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

Examen Architecture Distribuée JEE

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC$_3$

Architecture Distribuée

Spring Cloud Gateway, Netflix Eureka Server

Spring Security, JWT, KAFKA Streams, Angular

Réalisé par :

- EL ASSIMI Ahmed

le lien de projet https://github.com/ahmed081/examen-backend-banque-app

2020/2021

Objectif

A. Conception :

1. Etablir une architecture technique du projet qui montre l'ensemble des micro-services de l'application
2. Etablir un diagramme de classes qui montre les entités métier de l'application.

B. Implémentation :

1. Implémentation du micro-service de gestion des clients :
    a. Créer l'entité JPA Client
    b. Créer l'interface JPARepository, basée sur Spring Data
    c. Créer une API Restful permettant de gérer les clients en utilisant Spring Data Rest.
    d. Tester ce micro-service

2. Implémentation du Micro-service pour la gestion des comptes et des opérations.
    a. Créer les entités JPA Compte et Operation
    b. Créer les interfaces JPA Repository basées sur Spring Data
    c. Créer une couche service (Interface et Implémentation) permettant d'implémenter les spécifications fonctionnelles suivantes en utilisant une interface OpenFeign pour l'accès aux données des clients :
        - Ajouter un compte
        - Effectuer un versement d'un montant dans un compte
        - Effectuter une retrait d'un montant d'un compte
        - Effectuer un virement d'un compte vers un autre
        - Consulter les opérations effectuées sur un compte en respectant la pagination.
        - Consulter un Compte contenant les informations sur le client.
        - Activer ou suspendre un compte
    d. Créer une RestController qui permet d'exposer les opérations de la couche service via RESTful.
    d. Créer une RestController qui permet d'exposer les opérations de la couche service via RESTful.
    e. Tester le micro-service
3. Mettre en place les services techniques :
    a. Spring Cloud Gateway
    b. Eureka Discovery Service
4. Tester l'application à travers la Gateway.
5. Mettre en place une solution de messagerie asynchrone basée sur le broker KAFKA qui permet de publier dans un TOPIC KAFKA toutes les opérations sur les comptes (Versement, retrait et virement) en utilisant Spring Cloud Gateway.
6. Mettre en place un Micro service permettant de faire du Data Analytics en implémentant une solution de Real time Stream Processing en utilisant KAFKA Streams et Spring Cloud Streams Functions. Ce micro-service devrait permettre de compter sur une fenêtre temporelle de 5 secondes, le nombre de d'opérations effectuées sur chaque compte. Pour tester Cette opération de streamprocessing, il est demandé de créer un Supplier Pooler qui permet de publier pour chaque seconde des opérations générée aléatoirement au niveau du micro-service comptes.
7. Mettre en place un service d'authentification basé sur Spring security et Json Web Token pour sécuriser l'accès aux micro-services fonctionnels.
8. Proposer une application front end Web basée sur Angular ou React.

Architecture et diagramme de classes

Architecture

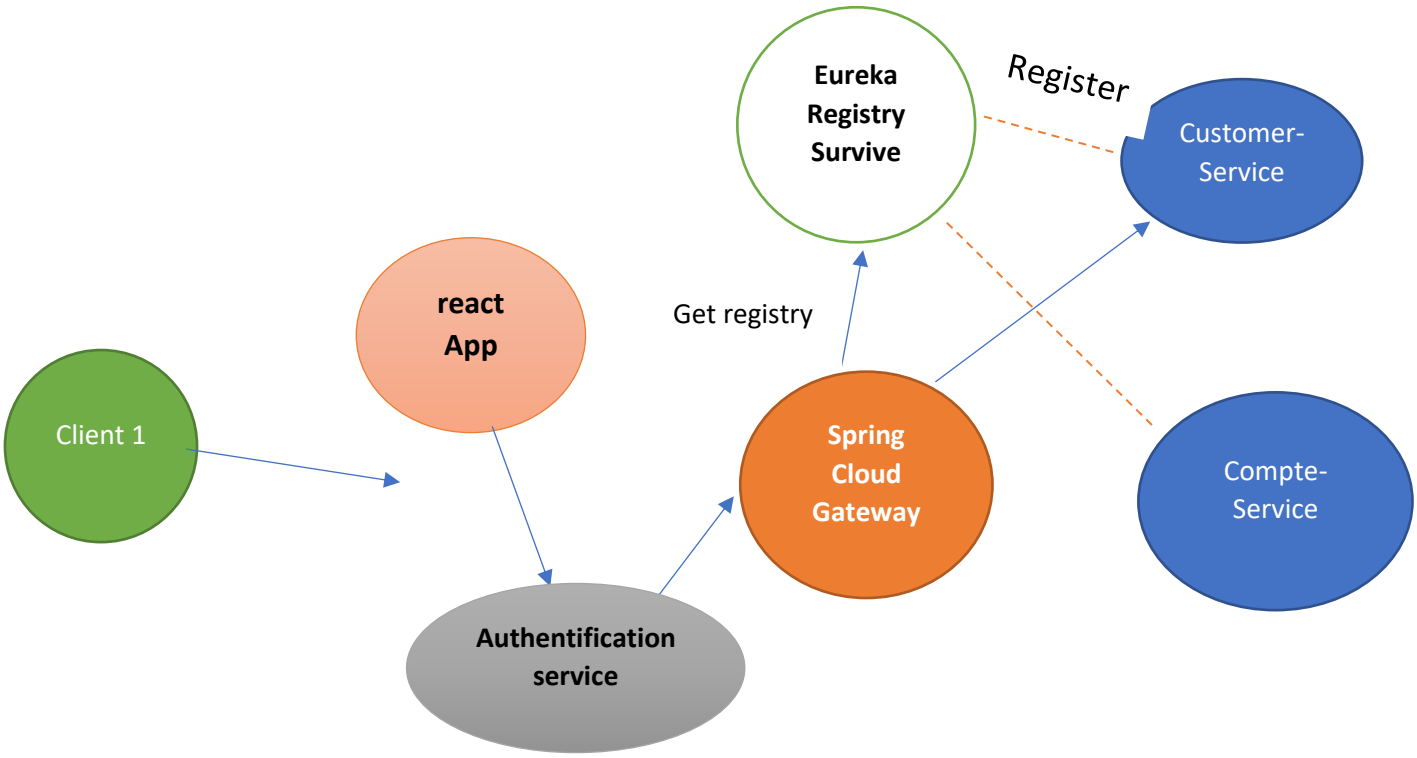
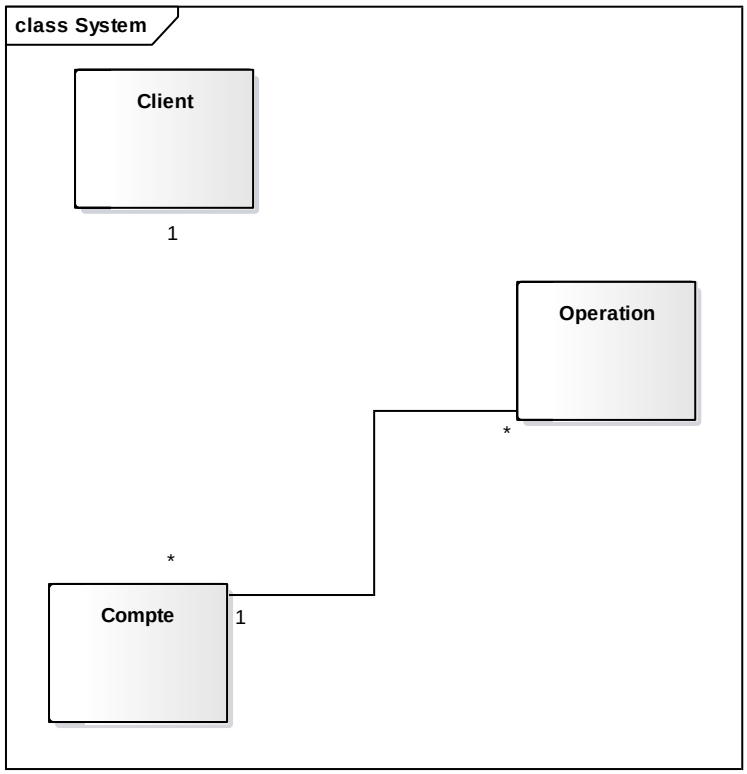


Diagramme de classes



Implémentation

Vous trouverez la partie code d'application dans le repo github :
https://github.com/ahmed081/examen-backend-banque-app

Les routes

```
/retrait
/comptetocompte
/versement
/getOpetration
/consulter-compte
/toggle-compte
```

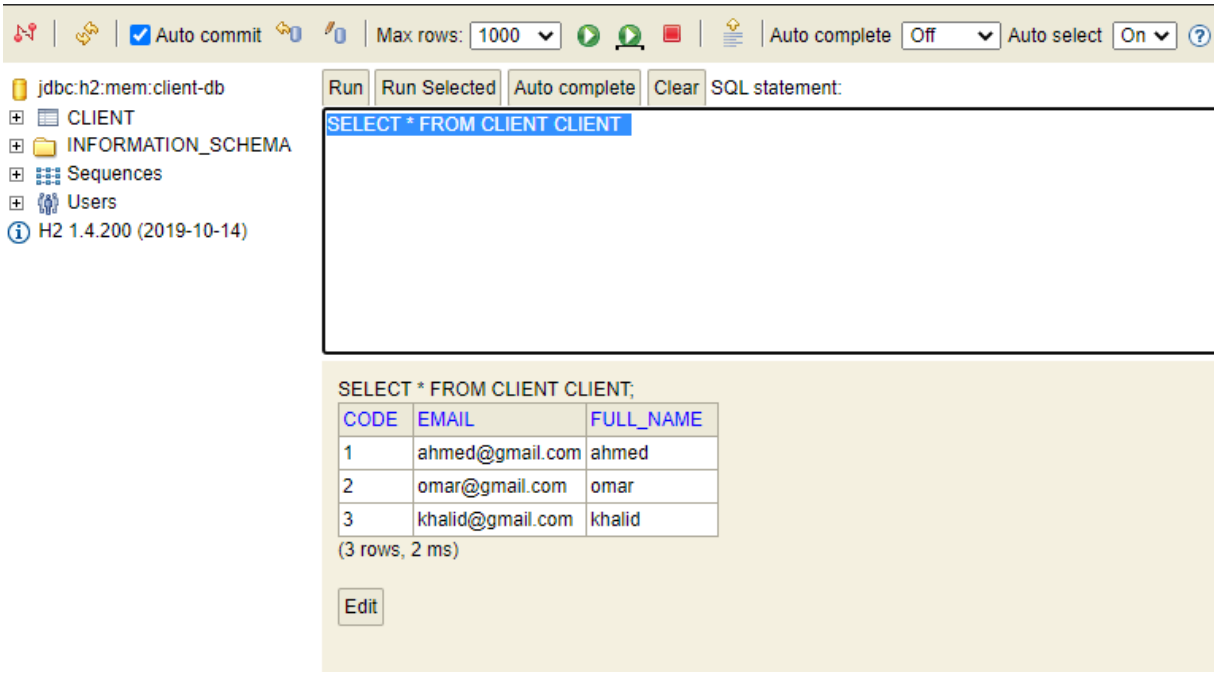
Réalisation

Micro service des clients

Database

```
CommandLineRunner start(ClientRepository clientRepository, RepositoryRestConfiguration restConfigration){
    return args->{
        restConfigration.exposeIdsFor(Client.class);
        clientRepository.save(new Client( code: null , fullName: "ahmed", email: "ahmed@gmail.com"));
        clientRepository.save(new Client( code: null , fullName: "omar", email: "omar@gmail.com"));
        clientRepository.save(new Client( code: null , fullName: "khalid", email: "khalid@gmail.com"));

        clientRepository.findAll().forEach(c->{
            System.out.println(c.getFullName());
        });
    };
```

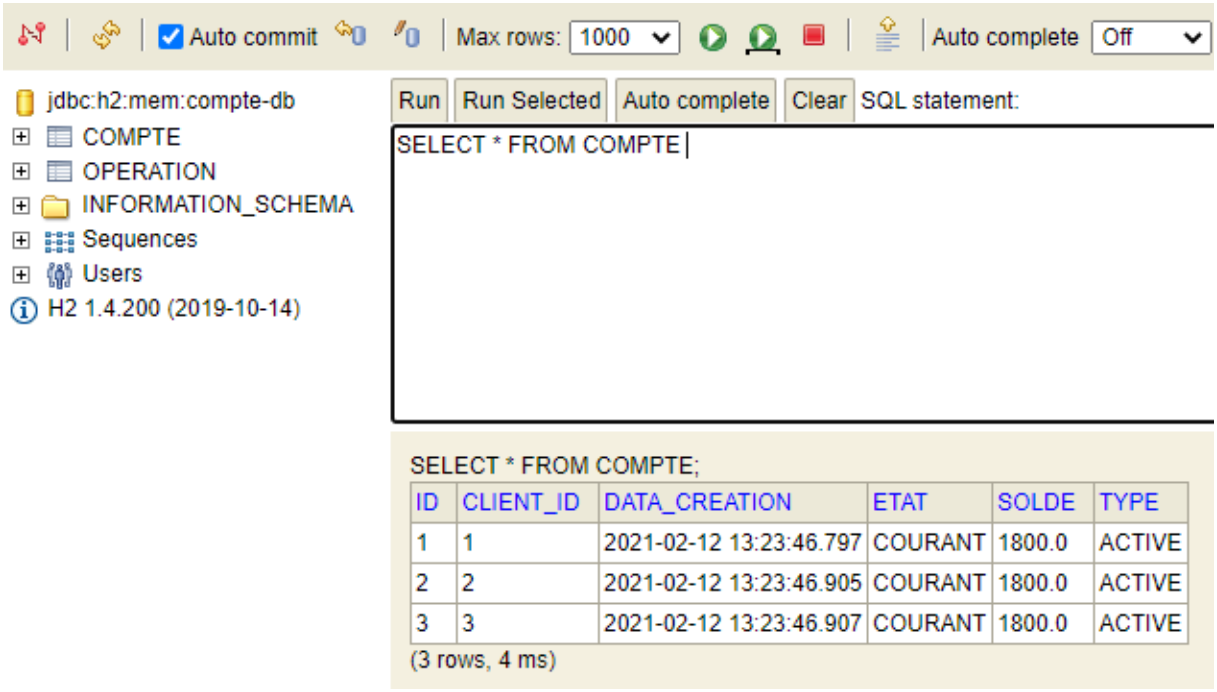

Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:client-db
CLIENT
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM CLIENT CLIENT
SELECT * FROM CLIENT CLIENT;
CODE EMAIL FULL_NAME
1 ahmed@gmail.com ahmed
2 omar@gmail.com omar
3 khalid@gmail.com khalid
(3 rows, 2 ms)
Edit

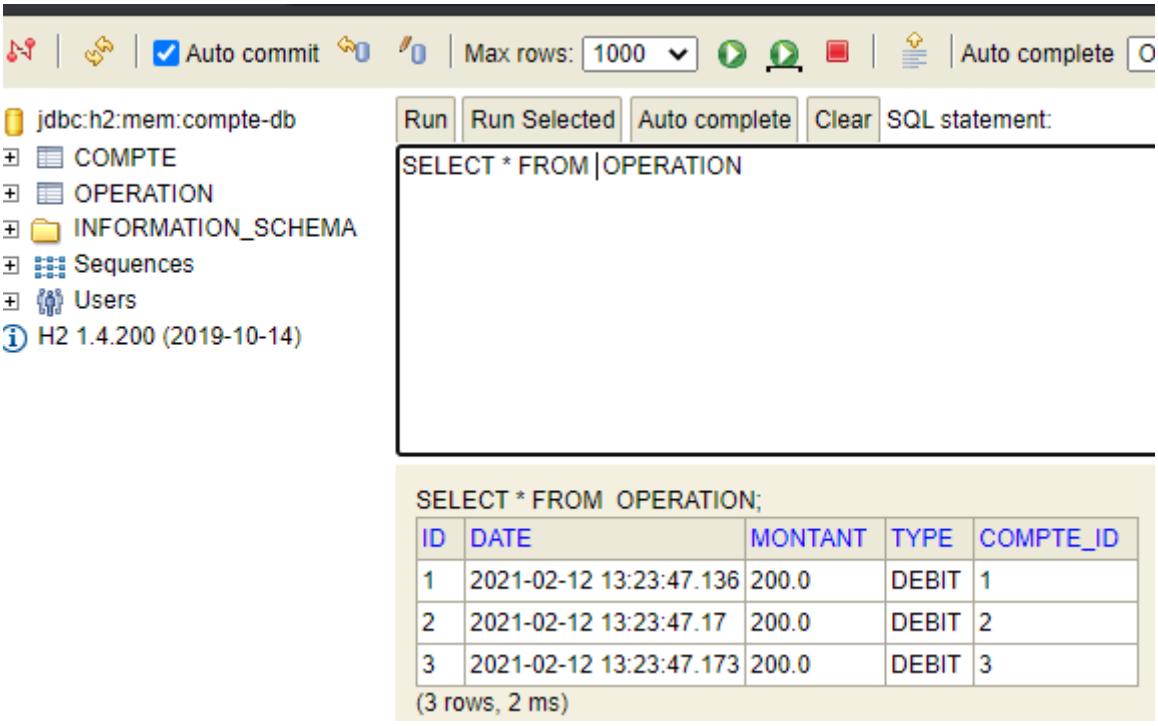
Restapi

Gateway

Micro service des comptes

Database






```
@Bean
CommandLineRunner start(
        CompteRepository compteRepository,
        OperationRepository operationRepository,
        ClientRestClient clientRestClient
){
    return args -> {

        clientRestClient.pageClient(0,20).forEach(client->{
            Compte compte = new Compte();
            compte.setSolde(2000.);
            compte.setEtat("COURANT");
```

```
            compte.setType("ACTIVE");
            compte.setData_creation(new Date());
            compte.setClientId(client.getCode());
            compteRepository.save(compte);
            System.out.println(client.getCode());
        });
        compteRepository.findAll().forEach(compte -> {
            Operation operation = new Operation();
            operation.setDate(new Date());
            operation.setType("DEBIT");
            operation.setCompte(compte);
            operation.setMontant(200.);
            compte.setSolde(compte.getSolde()-200);
            compteRepository.save(compte);
            operationRepository.save(operation);
            System.out.println(compte.getId());
        });
    };
}
```

Gateway

127.0.0.1:8080/COMPTE-SERVICE/consulter-compte?compte=1


```
{
    "id": 1,
    "solde": 1800,
    "data_creation": "2021-02-12T12:23:46.797+00:00",
    "type": "ACTIVE",
    "etat": "COURANT",
    "operations": [
        {
            "id": 1,
            "montant": 200,
            "date": "2021-02-12T12:23:47.136+00:00",
            "type": "DEBIT"
        }
    ],
    "clientId": 1,
    "client": {
        "code": 1,
        "fullName": null,
        "email": null
    }
}
```


http://127.0.0.1:8080/BILLING-SERVICE/add-compte

Save

POST http://127.0.0.1:8080/COMPTE-SERVICE/add-compte

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

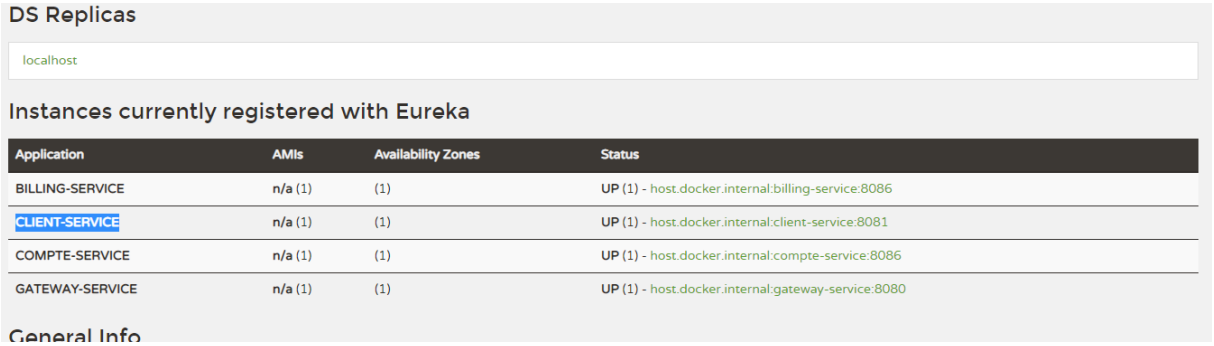

```
{
    "solde":2000,
    "type":"EPARGNE",
    "etat":"ACTIVE",
    "clientId":1
```


Body Cookies Headers (3) Test Results Status: 200 OK Time: 31 ms Size: 238 B Sa

Pretty Raw Preview Visualize JSON


```
{
    "id": 4,
    "solde": 2000.0,
    "data_creation": null,
    "type": "EPARGNE",
    "etat": "ACTIVE",
    "operations": null,
    "clientId": 1,
    "client": null
}
```

Eureka



Conclusion

Dans ce modeste travail j'ai pu mettre en pratique l'ensemble des techniques acquises dans le module de applications distribué avec Spring cloud mais malheureusement, je n'ai pas pu avoir arrivé à les objectifs de ce travail à faire, cela du a des problèmes technique au niveau de ma machine, et aussi l'apparition des erreurs au moment de développement ce qui ma arrive a consommer plus de temps de les faires résoudre. Ce rapport n'est que la première version de compte rendu, motivation de faire arriver à réussir tous les objectif de travail dans la 2eme et la dernier version.