



ECOLE NORMALE SUPÉRIURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

RAPPORT

Contrôle et Projet Programmation Distribuée

Élèves :
HANIF AYOUB

Enseignant :
mohamed YOUSSEFI

27 mai 2023

Table des matières

1	Introduction	2
2	Établir une architecture technique du projet	3
3	Établir un diagramme de classe global du projet	3
4	Développer le micro-service Immatriculation :	4
4.1	Entités JPA et Interface JpaRepository basées sur Spring data	4
4.1.1	Entité Owner	4
4.1.2	Entité Owner	4
4.1.3	Interface OwnerRepository	5
4.1.4	Interface VehiculeRepository	5
4.2	Les 4 web services REST, GraphQL, SOAP et GRPC	6
4.2.1	REST service	6
4.2.2	Test du REST service	7
4.2.3	GRAPHQL service	10
4.2.4	Soap service	13
4.2.5	GRPC service	14
5	Développer le micro-service Infraction :	16
5.1	Entité Infraction	17
5.2	Structure	17
6	Développer le micro-service Radar :	18
6.1	Entité Radar	18
6.2	Structure	19
7	Créer un application java qui permet de simuler un radar	19
7.1	Structure	19
8	Eureka Discovery Service	20
9	Gateway Service	20
10	Frontend avec Angular	21
10.1	Login page	21
10.2	Dashbord page	21
10.3	Liste des radars	22
10.4	Ajouter radar	22
10.5	Modifier radar	23
10.6	Liste des infractions	23
11	Conclusion	24

1 Introduction

Le projet consiste à développer un système de gestion d'immatriculation et d'infractions. Dans le cadre de ce projet, plusieurs étapes doivent être suivies, y compris l'établissement de l'architecture technique, la création d'un diagramme de classe global et le développement du micro-service d'immatriculation. Dans cette réponse.

un diagramme de classe global est créé pour représenter les différentes classes et leurs relations dans l'ensemble du projet. Cela permet d'avoir une vue d'ensemble des entités et de leurs interactions.

Ensuite, le développement du micro-service d'immatriculation est abordé. Cela implique la création d'entités JPA et d'une interface JpaRepository basées sur Spring Data pour gérer les données liées aux immatriculations des véhicules. En plus de cela, quatre web services sont développés : REST, GraphQL, SOAP et gRPC, pour fournir différentes interfaces de communication avec le micro-service.

Enfin, les quatre web services sont testés pour s'assurer de leur bon fonctionnement et de leur conformité aux exigences du projet.

2 Établir une architecture technique du projet

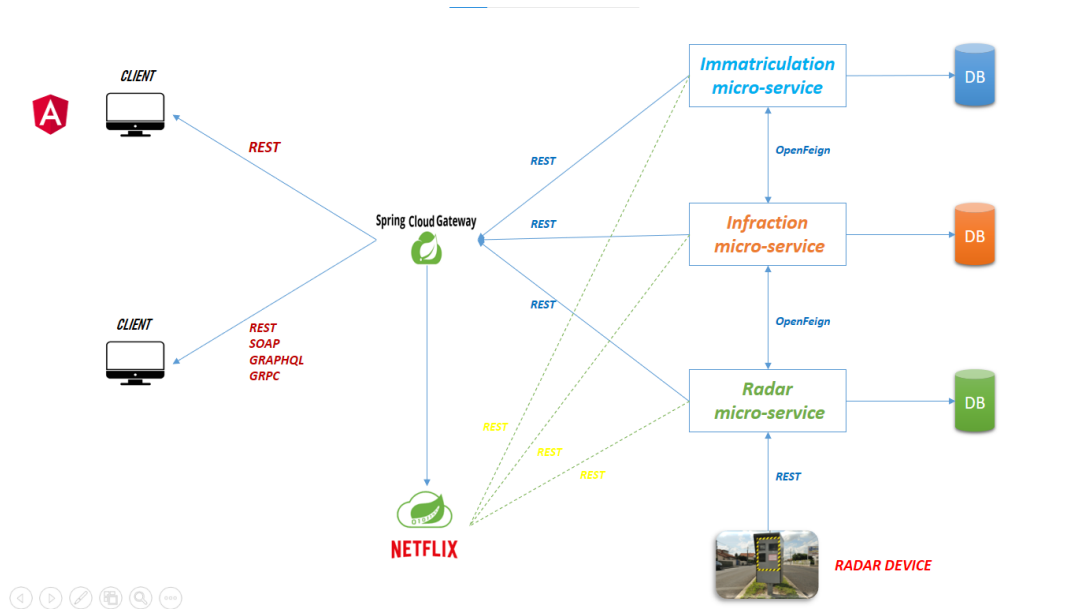


FIGURE 1

3 Établir un diagramme de classe global du projet

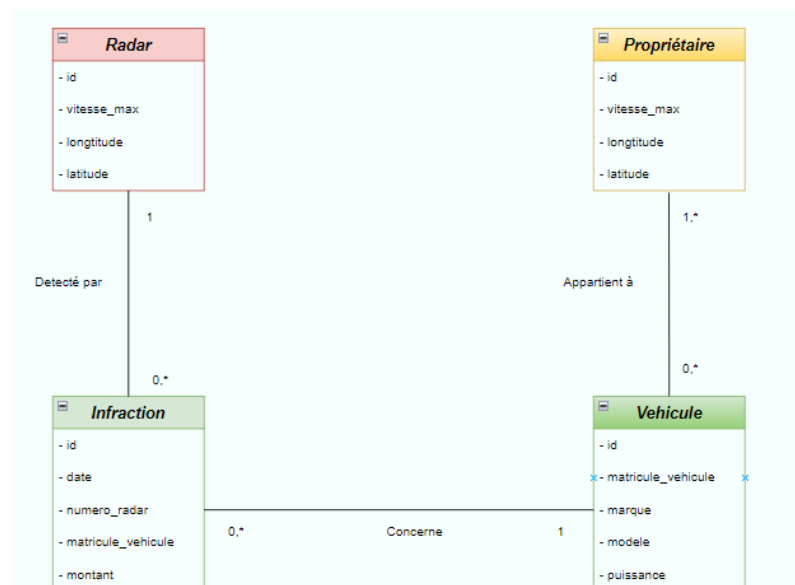


FIGURE 2

4 Développer le micro-service Immatriculation :

4.1 Entités JPA et Interface JpaRepository basées sur Spring data

4.1.1 Entité Owner

```

package ma.enset.immatriculationservice.entities;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.*;
import lombok.*;

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

6 usages
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString @Builder
public class Owner {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Date date_birth;
    private String email;
    @OneToMany(mappedBy = "owner")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<Vehicule> vehicules;
}

```

FIGURE 3

4.1.2 Entité Owner

```

package ma.enset.immatriculationservice.entities;

import com.fasterxml.jackson.annotation.JsonProperty;
import jakarta.persistence.*;
import lombok.*;

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

6 usages
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString @Builder
public class Owner {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private Date date_birth;
    private String email;
    @OneToMany(mappedBy = "owner")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<Vehicule> vehicules;
}

```

FIGURE 4

4.1.3 Interface OwnerRepository

```
package ma.enset.immatriculationservice.repositories;

import ma.enset.immatriculationservice.entities.Owner;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface OwnerRepository extends JpaRepository<Owner, Long> {
    |
}
```

FIGURE 5

4.1.4 Interface VehiculeRepository

```
package ma.enset.immatriculationservice.repositories;

import ma.enset.immatriculationservice.entities.Vehicule;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface VehiculeRepository extends JpaRepository<Vehicule, Long> {
    }
}
```

FIGURE 6

4.2 Les 4 web services REST, GraphQL, SOAP et GRPC

4.2.1 REST service

```
package ma.enset.immatriculationservice.web;

import ma.enset.immatriculationservice.entities.Owner;
import ma.enset.immatriculationservice.entities.Vehicule;
import ma.enset.immatriculationservice.repositories.OwnerRepository;
import ma.enset.immatriculationservice.repositories.VehiculeRepository;
import org.springframework.web.bind.annotation.*;

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

@RestController
public class ImmatriculationRestController {

    7 usages
    private OwnerRepository ownerRepository;
    8 usages
    private VehiculeRepository vehiculeRepository;

    public ImmatriculationRestController(OwnerRepository ownerRepository, VehiculeRepository vehiculeRepository) {
        this.ownerRepository=ownerRepository;
        this.vehiculeRepository=vehiculeRepository;
    }
}
```

FIGURE 7

```
@GetMapping("/vehicules")
public List<Vehicule> getAllVehicules() { return this.vehiculeRepository.findAll(); }

@GetMapping("/vehicules/{id}")
public Optional<Vehicule> getVehicule(@PathVariable Long id) { return this.vehiculeRepository.findById(id); }

@PostMapping("/vehicule")
public Vehicule save(@RequestBody Vehicule vehicule){
    this.ownerRepository.save(vehicule.getOwner());
    return this.vehiculeRepository.save(vehicule);
}

@PutMapping("/vehicule/{id}")
public Vehicule updateVehicule(@PathVariable Long id,@RequestBody Vehicule vehicule){
    Optional<Vehicule> vehiculeToUpdate=this.vehiculeRepository.findById(id);
    if(vehicule.getMarque()!=null) vehiculeToUpdate.get().setMarque (vehicule.getMarque());
    if(vehicule.getModel()!=null) vehiculeToUpdate.get().setModel (vehicule.getModel());
    if(vehicule.getPower()!=0) vehiculeToUpdate.get().setPower (vehicule.getPower());
    if(vehicule.getMatricule_number()!=null) vehiculeToUpdate.get().setMatricule_number (vehicule.getMatricule_number());
    if(vehicule.getOwner()!=null) vehiculeToUpdate.get().setOwner (vehicule.getOwner());
    return this.vehiculeRepository.save(vehiculeToUpdate.get());
}

@DeleteMapping("/vehicule/{id}")
public void deleteVehicule(@PathVariable Long id){
    this.vehiculeRepository.deleteById(id);
}
}
```

FIGURE 8

```

@GetMapping("/owners")
public List<Owner> getAllOwners() { return this.ownerRepository.findAll(); }

@GetMapping("/owners/{id}")
public Optional<Owner> getOwner(@PathVariable Long id) { return this.ownerRepository.findById(id); }

@PostMapping("/owner")
public Owner save(@RequestBody Owner owner){
    return this.ownerRepository.save(owner);
}

@PutMapping("/owner/{id}")
public Owner updateOwner(@PathVariable Long id, @RequestBody Owner owner){
    Optional<Owner> ownerToUpdate=this.ownerRepository.findById(id);
    if(owner.getName()!=null) ownerToUpdate.get().setName (owner.getName());
    if(owner.getEmail()!=null) ownerToUpdate.get().setEmail (owner.getEmail());
    if(owner.getDate_birth()!=null) ownerToUpdate.get().setDate_birth (owner.getDate_birth());
    if(owner.getVehicules()!=null) ownerToUpdate.get().setDate_birth (owner.getDate_birth());
    return this.ownerRepository.save(owner);
}

@DeleteMapping("/owner/{id}")
public void deleteOwner(@PathVariable Long id){
    this.ownerRepository.deleteById(id);
}

```

FIGURE 9

4.2.2 Test du REST service

==> *Au navigateur*

Afficher les véhicules

← → ↻ localhost:8080/vehicules

```

1 // 20230515190538
2 // http://localhost:8080/vehicules
3
4 [
5   {
6     "id": 1,
7     "matricule_number": 12345,
8     "model": "model ABC",
9     "marque": "marque ABC",
10    "power": 999,
11    "owner": {
12      "id": 1,
13      "name": "ali",
14      "date_birth": "2023-05-15T18:03:13.453+00:00",
15      "email": "email@email"
16    }
17  },
18   {
19     "id": 2,
20     "matricule_number": 12345,
21     "model": "model ABC",
22     "marque": "marque ABC",
23     "power": 999,
24     "owner": {
25       "id": 2,
26       "name": "anas",
27       "date_birth": "2023-05-15T18:03:13.534+00:00",
28       "email": "email@email"
29     }
30   }
31 ]

```

FIGURE 10

Afficher une véhicule

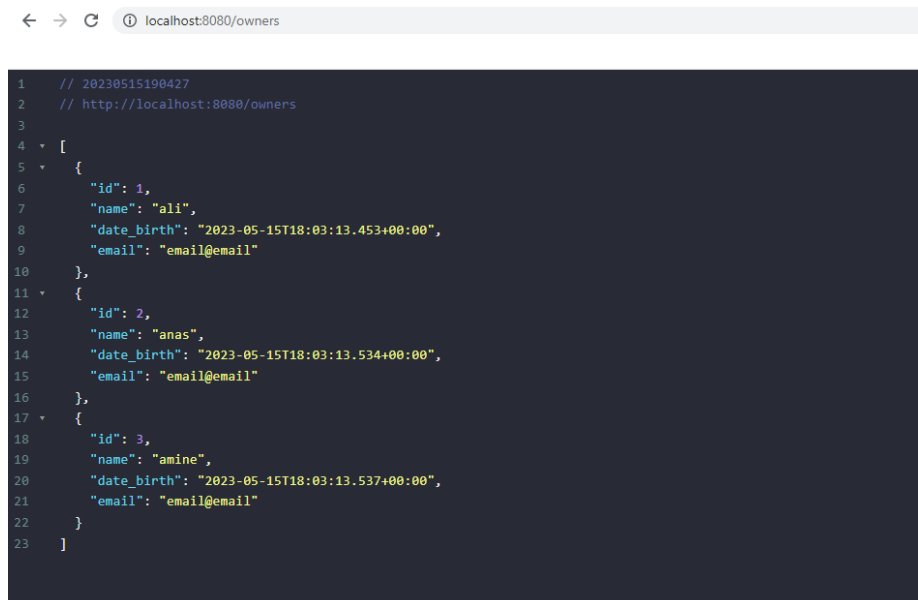


```

1 // 20230515190602
2 // http://localhost:8080/vehicules/3
3
4 {
5   "id": 3,
6   "matricule_number": 12345,
7   "model": "model ABC",
8   "marque": "marque ABC",
9   "power": 999,
10  "owner": {
11    "id": 3,
12    "name": "amine",
13    "date_birth": "2023-05-15T18:03:13.537+00:00",
14    "email": "email@email"
15  }
16 }
  
```

FIGURE 11

Afficher les propriétaires



```

1 // 20230515190427
2 // http://localhost:8080/owners
3
4 [
5   {
6     "id": 1,
7     "name": "ali",
8     "date_birth": "2023-05-15T18:03:13.453+00:00",
9     "email": "email@email"
10  },
11  {
12    "id": 2,
13    "name": "anas",
14    "date_birth": "2023-05-15T18:03:13.534+00:00",
15    "email": "email@email"
16  },
17  {
18    "id": 3,
19    "name": "amine",
20    "date_birth": "2023-05-15T18:03:13.537+00:00",
21    "email": "email@email"
22  }
23 ]
  
```

FIGURE 12

Afficher un propriétaire

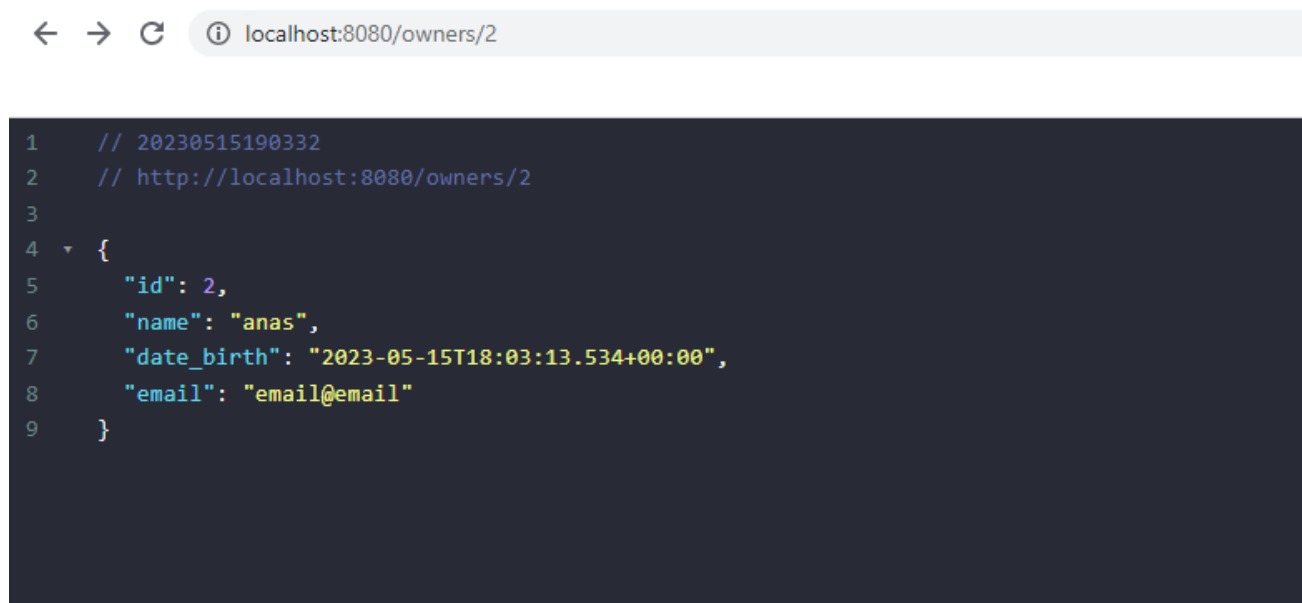


FIGURE 13

\implies Avec Postman

Modifier une véhicule

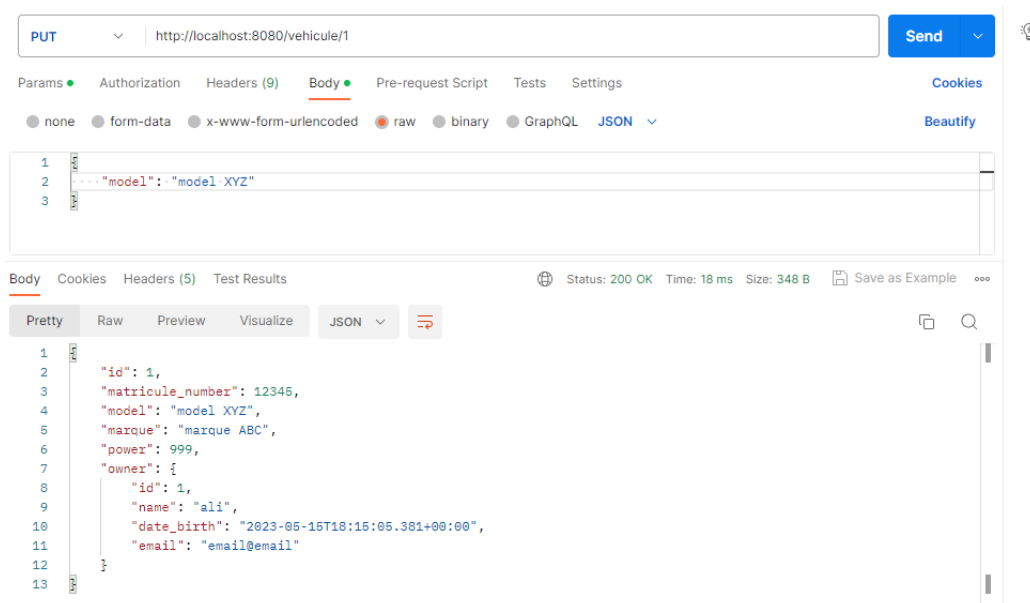


FIGURE 14

Suppression d'une véhicule

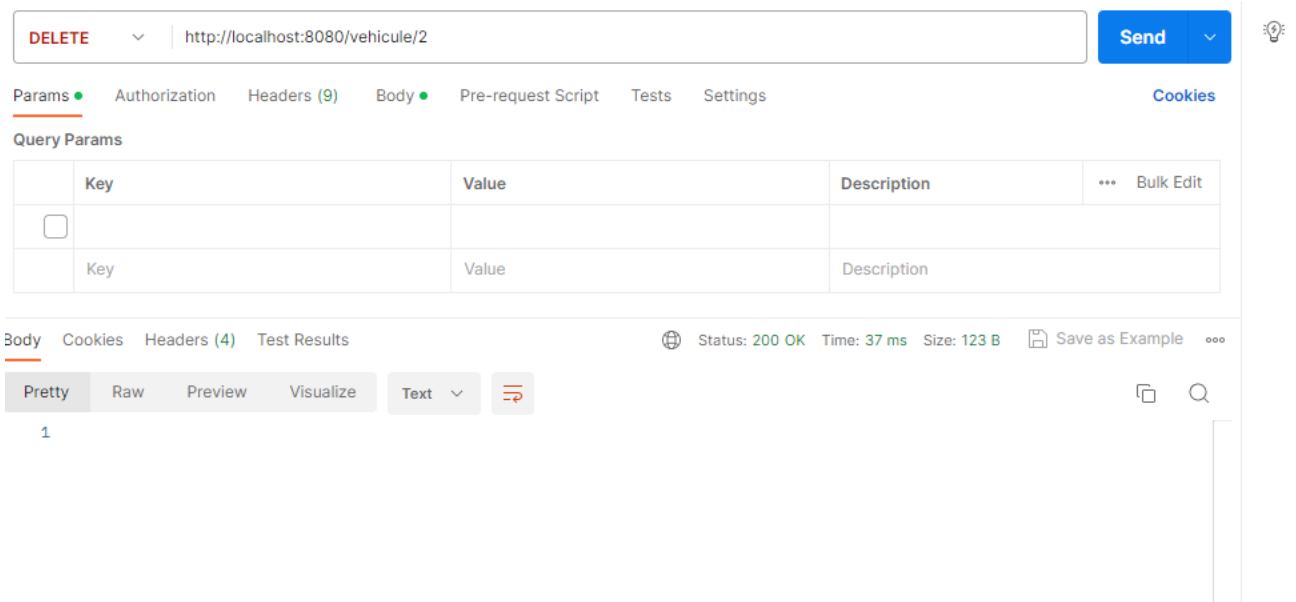


FIGURE 15

4.2.3 GRAPHQL service

Fichier schema.graphqls

```

type Query {
  vehiculesList:[Vehicule]
  vehiculeById(id:Float):Vehicule
  ownersList:[Owner]
  ownerById(id:Float):Owner
}
type Mutation {
  addVehicule(vehicule:VehiculeInput):Vehicule
  updateVehicule(id:Float,vehicule:VehiculeInput):Vehicule
  deleteVehicule(id:Float):String
  addOwner(owner:OwnerInput):Owner
  updateOwner(id:Float,owner:OwnerInput):Owner
  deleteOwner(id:Float):String
}
type Vehicule{
  id:Float
  matricule_number:Float
  model:String
  marque:String
  power:Float
  owner:Owner
}
    
```

FIGURE 16

```

input VehiculeInput{
  matricule_number:Float
  model:String
  marque:String
  power:Float
  owner:OwnerInput
}

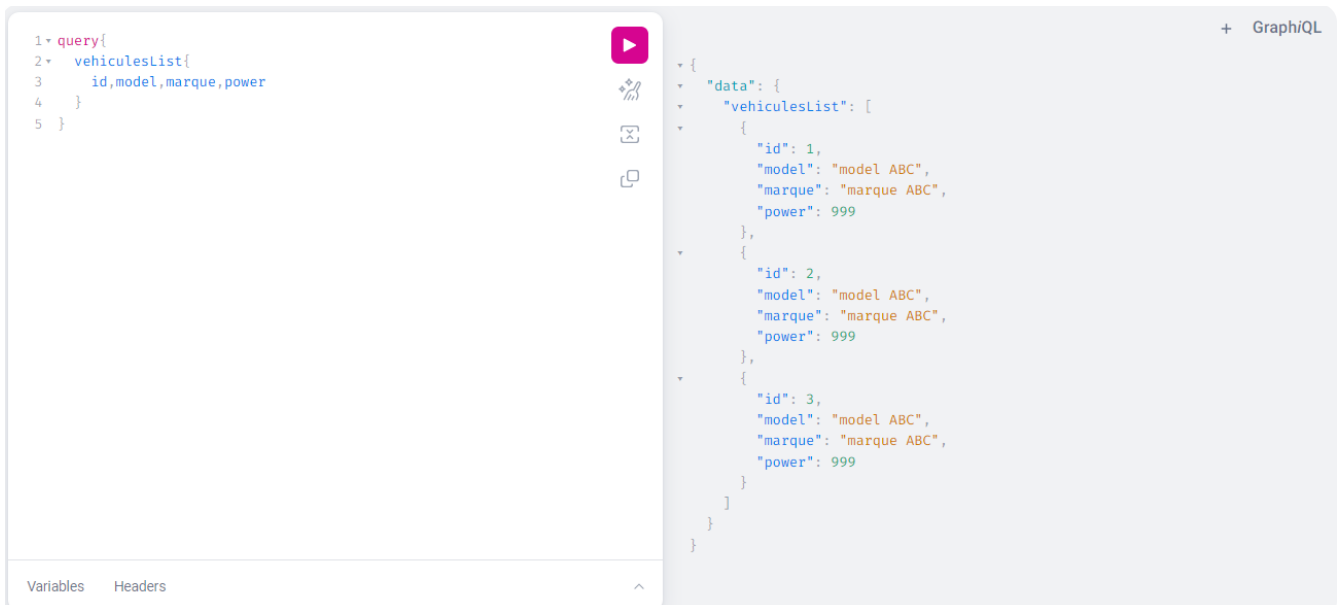
type Owner{
  id:Float
  name:String
  email:String
  date_birth:Float
  vehicules:[Vehicule]
}

input OwnerInput{
  name:String
  email:String
  date_birth:Float
}

```

FIGURE 17

Lister les véhicules



The screenshot shows the GraphQL Playground interface. On the left, the query editor contains the following query:

```

1 query{
2   vehiculesList{
3     id,model,marque,power
4   }
5 }

```

On the right, the JSON response is displayed:

```

{
  "data": {
    "vehiculesList": [
      {
        "id": 1,
        "model": "model ABC",
        "marque": "marque ABC",
        "power": 999
      },
      {
        "id": 2,
        "model": "model ABC",
        "marque": "marque ABC",
        "power": 999
      },
      {
        "id": 3,
        "model": "model ABC",
        "marque": "marque ABC",
        "power": 999
      }
    ]
  }
}

```

At the bottom left, there are tabs for 'Variables' and 'Headers'. At the top right, there is a '+ GraphQL' button.

FIGURE 18

Véhicule par id

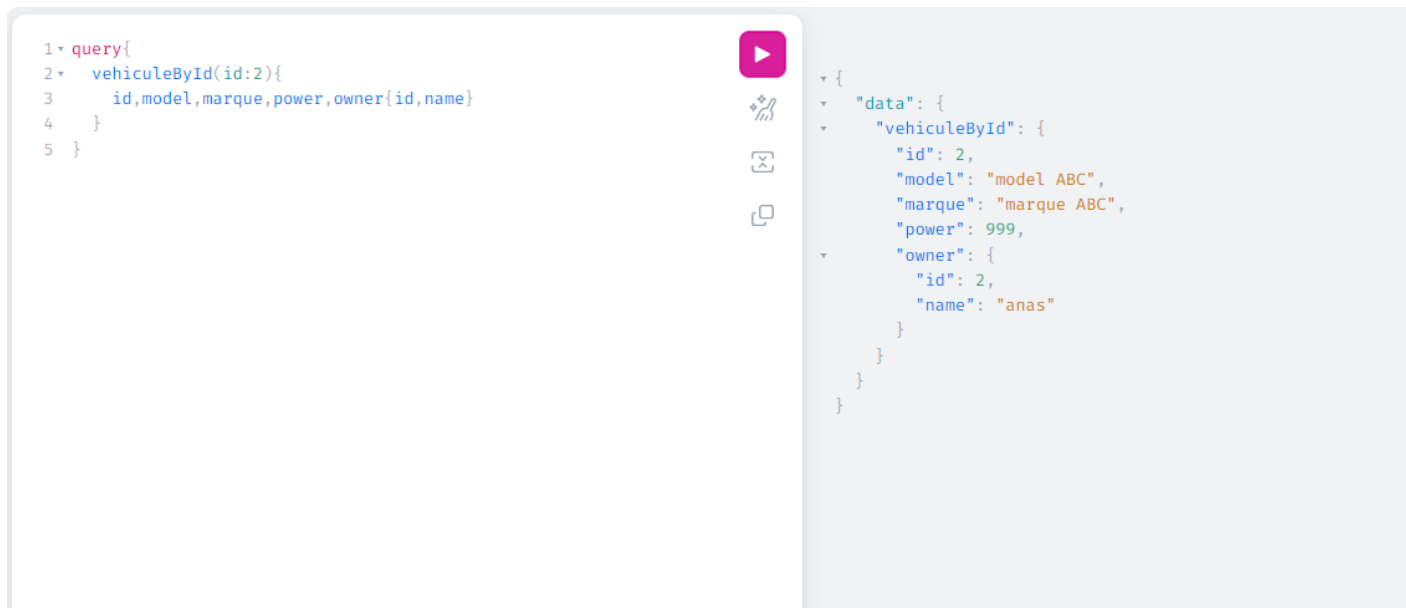


FIGURE 19

Modifier un owner



FIGURE 20

4.2.4 Soap service

Genaration du WSDL

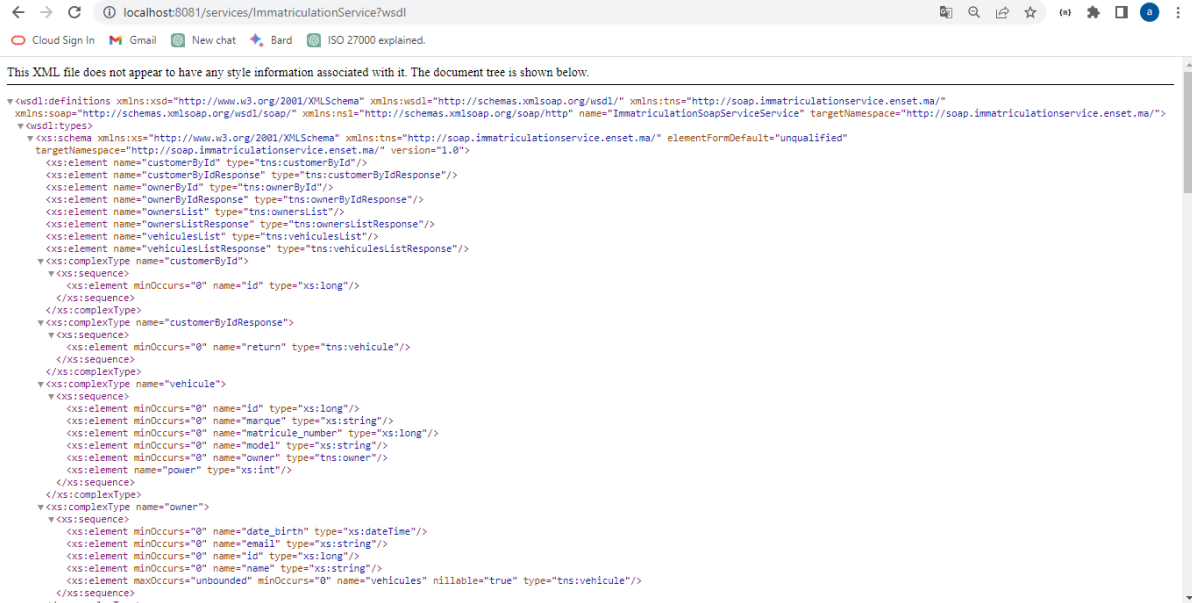


FIGURE 21

Genaration du WSDL

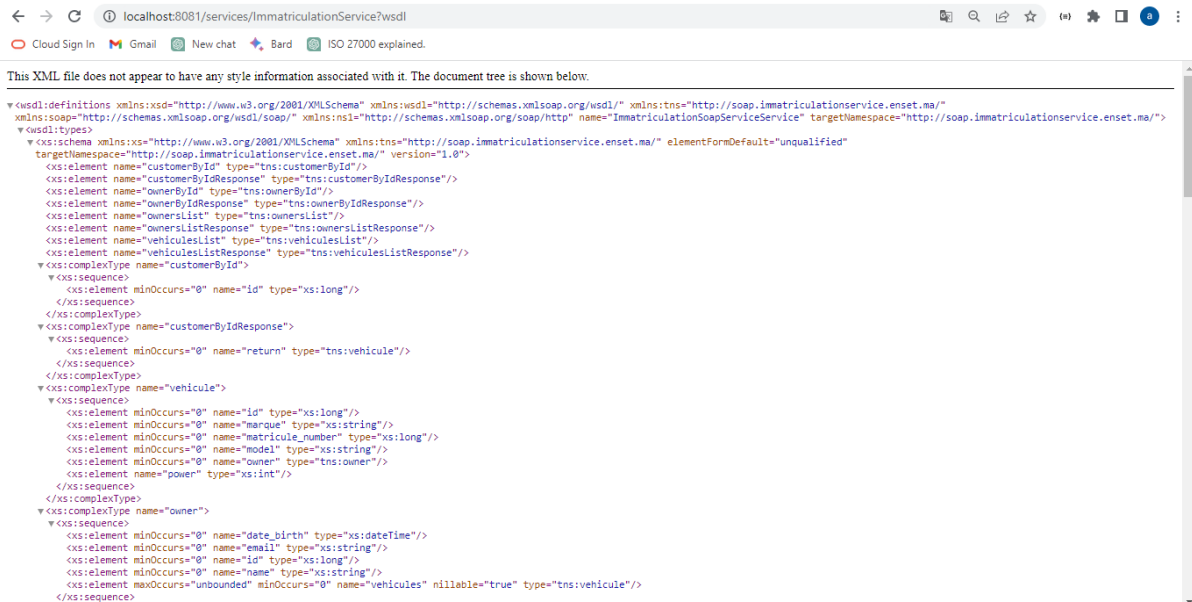


FIGURE 22

testing with SOAP UI

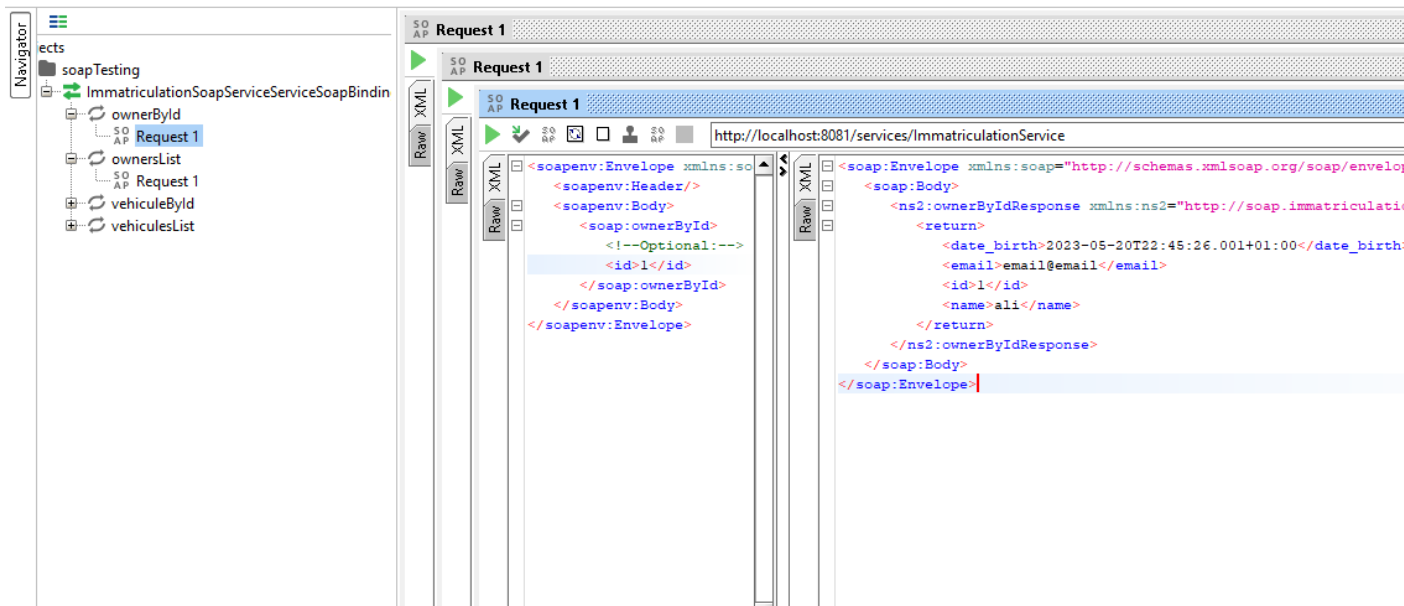


FIGURE 23

4.2.5 GRPC service

Fichier proto

```
syntax="proto3";
option java_package="ma.enset.immatriculationservice.grpc.stubs";

service ImmatriculationService{
  rpc getVehiculesList(ListRequest) returns (getVehiculesListResponse);
  rpc getOwnersList(ListRequest) returns (getOwnersListResponse);
  rpc getVehicule(id) returns (Vehicule);
  rpc getOwner(id) returns (Vehicule);
}

message ListRequest{}

message getVehiculesListResponse{
  repeated Vehicule vehicule=1;
}

message getOwnersListResponse{
  repeated Owner owner=1;
}

message id{
  int64 id=1;
}
```

FIGURE 24

```
message Vehicule{  
    int64 id=1;  
    int64 matricule_number=2;  
    string model=3;  
    string marque=4;  
    int64 power=5;  
    Owner owner=6;  
}  
message Owner{  
    int64 id=1;  
    string name=2;  
    string date_birth=3;  
    string email=4;  
    repeated Vehicule vehicules=5;  
}
```

FIGURE 25

Genaration du stubs

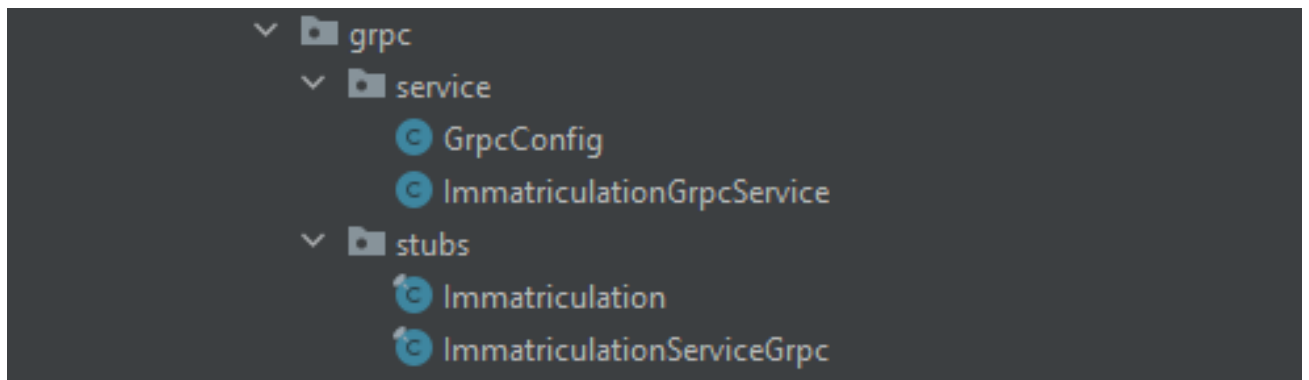


FIGURE 26

Test avec BloomRPC

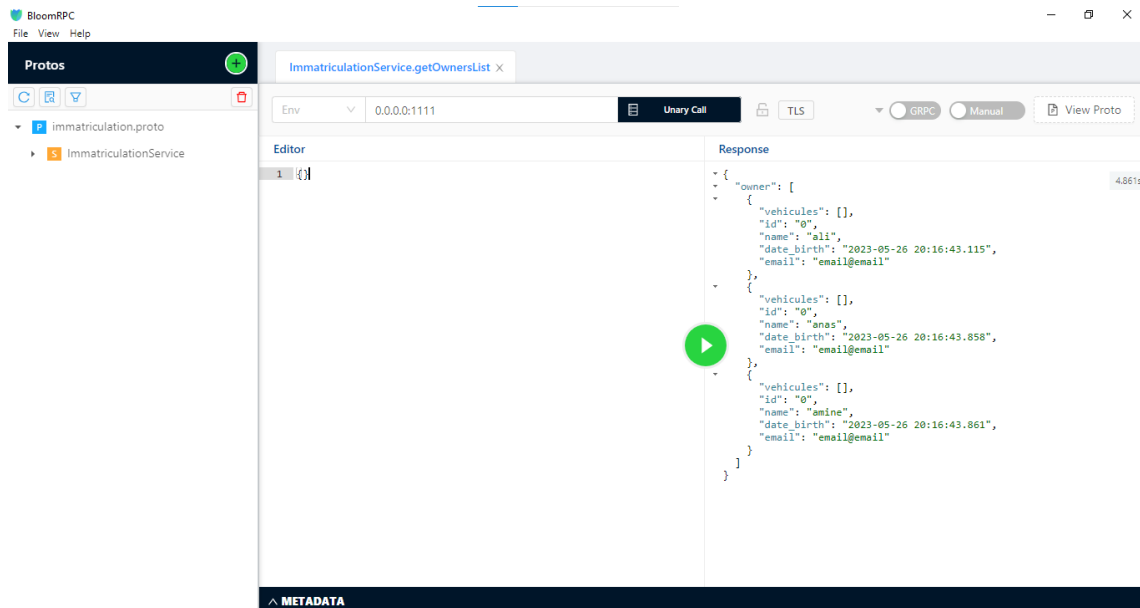


FIGURE 27

5 Développer le micro-service Infraction :

Le microservice Infraction est responsable de la gestion des violations au sein du système. Il gère les informations suivantes pour chaque violation :

- **ID de violation** : un identifiant unique pour chaque violation.
- **Date** : La date à laquelle la violation s'est produite.
- **Numéro de radar** : Le numéro d'identification du radar qui a détecté l'infraction.
- **Numéro d'immatriculation du véhicule** : Le numéro d'immatriculation du véhicule impliqué dans l'infraction.
- **Vitesse du véhicule** : La vitesse à laquelle le véhicule se déplaçait pendant l'infraction.
- **Limite de vitesse maximale du radar** : la limite de vitesse maximale définie par le radar.
- **Montant de l'amende** : le montant de l'amende imposée pour l'infraction.

5.1 Entité Infraction

```

package ma.enset.infractionservice.entities;

import ...

@Entity
@Data @AllArgsConstructor @NoArgsConstructor @Builder
public class Infraction {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date date;
    private Long radarNumber;
    private Long vehiculeMatricule;
    private Long montant;
    private int vehiculeSpeed;
    private int maxSpeedRadar;
}

```

FIGURE 28

5.2 Structure

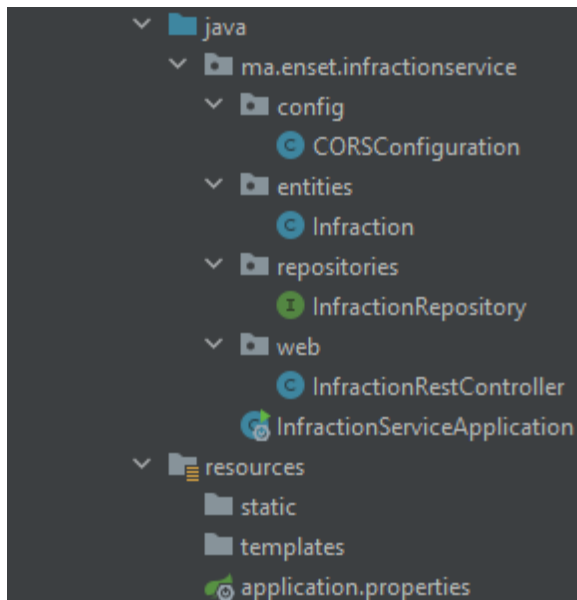


FIGURE 29

6 Développer le micro-service Radar :

Le Microservice Radar est responsable de la gestion des entités radar au sein du système. Il gère les informations suivantes pour chaque radar :

- **ID radar** : un identifiant unique pour chaque radar.
- **Limite de vitesse maximale** : La vitesse maximale autorisée pour les véhicules traversant la zone de surveillance du radar
- **Coordonnées** : Les coordonnées géographiques (longitude et latitude) de l'emplacement du radar.

6.1 Entité Radar

```

package ma.enset.radarservice.entities;

import ...

@Hanif-Ayoub
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @Builder @ToString
public class Radar {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    private int vitesseMax;
    private int longitude;
    private int latitude;
    private boolean state;
}

```

FIGURE 30

6.2 Structure

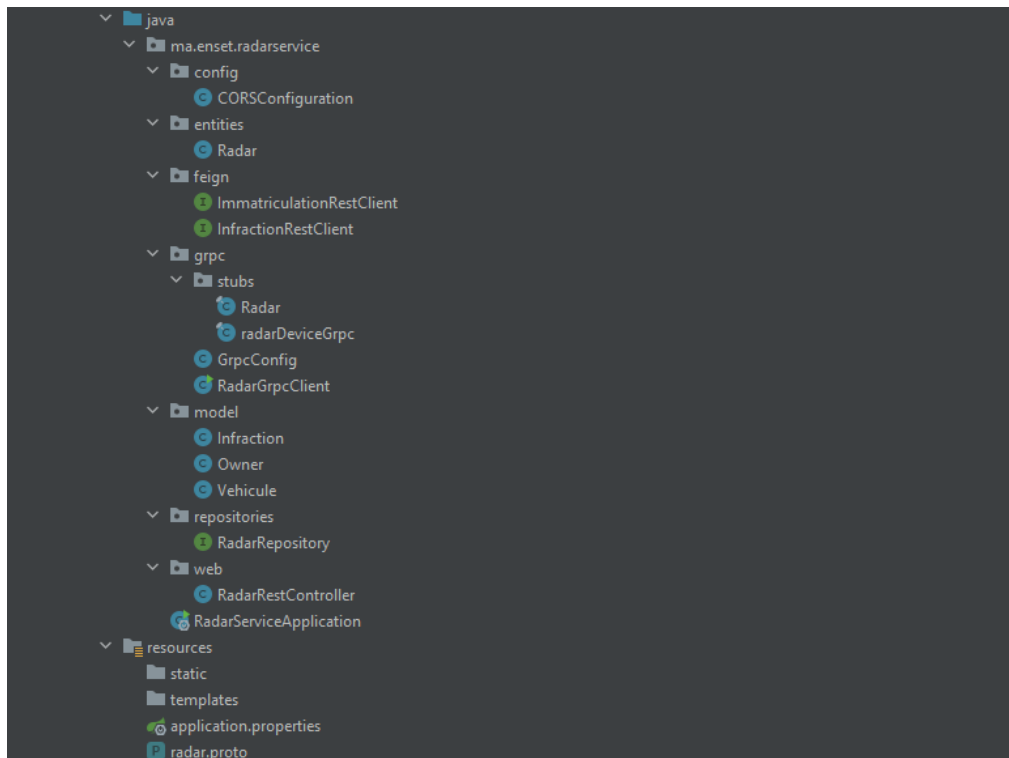


FIGURE 31

7 Créer un application java qui permet de simuler un radar

Une application Java qui simule un système radar générant des excès de vitesse aléatoires et les envoyant au Radar-Service.

7.1 Structure

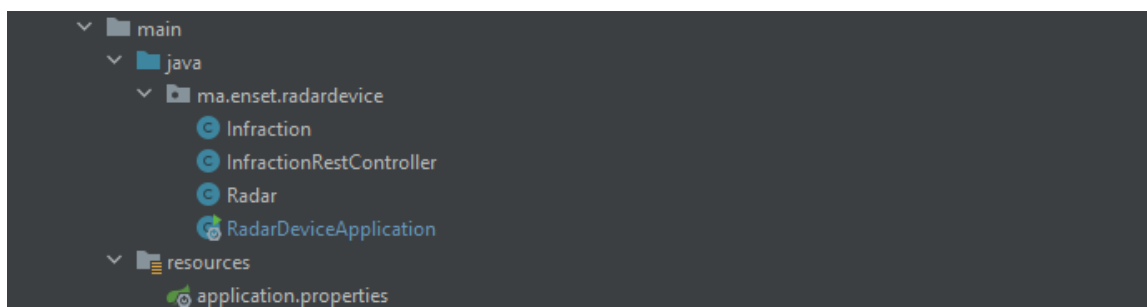


FIGURE 32

8 Eureka Discovery Service

composant côté serveur dans la pile OSS de Netflix qui permet aux services de s'enregistrer et de se découvrir dans une architecture de microservices.

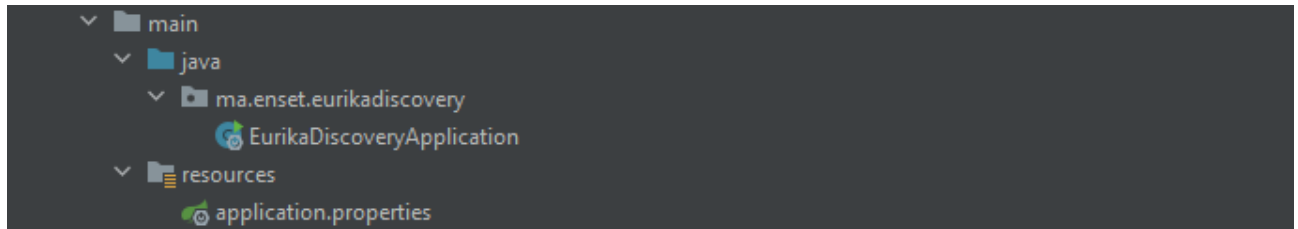


FIGURE 33

9 Gateway Service

Spring Cloud Gateway Il fournit un point d'entrée centralisé pour le routage et le filtrage des demandes vers des microservices dans un système distribué, permettant un routage dynamique et évolutif basé sur divers critères.

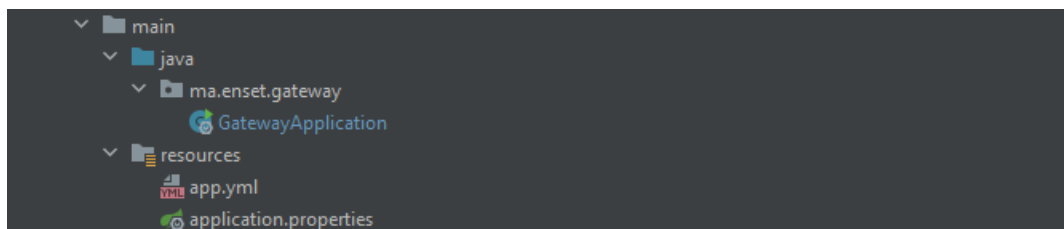


FIGURE 34

10 Frontend avec Angular

10.1 Login page

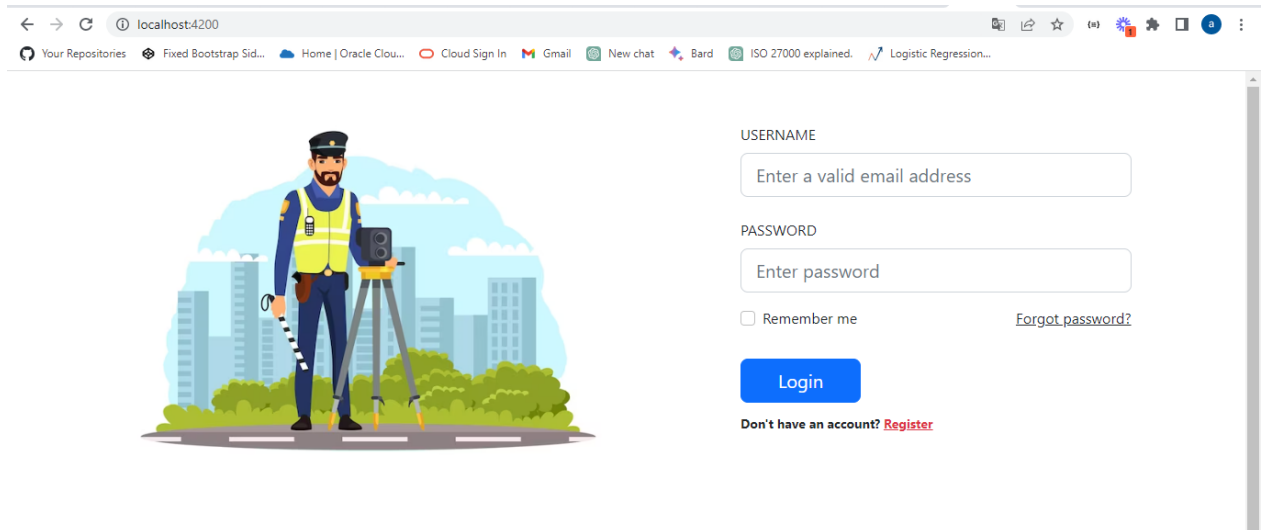


FIGURE 35

10.2 Dashbord page

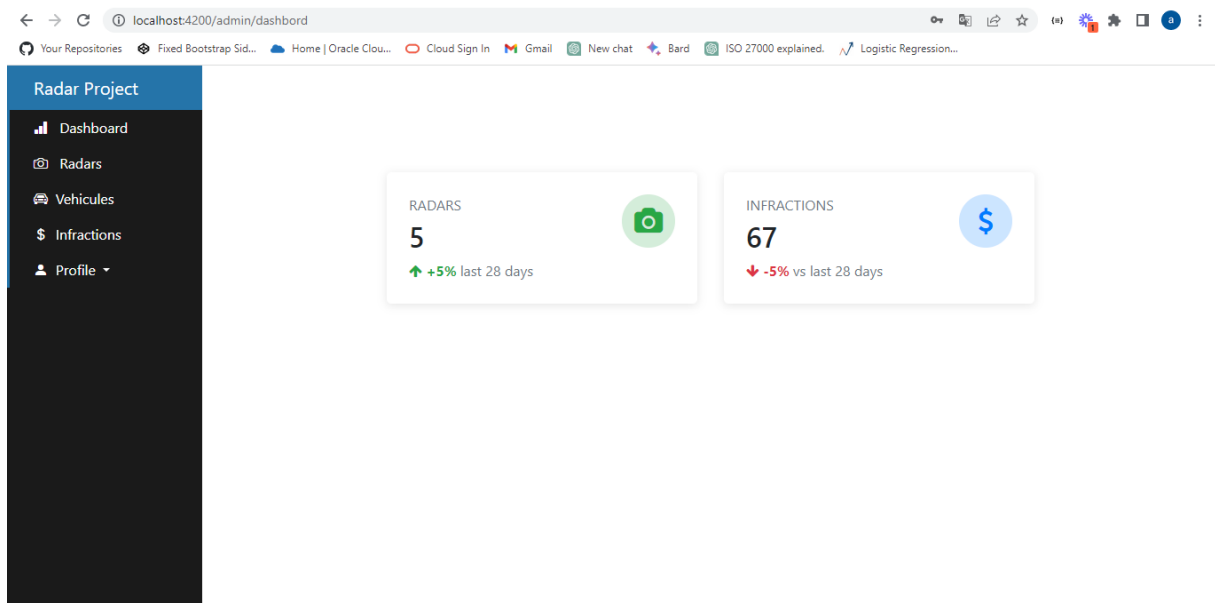


FIGURE 36

10.3 Liste des radars

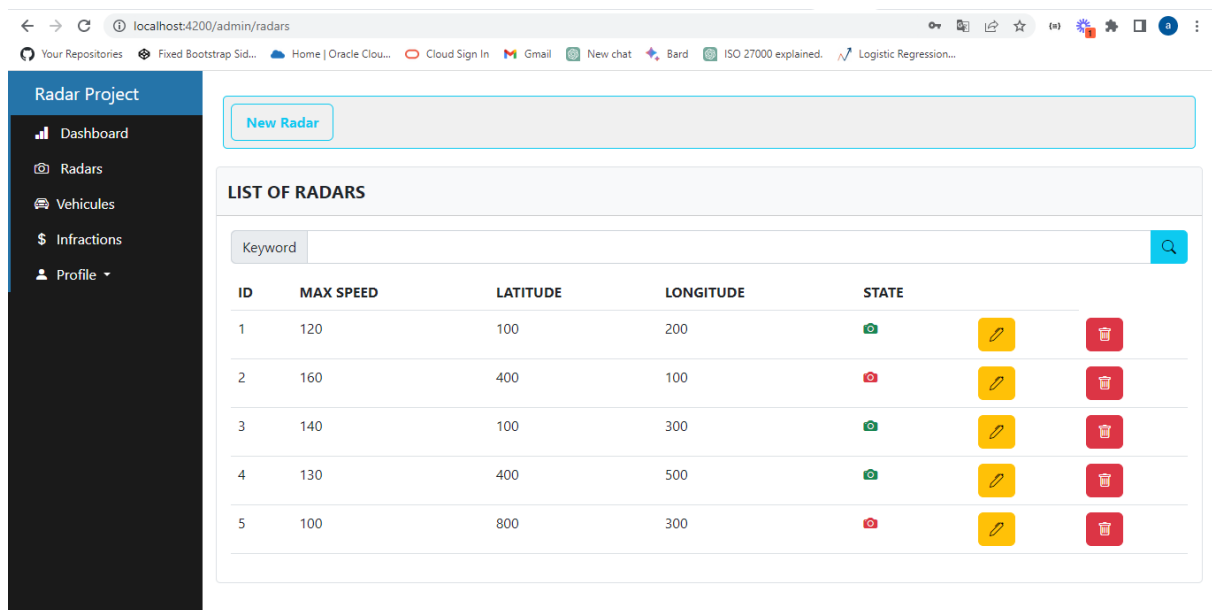


FIGURE 37

10.4 Ajouter radar

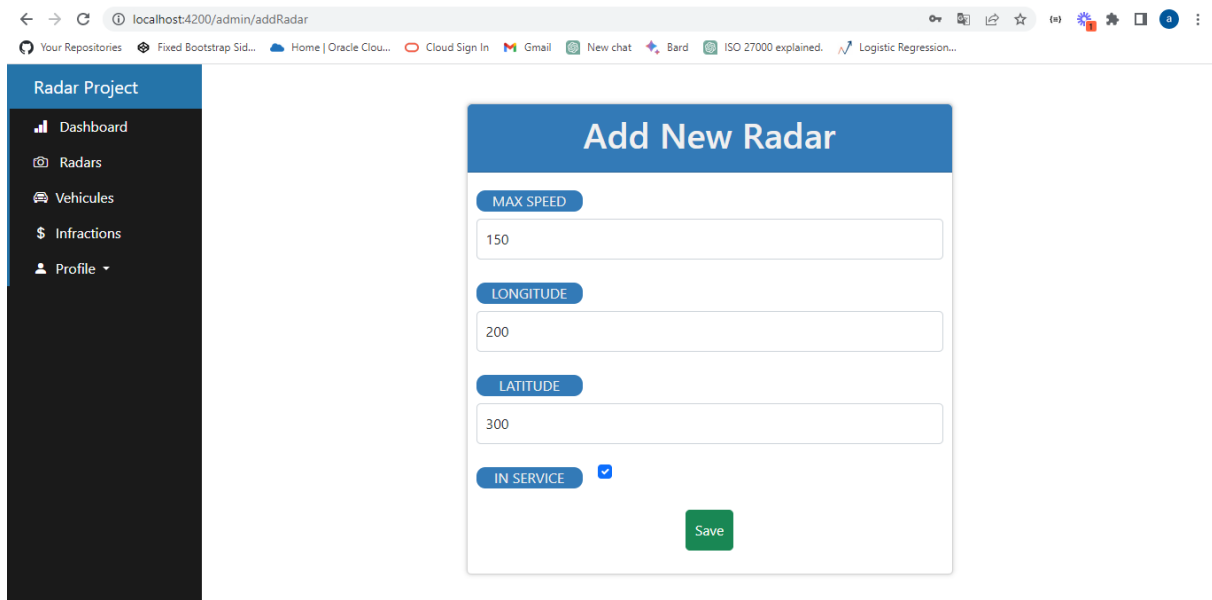


FIGURE 38

10.5 Modifier radar

The screenshot shows a web application interface for editing a radar. On the left is a sidebar with the 'Radar Project' menu, including links to Dashboard, Radars, Vehicules, Infractions, and Profile. The main content area is titled 'Editing Radar' and contains the following fields:

- ID:** A yellow button with the value '4'.
- vitesseMax:** A text input field with the value '130'.
- LONGITUDE:** A text input field with the value '500'.
- LATITUDE:** A text input field with the value '400'.
- IN SERVICE:** A checkbox that is checked.
- Save:** A yellow button at the bottom right.

FIGURE 39

10.6 Liste des infractions

The screenshot shows a web application interface for listing infractions. On the left is the same sidebar as in Figure 39. The main content area is titled 'LIST OF INFRACTIONS' and features a search bar with the placeholder 'Keyword'. Below the search bar is a table with the following data:

ID	DATE	VEHICULE_MATRICULE	VEHICULE_SPEED	MONTANT	
2	2023-05-27T00:21:07.325+00:00	32828	145	643	
4	2023-05-27T00:21:57.327+00:00	17298	133	985	
6	2023-05-27T00:23:20.743+00:00	82540	143	476	
8	2023-05-27T00:23:33.796+00:00	32828	126	394	
9	2023-05-27T00:23:35.797+00:00	17298	137	778	
10	2023-05-27T00:23:36.798+00:00	32828	114	184	

FIGURE 40

11 Conclusion

En conclusion, le projet a réussi à mettre en place un système distribué basé sur des micro-services pour gérer et automatiser le processus des infractions liées aux dépassements de vitesse. Les trois micro-services, à savoir la gestion des radars, l'immatriculation des véhicules et la gestion des infractions, ont été développés avec succès. Des technologies telles que REST, GraphQL, SOAP et gRPC ont été utilisées pour les web services, permettant une communication flexible entre les micro-services. De plus, une application frontend a été développée avec Angular pour offrir une interface utilisateur conviviale.