Architecture Microservice
Projet Spring Boot

Binôme:
- Sagodira Yoan
- Sagodira Sébastien

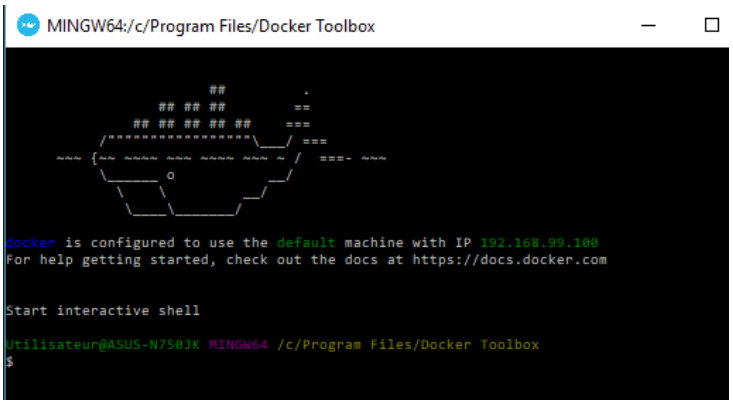
Compilation/Exécution du projet avec Docker

Etape 1 : Récupérer les deux solutions du projet à l'adresse Github :

- 1ère solution BankAccount : TP_SagodiraYoan_SagodiraSebastien
- 2ème solution BankingOperation : Tp_BankingOperation_SagodiraYoan_SagodiraSebastien

Etape 2 : Création d'adresse IP

- Exécuter le Docker Quickstart terminal pour générer un adresse IP :



- Dans votre IDE, modifiez si besoin dans chacun des controllers des deux solutions, l'adresse IP lors des appels REST par l'adresse IP nouvellement générée. Cela est nécessaire uniquement si celle-ci est différente de :
    - 192.168.99.100

Etape 3 : Build et création des .jar

- Ouvrir l'invité de commande (cmd) et se positionner à la racine d'une des 2 Solutions
- Taper la commande : mvn clean install spring-boot:run
    - Cela permet de créer un .jar et de run la solution. On peut ensuite la stopper (Ctrl+C)
- Réitérer l'opération pour la seconde solution

Etape 4 : Création des images docker

- Dans cmd, se positionner à la racine de la solutions 1 (BankAccount)
    - Taper la commande : docker build -f Dockerfile -t bankaccount .
    - Attention à bien mettre le . à la fin de la commande
- Se positionner à la racine de la solution 2 (BankingOperation)
    - Taper la commande : docker build -f Dockerfile -t bankingoperation .
    - Attention à bien mettre le . à la fin de la commande

Cela créé les images docker. Pour vérifier qu'elles ont été créées, taper la commande : docker images

```
D:\yoan\Dauphine\Architecture Micro Service\TP_SagodiraYoan_SagodiraSebastien>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
bankaccount         latest              491ff2be2c6c        37 seconds ago      661MB
bankingoperation    latest              432ab45b35cd        2 minutes ago       661MB
openjdk             8                   5f4603da3fbc        10 days ago         624MB
```

Etape 5 : Container et run

- Dans cmd taper les commandes :
  - docker run -p 8000:8000 bankaccount
  - docker run -p 8001:8001 bankingoperation

Les deux Micro services sont maintenant disponibles aux URI :

- http://192.168.99.100:8000/
- http://192.168.99.100:8001/

Documentation Technique

Choix Techniques

- Langage Java (imposé)
- Framework Spring Boot (imposé)
- Build avec Maven
- Spring Initializr
- Base de données embarquée H2
- Conteneurisation avec Docker
- Architecture Microservices
- Service web REST (Basé sur le Protocol HTTP)

Schéma Architecture

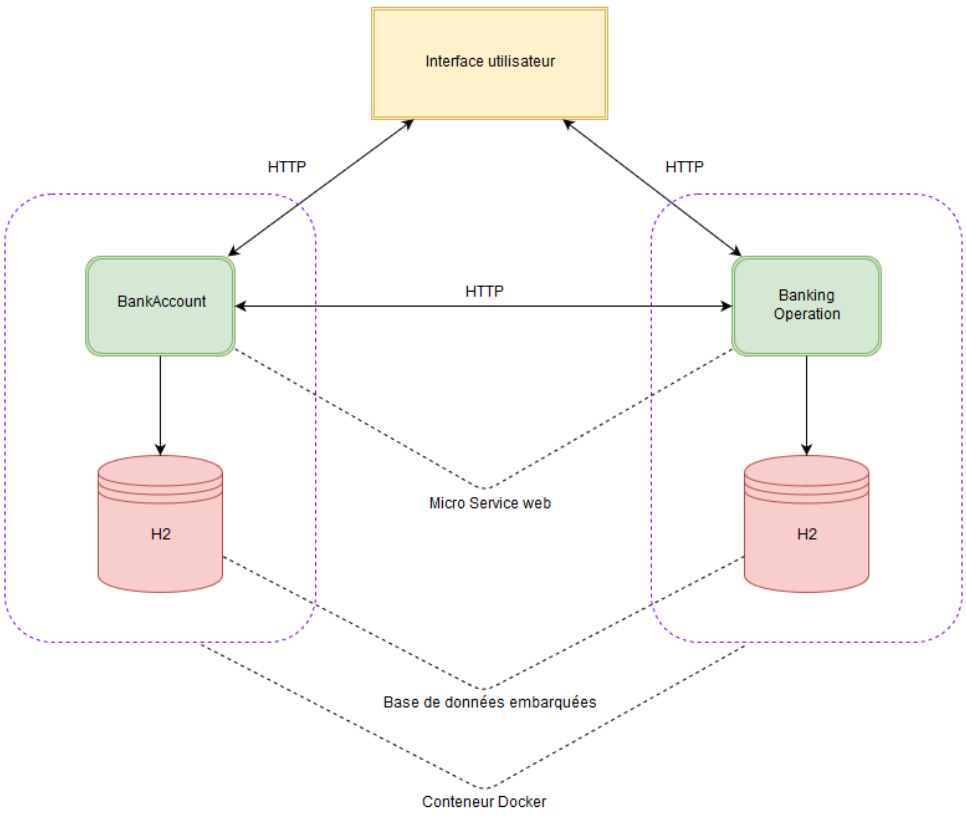
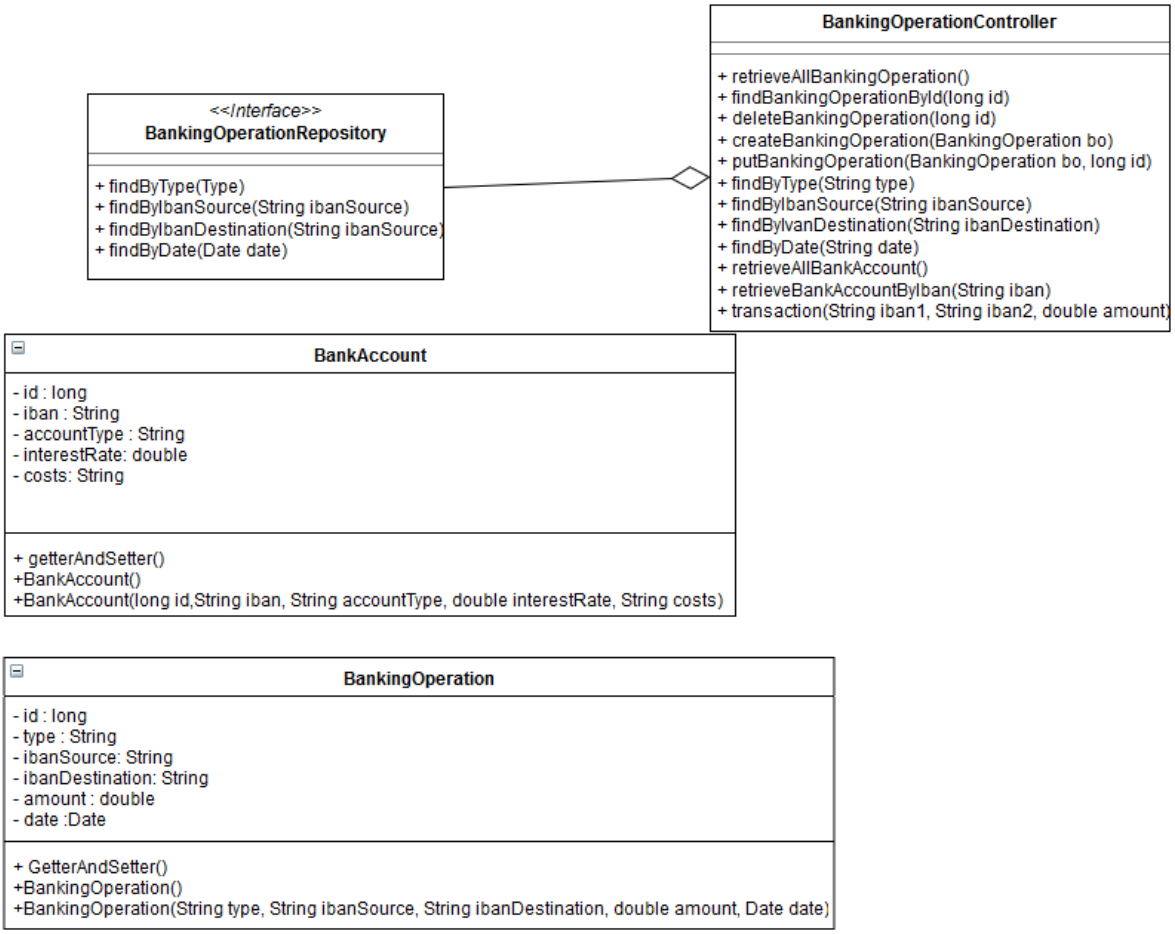


Diagramme de classes

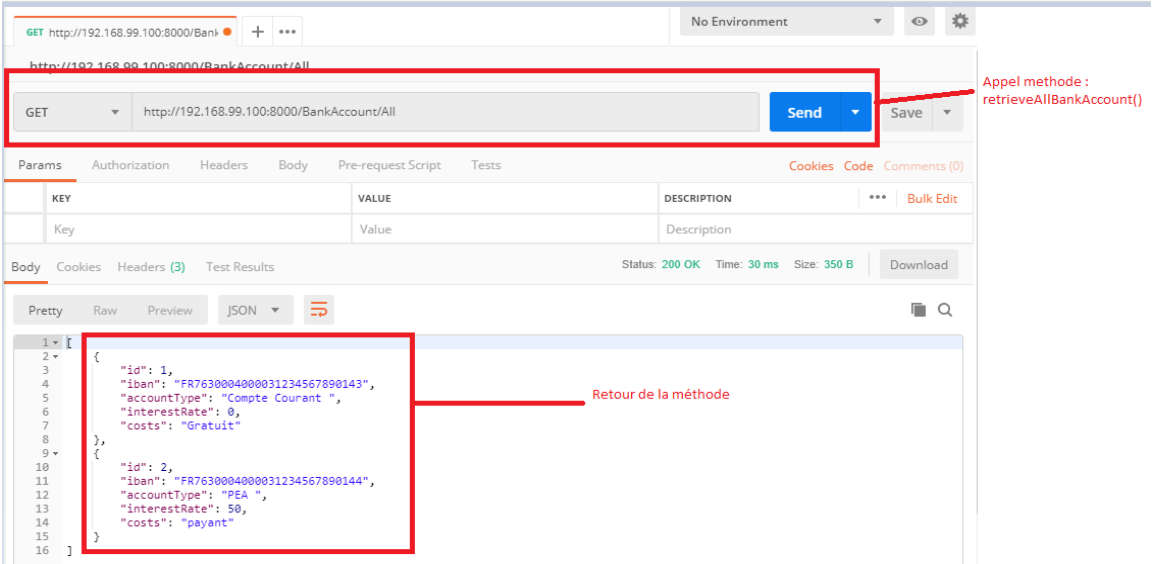


Tests du projet avec Postman

- Download Postman : https://www.getpostman.com/downloads/
- Nous vous présenterons 3 cas généraux afin de conceptualiser la communication entre Micro Services

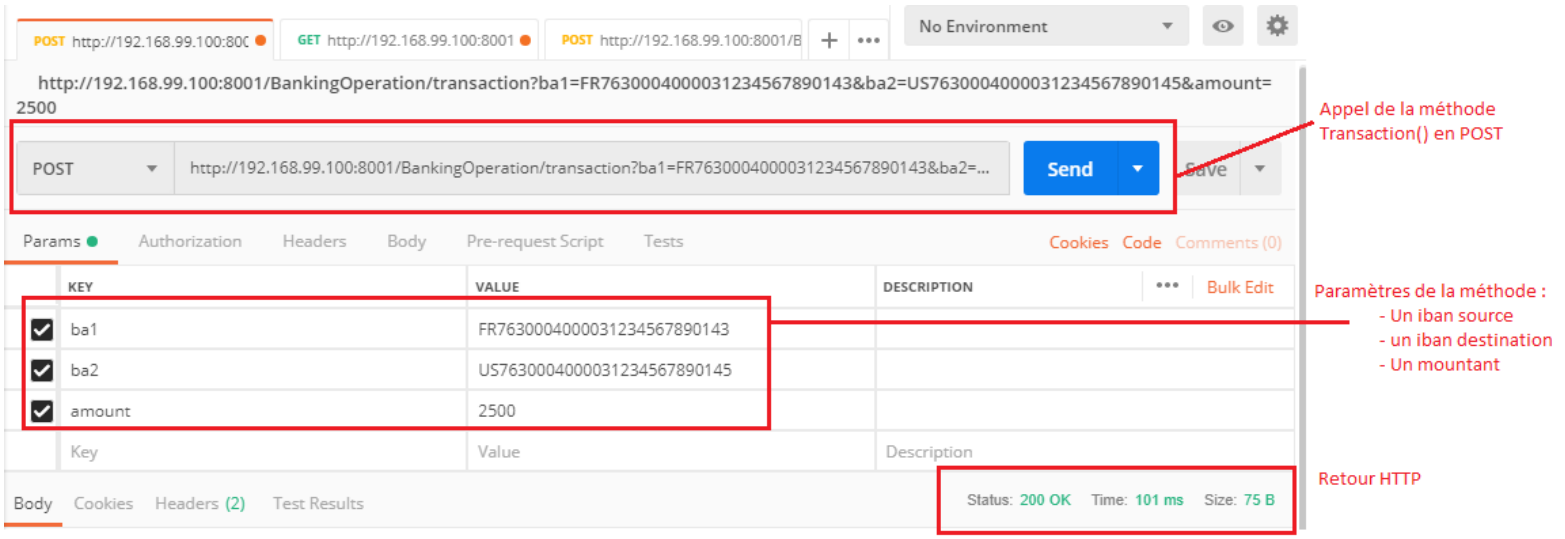
Microservice BankAccount (Solution 1)

- Test de la méthode retrieveAllBankAccount méthode avec la méthode GET de HTTP



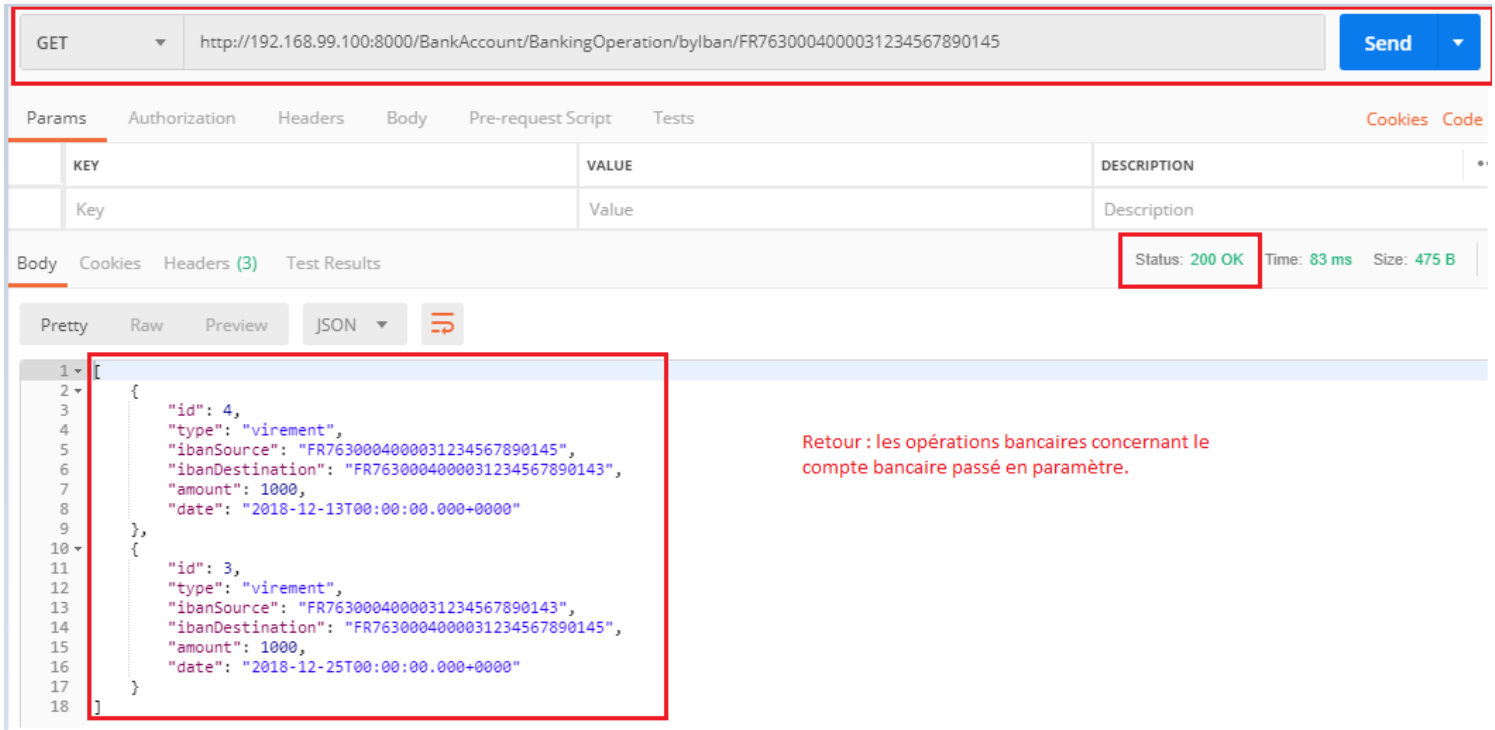
Microservice BankingOperation (Solution 2)

- Test de la méthode virement avec la méthode POST de http
- Cette méthode ajoute 2 transactions en base de données (pour chaque sens de la transaction)



Communication entre les deux Micro Services

- Test de la méthode retrieveAllBankingOperationByIban du Micro service de BankAccount (Solution 1).
- Celle-ci va récupérer les opérations bancaires impliquant le compte passé en paramètre avec les méthodes du Micro Service BankingOperation (Solution 2) en utilisant REST :
  - findByIbanSource(String iban)
  - findByIbanDestination(String iban)



Voici le code pour l'appel REST pour plus de compréhension :

```java
@RequestMapping(value="/BankingOperation/byIban/{iban}",method = {RequestMethod.GET})
public List<BankingOperation> retrieveAllBankingOperationByIban(@PathVariable String iban) {


    List<BankingOperation> bankingOperations = new ArrayList<BankingOperation>();

    RestTemplate restTemplate = new RestTemplate();
    ResponseEntity<List> responseEntity = restTemplate.getForEntity( url: "http://192.168.99.100:8001/BankingOperation/byIbanSource/" + iban,List.class);
    ResponseEntity<List> responseEntity2 = restTemplate.getForEntity( url: "http://192.168.99.100:8001/BankingOperation/byIbanDestination/" + iban,List.class
    bankingOperations = responseEntity.getBody();
    bankingOperations.addAll(responseEntity2.getBody());
```

Bilan du projet

Ce que l'on a aimé/appris (Dans l'ordre de préférence):

- Découverte des bases de données embarquées
- Découverte du framework Spring Boot (vraiment très utiles car nous n'avions jamais utilisé de framework pour du Java jusqu'à maintenant)
- Découverte de Docker
- Utilisation de REST pour communiquer entre différents services

Ce que l'on a moins aimé :

- Pas assez de cours pratiques (TP avec l'aide du prof)
- Mauvaise répartition des modules de Projets. En effet tous les projets informatiques sont à rendre la même semaine (avec pour chacun une soutenance) ce qui fait un délai très court pour chacun des projets. Pourquoi ne pas envisager de mettre ce cours en début d'année pour les prochains étudiants ?

Réussites :

- Utilisation de Spring Boot
- Architecture divisée en 2 micro Services.
- Utilisation d'API REST
- Compréhension de JPA
- Réalisation des besoins utilisateurs pour le projet :
    - Gérer des comptes bancaires (CRUD)
    - Gérer des opérations bancaires
    - Faire communiquer les deux solutions sous forme de Micro service
- Création de Container avec Docker

Echec/difficultés :

- Pas d'interface utilisateur : nous souhaitions tester nos services à travers des tests JUnit. Cependant, afin de redistribuer notre temps dans d'autres projets. Nous avons finalement opté pour une solution qui nous paraissait plus simple : l'utilisation de l'outil Postman pour tester nos appels à nos services.
- L'implémentation de REST fut très compliquée pour nous au départ.