

Département Mathématiques et Informatique

Cycle Ingénieur

« Ingénierie Informatique – Big Data et Cloud Computing »

COMPTE-RENDU:

Examen de fin de module Systèmes Distribués

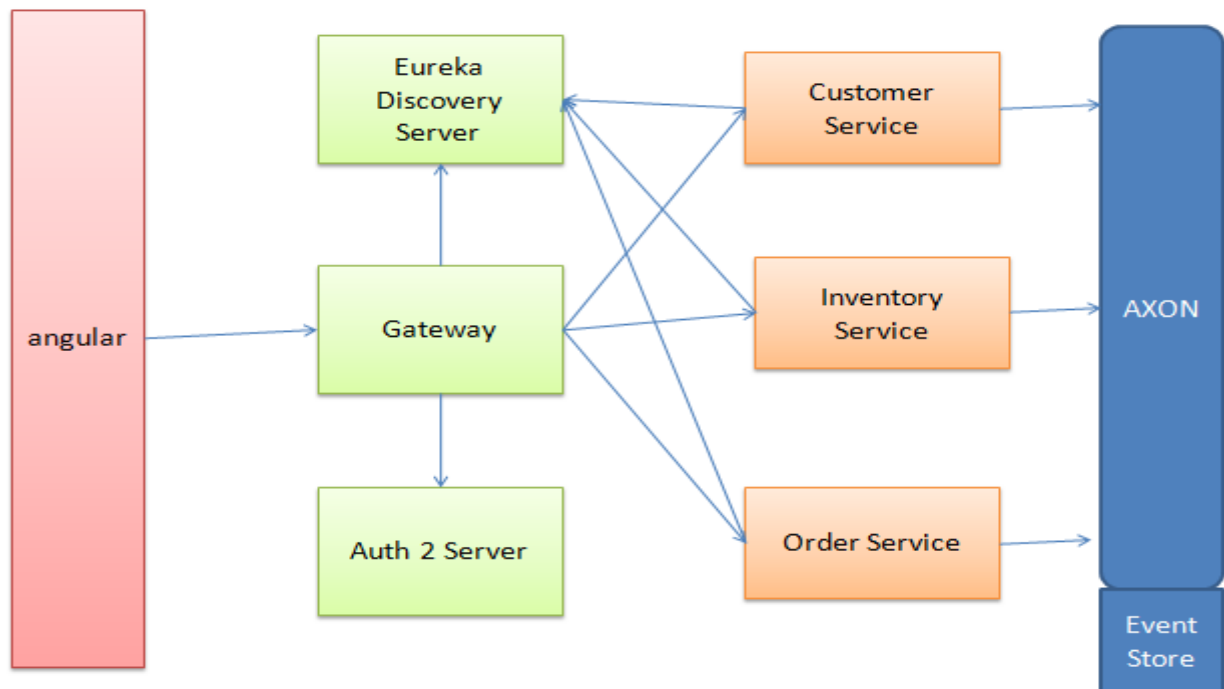
Réalisé par: Asmaa ELASRI

Encadré par: Pr. Mohamed YOUSSEFI

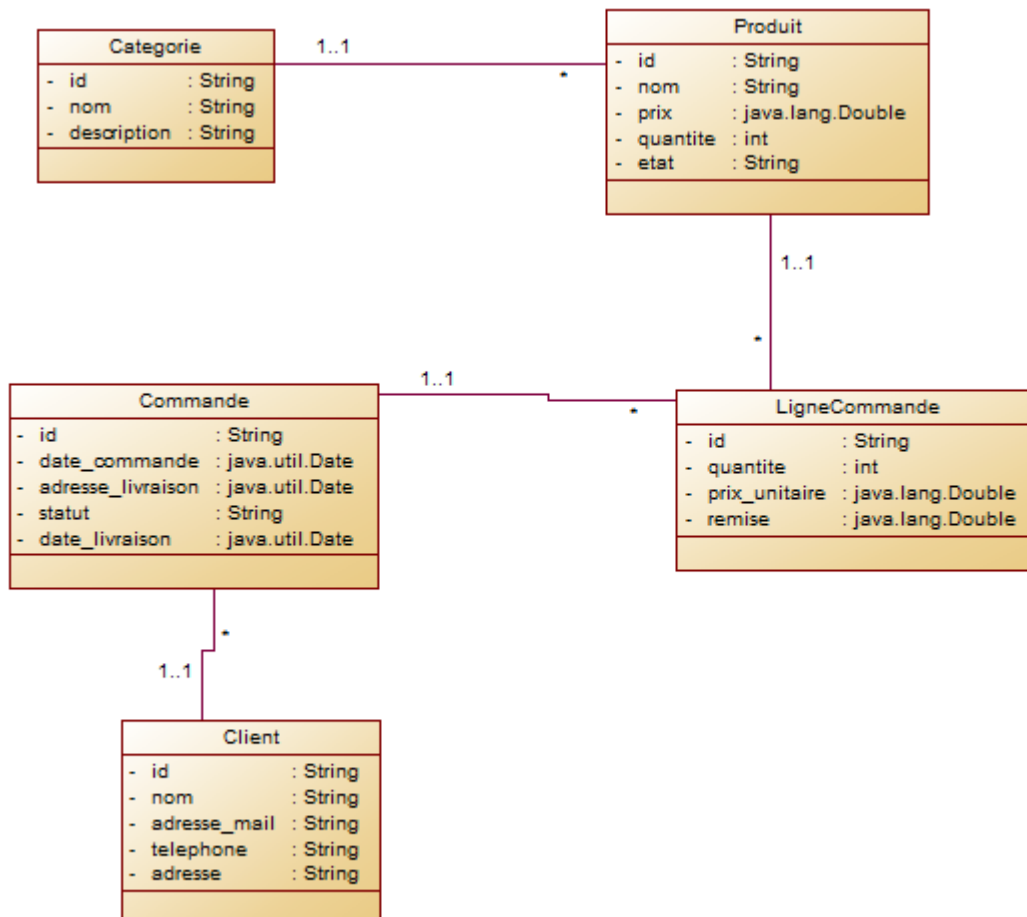
Année Universitaire : 2022-2023

Travail à faire

1. Établir une architecture technique du projet



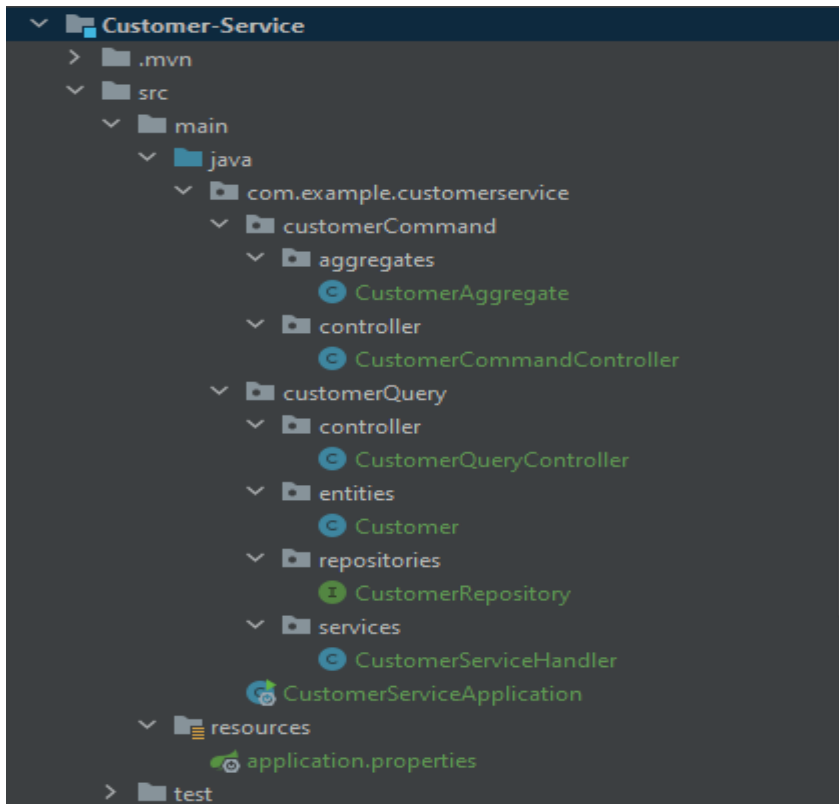
2. Établir un diagramme de classe global du projet



3. Déployer le serveur AXON Server ou KAFKA Broker

4. Développer le micro-service Customer-Service

La structure :



CustomerCommand:

CustomerAggregate

```
16 @Aggregate @NoArgsConstructor
17 public class CustomerAggregate {
18     1 usage
19     @AggregateIdentifier
20     private String id ;
21     2 usages
22     private String nom ;
23     2 usages
24     private String adresse ;
25     2 usages
26     private String email ;
27     2 usages
28     private String telephone ;
29 }
```

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(); }

```

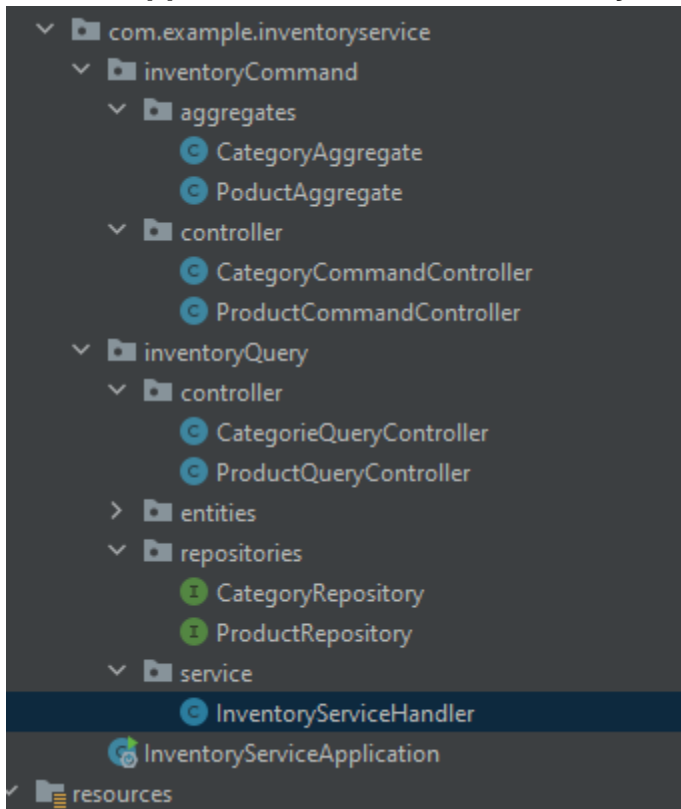
```
no usages new *
33 @GetMapping("/getCustomer/{id}")
34 public Customer getCustomer(@PathVariable String id){
35     return queryGateway.query(new GetCustomerById(id),
36         ResponseTypes.instanceOf(Customer.class)).join();
37 }
38
no usages new *
39 @QueryHandler
40 public Customer on(GetCustomerById 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  ➔ {
27      return commandGateway.send(new CreateCategorieCommand(
28          UUID.randomUUID().toString(),
29          request.getNom(),
30          request.getDescription()
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
55  ➔ no usages
56  @EventSourcingHandler
57  @ public void on(CategorieUpdatedEvent event) {
58      this.id = event.getId();
59      this.nom = event.getNom();
60      this.description = event.getDescription();
61  }
62  }

```

```

34  no usages
35  @PutMapping("/update")
36  @ public CompletableFuture<String> updateCategory(@RequestBody UpdateCategorieRequestDTO request) {
37  ⬅      return commandGateway.send(new UpdateCategorieCommand(
38          request.getId(),
39          request.getNom(),
40          request.getDescription()
41      ));
42  }

```

La même chose pour product

```
no usages
28 ➤ @CommandHandler
29  @ public ProductAggregate(CreateProductCommand command) {
30      AggregateLifecycle.apply(
31  ⬅      new ProductCreatedEvent(
32          command.getId(),
33          command.getNom(),
34          command.getPrix(),
35          command.getQte(),
36          command.getEtat()
37      )
38  );
39  }
40
no usages
41 ➤ @EventSourcingHandler
42  @ public void on(ProductCreatedEvent event) {
43      this.id = event.getId();
44      this.nom = event.getNom();
45      this.prix = event.getPrix();
46      this.etat = event.getEtat();
47      this.qteStock = event.getQte();
48  }
49  }
```

```
27  @PostMapping("/create")
28  @ public CompletableFuture<String> createProduct(@RequestBody CreateProductRequestDTO request)
29  ⬅  {
30      return commandGateway.send(new CreateProductCommand(
31          UUID.randomUUID().toString(),
32          request.getNom(),
33          request.getPrix(),
34          request.getQte(),
35          request.getEtat()
36      ));
37  }
38
no usages
38  @PutMapping("/update")
39  @ public CompletableFuture<String> updateProduct(@RequestBody UpdateProductRequestDTO request)
40  ⬅  {
41      return commandGateway.send(new UpdateProductCommand(
42          request.getId(),
43          request.getNom(),
44          request.getPrix(),
45          request.getQte(),
46          request.getEtat()
47      ));
48  }
```

Query

```

23  ➔ @EventHandler
24  @ public void on(CategorieCreatedEvent event) {
25      log.info("*****");
26      log.info("CategorieCreatedEvent received");
27      Categorie categorie = new Categorie() ;
28      categorie.setId(event.getId());
29      categorie.setDescription(event.getDescription());
30      categorie.setNom(event.getNom());
31      categoryRepository.save(categorie) ;
32  }
33
no usages
34  ➔ @EventHandler
35  @ public void on(CategorieUpdatedEvent event) {
36      log.info("*****");
37      log.info("CategorieUpdatedEvent received");
38      Categorie categorie= categoryRepository.findById(event.getId()).get();
39      categorie.setDescription(event.getDescription());
40      categorie.setNom(event.getNom());
41      categoryRepository.save(categorie) ;
42  }

```

```

44  ➔ @EventHandler
45  @ public void on(ProductCreatedEvent event) {
46      log.info("*****");
47      log.info("ProductCreatedEvent received");
48      Product product = new Product() ;
49      product.setId(event.getId());
50      product.setNom(event.getNom());
51      product.setPrix(event.getPrix());
52      product.setQte(event.getQte());
53      product.setEtat(event.getEtat());
54
55      Categorie categorie = categoryRepository.findById(event.getCategorie()).get();
56      if (categorie != null) {
57          product.setCategorie(categorie);
58      }
59      productRepository.save(product) ;
60  }
61
no usages
62  ➔ @EventHandler
63  @ public void on(ProductUpdatedEvent event) {
64      log.info("*****");
65      log.info("ProductUpdatedEvent received");
66      Product product = productRepository.findById(event.getId()).get();
67      product.setNom(event.getNom());
68      product.setPrix(event.getPrix());

```

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.**