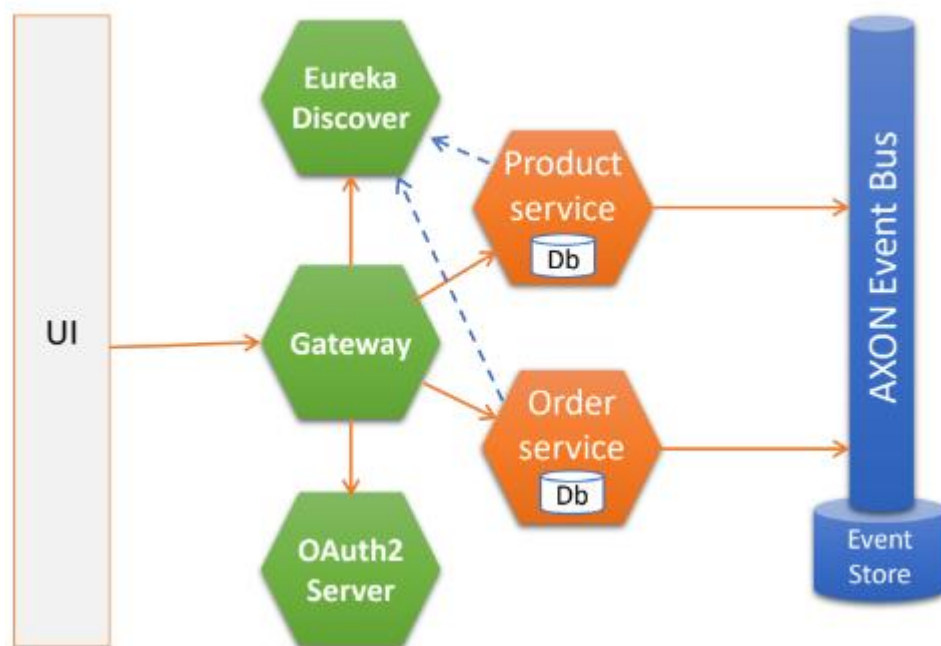


COMPTE-RENDU : EXAMEN DE SYSTEMES DISTRIBUES

Filière : « Ingénierie Informatique : Big Data et Cloud
Computing » II-BDCC



Réalisé par :

Khadija BENJILALI

Encadré par :

Pr. Mohamed YOUSSEFI




Année Universitaire : 2022-2023

Sommaire

Travail à faire.....	3
Travail à faire :	3
PARTIE 1 : Structure et Architecture du projet	4
1. Architecture technique du projet	4
2. Diagramme de classe global du projet	5
3. Maven Dependencies de radar-service :	5
4. Common-api :	5
5. BaseCommand	6
6. BaseEvent	6
PARTIE 2 : Création de radar-service.....	6
1. Structure de radar-service :	6
2. Partie Ecriture:	6
3. Partie Lecture:	10

Travail à faire

On souhaite créer un système distribué basé sur les micro-services en utilisant une architecture pilotée par les événements respectant les deux patterns Event Sourcing et CQRS. Cette application devrait permettre de gérer des commandes de produits appartenant à des clients. L'application se compose des micro-services fonctionnels suivants :

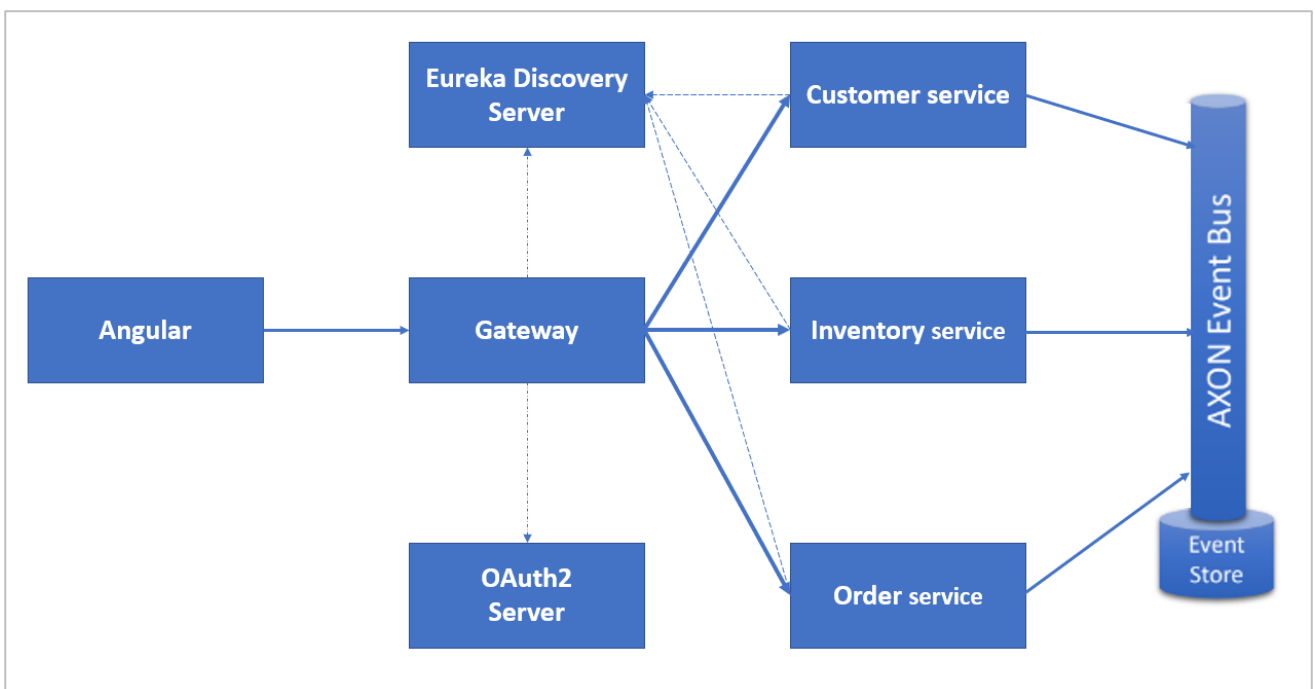
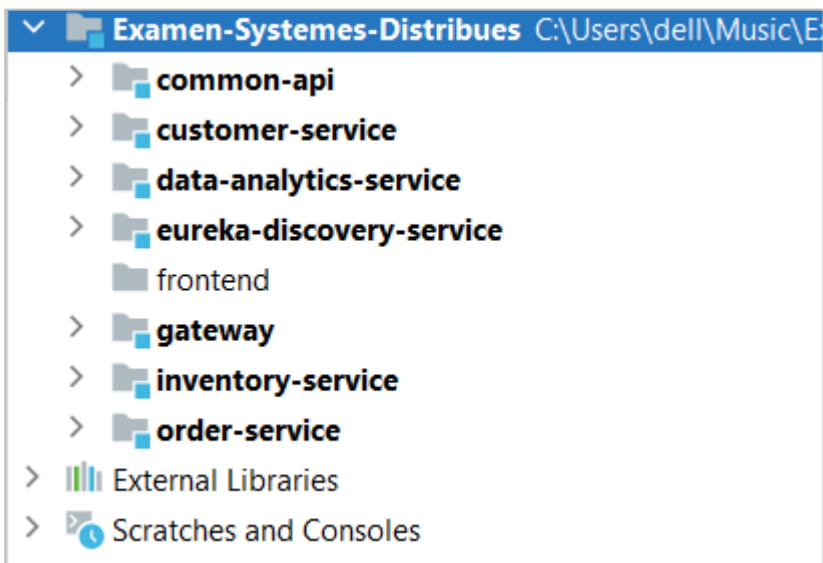
-  Le micro-service « Customer-Service » qui permet de gérer les clients. Chaque client est défini par son id, son nom, son adresse, son adresse mail et son téléphone
-  Le micro-service Inventory-Service qui permet de gérer les produits. Chaque produit appartient à une catégorie. Une catégorie est définie par son id, son nom, et sa description. Un produit est défini par son id, son nom, son prix, sa quantité en stock et son état (Disponible, Rupture, Production, Abandon)
-  Le micro-service « Order-Service » qui permet de gérer des commandes. Chaque commande concerne un client et contient plusieurs lignes de commandes. Chaque ligne de commande concerne un produit. Une ligne de commande est identifiée par son id, la quantité de produit, le prix unitaire du produit, la remise. Une commande est définie par son id, la date de commande, la date de livraison, adresse livraison, son statut (Créé, Activée, Abandonnée, en Préparation, expédiée, livrée)

Travail à faire :

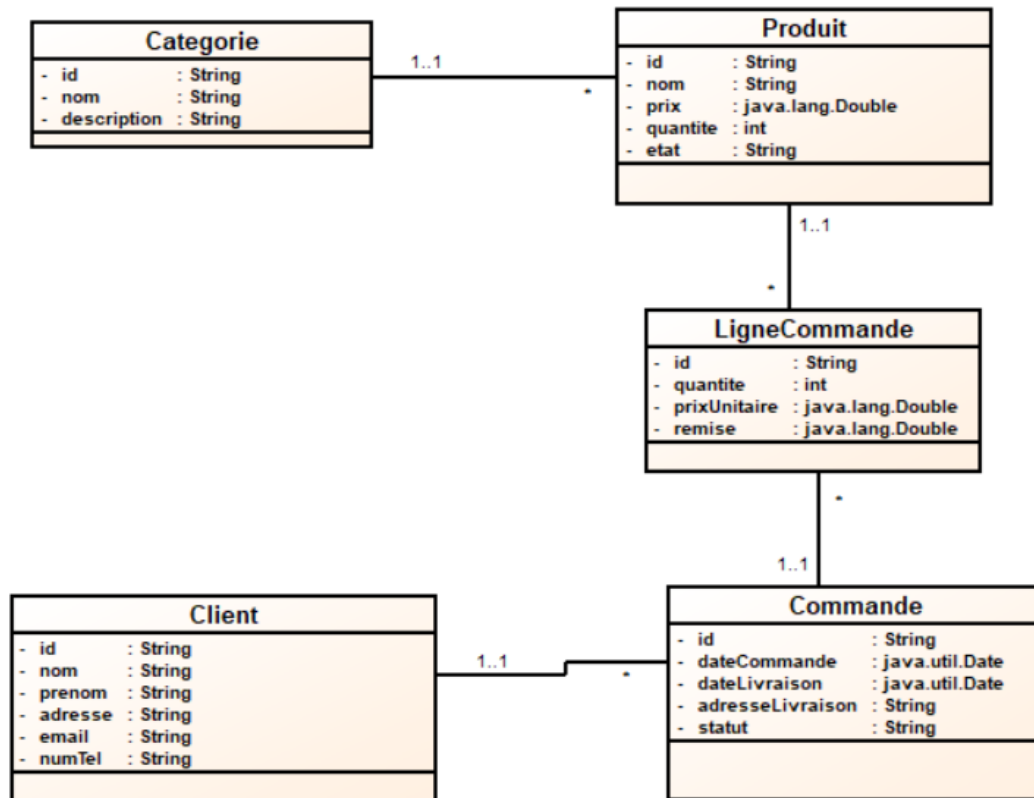
- **Établir une architecture technique du projet**
- **2. Établir un diagramme de classe global du projet**
- **3. Déployer le serveur AXON Server ou KAFKA Broker**
- **4. Développer le micro-service Customer-Service**
- **5. Développer le micro-service Inventory-Service**
- **6. Développer le micro-service Order-Service**
- **7. Mettre en place les services techniques de l'architecture micro-service (Gateway, Eureka ou Consul Discovery service, Config Service)**

PARTIE 1 : Structure et Architecture du projet

1. Architecture technique du projet



2. Diagramme de classe global du projet



3. Maven Dependencies de radar-service :

Au démarrage on ne va pas utiliser le serveur AXON

```
<dependency>
  <groupId>org.axonframework</groupId>
  <artifactId>axon-spring-boot-starter</artifactId>
  <version>4.6.2</version>
  <exclusions>
    <exclusion>
      <groupId>org.axonframework</groupId>
      <artifactId>axon-server-connector</artifactId>
    </exclusion>
  </exclusions>
</dependency>
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.6.12</version>
</dependency>
```

```
<dependency>
  <groupId>com.example</groupId>
  <artifactId>common-api</artifactId>
  <version>0.0.1-SNAPSHOT</version>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>8.0.31</version>
  <scope>runtime</scope>
</dependency>
```

4. Common-api :

En plus des modules représentant les différents micro-services, le projet utilise un module « common-api » sous forme d'un projet maven qui déclare les composants communs aux différents projets comme les Commandes, les Événements, les Queries, les DTOs, etc.

5. BaseCommand

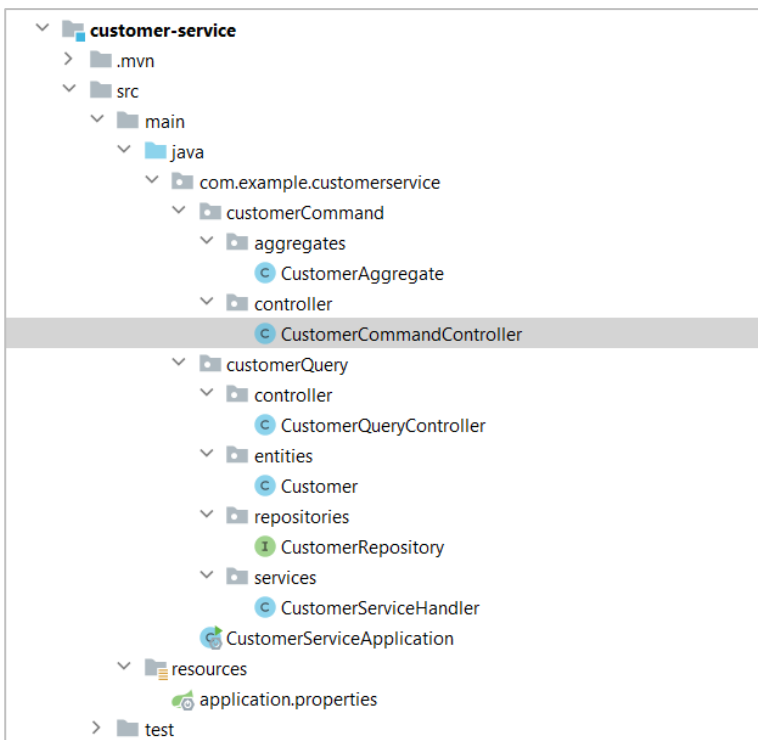
```
public class BaseCommand<T> {  
    @TargetAggregateIdentifier  
    private T id ;  
  
    public BaseCommand(T id) { this.id = id; }  
  
    public T getId() { return id; }  
}
```

6. BaseEvent

```
public class BaseEvent <T>{  
    private T id ;  
    public BaseEvent(T id ) { this.id =id ; }  
    public T getId() { return id; }  
}
```

PARTIE 2 : Création de customer-service

1. Structure de radar-service :



2. Partie Ecriture:

- [Aggregate Radar](#)

```

@Aggregate
public class CustomerAggregate {

    @AggregateIdentifier
    private String id ;
    private String nom ;
    private String adresse ;
    private String email ;
    private String telephone ;

    public CustomerAggregate() {
    }

    @CommandHandler
    public CustomerAggregate(CreateCustomerCommand command){
        if(command.getNom().isEmpty()){
            throw new RuntimeException("Le nom ne peut pas etre vide");
        }
        AggregateLifecycle.apply(new CustomerCreatedEvent(
            command.getId(),
            command.getNom(),
            command.getAdresse(),
            command.getEmail(),
            command.getTelephone()
        ));
    }
}

```

```

@EventSourcingHandler
public void on(CustomerCreatedEvent event) {
    this.id = event.getId();
    this.nom = event.getNom();
    this.adresse = event.getAdresse();
    this.email = event.getEmail();
    this.telephone = event.getTelephone();
}

@CommandHandler
public void handle(UpdateCustomerCommand command) {
    AggregateLifecycle.apply(new CustomerUpdatedEvent(
        command.getId(),
        command.getNom(),
        command.getAdresse(),
        command.getEmail(),
        command.getTelephone()
    ));
}

@EventSourcingHandler
public void on(CustomerUpdatedEvent event) {
    this.nom = event.getNom();
    this.adresse = event.getAdresse();
    this.email = event.getEmail();
    this.telephone = event.getTelephone();
}

```

- Ajouter un customer à partir Postman

```

@PostMapping("/createCustomer")
public CompletableFuture<String> createCustomer(@RequestBody CreateCustomerRequestDTO request)
{
    return commandGateway.send(
        new CreateCustomerCommand(
            UUID.randomUUID().toString(),
            request.getNom(),
            request.getAdresse(),
            request.getEmail(),
            request.getTelephone()
        ));
}

```

http://localhost:8081/customer/commands/createCustomer

POST

http://localhost:8081/customer/commands/createCustomer

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "nom": "Imane",
3   "adresse": "Casablanca",
4   "email": "Imane@gmail.com",
5   "telephone": "0654240978"
6 }

```

Body Cookies Headers (5) Test Results

Status: 200 OK

Pretty

Raw

Preview

Visualize

Text

1 60ec5ab1-3abc-4c21-847b-412a85af609f

- Modifier un customer à partir Postman

```

@PutMapping("/updateCustomer")
public CompletableFuture<String> updateCustomer(@RequestBody UpdateCustomerRequestDTO request){
    return commandGateway.send(
        new UpdateCustomerCommand(
            request.getId(),
            request.getNom(),
            request.getAdresse(),
            request.getEmail(),
            request.getTelephone()
        ));
}

```


http://localhost:8081/customer/commands/updateCustomer

PUT http://localhost:8081/customer/commands/updateCustomer

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

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

```

1  {
2    "id": "551085b5-9c95-43c7-89ec-50a6cd718000",
3    "nom": "Ahlam",
4    "adresse": "Rabat",
5    "email": "Ahlam@gmail.com",
6    "telephone": "000000000"
  }

```

Body Cookies Headers (4) Test Results Status: 200 OK

Pretty Raw Preview Visualize Text

1 |

- Event Store de Radar

```

@GetMapping("/eventStore/{customerId}")
public Stream getEventsForAccount(@PathVariable(value = "customerId") String customerId)
{
    return eventStore.readEvents(customerId).asStream();
}

```

localhost:8081/customer/commands/eventStore/551085b5-9c95-43c7-89ec-50a6cd718000

```

1  [
2    {
3      "type": "CustomerAggregate",
4      "aggregateIdentifier": "551085b5-9c95-43c7-89ec-50a6cd718000",
5      "sequenceNumber": 0,
6      "identifier": "54947587-85cd-4000-8ea6-f6797abfba7f",
7      "timestamp": "2022-12-26T09:30:09.285Z",
8      "payload": {
9        "id": "551085b5-9c95-43c7-89ec-50a6cd718000",
10       "nom": "Ahlam",
11       "adresse": "Rabat",
12       "email": "Ahlam@gmail.com",
13       "telephone": "75431158"
14     },
15     "payloadType": "com.example.commonapi.events.CustomerCreatedEvent",
16     "metaData": {}
17   },
18   {
19     "type": "CustomerAggregate",
20     "aggregateIdentifier": "551085b5-9c95-43c7-89ec-50a6cd718000",
21     "sequenceNumber": 1,
22     "identifier": "7d6ef6e9-31db-41cd-8d2a-52d14d25ac33",
23     "timestamp": "2022-12-26T09:33:59.308Z",
24     "payload": {
25       "id": "551085b5-9c95-43c7-89ec-50a6cd718000",
26       "nom": "Ahlam",
27       "adresse": "Rabat",
28       "email": "Ahlam@gmail.com",
29       "telephone": "000000000"
30     },
31     "payloadType": "com.example.commonapi.events.CustomerUpdatedEvent",
32     "metaData": {}
33   }
34 ]
35
36

```

3. Partie Lecture:

- Entity Customer

```
@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Customer {
    @Id
    private String id;
    private String nom ;
    private String adresse ;
    private String email ;
    private String telephone ;
}
```

- Consulter tous les customer

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

```
@QueryHandler
public List<Customer> on(GetAllCustomersQuery query) { return customerRepository.findAll(); }
```

Request URL

http://localhost:8081/customer/queries/getAllCustomers

Server response

Code










Details

200

Response body

```
[
  {
    "id": "551085b5-9c95-43c7-89ec-50a6cd718000",
    "nom": "Ahlam",
    "adresse": "Rabat",
    "email": "Ahlam@gmail.com",
    "telephone": "75431158"
  },
  {
    "id": "60ec5ab1-3abc-4c21-847b-412a85af609f",
    "nom": "Imane",
    "adresse": "Casablanca",
    "email": "Imane@gmail.com",
    "telephone": "0654240978"
  },
  {
    "id": "cec41139-f83b-4166-95bd-5c001f05b706",
    "nom": "khadija",
    "adresse": "Nador",
    "email": "khadija@gmail.com",
    "telephone": "123455678"
  }
]
```

+ Options

				id	adresse	email	nom	telephone
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	551085b5-9c95-43c7-89ec-50a6cd718000	Rabat	Ahlam@gmail.com	Ahlam	75431158
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	60ec5ab1-3abc-4c21-847b-412a85af609f	Casablanca	Imane@gmail.com	Imane	0654240978
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	cec41139-f83b-4166-95bd-5c001f05b706	Nador	khadija@gmail.com	khadija	123455678

- Consulter un radar à partir son id

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

Request URL

http://localhost:8081/customer/queries/getCustomer/cec41139-f83b-4166-95bd-5c001f05b706

Server response

Code

Details

200

Response body

```
{
  "id": "cec41139-f83b-4166-95bd-5c001f05b706",
  "nom": "khadija",
  "adresse": "Nador",
  "email": "khadija@gmail.com",
  "telephone": "123455678"
}
```

- Résumé :

customer-command-controller

PUT /customer/commands/updateCustomer

POST /customer/commands/createCustomer

GET /customer/commands/eventStore/{customerId}

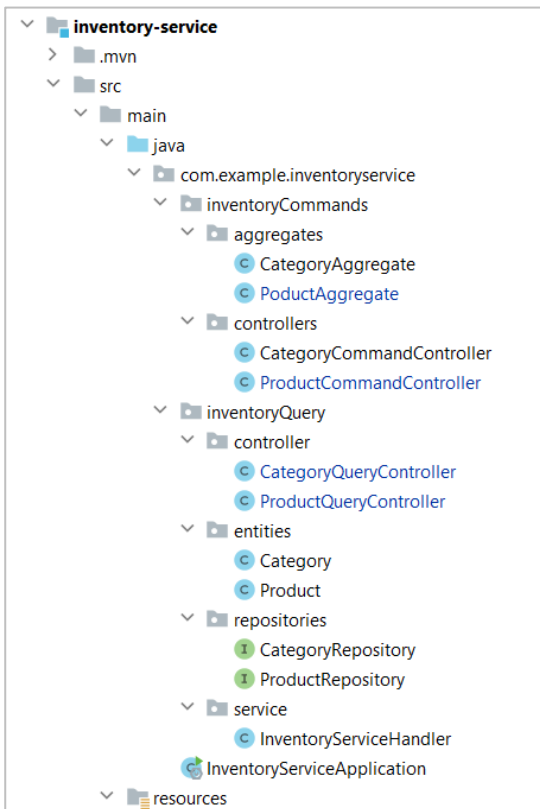
customer-query-controller

GET /customer/queries/getCustomer/{id}

GET /customer/queries/getAllCustomers

PARTIE 3 : Création de inventory-service

1. Structure de radar-service :



2. Partie Ecriture:

- Aggregate Product

```
@Aggregate
public class ProductAggregate {
    @AggregateIdentifier
    private String id;
    private String nom ;
    private double prix ;
    private int qteStock ;
    private ProductEtat etat ;

    public ProductAggregate() {
    }

    @CommandHandler
    public ProductAggregate(CreateProductCommand command) {
        AggregateLifecycle.apply(
            new ProductCreatedEvent(
                command.getId(),
                command.getNom(),
                command.getPrix(),
                command.getQteStock(),
                command.getEtat(),
                command.getCategory())
        );
    }
}
```

```

@EventSourcingHandler
public void on(ProductCreatedEvent event) {
    this.id = event.getId();
    this.nom = event.getNom();
    this.prix = event.getPrix();
    this.qteStock = event.getQteStock();
    this.etat = event.getEtat();
}

@CommandHandler
public void handle(UpdateProductCommand command) {
    AggregateLifecycle.apply(
        new ProductUpdatedEvent(
            command.getId(),
            command.getNom(),
            command.getPrix(),
            command.getQteStock(),
            command.getEtat(),
            command.getCategory()
        )
    );
}

@EventSourcingHandler
public void on(ProductUpdatedEvent event) {
    this.id = event.getId();
    this.nom = event.getNom();
    this.prix = event.getPrix();
    this.qteStock = event.getQteStock();
    this.etat = event.getEtat();
}
}

```

- Aggregate Category

```

@Aggregate
public class CategoryAggregate {
    @AggregateIdentifier
    private String id;
    private String nom ;
    private String description ;

    public CategoryAggregate() {
    }

    @CommandHandler
    public CategoryAggregate(CreateCategoryCommand command) {
        AggregateLifecycle.apply(
            new CategoryCreatedEvent(
                command.getId(),
                command.getNom(),
                command.getDescription()
            )
        );
    }
}

```

```

    @EventSourcingHandler
    public void on(CategoryCreatedEvent event) {
        this.id = event.getId();
        this.nom = event.getNom();
        this.description = event.getDescription();
    }

    @CommandHandler
    public void handle(UpdateCategoryCommand command) {
        AggregateLifecycle.apply(
            new CategoryUpdatedEvent(
                command.getId(),
                command.getNom(),
                command.getDescription()
            )
        );
    }

    @EventSourcingHandler
    public void on(CategoryUpdatedEvent event) {
        this.id = event.getId();
        this.nom = event.getNom();
        this.description = event.getDescription();
    }
}

```

- Les services de category et product

product-command-controller

PUT /proprietaire/commands/update

POST /proprietaire/commands/create

GET /proprietaire/commands/eventStore/{id}

category-command-controller

PUT /category/commands/update

POST /category/commands/create

GET /category/commands/eventStore/{id}

3. Partie Lecture:

- Entity Product

```

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor

public class Product {
    @Id
    private String id ;
    private String nom ;
    private double prix ;
    private int qte ;
    private ProductEtat etat ;

    @ManyToOne
    private Category category;
}

```

- Entity Category

```

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Category {
    @Id
    private String id ;
    private String description ;
    private String nom ;

    @OneToMany(mappedBy = "category")
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private List<Product> products = new ArrayList<>();
}

```

- Les services de category et product

product-query-controller

GET /products/queries/getProprietaire/{id}

GET /products/queries/allProducts

category-query-controller

GET /category/queries/getCategory/{id}

GET /category/queries/allCategory