Systèmes Distribué

COMPTE RENDU DE L'EXAMEN

Pr. Mohammed YOUSSFI.

Réalisé par :

- **FAJRI WALID**

Master Intelligence Artificiel et Analyse des données.

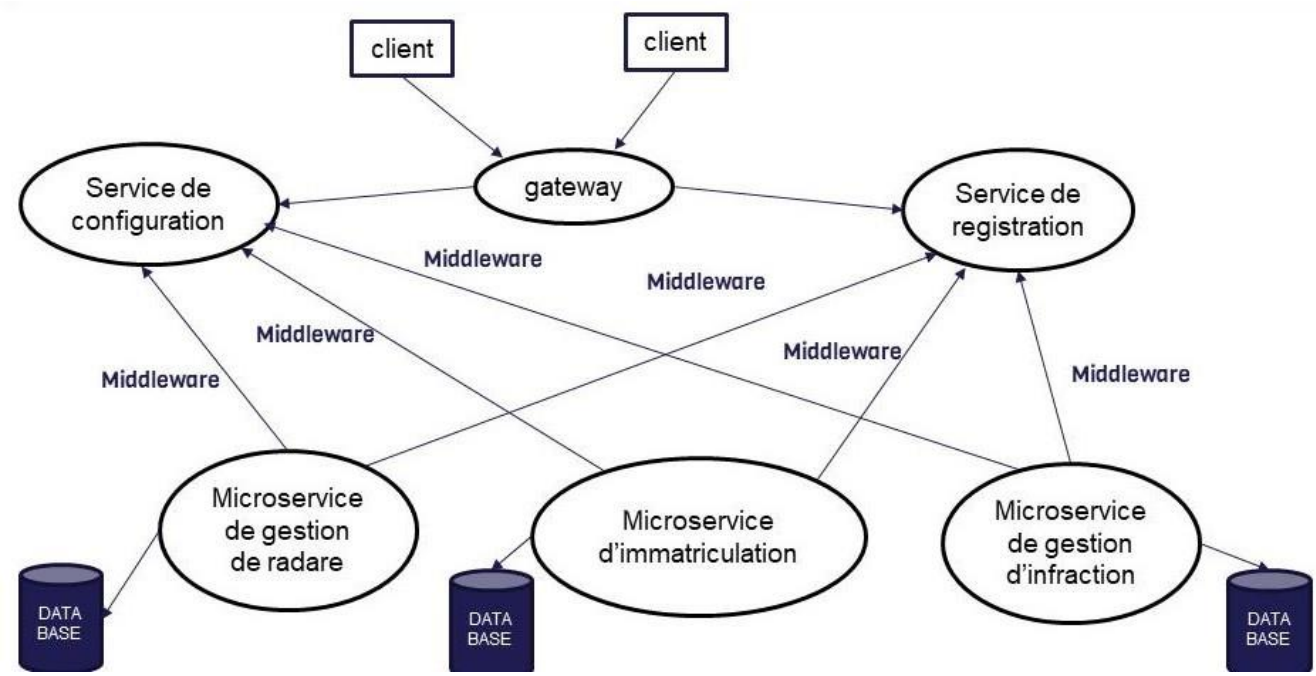
Objectif :

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 dépassements de vitesses détectés par des radars automatiques. Le système se compose de trois micro-services :

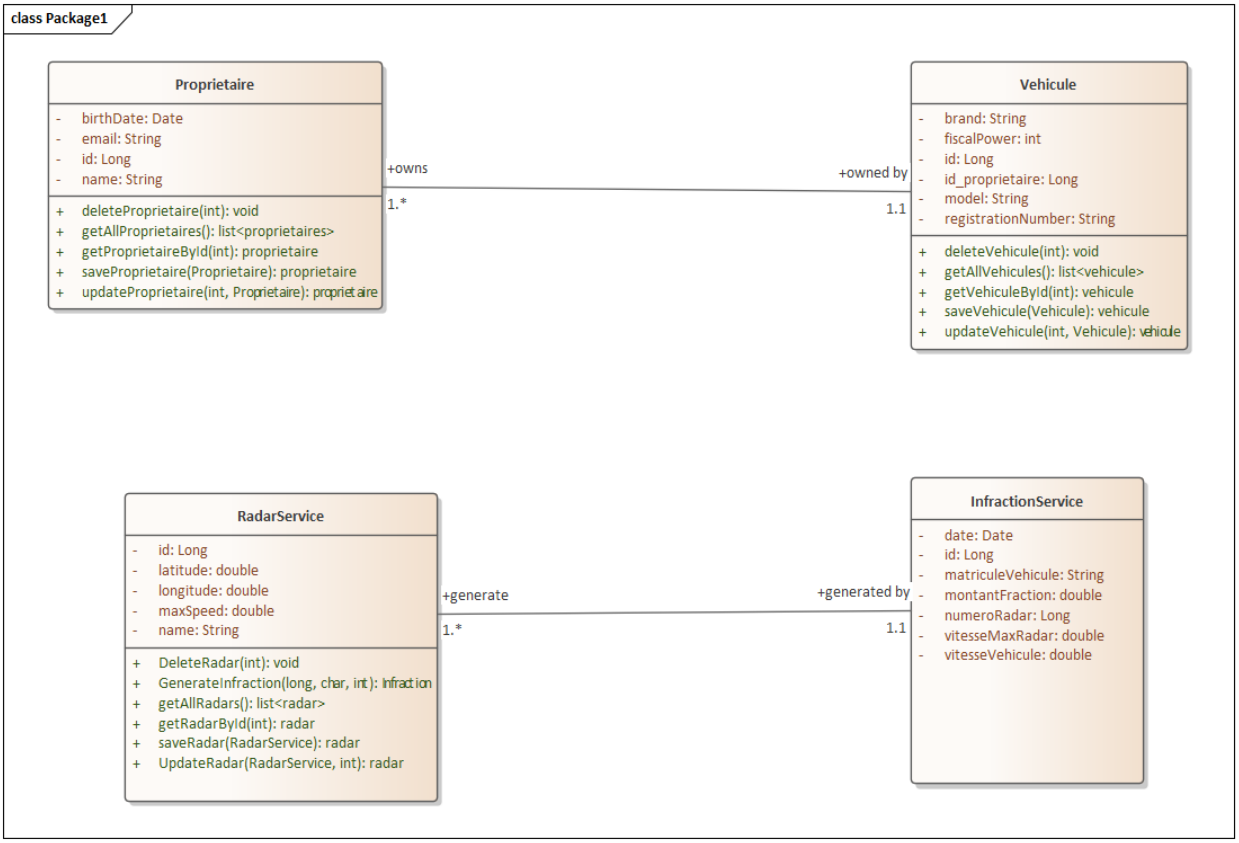
- Le micro-service qui permet de gérer les radars. Chaque radar est défini par son id, sa vitesse maximale, des coordonnées : Longitude et Latitude.
- Le micro-service d'immatriculation qui permet de gérer des véhicules appartenant des propriétaires. Chaque véhicule appartient à un seul propriétaire. 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.

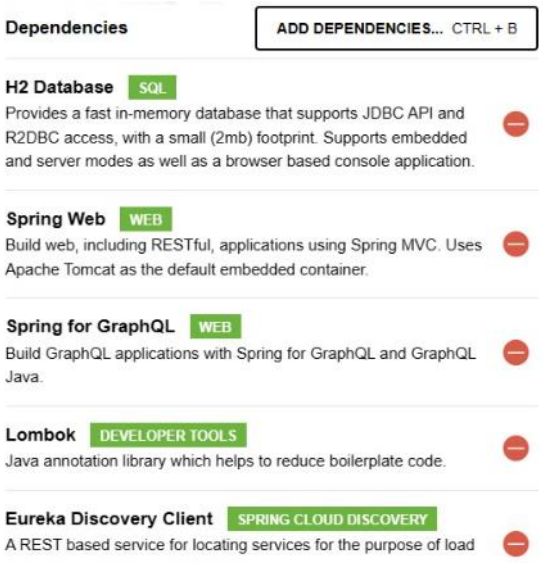
1. Établir une architecture technique du projet :

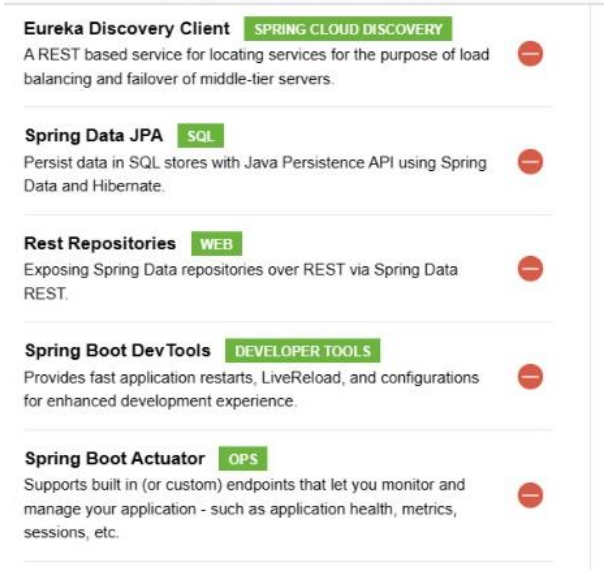


2. Établir un diagramme de classe global du projet :



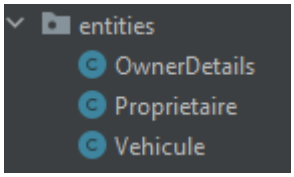
On crée un projet spring à partir de spring starter : en ajoutant les dependances lombok, spring bootdevtools,spring web , spring data jpa, h2 database et spring for graphql .





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

On commence par la création des entités Propriétaire et Vehicule dans un package Entités :



L'entité Véhicule :

```
package com.example.immatriculationservice.entities;


import ...


@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Vehicule {
    1 usage
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String registrationNumber;
    1 usage
    private String brand;
    1 usage
    private String model;
    1 usage
    private int fiscalPower;
    1 usage
```

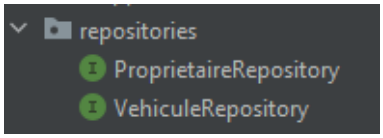
L’entité Propriétaire :

```java
package com.example.immatriculationservice.entities;

import ...

@Entity
@Data @NoArgsConstructor @AllArgsConstructor  @Builder
public class Proprietaire {
    1 usage
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String name;
    1 usage
    private LocalDate birthDate;
    1 usage
    private String email;
    1 usage
    @OneToMany(mappedBy = "proprietaire", fetch = FetchType.EAGER)
    @JsonIgnore
    private Collection<Vehicule> vehicules;
    //
```

L’entité Owner Details :

```java
package com.example.immatriculationservice.entities;

import ...

1 usage
@Data
@NoArgsConstructor
@AllArgsConstructor
public class OwnerDetails {
    no usages
    private String name;
    no usages
    private String email;


}
```

Maintenant dans un package Repositories, on va créer les deux interfaces VehiculeRepository et ProprietaireRepository. :



VehiculeRepository :

```java
package com.example.immatriculationservice.repositories;

import ...

9 usages
@RepositoryRestResource
public interface VehiculeRepository extends JpaRepository<Vehicule,Long> {
    1 usage
    public Collection<Vehicule> getVehiculeByProprietaireId(long id);
    //Vehicule findByMatricule(String matricule);


    //used in gRPC
    1 usage
    @Query("SELECT v FROM Vehicule v WHERE v.id LIKE ?1")
    Vehicule selectById(Long id);

    1 usage
    @Query("SELECT v.proprietaire.id FROM Vehicule v WHERE v.id LIKE ?1")
    long selectProprietaireByIdVehicule(long id);

    1 usage
```

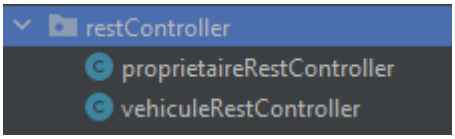
ProprietaireRepository :

```java
package com.example.immatriculationservice.repositories;

import com.example.immatriculationservice.entities.Proprietaire;
import com.example.immatriculationservice.entities.Vehicule;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

12 usages
@RepositoryRestResource
public interface ProprietaireRepository extends JpaRepository<Proprietaire,Long> {

}
```

3. Les 4 web services REST, GraphQL, SOAP et GRPC:

b. Pour le web service REST :

Dans un Package appeler Web, nous allons créer les classes suivantes :



VehiculeRestController :

```
package com.example.immatriculationservice.web.restController;

import ...

no usages
@RestController
@AllArgsConstructor
@CrossOrigin(origins = "*", allowedHeaders = "*", methods = {RequestMethod.GET, RequestMethod.POST, RequestMethod
public class vehiculeRestController {
    13 usages
    private VehiculeRepository vehiculeRepo;
    1 usage
    private ProprietaireRepository proprietaireRepository;

    no usages
    @GetMapping("/vehicules")
    public List<Vehicule> getAllVehicules() { return vehiculeRepo.findAll(); }

    no usages
    @GetMapping("/vehicules/{id}")
    public Vehicule getVehiculeById(@PathVariable Long id) { return vehiculeRepo.findById(id).orElse( other: null);

    no usages
    @GetMapping("/vehicules/matricules/{matricule}") // hadiii
    public ResponseEntity<Vehicule> findByMatricule(@PathVariable String matricule){
```

ProprietaireRestController :

```java
package com.example.immatriculationservice.web.restController;

import ...

no usages
@RestController
@AllArgsConstructor
@CrossOrigin(origins = "http://localhost:4200")
public class proprietaireRestController {
    5 usages
    private ProprietaireRepository proprietaireRepo;


    no usages
    @GetMapping("/proprietaires")
    public List<Proprietaire> getAllProprietaires() { return proprietaireRepo.findAll(); }

    no usages
    @GetMapping("/proprietaires/{id}")
    public Proprietaire getProprietaireById(@PathVariable Long id) {
        return proprietaireRepo.findById(id).orElse( other: null);
    }

    no usages
    @PostMapping("/proprietaires")
    public Proprietaire saveProprietaire(@RequestBody Proprietaire proprietaire) {
```

Dans un Package Appelé Service, on va créer la classe exceptionHandler:

```java
package com.example.immatriculationservice.service;

import graphql.ErrorClassification;
import graphql.GraphQLError;
import graphql.language.SourceLocation;
import graphql.schema.DataFetchingEnvironment;
import org.springframework.graphql.execution.DataFetcherExceptionResolverAdapter;
import org.springframework.stereotype.Component;

import java.util.List;
no usages
@Component
public class exceptionHandler extends DataFetcherExceptionResolverAdapter {
    no usages
    @Override
    protected GraphQLError resolveToSingleError(Throwable ex, DataFetchingEnvironment env) {
        return new GraphQLError() {
            @Override
            public String getMessage() { return ex.getMessage(); }

            no usages
            @Override
            public List<SourceLocation> getLocations() { return null; }

            no usages
            @Override
```

Dans un Package Appelé DTO, on va créer la classe suivante :

```java
package com.example.immatriculationservice.dto;

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

6 usages
@ Data
@NoArgsConstructor @AllArgsConstructor
public class proprietaireRequest{
    no usages
    private String name;
    no usages
    private String email;
}
```

Pour faciliter le transfert des objets entre les différents couches du projet, nous allons créer les mapper pour chacune de nos entités dans un package appelé mappers:

ProprietaireMapper :

```java
package com.example.immatriculationservice.mapper;

import com.example.immatriculationservice.dto.proprietaireRequest;
import com.example.immatriculationservice.entities.Proprietaire;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

4 usages
@Component
public class ProprietaireMapper {
    no usages
    private  ModelMapper modelMapper;
    2 usages
    public Proprietaire from(proprietaireRequest request){
        Proprietaire proprietaire=new Proprietaire();
        proprietaire.setName(request.getName());
        proprietaire.setEmail(request.getEmail());
        return proprietaire;
    }


}
```

c. Pour le web service GraphQL :

VehiculeGraphQlController :

```
package com.example.immatriculationservice.web.graphqlController;


import ...

//@Controller
no usages
@AllArgsConstructor
public class vehiculeGraphQLController {
    6 usages
    private VehiculeRepository vehiculeRepo;

    no usages
    @QueryMapping
    public List<Vehicule> getAllVehicules() { return vehiculeRepo.findAll(); }

    no usages
    @QueryMapping
    public Vehicule getVehiculeById(@Argument Long id) { return vehiculeRepo.findById(id).orElse( other: null); }

    no usages
    @MutationMapping
    public Vehicule saveVehicule(@Argument Vehicule vehicule) { return vehiculeRepo.save(vehicule); }

    no usages
```

ProprietaireGraphQlController :

```
package com.example.immatriculationservice.web.graphqlController;


import ...

//@Controller
no usages
@AllArgsConstructor
public class proprietaireGraphQLController {
    1 usage
    private ProprietaireMapper proprietaireMapper;
    6 usages
    private ProprietaireRepository proprietaireRepo;

    //retourner une liste des proprietaires
    no usages
    @QueryMapping
    public List<Proprietaire> getAllProprietaires() { return proprietaireRepo.findAll(); }

    no usages
    @QueryMapping
    public Proprietaire getProprietaireById(@Argument Long id) {
        Proprietaire proprietaire=proprietaireRepo.findById(id).orElse( other: null);
        if(proprietaire==null) throw new RuntimeException(String.format("Owner %s not found",id));
        return proprietaire;
    }
```

Shéma de QraphQl :

```
type Query {
    getAllVehicules: [Vehicule],
    getVehiculeById(id : Int): Vehicule,
    getAllProprietaires: [Proprietaire],
    getProprietaireById(id : Int): Proprietaire
}


type Mutation {
    saveVehicule(vehicule : VehiculeType):Vehicule
    updateVehicule(vehicule : VehiculeType, id : Int):Vehicule
    deleteVehicule(id : Int):String

    saveProprietaire(proprietaire : ProprietaireType):Proprietaire
    updateProprietaire(proprietaire : ProprietaireType, id : Int):Proprietaire
    deleteProprietaire(id : Int):String
}

type Vehicule {
    id : Int,
    numMatricule : String,
    marque : String,
    puissanceFiscale : String,
```

d. Pour le web service GRPC :

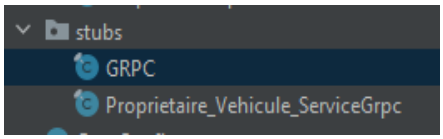
Dans un package appelé GRPC, on va créer deux packages appelés Stubs et SERVICE :

Pour le package Service : on crée la classe suivante ;

La classe ProprietaireGrpc :

```
package com.example.immatriculationservice.grpc.services;                    ⚠ 2  ✓ 19

import ...


//@GrpcService
no usages
@AllArgsConstructor
public class ProprietaireGrpc extends Proprietaire_Vehicule_ServiceGrpc.Proprietaire_Vehicule_ServiceImplBase
    3 usages
    private final ProprietaireRepository proprietaireRepository;
    4 usages
    private final VehiculeRepository vehiculeRepository;

    1 usage
    @Override
    public void getProprietaireById(GRPC.Proprietaireid request, StreamObserver<GRPC.Proprietaire> responseObs
        Long id=request.getId();
        Proprietaire proprietaire=proprietaireRepository.findById(id).orElseThrow( exceptionSupplier: null);
        if (proprietaire!=null) {
            GRPC.Proprietaire proprietaireResponse = GRPC.Proprietaire.newBuilder()
                    .setId(Math.toIntExact(proprietaire.getId()))
                    .setNom(proprietaire.getName())
                    .setEmail(proprietaire.getEmail())
                    .build();
            responseObserver.onNext(proprietaireResponse);
```

Dans le package Stub :



Dans GRPC :

```
/.../

package com.example.immatriculationservice.grpc.stubs;

669 usages
public final class GRPC {
  no usages
  private GRPC() {}
  1 usage
  public static void registerAllExtensions(
      com.google.protobuf.ExtensionRegistryLite registry) {
  }

  no usages
  public static void registerAllExtensions(
      com.google.protobuf.ExtensionRegistry registry) {
    registerAllExtensions(
        (com.google.protobuf.ExtensionRegistryLite) registry);
  }
  11 usages  2 implementations
  public interface VehiculeOrBuilder extends
      // @@protoc_insertion_point(interface_extends:Vehicule)
      com.google.protobuf.MessageOrBuilder {

    /**
```

Dans GRPC :

```
package com.example.immatriculationservice.grpc.stubs;

import static io.grpc.MethodDescriptor.generateFullMethodName;
import static io.grpc.stub.ClientCalls.asyncBidiStreamingCall;
import static io.grpc.stub.ClientCalls.asyncClientStreamingCall;
import static io.grpc.stub.ClientCalls.asyncServerStreamingCall;
import static io.grpc.stub.ClientCalls.asyncUnaryCall;
import static io.grpc.stub.ClientCalls.blockingServerStreamingCall;
import static io.grpc.stub.ClientCalls.blockingUnaryCall;
import static io.grpc.stub.ClientCalls.futureUnaryCall;
import static io.grpc.stub.ServerCalls.asyncBidiStreamingCall;
import static io.grpc.stub.ServerCalls.asyncClientStreamingCall;
import static io.grpc.stub.ServerCalls.asyncServerStreamingCall;
import static io.grpc.stub.ServerCalls.asyncUnaryCall;
import static io.grpc.stub.ServerCalls.asyncUnimplementedStreamingCall;
import static io.grpc.stub.ServerCalls.asyncUnimplementedUnaryCall;

23 usages
@javax.annotation.Generated(
    value = "by gRPC proto compiler (version 1.15.0)",
    comments = "Source: gRPC.proto")
public final class Proprietaire_Vehicule_ServiceGrpc {
```

```
private Proprietaire_Vehicule_ServiceGrpc() {}                                    ⚠ 8 ^

6 usages
public static final String SERVICE_NAME = "Proprietaire_Vehicule_Service";

// Static method descriptors that strictly reflect the proto.
3 usages
private static volatile io.grpc.MethodDescriptor<com.example.immatriculationservice.grpc.stubs.GRPC.P
    com.example.immatriculationservice.grpc.stubs.GRPC.Proprietaire> getGetProprietaireByIdMethod;

6 usages
@io.grpc.stub.annotations.RpcMethod(
    fullMethodName = SERVICE_NAME + '/' + "getProprietaireById",
    requestType = com.example.immatriculationservice.grpc.stubs.GRPC.Proprietaireid.class,
    responseType = com.example.immatriculationservice.grpc.stubs.GRPC.Proprietaire.class,
    methodType = io.grpc.MethodDescriptor.MethodType.UNARY)
public static io.grpc.MethodDescriptor<com.example.immatriculationservice.grpc.stubs.GRPC.Proprietair
    com.example.immatriculationservice.grpc.stubs.GRPC.Proprietaire> getGetProprietaireByIdMethod() {
  io.grpc.MethodDescriptor<com.example.immatriculationservice.grpc.stubs.GRPC.Proprietaireid, com.exa
  if ((getGetProprietaireByIdMethod = Proprietaire_Vehicule_ServiceGrpc.getGetProprietaireByIdMethod)
    synchronized (Proprietaire_Vehicule_ServiceGrpc.class) {
      if ((getGetProprietaireByIdMethod = Proprietaire_Vehicule_ServiceGrpc.getGetProprietaireByIdMet
        Proprietaire_Vehicule_ServiceGrpc.getGetProprietaireByIdMethod = getGetProprietaireByIdMethod
            io.grpc.MethodDescriptor.<~>newBuilder()
            .setType(io.grpc.MethodDescriptor.MethodType.UNARY)
            .setFullMethodName(generateFullMethodName(
```

GrpcConfig :

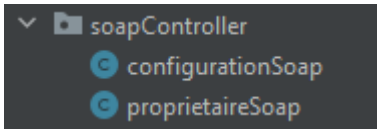
```
package com.example.immatriculationservice.grpc;


import ...

no usages
@Configuration
@ImportAutoConfiguration({
        net.devh.boot.grpc.client.autoconfigure.GrpcClientAutoConfiguration.class,
        net.devh.boot.grpc.client.autoconfigure.GrpcClientMetricAutoConfiguration.class,
        net.devh.boot.grpc.client.autoconfigure.GrpcClientHealthAutoConfiguration.class,
        net.devh.boot.grpc.client.autoconfigure.GrpcClientSecurityAutoConfiguration.class,
        net.devh.boot.grpc.client.autoconfigure.GrpcClientTraceAutoConfiguration.class,
        net.devh.boot.grpc.client.autoconfigure.GrpcDiscoveryClientAutoConfiguration.class,

        net.devh.boot.grpc.common.autoconfigure.GrpcCommonCodecAutoConfiguration.class,
        net.devh.boot.grpc.common.autoconfigure.GrpcCommonTraceAutoConfiguration.class,

        net.devh.boot.grpc.server.autoconfigure.GrpcAdviceAutoConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcHealthServiceAutoConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcMetadataConsulConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcMetadataEurekaConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcMetadataNacosConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcMetadataZookeeperConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcReflectionServiceAutoConfiguration.class,
        net.devh.boot.grpc.server.autoconfigure.GrpcServerAutoConfiguration.class,
```

Grpc.proto :

```
syntax ="proto3";
option  java_package="com.example.immatriculationservice.grpc.stubs"; // ou put file copiler
message Vehicule{
  int32 id=1;
  string nom=2;
  string matricule=3;
  string marque =4;
  int32 puissFiscal=5;
  string modele=6;
  Proprietaire Proprietaire=7;
}
message VehiculeId{
  int32 id=1;
}
message Proprietaire{
  int32 id=1;
  string nom=2;
  string birthDate=3;
  string email=4;
}
message Proprietaireid{
  int64 id=1;
}
message GetVehiculeListRequest{}
```

e. Pour le web service SOAP :

Dans le package Web on a créé autre package qui s'appelle soap Controller qui contient deux Classes :



ConfigurationSoap :

```java
package com.example.immatriculationservice.web.soapController;

import ...

//@Configuration
no usages
@AllArgsConstructor
public class configurationSoap {
    1 usage
    private Bus bus;
    1 usage
    private proprietaireSoap proprietaireSoap;
    no usages
    @Bean
    public EndpointImpl endpoint(){
        EndpointImpl endpoint=new EndpointImpl(bus,proprietaireSoap);
        endpoint.publish( addr: "/ProprietaireService");
        return endpoint;
    }
}
```

ProprietaireSoap :

```java
package com.example.immatriculationservice.web.soapController;

import ...

//@Component
1 usage
@AllArgsConstructor
@CrossOrigin(origins = "http://localhost:4200/")
public class proprietaireSoap {
    3 usages
    private ProprietaireRepository proprietaireRepository;
    1 usage
    private ProprietaireMapper mapper;
    no usages
    @WebMethod
    public List<Proprietaire> proprietaireList() { return proprietaireRepository.findAll(); }
    //return owner
    no usages
    @WebMethod
    public Proprietaire proprietaireById(@WebParam(name = "id") Long id){
        return proprietaireRepository.findById(id).orElse( other: null);
    }
    //savemethod
```

f. Tester les 4 web services:

On commence par créer les objets :

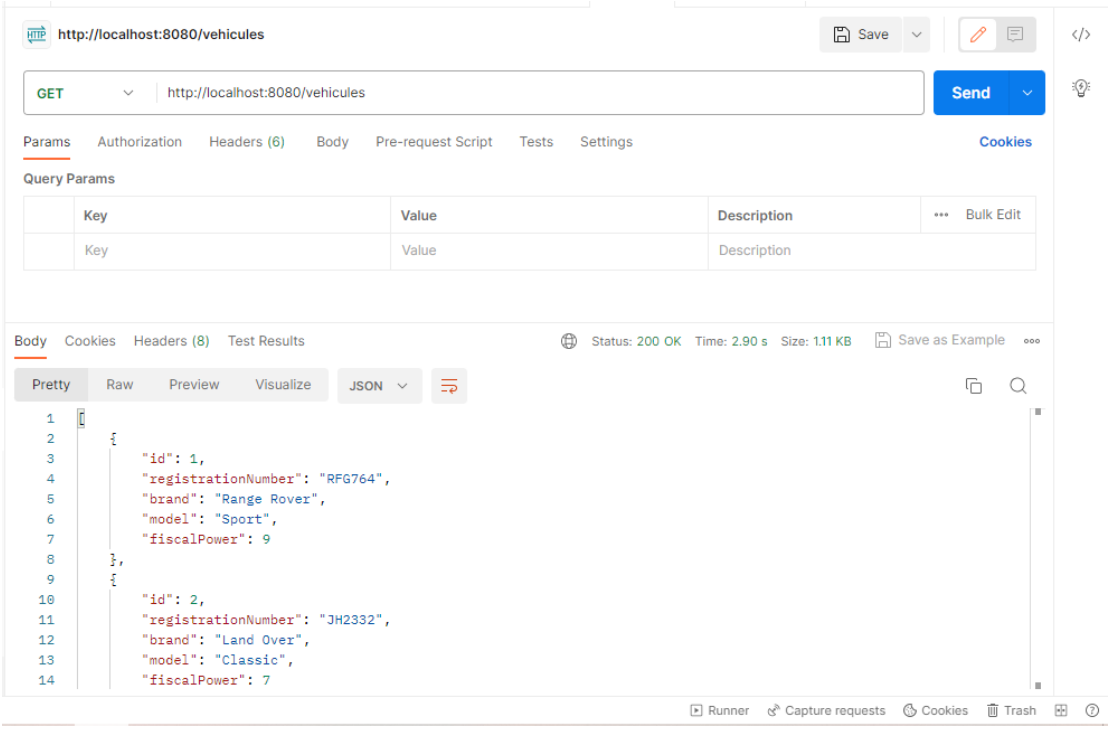
```
package com.example.immatriculationservice;

import ...

1 usage
@SpringBootApplication

public class ImmatriculationServiceApplication {

    no usages
    public static void main(String[] args) { SpringApplication.run(ImmatriculationServiceApplication.class,

    no usages
    @Bean
    public CommandLineRunner init(ProprietaireRepository proprietaireRepository, VehiculeRepository vehicul
        return args -> {
            // Create owners
            Proprietaire proprietaire1 = new Proprietaire();
            proprietaire1.setName("Walid");
            proprietaire1.setBirthDate(LocalDate.of( year: 2001,  month: 5,  dayOfMonth: 5));
            proprietaire1.setEmail("walid.etu@gmail.com");

            Proprietaire proprietaire2 = new Proprietaire();
            proprietaire2.setName("yassine");
            proprietaire2.setBirthDate(LocalDate.of( year: 1993,  month: 6,  dayOfMonth: 17));
            proprietaire2.setEmail("yassine@gmail.com");
```

Et on configure les propriétés de l'application :

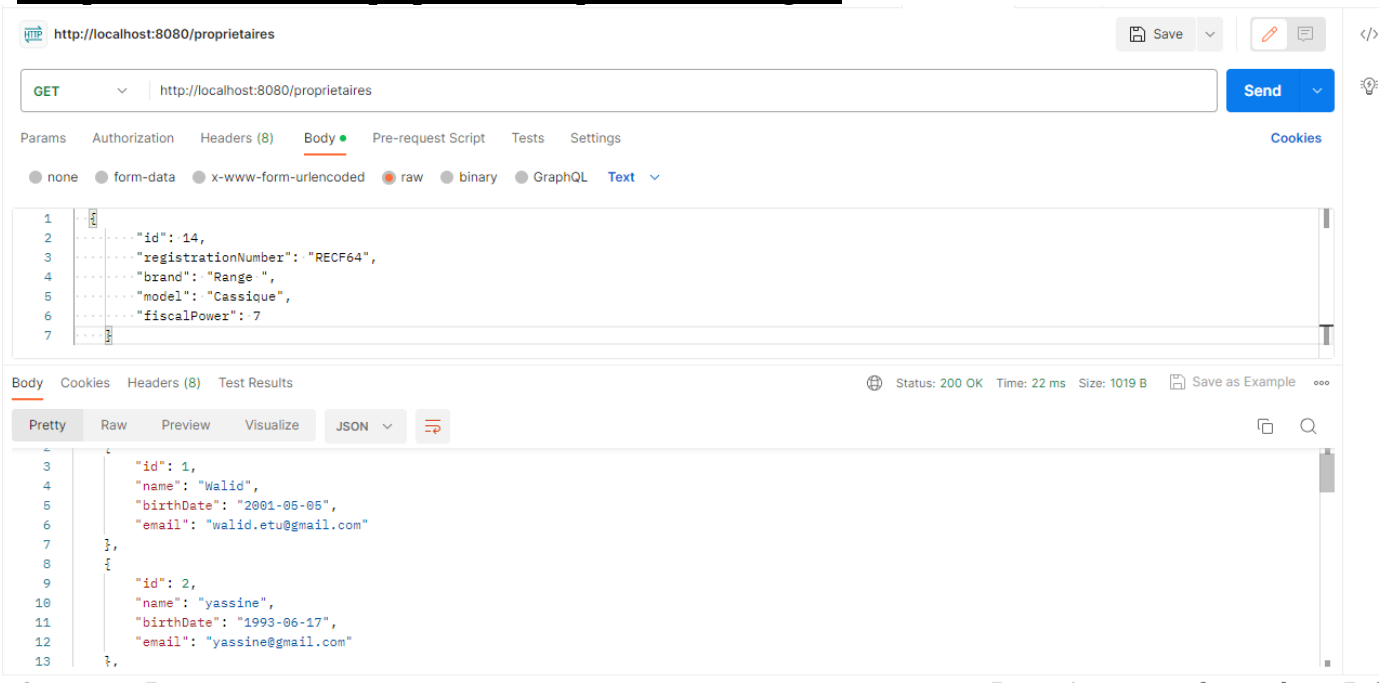
```
spring.port=8080
spring.application.name=immatriculation-service
spring.datasource.url=jdbc:h2:mem:immatriculation-db
spring.h2.console.enabled=true
#spring.graphql.graphiql.enabled=true
#grpc.server.port=8888
#grpc.server.address=*
spring.cloud.discovery.enabled=true
```

Maintenant on lance le micro service, et on teste les méthodes qu'on a développé a travers l'outil Postman : ***pour le Rest :***

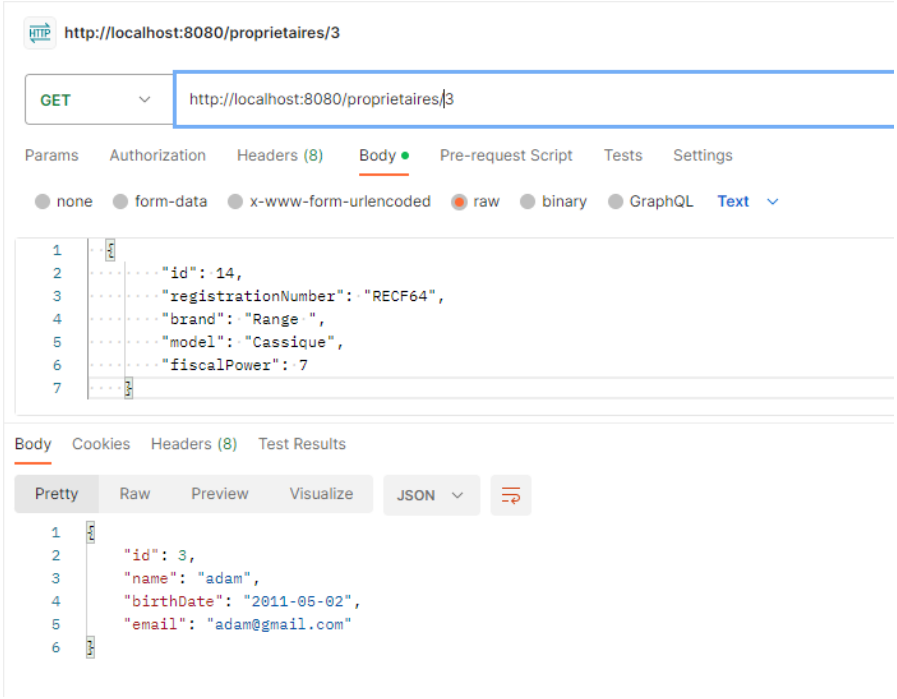
Récupération de toutes les véhicules : par la méthode get :



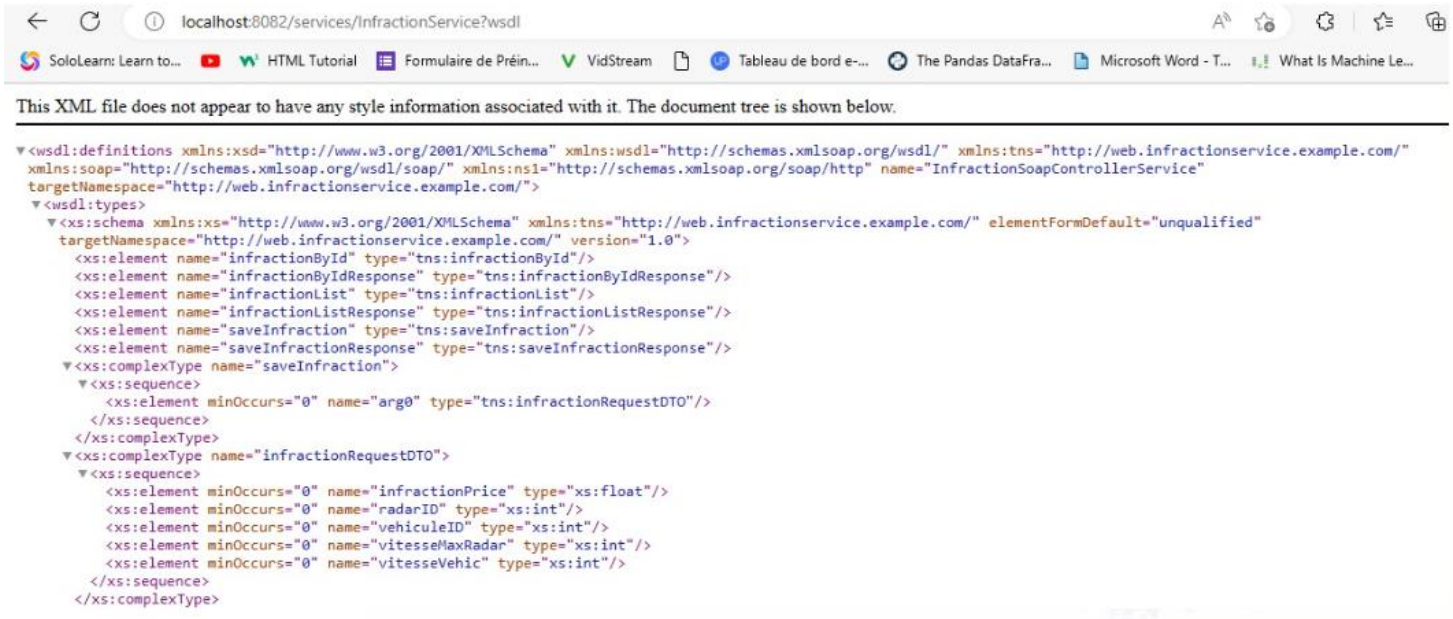
Récupération de toutes les proprietarires : par la méthode get :

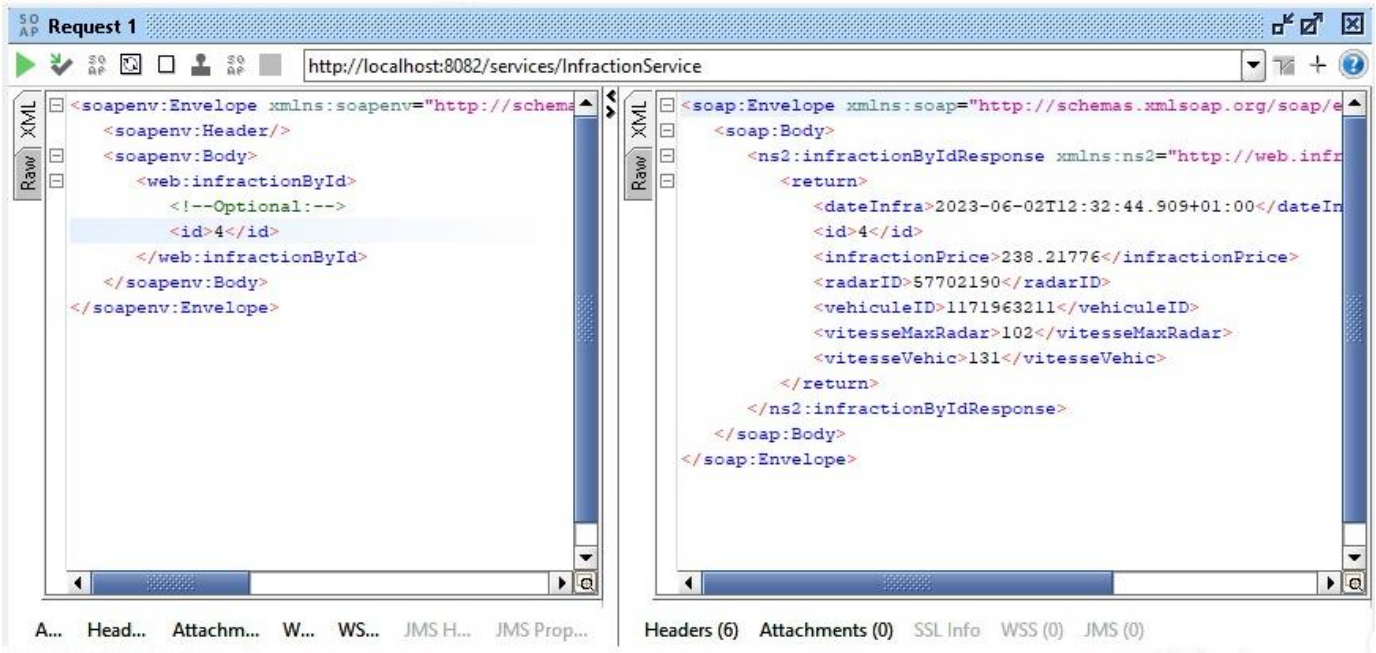


Récupération proprietarires par Id :



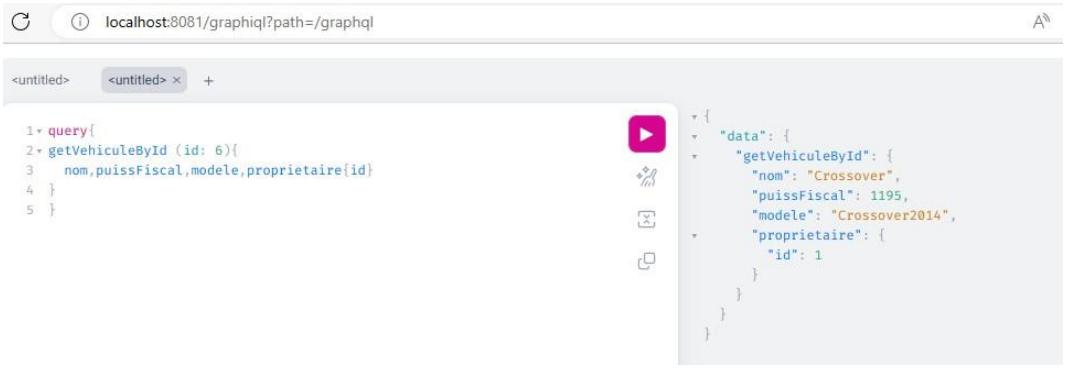
Maintenant on lance le micro service, et on teste les méthodes qu'on a développé à travers l'outil SOAPUI : ***pour le SOAP :***



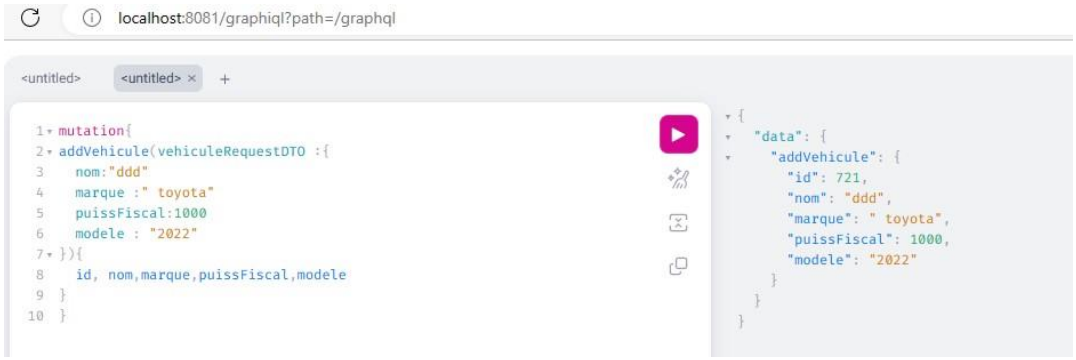

Request 1
http://localhost:8082/services/InfractionService
XML
Raw
<soapenv:Envelope xmlns:soapenv="http://schema
<soapenv:Header/>
<soapenv:Body>
<web:infractionById>
<!--Optional:-->
<id>4</id>
</web:infractionById>
</soapenv:Body>
</soapenv:Envelope>
A... Head... Attachm... W... WS... JMS H... JMS Prop...
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/e
<soap:Body>
<ns2:infractionByIdResponse xmlns:ns2="http://web.infr
<return>
<dateInfra>2023-06-02T12:32:44.909+01:00</dateIn
<id>4</id>
<infractionPrice>238.21776</infractionPrice>
<radarID>57702190</radarID>
<vehiculeID>1171963211</vehiculeID>
<vitesseMaxRadar>102</vitesseMaxRadar>
<vitesseVehic>131</vitesseVehic>
</return>
</ns2:infractionByIdResponse>
</soap:Body>
</soap:Envelope>
Headers (6) Attachments (0) SSL Info WSS (0) JMS (0)

Maintenant, on passe vers le teste de web service GraphQL :

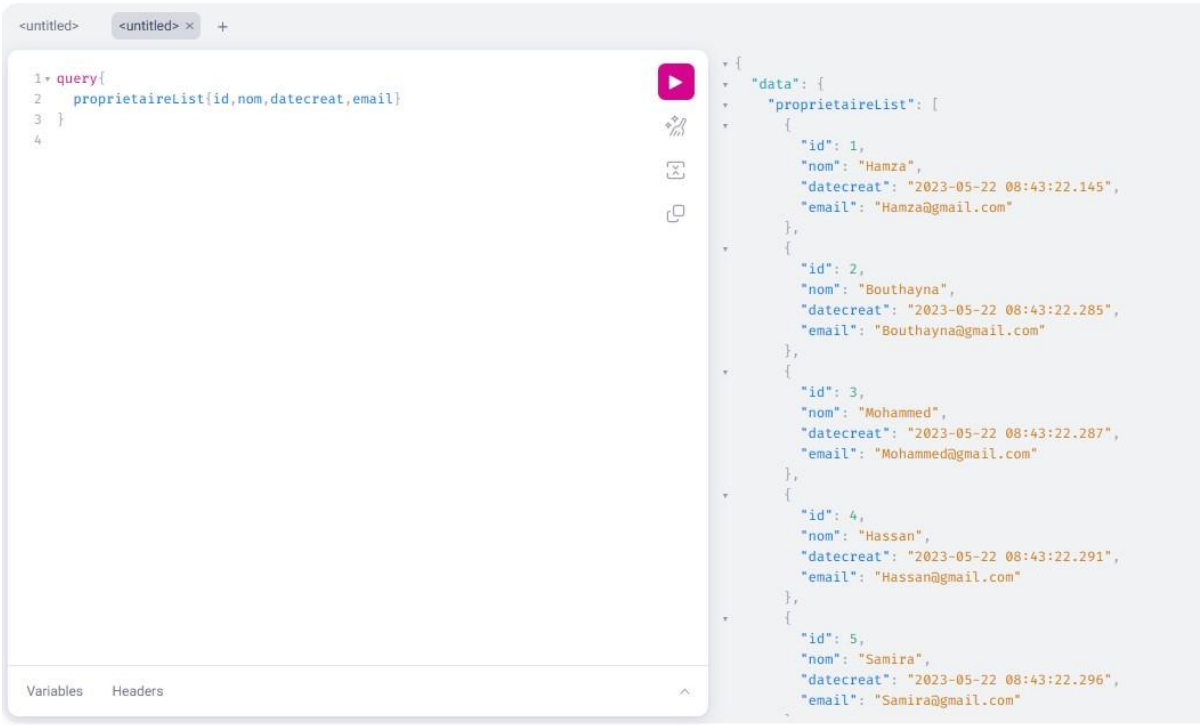
La récupération d'un véhicule par identifiant :



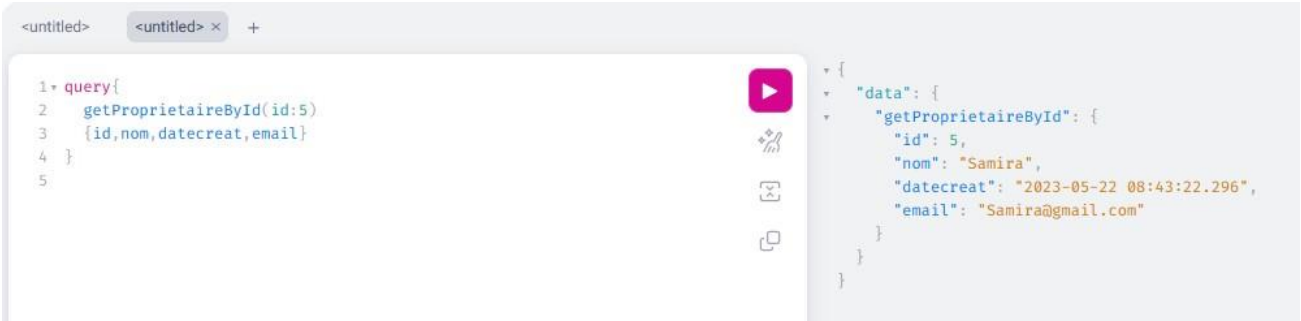
L'ajout d'un véhicule :



Pour l'entité proprietaire, on a ***proprietaireList*** :



La récupération d'un propriétaire par identifiant **:**



4. Développer le micro-service Infractions:

Dans le package entities on a créé infraction :

```java
@Data @NoArgsConstructor @AllArgsConstructor @ToString @Builder
public class Infraction {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private Date date;
    no usages
    private Boolean paid = Boolean.FALSE;
    no usages
    private  Long numeroRadar;
    no usages
    private Double vitesseMaxRadar;
    no usages
    private String matriculeVehicule;
    no usages
    private double vitesseVehicule;
    no usages
    private double montantFraction;
    no usages
    private String ownerName;
}
```

Dans le package repositories on a créé l'interface infractionrepository :

```java
package com.example.infractionservices.repositories;

import com.example.infractionservices.entities.Infraction;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

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

Dans le package web on a créé la classe InfractionRestRepository :

```java
package com.example.infractionservices.web;

import ...


no usages
@RestController
@AllArgsConstructor
@RestResource
@CrossOrigin(origins = "http://localhost:4200")
public class InfractionRestController {

    6 usages
    private InfractionRepository infractionRepository;

    //private VehiculeRestClient vehicleRestClient;
    //private RadarRestClient radarRestClient;

    no usages
    @GetMapping("/infractions")
    public List<Infraction> getAllInfractions() { return infractionRepository.findAll(); }

    no usages
    @GetMapping("/infractions/{id}")
    public Infraction getOneInfraction(@PathVariable long id) { return infractionRepository.findById(id).get(); }

    //Update method :
```

La création des infractions :

```java
package com.example.infractionservices;

import ...

1 usage
@SpringBootApplication
public class InfractionServicesApplication {
    no usages
    public static void main(String[] args) {
        SpringApplication.run(InfractionServicesApplication.class, args);
    }

    no usages
    @Bean
    public CommandLineRunner demo(InfractionRepository infractionRepository) {
        return args -> {
            // Create 6 instances of Infraction
            Infraction infraction1 = Infraction.builder()
                    .date(new Date())
                    .numeroRadar(2L)
                    .vitesseMaxRadar(100.0)
                    .matriculeVehicule("12AABB")
                    .vitesseVehicule(120.0)
                    .montantFraction(50.0)
                    .ownerName("Messi")
                    .build();
```

Application.Properties :

```
spring.application.name=infraction-service
server.port=8081
spring.datasource.url=jdbc:h2:mem:infraction-db
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=true
```

5. Développer le micro-service Radar:

Dans le package entities on a créé Radar :

```
package com.example.radarservice.entities;

import ...

20 usages
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Radar {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String name;
    no usages
    private Double maxSpeed = 120.0;
    no usages
    private boolean status; // recently added
    //private  String numeroRadar;
    no usages
    private double longitude;
    no usages
    private double latitude;

}
```

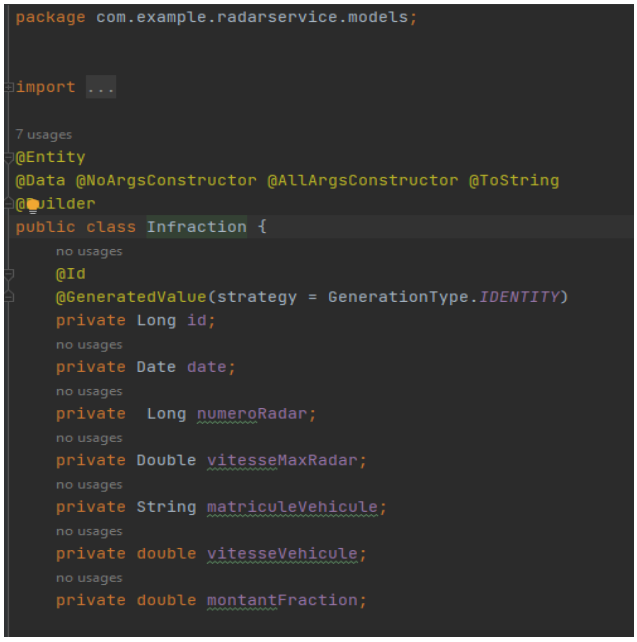
Dans le package feignClients on a créé ImmatriculationClient :

```java
package com.example.radarservice.feignClients;


import ...

2 usages
@FeignClient(name = "IMMATRICULATION-SERVICE", url = "http://localhost:8080")
public interface ImmatriculationClient {
    no usages
    @GetMapping("/proprietaires/{id}")
    Proprietaire getProprietaire(@PathVariable Long id);

    1 usage
    @GetMapping("/vehicules/matricules/{matricule}")
    Vehicule getVehiculeByMatricule(@PathVariable String matricule);

}
```

Dans le package feignClients on a créé InfractionClient :

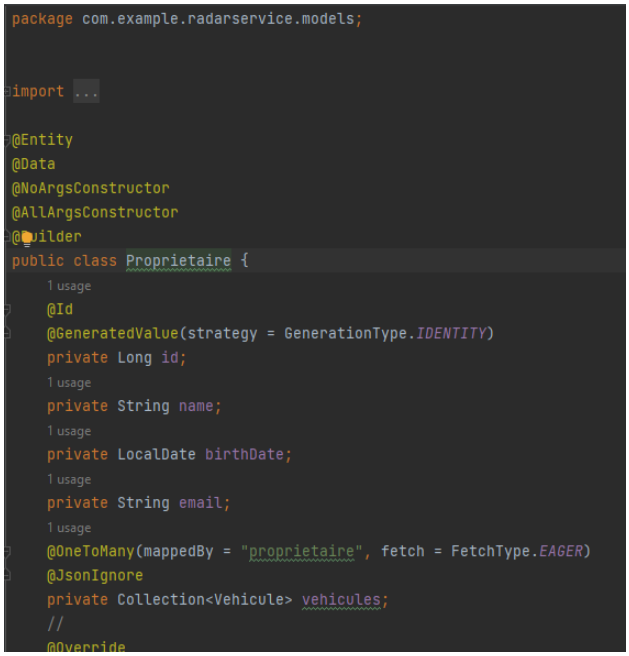
```java
package com.example.radarservice.feignClients;


import ...

2 usages
@FeignClient(name = "INFRACTION-SERVICE",url = "http://localhost:8081")
public interface InfractionClient {
    1 usage
    @PostMapping ("/infractions")
    Infraction createInfraction(@RequestBody Infraction infraction);

}
```

Dans le package models on a créé la classe Infraction :



```
package com.example.radarservice.models;


import ...

7 usages
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @ToString
@Builder
public class Infraction {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private Date date;
    no usages
    private  Long numeroRadar;
    no usages
    private Double vitesseMaxRadar;
    no usages
    private String matriculeVehicule;
    no usages
    private double vitesseVehicule;
    no usages
    private double montantFraction;
```

Dans le package models on a créé la classe Proprietaire :



```
package com.example.radarservice.models;


import ...

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Proprietaire {
    1 usage
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String name;
    1 usage
    private LocalDate birthDate;
    1 usage
    private String email;
    1 usage
    @OneToMany(mappedBy = "proprietaire", fetch = FetchType.EAGER)
    @JsonIgnore
    private Collection<Vehicule> vehicules;
    //
    @Override
```

Dans le package models on a créé la classe Vehicule :

```java
package com.example.radarservice.models;


import ...


@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Vehicule {
    1 usage
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    1 usage
    private String registrationNumber;
    1 usage
    private String brand;
    1 usage
    private String model;
    1 usage
    private int fiscalPower;
    no usages
    @ManyToOne
    @JsonIgnore
    Proprietaire proprietaire;
```

Dans le package repositories on a créé l'interface RadarRepository :

```java
package com.example.radarservice.repositories;

import ...

4 usages
public interface RadarRepository extends JpaRepository<Radar, Long> {
    1 usage
    Page<Radar> findByNameContains(String keyword, Pageable pageable);
    //Optional<Radar> findByNumero(String numero);
}
```

Dans le package web on a créé l'interface RadarRestController :

```java
no usages
@RestController
@AllArgsConstructor
@CrossOrigin(origins = "http://localhost:4200")
public class RadarRestController {
    10 usages
    private RadarRepository radarRepository;
    1 usage
    private ImmatriculationClient immatriculationClient;
    1 usage
    private InfractionClient infractionClient;


    no usages
    @GetMapping("/radars")
    public List<Radar> getAllRadars() { return radarRepository.findAll(); }

    no usages
    @GetMapping("/radars/{id}")
    public Radar getRadarById(@PathVariable Long id) {
        if (id == null) {
            throw new IllegalArgumentException("Radar ID cannot be null");
        }

        return radarRepository.findById(id).orElse( other: null);
    }
```

La création des Radars :

```java
package com.example.radarservice;

import ...

1 usage
@SpringBootApplication
@EnableFeignClients
public class RadarServiceApplication {

    no usages
    public static void main(String[] args) { SpringApplication.run(RadarServiceApplication.class, args); }

    no usages
    @Bean
    public CommandLineRunner init(RadarRepository radarRepository) {
        return args -> {
            Radar radar = Radar.builder()
                    .name("Walid Radar")
                    .maxSpeed(100.0)
                    .longitude(123.456)
                    .latitude(78.910)
                    //.numeroRadar("P00")
                    .build();

            radarRepository.save(radar);
            Radar radar1 = Radar.builder()
                    .name("Adam Radar")
```

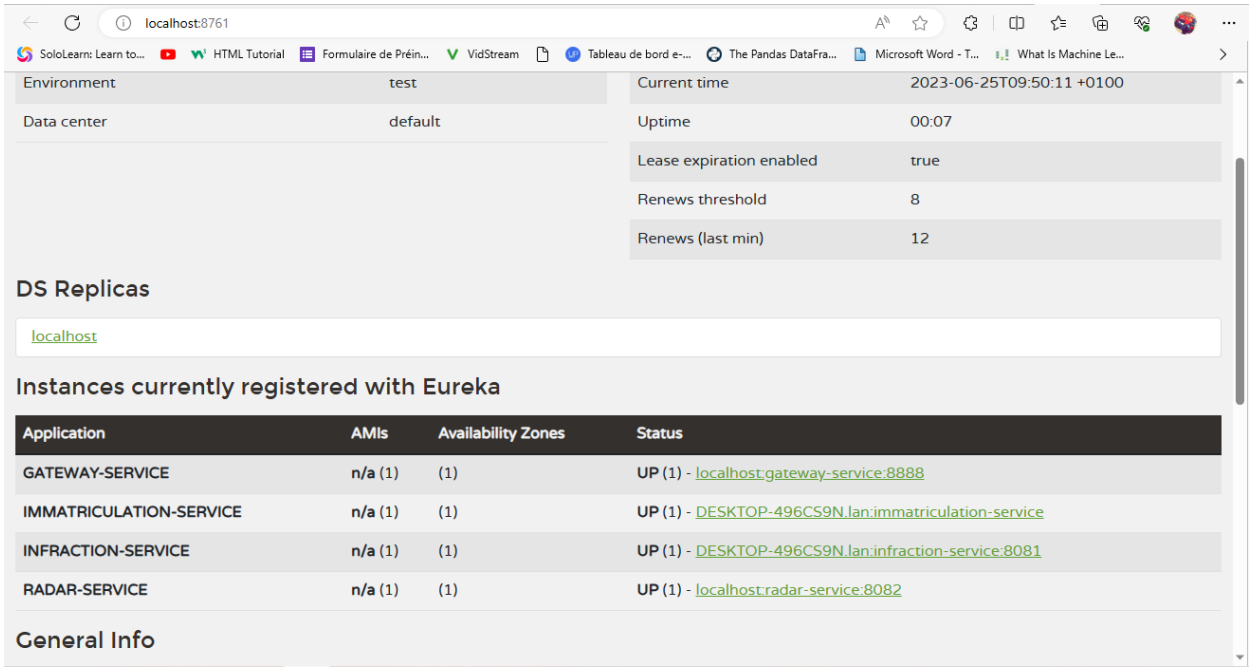
Application.properties :

```
spring.application.name=radar-service
server.port=8082
spring.datasource.url=jdbc:h2:mem:radar-db
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=true
```

6. Mettre en place les services techniques de l'architecture micro-service (Gateway, Eureka Discovery service)

Gateway est responsable de la réception des requêtes client et de la communication avec le service eureka :

Le service eureka:

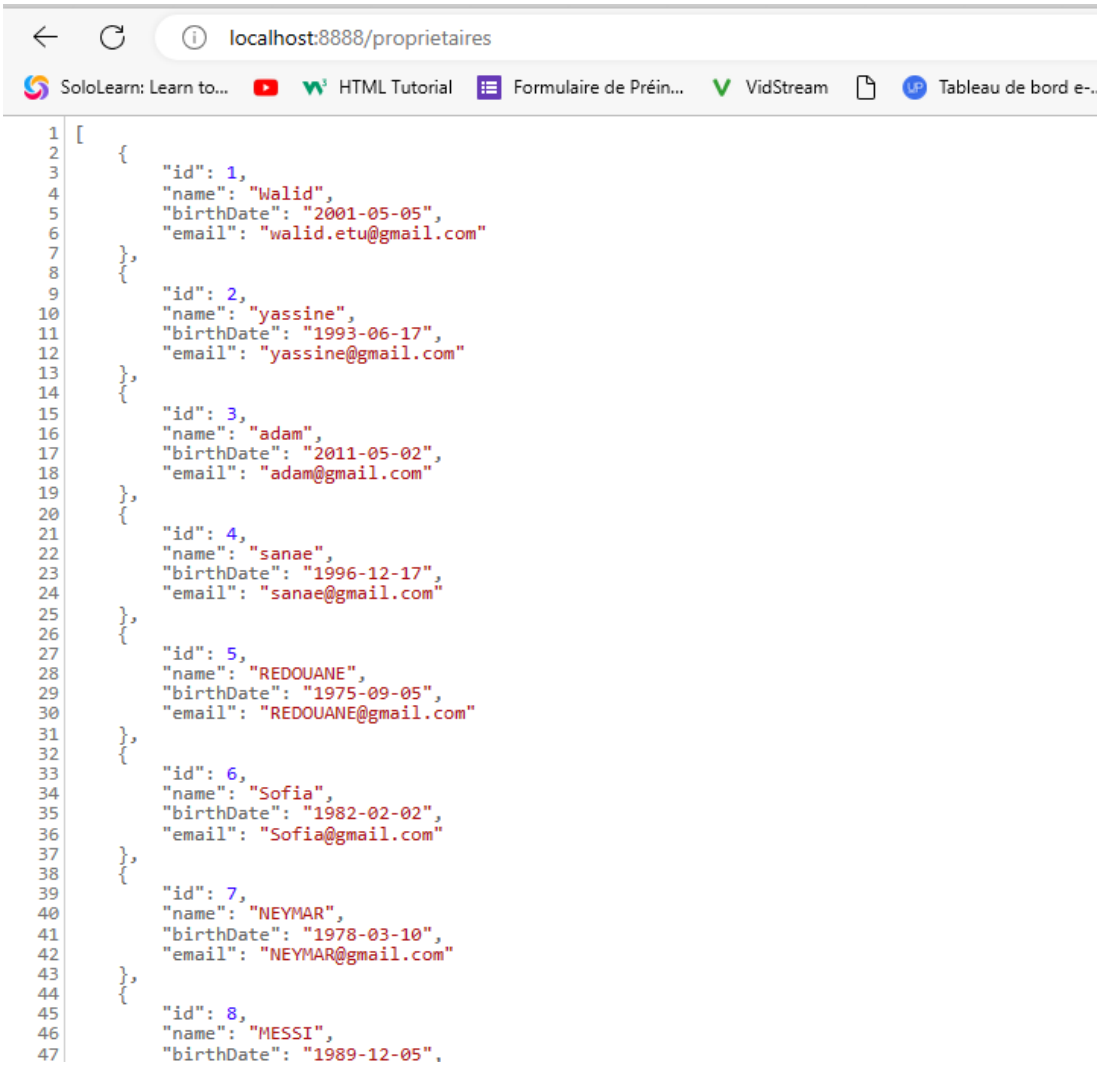


Le service GATEWAY :

```java
package com.example.gatewayprojet;

import ...

1 usage
@SpringBootApplication
public class GatewayProjetApplication {

    no usages
    public static void main(String[] args) { SpringApplication.run(GatewayProjetApplication.class, args); }
    no usages
    @Bean
    RouteLocator routeLocator(RouteLocatorBuilder builder){
        return builder.routes()
                .route((r)->r.path("/proprietaires/**").uri("http://localhost:8080/"))
                .route((r)->r.path("/vehicules/**").uri("http://localhost:8080/"))
                .route((r)->r.path("/infractions/**").uri("http://localhost:8081/"))
                .route((r)->r.path("/radars/**").uri("http://localhost:8082/"))
//              .route((r)->r.path("/proprietaires/**").uri("lb://IMMATRICULATION-SERVICE"))
//              .route((r)->r.path("/vehicules/**").uri("lb://INFRACTION-SERVICE"))
//              .route((r)->r.path("/infractions/**").uri("lb://IMMATRICULATION-SERVICE"))
                .build();
    }

//  @Bean
//  DiscoveryClientRouteDefinitionLocator definitionLocator(
//          ReactiveDiscoveryClient rdc,
```

On essaie d'accéder aux services depuis le port de la gateway :

Le microservice Proprietaire :

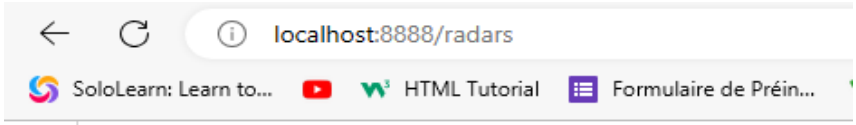



```
[
    {
        "id": 1,
        "name": "Walid",
        "birthDate": "2001-05-05",
        "email": "walid.etu@gmail.com"
    },
    {
        "id": 2,
        "name": "yassine",
        "birthDate": "1993-06-17",
        "email": "yassine@gmail.com"
    },
    {
        "id": 3,
        "name": "adam",
        "birthDate": "2011-05-02",
        "email": "adam@gmail.com"
    },
    {
        "id": 4,
        "name": "sanae",
        "birthDate": "1996-12-17",
        "email": "sanae@gmail.com"
    },
    {
        "id": 5,
        "name": "REDOUANE",
        "birthDate": "1975-09-05",
        "email": "REDOUANE@gmail.com"
    },
    {
        "id": 6,
        "name": "Sofia",
        "birthDate": "1982-02-02",
        "email": "Sofia@gmail.com"
    },
    {
        "id": 7,
        "name": "NEYMAR",
        "birthDate": "1978-03-10",
        "email": "NEYMAR@gmail.com"
    },
    {
        "id": 8,
        "name": "MESSI",
        "birthDate": "1989-12-05",
```

Le microservice infraction :

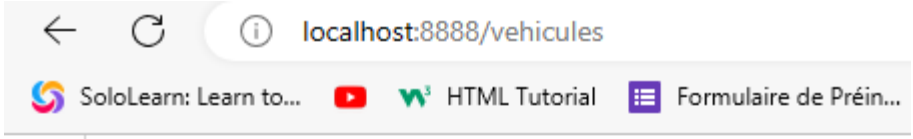

```
[
    {
        "id": 1,
        "date": "2023-06-25T08:48:59.243+00:00",
        "paid": null,
        "numeroRadar": 2,
        "vitesseMaxRadar": 120,
        "matriculeVehicule": "12AABB",
        "vitesseVehicule": 120,
        "montantFraction": 50,
        "ownerName": "NEYMAR"
    },
    {
        "id": 2,
        "date": "2023-06-25T08:48:59.243+00:00",
        "paid": null,
        "numeroRadar": 2,
        "vitesseMaxRadar": 80,
        "matriculeVehicule": "z20DD",
        "vitesseVehicule": 90,
        "montantFraction": 30,
        "ownerName": "HAKIMI"
    },
    {
        "id": 3,
        "date": "2023-06-25T08:48:59.243+00:00",
        "paid": null,
        "numeroRadar": 3,
        "vitesseMaxRadar": 80,
        "matriculeVehicule": "XYZ789",
        "vitesseVehicule": 80,
        "montantFraction": 90,
        "ownerName": "MESSI "
    },
    {
        "id": 4,
        "date": "2023-06-25T08:48:59.243+00:00",
        "paid": null,
        "numeroRadar": 2,
        "vitesseMaxRadar": 120,
        "matriculeVehicule": "UIO890",
        "vitesseVehicule": 130,
        "montantFraction": 20,
        "ownerName": "ADAM "
    }
]
```


Le microservice radar :



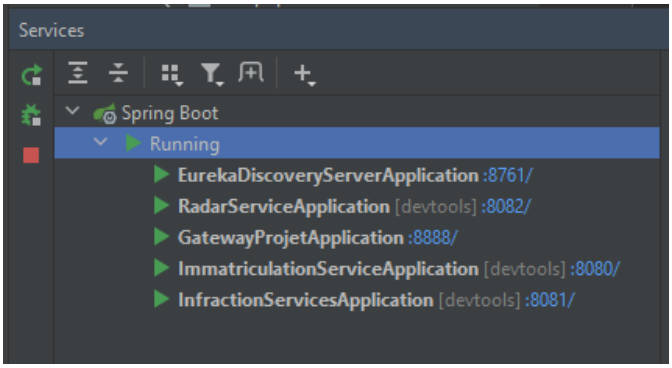

```
[
    {
        "id": 1,
        "name": "Test Radar",
        "maxSpeed": 100,
        "status": false,
        "longitude": 123.456,
        "latitude": 78.91
    },
    {
        "id": 2,
        "name": "adam Radar",
        "maxSpeed": 100.3,
        "status": false,
        "longitude": 130.456,
        "latitude": 30.91
    },
    {
        "id": 3,
        "name": "walid Radar",
        "maxSpeed": 130.9,
        "status": false,
        "longitude": 123.456,
        "latitude": 78.91
    }
]
```

Le microservice Vehicules :



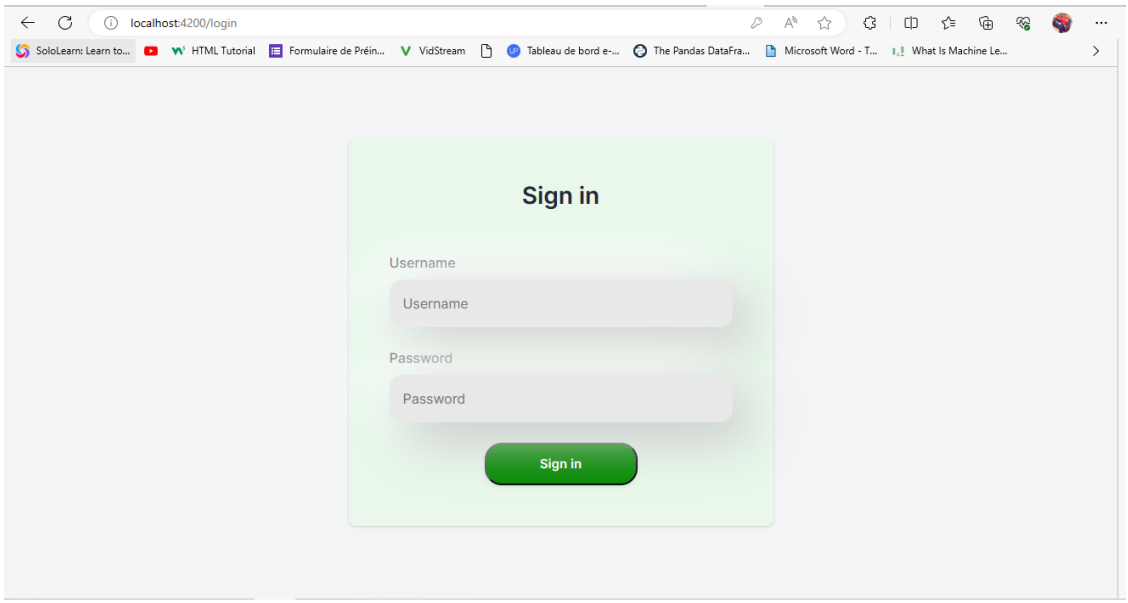

```
[
    {
        "id": 1,
        "registrationNumber": "RFG764",
        "brand": "Range Rover",
        "model": "Sport",
        "fiscalPower": 9
    },
    {
        "id": 2,
        "registrationNumber": "JH2332",
        "brand": "Land Over",
        "model": "Classic",
        "fiscalPower": 7
    },
    {
        "id": 3,
        "registrationNumber": "HJG412",
        "brand": "BMW",
        "model": "2021",
        "fiscalPower": 10
    },
    {
        "id": 4,
        "registrationNumber": "87TGH",
        "brand": "DACIA",
        "model": "SANDRO",
        "fiscalPower": 6
    },
    {
        "id": 5,
        "registrationNumber": "JKG2367",
        "brand": "FIAT",
        "model": "2009",
        "fiscalPower": 4
    },
    {
        "id": 6,
        "registrationNumber": "I5612",
        "brand": "Mercedes",
        "model": "not",
        "fiscalPower": 9
    },
    {
```


Démarrer tous les Micro Services :

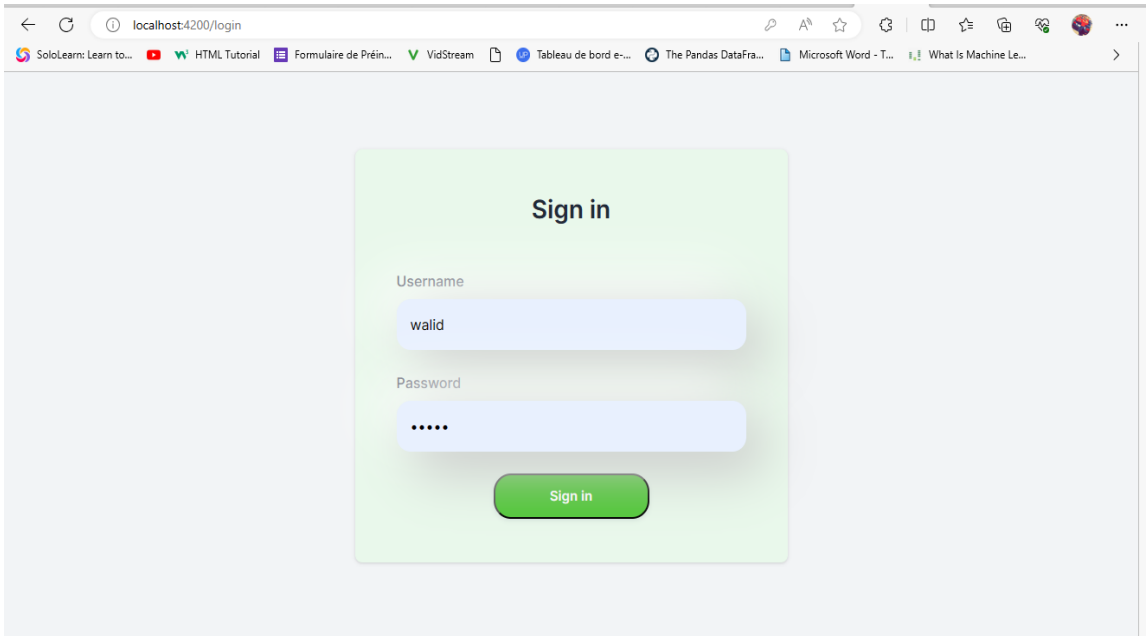


La Partie Front End avec l'utilisation de Angular :

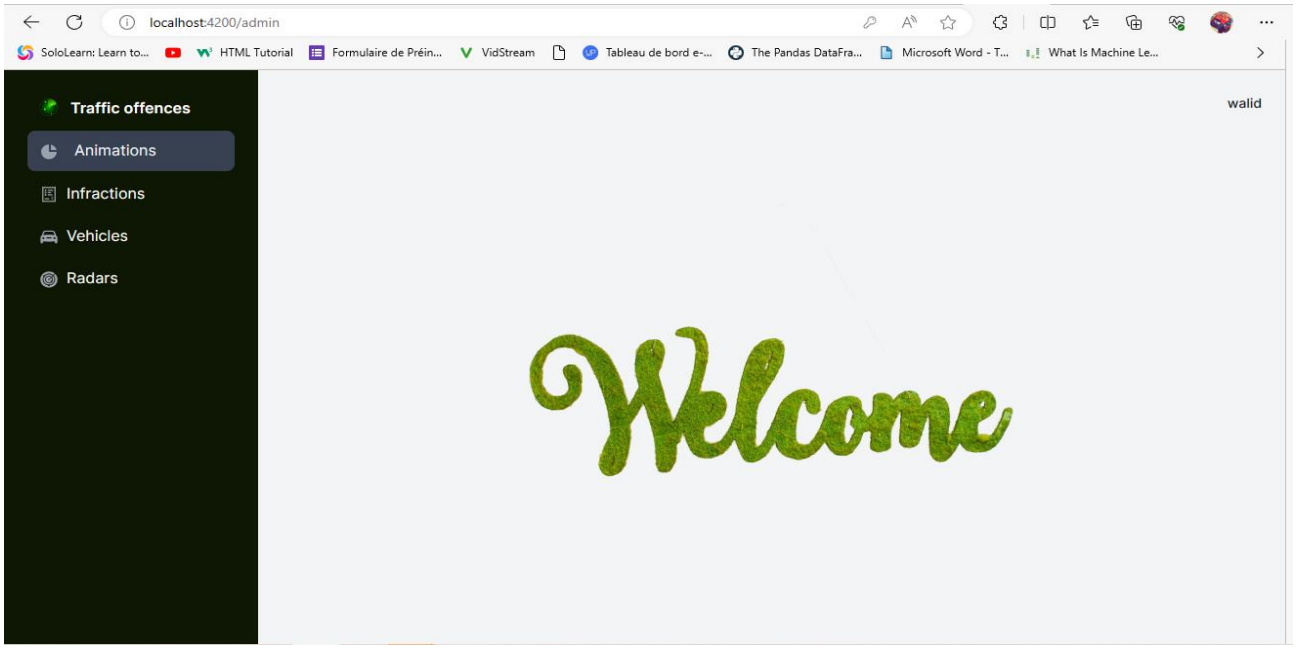
La page de Login :



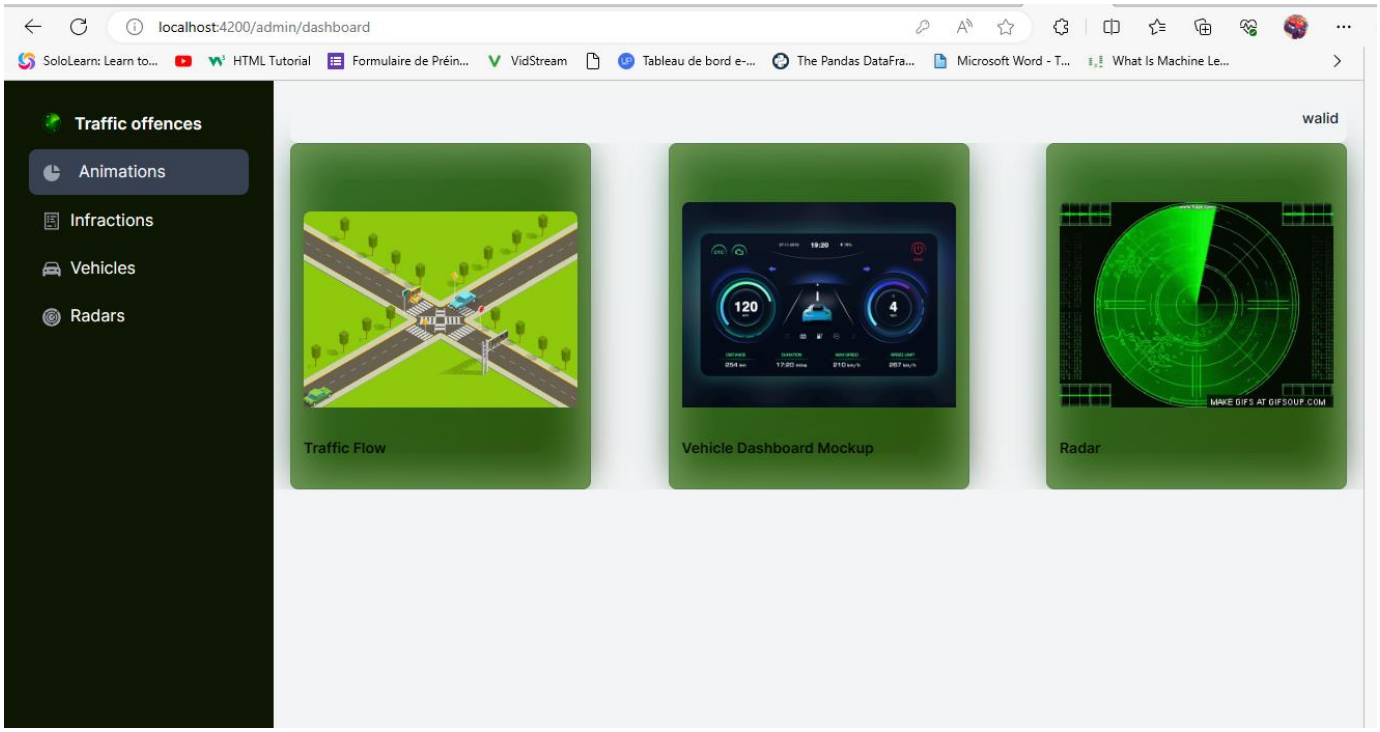
Saisir le nom d'utilisateur et le mot de passe :



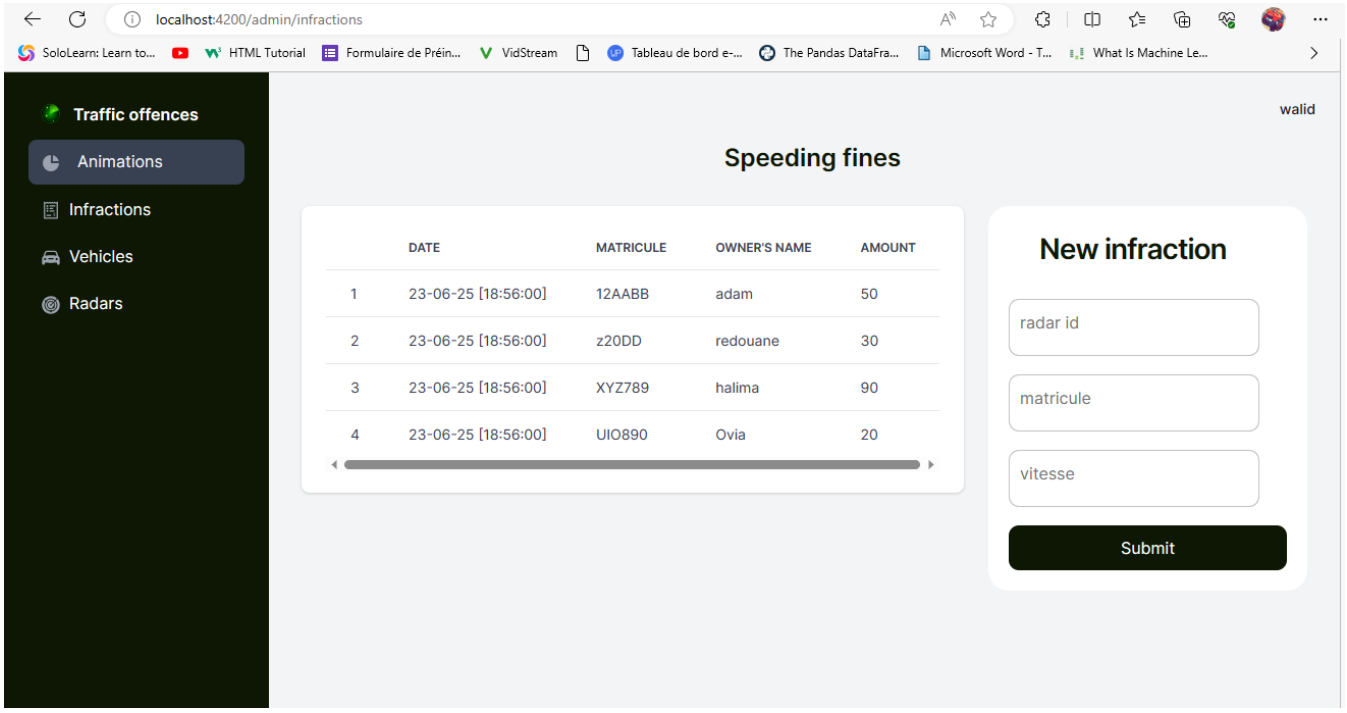
La page d’accueil de Dashboard :



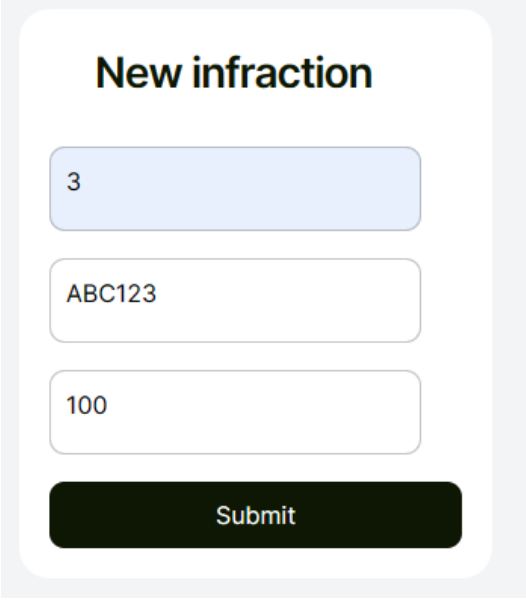
La page d’animations de Dashboard :



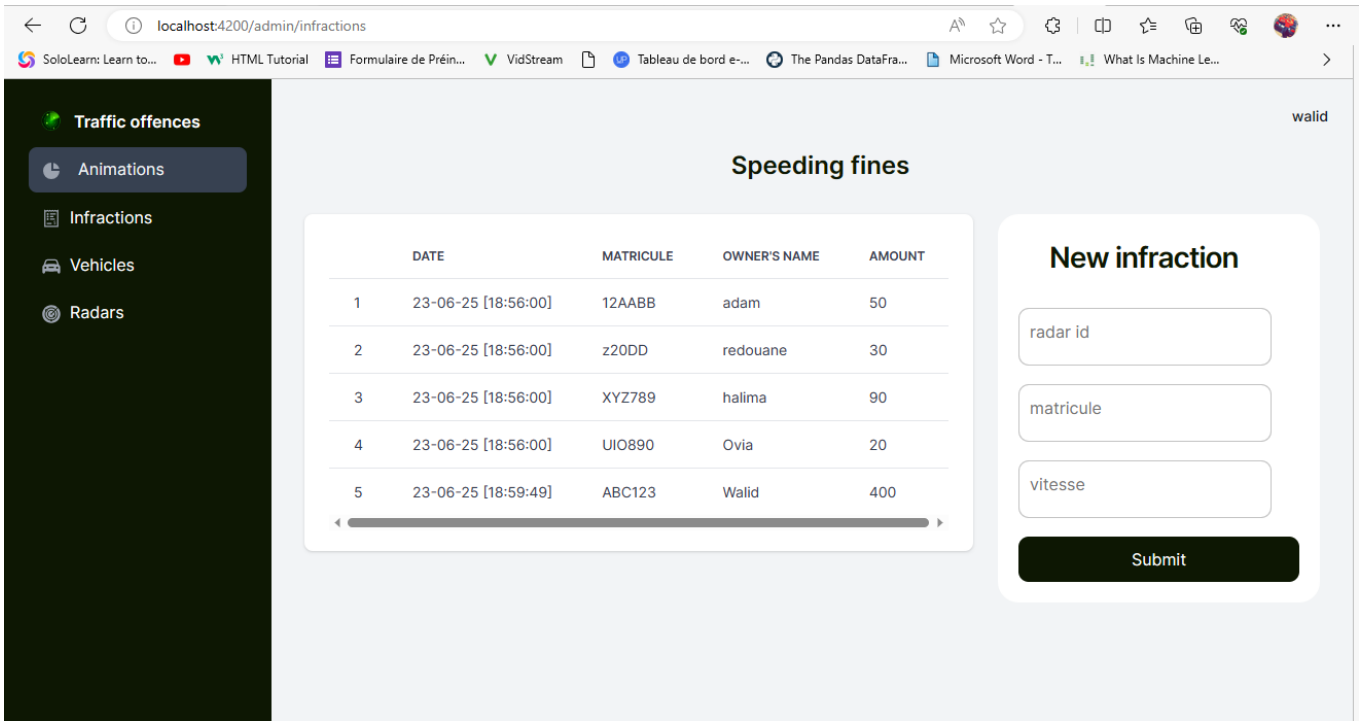
La page d’infraction de Dashboard :



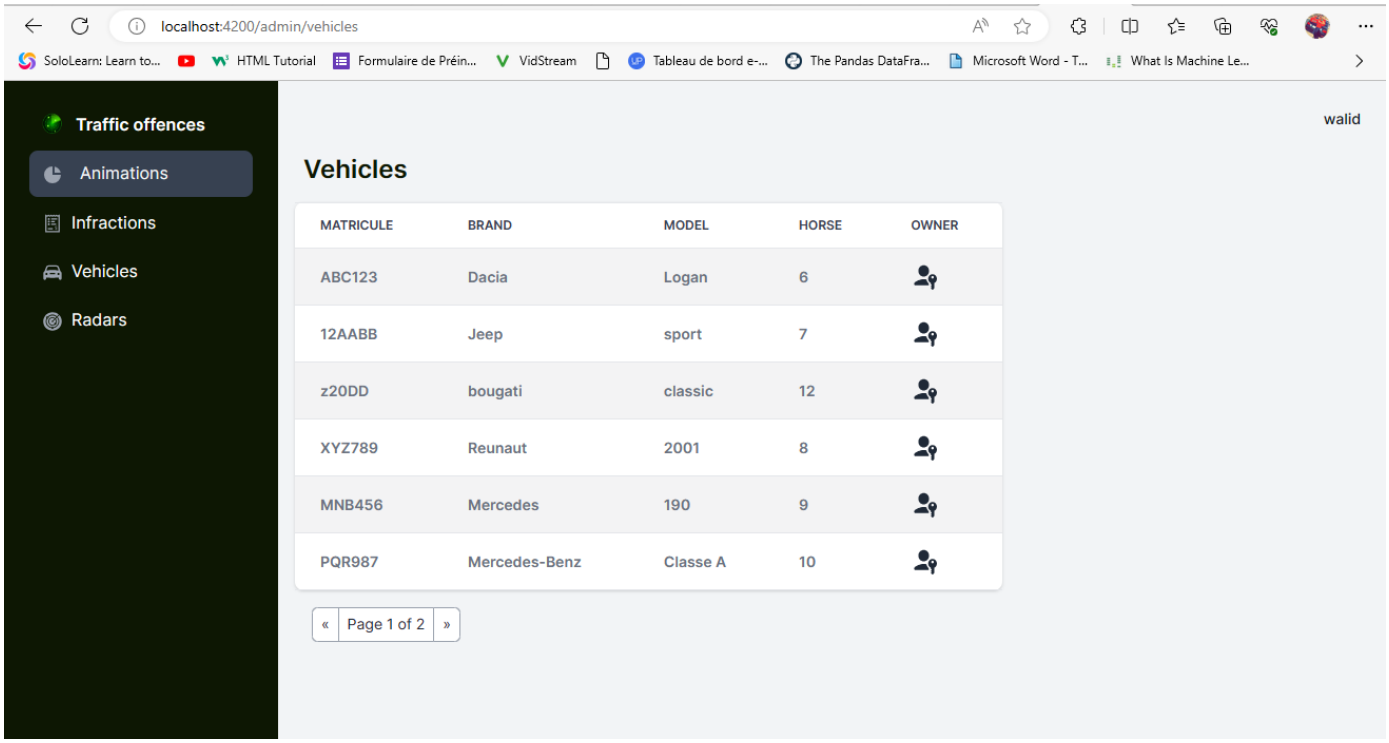
Ajouter une infraction :



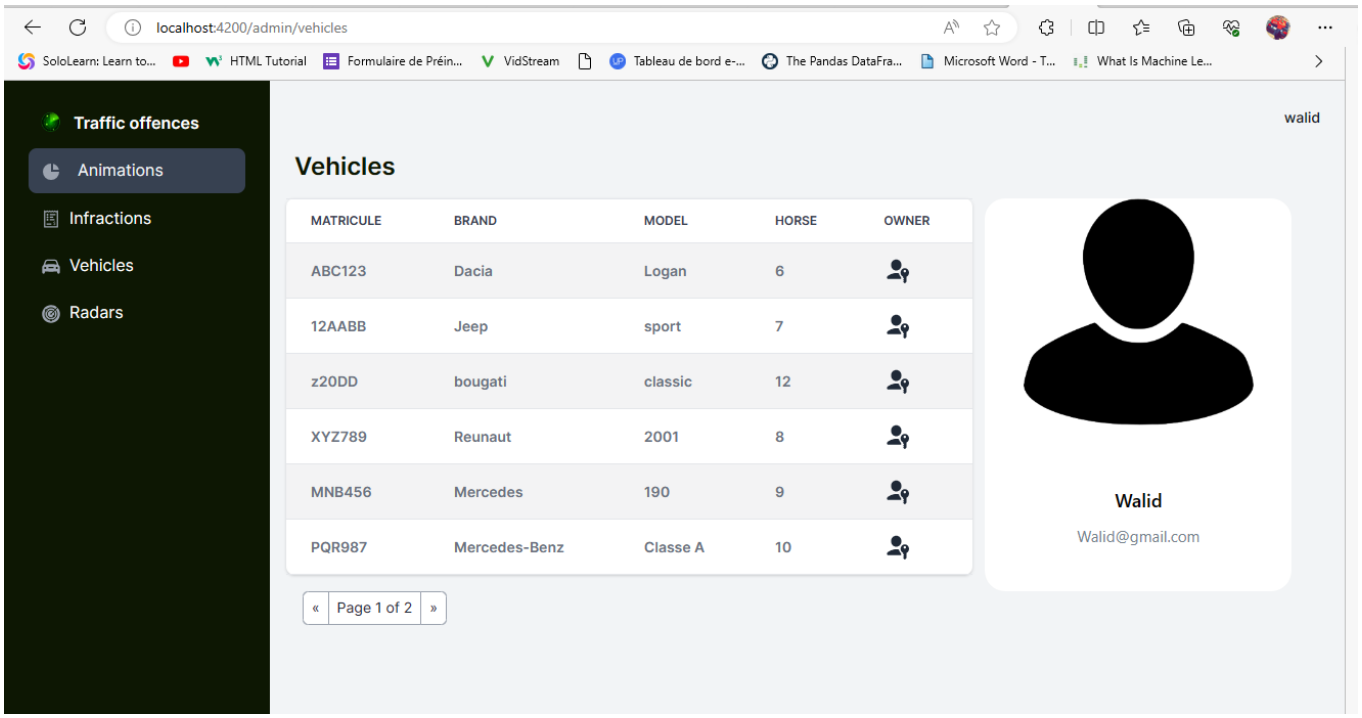
Voilà l’infraction a ajoutée :



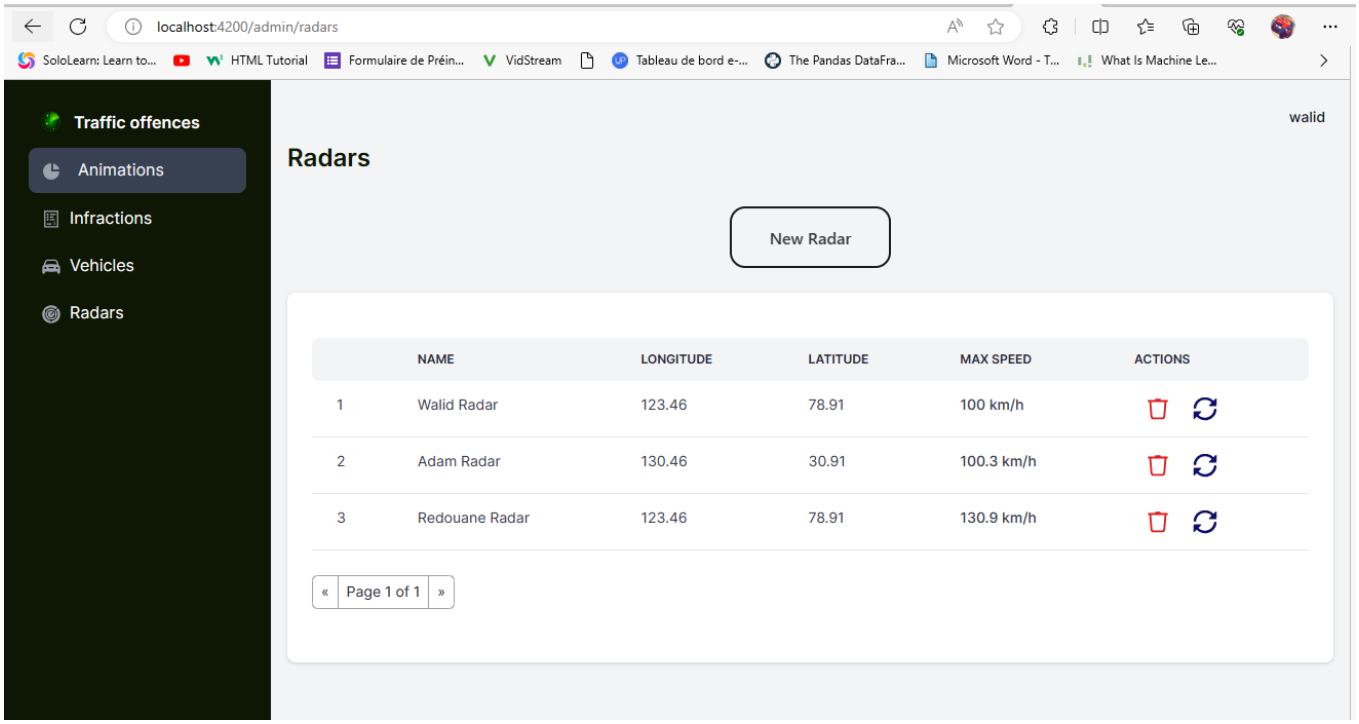
La page Vehicule de Dashboard :



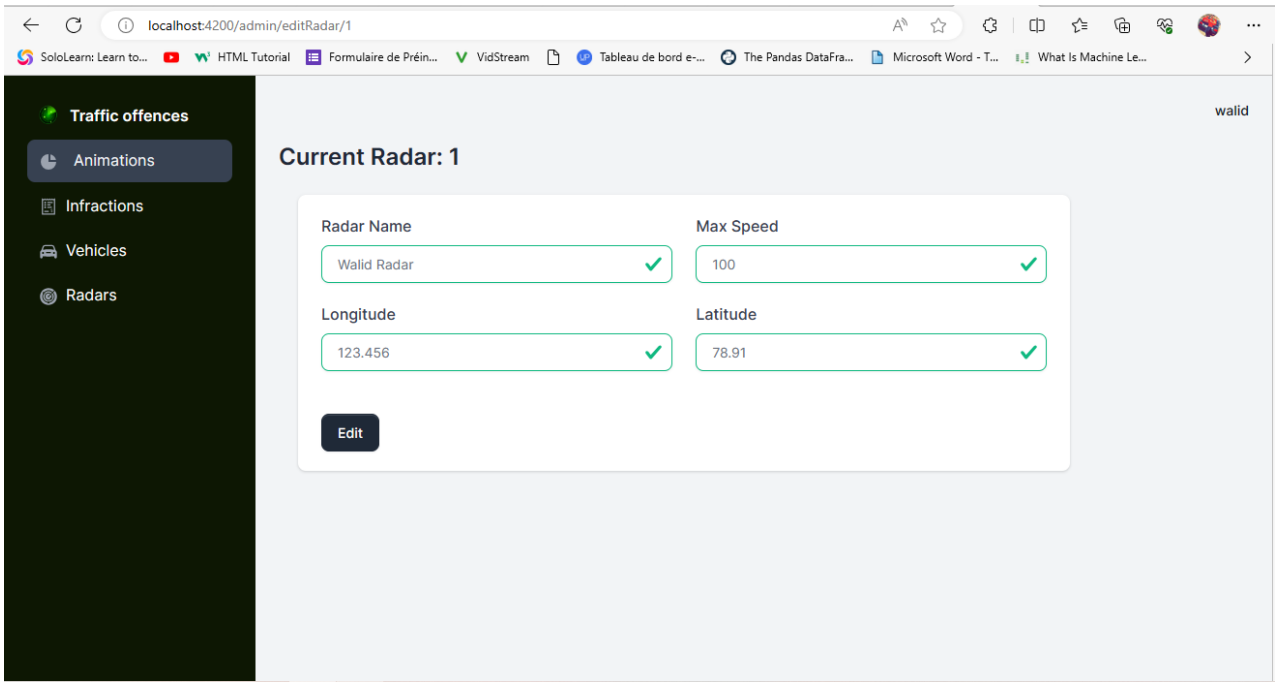
En cliquant sur le propriétaire de chaque ligne, le nom et l'adresse électronique du propriétaire s'affichent :



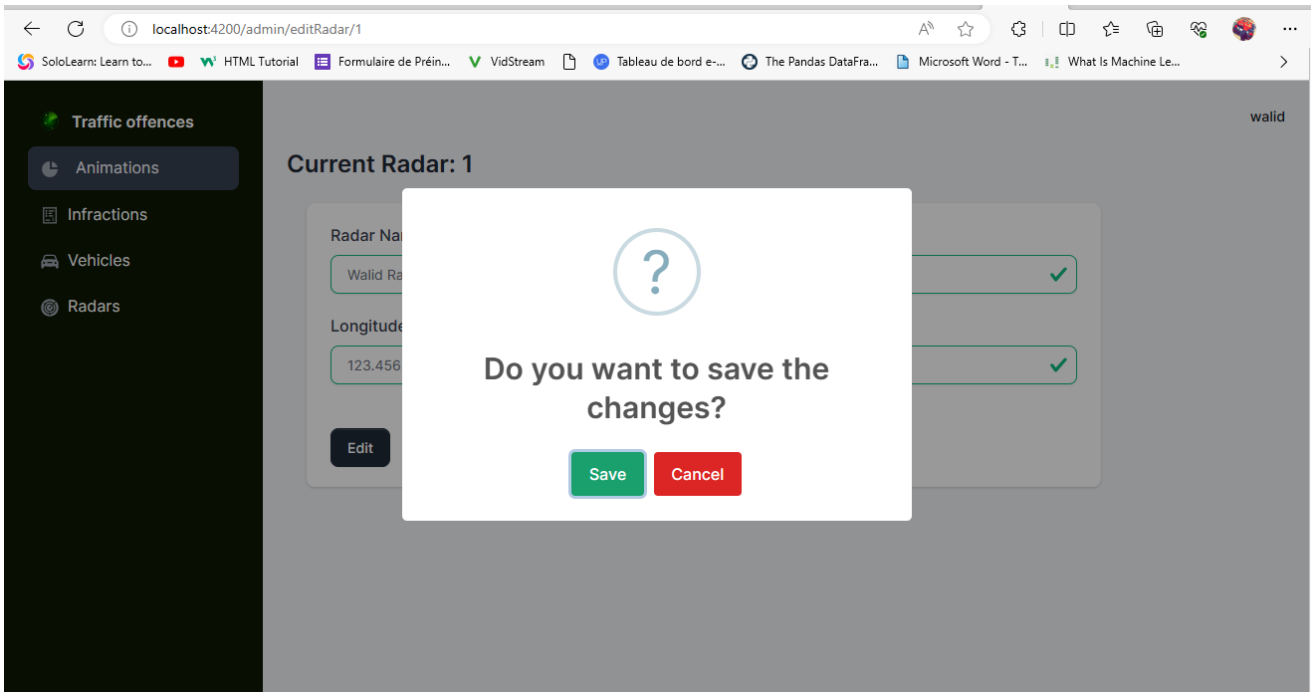
La page Radar de Dashboard :

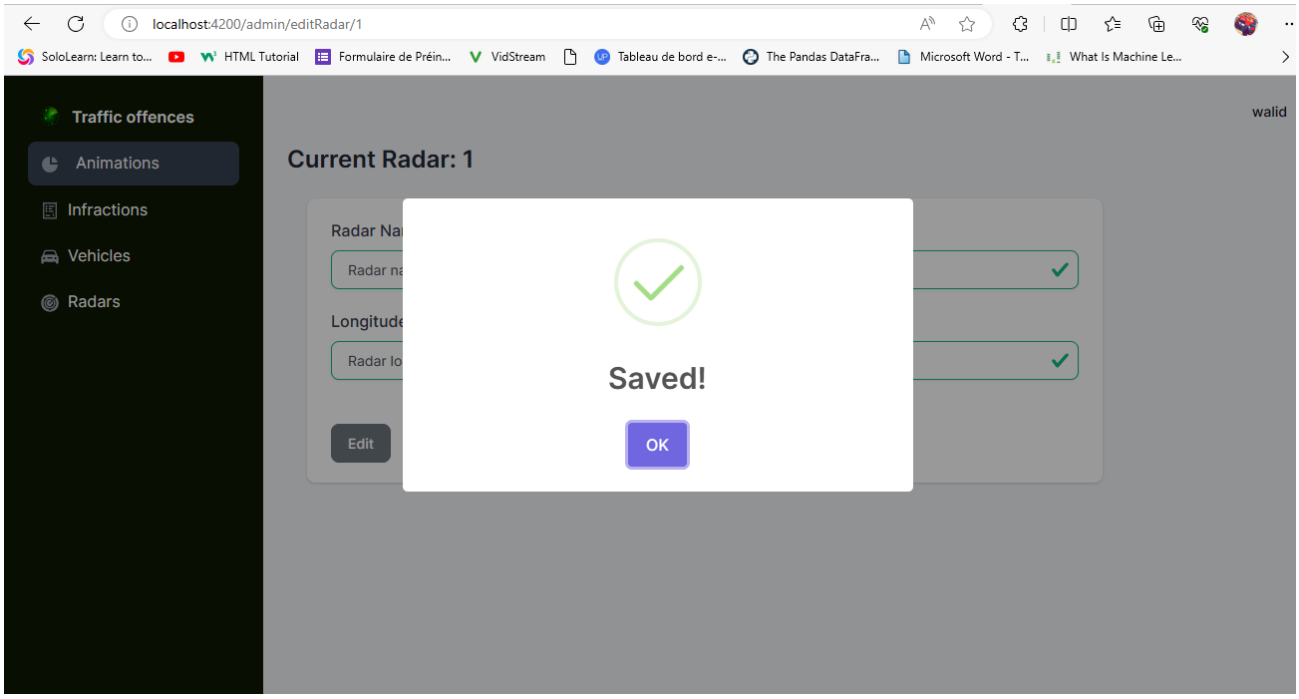


Modification d’un radar :

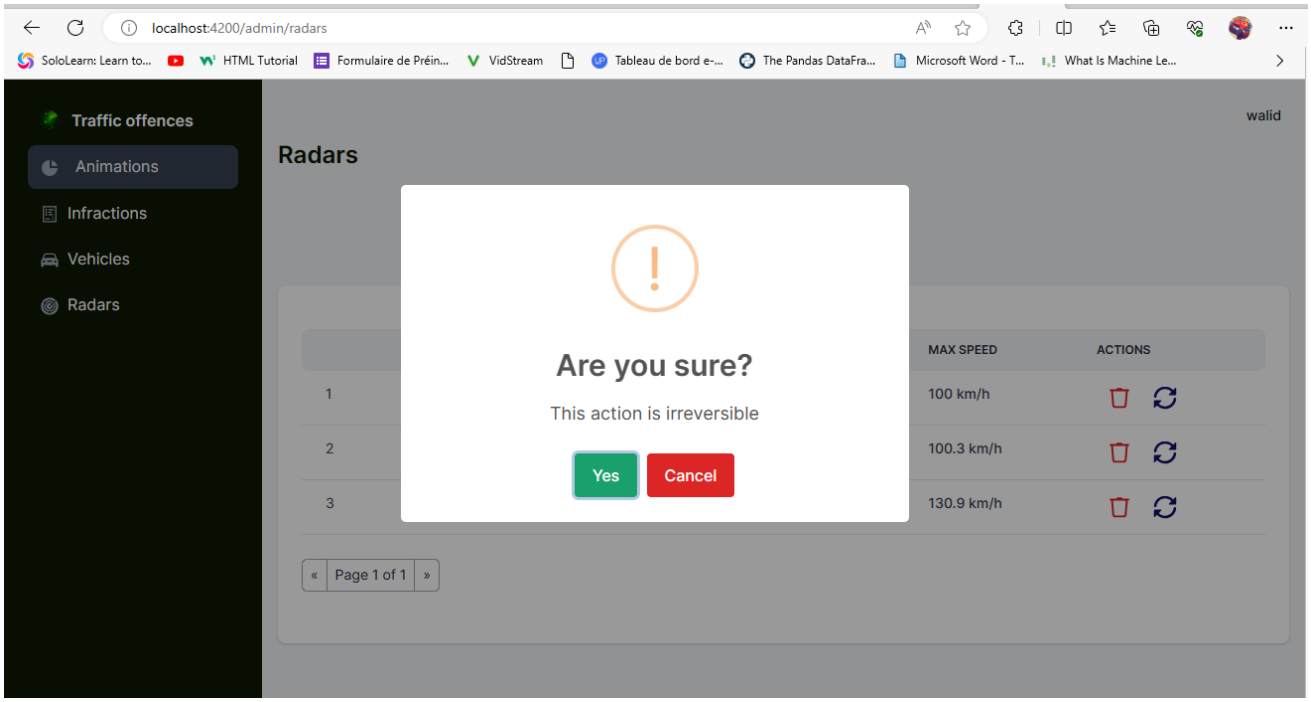


La confirmation :

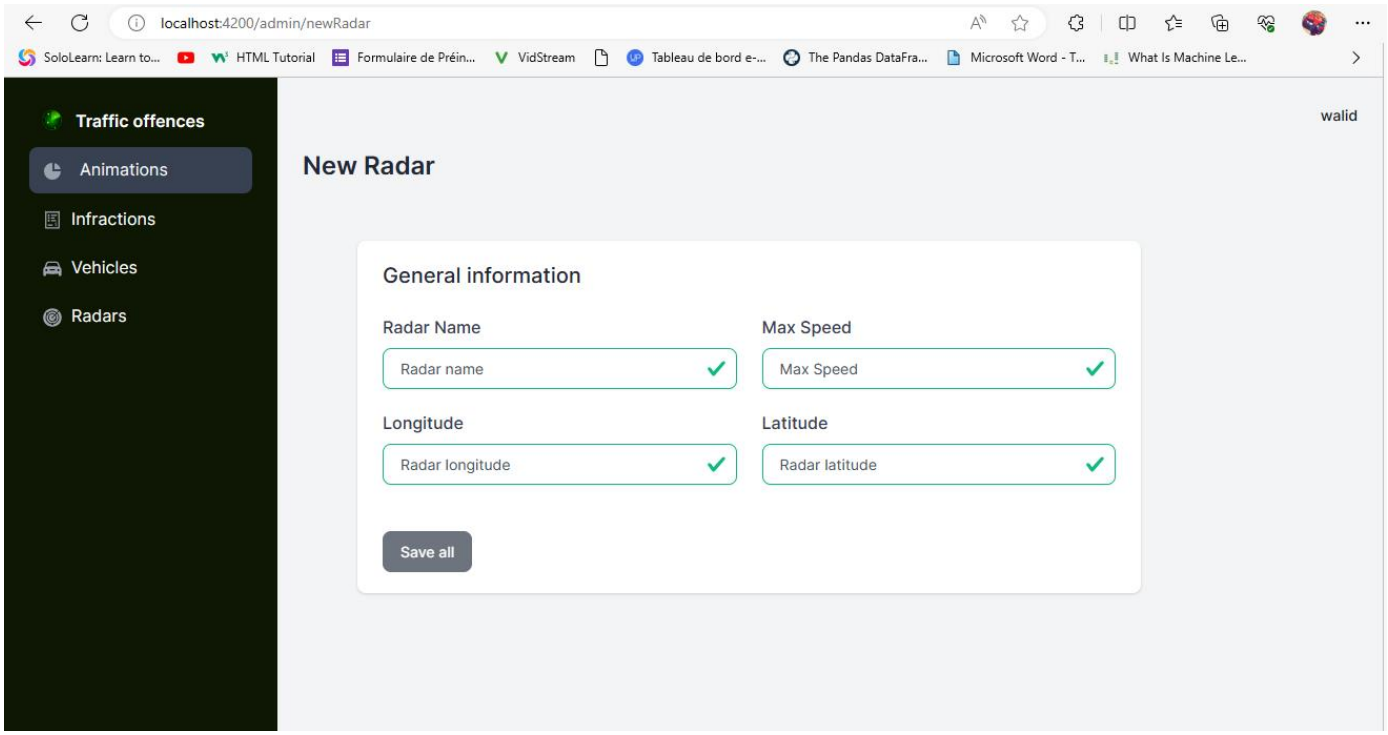




La suppression d’un rada:



Ajouter un nouveau radar :



Logout :