

Système Distribué

Master SDIA

Réalisé par :

MAHFOUD SANAA

Table des matières

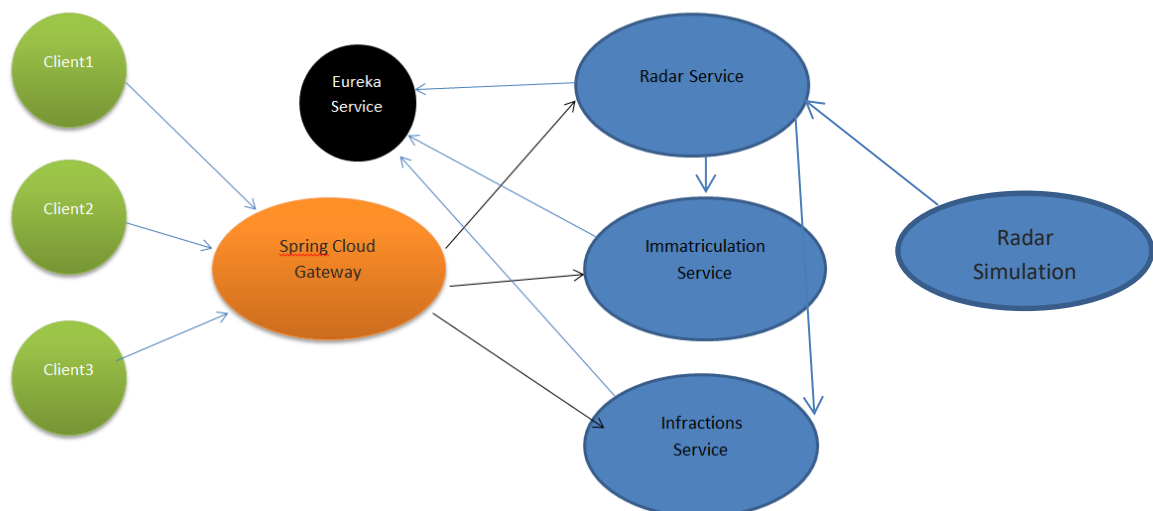
I.	Introduction.....	3
I.	Le micro Service Immatriculation :.....	4
1-	Rest & GraphQL Service :.....	6
2-	GRPC Service :.....	15
3-	Soap Service	17
II.	Le micro Service RADAR	20
1-	Entities :.....	20
2-	Service :	22
3-	Controllers :	22
4-	RadarApplication :	25
5-	Application.properties :.....	26
6-	Les tests :	26
7-	Service GRPC :	29
8-	Le test pour enregistrer une infraction avec bloomRPC :	31
III.	Micro Service Infraction :	31
1-	Entities :	31
1.	Repository :	33
2.	Service Client :	33
3.	Rest Controller :	33
5-	Application.properties :.....	34
6-	Application :	34
IV.	Radar simulation.....	35
1-	Fichier proto :	35
2-	Application :	35
3-	Les tests pour enregistrer une infraction :	36
V.	Eureka Service	37
1-	application.properties	37
VI.	Gateway service	38
1-	application.yml	38
VII.	Conclusion	39

I. Introduction

On souhaite créer un système distribué basé sur les micro-services Cette application devrait permettre de gérer et d'automatiser le processus des infractions concernant des véhicules suites à des dépassement de vitesses détectés par des radars automatiques. Le système se compose de trois micro-services :

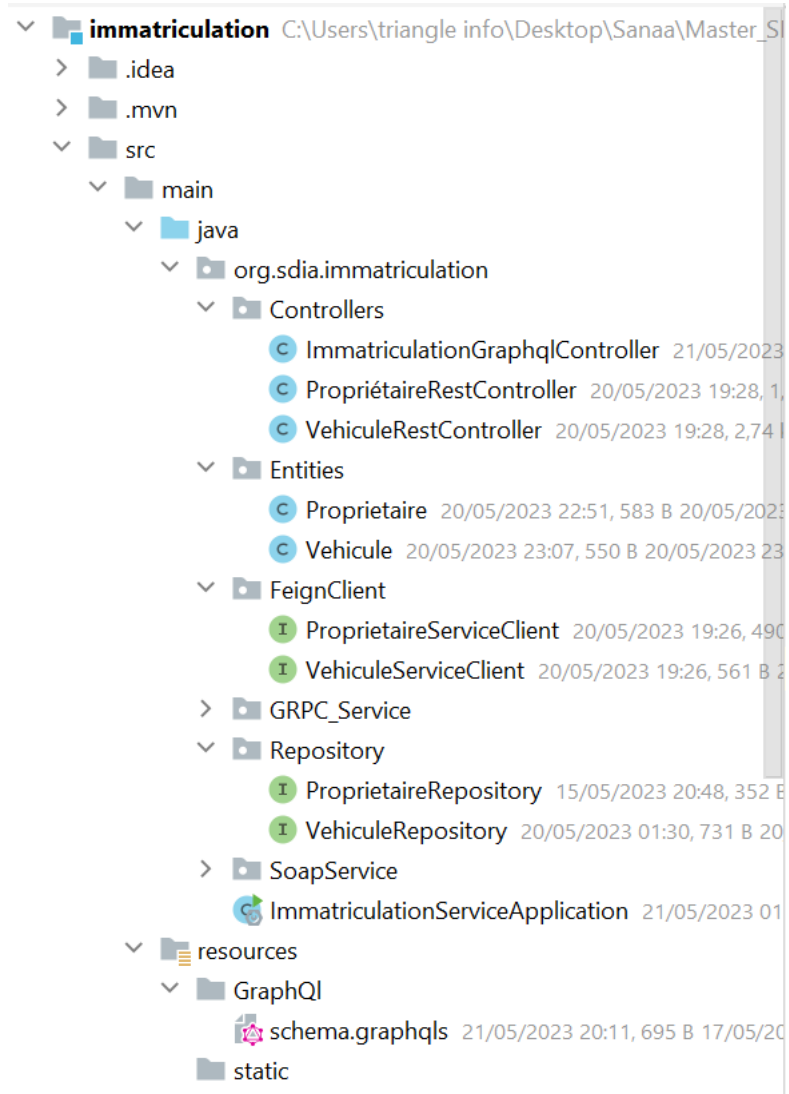
- Le micro-service qui permet de gérer les radars. Chaque radar est défini par son id, sa vitesse maximale, des coordonnées : Longitude et Latitude.
- Le micro-service d'immatriculation qui permet de gérer des véhicules appartenant des propriétaires. Chaque véhicule appartient à un seul propriétaire. 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. En plus des opérations classiques de consultation et de modifications de données, le système doit permettre de poster un dépassement de vitesse qui va se traduire par une infraction. En plus, il doit permettre à un propriétaire de consulter ses infractions.

Voilà l'architecture technique du projet :



I. Le micro Service Immatriculation :

a. Entités JPA et Interface JpaRepository basées sur Spring data :



Proprietaire Entity :

```
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Proprietaire {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idP;
    private String nom;
    private Date dateNaissance;
    private String email;
}
```

Vehicule Entity :

```
import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NoArgsConstructor;
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Vehicule {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idv;
    private String mat;
    private String Marque;
    private Integer puissanceFiscal;
    private String Modele;
    private Long idProprietaire;
    @Transient private Proprietaire proprietaire ;
}
```

1- Rest & Graphql Service :

- **ProprietaireRepository :**

```
import org.sdia.immatriculation.Entities.Proprietaire;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.rest.core.annotation.RepositoryRestResource;
@RepositoryRestResource
public
interface ProprietaireRepository extends JpaRepository<Proprietaire, Long>
{ }
```

- **VehiculeRepository :**

```
import org.sdia.immatriculation.Entities.Vehicule;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.Optional;
public interface VehiculeRepository extends JpaRepository<Vehicule, Long> {
    Optional<Vehicule> findByMat (String mat);
}
```

- **PropriétaireRestController :**

```
import java.util.List;

@AllArgsConstructor
@RestController
public class PropriétaireRestController {
    private ProprietaireRepository proprietaireRepository;
    //private ProprietaireServiceClient proprietaireServiceClient;
    //private VehiculeServiceClient vehiculeServiceClient;

    @GetMapping("/proprietaires/full/{id}")
    public Proprietaire getProprietaire(@PathVariable(name = "id") Long id)
    {
        Proprietaire proprietaire =
        proprietaireRepository.findById(id).get();
        return proprietaire;
    }

    @GetMapping("/proprietaires")
    public List<Proprietaire> getAllProprietaire() {
        return proprietaireRepository.findAll();
    }

    @PostMapping("/proprietaire")
    public Proprietaire save(@RequestBody Proprietaire request ) {
        return proprietaireRepository.save(request);
    }

    @PutMapping("/proprietaire/{id}")
    public Proprietaire update(@PathVariable Long id, @RequestBody
    Proprietaire proprietaire ) { }
```

```

        Proprietaire
        proprietaire1=proprietaireRepository.findById(id).orElseThrow();
        if
        (proprietaire.getNom()!=null)proprietaire1.setNom(proprietaire.getNom());
        if
        (proprietaire.getDateNaissance()!=null)proprietaire1.setDateNaissance(proprietaire.getDateNaissance());
        return proprietaireRepository.save(proprietaire1);
    }
    @DeleteMapping("/proprietaire/{id}")
    public void deleteProprietaire(@PathVariable Long id){
        proprietaireRepository.deleteById(id);
    }
}

```

• VehiculeRestController :

```

import java.util.List;
import java.util.Optional;
@RestController @AllArgsConstructor
public class VehiculeRestController {
    private VehiculeRepository vehiculeRepository;
    private ProprietaireRepository proprietaireRepository;
    // private ProprietaireServiceClient proprietaireServiceClient;
    // private VehiculeServiceClient vehiculeServiceClient;
    @GetMapping("/vehicules/full/{id}")
    public Vehicule getVhicule(@PathVariable(name = "id") Long id) {
        Vehicule vehicule = vehiculeRepository.findById(id).get();
        //Optional<Proprietaire>
        proprietaire=proprietaireRepository.findById(vehicule.getIdProprietaire());
        //vehicule.setProprietaire(proprietaire);
        return vehicule;
    }
    @GetMapping("/immatric/{mat}")
    public Vehicule findByMat(@PathVariable(name = "mat") String mat) {
        System.out.println(mat);
        Vehicule vehicule = vehiculeRepository.findByMat(mat).get();
        return vehicule;
    }
    @GetMapping("/vehicules")
    public List<Vehicule> getAllVehicule() {
        return vehiculeRepository.findAll();
    }
    @PostMapping("/vehicules")
    public Vehicule save(@RequestBody Vehicule request ) {
        return vehiculeRepository.save(request);
    }
    @PutMapping("/vehicules/{id}")
    public Vehicule update(@PathVariable Long id,@RequestBody Vehicule
    vehicule ) {
        Vehicule vehicule1=vehiculeRepository.findById(id).orElseThrow();
        if (vehicule.getMat()!=null)vehicule1.setMat(vehicule.getMat());
        if
        (vehicule.getMarque()!=null)vehicule1.setMarque(vehicule.getMarque());
        if
        (vehicule.getPuissanceFiscal()!=null)vehicule1.setPuissanceFiscal(vehicule.
        getPuissanceFiscal());
    }
}

```

```

if (vehicule.getModele() != null) vehicule1.setModele(vehicule.getModele());
    if
(vehicule.getIdProprietaire() != null) vehicule1.setIdProprietaire(vehicule.getIdProprietaire());

if (vehicule.getProprietaire() != null) vehicule1.setProprietaire(vehicule.getProprietaire());
    return vehiculeRepository.save(vehicule1);
}
@DeleteMapping("/vehicules/{id}")
public void deletePVehicule(@PathVariable Long id) {
    vehiculeRepository.deleteById(id);
}
}

```

- **ImmatriculationGraphqlController :**

```

import java.util.List;
import java.util.Optional;

@Controller
public class ImmatriculationGraphqlController {
    @Autowired
    private ProprietaireRepository proprietaireRepository;
    @Autowired
    private VehiculeRepository vehiculeRepository;
    @QueryMapping
    public List<Proprietaire> proprietaireList() {
        return proprietaireRepository.findAll();
    }
    @QueryMapping
    public List<Vehicule> vehiculeList() {
        return vehiculeRepository.findAll();
    }
    @QueryMapping
    public Proprietaire getProprietaireById(@Argument Long id)
    {
        Optional<Proprietaire> proprietaire=
        proprietaireRepository.findById(id);
        Proprietaire proprietaire1=new Proprietaire();

        proprietaire1.setDateNaissance(proprietaire.get().getDateNaissance());
        proprietaire1.setNom(proprietaire.get().getNom());
        proprietaire1.setIdP(proprietaire.get().getIdP());
        proprietaire1.setEmail(proprietaire.get().getEmail());
        return proprietaire1;
    }
    @MutationMapping
    public Proprietaire saveNewProprietaire(@Argument Proprietaire request) {
        return proprietaireRepository.save(request);
    }
    @MutationMapping
    public Proprietaire updateProprietaire(@Argument Proprietaire request)
    {
        Optional<Proprietaire> proprietaire=
        proprietaireRepository.findById(request.getIdP());
        Proprietaire proprietaire1=new Proprietaire();

```



```

proprietaire1.setDateNaissance(proprietaire.get().getDateNaissance());
proprietaire1.setNom(proprietaire.get().getNom());
proprietaire1.setIdP(proprietaire.get().getIdP());
proprietaire1.setEmail(proprietaire.get().getEmail());
proprietaireRepository.delete(proprietaire1);
return proprietaireRepository.save(request);
}
@MutationMapping
public Boolean deleteProprietaire(@Argument Long id) {
    proprietaireRepository.deleteById(id);
    return true;
}
}

```

- **application.properties :**

```

spring.cloud.discovery.enabled=true
spring.h2.console.enabled=true
server.port=8084
spring.application.name=immatriculation-service
spring.datasource.url=jdbc:h2:mem:immatriculation-db
spring.graphql.graphiql.enabled=true
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

```

- **Schema.graphqls :**

```

type Query{
    proprietaireList:[Proprietaire]
    vehiculeList:[Vehicule]
    getProprietaireById(id:Int): Proprietaire
}
type Mutation {
    saveNewProprietaire(request: ProprietaireRequest) : Proprietaire,
    updateProprietaire(request: ProprietaireRequest) : Proprietaire,
    deleteProprietaire(id : Int) : Boolean
}

type Proprietaire{
    id:Float,
    nom:String,
    dateNaissance:String,
    email:String
}
input ProprietaireRequest{
    id:Float,
    nom:String,
    dateNaissance:String,
    email:String
}
type Vehicule{
    idv: Float,
    mat:String,
    marque : String,
    puissanceFiscal: Int,
    modele: String,

```

```

    idProprietaire: Float
}

```

• ImmatriculationServiceApplication :

```

import org.sdia.immatriculation.Entities.Proprietaire;
import org.sdia.immatriculation.Entities.Vehicule;
import org.sdia.immatriculation.Repository.ProprietaireRepository;
import org.sdia.immatriculation.Repository.VehiculeRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;

import java.text.SimpleDateFormat;
import java.util.Date;

@SpringBootApplication
@EnableFeignClients
public class ImmatriculationServiceApplication {
    public static void main(String[] args)
    {SpringApplication.run(ImmatriculationServiceApplication.class, args);
    }

    @Bean
    CommandLineRunner start(ProprietaireRepository
    proprietaireRepository, VehiculeRepository vehiculeRepository)
    {

        return args -> {
            String dateString = "2023-05-15";
            SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd");
            Date date = format.parse(dateString);
            Proprietaire proprietaire1= new
            Proprietaire(null, "Mahfoud", date, "Sanaa.mahfoud88@gmail.com");
            Proprietaire proprietaire2=new
            Proprietaire(null, "Maftouh", date, "Oumaima.maftouh12@gmail.com");
            proprietaireRepository.save(proprietaire1);
            proprietaireRepository.save(proprietaire2);
            Vehicule vehicule1=new
            Vehicule(null, "8965244", "Mercedes", 585, "AMG
            GTR", proprietaire1.getIdP(), proprietaire1);
            Vehicule vehicule2=new
            Vehicule(null, "JHfdtdtt", "Ferrari", 570, "458", proprietaire2.getIdP(), pr
            oprietaire2);
            vehiculeRepository.save(vehicule1);
            vehiculeRepository.save(vehicule2);

        };
    }
}

```

- Les tests :

```
localhost:8084/proprietaires

Interface graphique... MAN Méthode d'extracti... Python Examinons un peu... New

[
  {
    "idP": 1,
    "nom": "Mahfoud",
    "dateNaissance": "2023-05-14T22:00:00.000+00:00",
    "email": "Sanaa.mahfoud88@gmail.com"
  },
  {
    "idP": 2,
    "nom": "Maftouh",
    "dateNaissance": "2023-05-14T22:00:00.000+00:00",
    "email": "Oumaima.maftouh12@gmail.com"
  }
]
```

```
localhost:8084/vehicules

Interface graphique... MAN Méthode d'extracti... Python Examin

[
  {
    "idv": 1,
    "mat": "8965244",
    "puissanceFiscal": 585,
    "idProprietaire": 1,
    "proprietaire": null,
    "modele": "AMG GTR",
    "marque": "Mercedes"
  },
  {
    "idv": 2,
    "mat": "JHfdtdtt",
    "puissanceFiscal": 570,
    "idProprietaire": 2,
    "proprietaire": null,
    "modele": "458",
    "marque": "Ferrari"
  }
]
```

➤ On utilise l'utile de test « Postman »

The image shows two screenshots of the Postman application interface. Both screenshots show a GET request being made to a local server.

Top Screenshot:

- Method: GET
- URL: `http://localhost:8084/proprietaires`
- Body tab is selected, showing a JSON array of two objects:

```
[
  {
    "idP": 1,
    "nom": "Mahfoud",
    "dateNaissance": "2023-05-14T22:00:00.000+00:00",
    "email": "Sanaa.mahfoud88@gmail.com"
  },
  {
    "idP": 2,
    "nom": "Maftouh",
    "dateNaissance": "2023-05-14T22:00:00.000+00:00",
    "email": "Oumaima.maftouh12@gmail.com"
  }
]
```

Bottom Screenshot:

- Method: GET
- URL: `http://localhost:8084/vehicules`
- Body tab is selected, showing a JSON array of two objects:

```
[
  {
    "idv": 1,
    "mat": "8965244",
    "puissanceFiscal": 585,
    "idProprietaire": 1,
    "proprietaire": null,
    "modele": "AMG GTR",
    "marque": "Mercedes"
  },
  {
    "idv": 2,
    "mat": "JHfdtdtt",
    "puissanceFiscal": 570,
    "idProprietaire": 2,
    "proprietaire": null,
    "modele": "458",
    "marque": "Ferrari"
  }
]
```

POST http://localhost:8084/proprietaire

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

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

```
1 {
2   .... "nom": "Sanim",
3   .... "dateNaissance": "2023-05-14T22:00:00.000+00:00",
4   .... "email": "Meryem.sanim@gmail.com"
5 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "idP": 3,
3   "nom": "Sanim",
4   "dateNaissance": "2023-05-14T22:00:00.000+00:00",
5   "email": "Meryem.sanim@gmail.com"
6 }
```

<untitled> <untitled> x <untitled> <untitled> + GraphQL

```
1 query(proprietaireList{id,nom,email,dateNaissance})
```





```
{
  "data": {
    "proprietaireList": [
      {
        "id": null,
        "nom": "Mahfoud",
        "email": "Sanaa.mahfoud88@gmail.com",
        "dateNaissance": "2023-05-15 00:00:00.0"
      },
      {
        "id": null,
        "nom": "Maftouh",
        "email": "Oumaima.maftouh12@gmail.com",
        "dateNaissance": "2023-05-15 00:00:00.0"
      },
      {
        "id": null,
        "nom": "Sanim",
        "email": "Meryem.sanim@gmail.com",
        "dateNaissance": "2023-05-15 00:00:00.0"
      }
    ]
  }
}
```

Variables Headers

Accédez aux paramètres pour activer Windows.

<untitled> <untitled> x <untitled> <untitled> +





```
1 query{vehiculeList{idv,mat,marque,modele}}
```



```
{
  "data": {
    "vehiculeList": [
      {
        "idv": 1,
        "mat": "8965244",
        "marque": "Mercedes",
        "modele": "AMG GTR"
      },
      {
        "idv": 2,
        "mat": "JHfdtdtt",
        "marque": "Ferrari",
        "modele": "458"
      }
    ]
  }
}
```

<untitled> x <untitled> <untitled> <untitled> + GraphQL





```
1 mutation{saveNewProprietaire(request:{
2   id:1,
3   nom:"Mahfoud",
4   dateNaissance:"07/07/2001",
5   email:"Sakura.sanaa77@gmail.com"
6 }}){id,nom,dateNaissance,email}
7 }
```



```
{
  "data": {
    "saveNewProprietaire": {
      "id": null,
      "nom": "Mahfoud",
      "dateNaissance": "Sat Jul 07 00:00:00 CEST 2001",
      "email": "Sakura.sanaa77@gmail.com"
    }
  }
}
```

<untitled> <untitled> <untitled> <untitled> x +

```
1 mutation{deleteProprietaire(id:1)}
```



```
{
  "data": {
    "deleteProprietaire": true
  }
}
```

2- GRPC Service :

- **Immat.proto :**

```
syntax="proto3";
option java_package="org.sdia.immatriculation.GRPC_Service.stubs";
service ImmatriculationService{
    // déclarer les methodes
    rpc affichagePro(proprietaire) returns(proprietaire); // unary model
    rpc affichageVehi(vehicule) returns(vehicule); // unary model
}
message proprietaire{
    double idp=1;
    string nom=2;
    string dateNaissance=3;
}

message vehicule{
    double idv=1;
    string numMatricule=2;
    string marque=3;
    int32 puissanceFiscal=4;
    string modele=5;
    proprietaire idProprietaire=6;
}
```

- **ImmatriculationGrpcSERVICE :**

```
import io.grpc.stub.StreamObserver;
import org.sdia.immatriculation.GRPC_Service.stubs.Immat;
import
org.sdia.immatriculation.GRPC_Service.stubs.ImmatriculationServiceGrpc;

public class ImmatGrpcService extends
ImmatriculationServiceGrpc.ImmatriculationServiceImplBase{
    @Override
    public void affichagePro(Immat.proprietaire proprietaire,
StreamObserver<Immat.proprietaire> responseProprietaire){
        double idp=proprietaire.getIdp();
        String nom=proprietaire.getNom();
        String DateNaiss=proprietaire.getDateNaissance();
        Immat.proprietaire proprietaire1=Immat.proprietaire.newBuilder()
            .setIdp(idp)
            .setNom(nom)
            .setDateNaissance(DateNaiss)
            .build();
        responseProprietaire.onNext(proprietaire1);
        responseProprietaire.onCompleted();
    }
    @Override
    public void affichageVehi(Immat.vehicule vehicule,
StreamObserver<Immat.vehicule> responseVehicul){
        double idv=vehicule.getIdv();
        String numMatricule=vehicule.getNumMatricule();
        String marque=vehicule.getMarque();
```

```

        int puissanceFiscal=vehicule.getPuissanceFiscal();
        String modele=vehicule.getModele();
        Immat.propretaire proprietaire=vehicule.getIdProprietaire();
        Immat.vehicule vehicule1=Immat.vehicule.newBuilder()
                .setIdv(idv)
                .setNumMatricule(numMatricule)
                .setMarque(marque)
                .setPuissanceFiscal(puissanceFiscal)
                .setModele(modele)
                .setIdProprietaire(proprietaire)
                .build();
        responseVehicul.onNext(vehicule1);
        responseVehicul.onCompleted();
    }

}

```

- **GrpcServer :**

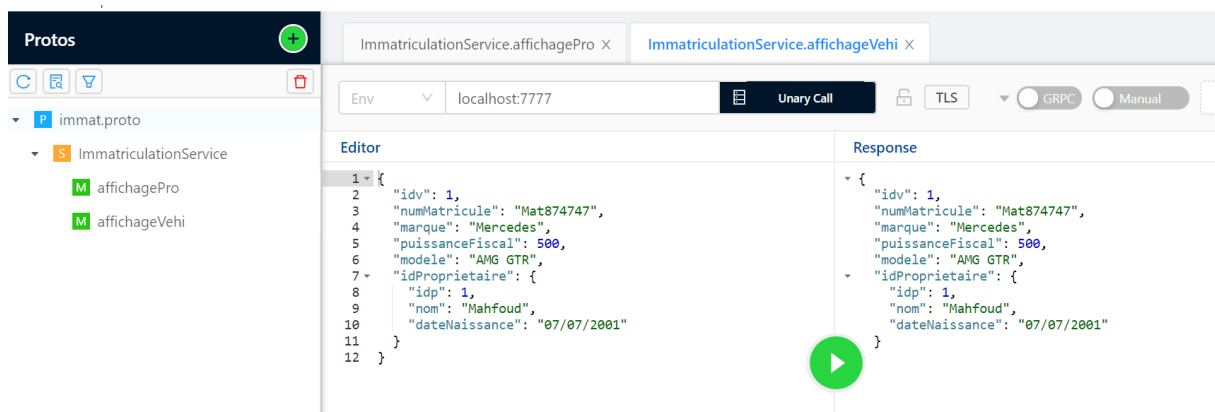
```

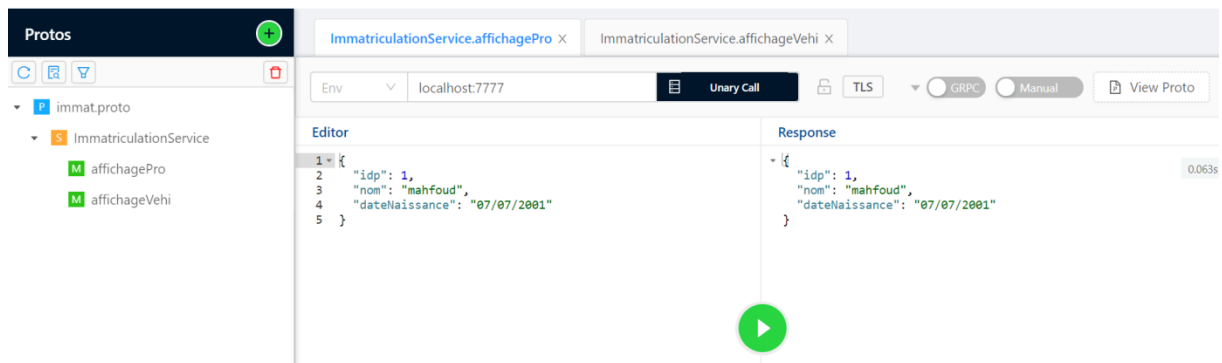
import java.io.IOException;

public class GrpcServer {
    public static void main(String[] args) throws IOException,
        InterruptedException {
        Server server= ServerBuilder.forPort(7777)
                .addService(new ImmatGrpcService())
                .build();
        server.start();
        server.awaitTermination();
    }
}

```

- **Les tests avec bloomRPC :**





3- Soap Service

- **Immatriculation Service soap :**

```
import jakarta.jws.WebMethod;
import jakarta.jws.WebParam;
import jakarta.jws.WebService;
import org.sdia.immatriculation.Entities.Proprietaire;
import org.sdia.immatriculation.Entities.Vehicule;

import java.util.Date;
import java.util.List;

@WebService(serviceName="ImmatriculationWS")
public class ImmatriculationServiceSoap {
    @WebMethod(operationName="AfficheProprietaire")
    public Proprietaire getProprietaire(@WebParam(name="idp") Long idp) {
        return new Proprietaire(idp, "Mahfoud", new Date(), "Sanaa.mahfoud88@gmail.com");
    }
    @WebMethod
    public List<Proprietaire> listProprietaire() {
        return List.of(new Proprietaire(Long.parseLong("1"), "Mahfoud", new Date(), "Sanaa.mahfoud88@gmail.com"),
            new Proprietaire(Long.parseLong("2"), "Maftouh", new Date(), "Oumaima.maftouh12@gmail.com"),
            new Proprietaire(Long.parseLong("3"), "Sanim", new Date(), "Meryem.sanim10@gmail.com"));
    }
    @WebMethod(operationName="AfficheVehicule")
    public Vehicule getVehicule(@WebParam(name="vehicule") Vehicule vehicule) {
        return vehicule;
    }
    @WebMethod
    public List<Vehicule> listVehicule() {
        return List.of(new Vehicule(Long.parseLong("1"), "Mat36536366", "Mercedes", 585, "AMG GT", Long.parseLong("1"), null),

```

```

        new
Vehicule(Long.parseLong("2"), "Mat36539864", "BMW", 231, "I8", Long.parseLong
("2"), null),

        new
Vehicule(Long.parseLong("3"), "Mat36577664", "Ferrari", 570, "458", Long.pars
eLong("3"), null)
    );
}
}

```

- **Server soap :**

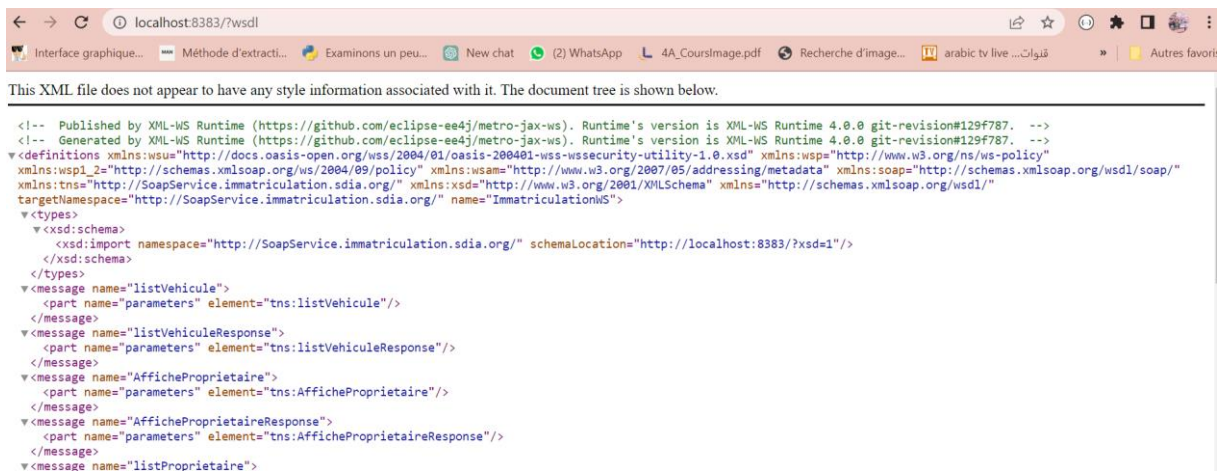
```

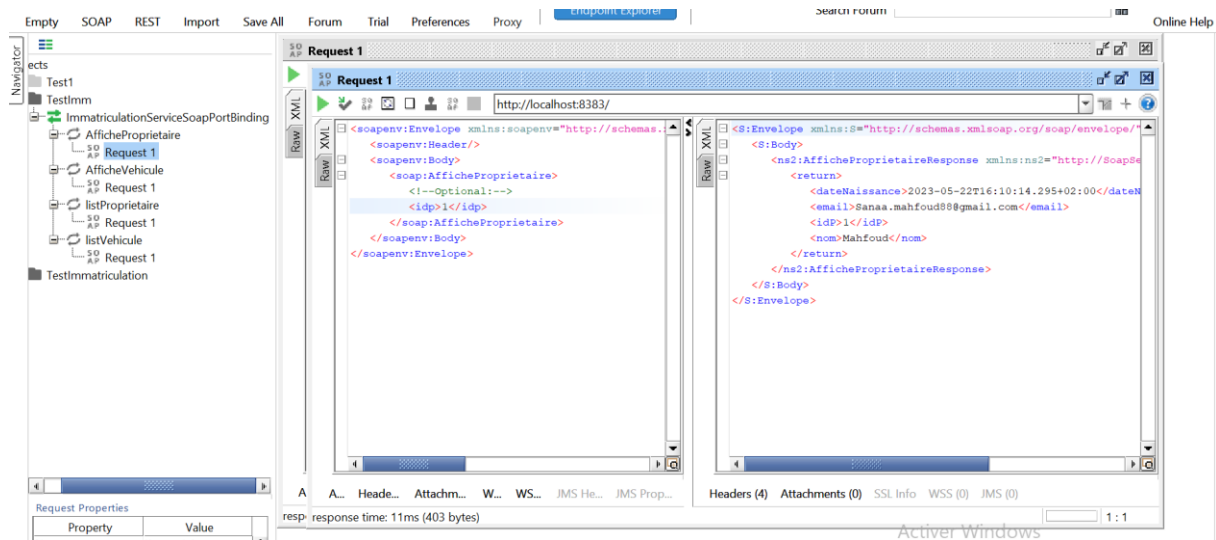
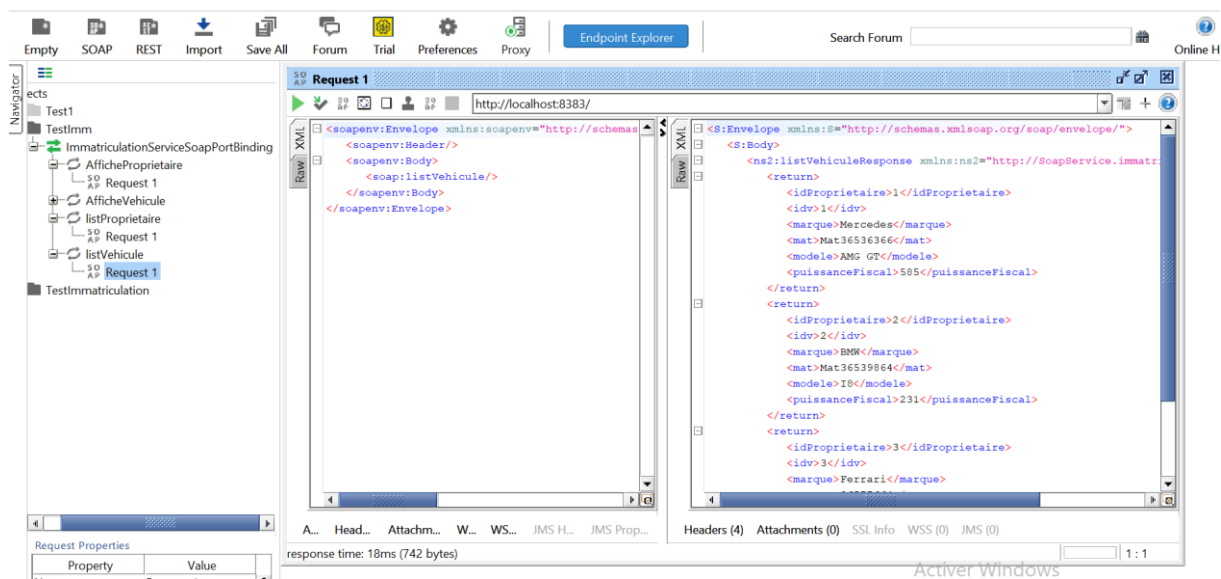
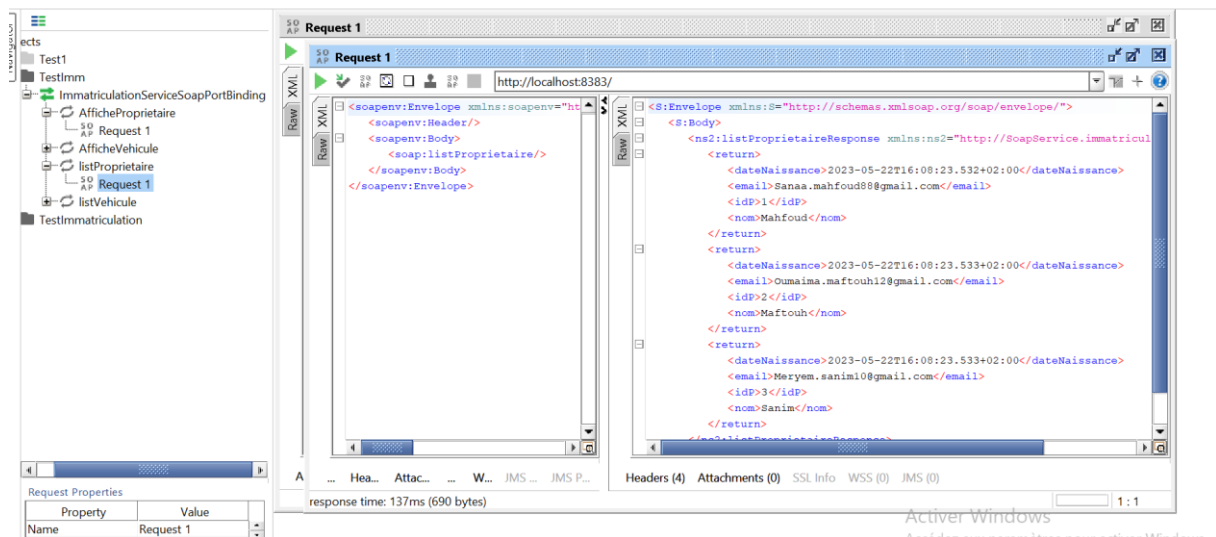
import jakarta.xml.ws.Endpoint;

public class ServerSoap {
    public static void main(String[] args) {
        // Endpoint est une classe de ws qui permet de demarrer un
        // petite serveur http. 0.0.0.0 permet a des application distante de
        // consulter ce web service
        Endpoint.publish("http://0.0.0.0:8383/", new
        ImmatriculationServiceSoap());
        System.out.println("Web service deployé sur port 8383");
    }
}

```

- **Les tests : On utilise « SoapUI »**





II. Le micro Service RADAR

1- Entities :

- **Radar :**

```
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class Radar {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Integer vitessMax;
    private String Longitude ;
    private String Latitude;
}
```

- **Proprietaire :**

```
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Date;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class proprietaire {
    private Long idP;
    private String nom;
    private Date dateNaissance;
}
```

- **Vehicule :**

```
import jakarta.persistence.Transient;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
```

```

@AllArgsConstructor
@NoArgsConstructor
public class vehicule {
    private Long idv;
    private String mat;
    private String Marque;
    private Integer puissanceFiscal;
    private String Modele;
    private Long idProprietaire;
    @Transient
    private proprietaire proprietaire ;
}

```

- **Infraction :**

```

import jakarta.persistence.Transient;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Infraction {
    private Long idI;
    private Date date;
    private int vitesseMax;
    private Double vitessVehicul;
    private Double montantInfr;
    private Long radarID;
    private String vehiculMatr;
    private Long vehiculId;
    @Transient
    private Radar radar;
    @Transient
    private vehicule vehicule;
}

```

- **DetectVitesse :**

```

import lombok.Data;
import lombok.ToString;

@Data
@ToString
public class DetectVitesse {
    public static String matriculeVoiture;
    public static double vitessVoiture;
    public static double idRadar;
}

```

2- Service :

- **ImmatriculationServiceClient :**

```
import org.sdia.Radar.Entities.proprietaire;
import org.sdia.Radar.Entities.vehicule;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="immatriculation-service")
public interface ImmatriculationServiceClient {
    @GetMapping("/proprietaires/full/{id}")
    proprietaire findProprietaireById(@PathVariable("id") Long id);

    @GetMapping("/vehicules/full/{id}")
    vehicule findVehiculeById(@PathVariable("id") Long id);
    // @GetMapping("/immatric/{mat}")
    @GetMapping("/immatric/{mat}")
    vehicule findByMat(@PathVariable(name = "mat") String mat);
    @GetMapping("/vehicules/{idProprietaire}")
    proprietaire findProprietaireById(@PathVariable("idProprietaire")
Long idProprietaire);
}
```

- **InfractionServiceClient :**

```
import org.sdia.Radar.Entities.Infraction;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;

@FeignClient(name="infraction-service")
public interface InfractionServiceClient {
    @PostMapping("/infractions")
    Infraction save(@RequestBody Infraction request );
}
```

3- Controllers :

- **RadarRestController :**

```
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;
import org.sdia.Radar.DetectGrpcService.Service.RestData;
import org.sdia.Radar.Entities.Infraction;
import org.sdia.Radar.Entities.Radar;
import org.sdia.Radar.Entities.proprietaire;
import org.sdia.Radar.Entities.vehicule;
import org.sdia.Radar.Repository.radarRepository;
import org.sdia.Radar.Service.ImmatriculationServiceClient;
```

```

import org.sdia.Radar.Service.InfractionServiceClient;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Date;
import java.util.List;
import java.util.Optional;

@AllArgsConstructor
@NoArgsConstructor
@RestController
public class radarRestController {
    @Autowired
    private radarRepository radarRepository;
    @Autowired
    private InfractionServiceClient infractionServiceClient;
    @Autowired
    private ImmatriculationServiceClient immatriculationServiceClient;

    @GetMapping("/radars/{id}")
    public Radar getRadars(@PathVariable(name = "id") Long id) {
        Radar radar = radarRepository.findById(id).get();
        return radar;
    }
    @GetMapping("/vehicule/{id}")
    public vehicule findVehicule(@PathVariable(name = "id") Long id) {
        System.out.println(id);
        vehicule vehicule =
        immatriculationServiceClient.findVehiculeById(id);
        return vehicule;
    }
    @GetMapping("/immatric/{mat}")
    public vehicule findVehiculeMat(@PathVariable(name = "mat") String
mat) {
        System.out.println(mat);
        vehicule vehicule1 =
        immatriculationServiceClient.findByMat(mat);
        return vehicule1;
    }
    @PostMapping("/api/data")
    public Optional<Radar> receiveDetectData(@RequestBody RestData
restData) {
        // Traiter les données reçues de gRPC
        String vehiculeMatriculation =
restData.getVehiculeMatriculation();
        //Double idRadar = Request.getIdRadar();
        Long
idRadar=Double.valueOf(restData.getIdRadar()).longValue();
        Double vehiculeVitesse = restData.getVehiculeVitesse();
        Optional<Radar> radar=radarRepository.findById(idRadar);
        vehicule
vehicule1=immatriculationServiceClient.findByMat(vehiculeMatriculation
);
        proprietaire
proprietaire1=immatriculationServiceClient.findProprietaireById(vehicu
le1.getIdProprietaire());
        System.out.println(proprietaire1);
        Infraction infraction=new Infraction(null,new
Date(),radar.get().getVitesseMax(),
vehiculeVitesse,500.0,idRadar,vehiculeMatriculation,vehicule1.getIdv()
,null,null);

```

```

        infractionServiceClient.save(infraction);
        return radar;
    }

    @GetMapping("/radars")
    public List<Radar> getAllRadars() {
        return radarRepository.findAll();
    }

    @PostMapping("/radars")
    public Radar save(@RequestBody Radar request) {
        return radarRepository.save(request);
    }

    @PutMapping("/radars/{id}")
    public Radar update(@PathVariable Long id, @RequestBody Radar radar) {
        Radar radar1 = radarRepository.findById(id).orElseThrow();
        if (radar.getVitesseMax() != null) radar1.setVitesseMax(radar.getVitesseMax());
        if (radar.getLongitude() != null) radar1.setLongitude(radar.getLongitude());
        if (radar.getLatitude() != null) radar1.setLatitude(radar.getLatitude());
        return radarRepository.save(radar1);
    }

    @DeleteMapping("/radars/{id}")
    public void deleteRadar(@PathVariable Long id) {
        radarRepository.deleteById(id);
    }

    public proprietaire findProprietaireById(Long id) {
        return immatriculationServiceClient.findProprietaireById(id);
    }
}

```

• RadarGraphQL :

```

import org.sdia.Radar.Entities.Radar;
import org.sdia.Radar.Repository.radarRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.graphql.data.method.annotation.QueryMapping;
import org.springframework.stereotype.Controller;

import java.util.List;

@Controller
public class radarGraphQLController {
    @Autowired
    private radarRepository radarRepository;
    @QueryMapping
    public List<Radar> RadarsList() {
        return radarRepository.findAll();
    }
}

```


4- RadarApplication :

```
import org.sdia.Radar.Entities.Radar;
import org.sdia.Radar.Entities.vehicule;
import org.sdia.Radar.Repository.radarRepository;
import org.sdia.Radar.Service.ImmatriculationServiceClient;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import
org.springframework.data.rest.core.config.RepositoryRestConfiguration;
@EnableFeignClients
@SpringBootApplication
public class RadarApplication {

    public static void main(String[] args) {
        SpringApplication.run(RadarApplication.class, args);
    }

    @Bean
    CommandLineRunner start(radarRepository radarRepository,
RepositoryRestConfiguration repositoryRestConfiguration,
ImmatriculationServiceClient immatriculationServiceClient) {
        repositoryRestConfiguration.exposeIdsFor(Radar.class);
        return args -> {

            radarRepository.save(new Radar(null, 120, "2.3522°E",
"48.8566°N"));
            radarRepository.save(new Radar(null, 80, "74.0060°W",
"40.7128°N"));
            radarRepository.save(new Radar(null, 140, "-43.1729°W", " -
22.9068°S"));
            radarRepository.findAll().forEach(System.out::println);

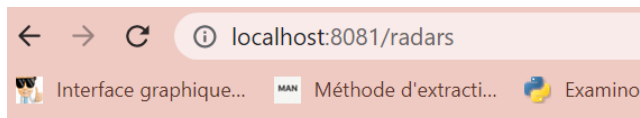
        };

    }
}
```

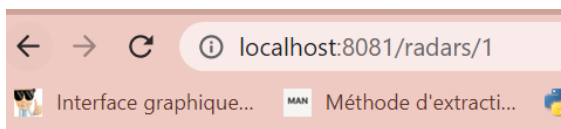
5- Application.properties :

```
spring.cloud.discovery.enabled=true
server.port=8081
spring.h2.console.enabled=true
spring.application.name=radar-service
management.endpoints.web.exposure.include=*
spring.datasource.url=jdbc:h2:mem:radar-db
eureka.client.service-url.defaultZone=http://localhost:8761/eureka
spring.main.allow-bean-definition-overriding=true
```

6- Les tests :



```
[
  {
    "id": 1,
    "vitessMax": 120,
    "longitude": "2.3522°E",
    "latitude": "48.8566°N"
  },
  {
    "id": 2,
    "vitessMax": 80,
    "longitude": "74.0060°W",
    "latitude": "40.7128°N"
  },
  {
    "id": 3,
    "vitessMax": 140,
    "longitude": "-43.1729°W",
    "latitude": "-22.9068°S"
  }
]
```



```
{
  "id": 1,
  "vitessMax": 120,
  "longitude": "2.3522°E",
  "latitude": "48.8566°N"
}
```

HTTP <http://localhost:8081/radars>

GET <http://localhost:8081/radars>

Params Authorization Headers (6) Body Pre-request Script Tests Se

Query Params

Key	Value
-----	-------

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "vitessMax": 120,
5     "longitude": "2.3522°E",
6     "latitude": "48.8566°N"
7   },
8   {
9     "id": 2,
10    "vitessMax": 80,
11    "longitude": "74.0060°W",
12    "latitude": "40.7128°N"
13  },
14 ]
```

HTTP <http://localhost:8081/radars>

POST <http://localhost:8081/radars>

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

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ Gr

```
1 {
2   ....{
3     ....{
4       ...."vitessMax": 160,
5       ...."longitude": "2.3522°E",
6       ...."latitude": "48.8566°N"
7     }
8   }
9 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 5,
3   "vitessMax": 160,
4   "longitude": "2.3522°E",
5   "latitude": "48.8566°N"
6 }
```


7- Service GRPC :

- **DetectGrpcService**

```
import io.grpc.stub.StreamObserver;
import org.sdia.Radar.DetectGrpcService.Stubs.Detect;
import org.sdia.Radar.DetectGrpcService.Stubs.DetectServiceGrpc;
import org.sdia.Radar.Service.ImmatriculationServiceClient;

import org.springframework.http.HttpEntity;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpMethod;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;

public class DetectGrpcService extends
DetectServiceGrpc.DetectServiceImplBase{

    private ImmatriculationServiceClient immatriculationServiceClient;
    private RestTemplate restTemplate;
    public DetectGrpcService() {
        this.restTemplate = new RestTemplate();
    }
    @Override
    public StreamObserver<Detect.DetectVitesse> envoyer (StreamObserver
<Detect.Response> response) {
        return new StreamObserver<Detect.DetectVitesse>() {
            @Override
            public void onNext(Detect.DetectVitesse Request) {
                // Extraire les données de l'objet Request
                String vehiculeMatriculation =
Request.getVehiculeMatriculation();
                //Double idRadar = Request.getIdRadar();
                Long
idRadar=Double.valueOf(Request.getIdRadar()).longValue();
                Double vehiculeVitesse = Request.getVehiculeVitesse();

                // Préparer les données pour l'API REST
                RestData restData = new RestData(vehiculeMatriculation,
vehiculeVitesse,idRadar);

                // Envoyer les données à l'API REST
                String apiUrl = "http://localhost:8081/api/data"; // URL de
votre API REST
                HttpHeaders headers = new HttpHeaders();
                headers.setContentType(MediaType.APPLICATION_JSON);
                HttpEntity<RestData> requestEntity = new
HttpEntity<>(restData, headers);
                ResponseEntity<Void> responseEntity =
restTemplate.exchange(apiUrl, HttpMethod.POST, requestEntity, Void.class);

                // Envoyer la réponse gRPC
            }
        };
    }
}
```

```

        Detect.Response response1 =
Detect.Response.newBuilder().setResponse(1).build();
        response.onNext(response1);

        //proprietaire proprietaire =
radarRestController.findProprietaireById(vehicule.getIdProprietaire());
        response.onNext(response1);

    }
    @Override
    public void onError(Throwable throwable) {
    }

    @Override
    public void onCompleted() {
        response.onCompleted();
    }
}
};
}
}

```

- **Server :**

```

import io.grpc.Server;
import io.grpc.ServerBuilder;
import org.sdia.Radar.DetectGrpcService.Service.DetectGrpcService;
import org.sdia.Radar.Entities.DetectVitesse;

import java.io.IOException;

public class server {
    public static void main(String[] args) throws IOException,
InterruptedException {
        Server server= ServerBuilder.forPort(4444)
            .addService(new DetectGrpcService())
            .build();
        server.start();
        server.awaitTermination();

    }
}

```

- **Fichier proto :**

```

syntax="proto3";
option java_package="org.sdia.Radar.DetectGrpcService.Stubs";
service DetectService{
    rpc envoyer(stream DetectVitesse) returns(stream Response);
}
message DetectVitesse{

```

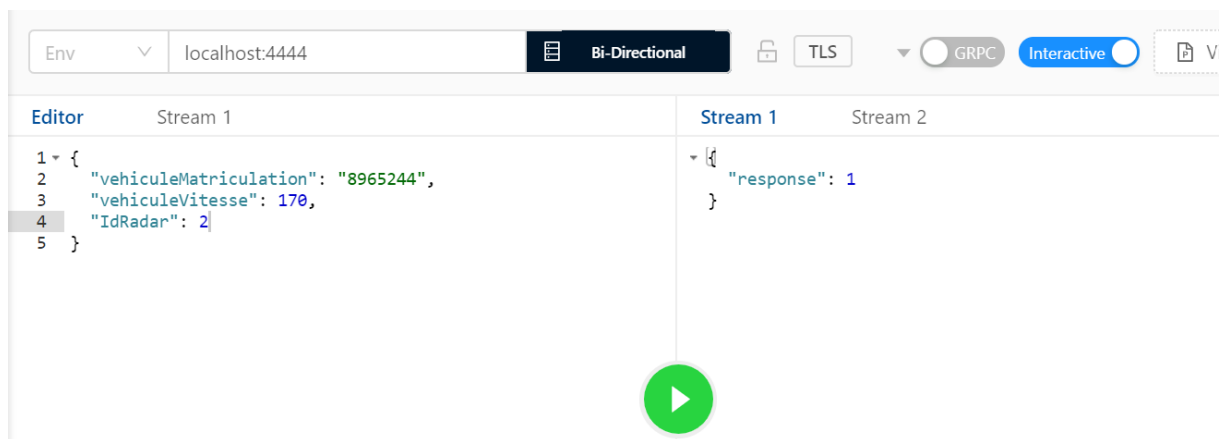
```

    string vehiculeMatriculation=1;
    double vehiculeVitesse=2;
    double IdRadar=3;
}

message Response{
    int32 response=1;
}

```

8- Le test pour enregistrer une infraction avec bloomRPC :



III. Micro Service Infraction :

1- Entities :

- **Infraction :**

```

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Infraction {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long idI;
    private Date date;
    private int vitesseMax;
    private int vitessVehicul;
    private Double montantInfr;
    private Long radarID;
}

```

```

        private String vehiculMatr;
        private Long vehiculId;
        @Transient
        private Radar radar;
        @Transient
        private Vehicule vehicule;
    }

```

- **Proprietaire :**

```

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import java.util.Date;
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Proprietaire {
    private Long idP;
    private String nom;
    private Date dateNaissance;
}

```

- **Radar :**

```

import lombok.Data;
@Data
public class Radar {
    private Long id;
    private Integer vitessMax;
    private String Longitude ;
    private String Latitude;
}

```

- **Radar :**

```

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Vehicule {
    private Long id;
    private String numMatricule;
    private String Marque;
    private Integer puissanceFiscal;
    private String Modele;
    private Long idProprietaire;
    @Transient private Proprietaire proprietaire ;
}

```


1. Repository :

```
import org.sdia.Infractions.Entities.Infraction;
import org.springframework.data.jpa.repository.JpaRepository;
import
org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface InfractionRepository extends
JpaRepository<Infraction, Long> {
}
```

2. Service Client :

- **ImmatriculationServiceClient**

```
import org.sdia.Infractions.Entities.Proprietaire;
import org.sdia.Infractions.Entities.Vehicule;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="immatriculation-service")
public interface ImmatriculationServiceClient {
    @GetMapping("/proprietaires/{id}")
    Proprietaire findProprietaireById(@PathVariable("id") Long id);
    @GetMapping("/vehicules/{id}")
    Vehicule findVehiculeById(@PathVariable("id") Long id);
}
```

- **RadarServiceClient**

```
import org.sdia.Infractions.Entities.Radar;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name="radar-service")
public interface RadarServiceClient {
    @GetMapping("/radars/{id}")
    Radar findRadarById(@PathVariable("id") Long id);
}
```

3. Rest Controller :

```
import lombok.AllArgsConstructor;
import org.sdia.Infractions.Entities.Infraction;
import org.sdia.Infractions.Entities.Radar;
import org.sdia.Infractions.Entities.Vehicule;
import org.sdia.Infractions.Repository.InfractionRepository;
import org.sdia.Infractions.ServiceClient.ImmatriculationServiceClient;
import org.sdia.Infractions.ServiceClient.RadarServiceClient;
import org.springframework.web.bind.annotation.*;

import java.util.List;
```

```

@AllArgsConstructor
@RestController
public class InfractionRestController {
    private InfractionRepository infractionRepository;
    private ImmatriculationServiceClient immatriculationServiceClient;
    private RadarServiceClient radarServiceClient;

    @GetMapping("/infractions/full/{id}")
    public Infraction getInfraction(@PathVariable(name="id") Long id){
        Infraction infraction=infractionRepository.findById(id).get();
        Radar
        radar=radarServiceClient.findRadarById(infraction.getRadarID());
        infraction.setRadar(radar);
        Vehicule
        vehicule=immatriculationServiceClient.findVehiculeById(infraction.getVehiculeId());
        infraction.setVehicule(vehicule);
        return infraction; }
    @GetMapping("/infractions")
    public List<Infraction> getAllInfraction(){
        return infractionRepository.findAll();
    }
    @PostMapping("/infractions")
    public Infraction save(@RequestBody Infraction request ){
        return infractionRepository.save(request);
    }
    @DeleteMapping("/infractions/{id}")
    public void deleteInfraction(@PathVariable Long id){
        infractionRepository.deleteById(id);
    }
}

```

5- Application.properties :

```

spring.cloud.discovery.enabled=true
server.port=8082
spring.application.name=infraction-service
management.endpoints.web.exposure.include=*
spring.datasource.url=jdbc:h2:mem:infraction-db
eureka.client.service-url.defaultZone=http://localhost:8761/eureka

```

6- Application :

```

import org.sdia.Infractions.Repository.InfractionRepository;
import org.sdia.Infractions.ServiceClient.ImmatriculationServiceClient;
import org.sdia.Infractions.ServiceClient.RadarServiceClient;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
@EnableFeignClients
@SpringBootApplication
public class InfractionsApplication {

    public static void main(String[] args) {
        SpringApplication.run(InfractionsApplication.class, args);
    }
}

```

```

    }

    @Bean
    CommandLineRunner start(InfractionRepository infractionRepository,
                            ImmatriculationServiceClient
                            immatriculationServiceClient, RadarServiceClient radarServiceClient) {

        return args -> {

            //Radar radar = radarServiceClient.findRadarById(1L);
            //System.out.println(radar.getVitesseMax());
            //Vehicule vehicule =
            immatriculationServiceClient.findVehiculeById(1L);
            //Infraction infraction = new Infraction(null, new Date(), 120,
            140, 500.0, radar.getId(), vehicule.getNumMatricule(), vehicule.getId(),
            null, null);
            //infractionRepository.save(infraction);

        };
    }
}

```

IV. Radar simulation

1- Fichier proto :

```

syntax="proto3";
option java_package="org.sdia.Stubs";
service DetectService{
    rpc envoyer(stream DetectVitesse) returns(stream Response);
}
message DetectVitesse{
    string vehiculeMatriculation=1;
    double vehiculeVitesse=2;
    double IdRadar=3;
}
message Response{
    int32 response=1;
}

```

2- Application :

```

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.stub.StreamObserver;
import org.sdia.Stubs.Detect;
import org.sdia.Stubs.DetectServiceGrpc;

import java.io.IOException;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) throws IOException {
        ManagedChannel managedChannel =
        ManagedChannelBuilder.forAddress("localhost", 4444)
        .usePlaintext()
    }
}

```

```

        .build();
        DetectServiceGrpc.DetectServiceStub asynStub =
        DetectServiceGrpc.newStub(managedChannel);
        StreamObserver<Detect.DetectVitesse> performStream =
        asynStub.envoyer(new StreamObserver<Detect.Response>() {
            @Override
            public void onNext(Detect.Response response) {
                System.out.println(response);
            }

            @Override
            public void onError(Throwable throwable) {

            }

            @Override
            public void onCompleted() {
                System.out.println("END---");
            }
        });

        Scanner sc = new Scanner(System.in);
        while (true) {

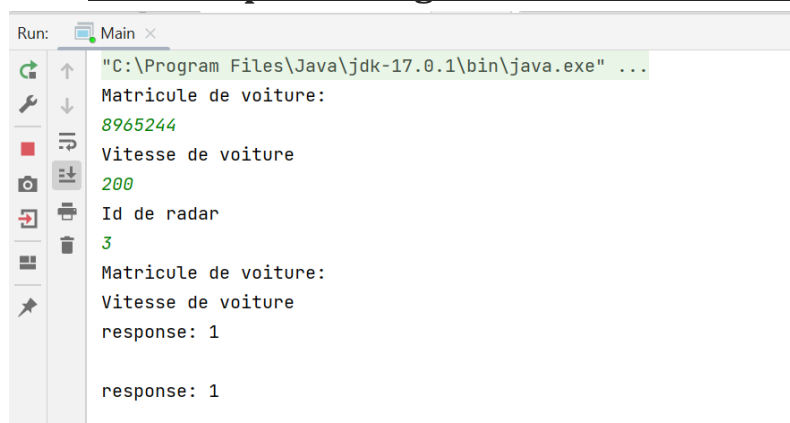
            System.out.println("Matricule de voiture:");
            String Matr = sc.nextLine();
            System.out.println("Vitesse de voiture");
            double Vitess = sc.nextDouble();
            System.out.println("Id de radar");
            double id = sc.nextDouble();

            Detect.DetectVitesse request =
            Detect.DetectVitesse.newBuilder()
                .setVehiculeMatriculation(Matr)
                .setVehiculeVitesse(Vitess)
                .setIdRadar(id)
                .build();
            performStream.onNext(request);

        }
    }
}

```

3- Les tests pour enregistrer une infraction :



Run

Run Selected

Auto complete

Clear

SQL statement:

SELECT * FROM INFRACTION

SELECT * FROM INFRACTION;

IDI	DATE	MONTANT_INFR	RADARID	VEHICUL_ID	VEHICUL_MATR	VITESS_VEHICUL	VITESSE_MAX
1	2023-05-27 20:49:31.242	500.0	2	1	8965244	170	80
2	2023-05-27 20:56:40.418	500.0	3	1	8965244	200	140

(2 rows, 2 ms)

Edit

V. Eureka Service

1- application.properties

```
server.port=8761
# dont register server itself as a client.
eureka.client.fetch-registry=false
# Does not register itself in the service registry.
eureka.client.register-with-eureka=false
```

DS Replicas			
localhost			
Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
IMMATRICULATION-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-O7RFTDV:immatriculation-service:8084
INFRACTION-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-O7RFTDV:infraction-service:8082
RADAR-SERVICE	n/a (1)	(1)	UP (1) - DESKTOP-O7RFTDV:radar-service:8081

VI. Gateway service

1- application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id: r1
          uri: http://localhost:8084/
          predicates:
            - Path= /vehicules/**

        - id: r2
          uri: http://localhost:8084/
          predicates:
            - Path= /proprietaires/**

        - id: r3
          uri: http://localhost:8081/
          predicates:
            - Path= /radars/**

        - id: r4
          uri: http://localhost:8082/
          predicates:
            - Path= /infractions/**

      discovery:
        enabled: true
server:
  port: 8888
```

VII. Conclusion

En conclusion, le système distribué basé sur les microservices proposé vise à gérer et automatiser le processus des infractions liées aux dépassements de vitesse détectés par des radars automatiques. Il est composé de trois microservices distincts et indépendants : le microservice de gestion des radars, le microservice d'immatriculation des véhicules et le microservice de gestion des infractions. Le microservice de gestion des radars permet de définir les radars par leur identifiant, leur vitesse maximale et leurs coordonnées géographiques (longitude et latitude).

Le microservice d'immatriculation gère les véhicules appartenant à des propriétaires. Chaque propriétaire est défini par son identifiant, son nom, sa date de naissance, son email, tandis que chaque véhicule est caractérisé par son identifiant, son numéro de plaque d'immatriculation, sa marque, sa puissance fiscale et son modèle.

Le microservice de gestion des infractions enregistre les infractions détectées, avec des informations telles que l'identifiant de l'infraction, la date, le numéro du radar ayant détecté le dépassement, le numéro de plaque d'immatriculation du véhicule concerné, la vitesse du véhicule, la vitesse maximale du radar et le montant de l'infraction. En plus des fonctionnalités de consultation et de modification des données, le système permet également de signaler un dépassement de vitesse, ce qui entraînera la création d'une infraction. De plus, il offre la possibilité à un propriétaire de consulter ses propres infractions.

En utilisant l'architecture des microservices, ce système peut être développé, déployé et évolué de manière indépendante pour chaque service, ce qui facilite la maintenance, la mise à l'échelle et les mises à jour spécifiques à chaque fonctionnalité.