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

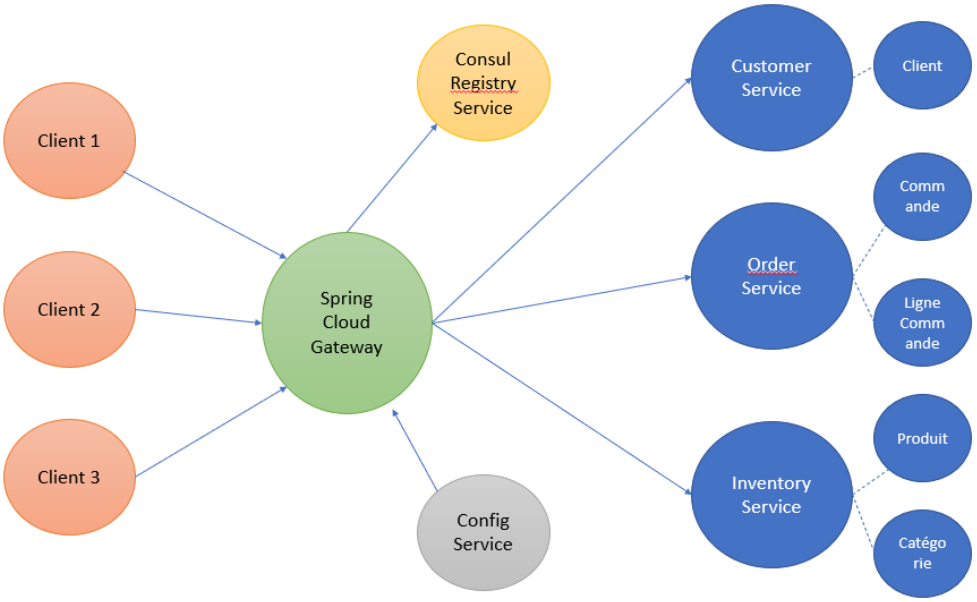
Examen Final

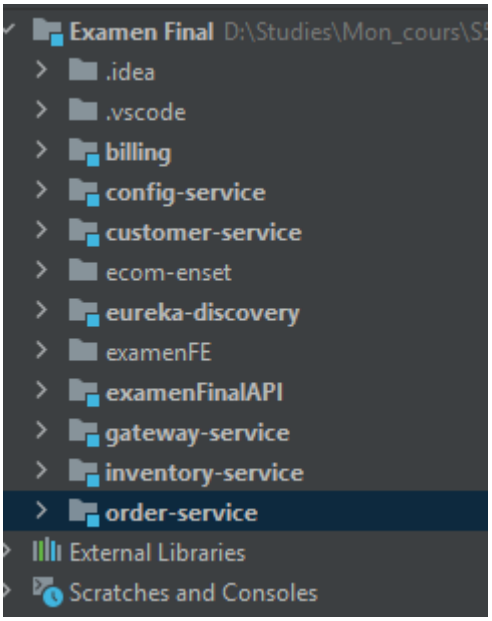
Réalisé par :

Soukaina EL KAMOUNI

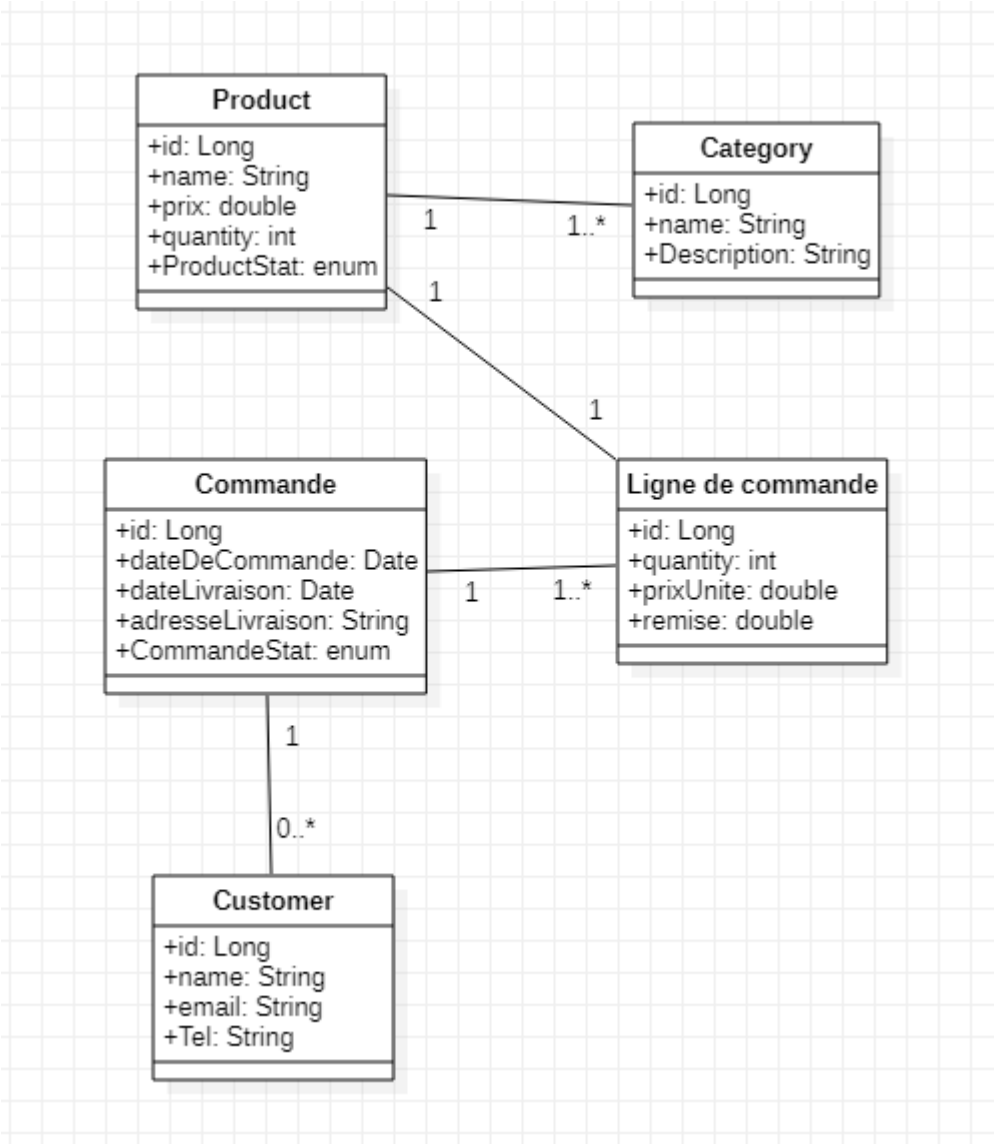
Année Universitaire : 2022-2023

1. Établir une architecture technique du projet



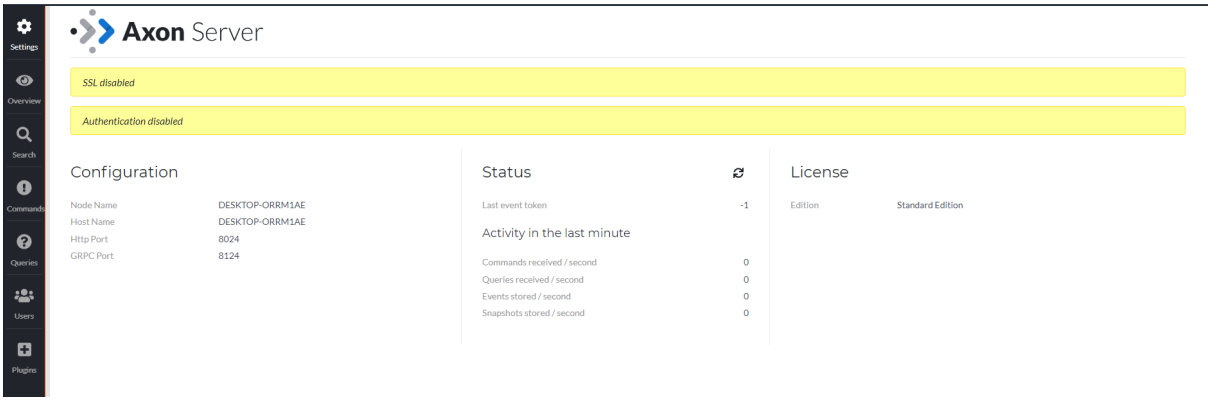


2. Établir un diagramme de classe global du projet



3. Déployer le serveur AXON Server ou KAFKA Broker

```
D:\Studies\Mon_cours\S5\MicroService plus\AxonServer-4.6.7>java -jar axonserver.jar
    _                   ____
   / \   __  _____  _ __ / ___|  ___ _ ____   _____ _ __
  / _ \  \ \/ / _ \| '_ \\___ \ / _ \ '__\ \ / / _ \ '__|
 / ___ \  >  < (_) | | | |___) |  __/ |   \ V /  __/ |
/_/   \_\/_/\_\___/|_| |_|____/ \___|_|    \_/ \___|_|
 Standard Edition                    Powered by AxonIQ

version: 4.6.7
2022-12-26 12:11:16.974  INFO 19340 --- [           main] io.axoniq.axonserver.AxonServer          : Starting AxonServer using Java 1.8.0_20
2 on DESKTOP-ORRM1AE with PID 19340 (D:\Studies\Mon_cours\S5\MicroService plus\AxonServer-4.6.7\axonserver.jar started by user in D:\Studies
\Mon_cours\S5\MicroService plus\AxonServer-4.6.7)
2022-12-26 12:11:16.980  INFO 19340 --- [           main] io.axoniq.axonserver.AxonServer          : No active profile set, falling back to
1 default profile: "default"
2022-12-26 12:11:20.915  INFO 19340 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8024 (
http)
2022-12-26 12:11:21.188  INFO 19340 --- [           main] A.i.a.a.c.MessagingPlatformConfiguration : Configuration initialized with SSL DISA
BLED and access control DISABLED.
2022-12-26 12:11:24.775  INFO 19340 --- [           main] io.axoniq.axonserver.AxonServer          : Axon Server version 4.6.7
2022-12-26 12:11:31.128  INFO 19340 --- [           main] io.axoniq.axonserver.grpc.Gateway        : Axon Server Gateway started on port: 81
24 - no SSL
2022-12-26 12:11:31.141  INFO 19340 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8024 (http)
with context path ''
2022-12-26 12:11:31.170  INFO 19340 --- [           main] io.axoniq.axonserver.AxonServer          : Started AxonServer in 14.752 seconds (J
VM running for 15.434)
```



1. Customer Service:

a. Query:

i. Entities:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Customer {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String name;
   private String email;
   private String tel;
}
```

ii. Repositories:

```
@RepositoryRestResource
public interface CustomerRepository extends
JpaRepository<Customer, Long> {
}
```

iii. Security:

```
@Configuration
```

```
public class KeycloakAdapterConfig {
   @Bean
   public KeycloakSpringBootConfigResolver
keycloakConfigResolver(){
       return new KeycloakSpringBootConfigResolver();
   }
}
```

and:

```
@KeycloakConfiguration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends
KeycloakWebSecurityConfigurerAdapter {
   @Override
   protected SessionAuthenticationStrategy
sessionAuthenticationStrategy() {
       return new
RegisterSessionAuthenticationStrategy(new
SessionRegistryImpl());
   }

   @Override
   protected void configure(AuthenticationManagerBuilder
auth) throws Exception {

auth.authenticationProvider(keycloakAuthenticationProvider
());
           }

   @Override
   protected void configure(HttpSecurity http) throws
Exception {
       super.configure(http);

       http.csrf().disable();

http.authorizeRequests().antMatchers("/h2-console/**").per
mitAll();
       http.headers().frameOptions().disable();
```

```
http.authorizeRequests().anyRequest().authenticated();
    }
}
```

iv. Web:

```
@RestController
@RefreshScope
public class CustomerConfigTestController {
    @Value("${global.params.p1}")
    private String p1;
    @Value("${global.params.p2}")
    private String p2;
    @Value("${customer.params.x}")
    private String x;
    @Value("${customer.params.y}")
    private String y;

    @GetMapping("/params")
    public Map<String,String> params(){
        return Map.of("p1",p1,"p2",p2,"x",x,"y",y);
    }
}
```

v. Application Properties:

```
server.port=8081
spring.application.name=customer-service
spring.config.import=optional:configserver:http://localhos
t:8888
spring.zipkin.base-url=http://127.0.0.1:9411/
keycloak.realm=spring-boot-microservices-realm
keycloak.resource=spring-boot-client
keycloak.bearer-only=true
keycloak.auth-server-url=http://localhost:8080
keycloak.ssl-required=none
keycloak.principal-attribute=name
keycloak.realm-key=
#the keycloack app didn't work for me, so will try it
later
```

b. Commands:

i. Aggregate:

```
public class CustomerAggregate {
    @AggregateIdentifier
    private Long id;
    private String name;
    private String email;
    private String tel;

    public CustomerAggregate() {
    }

    @CommandHandler
    public CustomerAggregate(CustomerCreatedCommand
command){
        if (command.getName() == null ||
command.getName().isEmpty()) {
            throw new IllegalArgumentException("Name cannot
be empty");
        }
        AggregateLifecycle.apply(new CustomerCreatedEvent(
                command.getId(),
                command.getName(),
                command.getTel(),
                command.getEmail()));
    }

    @EventSourcingHandler
    public void on(CustomerCreatedEvent event) {
        this.id = event.getId();
        this.name = event.getName();
        this.tel = event.getTel();
        this.email = event.getEmail();
    }
}
```

ii. Controllers:

```
@RestController
@RequestMapping("/command/customer")
```

```
@AllArgsConstructor
@Service
public class CustomerCommandController {
   private CommandGateway commandGateway;
   @PostMapping(path = "/create")
   public CompletableFuture<String>
createOwner(@RequestBody CreateCustumerRequestDTO
createCustumerRequestDTO) {
       CompletableFuture<String> response =
commandGateway.send(new CustomerCreatedEvent(
               UUID.randomUUID().getMostSignificantBits(),
               createCustumerRequestDTO.getName(),
               createCustumerRequestDTO.getTel(),
               createCustumerRequestDTO.getEmail()
       ));
       return response;
   }
}
```

2. Inventory Service:
    a. Commands:
        i. Aggregate:

- Catégorie:

```
public class CategorieAggregate {
   @AggregateIdentifier
   private Long id;
   private String name;
   private String Description;
   public CategorieAggregate(){}

   @CommandHandler
   public CategorieAggregate(CategorieCreatedCommand
command){
       if (command.getName() == null ||
command.getName().isEmpty()) {
           throw new IllegalArgumentException("Name cannot
be empty");
       }
```

```
        AggregateLifecycle.apply(new CategorieCreatedEvent(
                command.getId(),
                command.getName(),
                command.getDescription()));
    }

    @EventSourcingHandler
    public void on(CategorieCreatedEvent event) {
        this.id = event.getId();
        this.name = event.getName();
        this.Description = event.getDescription();
    }
}
```

- Product:

```
public class ProductAggregate {
    @AggregateIdentifier
    private Long id;
    private String name;
    private double price;
    private int quantity;
    public ProductStatus productStatus;
    public ProductAggregate(){}
    @CommandHandler
    public ProductAggregate(ProductCreatedCommand command){
        if (command.getName() == null ||
command.getName().isEmpty()) {
            throw new IllegalArgumentException("Name cannot
be empty");
        }
        AggregateLifecycle.apply(new ProductCreatedEvent(
                command.getId(),
                command.getName(),
                command.getPrice(),
                command.getQuantity(),
                command.getProductStatus()));
    }

    @EventSourcingHandler
    public void on(ProductCreatedEvent event) {
```

```
        this.id = event.getId();
        this.name = event.getName();
        this.price = event.getPrice();
        this.quantity = event.getQuantity();
        this.productStatus=event.getProductStatus();
    }
}
```

ii. Controllers:

- Catégorie:

```
@RestController
@RequestMapping("/command/categorie")
@AllArgsConstructor
@Service
public class CategorieCommandController {
    private CommandGateway commandGateway;
    @PostMapping(path = "/create")
    public CompletableFuture<String>
createOwner(@RequestBody CreateCategorieRequestDTO
createCategorieRequestDTO) {
        CompletableFuture<String> response =
commandGateway.send(new CategorieCreatedEvent(
                UUID.randomUUID().getMostSignificantBits(),
                createCategorieRequestDTO.getName(),
                createCategorieRequestDTO.getDescription()
        ));
        return response;
    }
}
```

- Product:

```
@RestController
@RequestMapping("/command/product")
@AllArgsConstructor
@Service
public class ProductCommandController {
    private CommandGateway commandGateway;
    @PostMapping(path = "/create")
    public CompletableFuture<String>
createOwner(@RequestBody CreateProductRequestDTO
createProductRequestDTO) {
```

```
        CompletableFuture<String> response =
commandGateway.send(new ProductCreatedEvent(
                UUID.randomUUID().getMostSignificantBits(),
                createProductRequestDTO.getName(),
                createProductRequestDTO.getPrice(),
                createProductRequestDTO.getQuantity(),
                createProductRequestDTO.getProductStatus()
        ));
        return response;
    }
}
```

b. Query:

i. Entities:

- Catégorie:

```
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Categorie {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String Description;
}
```

- Product:

```
@Entity @Data @NoArgsConstructor @AllArgsConstructor
@Builder
public class Product {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
    private int quantity;
    public ProductStatus productStatus;
    @Transient
```

```
    private Categorie category;
}
```

ii. Enum:

```
public enum ProductStatus {
    Disponible, Rupture, Production, Abandon
}
```

iii. Repositories:

- Product:

```
@RepositoryRestResource
public interface ProductRepository extends
JpaRepository<Product, Long> {
}
```

- Catégorie:

```
@RepositoryRestResource
public interface CategorieRepository extends
JpaRepository<Categorie, Long> {
}
```

iv. Security:

```
@Configuration
public class KeycloakAdapterConfig {
    @Bean
    public KeycloakSpringBootConfigResolver
keycloakConfigResolver(){
        return new KeycloakSpringBootConfigResolver();
    }
}
```

And:

```
@KeycloakConfiguration
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfig extends
KeycloakWebSecurityConfigurerAdapter {
    @Override
```

```
    protected SessionAuthenticationStrategy
sessionAuthenticationStrategy() {
        return new
RegisterSessionAuthenticationStrategy(new
SessionRegistryImpl());
    }
    @Override
    protected void configure(AuthenticationManagerBuilder
auth) throws Exception {

auth.authenticationProvider(keycloakAuthenticationProvider
());
            }
    @Override
    protected void configure(HttpSecurity http) throws
Exception {
        super.configure(http);
        http.csrf().disable();

http.authorizeRequests().antMatchers("/h2-console/**").per
mitAll();
        http.headers().frameOptions().disable();

http.authorizeRequests().anyRequest().authenticated();
    }
}
```

v. Application Properties:

```
server.port=8082
spring.application.name=inventory-service
spring.config.import=optional:configserver:http
://localhost:8888
spring.zipkin.base-url=http://127.0.0.1:9411/
keycloak.realm=spring-boot-microservices-realm
keycloak.resource=spring-boot-client
keycloak.bearer-only=true
keycloak.auth-server-url=http://localhost:8080
```

```
keycloak.ssl-required=none
```

3. Order Service:
    a. Query:
        i. Entities:

- Ligne de Commande:

```
@Entity
@Table(name = "ligneCommande") @Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class ligneCommande {
    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;
    private int quantity;
    private double prixUnite;
    private double remise;
    @Transient
    private ProductItem product;
    @ManyToOne
    @JsonProperty(access =
JsonProperty.Access.WRITE_ONLY)
    private Order order;

    public double getAmount(){
        return
this.product.getPrice()*this.product.getQuantit
y()*(1- this.product.getRemise());
    }
}
```

- Order:

```
@Entity @Table(name = "orders") @Data
@NoArgsConstructor @AllArgsConstructor @Builder
public class Order {
   @Id @GeneratedValue(strategy =
GenerationType.IDENTITY)
   private Long id;
   private Date createdAt;
   private OrderStatus status;
   private Long customerId;
   @Transient
   private Customer customer;
   @OneToMany(mappedBy = "order")
   private List<ligneCommande> ligneCommandes;

   @OneToMany(mappedBy = "order")
   private List<ProductItem> products;

   public double getTotal(){
       double somme=0;
       for(ligneCommande pi:ligneCommandes){
           somme+=pi.getPrixUnite();
       }
       return somme;
   }
}
```

b. Commands:

i. Aggregate:

- Ligne de Commande:

```
public class ligneCommandeAggregate {
   @AggregateIdentifier
```

```
    private Long id;
    private int quantity;
    private double prixUnite;
    private double remise;
    public ligneCommandeAggregate(){}

    @CommandHandler
    public
ligneCommandeAggregate(LigneCommandeCreatedComm
and command){
        if (command.getQuantity() == 0) {
            throw new
IllegalArgumentException("Quantity cannot be
empty");
        }
        AggregateLifecycle.apply(new
LigneCommandeCreatedEvent(
                command.getId(),
                command.getQuantity(),
                command.getPrixUnite(),
                command.getRemise()));
    }

    @EventSourcingHandler
    public void on(LigneCommandeCreatedEvent
event) {
        this.id = event.getId();
        this.quantity = event.getQuantity();
        this.prixUnite = event.getPrixUnite();
        this.remise = event.getRemise();
    }
}
```

- Order:

```
public class OrderAggregate {
    @AggregateIdentifier
    private Long id;
    private Date createdAt;
    private OrderStatus status;
    private Long customerId;

    public OrderAggregate(){}

    @CommandHandler
    public OrderAggregate(OrderCreatedCommand
command){
        if (command.getStatus() == CANCELED ||
command.getStatus() == DELIVERED) {
            throw new
IllegalArgumentException("Order Delivered or
Cnaceled");
        }
        AggregateLifecycle.apply(new
OrderCreatedEvent(
                command.getId(),
                command.getCreatedAt(),
                command.getStatus(),
                command.getCustomerId()));
    }

    @EventSourcingHandler
    public void on(OrderCreatedEvent event) {
        this.id = event.getId();
        this.createdAt = event.getCreatedAt();
        this.status = event.getStatus();
        this.customerId = event.getCustomerId();
```

```
    }
}
```

ii. Controllers:

- Ligne de commande:

```
@RestController
@RequestMapping("/command/lignedecommande")
@AllArgsConstructor
@Service
public class ligneCommandeCommandController {
    private CommandGateway commandGateway;
    @PostMapping(path = "/create")
    public CompletableFuture<String>
createOwner(@RequestBody
CreateLigneCommandeRequestDTO
createLigneCommandeRequestDTO) {
        CompletableFuture<String> response =
commandGateway.send(new
LigneCommandeCreatedEvent(

UUID.randomUUID().getMostSignificantBits(),

createLigneCommandeRequestDTO.getQuantity(),

createLigneCommandeRequestDTO.getRemise(),

createLigneCommandeRequestDTO.getRemise()
        ));
        return response;
    }
```

- Order:

```
@RestController
```

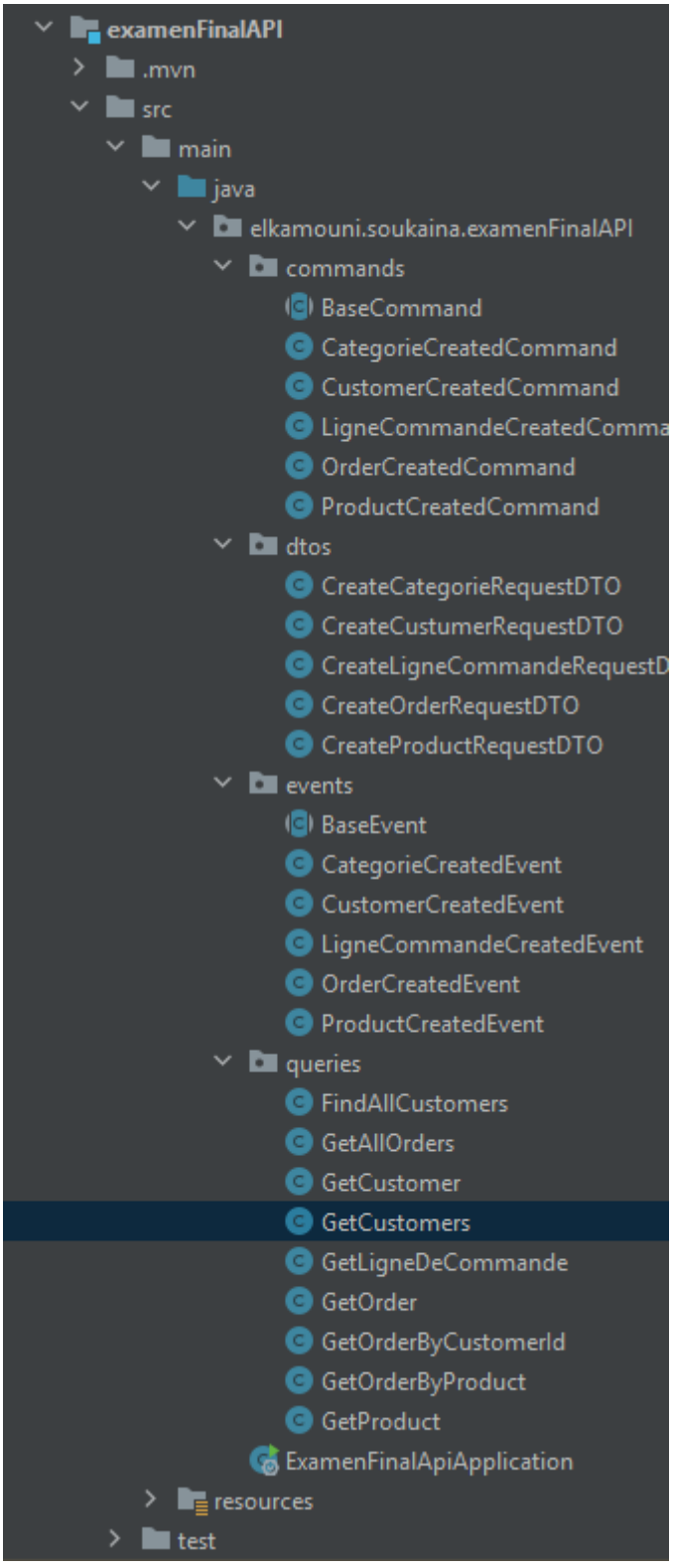
```
@RequestMapping("/command/order")
@AllArgsConstructor
@Service
public class OrderCommandController {
   private CommandGateway commandGateway;
   @PostMapping(path = "/create")
   public CompletableFuture<String>
createOwner(@RequestBody CreateOrderRequestDTO
createOrderRequestDTO) {
       CompletableFuture<String> response =
commandGateway.send(new OrderCreatedEvent(

UUID.randomUUID().getMostSignificantBits(),

createOrderRequestDTO.getCreatedAt(),

createOrderRequestDTO.getStatus(),

createOrderRequestDTO.getCustomerId()
       ));
       return response;
   }
}
```

c. Application Properties:

```
server.port=8083
spring.application.name=order-service
spring.config.import=optional:configserver:http
://localhost:8888
logging.level.elkamouni.soukaina.orderservice.Q
uery.services.CustomerRestClientService=debug
logging.level.elkamouni.soukaina.orderservice.Q
uery.services.InventoryRestClientService=debug
```

```
feign.client.config.default.loggerLevel=full
keycloak.realm=spring-boot-microservices-realm
keycloak.resource=spring-boot-client
keycloak.bearer-only=true
keycloak.auth-server-url=http://localhost:8080
keycloak.ssl-required=none
spring.main.allow-bean-definition-overriding=tr
ue
```

4. Our API:


examenFinalAPI
.mvn
src
main
java
elkamouni.soukaina.examenFinalAPI
commands
BaseCommand
CategorieCreatedCommand
CustomerCreatedCommand
LigneCommandeCreatedComma
OrderCreatedCommand
ProductCreatedCommand
dtos
CreateCategorieRequestDTO
CreateCustumerRequestDTO
CreateLigneCommandeRequestD
CreateOrderRequestDTO
CreateProductRequestDTO
events
BaseEvent
CategorieCreatedEvent
CustomerCreatedEvent
LigneCommandeCreatedEvent
OrderCreatedEvent
ProductCreatedEvent
queries
FindAllCustomers
GetAllOrders
GetCustomer
GetCustomers
GetLigneDeCommande
GetOrder
GetOrderByCustomerId
GetOrderByProduct
GetProduct
ExamenFinalApiApplication
resources
test

Je vais juste me focaliser sur les commandes en relation avec le customer:

a. Commands:

```
public class CustomerCreatedCommand extends BaseCommand<Long>{
   @Getter
   private String name;
   @Getter private String email;
   @Getter private String tel;
   public CustomerCreatedCommand(Long id, String name, String
email, String tel) {
       super(id);
       this.email=email;
       this.name=name;
       this.tel=tel;
   }
}
```

b. DTOs:

```

@Data
@NoArgsConstructor
@AllArgsConstructor

public class CreateCustumerRequestDTO {
   private String name;
   private String email;
   private String tel;
}
```

c. Events:

```
public class CustomerCreatedEvent extends BaseEvent<Long>{
   @Getter private String name;
   @Getter private String email;
   @Getter private String tel;
   public CustomerCreatedEvent(Long id, String name, String email,
String tel) {
       super(id);
       this.email=email;
       this.name=name;
```

```
        this.tel=tel;
    }
}
```

5. Results: