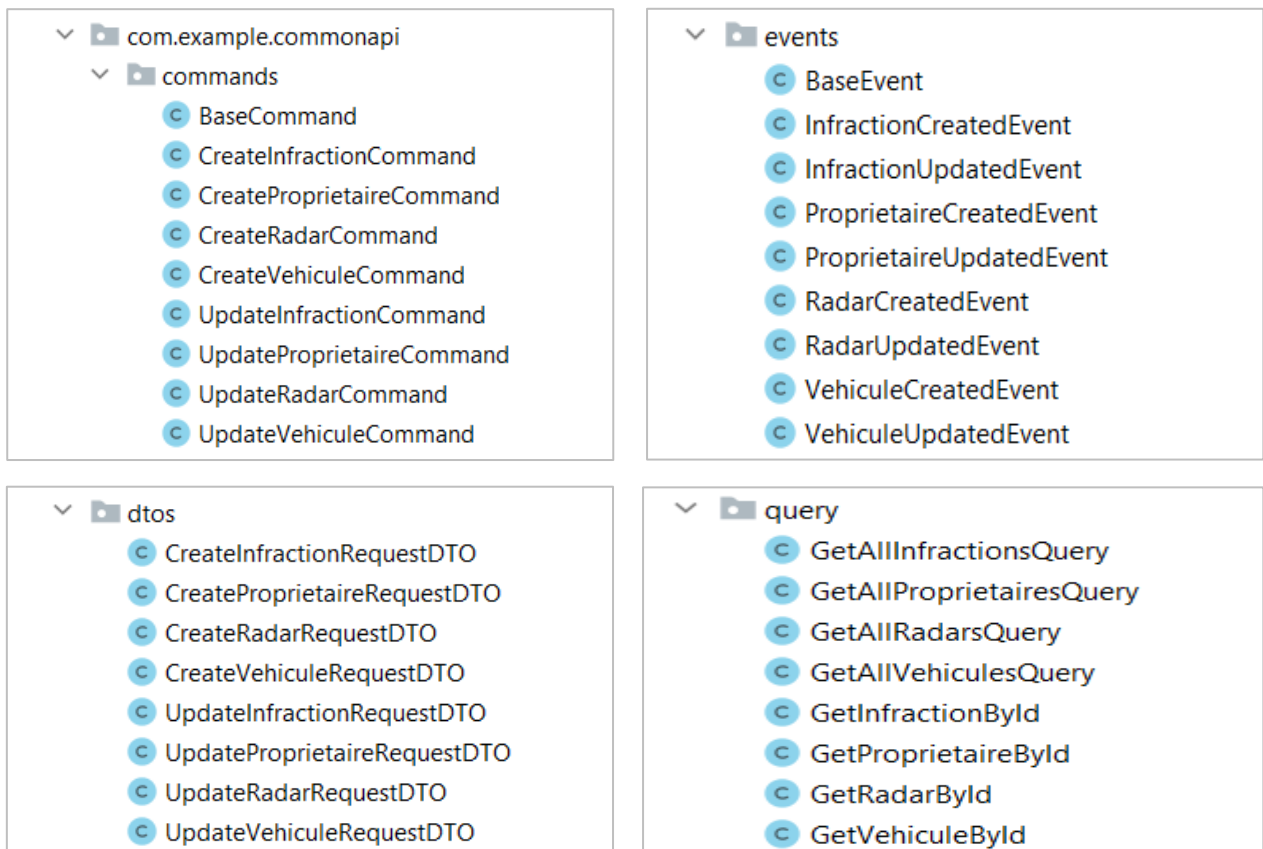
```
<dependency>
  <groupId>org.axonframework</groupId>
  <artifactId>axon-spring-boot-starter</artifactId>
  <version>4.6.2</version>
  <exclusions>
    <exclusion>
      <groupId>org.axonframework</groupId>
      <artifactId>axon-server-connector</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.12</version>
</dependency>
```

```
<dependency>
  <groupId>com.example</groupId>
  <artifactId>common-api</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.0.31</version>
  <scope>runtime</scope>
</dependency>
```

#### 4. Common-api :

En plus des modules représentant les différents micro-services, le projet utilise un module « common-api » sous forme d'un projet maven qui déclare les composants communs aux différents projets comme les Commandes, les Événements, les Queries, les DTOs, etc.



#### 5. BaseCommand

```
public class BaseCommand<T> {
    @TargetAggregateIdentifier
    private T id ;

    public BaseCommand(T id) { this.id = id; }
    public T getId() { return id; }
}
```

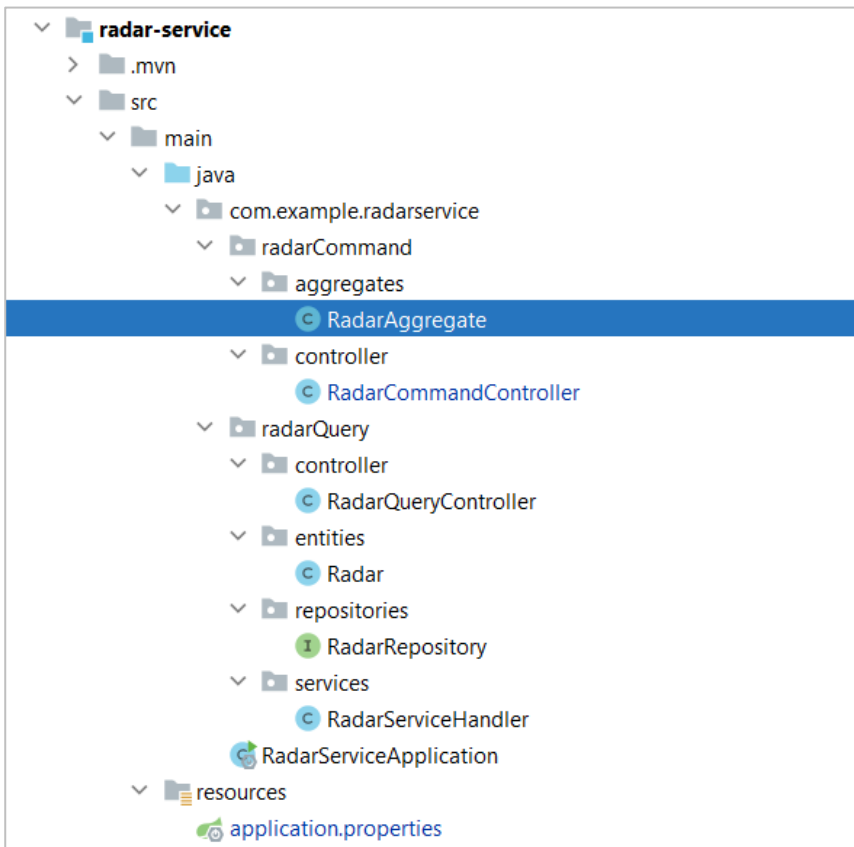
#### 6. BaseEvent

```
public class BaseEvent <T> {
    private T id ;

    public BaseEvent(T id ) { this.id =id ; }
    public T getId() { return id; }
}
```

## PARTIE 2 : Création de radar-service

### 1. Structure de radar-service :



### 2. Partie Ecriture:

- Aggregate Radar

```
13  @Aggregate
14  public class RadarAggregate {
15      @AggregateIdentifier
16      private String id ;
17      private double MaxVitesse ;
18      private double longitude ;
19      private double latitude ;
20
21      public RadarAggregate() {
22      }
23
24      @CommandHandler
25      @
26      public RadarAggregate(CreateRadarCommand command){
27          if(command.getMaxVitesse()<0){
28              throw new RuntimeException("La vitesse ne peut pas etre negative");
29          }
30          AggregateLifecycle.apply(new RadarCreatedEvent(
31              command.getId(),
32              command.getMaxVitesse(),
33              command.getLongitude(),
34              command.getLatitude()
35          ));
36      }
```

```

    @EventSourcingHandler
    public void on(RadarCreatedEvent event) {
        this.id = event.getId();
        this.MaxVitesse = event.getMaxVitesse();
        this.longitude = event.getLongitude();
        this.latitude = event.getLatitude();
    }

    @CommandHandler
    public void handle(UpdateRadarCommand command) {
        AggregateLifecycle.apply(new RadarUpdatedEvent(
            command.getId(),
            command.getMaxVitesse(),
            command.getLongitude(),
            command.getLatitude()
        ));
    }

    @EventSourcingHandler
    public void on(RadarUpdatedEvent event) {
        this.MaxVitesse = event.getMaxVitesse();
        this.longitude = event.getLongitude();
        this.latitude = event.getLatitude();
    }

```

- Ajouter un radar à partir Postman

```

@PostMapping("/createRadar")
public CompletableFuture<String> createRadar(@RequestBody CreateRadarRequestDTO request) {
    return commandGateway.send(
        new CreateRadarCommand(
            UUID.randomUUID().toString(),
            request.getMaxVitesse(),
            request.getLongitude(),
            request.getLatitude()
        ));
}

```

http://localhost:8081/radar/commands/createRadar

POST http://localhost:8081/radar/commands/createRadar

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

```

1 {
2   ... "MaxVitesse": 123,
3   ... "longitude": 39,
4   ... "latitude": 12
5 }

```

Body Cookies Headers (5) Test Results Status: 200 OK

Pretty Raw Preview Visualize Text

```

1 a9cb0683-8569-4c26-9a0c-0364a90cd46c

```

- Event Store de Radar

```

@GetMapping("/eventStore/{radarId}")
public Stream getEventsForAccount(@PathVariable(value = "radarId") String radarId)
{
    return eventStore.readEvents(radarId).asStream();
}

```

localhost:8081/radar/commands/eventStore/a9cb0683-8569-4c26-9a0c-0364a90cd46c

```

1 [
2   {
3     "type": "RadarAggregate",
4     "aggregateIdentifier": "a9cb0683-8569-4c26-9a0c-0364a90cd46c",
5     "sequenceNumber": 0,
6     "identifier": "31e58038-da2e-41ac-8ce6-a839f28e0cd1",
7     "timestamp": "2022-12-26T00:09:41.457Z",
8     "payload": {
9       "id": "a9cb0683-8569-4c26-9a0c-0364a90cd46c",
10      "longitude": 39.0,
11      "latitude": 12.0,
12      "maxVitesse": 0.0
13    },
14    "payloadType": "com.example.commonapi.events.RadarCreatedEvent",
15    "metaData": {}
16  }
17 ]
18 ]

```

### 3. Partie Lecture:

- Entity Radar

```

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Radar {
    @Id
    private String id;
    private double MaxVitesse;
    private double longitude;
    private double latitude;
}

```



- Consulter tous les radars

```
@GetMapping("/getAllRadars")
public List<Radar> getAllRadars(){
    return queryGateway.query(new GetAllRadarsQuery(),
        ResponseTypes.multipleInstancesOf(Radar.class)).join();
}
```










GET ▼ http://localhost:8081/radar/queries/getAllRadars

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Body Cookies Headers (8) Test Results 🌐 Status: 200 OK

Pretty Raw Preview Visualize JSON ▼ 🔄

```
1 [
2   {
3     "id": "a9cb0683-8569-4c26-9a0c-0364a90cd46c",
4     "longitude": 39.0,
5     "latitude": 12.0,
6     "maxVitesse": 0.0
7   }
]
```

				id	max_vitesse	latitude	longitude
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	1b647999-0ed8-4a26-8c73-69da6b403359	0	9000	123
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	743e737e-6d68-435f-973d-1be548810e5d	0	1200	899
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	a9cb0683-8569-4c26-9a0c-0364a90cd46c	0	12	39

- Consulter un radar à partir son id

```
@GetMapping("/getRadar/{id}")
public Radar getRadar(@PathVariable String id){
    return queryGateway.query(new GetRadarById(id),
        ResponseTypes.instanceOf(Radar.class)).join();
}
```

Request URL

http://localhost:8081/radar/queries/getRadar/1b647999-0ed8-4a26-8c73-69da6b403359

Server response

Code	Details
200	<p>Response body</p> <pre><code>{   "id": "1b647999-0ed8-4a26-8c73-69da6b403359",   "longitude": 123,   "latitude": 9000,   "maxVitesse": 0 }</code></pre>