Projet Micro Services
Naïm Bendjebbour & Pistis Ipanitt

Table des matières

I. Objectif du document

Dans ce document nous allons décrire l'ensemble des choix technique que nous avons fait et définir ce que nous avons appris durant ce projet.

Ce projet nous a permis de nous familiariser au développement d'application micro-services en Java avec le Framework Spring Boot. Nous n'avions jamais utilisé ces technologies et ce fût une bonne découverte.

II. Architecture

A. Diagramme de classe métier Service Compte

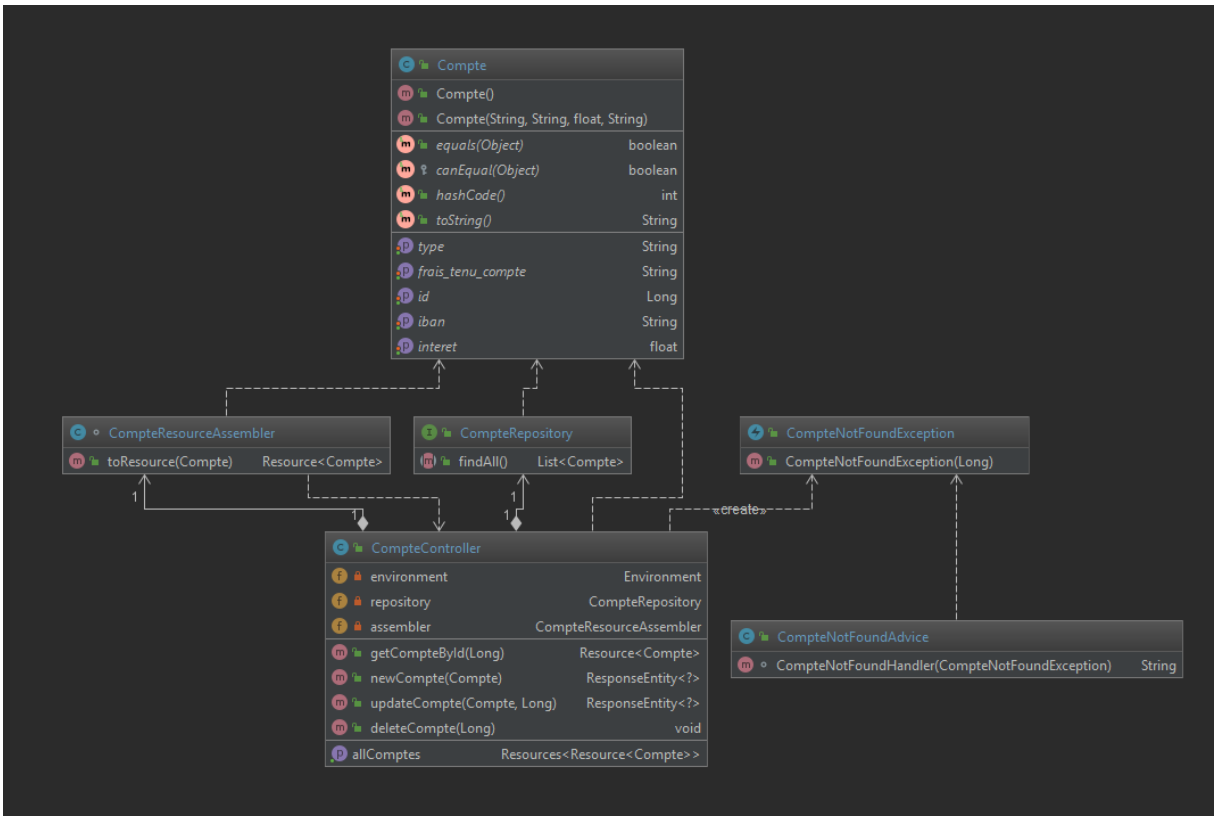

Figure 1 - Diagramme de classe micro service compte

B. Diagramme de classe métier Service Opération

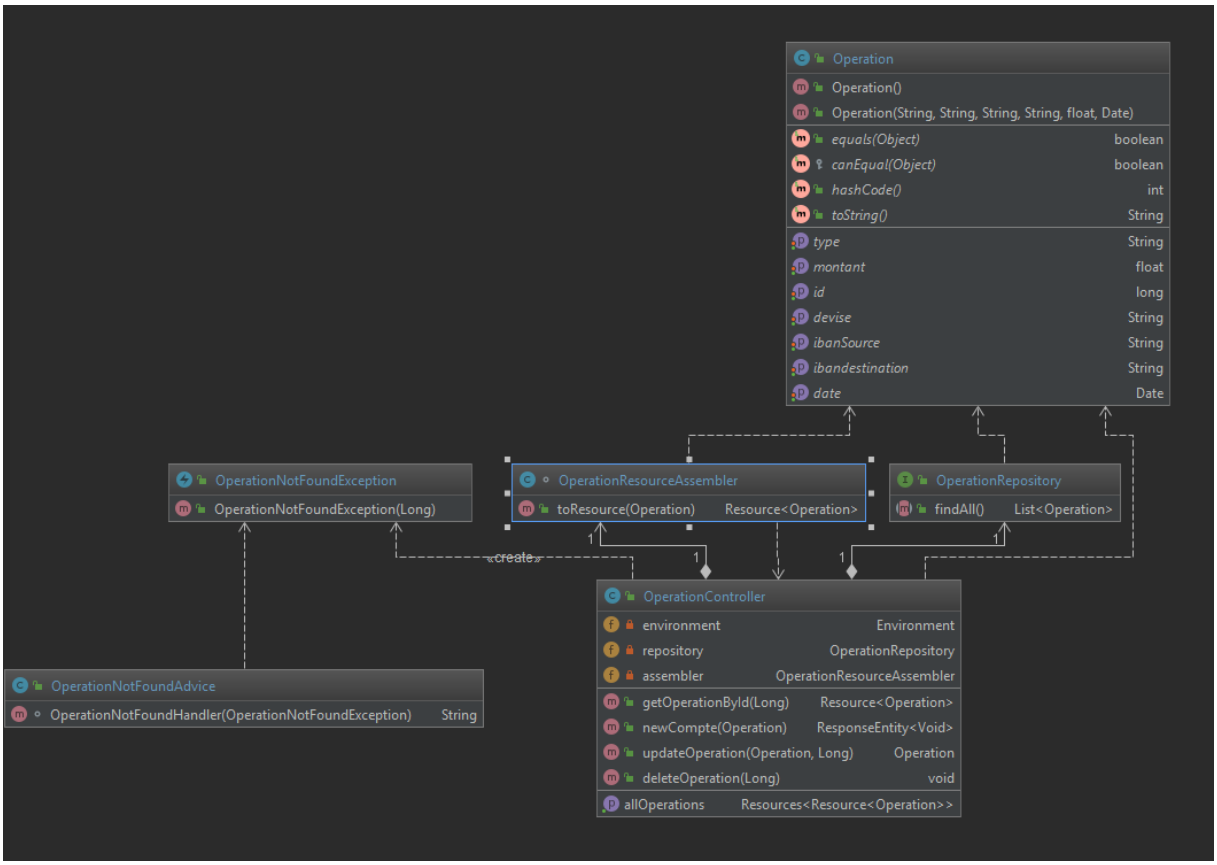

Figure 2- Diagramme de classe micro - service Operation

C. Schéma d'architecture

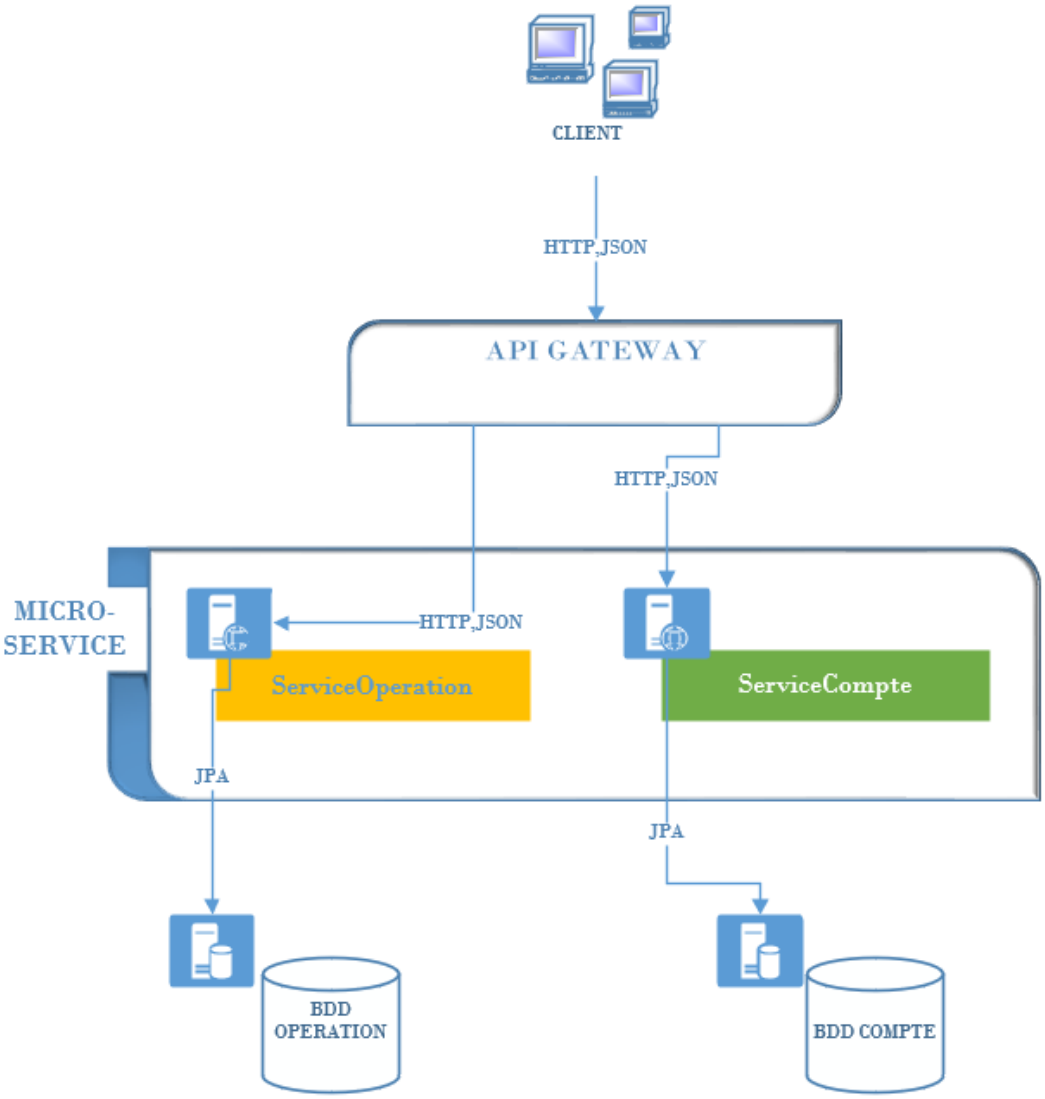

Figure 3 Schéma d'architecture générale

Notre projet se décomposent en trois parties :

Une application cliente qui consomme nos deux services RESTFULL.

Le service Operation deployer dans un conteneur docker ainsi que sa base de données.

Le service Compte déployer dans un conteneur docker ainsi que sa base de données.

L'api gateway permet un unique point d'entrer pour consommer nos micro-services.

III. Choix Technique

Durant ce projet nous avons mis l'accent sur les principes REST ainsi que sur les problématique DevOps.

Notre objectif est d'acquérir un maximum de connaissance dans ces domaines qui nous étaient étranger.

A. Problématique D'architecture.

Nous avons décidé que nous devions découper l'application en service complétement indépendant sans aucune dépendance. Cela pour tirer tous les avantages de l'architecture micro-service en termes de scalabilité et de disponibilité.

Nous nous sommes posé la question suivante : Comment les clients d'une application basée sur Micro-services accèdent-ils à des services individuels ?

- La granularité des API fournies par micro-services est souvent différente de celle dont un client a besoin. Les micro-services fournissent généralement des API fines, ce qui signifie que les clients doivent interagir avec plusieurs services. Par exemple, pour créer une opération dans le service Opération, nous devons d'abords vérifier que les IBAN existent dans le service compte et que le montant de l'opération correspond bien au solde du compte.

- Différents clients ont besoin de données différentes. Par exemple, la version du navigateur de bureau d'une page de détails sur le produit est généralement plus élaborée que la version mobile.
- Les performances du réseau sont différentes pour différents types de clients. Par exemple, un réseau mobile est généralement beaucoup plus lent et a une latence bien supérieure à celle d'un réseau non mobile.
- Le nombre d'instances de service et leurs emplacements (hôte + port) changent de manière dynamique.

B. Solution

Nous avons décidé d'implémentez une passerelle API qui constitue le point d'entrée unique pour notre client et dans une vision à long terme à tous les clients. La passerelle API traite les demandes de deux manières. Certaines demandes sont simplement proxy / acheminées vers le service approprié. Il traite les autres demandes en s'organisant vers plusieurs services et en effectuant des contrôles.

Cette solution est pour nous la meilleurs car elle permet de facilement faire évoluer nos applications.

Plutôt que de fournir une API de style unique, la passerelle d'API peut exposer une API différente pour chaque client. Par exemple, la passerelle API Netflix exécute un code d'adaptateur spécifique au client qui fournit à chaque client une API qui convient le mieux à ses exigences.

Nous avons choisi Spring MVC ainsi que Thymeleaf pour créer notre client.

C. Implémentation service RESTFULL

N'ayant jamais développer de service REST, nous sommes tombés sur le modèle de maturité de Leonard Richarson pour qualifier les api webservice.

Leonard Richarson classe les api web service selon 3 niveaux :

Le niveau 0 c'est le minimum acceptable dans la communication HTTP, toutes les requêtes sont envoyées à la même url, quelle que soit la demande, tous les requêtes sont envoyées avec le verbe POST …. Ce niveau est similaire à SOAP.

Dans le niveau 1 on introduit la notion de ressource, en fonction de la donnée métier manipulée.

Dans le niveau 2, on utilise les verbes et les codes numériques de retours du protocole http.

Le niveau 3 HATEOAS (*Hypermedia As The Engine Of Application State*). Ce niveau introduit un code de retour cohérent pour chaque verbe ainsi que des liens dans les ressources retournées via REST.

HATEOAS n'est pas un standard, mais nous avons vu que c'était une très bonne pratique de programmation dans le monde web. Cela permet pratiquement d'auto-documenter notre API

Nous avons donc utilisé Spring HATEOAS, pour produire nos services.

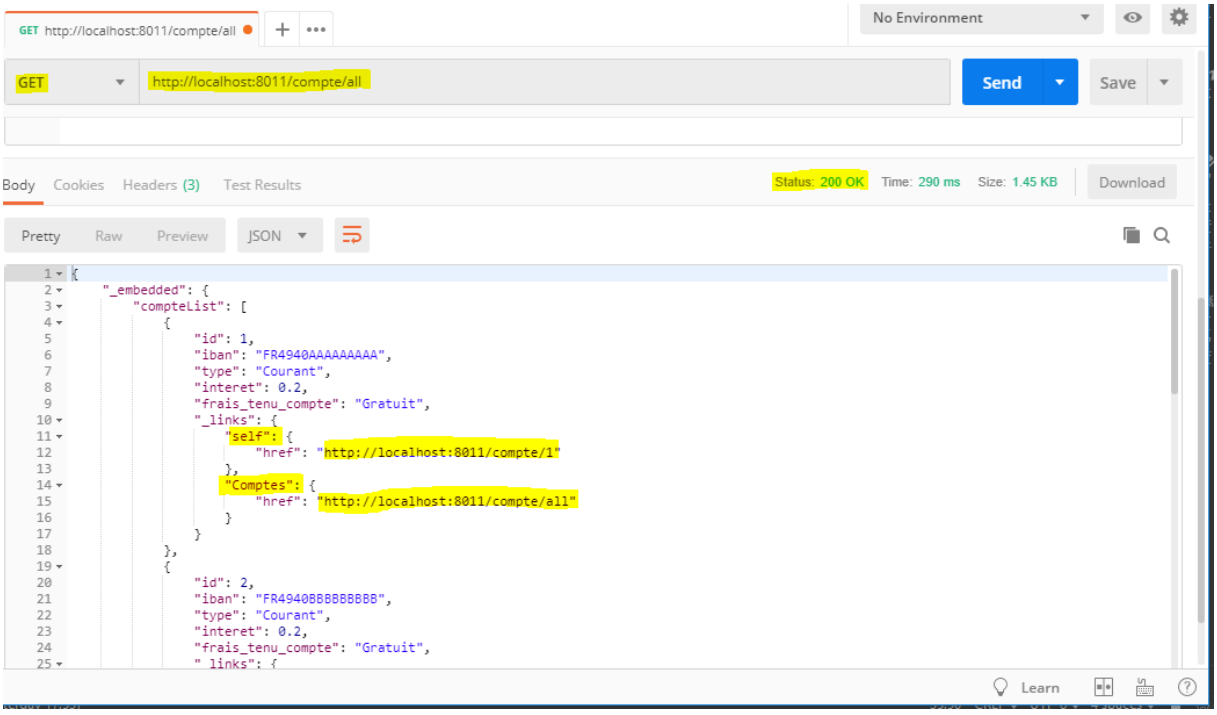

Figure 4- Ressource HATEOAS

D. TEST KarateDSL

Nous avons choisi d'effectuer nos tests à l'aide de KarateDSL, cette api permet d'exploiter la notion de feature de Cucumber et génère un rapport au format HTML.

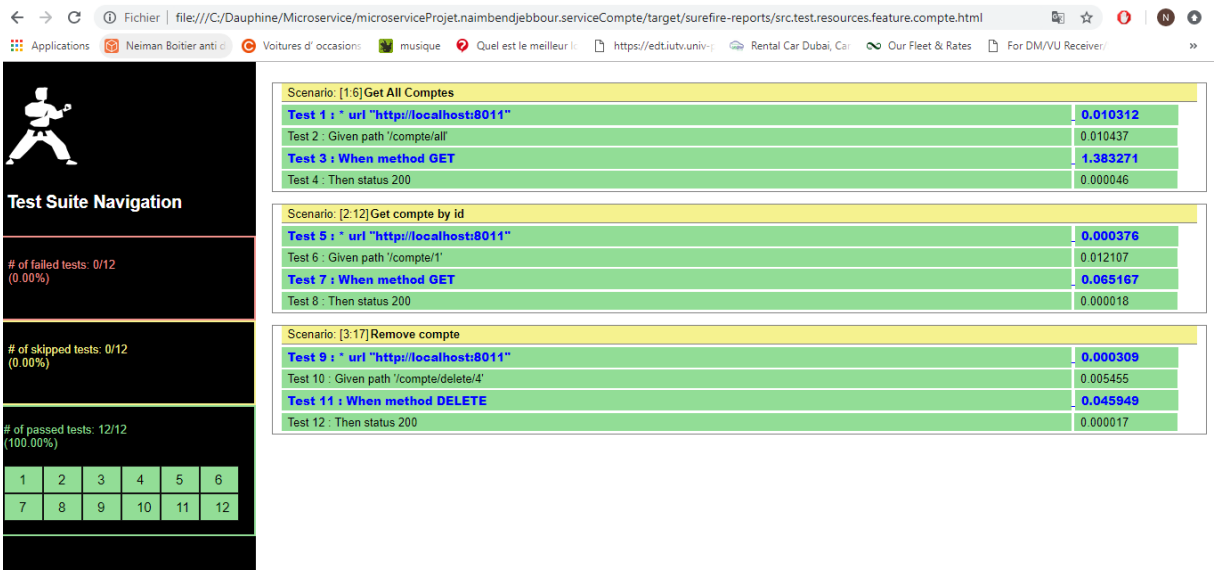

Figure 5- Rapport de tests KarateDSL

IV. Déploiement du projet à l'aide de docker

Récupération des sources :

```
git clone https://github.com/NaimBendjebbour/Projet_Microservice.git
```

Compilation : Pour chacun des services placez vous à la racine du projet et exécutez :

mvn clean install -DskipTests

Construction du conteneur : à la racine de chaque projet exécutez :

```
>docker build -f Dockerfile -t serviceoperation .
> docker build -f Dockerfile -t servicecompte .
```

Lancement du conteneur :

```
docker run -p 8010:8010 serviceoperation
docker run -p 8011:8011 servicecompte
```

Afin de vérifier que le container est bien lancé :

>docker info

```
Containers: 1
 Running: 1
 Paused: 0
 Stopped: 0
Images: 5
Server Version: 18.09.1
Storage Driver: overlay2
 Backing Filesystem: extfs
 Supports d_type: true
 Native Overlay Diff: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 9754871865f7fe2f4e74d43e2fc7ccd237edcbce
runc version: 96ec2177ae841256168fcf76954f7177af9446eb
init version: fec3683
Security Options:
 seccomp
  Profile: default
```

Capture lors du lancement :

```
Successfully built 7ddbb146b72a
Successfully tagged servicecompte:latest
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build context will have '-rwxr-xr-x
' permissions. It is recommended to double check and reset permissions for sensitive files and directories.

C:\Dauphine\Projet_MicroService\microserviceProjet.naimbendjebbour.serviceCompte>docker run -p 8001:8001 servicecompte

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.1.1.RELEASE)

2019-02-09 02:41:15.559 DEBUG 1 --- [           main] o.s.web.context.ContextLoader            : Published root WebApplicationContext as ServletContext attribute with n
ame [org.springframework.web.context.WebApplicationContext.ROOT]
2019-02-09 02:41:15.564  INFO 1 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 5943 ms
Hibernate: drop table compte if exists
Hibernate: drop sequence if exists hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate: create table compte (id bigint not null, frais_tenu_compte varchar(255), iban varchar(255), interet float not null, type varchar(255), primary key (id))
2019-02-09 02:41:19.365 DEBUG 1 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Patterns [/**/favicon.ico] in 'faviconHandlerMapping'
2019-02-09 02:41:19.758 DEBUG 1 --- [           main] s.w.s.m.m.a.RequestMappingHandlerAdapter : ControllerAdvice beans: 0 @ModelAttribute, 0 @InitBinder, 1 RequestBody
Advice, 1 ResponseBodyAdvice
2019-02-09 02:41:20.068  WARN 1 --- [           main] aWebConfiguration$JpaWebMvcConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database quer
ies may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2019-02-09 02:41:20.134 DEBUG 1 --- [           main] s.w.s.m.m.a.RequestMappingHandlerMapping : 7 mappings in 'requestMappingHandlerMapping'
2019-02-09 02:41:20.160 DEBUG 1 --- [           main] o.s.w.s.handler.SimpleUrlHandlerMapping  : Patterns [/webjars/**, /**] in 'resourceHandlerMapping'
2019-02-09 02:41:20.186 DEBUG 1 --- [           main] .m.m.a.ExceptionHandlerExceptionResolver : ControllerAdvice beans: 1 @ExceptionHandler, 1 ResponseBodyAdvice
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
Hibernate: call next value for hibernate_sequence
Hibernate: insert into compte (frais_tenu_compte, iban, interet, type, id) values (?, ?, ?, ?, ?)
```

V. Mapping Service REST

A. Service Compte

METHODE	URL	DESCRIPTION
GET	/compte/{id}	Récupérer un compte
GET	/compte/all	Récupérer liste des compte
POST	/compte/add	Ajouter un compte
Put	/compte/update/{id}	Mise à jour du compte
Delete	/compte/delete/{id}	Supprimer un compte

B. Service Opération

METHODE	URL	DESCRIPTION
GET	/operation/{id}	Récupérer une operation
GET	/operation/all	Récupérer liste des operations
POST	/operation/add	Ajouter une operation
Put	/operation/update/{id}	Mise à jour d'une operation
Delete	/operation/delete/{id}	Supprimer une operation

VI. Bilan.

Ce projet fut enrichissant, nous ne connaissions absolument rien des technologies web. Le projet nous a permis de découvrir et prendre en main REST.

Nous avons eu des difficultés a conteneurisé nos services à l'aide de docker. En effet HyperV était en conflit avec docker lorsque nous essayons d'attribuer une IP statique à nos containers. La documentation nous a permis de passer cette étape.

La grippe 2019 ne nous a pas permis de terminer notre client, nous avons testé nos apis à l'aide de Postman ainsi que nos tests à l'aide de KarateDSL.

Nous allons quand même terminer ce client afin d'acquérir toutes les connaissances nécessaires et espérons vous le présenter à la soutenance.

En somme ce projet nous a permis de faire une veille technologique global autour des technologies du web et du monde java. Nous avons aimé la simplicité de Spring HATEOAS ainsi que Thymeleaf.

Nous avons été frappés sévèrement par la grippe ce qui nous a mis dans une situation délicate ou ne devions gérer plusieurs projets à la fois.

Cela nous a permis de nous confronter au aléa de la vie d'un projet et de nous adapter.