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

Cycle Ingénieur

« Ingénierie Informatique – Big Data et Cloud Computing »

COMPTE-RENDU:

Examen de fin de module Systèmes Distribués

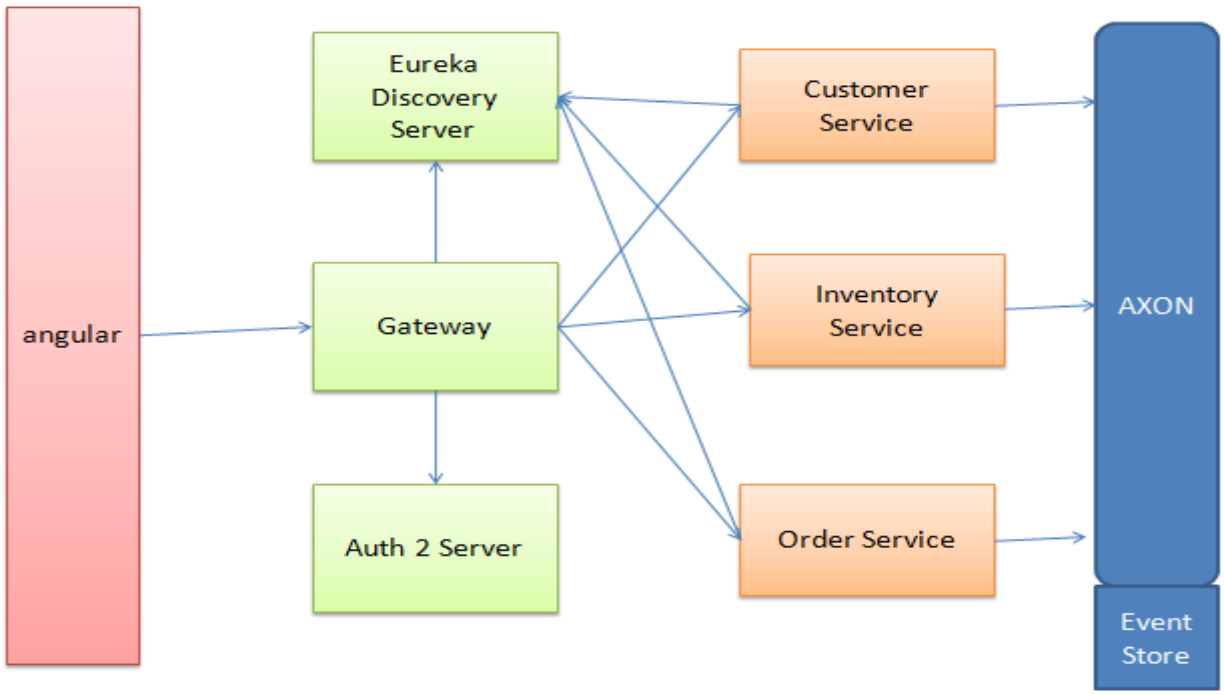
Réalisé par: Asmaa ELASRI **Encadré par:** Pr. Mohamed YOUSSFI

Année Universitaire : 2022-2023

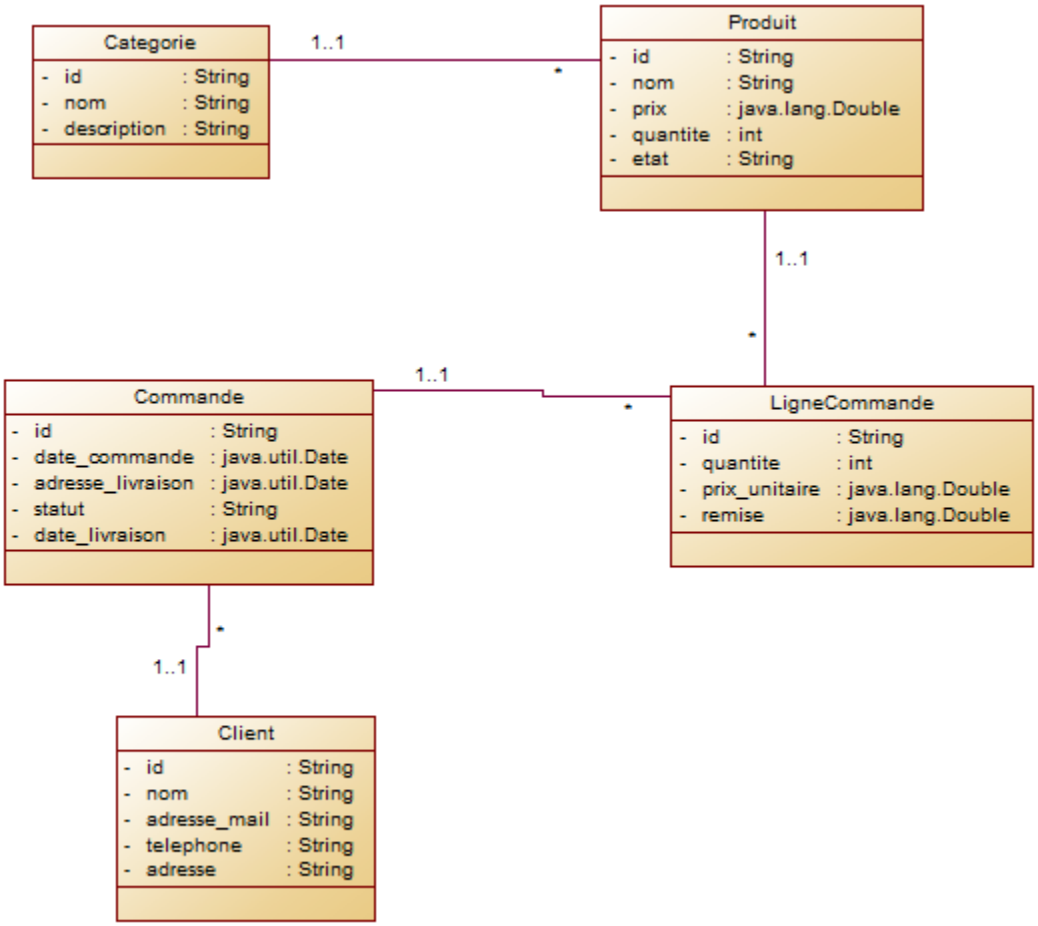
Sommaire

Travail à faire

1. **Établir une architecture technique du projet**



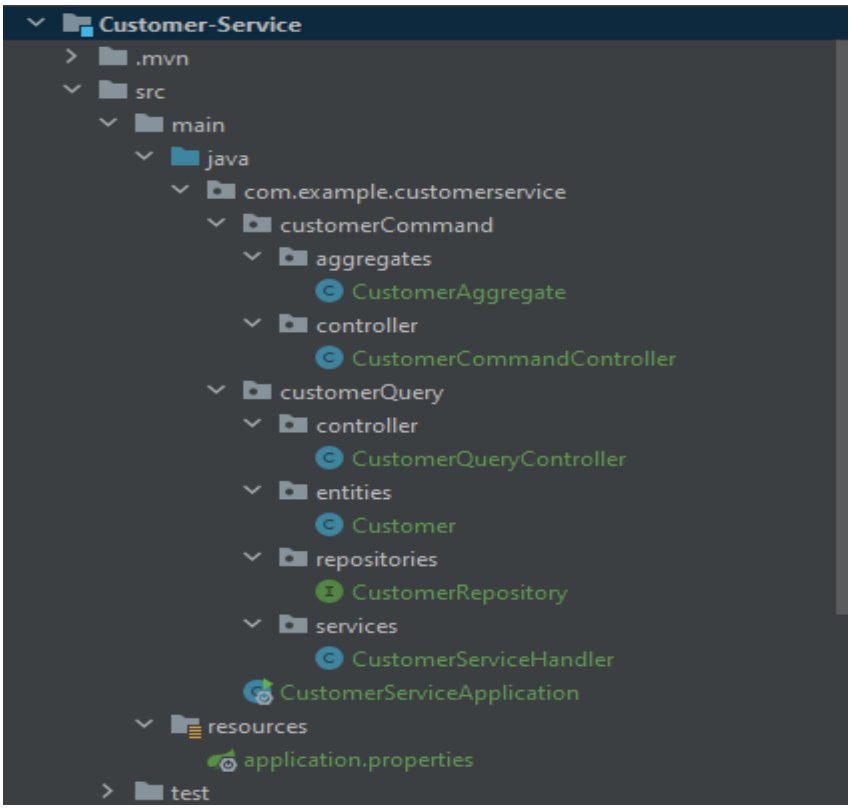
2. Établir un diagramme de classe global du projet



3. Déployer le serveur AXON Server ou KAFKA Broker

4. Développer le micro-service Customer-Service

La structure :



CustommerCommand:

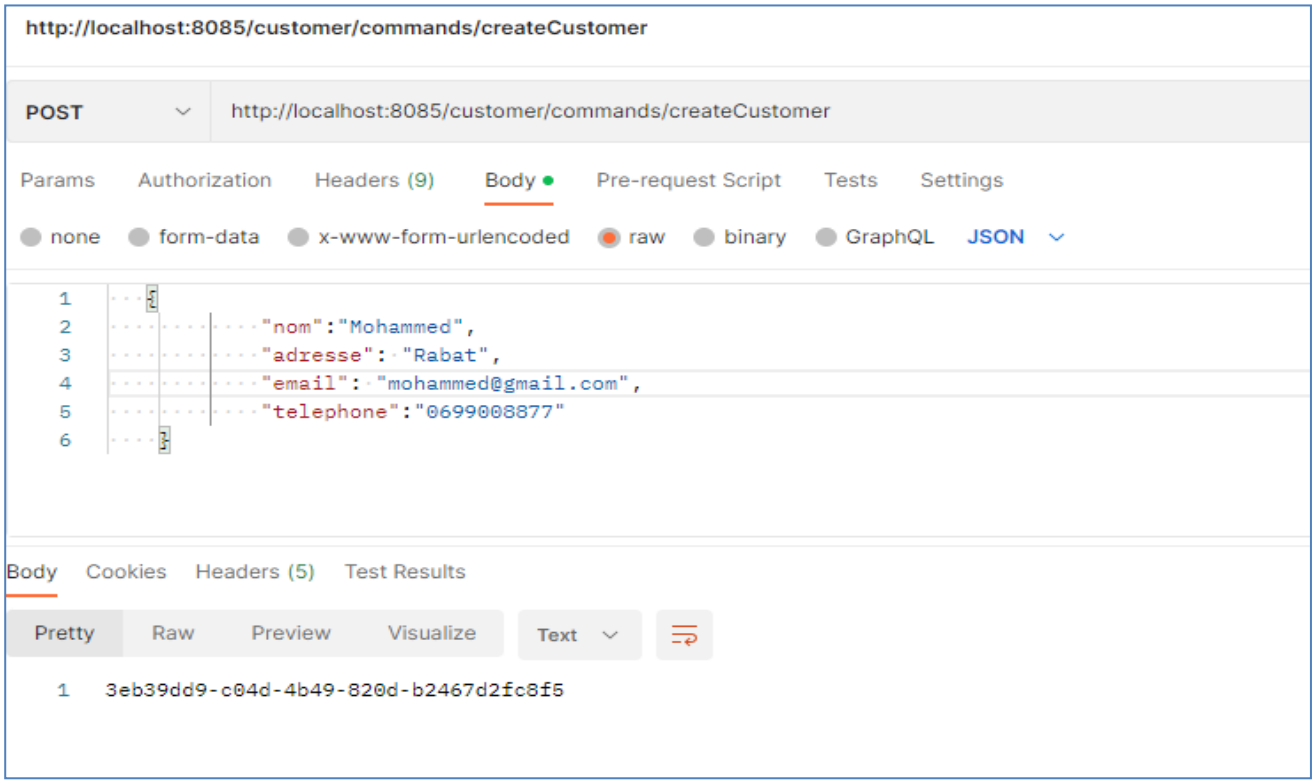
CustomerAggregate

```
@Aggregate @NoArgsConstructor
public class CustomerAggregate {
    1 usage
    @AggregateIdentifier
    private  String id ;
    2 usages
    private String nom ;
    2 usages
    private String adresse ;
    2 usages
    private  String email  ;
    2 usages
    private String telephone ;

    no usages  new *
```

CreateCustomer

```java
@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()
    ));
}

no usages  new *
@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();
}
```

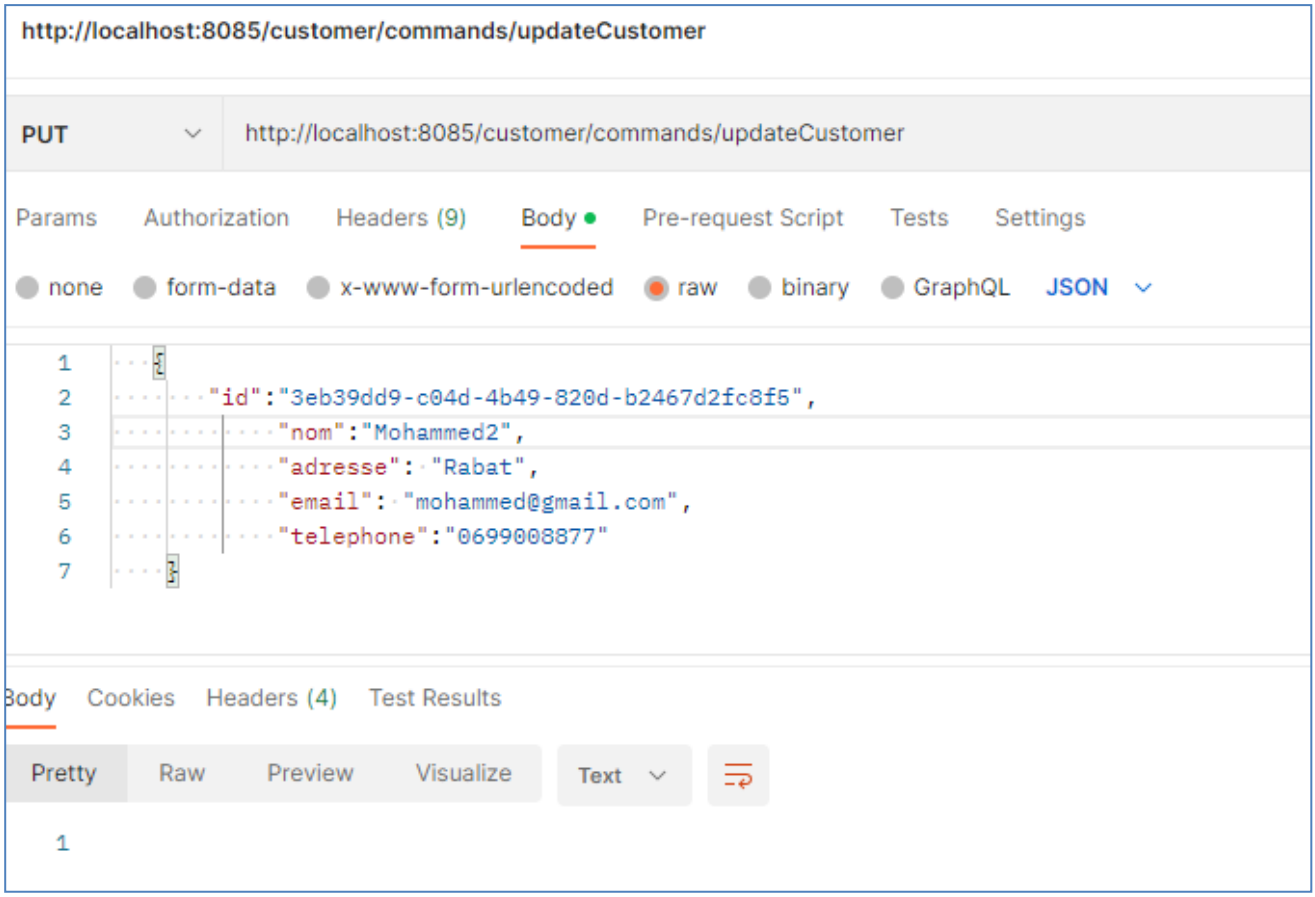
```java
@PostMapping("/createCustomer")
public CompletableFuture<String> createCustomer(@RequestBody CreateCustomerRequestDTO request)
{
    return  commandGateway.send(
            new CreateCustomerCommand(
                    UUID.randomUUID().toString(),
                    request.getNom(),
   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



UpdateCustomer

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

no usages  new *
@EventSourcingHandler
public void on(CustomerUpdatedEvent event) {
    this.nom = event.getNom();
    this.adresse = event.getAdresse();
    this.email = event.getEmail();
    this.telephone = event.getTelephone();
}
```

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



La partie de lecture :

```java
@CrossOrigin("*")
public class CustomerQueryController {
    2 usages
    private QueryGateway queryGateway;
    2 usages
    private CustomerRepository customerRepository;

    no usages  new *
    @GetMapping("/getAllCustomers")
    public List<Customer> getAllCustomers(){
        return queryGateway.query(new GetAllCustomersQuery(),
                ResponseTypes.multipleInstancesOf(Customer.class)).join();
    }
    no usages  new *
    @QueryHandler
    public List<Customer> on(GetAllCustomersQuery query) { return customerRepository.findAll(); }
```

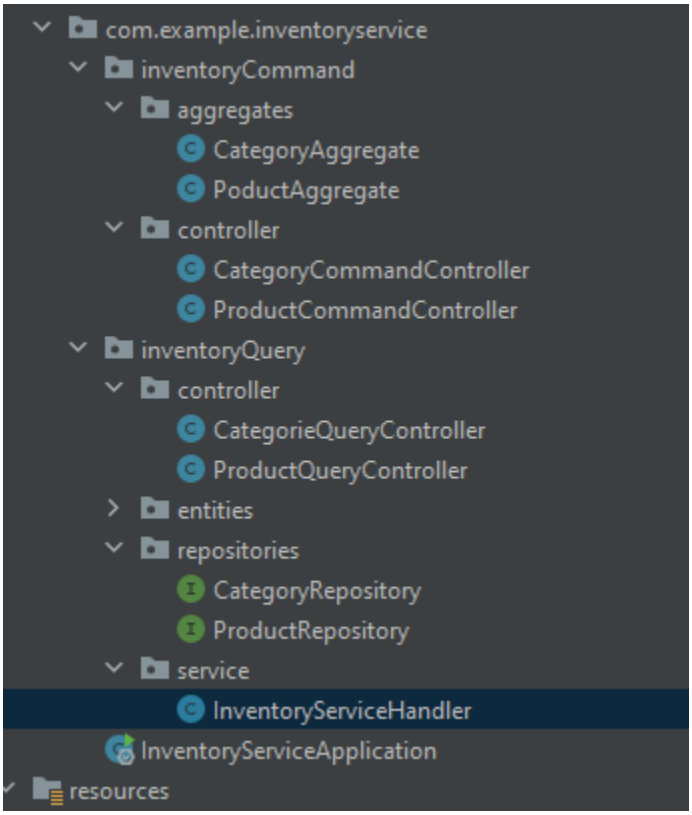
```java
    no usages  new *
    @GetMapping("/getCustomer/{id}")
    public Customer getCustomer(@PathVariable String id){
        return queryGateway.query(new GetCustomerById(id),
                ResponseTypes.instanceOf(Customer.class)).join();
    }

    no usages  new *
    @QueryHandler
    public Customer on(GetCustomerById query){
        return customerRepository.findById(query
                .getId()).get();
    }
}
```

```java
    private CustomerRepository customerRepository;

    no usages  new *
    @EventHandler
    public void on(CustomerCreatedEvent event){
        Customer customer = new Customer();
        customer.setId(event.getId());
        customer.setNom(event.getNom());
        customer.setAdresse(event.getAdresse());
        customer.setEmail(event.getEmail());
        customer.setTelephone(event.getTelephone());
        customerRepository.save(customer);
    }

    no usages  new *
    @QueryHandler
    public List<Customer> on(GetAllCustomersQuery query) { return customerRepository.findAll(); }
}
```

localhost:8085/customer/queries/getAllCustomers


```
[
  {
    "id": "3eb39dd9-c04d-4b49-820d-b2467d2fc8f5",
    "nom": "Mohammed",
    "adresse": "Rabat",
    "email": "mohammed@gmail.com",
    "telephone": "0699008877"
  },
  {
    "id": "6f0b94bd-8cfa-4c5e-8c4a-5ffdc1911697",
    "nom": "ahmed",
    "adresse": "Casablanca",
    "email": "ahmed@gmail.com",
    "telephone": "0699008877"
  }
]
```

5. Développer le micro-service Inventory-Service



Category aggregate

```java
public class CategoryAggregate {
    2 usages
    @AggregateIdentifier
    private String id;
    2 usages
    private String nom ;
    2 usages
    private String description ;

    no usages
    public CategoryAggregate() {
    }
```

```java
@CommandHandler
public CategoryAggregate(CreateCategorieCommand command) {
    AggregateLifecycle.apply(
            new CategorieCreatedEvent(
                    command.getId(),
                    command.getNom(),
                    command.getDescription()

            )
    );
}

no usages
@EventSourcingHandler
public void on(CategorieCreatedEvent event) {
    this.id = event.getId();
    this.nom = event.getNom();
    this.description = event.getDescription();
}
```

Pour le controller

```java
@PostMapping("/create")
public CompletableFuture<String> createCategory(@RequestBody CreateCategorieRequestDTO request)
    return commandGateway.send(new CreateCategorieCommand(
            UUID.randomUUID().toString(),
            request.getNom(),
            request.getDescription()

    ));
}
```

Update category

```java
    @CommandHandler
    public void handle(UpdateCategorieCommand command) {
        AggregateLifecycle.apply(
                new CategorieUpdatedEvent(
                        command.getId(),
                        command.getNom(),
                        command.getDescription()
                )
        );
    }

    no usages
    @EventSourcingHandler
    public void on(CategorieUpdatedEvent event) {
        this.id = event.getId();
        this.nom = event.getNom();
        this.description = event.getDescription();
    }
}
```

```java
    @PutMapping("/update")
    public CompletableFuture<String> updateCategory(@RequestBody UpdateCategorieRequestDTO request
        return commandGateway.send(new UpdateCategorieCommand(
                request.getId(),
                request.getNom(),
                request.getDescription()
        ));
    }
```

Query

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)

8. Développer un micro-service qui permet faire du Real time Data Analytics en utilisant Kafka Streams (Nombre et total des commandes sur une fenêtre temporelle de 5 secondes)

9. Développer votre application Frontend avec Angular ou React

10. Sécuriser votre système avec un système de d'authentification OAuth2, OIDC avec Keycloak ou un service d'authentification basé sur Spring Security et JWT

11. Écrire un script docker-compose.yml pour le déploiement de ce système distribué dans des conteneurs docker.

PARTIE 1 :

1. **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)**
8. **Développer un micro-service qui permet faire du Real time Data Analytics en utilisant Kafka Streams (Nombre et total des commandes sur une fenêtre temporelle de 5 secondes)**
9. **Développer votre application Frontend avec Angular ou React**
10. **Sécuriser votre système avec un système de d'authentification OAuth2, OIDC avec Keycloak ou un service d'authentification basé sur Spring Security et JWT**
11. **Écrire un script docker-compose.yml pour le déploiement de ce système distribué dans des conteneurs docker.**