Ecole Normale Supérieure de l'Enseignement Technique Mohammedia

Université Hassan II de Casablanca

ENSET

ا لمدرسة العليا لأساتذة التعليم التقني المحمدية

جامعة الحسن الثاني بالدار البيضاء

DÉPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Rapport

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Examen Blanc

Réalisé par :

Soukaina EL KAMOUNI

Année Universitaire : 2022-2023

I. Radar Service:

A. Radar Commands:

1. Aggregate:

```
@Aggregate
public class RadarAggregate {
    @AggregateIdentifier
    private String radarId;
    private double maxSpeed;
    private double latitude;
    private double longitude;

    public RadarAggregate() {
        //Required by Axon
    }

    @CommandHandler
    public RadarAggregate(CreateRadarCommand
command){
        if (command.maxSpeed < 0) {
            throw new IllegalArgumentException("Max
Speed cannot be negative");
        }
        AggregateLifecycle.apply(new
RadarCreatedEvent(
                command.getId(),
                command.getMaxSpeed(),
                command.getLatitude(),
                command.getLongitude()));
    }

    @CommandHandler
    public RadarAggregate(PassedVehiculeRadarCommand
command){
        AggregateLifecycle.apply(new
RadarCatchSpeedEvent(
                command.getId(),
```

```
                command.getMatricule(),
                command.getVehicleSpeed(),
                command.getRadarId(),
                command.getRadarSpeed()
                ));
    }

    @EventSourcingHandler //change the state of the
aggregate
    public void on(RadarCreatedEvent event){
        this.radarId = event.getId();
        this.maxSpeed = event.getMaxSpeed();
        this.latitude = event.getLatitude();
        this.longitude = event.getLongitude();
    }
}
```

2. Controller:

```
@RestController
@RequestMapping("/command/radar")
@AllArgsConstructor
@Service
public class RadarCommandController {
    private CommandGateway commandGateway;

    @PostMapping(path = "/create")
    public CompletableFuture<String>
createRadar(@RequestBody CreateRadarRequestDTO
createRadarRequestDTO) {
        CompletableFuture<String> response =
commandGateway.send(new CreateRadarCommand(
                UUID.randomUUID().toString(),
                createRadarRequestDTO.getMaxSpeed(),
                createRadarRequestDTO.getLatitude(),
                createRadarRequestDTO.getLongitude()
        ));
        return response;
```

```
    }

    @PostMapping(path = "/passingVehicles")
    public ResponseEntity<String>
passingVehicles(@RequestBody PassingVehiculeDTO
passingVehiculeDTO) {
        CompletableFuture<String> response =
commandGateway.send(new PassedVehiculeRadarCommand(
                UUID.randomUUID().toString(),
                passingVehiculeDTO.getMatricule(),
                passingVehiculeDTO.getVehicleSpeed(),
                passingVehiculeDTO.getRadarId(),
                passingVehiculeDTO.getRadarSpeed()
        ));
        return new ResponseEntity<>(response.join(),
HttpStatus.OK);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String>
handleException(Exception e) {
        return new
ResponseEntity<String>(e.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

3. Main App:

```
@SpringBootApplication
public class RadarCommandApplication {

  public static void main(String[] args) {

SpringApplication.run(RadarCommandApplication.class,
args);
  }
  @Bean
```

```
  public SimpleCommandBus axonServerCommandBus(){
     return SimpleCommandBus.builder().build();
  }

}
```

4. Application Properties:

```
server.port=8080
spring.application.name=radar-command-side-service
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

B. Radar Query:

1. Controller:

```
@RestController
@RequestMapping("/query/radar")
@AllArgsConstructor
public class RadarRestController {
   private QueryGateway queryGateway;

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

2. Entities:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Radar {
   @Id private String id;
```

```
    private double maxSpeed;
    private double latitude;
    private double longitude;
}
```

3. Repositories:

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

4. Services:

```java
@Service
@AllArgsConstructor
public class RadarServiceHandler {
    private RadarRepository radarRepository;
    @EventHandler
    @Transactional
    public void on(RadarCreatedEvent event){
        Radar radar = new Radar();
        radar.setId(event.getId());
        radar.setLatitude(event.getLatitude());
        radar.setLongitude(event.getLongitude());
        radar.setMaxSpeed(event.getMaxSpeed());
        radarRepository.save(radar);
    }
    @QueryHandler
    public List<Radar> on(FindAllRadars query){
        return radarRepository.findAll();
    }
}
```

5. Application Properties:

```
server.port=8083

spring.datasource.url=jdbc:h2:mem:radar-db

spring.h2.console.enabled=true

spring.application.name=radar-query-side-service

spring.cloud.discovery.enabled=true

eureka.instance.prefer-ip-address=true
```

II. Immatriculation Service:

A. Immatriculation Query:

1. Controllers:

- Owner Query Controller:

```
@RestController
@RequestMapping("/query/owner")
@AllArgsConstructor
@Service
public class OwnerQueryController {
   private QueryGateway queryGateway;


   @GetMapping(path = "/")
   public List<Owner> getOwner() {
       return queryGateway.query(new GetOwners(),
ResponseTypes.multipleInstancesOf(Owner.class)).join(
);
   }

   @GetMapping(path = "/{id}")
   public Owner getOwner(@PathVariable String id) {
       return queryGateway.query(new GetOwner(id),
Owner.class).join();
   }
```

```
    @GetMapping(path = "/infraction/{id}")
    public List<InfractionResponseDTO>
getInfractionsByOwnerId(@PathVariable String id) {
        List<Vehicule> vehicules =
queryGateway.query(new GetVehiculesByOwnerId(id),
ResponseTypes.multipleInstancesOf(Vehicule.class)).jo
in();
        List<InfractionResponseDTO>
infractionResponseDTOS = new ArrayList<>();
        for (Vehicule vehicule : vehicules) {

infractionResponseDTOS.addAll(queryGateway.query(new
GetInfractionsByVehicle(vehicule.getMatricule()),
ResponseTypes.multipleInstancesOf(InfractionResponseD
TO.class)).join());
        }
        return infractionResponseDTOS;
    }
}
```

- Vehicle Query Controller:

```
@RestController
@RequestMapping("/query/vehicule")
@AllArgsConstructor
@Service
public class VehiculeQueryController {
    private QueryGateway queryGateway;


    @GetMapping(path = "/")
    public List<Vehicule> getVehicules() {
        return queryGateway.query(new GetVehicules(),
ResponseTypes.multipleInstancesOf(Vehicule.class)).jo
in();
    }

    @GetMapping(path = "/{id}")
```

```
    public Owner getVehicule(@PathVariable String id)
{
        return queryGateway.query(new
GetVehicule(id), Owner.class).join();
    }

    @GetMapping(path = "/byMartricule/{matricule}")
    public List<InfractionResponseDTO>
getVehiculeByMatricule(@PathVariable String
matricule) {
        return queryGateway.query(new
GetInfractionsByVehicle(matricule),
ResponseTypes.multipleInstancesOf(InfractionResponseD
TO.class)).join();
    }
}
```

2. Entities:

- Owner:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Owner {
    @Id
    private String id;
    private String name;
    private Date dateOfBirth;
    private String email;
    @OneToMany(mappedBy = "owner")
    private List<Vehicule> vehicules;
}
```

- Vehicle:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Vehicule {
    @Id
    private String id;
```

```
    private String matricule;
    private String marque;
    private String modele;
    private int puissance;
    @ManyToOne
    @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY)
    private Owner owner;
    private String proprietaireId;
}
```

3. Repositories:

- Owner Repo:

```
public interface OwnerRepository extends
JpaRepository<Owner, String> {
}
```

- Vehicle Repo:

```
public interface VehiculeRepository extends
JpaRepository<Vehicule, String> {
    List<Vehicule> findByProprietaireIdEquals(String
id);
}
```

4. Services:

- Owner Service handler:

```
@Service
@AllArgsConstructor
@Slf4j
public class OwnerServiceHandler {
    private OwnerRepository ownerRepository;
    @EventHandler
    @Transactional
    public void on(OwnerCreatedEvent event) {
        log.info("OwnerCreatedEvent: {}", event);
```

```
        Owner owner = new Owner();
        owner.setId(event.getId());
        owner.setName(event.getName());
        owner.setDateOfBirth(event.getDateOfBirth());
        owner.setEmail(event.getEmail());
        ownerRepository.save(owner);
    }
    @QueryHandler
    public List<Owner> on(GetOwners query) {
        return ownerRepository.findAll();
    }
    @QueryHandler
    public Owner on(GetOwner query) {
        return
ownerRepository.findById(query.getId()).get();
    }
}
```

- Vehicle Service handler:

```
@Service
@AllArgsConstructor
@Slf4j
public class VehiculeServiceHandler {
    private VehiculeRepository vehiculeRepository;
    private OwnerRepository ownerRepository;
    @EventHandler
    @Transactional
    public void on(VehiculeCreatedEvent event) {
        log.info("VehiculeCreatedEvent: {}", event);
        Owner owner =
ownerRepository.findById(event.getProprietaire()).ge
t();
        Vehicule vehicule = new Vehicule();
        vehicule.setId(event.getId());
        vehicule.setMatricule(event.getMatricule());
        vehicule.setMarque(event.getMarque());
        vehicule.setModele(event.getModele());
        vehicule.setPuissance(event.getPuissance());
```

```
        vehicule.setOwner(owner);
        vehicule.setProprietaireId(owner.getId());
        vehiculeRepository.save(vehicule);
    }
    @QueryHandler
    public List<Vehicule> on(GetVehicules query) {
        return vehiculeRepository.findAll();
    }
    @QueryHandler
    public Vehicule on(GetVehicule query) {
        return
vehiculeRepository.findById(query.getId()).get();
    }
    @QueryHandler
    public List<Vehicule> on(GetVehiculesByOwnerId
query) {
        return
vehiculeRepository.findByProprietaireIdEquals(query.
getId());
    }
}
```

5. Application Properties:

```
server.port=8082
spring.datasource.url=jdbc:h2:mem:vehicle-registrati
on-db
spring.h2.console.enabled=true
spring.application.name=vehicle-registration-query-s
ide-service
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

B. Immatriculation Command:

1. Aggregate:

- Owner Aggregate:

```
@Aggregate
```

```
public class OwnerAggregate {
    @AggregateIdentifier
    private String id;
    private String name;
    private Date dateOfBirth;
    private String email;

    public OwnerAggregate() {
        // Required by Axon
    }
    @CommandHandler
    public OwnerAggregate(CreateOwnerCommand command)
{
        if (command.getName() == null ||
command.getName().isEmpty()) {
            throw new IllegalArgumentException("Name
cannot be empty");
        }
        AggregateLifecycle.apply(new
OwnerCreatedEvent(
                command.getId(),
                command.getName(),
                command.getDateOfBirth(),
                command.getEmail()));
    }
    @EventSourcingHandler
    public void on(OwnerCreatedEvent event) {
        this.id = event.getId();
        this.name = event.getName();
        this.dateOfBirth = event.getDateOfBirth();
        this.email = event.getEmail();
    }
}
```

- Vehicle Aggregate:

```
@Aggregate
public class VehiculeAggregate {
    @AggregateIdentifier
```

```
    private String id;
    private String matricule;
    private String marque;
    private String modele;
    private int puissance;
    private String proprietaire;

    public VehiculeAggregate() {
        // Required by Axon
    }
    @CommandHandler
    public VehiculeAggregate(CreateVehiculeCommand
command) {
        if (command.getMatricule() == null ||
command.getMatricule().isEmpty()) {
            throw new
IllegalArgumentException("Matricule cannot be
empty");
        }
        AggregateLifecycle.apply(new
VehiculeCreatedEvent(
                command.getId(),
                command.getMatricule(),
                command.getMarque(),
                command.getModele(),
                command.getPuissance(),
                command.getProprietaire()));
    }
    @EventSourcingHandler
    public void on(VehiculeCreatedEvent event) {
        this.id = event.getId();
        this.matricule = event.getMatricule();
        this.marque = event.getMarque();
        this.modele = event.getModele();
        this.puissance = event.getPuissance();
        this.proprietaire = event.getProprietaire();
    }
}
```

2. Controllers:

- Owner Command Controller:

```
@RestController
@RequestMapping("/command/owner")
@AllArgsConstructor
@Service
public class OwnerCommandController {
   private CommandGateway commandGateway;
   @PostMapping(path = "/create")
   public CompletableFuture<String>
createOwner(@RequestBody CreateOwnerRequestDTO
createOwnerRequestDTO) {
       CompletableFuture<String> response =
commandGateway.send(new CreateOwnerCommand(
               UUID.randomUUID().toString(),
               createOwnerRequestDTO.getName(),

createOwnerRequestDTO.getDateOfBirth(),
               createOwnerRequestDTO.getEmail()
       ));
       return response;
   }
}
```

- Vehicle Command Controller:

```
@RestController
@RequestMapping("/command/vehicule")
@AllArgsConstructor
@Service
public class VehiculeCommandController {
   private CommandGateway commandGateway;
   @PostMapping(path = "/create")
   public CompletableFuture<String>
createVehicule(@RequestBody CreateVehiculeRequestDTO
createVehiculeRequestDTO) {
       CompletableFuture<String> response =
commandGateway.send(new CreateVehiculeCommand(
```

```
                UUID.randomUUID().toString(),

createVehiculeRequestDTO.getMatricule(),
                createVehiculeRequestDTO.getMarque(),
                createVehiculeRequestDTO.getModele(),

createVehiculeRequestDTO.getPuissance(),

createVehiculeRequestDTO.getProprietaire()
        ));
        return response;
    }
}
```

3. Main App:

```
@SpringBootApplication
public class ImmatriculationCommandApplication {

  public static void main(String[] args) {

SpringApplication.run(ImmatriculationCommandApplicat
ion.class, args);
  }
  @Bean
  public CommandBus commandBus(){
     return SimpleCommandBus.builder().build();
  }

}
```

4. Application Properties:

```
server.port=8081
spring.application.name=vehicle-registration-command
-side-service
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

III. Infraction Service:

A. Infraction Command:

1. Aggregate:

```
@Aggregate
public class InfractionAggregate {
   @AggregateIdentifier
   private String id;
   private String matricule;
   private double vehicleSpeed;
   private Date date;
   private String radarId;
   private double maxSpeedAllowed;

   public InfractionAggregate() {
   }
   @CommandHandler
   public
InfractionAggregate(CreateInfractionCommand command)
{
       if (command.getMatricule() == null ||
command.getMatricule().isEmpty()) {
           throw new
IllegalArgumentException("Matricule cannot be
empty");
       }
       AggregateLifecycle.apply(new
InfractionCreatedEvent(
               command.getId(),
               command.getMatricule(),
               command.getVehicleSpeed(),
               command.getDate(),
               command.getRadarId(),
               command.getMaxSpeedAllowed()));
   }
   @EventSourcingHandler
   public void on(InfractionCreatedEvent event) {
       this.id = event.getId();
```

```
        this.matricule = event.getMatricule();
        this.vehicleSpeed = event.getVehicleSpeed();
        this.date = event.getDate();
        this.radarId = event.getRadarId();
        this.maxSpeedAllowed =
event.getMaxSpeedAllowed();
    }
}
```

2. Controller:

```
@RestController
@RequestMapping("/command/infraction")
@AllArgsConstructor
@Service
public class InfractionCommandController {
    private CommandGateway commandGateway;
    @PostMapping("/create")
    public CompletableFuture<String>
create(@RequestBody InfractionCreationRequestDTO
infractionCreationRequestDTO){
        CompletableFuture<String> response =
commandGateway.send(new CreateInfractionCommand(
                UUID.randomUUID().toString(),

infractionCreationRequestDTO.getMatricule(),

infractionCreationRequestDTO.getSpeed(),

infractionCreationRequestDTO.getDate(),

infractionCreationRequestDTO.getRadarId(),

infractionCreationRequestDTO.getRadarSpeed()
        ));
        return response;
    }
}
```

3. Main App:

```
@SpringBootApplication
public class InfractionCommandApplication {
  public static void main(String[] args) {

SpringApplication.run(InfractionCommandApplication.c
lass, args);
  }
  @Bean
  CommandBus commandBus(){
     return SimpleCommandBus.builder().build();
  }
}
```

4. Application Properties:

```
server.port=8090
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:contravention-db
spring.application.name=contravention-command-side-s
ervice
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

B. Infraction Query:

1. Controllers:

```
@RestController
@RequestMapping("/query/infraction")
@AllArgsConstructor
@Service
public class InfractionQueryHandler {
   private QueryGateway queryGateway;
   @GetMapping("/All")
```

```
    public CompletableFuture<List<Infraction>>
getAll(){
        return queryGateway.query(new
GetAllInfractions(),
ResponseTypes.multipleInstancesOf(Infraction.class));
    }
    @GetMapping("/byIdProprietaire/{id}")
    public CompletableFuture<List<Infraction>>
getByIdProprietaire(@PathVariable String id){
        return queryGateway.query(new
GetInfractionsByProprietaire(id),
ResponseTypes.multipleInstancesOf(Infraction.class));
    }
    @GetMapping("/byIdVehicule/{id}")
    public CompletableFuture<List<Infraction>>
getByIdVehicule(@PathVariable String id){
        return queryGateway.query(new
GetInfractionsByVehicle(id),
ResponseTypes.multipleInstancesOf(Infraction.class));
    }
    @GetMapping("/byOwnerId/{id}")
    public CompletableFuture<List<Infraction>>
getByOwnerId(@PathVariable String id){
        return queryGateway.query(new
GetInfractionsByOwnerId(id),
ResponseTypes.multipleInstancesOf(Infraction.class));
    }
    @GetMapping("/byId/{id}")
    public CompletableFuture<Infraction>
getById(@PathVariable String id){
        return queryGateway.query(new
GetInfraction(id),
ResponseTypes.instanceOf(Infraction.class));
    }
}
```

2. Entities:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Infraction {
   @Id
   private String id;
   private String matricule;
   private double vehicleSpeed;
   private Date dateInfraction;
   private String radarId;
   private double maxSpeedAllowed;
   private double amande;
}
```

3. Repository:

```
public interface InfractionRepository extends
JpaRepository<Infraction,String> {
   public List<Infraction>
findByMatriculeEquals(String matricule);
}
```

4. Service:

```
@Service
@AllArgsConstructor
@Slf4j
public class InfractionServiceHandler {
   private InfractionRepository
infractionRepository;
   @EventHandler
   public void on(InfractionCreatedEvent event) {
       log.info("InfractionCreatedEvent: {}");
       if(event.getMaxSpeedAllowed()<
event.getVehicleSpeed()){
```

```
            Infraction infraction = new Infraction();
            infraction.setId(event.getId());

infraction.setMatricule(event.getMatricule());

infraction.setDateInfraction(event.getDate());

infraction.setVehicleSpeed(event.getVehicleSpeed());

infraction.setMaxSpeedAllowed(event.getMaxSpeedAllow
ed());

infraction.setAmande(300+300*(event.getVehicleSpeed(
) -
event.getMaxSpeedAllowed())/event.getMaxSpeedAllowed
());
            infractionRepository.save(infraction);
        }
    }
    @EventHandler
    public void on (RadarCatchSpeedEvent event){
        log.info("RadarCatchSpeedEvent: {}");
        if(event.getRadarSpeed()<
event.getVehiculeSpeed()){
            Infraction infraction = new Infraction();
            infraction.setId(event.getId());

infraction.setMatricule(event.getMatricule());

infraction.setDateInfraction(event.getDate());

infraction.setVehicleSpeed(event.getVehiculeSpeed())
;

infraction.setMaxSpeedAllowed(event.getRadarSpeed())
;
```

```
infraction.setAmande(300+300*(event.getVehiculeSpeed
() - event.getRadarSpeed())/event.getRadarSpeed());
            infractionRepository.save(infraction);
        }
    }
    @QueryHandler
    public List<InfractionResponseDTO>
on(GetInfractionsByVehicle query) {
        List<Infraction> infractions =
infractionRepository.findByMatriculeEquals(query.get
Id());
        List<InfractionResponseDTO>
infractionResponseDTOS = new ArrayList<>();
        for (Infraction infraction: infractions) {
            InfractionResponseDTO
infractionResponseDTO = new InfractionResponseDTO();

infractionResponseDTO.setId(infraction.getId());

infractionResponseDTO.setMatricule(infraction.getMat
ricule());

infractionResponseDTO.setDateInfraction(infraction.g
etDateInfraction());

infractionResponseDTO.setVehicleSpeed(infraction.get
VehicleSpeed());

infractionResponseDTO.setMaxSpeedAllowed(infraction.
getMaxSpeedAllowed());

infractionResponseDTO.setAmande(infraction.getAmande
());

infractionResponseDTOS.add(infractionResponseDTO);
        }
        return infractionResponseDTOS;
```

```
    }
}
```

5. Application Properties:

```
server.port=8091
spring.application.name=contravention-query-side-ser
vice
spring.cloud.discovery.enabled=true
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:contraventions-db
eureka.instance.prefer-ip-address=true
```

IV. Gateway Service:

A. Application Properties:

```
server.port=8888
spring.application.name=GATEWAY-SERVICE
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

B. Main App:

```
@SpringBootApplication
public class GatewayApplication {
  public static void main(String[] args) {
     SpringApplication.run(GatewayApplication.class,
args);
  }
  @Bean
  DiscoveryClientRouteDefinitionLocator
discoveryClientRouteDefinitionLocator(ReactiveDiscov
eryClient rdc, DiscoveryLocatorProperties dlp) {
     return new
DiscoveryClientRouteDefinitionLocator(rdc, dlp);
```

```
  }
  @Bean
  public WebFilter corsFilter() {
     return (ServerWebExchange ctx, WebFilterChain
chain) -> {
        ServerHttpRequest request =
ctx.getRequest();
        if (CorsUtils.isCorsRequest(request)) {
           ServerHttpResponse response =
ctx.getResponse();
           HttpHeaders headers =
response.getHeaders();

headers.add("Access-Control-Allow-Origin", "*");

headers.add("Access-Control-Allow-Methods", "GET,
PUT, POST, DELETE, OPTIONS");
           headers.add("Access-Control-Max-Age",
"3600");

headers.add("Access-Control-Allow-Headers",
"x-requested-with, authorization, Content-Type,
Authorization, credential, X-XSRF-TOKEN");
           if (request.getMethod() ==
HttpMethod.OPTIONS) {
              response.setStatusCode(HttpStatus.OK);
              return Mono.empty();
           }
        }
        return chain.filter(ctx);
     };
  }
}
```

V. Eureka Discovery Service:

A. Application Properties:

```
server.port=8761
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
spring.application.name=EUREKA-SERVICE
```

For the main, we just add the following annotation:

```
@EnableEurekaServer
```