

## Travaux Pratiques

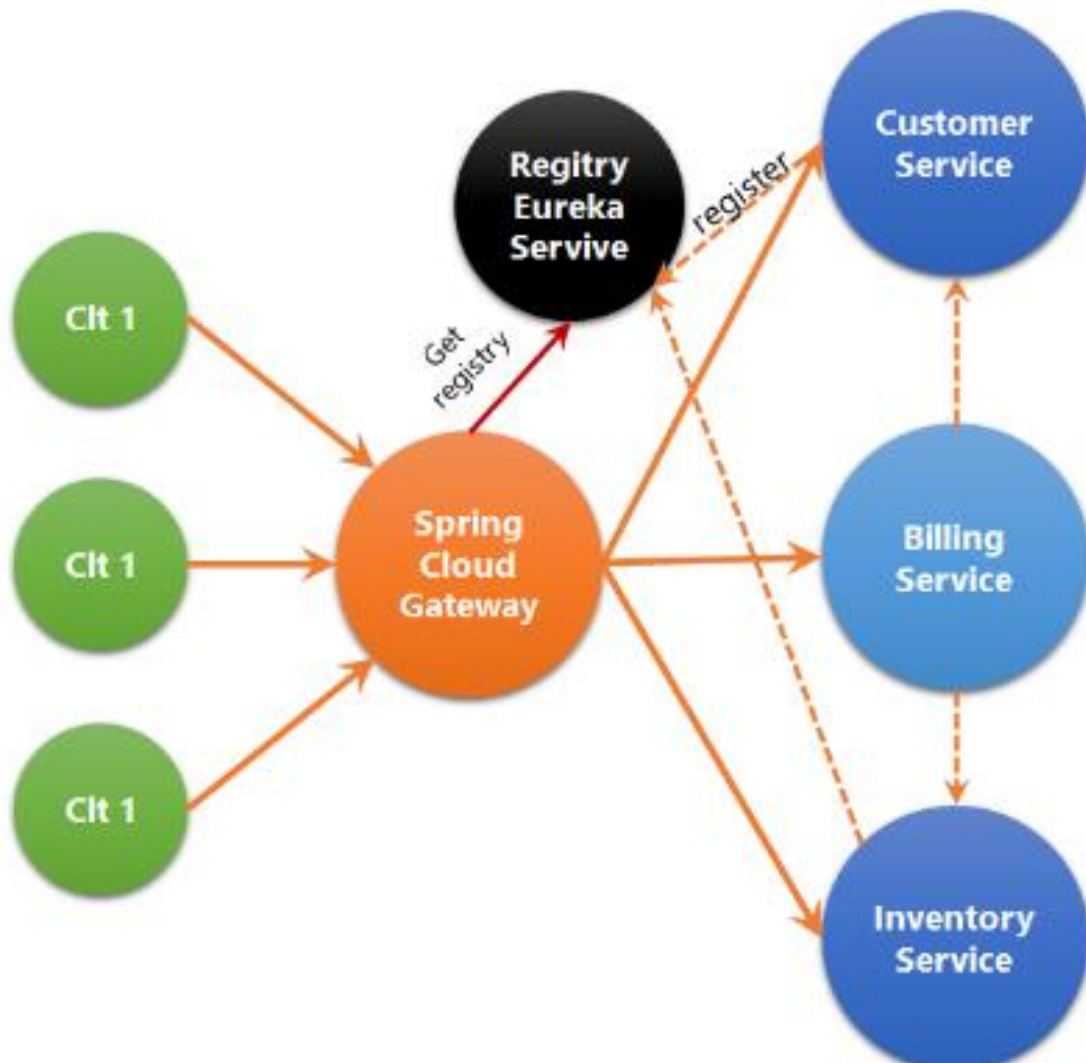
### FI BDCC S5 : TP de synthèse Systèmes Distribués, Stream et batch processing

**Mohamed HAMMANE**

- **Introduction et énoncé :**

L'objectif est de créer un système distribué basé sur les micro-services permettant de gérer les factures des clients en y intégrant un système de sécurité basé sur Keycloak, Un Bus de messagerie avec KAFKA, un service de Stream processing avec Kafka Streams et un service de Batch Processing avec Spring Batch.

A la fin, nous projetons appliquer pour cette application les patterns CQRS et Event Sourcing.



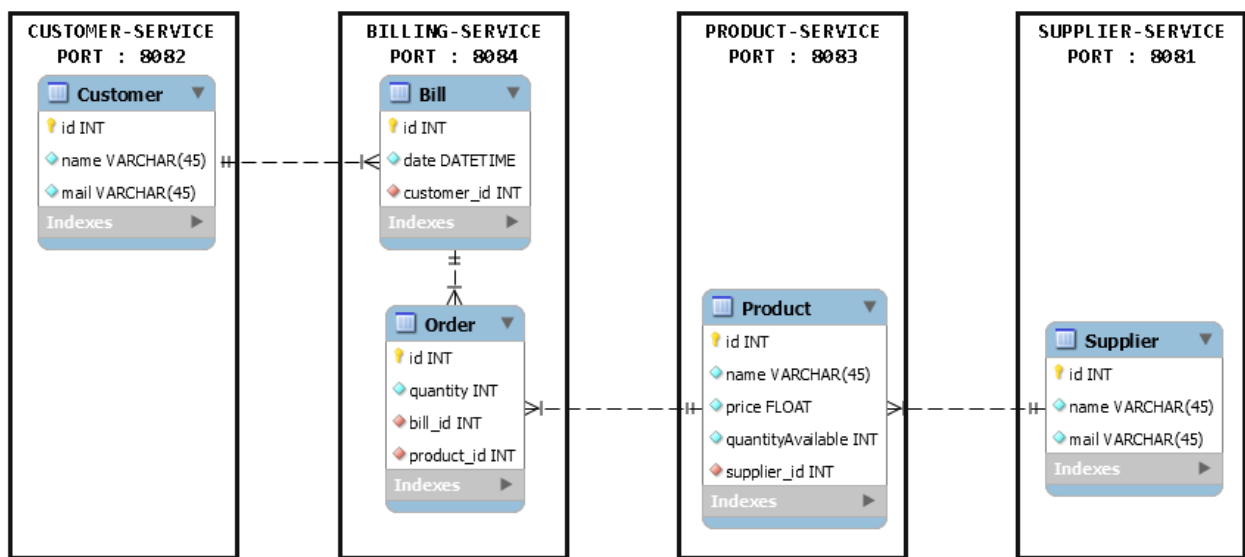
## ● Travail à faire :

1. Mettre en place les micro-services :
  - a. Customer-Service.
  - b. Inventory-Service.
  - c. Billing-Service.
  - d. Eureka Discovery Service.
  - e. Spring Cloud Gateway.
2. Mise en place du service de Sécurité avec Keycloak :
  - a. Mettre en place le serveur d'authentification OAuth2 Keycloak version 12.0.1.
  - b. Créer un Realm.
  - c. Le client à sécuriser en mode public client
  - d. Créer les rôles (USER, ADMIN, PRODUCT\_MANAGER, CUSTOMER\_MANAGER et BILLING\_MANAGER).
  - e. Créer quelques utilisateurs.
  - f. Affecter les rôles aux utilisateurs.
  - g. Tester l'authentification des utilisateurs en utilisant un client Rest comme ARC :
    - i. Authentification avec le mot de passe.
    - ii. Authentification avec le Refresh Token
  - h. Personnaliser le paramétrage des timeouts des tokens.
3. Sécuriser l'ensemble des micro-services fonctionnels en mode Bearer-Only en utilisant Spring Security et des adaptateurs Keycloak. On suppose que les micro-services ne sont accessible que pour les utilisateurs authentifiés avec leurs rôles respectifs : PRODUCT\_MANAGER, CUSTOMER\_MANAGER et BILLING\_MANAGER.
4. Développer une application Web Front End qui permet de gérer les produits, les clients et les factures en utilisant le Framework de votre choix : Angular, React ou Spring MVC avec Thymeleaf.
5. Sécuriser l'application FrontEnd en mode public client en mettant en place l'adaptateur Keycloak qui instaure un système d'authentification via Keycloak
6. Personnaliser la sécurité de la partie frontend en ajoutant les autres fonctionnalités fournies par Keycloak :
  - a. Auto-inscription des utilisateurs
  - b. Politique des mots de passe.
  - c. Double authentification OTP.
  - d. ...
7. Mise en place d'une solution de messagerie asynchrone avec le Broker KAFKA :
  - a. Mettre en place le Broker KAFKA.
  - b. Créer un micro-service Spring Boot qui permet de simuler un Producer KAFKA qui permet d'envoyer à un topic « FACTURATION » à chaque seconde un message contenant le numéro de la facture, le nom du client et le montant de la facture.
  - c. Créer un Micro-service Spring Boot qui permet de consommer les messages du Topic « FACTURATION » et de les enregistrer dans sa propre base de données et dans un fichier CSV, avec Une API REST qui permet de consulter les factures.

- d. Créer un Micro-service de Real Time Data Analytics en mode Stream Processing utilisant KAFKA Streams qui permet de traiter en temps réel les messages du Topic « FACTURATION » en produisant des statistiques comme le Total des factures reçus pour les 5 dernières secondes et le total des factures de chaque client.
8. Proposer une solution d'intégration de du BROKER KAFKA dans votre application.
9. Mettre en place un micro-service de batch processing avec Spring Batch permettant de traiter les données du fichier CSV de facturation produit par l'application.

## • Réalisation :

### 1. Mettre en place les micro-services :



Après avoir démarrer tous les services :


Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (1)	(1)	UP (1) - localhost:billing-service:8084
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - localhost:customer-service:8082
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - localhost:gateway-service:8888
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - localhost:product-service:8083
SUPPLIER-SERVICE	n/a (1)	(1)	UP (1) - localhost:supplier-service:8081


## 2. Mise en place du service de Sécurité avec Keycloak :


a) Mettre en place le serveur d'authentification OAuth2 Keycloak version 12.0.1 :





Welcome to **Keycloak**


**Administration Console**  
Please create an initial admin user to get started.  
  
Username  
Medomane  
  
Password  
.....  
  
Password confirmation  
.....  
  
[Create](#)

**Documentation >**  
  
User Guide, Admin REST API and Javadocs

**Keycloak Project >**

**Mailing List >**

**Report an issue >**



# Sign in to your account

Username or email  
Medomane

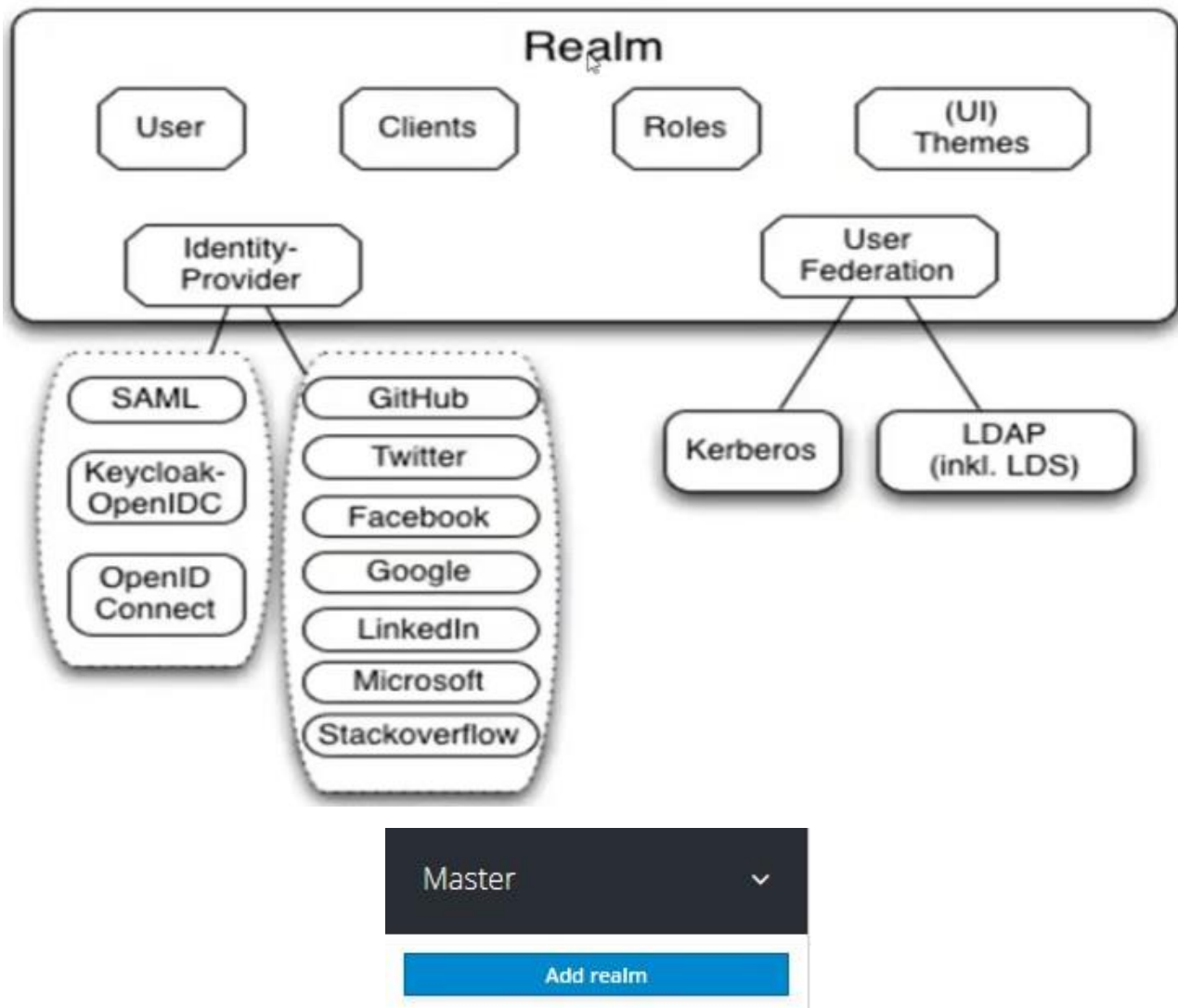
  

Password  
.....|

[Sign In](#)

b) Créer un Realm :



On va nommer notre realm "MySynthesisLab-realm" :

Add realm

Import

Name \*

Enabled ☒ ON

c) Le client à sécuriser en mode public client :

## Add Client

Import

Client ID \*

Client Protocol

Root URL

## Medomane-app

[Settings](#)
[Roles](#)
[Client Scopes](#)
[Mappers](#)
[Scope](#)
[Revocation](#)
[Sessions](#)
[Offline Access](#)
[Installation](#)

Client ID

Name

Description

Enabled ☒ ON

Always Display in Console ☐ OFF

Consent Required ☐ OFF

Login Theme

Client Protocol

Access Type

Standard Flow Enabled ☒ ON

d) Créer les rôles (USER, ADMIN, PRODUCT\_MANAGER, CUSTOMER\_MANAGER et BILLING\_MANAGER) :

## Roles

[Realm Roles](#)
[Default Roles](#)

Role Name	Composite	Description	Actions	
ADMIN	False		Edit	Delete
BILLING_MANAGER	False		Edit	Delete
CUSTOMER_MANAGER	False		Edit	Delete
PRODUCT_MANAGER	False		Edit	Delete
USER	False		Edit	Delete
offline_access	False	`\${role_offline-access}`	Edit	Delete
uma_authorization	False	`\${role_uma_authorization}`	Edit	Delete

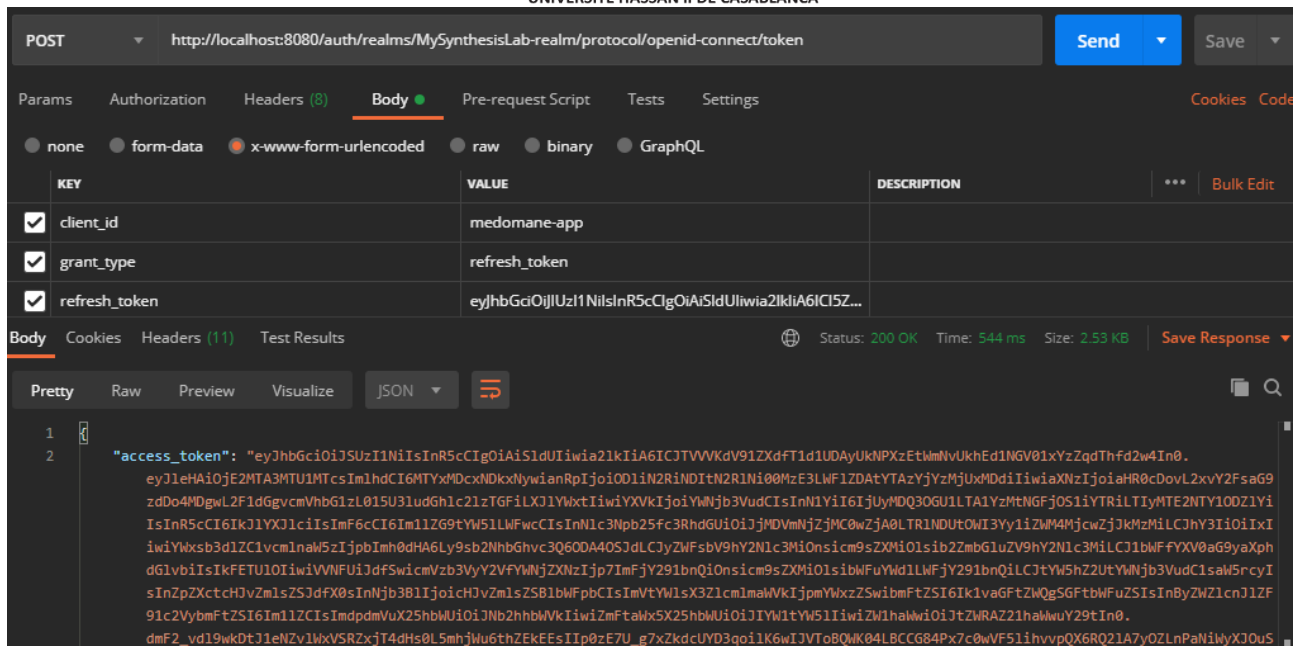
f) Affecter les rôles aux utilisateurs :

Même chose pour les autres utilisateurs.

- g) Tester l'authentification des utilisateurs en utilisant un client Rest :

- Authentification avec le mot de passe :

- Authentification avec le Refresh Token :



#### h) Personnaliser le paramétrage des timeouts des tokens :

SSO Session Idle     Hours 

Access Token Lifespan     Minutes 

### 3. Sécurisé l'ensemble des micro-services fonctionnels :

a) Supplier service :

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    super.configure(http);
    http.authorizeRequests()
        .antMatchers(HttpMethod.GET).permitAll()
        .antMatchers(HttpMethod.POST).hasAuthority("SUPPLIER_MANAGER")
        .antMatchers(HttpMethod.DELETE).hasAuthority("SUPPLIER_MANAGER");
    http.csrf().disable();
}
```

b) Product service :

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    super.configure(http);
    http.authorizeRequests()
        .antMatchers(HttpMethod.GET).permitAll()
        .antMatchers(HttpMethod.POST).hasAuthority("PRODUCT_MANAGER")
        .antMatchers(HttpMethod.DELETE).hasAuthority("PRODUCT_MANAGER");
    http.csrf().disable();
}
```

c) Customer service :

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



```

    .antMatchers(HttpMethod.GET).permitAll()
    .antMatchers(HttpMethod.POST).hasAuthority("CUSTOMER_MANAGER")
    .antMatchers(HttpMethod.DELETE).hasAuthority("CUSTOMER_MANAGER");
    http.csrf().disable();
  }

```

d) Billing service :

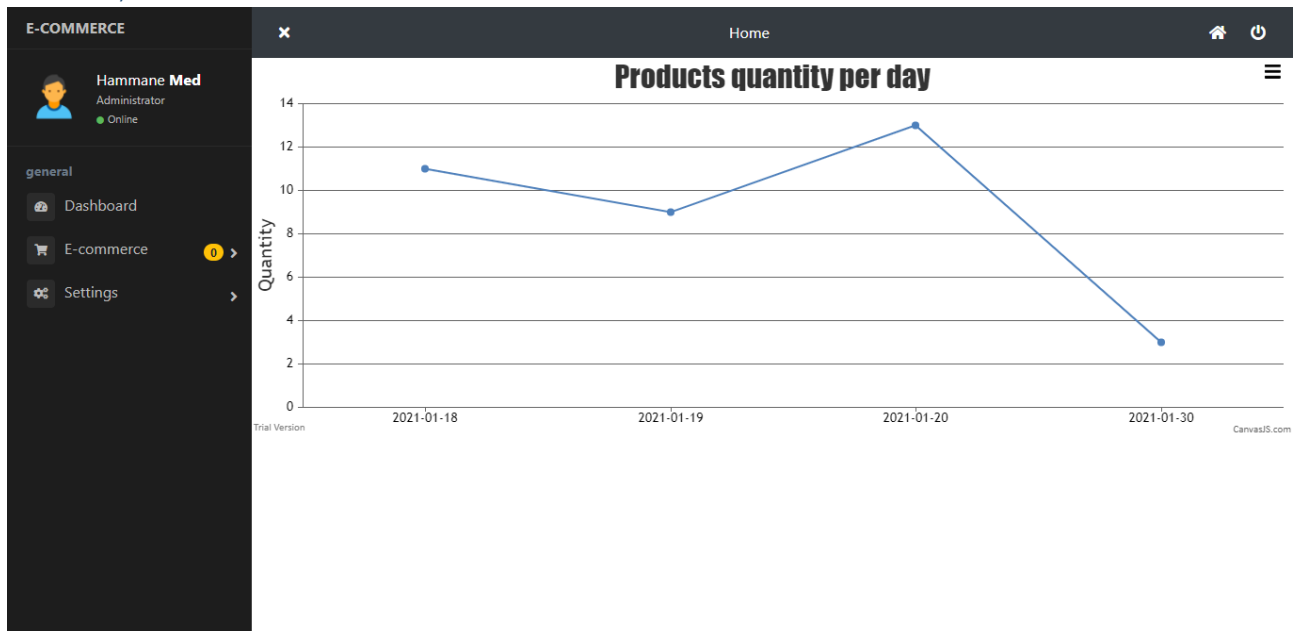
```

@Override
protected void configure(HttpSecurity http) throws Exception {
    super.configure(http);
    http.authorizeRequests()
        .antMatchers(HttpMethod.GET).permitAll()
        .antMatchers(HttpMethod.POST).authenticated();
    http.csrf().disable();
}

```

#### 4. Développer une application Web Front End qui permet de gérer les produits, les clients et les factures en utilisant Angular :

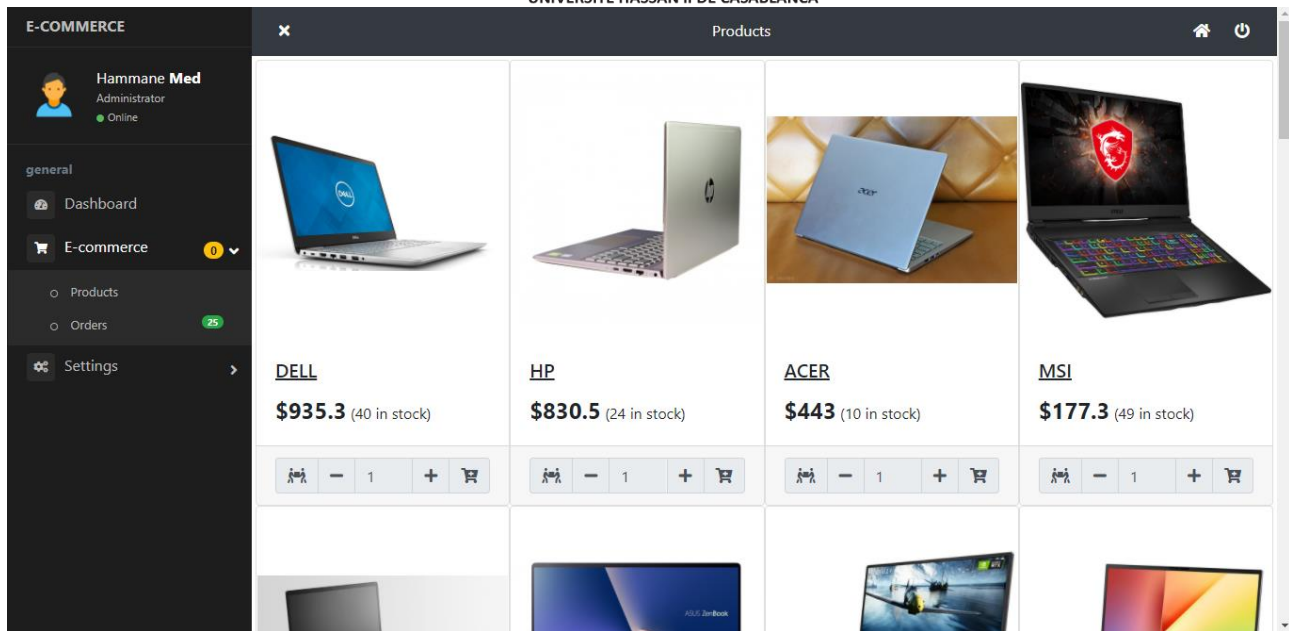
a) Dashboard :



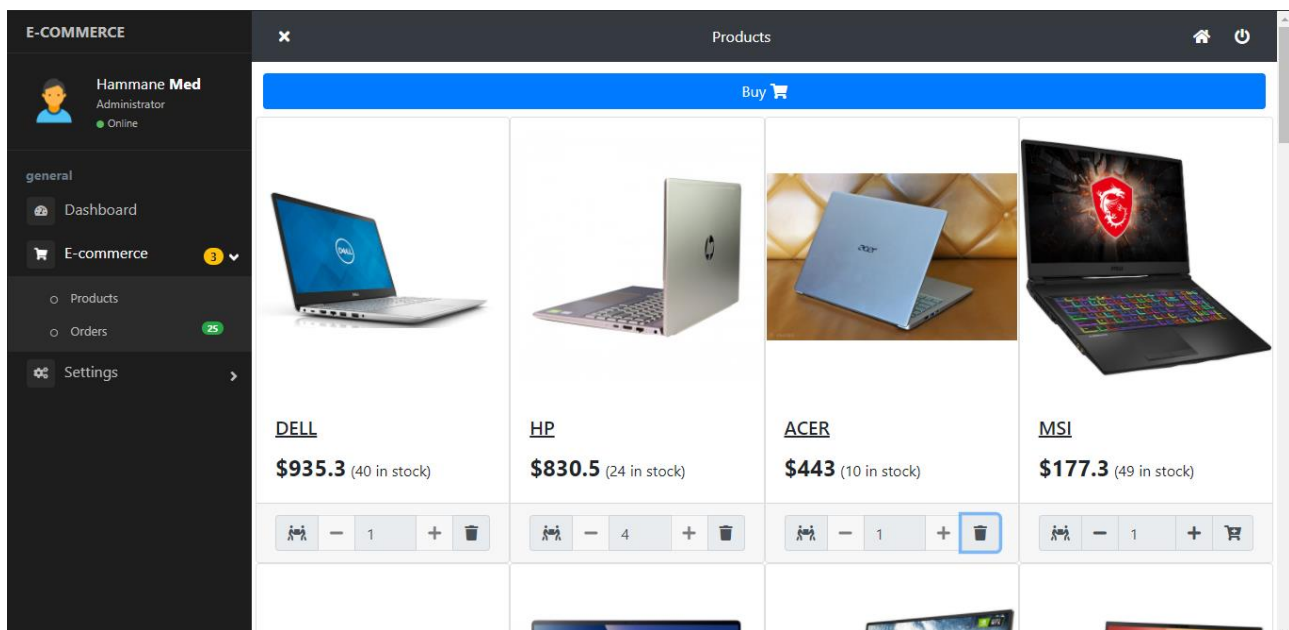
Cette page permet d'afficher le nombre d'article achetés par date.

b) E-Commerce :

La première partie concerne les produits après l'authentification le client peut acheter des produits en cliquant sur ajouter au panier .



après qu'il ajouter des produits au panier



Il peut les acheter en cliquant sur buy :

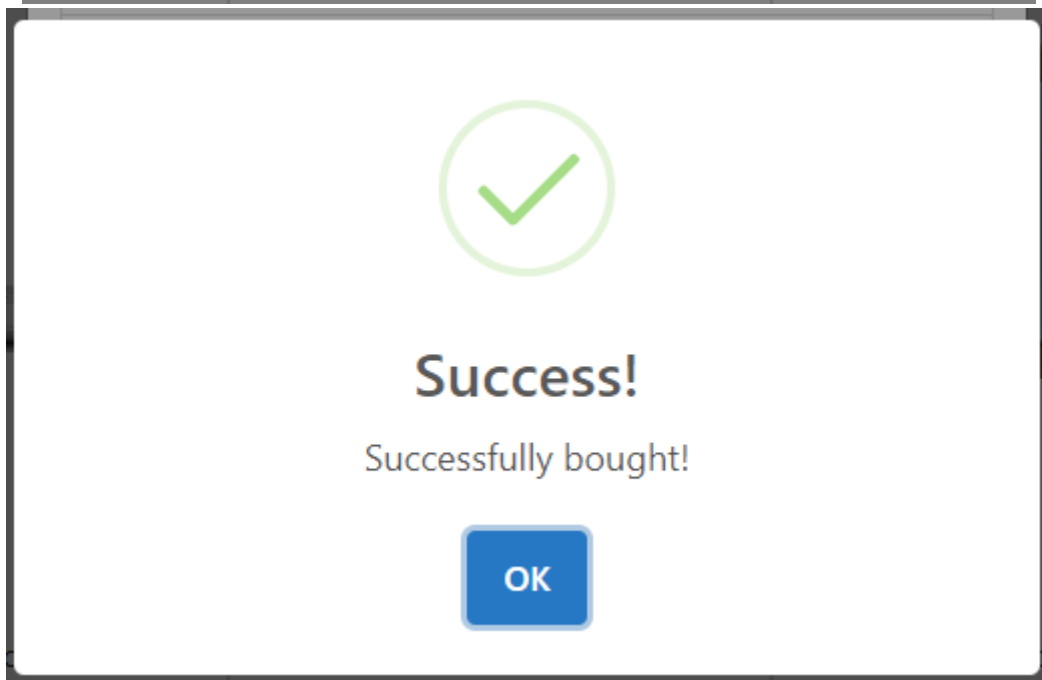
Billing : \$4700.3

1 of DELL with : \$935.3

4 of HP with : \$3322

1 of ACER with : \$443

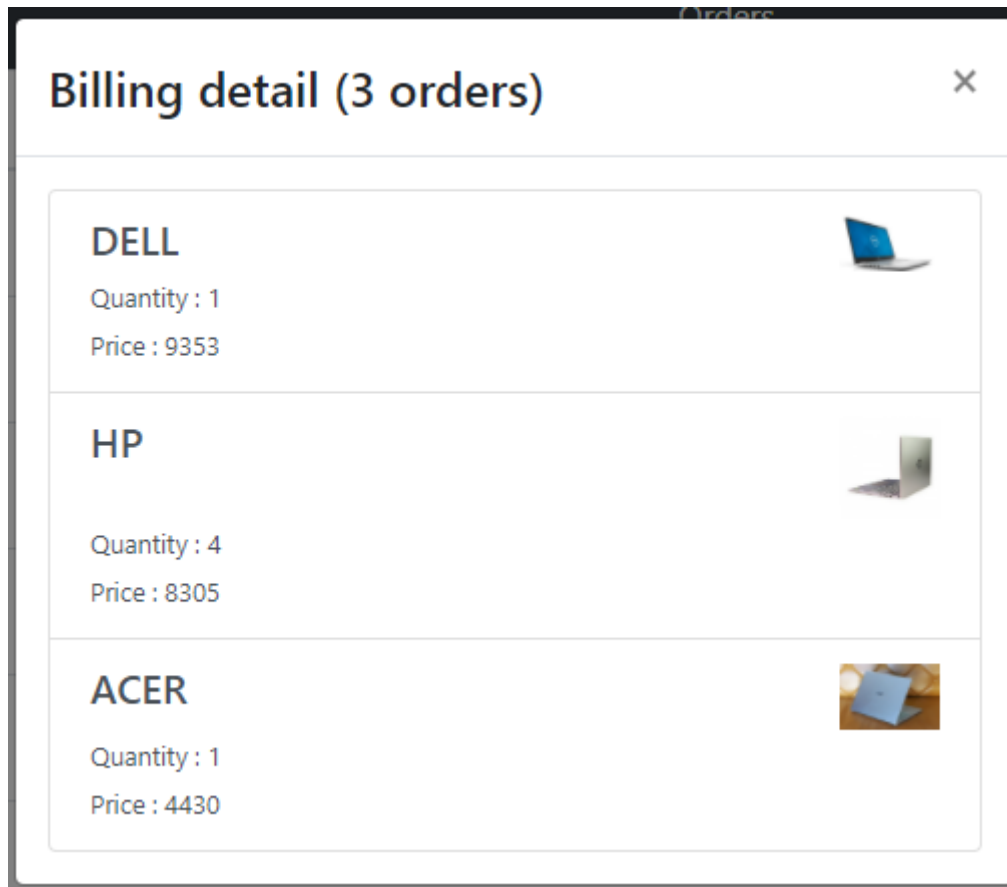
Buy



Et la deuxième partie concerne les factures :

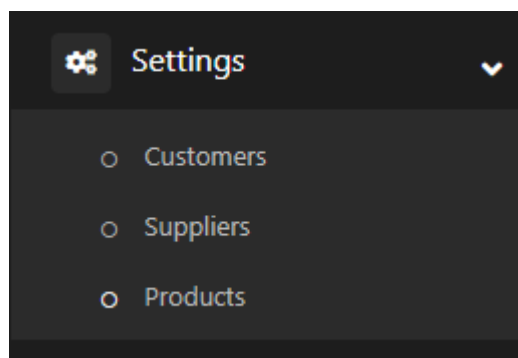
E-COMMERCE		Orders		
<div> <div>Hammane Med</div> <div>Administrator</div> <div>Online</div> </div> <div> <div>general</div> <div>Dashboard</div> <div>E-commerce 0</div> <div>Products</div> <div>Orders 28</div> <div>Settings</div> </div>	<div>Bill date</div> <div>2021-01-20</div> <div>2021-01-19</div> <div>2021-01-18</div> <div>2021-01-30</div> <div>2021-01-30</div>	Details		
		Orders		
		Orders		
		Orders		
		Orders		
		Orders		

Pour chaque facture on peut voir la liste des ordres :




c) Paramètres :

Cette partie concerne les paramètres des produits, fournisseurs et produits :



Ici la gestion des clients :

E-COMMERCE





Hammane Med


Administrator

Online

general

Dashboard

E-commerce0

Settings

Customers

Suppliers

Products

Customers

Name	Email	Details
Mohamed	med@gmail.com	<div><div></div><div></div></div>
Gueddi	gueddi@gmail.com	<div><div></div><div></div></div>
Anssari	anssari@gmail.com	<div><div></div><div></div></div>
Aymane	aymane@gmail.com	<div><div></div><div></div></div>
Yasser	yasser@gmail.com	<div><div></div><div></div></div>

## Add Customer

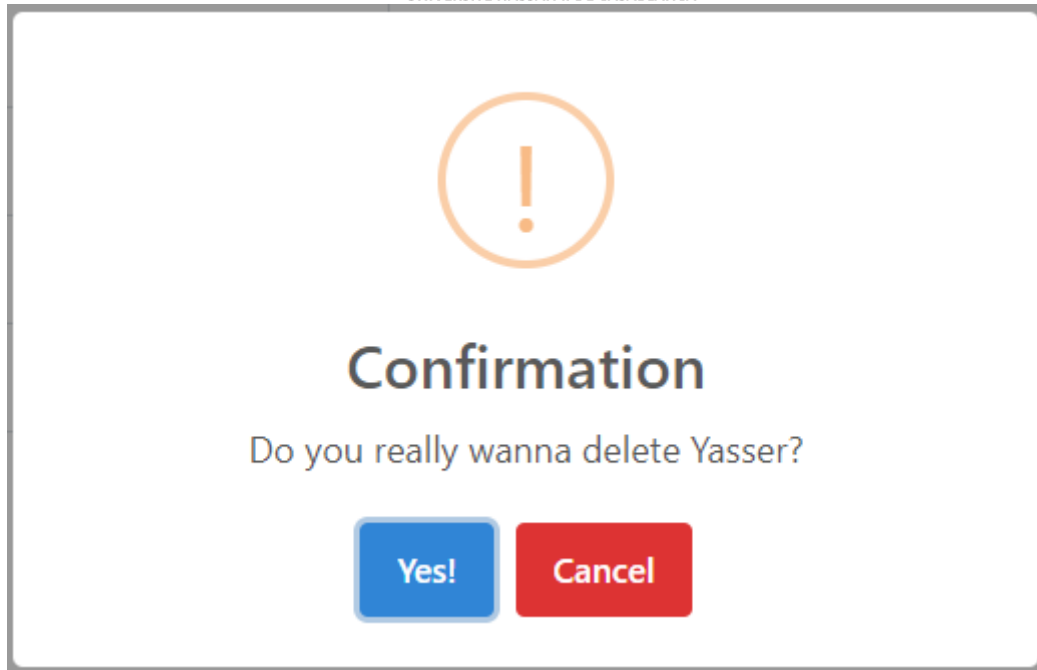
Name \*:
Email \*:

Add
Cancel















## Edit Customer

Name \*:
Email \*:






Validate
Cancel




Aussi pour la gestion des fournisseurs :

E-COMMERCE		Suppliers			
 <b>Hammane Med</b> Administrator <span style="color: green;">●</span> Online		Name	Email	Details	
general		Supplier1	supplier1@gmail.com		
 Dashboard		Supplier2	supplier2@gmail.com		
 E-commerce <span style="background-color: yellow; border-radius: 50%; padding: 0 5px;">0</span>		Supplier3	supplier3@gmail.com		
 Settings		Supplier4	supplier4@gmail.com		
<ul style="list-style-type: none"> <li>Customers</li> <li>Suppliers</li> <li>Products</li> </ul>		Supplier5	supplier5@gmail.com		

Et finalement la gestion des produits :

Name	Price	Available Quantity	Supplier	Details 
DELL	9353	39	Supplier1	 
HP	8305	20	Supplier1	 
ACER	4430	9	Supplier1	 
MSI	1773	49	Supplier1	 
MAC	7042	15	Supplier1	 
ASUS	4154	17	Supplier2	 
LEGION	8809	17	Supplier2	 
ASUS	3697	32	Supplier2	 
ASUS	1602	3	Supplier2	 
HP	3569	2	Supplier2	 


## Add Product




Name \*:

Price \*:

Available Quantity \*:

Supplier \*:


Edit Product




Name \*:

Price \*:

Available Quantity \*:

Supplier \*:



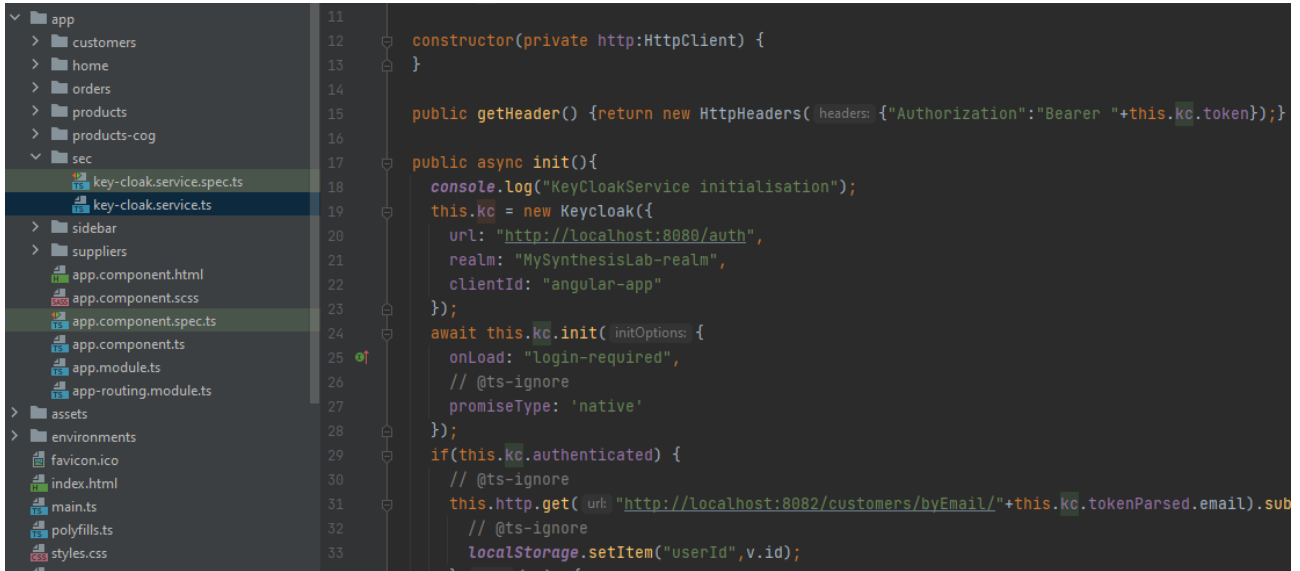
## Confirmation

Do you really wanna delete LEGION?



## 5. Sécuriser l'application FrontEnd en mode public client en mettant en place l'adaptateur Keycloak qui instaure un système d'authentification via Keycloak :

Après l'installation de la bibliothèque KeyCloak j'ai créé un service pour gérer la sécurité coté client ainsi le système d'authentification.



```

11
12 constructor(private http:HttpClient) {
13 }
14
15 public getHeader() {return new HttpHeaders( headers: {"Authorization":"Bearer "+this.kc.token});}
16
17 public async init(){
18   console.log("KeyCloakService initialisation");
19   this.kc = new Keycloak({
20     url: "http://localhost:8080/auth",
21     realm: "MySynthesisLab-realm",
22     clientId: "angular-app"
23   });
24   await this.kc.init( initOptions: {
25     onLoad: "login-required",
26     // @ts-ignore
27     promiseType: 'native'
28   });
29   if(this.kc.authenticated) {
30     // @ts-ignore
31     this.http.get( url: "http://localhost:8082/customers/byEmail/"+this.kc.tokenParsed.email).sub
32     // @ts-ignore
33     localStorage.setItem("userId",v.id);
  
```

## 6. Fonctionnalités :

Et aussi j'ai ajouté quelques fonctionnalités de keycloak dans la partie front-end.

## 7. Solution Kafka :

La première chose j'ai créé un Producer (le micro service kafka-service) qui génère des Orders pour chaque Costumer

```

public static void main(String[] args) { SpringApplication.run(KafkaServiceApplication.class, args); }

@Bean
CommandLineRunner start(KafkaTemplate<String, String> kafkaTemplate, CustomerRestClient customerRestClient, ProductRestClient productRestClient) {
  return args -> {
    var customers :int = customerRestClient.count();
    var products :int = productRestClient.count();
    Executors.newScheduledThreadPool( corePoolSize: 1).scheduleAtFixedRate(
      ()->{
        var customerId :double = Math.ceil(Math.random()*customers);
        int maxOrders = (int) Math.ceil(Math.random()*5);
        var orders = new ArrayList<Order>();
        for(var i=0;i<maxOrders;i++){
          var prodId :double = Math.ceil(Math.random()*products);
          var quantity :double = Math.ceil(Math.random()*10);
          orders.add(getOrder(quantity,prodId));
        }
        try {
          var order :String = new ObjectMapper().writeValueAsString(orders);
          kafkaTemplate.send(MyGlobals.TOPIC,String.valueOf(customerId),order);
        } catch (JsonProcessingException e) {
          e.printStackTrace();
        }
      }, initialDelay: 1, period: 1, TimeUnit.SECONDS
    );
  };
}
  
```

Puis il envoie le résultat dans un topic(FACTURATION).

Puis on j'ai créé le spring boot consumer qui effectue des achats et génère des factures dans le topic StreamTopic

```
@Service
public class Consumer {
    final KafkaTemplate<String, String> kafkaTemplate;

    public Consumer(KafkaTemplate<String, String> kafkaTemplate) { this.kafkaTemplate = kafkaTemplate; }

    @KafkaListener(topics = MyGlobals.TOPIC, groupId = "group1")
    public void onMessage(ConsumerRecord<String, String> message) throws Exception {
        var customerId : Long = Long.parseLong(message.key().replace( oldChar: '0', newChar: ' ').replace( oldChar: '.', newChar: ' ').trim());
        var orders : Order[] = new ObjectMapper().readValue(message.value(), Order[].class);
        System.out.println("*****Buy("+customerId+"*****");
        MyGlobals.buy(orders, customerId);
        kafkaTemplate.send( topic: "StreamTopic", String.valueOf(customerId), message.value());
    }
}
```

Puis on a maintenant notre Kafka stream consumer qui va faire real-time analytics de notre data pour l'envoyer par la suite à notre application front-end

Et voici la partie Stream processing :

On a d'abord la partie configuration de notre consumer

```
Properties properties=new Properties();
properties.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
properties.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-consumer");
properties.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
properties.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass().getName());
properties.put(StreamsConfig.COMMIT_INTERVAL_MS_CONFIG, 1000);
```

Puis la partie du stream processing :

```
var sb = new StreamsBuilder();
var kStream : KStream<String, String> = sb.stream( topic: "StreamTopic", Consumed.with(Serdes.String(), Serdes.String()));
kStream
    .map((k,v)->{
        try {
            var sum = (Long)(Arrays.stream(new ObjectMapper().readValue(v, Order[].class)).mapToDouble( Order::getQuantity).sum());
            return KeyValue.pair(k, sum);
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
            return KeyValue.pair(k, 0L);
        }
    }) KStream<String, Long>
    .groupByKey(Grouped.with(Serdes.String(), Serdes.Long())) KGroupedStream<String, Long>
    .windowedBy(TimeWindows.of(Duration.ofSeconds(5))) TimeWindowedKStream<String, Long>
    .reduce(Long::sum) KTable<Windowed<String>, Long>
    .toStream() KStream<Windowed<String>, Long>
    .map((k,v) -> KeyValue.pair(k.key(), v.toString())) KStream<String, String>
    .peek((k,v)-> System.out.println(k+" --> "+v))
    .to( s: "streamKafkaTopic", Produced.with(Serdes.String(), Serdes.String()));

var kStreams = new KafkaStreams(sb.build(), properties);
kStreams.start();
```

Cette partie se charge de grouper les factures par client puis calculer la somme des quantités dans un window de 5 seconds puis envoyer le résultat dans le topic streamKafkaTopic (topic qu'on va

utiliser pour la partie front-end en utilisant webflux websockets avec la bibliothèque Angular NGXS).

Et finalement pour la partie back-end j'ai créé un micro-service (reactivekafka) qui va se charger d'envoyer les messages du topic streamKafkaTopic vers la partie Angular en utilisant webflux.

Et voici le service :

```
@Service
public class KafkaServiceImpl implements KafkaService {
    private final Flux<ReceiverRecord<String, String>> testTopicStream;
    KafkaServiceImpl() {

        Properties kafkaProperties = new Properties();

        kafkaProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        kafkaProperties.put(ConsumerConfig.CLIENT_ID_CONFIG, "reactive-
consumer");
        kafkaProperties.put(ConsumerConfig.GROUP_ID_CONFIG, "sample-group");
        kafkaProperties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        kafkaProperties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        kafkaProperties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG,
"earliest");

        ReceiverOptions<String, String> receiverOptions =
ReceiverOptions.create(kafkaProperties);

        testTopicStream = createTopicCache(receiverOptions);
    }

    public Flux<ReceiverRecord<String, String>> getTestTopicFlux() {
        return testTopicStream;
    }

    private <T, G> Flux<ReceiverRecord<T, G>>
createTopicCache(ReceiverOptions<T, G> receiverOptions) {
        ReceiverOptions<T, G> options =
receiverOptions.subscription(Collections.singleton("streamKafkaTopic"));
        return KafkaReceiver.create(options).receive().cache();
    }
}
```

Et la configuration:

```
@Configuration
public class ReactiveWebSocketConfiguration {

    private final WebSocketHandler webSocketHandler;

    public ReactiveWebSocketConfiguration(WebSocketHandler webSocketHandler) {
        this.webSocketHandler = webSocketHandler;
    }

    @Bean
    public HandlerMapping webSocketHandlerMapping() {
        Map<String, WebSocketHandler> urlMap = new HashMap<>();
        urlMap.put("/websocket", webSocketHandler);
    }
}
```

```

CorsConfiguration corsConfiguration = new CorsConfiguration();
corsConfiguration.addAllowedOrigin("http://localhost:4200");
SimpleUrlHandlerMapping handlerMapping = new SimpleUrlHandlerMapping();
handlerMapping.setOrder(1);
handlerMapping.setUrlMap(urlMap);
return handlerMapping;
}

@Bean
public WebSocketHandlerAdapter handlerAdapter() {
    return new WebSocketHandlerAdapter();
}
}

```

Puis le Component:

```

@Component
public class ReactiveWebSocketHandler implements WebSocketHandler {

    private static final ObjectMapper json = new ObjectMapper();

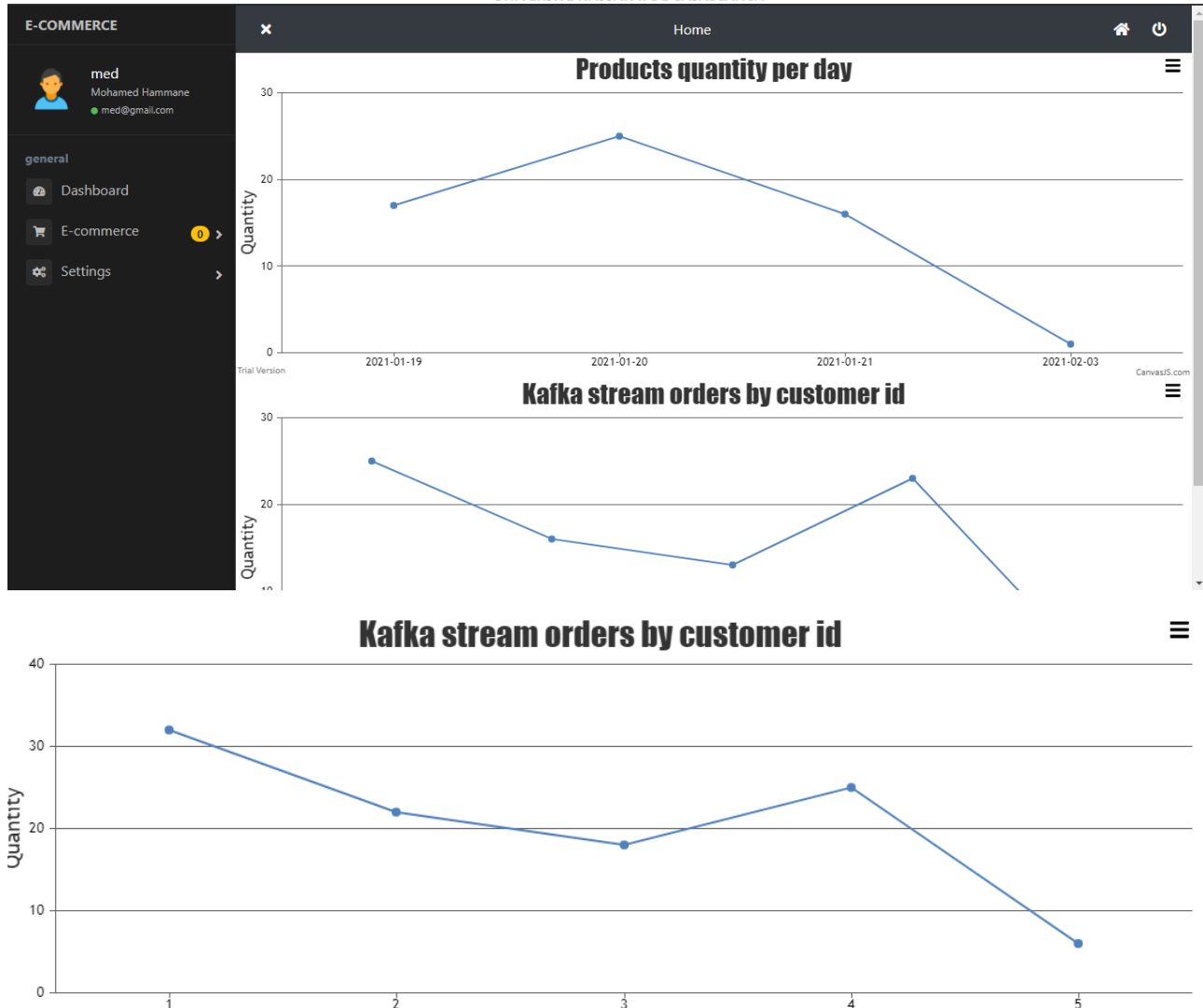
    private final KafkaService kafkaService;

    public ReactiveWebSocketHandler(KafkaService kafkaService) {
        this.kafkaService = kafkaService;
    }

    @Override
    public Mono<Void> handle(WebSocketSession webSocketSession) {
        return webSocketSession.send(kafkaService.getTestTopicFlux()
            .map(record -> {
                Message message = new Message("kafkaStream",
record.key()+"|"+record.value());
                System.out.println(record.key()+" --> "+record.value());
                try {
                    return json.writeValueAsString(message);
                } catch (JsonProcessingException e) {
                    return "Error while serializing to JSON";
                }
            })
            .map(webSocketSession::textMessage))
        .and(webSocketSession.receive().map(WebSocketMessage::getPayloadAsText).log());
    }
}

```

Pour la partie front-end avec Angular j'ai utilisé la bibliothèque NGXS puis avoir en temps réelle les données de notre topic streamKafkaTopic, et j'ai ajouter un Chart graphique qui represente ce changement en temps réelle :



### • Code source :

<https://github.com/Medomane/SynthesisLab> avec une vidéo démo.