

Travaux Pratiques

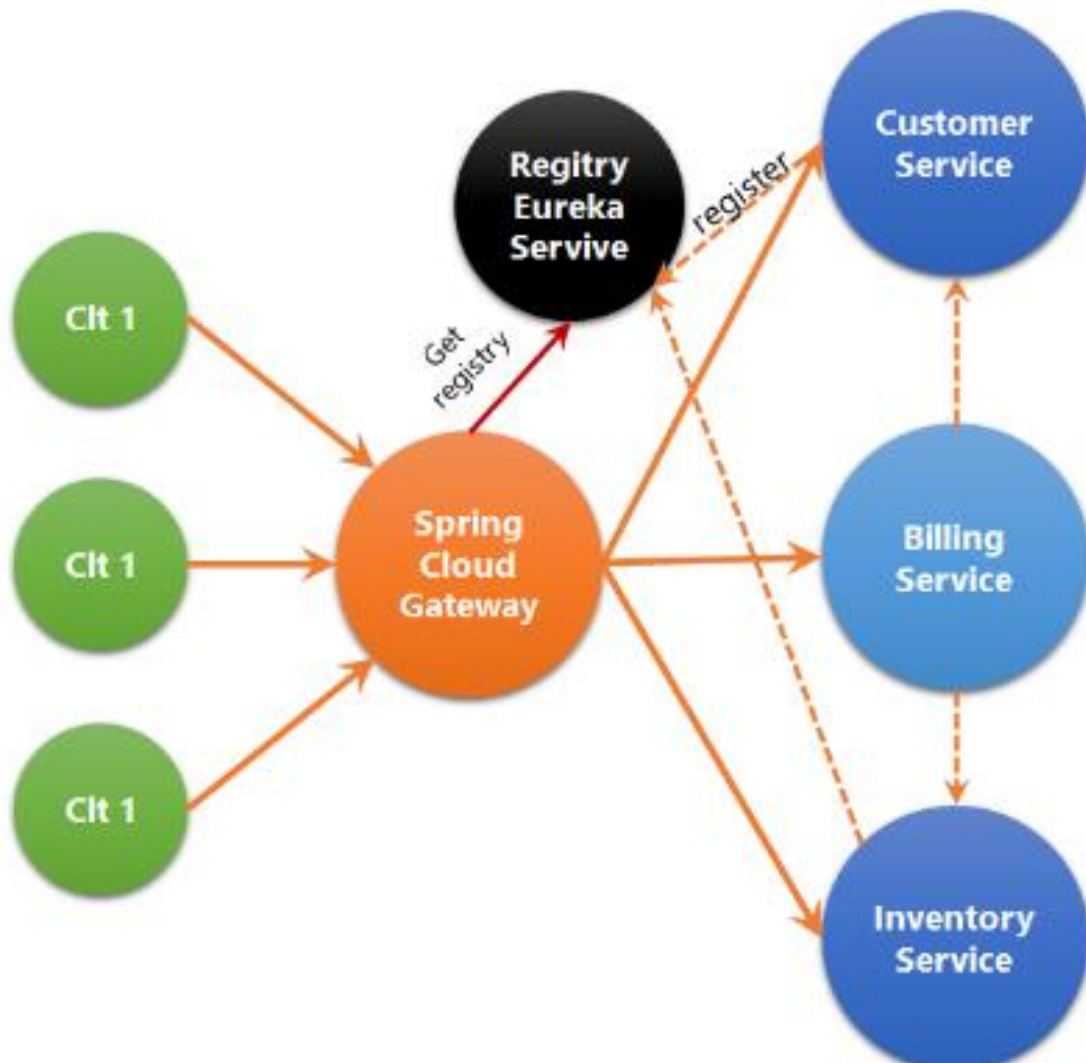
FI BDCC S5 : TP de synthèse Systèmes Distribués, Stream et batch processing

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.



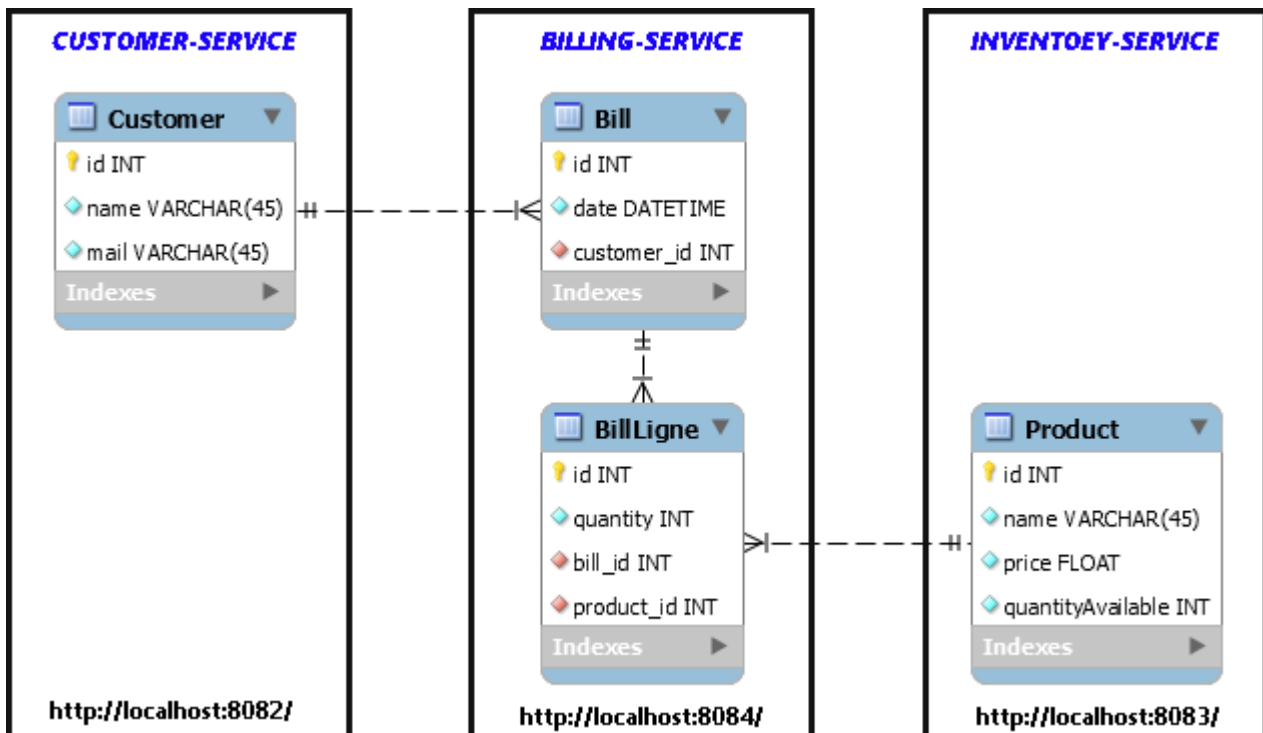
● Travail à faire :

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 timeout 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 accessibles 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
INVENTORY-SERVICE	n/a (1)	(1)	UP (1) - localhost:inventory-service:8083

On a démarré le service de sécurité qui utilise JWT, on ajoute des utilisateurs avec des rôles :

```

var admin : AppRole = appRoleRepository.save(new AppRole( id: null, role: "ADMIN"));
var user : AppRole = appRoleRepository.save(new AppRole( id: null, role: "USER"));
var cm : AppRole = appRoleRepository.save(new AppRole( id: null, role: "CUSTOMER_MANAGER"));
var bm : AppRole = appRoleRepository.save(new AppRole( id: null, role: "BILL_MANAGER"));
var pm : AppRole = appRoleRepository.save(new AppRole( id: null, role: "PRODUCT_MANAGER"));

var med = new AppUser( id: null, username: "Med", password: "1234", new ArrayList<>());
med.getRoles().add(admin);
med.getRoles().add(user);
med.Save(appUserRepository);
var gueddi = new AppUser( id: null, username: "Gueddi", password: "1234", new ArrayList<>());
gueddi.getRoles().add(user);
gueddi.getRoles().add(pm);
gueddi.Save(appUserRepository);
var yasser = new AppUser( id: null, username: "Yasser", password: "1234", new ArrayList<>());
yasser.getRoles().add(cm);
yasser.getRoles().add(user);
yasser.Save(appUserRepository);
var anssari = new AppUser( id: null, username: "Anssari", password: "1234", new ArrayList<>());
anssari.getRoles().add(pm);
anssari.getRoles().add(user);
anssari.Save(appUserRepository);
var aymane = new AppUser( id: null, username: "Aymane", password: "1234", new ArrayList<>());
aymane.getRoles().add(bm);
aymane.getRoles().add(user);
aymane.Save(appUserRepository);


```

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 >**



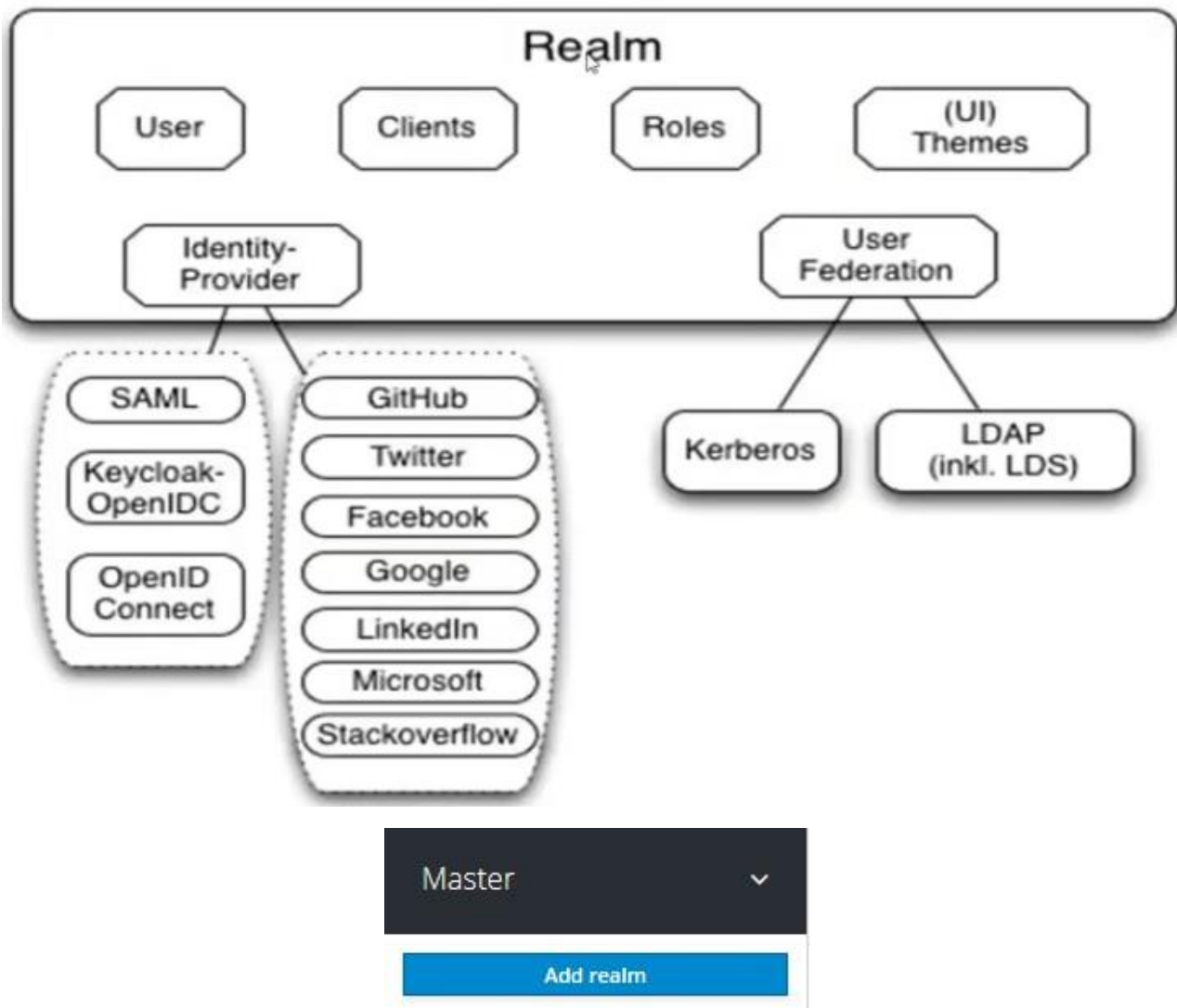
Sign in to your account

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 :

[illegible]

h) Personnaliser le paramétrage des timeouts des tokens :

SSO Session Idle   Hours 

Access Token Lifespan   Minutes 

- **Code source :**

<https://github.com/Medomane/SynthesisLab>