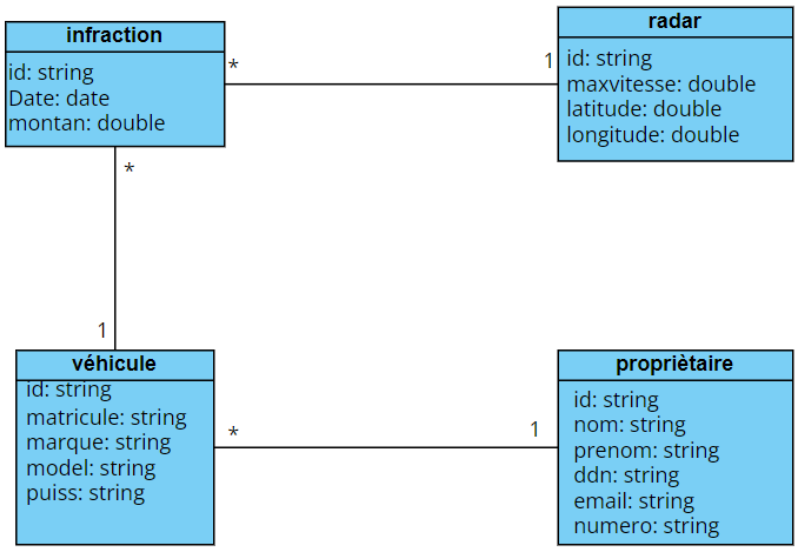
Examen Blanc

1. Diagramme de classe



2. Micro-service Radar

a. Commands

- Aggregates

```java
@Aggregate
public class RadarAggregate {
    @AggregateIdentifier
    private String id;
    private double vitesseMax;
    private double longtitude;
    private double latitude;

    public RadarAggregate() {
    }

    @CommandHandler
    public RadarAggregate(CreateRadarCommand command) {
        if (command.getVitesseMax() < 0) {
            throw new RuntimeException("Vitesse max must be positive");
        }

        AggregateLifecycle.apply(new RadarCreatedEvent(
                command.getId(),
                command.getVitesseMax(),
                command.getLongtitude(),
                command.getLatitude()
        ));
    }
```

- Controllers

```java
@RestController
@RequestMapping("/radar/commands")
@AllArgsConstructor
@CrossOrigin("*")
public class RadarCommandController {

    private CommandGateway commandGateway;
    private EventStore eventStore;

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

    @PutMapping("/updateRadar")
    public CompletableFuture<String> updateRadar(@RequestBody UpdateRadarRequestDTO request) {
        return commandGateway.send(new UpdateRadarCommand(request.getId(),
                request.getVitesseMax(), request.getLongtitude(), request.getLatitude()));
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception e) {
```

b. Queries

- Controllers

```java
@RestController
@RequestMapping("/radar/queries")
@AllArgsConstructor
@CrossOrigin("*")
public class RadarQueryController {
    private QueryGateway queryGateway;
    private RadarRepository radarRepository;

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

    @QueryHandler
    public List<Radar> on(GetAllRadarsQuery query) { return radarRepository.findAll(); }

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

- entities

```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Radar {
    @Id
    private String id;
    private double vitesseMax;
    private double longtitude;
    private double latitude;
}
```

- repositories

```
public interface RadarRepository extends JpaRepository<Radar,String> {

}
```

- services

```
@Service
@AllArgsConstructor
@Slf4j
public class RadarServiceHandler {
    private RadarRepository radarRepository;

    @EventHandler
    public void on(RadarCreatedEvent event){
        log.info("*************************");
        log.info("RadarCreatedEvent received");
        radarRepository.save(new Radar(event.getId(),event.getVitesseMax(),
                event.getLongtitude(),event.getLatitude()));
    }

    @EventHandler
    public void on(RadarUpdatedEvent event){
        log.info("*************************");
        log.info("RadarUpdatedEvent received");
        Radar radar = radarRepository.findById(event.getId()).get();
        radar.setVitesseMax(event.getVitesseMax());
        radar.setLongtitude(event.getLongtitude());
        radar.setLatitude(event.getLatitude());
        radarRepository.save(radar);
    }
```

c. Security

- Adapter Configuration

```
@Configuration
public class KeycloakAdapterConf {
    @Bean
    KeycloakSpringBootConfigResolver keycloakConfigResolver() { return new KeycloakSpringBootConfigResolver(); }
}
```

- Configuration

```
@KeycloakConfiguration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class KeycloakConfig extends KeycloakWebSecurityConfigurerAdapter {
    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.csrf().disable();
        http.authorizeRequests().anyRequest().permitAll();
//        http.authorizeRequests().anyRequest().authenticated();

    }
}
```

d. Application

```java
@SpringBootApplication
public class ExamenRadarServiceApplication {

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

        @Bean
        public CommandBus commandBus() { return SimpleCommandBus.builder().build(); }

        @Bean
        public XStream xStream() {
            XStream xStream = new XStream();

            xStream.allowTypesByWildcard(new String[] { "radarService.**" });
            return xStream;
        }

}
```

3. Micro-service Immatriculation

a. Commands

- Aggregates

agrégat propriétaire

```java
@Aggregate
public class ProprietaireAggregate {
    @AggregateIdentifier
    private String id;
    private String nom;
    private String prenom;
    private Date ddn;
    private String email;
    private String numTel;

    public ProprietaireAggregate() {
    }

    @CommandHandler
    public ProprietaireAggregate(CreateProprietaireCommand command) {
        AggregateLifecycle.apply(
                new ProprietaireCreatedEvent(
                        command.getId(),
                        command.getNom(),
                        command.getPrenom(),
                        command.getDdn(),
                        command.getEmail(),
                        command.getNumTel()
                )
```

agrégat vehicule

```java
@Aggregate
public class VehiculeAggregate {
    @AggregateIdentifier
    private String id;
    private String matricule;
    private String marque;
    private String modele;
    private String puissanceFiscale;
    private String proprietaireId;

    public VehiculeAggregate() {
    }

    @CommandHandler
    public VehiculeAggregate(CreateVehiculeCommand command) {
        AggregateLifecycle.apply(
                new VehiculeCreatedEvent(
                        command.getId(),
                        command.getMatricule(),
                        command.getMarque(),
                        command.getModele(),
                        command.getPuissanceFiscale(),
                        command.getProprietaireId()
                )
```

- Controllers

propriétaire controller

```java
@RestController
@RequestMapping("/proprietaire/commands")
@AllArgsConstructor
@CrossOrigin("*")
public class ProprietaireCommandController {
    private CommandGateway commandGateway;
    private EventStore eventStore;

    @PostMapping("/create")
    public CompletableFuture<String> createProprietaire(@RequestBody CreateProprietaireRequestDTO request) {
        return commandGateway.send(new CreateProprietaireCommand(
                UUID.randomUUID().toString(),
                request.getNom(),
                request.getPrenom(),
                request.getDdn(),
                request.getEmail(),
                request.getNumTel()
        ));
    }
```

véhicule controller

```java
@RestController
@RequestMapping("/vehicule/commands")
@AllArgsConstructor
@CrossOrigin("*")
public class VehiculeCommandController {
    private final CommandGateway commandGateway;
    private final EventStore eventStore;

    @PostMapping("/create")
    public CompletableFuture<String> create(@RequestBody CreateVehiculeRequestDTO request) {
        return commandGateway.send(new CreateVehiculeCommand(
                UUID.randomUUID().toString(),
                request.getMatricule(),
                request.getMarque(),
                request.getModele(),
                request.getPuissanceFiscale(),
                request.getProprietaireId()
        ));
    }
```

b. Queries

- Controllers

propriétaire controller

```
@RestController
@AllArgsConstructor
@RequestMapping("/proprietaire/queries")
@CrossOrigin("*")
public class ProprietaireQueryController {
    private QueryGateway queryGateway;
    private ProprietaireRepository proprietaireRepository;

    @GetMapping("/allProprietaires")
    public List<Proprietaire> getAllProprietaires() {
        return queryGateway.query(new GetAllProprietairesQuery(), ResponseTypes.multipleInstancesOf(Proprietaire
    }

    @QueryHandler
    public List<Proprietaire> on(GetAllProprietairesQuery query) { return proprietaireRepository.findAll(); }

    @GetMapping("/getProprietaire/{id}")
    public Proprietaire getProprietaire(@PathVariable String id) {
        return queryGateway.query(new GetProprietaireById(id),ResponseTypes.instanceOf(Proprietaire.class)).join
    }
```

véhicule controller

```
@RestController
@AllArgsConstructor
@RequestMapping("/vehicule/queries")
@CrossOrigin("*")
public class VehiculeQueryController {
    private QueryGateway queryGateway;
    private VehiculeRepository vehiculeRepository;

    @GetMapping("/allVehicules")
    public List<Vehicule> getAllVehicules() {
        return queryGateway.query(new GetAllVehiculesQuery(), ResponseTypes.multipleInstancesOf(Vehicule.class
    }

    @QueryHandler
    public List<Vehicule> on(GetAllVehiculesQuery query) { return vehiculeRepository.findAll(); }

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

- entities

propriétaire

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Proprietaire {
    @Id
    private String id;
    private String nom;
    private String prenom;
    private Date ddn;
    private String email;
    private String numTel;

    @OneToMany(mappedBy = "proprietaire")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<Vehicule> vehicules = new ArrayList<>();
}
```

véhicule

```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Vehicule {
    @Id
    private String id;
    private String marque;
    private String modele;
    private String matricule;
    private String puissanceFiscale;

    @ManyToOne
    private Proprietaire proprietaire;
}
```

- repositories

propriétaire

```java
public interface ProprietaireRepository extends JpaRepository<Proprietaire, String> {
}
```

véhicule

```java
public interface VehiculeRepository extends JpaRepository<Vehicule, String> {
}
```

- services

```java
@Service
@AllArgsConstructor
@Slf4j
public class ImmatriculationServiceHandler {
    private final ProprietaireRepository proprietaireRepository;
    private final VehiculeRepository vehiculeRepository;

    @EventHandler
    public void on(ProprietaireCreatedEvent event) {
        log.info("********************************");
        log.info("ProprietaireCreatedEvent received");
        Proprietaire proprietaire = new Proprietaire();
        proprietaire.setId(event.getId());
        proprietaire.setNom(event.getNom());
        proprietaire.setPrenom(event.getPrenom());
        proprietaire.setDdn(event.getDdn());
        proprietaire.setEmail(event.getEmail());
        proprietaire.setNumTel(event.getNumTel());
        proprietaireRepository.save(proprietaire);
    }
```

c. Security

- Adapter Configuration

```java
@Configuration
public class KeycloakAdapterConf {
    @Bean
    KeycloakSpringBootConfigResolver keycloakConfigResolver() { return new KeycloakSpringBootConfigResolver(); }
}
```

- Configuration

```java
@KeycloakConfiguration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class KeycloakConfig extends KeycloakWebSecurityConfigurerAdapter {
    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.csrf().disable();
        http.authorizeRequests().anyRequest().permitAll();
//        http.authorizeRequests().anyRequest().authenticated();

    }
}
```

d. Application

```java
@SpringBootApplication
public class ExamenImmatriculationServiceApplication {

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

        @Bean
        public CommandBus commandBus() { return SimpleCommandBus.builder().build(); }

        @Bean
        public XStream xStream() {
            XStream xStream = new XStream();

            xStream.allowTypesByWildcard(new String[] { "com.example.**" });
            return xStream;
        }

}
```

4. Micro-service Infraction

a. Commands

- Aggregates

```java
@Aggregate
public class InfracationAggregate {
    @AggregateIdentifier
    private String id;
    private Date date;
    private double montant;
    private String vehiculeId;
    private String radarId;

    public InfracationAggregate() {
    }

    @CommandHandler
    public InfracationAggregate(CreateInfractionCommand command) {
        AggregateLifecycle.apply(new InfractionCreatedEvent(
                command.getId(),
                command.getDate(),
                command.getMontant(),
                command.getVehiculeId(),
                command.getRadarId()
        ));
    }
```

- Controllers

```java
@RestController
@RequestMapping("/infraction/commands")
@AllArgsConstructor
@CrossOrigin("*")
public class InfracationCommandController {
    private final CommandGateway commandGateway;
    private final EventStore eventStore;

    @PostMapping("/create")
    public CompletableFuture<String> createInfraction(@RequestBody CreateInfractionRequestDTO createInfract
        return commandGateway.send(new CreateInfractionCommand(
                UUID.randomUUID().toString(),
                createInfractionRequestDTO.getDate(),
                createInfractionRequestDTO.getMontant(),
                createInfractionRequestDTO.getVehiculeId(),
                createInfractionRequestDTO.getRadarId()
        ));
    }
```

b. Queries

- Controllers

```java
@RestController
@RequestMapping("/infraction/queries")
@AllArgsConstructor
@CrossOrigin("*")
public class InfractionQueryController {
    private QueryGateway queryGateway;
    private InfractionRepository infractionRepository;

    @GetMapping("/getAllInfractions")
    public List<Infraction> getAllInfractions(){
        return queryGateway.query(new GetAllInfractionsQuery(),
                ResponseTypes.multipleInstancesOf(Infraction.class)).join();
    }

    @QueryHandler
    public List<Infraction> on(GetAllInfractionsQuery query) { return infractionRepository.fi

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

- entities

infraction

```java
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Infraction {
    @Id
    private String id;
    private Date date;
    private double montant;
    private String vehiculeId;
    private String radarId;

    @Transient
    private Vehicule vehicule;

    @Transient
    private Radar radar;
```

radar

```java
@Data @AllArgsConstructor @NoArgsConstructor
public class Radar {
    private String id;
    private double vitesseMax;
    private double longtitude;
    private double latitude;
}
```

véhicule

```java
@Data @AllArgsConstructor @NoArgsConstructor
public class Vehicule {
    private String id;
    private String marque;
    private String modele;
    private String matricule;
    private String puissanceFiscale;
    private String proprietaireId;
}
```

- repositories

```
public interface InfractionRepository extends JpaRepository<Infraction,String> {

}
```

- services

```
@Service
@AllArgsConstructor
@Slf4j
public class InfractionServiceHandler {
    private InfractionRepository infractionRepository;

    @EventHandler
    public void on(InfractionCreatedEvent event){
        log.info("**************************");
        log.info("InfractionCreatedEvent received");
        Infraction infraction = new Infraction();
        infraction.setId(event.getId());
        infraction.setDate(event.getDate());
        infraction.setMontant(event.getMontant());
        infraction.setVehiculeId(event.getVehiculeId());
        infraction.setRadarId(event.getRadarId());
        infractionRepository.save(infraction);
    }
```

c. Security

- Adapter Configuration

```
@Configuration
public class KeycloakAdapterConf {
    @Bean
    KeycloakSpringBootConfigResolver keycloakConfigResolver() { return new KeycloakSpringBootConfigResolver(); }
}
```

- Configuration

```
@KeycloakConfiguration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class KeycloakConfig extends KeycloakWebSecurityConfigurerAdapter {
    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.csrf().disable();
        http.authorizeRequests().anyRequest().permitAll();
//        http.authorizeRequests().anyRequest().authenticated();
```

d. Application

```java
@SpringBootApplication
public class ExamenInfractionServiceApplication {

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

        @Bean
        public CommandBus commandBus() { return SimpleCommandBus.builder().build(); }

        @Bean
        public XStream xStream() {
            XStream xStream = new XStream();

            xStream.allowTypesByWildcard(new String[] { "com.example.**" });
            return xStream;
        }

}
```

5. Discovery Service

```java
@SpringBootApplication
@EnableEurekaServer
public class DiscoveryServiceApplication {

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

}
```

6. Gateway service

Configuration

```java
@Configuration
public class GatewayConfig {
    @Bean
    DiscoveryClientRouteDefinitionLocator discoveryClientRouteDefinitionLocator (ReactiveDiscoveryClient rdc, Disc
        return new DiscoveryClientRouteDefinitionLocator(rdc,dlp);
    }
}
```

Application

```java
@SpringBootApplication
public class GatwayServiceApplication {

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

}
```