

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 YOUSSEFI

Année Universitaire : 2022-2023

Sommaire

| | |
|--|-------------------------------------|
| Département Mathématiques et Informatique | 1 |
| « Ingénierie Informatique – Big Data et Cloud Computing » | 1 |
| Activité pratique N° 3 - Architectures Micro services avec (Spring Cloud Config, Consul Discovery, Consul Config,Vault) | Error! Bookmark not defined. |
| Travail à faire | 4 |
| PARTIE 1 : Configuration des microservices | 4 |
| 1. Démarrer le service Consul Discovery. | Error! Bookmark not defined. |
| 2. Activer le service de configuration des microservices: | Error! Bookmark not defined. |
| 3. Création d'un dossier contient les fichiers de configuration des ms : | Error! Bookmark not defined. |
| 4. Configuration partagée par les microservices : | Error! Bookmark not defined. |
| 5. Exemple de configuration (Customer-service) : | Error! Bookmark not defined. |
| 6. Configuration de service de configuration crée : | Error! Bookmark not defined. |
| 7. Démarrer le service de configuration : | Error! Bookmark not defined. |
| PARTIE 2 : Création de Gateway | Error! Bookmark not defined. |
| 1. Configuration automatique de Gateway : | Error! Bookmark not defined. |
| 2. Configuration automatique de Gateway : | Error! Bookmark not defined. |
| PARTIE 3 : Création de customer-service | Error! Bookmark not defined. |
| 1. Structure du projet: | Error! Bookmark not defined. |
| 2. Configuration de service : | Error! Bookmark not defined. |
| 3. Entité de Customer: | Error! Bookmark not defined. |
| 4. Repository de customer-service: | Error! Bookmark not defined. |
| 5. Récupérer la configuration de customer-service : | Error! Bookmark not defined. |
| 6. Tester le customer-service : | Error! Bookmark not defined. |
| PARTIE 4 : Création de inventory-service | Error! Bookmark not defined. |
| 1. Configuration de service : | Error! Bookmark not defined. |
| 2. Entité de Product : | Error! Bookmark not defined. |
| 3. Projection de inventory -service: | Error! Bookmark not defined. |
| 4. Tester le inventory -service : | Error! Bookmark not defined. |
| PARTIE 5 : Création de order-service | Error! Bookmark not defined. |
| 1. Structure du projet : | Error! Bookmark not defined. |
| 2. Configuration de service : | Error! Bookmark not defined. |
| 3. Entité de Order: | Error! Bookmark not defined. |
| 4. Entité ProductItem: | Error! Bookmark not defined. |

5. Les modules utilisés dans order-service :..... Error! Bookmark not defined.
6. Communiquer le order-service avec customer-service avec OpenFeign : Error! Bookmark not defined.
7. Récupérer le customer et la liste des produits d'ordre : Error! Bookmark not defined.
8. Tester le order-service : Error! Bookmark not defined.

PARTIE 6 : Création de billing-service Error! Bookmark not defined.

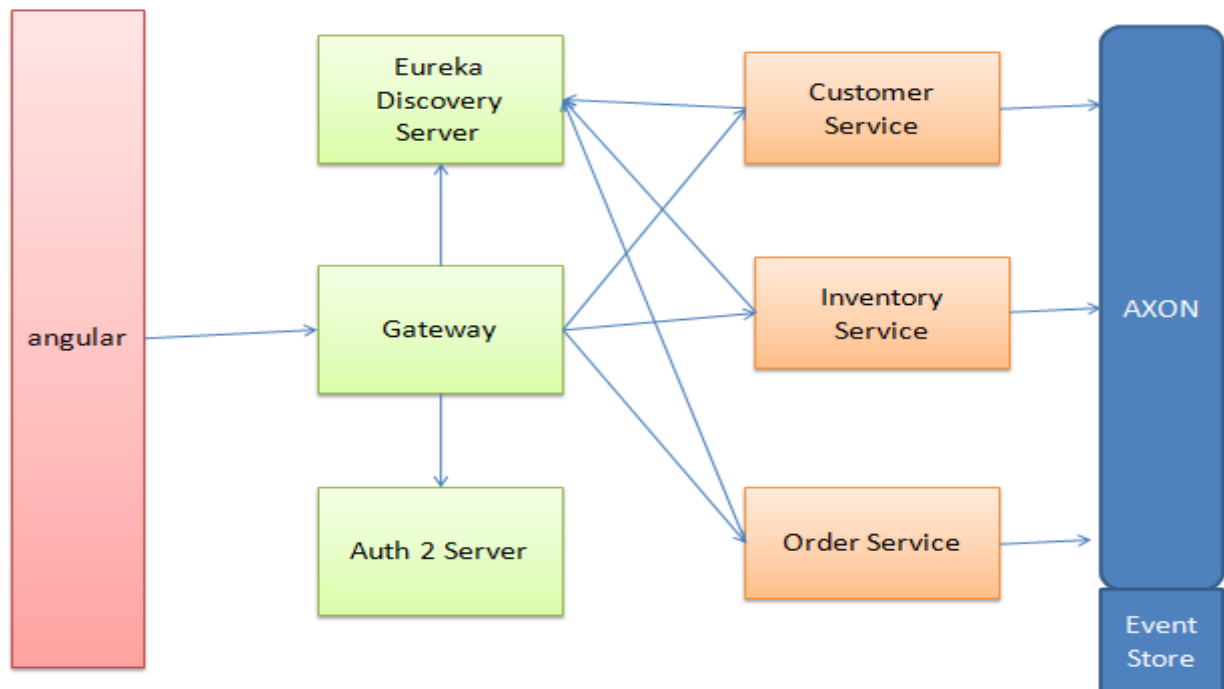
1. Dépendances Error! Bookmark not defined.
2. Structure..... Error! Bookmark not defined.
3. Configuration de billing-service : Error! Bookmark not defined.
4. Partager le secret avec Vault : Error! Bookmark not defined.

PARTIE 7 : Frontend avec Angular..... Error! Bookmark not defined.

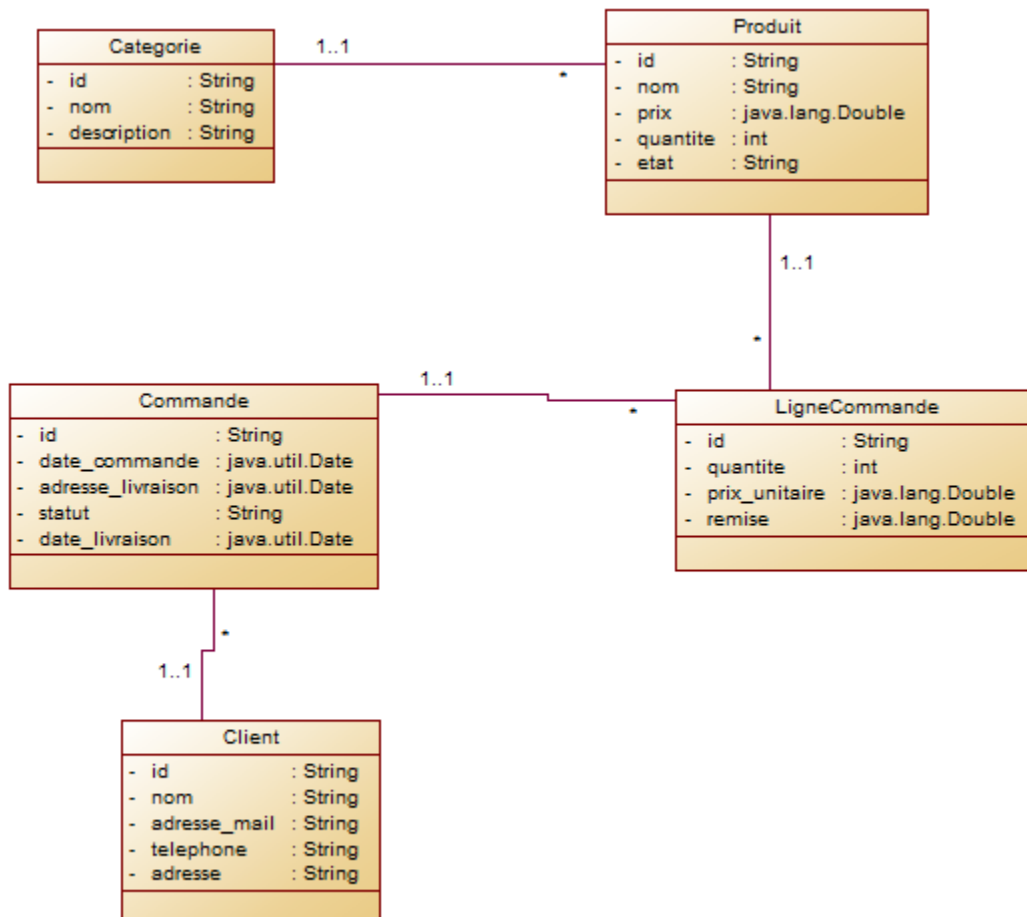
1. Liste des produits : Error! Bookmark not defined.
2. Liste des customer : Error! Bookmark not defined.
3. Liste des order de customer 1 : Error! Bookmark not defined.
4. Détails de order 5 : Error! Bookmark not defined.

Travail à faire

1. Établir une architecture technique du projet



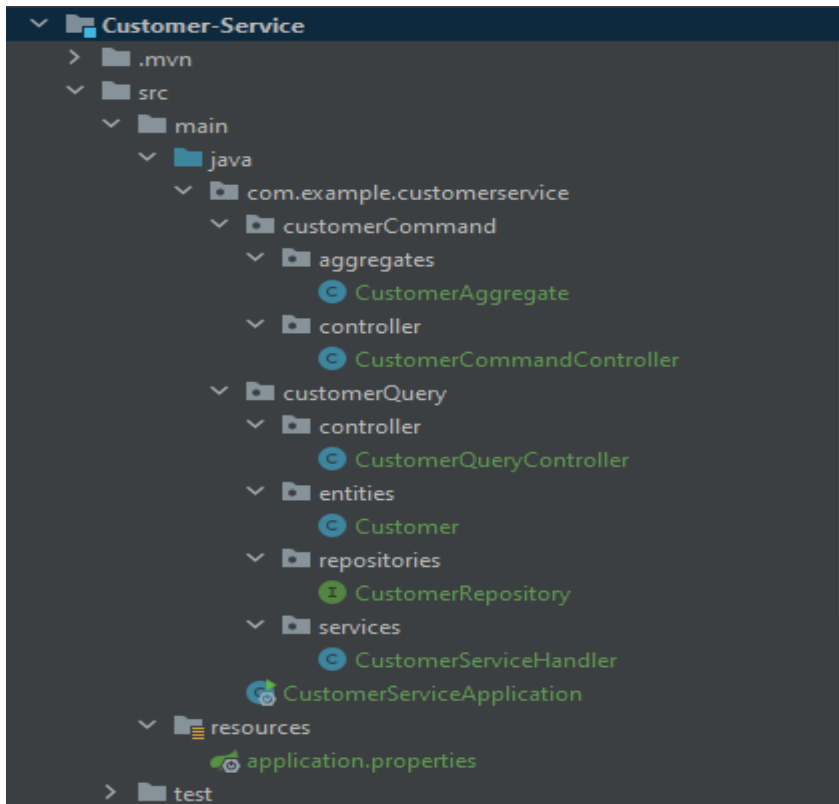
2. Établir un diagramme de classe global du projet



3. Déployer le serveur AXON Server ou KAFKA Broker

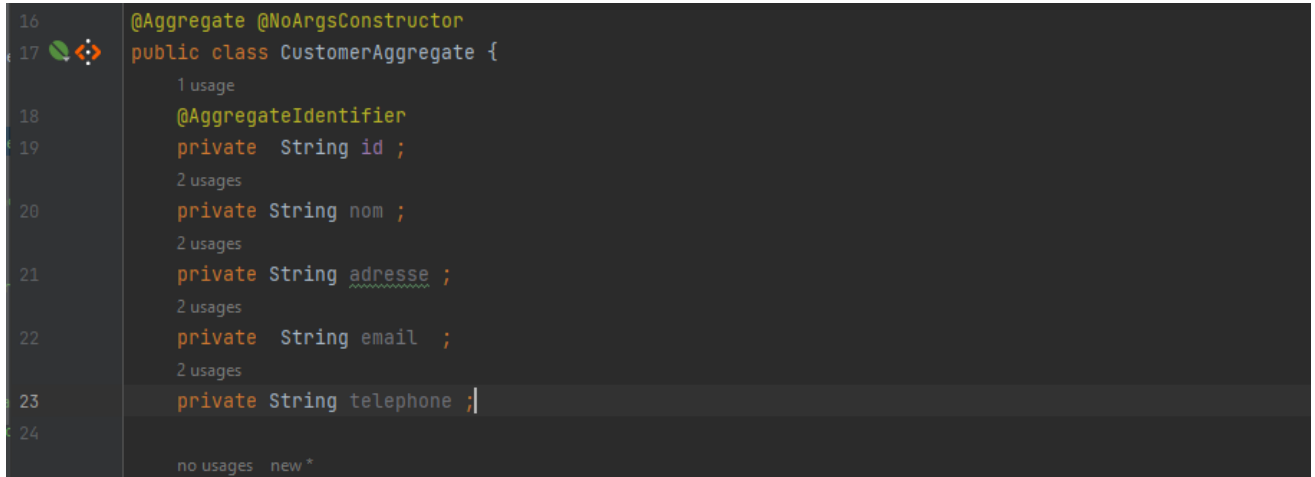
4. Développer le micro-service Customer-Service

La structure :



CustomerCommand:

CustomerAggregate



CreateCustomer

```

25  ➔ @CommandHandler
26  @ public CustomerAggregate(CreateCustomerCommand command){
27      if(command.getNom().isEmpty()){
28          throw new RuntimeException("Le nom ne peut pas être vide");
29      }
30      AggregateLifecycle.apply(new CustomerCreatedEvent(
31          command.getId(),
32          command.getNom(),
33          command.getAdresse(),
34          command.getEmail(),
35          command.getTelephone()
36      ));
37  }
38
39  ➔ no usages new *
40  @EventSourcingHandler
41  public void on(CustomerCreatedEvent event) {
42      this.id = event.getId();
43      this.nom = event.getNom();
44      this.adresse = event.getAdresse();
45      this.email = event.getEmail();
46      this.telephone = event.getTelephone();
47  }
48
49  @PostMapping("/createCustomer")
50  public CompletableFuture<String> createCustomer(@RequestBody CreateCustomerRequestDTO request)
51  {
52      return commandGateway.send(
53          new CreateCustomerCommand(
54              UUID.randomUUID().toString(),
55              request.getNom(),
56              request.getAdresse(),
57              request.getEmail(),
58              request.getTelephone()
59          ));
60  }

```

http://localhost:8085/customer/commands/createCustomer

POST http://localhost:8085/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": "Mohammed",
3    "adresse": "Rabat",
4    "email": "mohammed@gmail.com",
5    "telephone": "0699008877"
6  }

```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text

```

1  3eb39dd9-c04d-4b49-820d-b2467d2fc8f5

```

UpdateCustomer

```

48  @CommandHandler
49  @
50  public void handle(UpdateCustomerCommand command) {
51      AggregateLifecycle.apply(new CustomerUpdatedEvent(
52          command.getId(),
53          command.getNom(),
54          command.getAdresse(),
55          command.getEmail(),
56          command.getTelephone()
57      ));
58  }
59  no usages new *
60  @EventSourcingHandler
61  public void on(CustomerUpdatedEvent event) {
62      this.nom = event.getNom();
63      this.adresse = event.getAdresse();
64      this.email = event.getEmail();
65      this.telephone = event.getTelephone();
66  }

```

```

38     @PostMapping("/updateCustomer")
39     public CompletableFuture<String> updateCustomer(@RequestBody UpdateCustomerRequestDTO request){
40         return commandGateway.send(
41             new UpdateCustomerCommand(
42                 request.getId(),
43                 request.getNom(),
44                 request.getAdresse(),
45                 request.getEmail(),
46                 request.getTelephone()
47             )
48         );
49     }

```

http://localhost:8085/customer/commands/updateCustomer

PUT http://localhost:8085/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": "3eb39dd9-c04d-4b49-820d-b2467d2fc8f5",
3    "nom": "Mohammed2",
4    "adresse": "Rabat",
5    "email": "mohammed@gmail.com",
6    "telephone": "0699008877"
7  }

```

Body Cookies Headers (4) Test Results

Pretty Raw Preview Visualize Text

1

La partie de lecture :

```

18     @CrossOrigin("*")
19     public class CustomerQueryController {
20         private QueryGateway queryGateway;
21         private CustomerRepository customerRepository;
22
23         no usages new *
24         @GetMapping("/getAllCustomers")
25         public List<Customer> getAllCustomers(){
26             return queryGateway.query(new GetAllCustomersQuery(),
27                                     ResponseTypes.multipleInstancesOf(Customer.class)).join();
28         }
29
30         no usages new *
31         @QueryHandler
32         public List<Customer> on(GetAllCustomersQuery query) { return customerRepository.findAll(); }

```



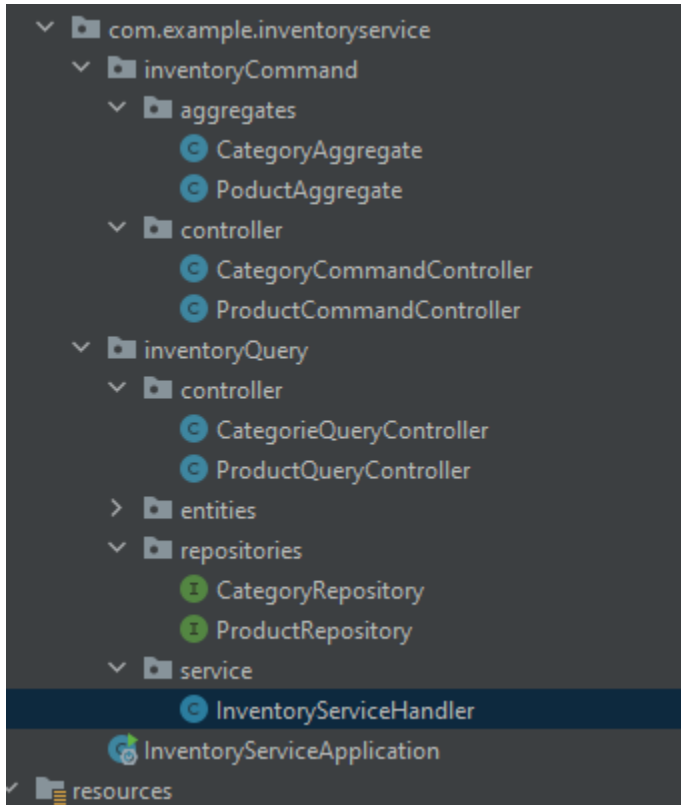
```
no usages new *
33 @GetMapping("/getCustomer/{id}")
34 public Customer getCustomer(@PathVariable String id){
35     return queryGateway.query(new GetCustomerById(id),
36         ResponseTypes.instanceOf(Customer.class)).join();
37 }
38
no usages new *
39 @QueryHandler
40 public Customer on(GetCustomerById query){
41     return customerRepository.findById(query
42         .getId()).get();
43 }
44 }

19 private CustomerRepository customerRepository;
20
no usages new *
21 @EventHandler
22 @ public void on(CustomerCreatedEvent event){
23     Customer customer = new Customer();
24     customer.setId(event.getId());
25     customer.setNom(event.getNom());
26     customer.setAdresse(event.getAdresse());
27     customer.setEmail(event.getEmail());
28     customer.setTelephone(event.getTelephone());
29     customerRepository.save(customer);
30 }
31
no usages new *
32 @QueryHandler
33 public List<Customer> on(GetAllCustomersQuery query) { return customerRepository.findAll(); }
36 }
```

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

```
16 public class CategoryAggregate {  
17     2 usages  
18     @AggregateIdentifier  
19     private String id;  
20     2 usages  
21     private String nom ;  
22     2 usages  
23     private String description ;  
24  
25     no usages  
26     public CategoryAggregate() {  
27     }  
28 }
```

```

25  ➔ @CommandHandler
26  @ public CategoryAggregate(CreateCategorieCommand command) {
27      AggregateLifecycle.apply(
28  <-      new CategorieCreatedEvent(
29          command.getId(),
30          command.getNom(),
31          command.getDescription()
32      )
33  )
34  };
35  }
36
37  no usages
38  ➔ @EventSourcingHandler
39  @ public void on(CategorieCreatedEvent event) {
40      this.id = event.getId();
41      this.nom = event.getNom();
42      this.description = event.getDescription();
43  }

```

Pour le controller

```

24  @PostMapping("/create")
25  @ public CompletableFuture<String> createCategory(@RequestBody CreateCategorieRequestDTO request)
26  <-      return commandGateway.send(new CreateCategorieCommand(
27          UUID.randomUUID().toString(),
28          request.getNom(),
29          request.getDescription()
30      ));
31  }
32

```

Update category

```

44  ➔ @CommandHandler
45  @ public void handle(UpdateCategorieCommand command) {
46      AggregateLifecycle.apply(
47  ⬅      new CategorieUpdatedEvent(
48          command.getId(),
49          command.getNom(),
50          command.getDescription()
51      )
52  );
53  }
54
no usages
55  ➔ @EventSourcingHandler
56  @ public void on(CategorieUpdatedEvent event) {
57      this.id = event.getId();
58      this.nom = event.getNom();
59      this.description = event.getDescription();
60  }
61  }
62
no usages
34  @PutMapping("/update")
35  @ public CompletableFuture<String> updateCategory(@RequestBody UpdateCategorieRequestDTO request
36  ⬅      return commandGateway.send(new UpdateCategorieCommand(
37          request.getId(),
38          request.getNom(),
39          request.getDescription()
40      ));
41  }
42

```

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.**