

Architecture microservices

# Projet - currency

---

Elie ABI HANNA DAHER

Bilal EL CHAMI

3 juillet 2018

Lien github : <https://github.com/elieahd/architecture-microservices>

## Introduction

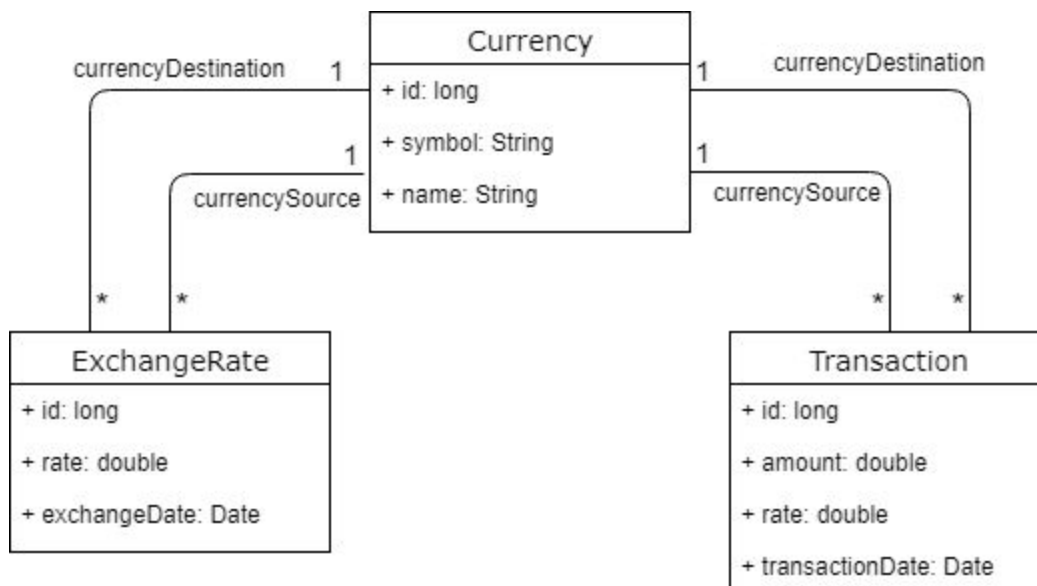
Objectif du projet : concevoir une application permettant d'échanger de l'argent, tout en gérant les taux de change entre deux devises.

Travail effectué :

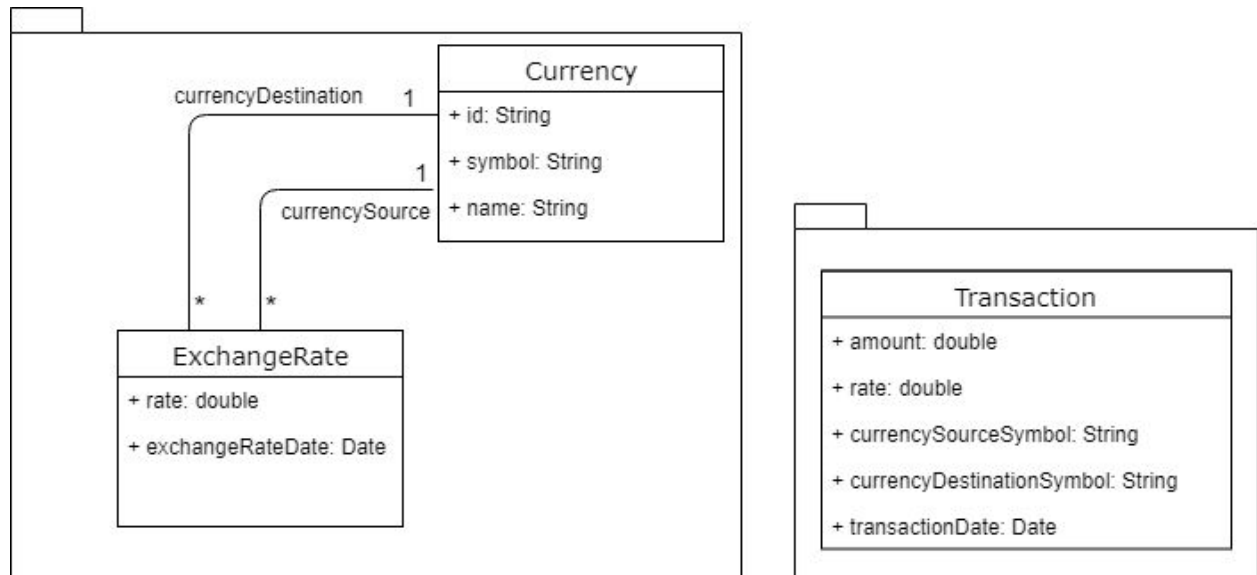
- découpage de l'application sous forme de microservices
- développement d'un backend reposant sur le Framework Spring Boot
- développement d'une application cliente web reposante sur Angular 6 et Bootstrap 4
- conteneurisation des applications à l'aide de Docker

## Diagrammes de classes - métier

Avant le découpage de l'application en microservice

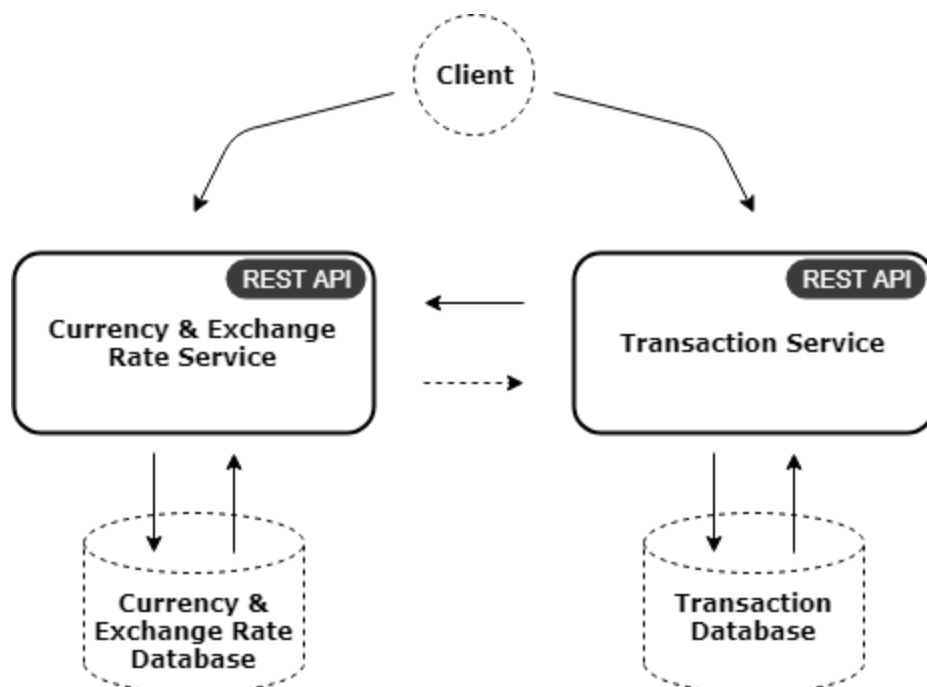


## Après le découpage de l'application en microservice



## Architecture microservices

La solution “currency-exchange” a été décomposé en deux microservices : *currency-exchange\_currency* et *currency-exchange\_transaction*. Ce sont des applications déployables indépendamment, organisées autour de leurs propres domaines d'activité.



## Contraintes d'intégrités

Après la décomposition de l'application en deux microservices ayant chacun sa propre base de données, nous avons perdu les contraintes d'intégrités "clés étrangères" entre les tables "Transaction" et "Currency". Par conséquent, nous avons ajouté les colonnes "CURRENCY\_SOURCE\_SYMBOL" et "CURRENCY\_DESTINATION\_SYMBOL" pour établir des dépendances fonctionnelles.

Nous avons laissé les classes "Currency" et "ExchangeRate" dans le service "currency-exchange\_transaction" qui représentent les DTOs.

NB : Ces classes ne sont pas des entités dans ce microservice, donc il n'existe pas des tables correspondantes dans la base de données.

## Définition des API REST

### Transaction service

Method	Path	Description	Available from UI
GET	/transactions	Récupérer toutes les transactions	×
GET	/transactions/{transactionId}	Récupérer une transaction par identifiant	
POST	/transactions	Ajouter ou modifier une transaction	
DELETE	/transactions/{transactionId}	Supprimer une transaction par identifiant	
POST	/transactions/exchange/from/{from}/to/{to}/amount/{amount}	Faire une transaction entre deux devises	×

## Currency & Exchange Rate service

### Currency

Method	Path	Description	Available from UI
GET	/currency	Récupérer toutes les devises	×
GET	/currency/{currencyId}	Récupérer une devise par identifiant	×
POST	/currency	Ajouter ou modifier une devises	×
DELETE	/currency/{currencyId}	Supprimer une devise par identifiant	×

### Exchange Rate

Method	Path	Description	Available from UI
GET	/exchange-rate	Récupérer tous les taux de change	×
GET	/exchange-rate/{exchangeRateId}	Récupérer un taux spécifique par identifiant	
POST	/exchange-rate	Ajouter ou modifier un taux de change	×
DELETE	/exchange-rate/{exchangeRateId}	Supprimer un taux de change	×
GET	/exchange-rate/latest/from/{from}/to/{to}	Trouver le dernier taux entre deux devises	×
GET	/exchange-rate/from/{from}/to/{to}	Trouver tous les taux entre deux devises	×

## Commandes d'exécution

Les applications peuvent être déployées localement en lançant les fichiers jar avec la commande “java” ou bien déployés dans des conteneurs docker. Voici les différentes commandes d'exécution pour chaque plateforme.

Par défaut, l'application “currency-exchange\_transaction” écoute sur le port 8080 et “currency-exchange\_currency” sur 8000.

### Local

```
$ mvn clean
$ mvn install
$ cd target
# L'application currency-exchange_transaction
$ java -jar currency-exchange_transaction.jar --currency_app_hostname=localhost
# L'application currency-exchange_currency
$ java -jar currency-exchange_currency.jar
```

### Docker

#### L'application “Currency”

```
# Build a docker image
docker build -f Dockerfile -t currency-exchange_currency .
# Check if the image is created
docker images
# Run image
docker run -p 8000:8000 --name AppCurrency currency-exchange_currency
```

#### L'application “Transaction”

```
# Build a docker image
docker build -f Dockerfile -t currency-exchange_transaction .
# Check if the image is created
docker images
# Run image and link it to the currency app
# to enable http communication between the two containers
docker run -p 8080:8080 --link AppCurrency --name AppTransaction
currency-exchange_transaction
```

## Frontend

Une application frontend a été mise en place en utilisant Angular 6, Bootstrap 4 et Chart.js.

Le frontend *my-currency-app* contient les fonctionnalités suivantes :

- Effectuer une transaction
- Gérer les devises
- Gérer les taux de changes
- Visualiser la liste des transactions effectuées

Voici quelques captures écrans de cette application :

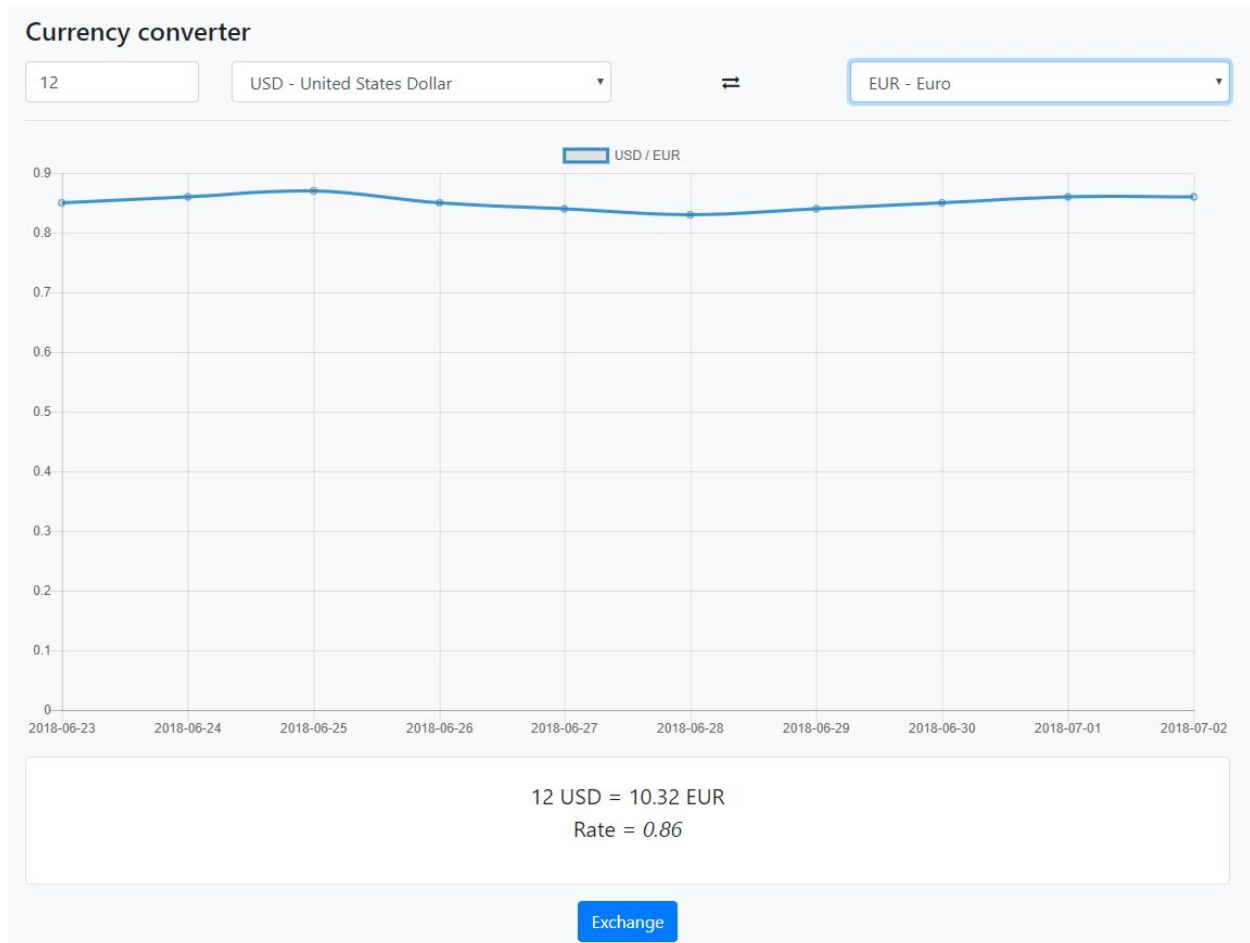
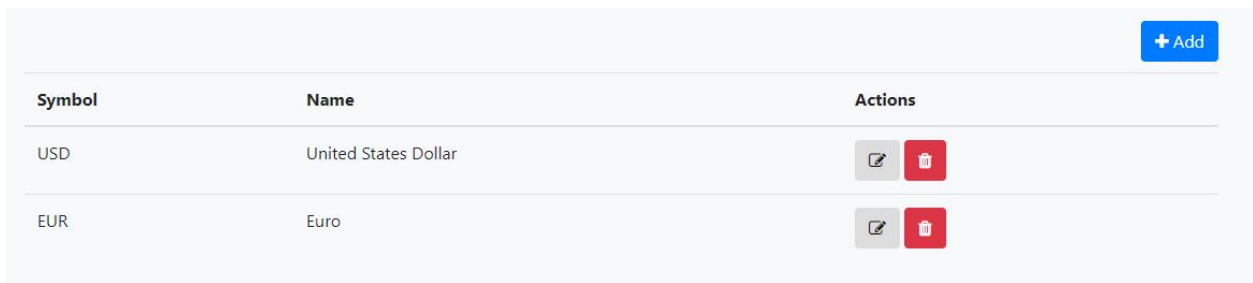


Figure 1 : Capture d'écran de la page web pour réaliser une transaction







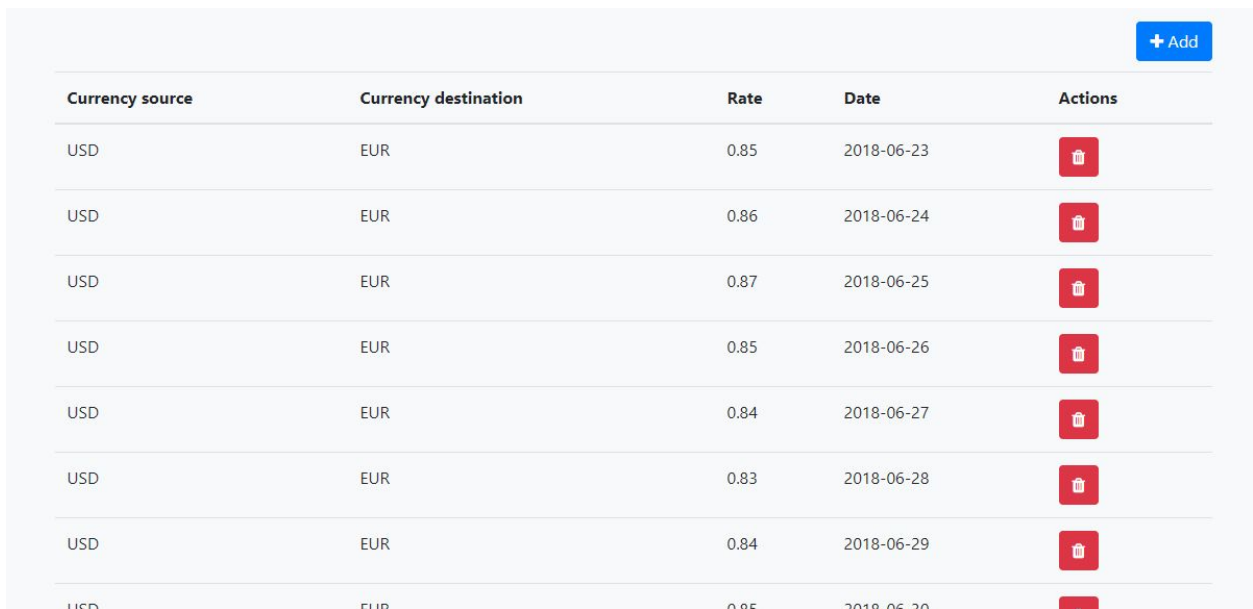
Symbol	Name	Actions
USD	United States Dollar	 
EUR	Euro	 

Figure 2 : Capture d'écran de la page web permettant de gérer les devises











Currency source	Currency destination	Rate	Date	Actions
USD	EUR	0.85	2018-06-23	
USD	EUR	0.86	2018-06-24	
USD	EUR	0.87	2018-06-25	
USD	EUR	0.85	2018-06-26	
USD	EUR	0.84	2018-06-27	
USD	EUR	0.83	2018-06-28	
USD	EUR	0.84	2018-06-29	
USD	EUR	0.85	2018-06-30	

Figure 3: Capture d'écran de la page web permettant de gérer les taux de changes



Currency source	Currency destination	Amount	Rate	Date
EUR	USD	10	1.17	2018-04-03
EUR	USD	20	1.15	2018-05-03
EUR	USD	20	1.2	2018-06-03
USD	EUR	12	1.16	2018-07-02

Figure 4 : Capture d'écran de la page web permettant de visualiser la liste des transactions





## Bilan du projet

Ce projet nous a permis de concevoir le concept SOA, et d'implémenter une application sous forme de microservices en utilisant le Framework Spring Boot, le gestionnaire de code GitHub et Docker.

En plus le choix de la technologie de l'application "frontend" nous a permis de pratiquer et de renforcer notre connaissance dans le framework Angular.

Nous avons rencontré des difficultés en installant Docker sur un de nos pc, donc nous avons dû travailler sur une seule machine concernant cette partie.

Une autre difficulté rencontrée était d'établir une communication entre les 2 conteneurs docker.

Nous aurions préféré voir les notions du modèle de maturité de Richardson avant la dernière séance afin d'implémenter les codes retours HTTP dans notre solution pour atteindre le niveau 2.

