Projet Architecture Micro-Services

Présentation

Pour ce projet il a fallu créer une application multiservice. Pour le développement j'ai utilisé l'outil IntelliJ.

Cet outil a permis de créer les classes .java et les fichiers de configuration comme les .xml, les .sql et les properties tout en profitant de d'une collaboration avec les autres outils mise à disposition par IntelliJ.

La compilation a donc été effectuée grâce à Maven, communiquant avec IntelliJ et ses codes sources.

La base e données a été gérée par H2 mais j'ai rencontré des difficultés que je détaillerai plus tard dans ce document.

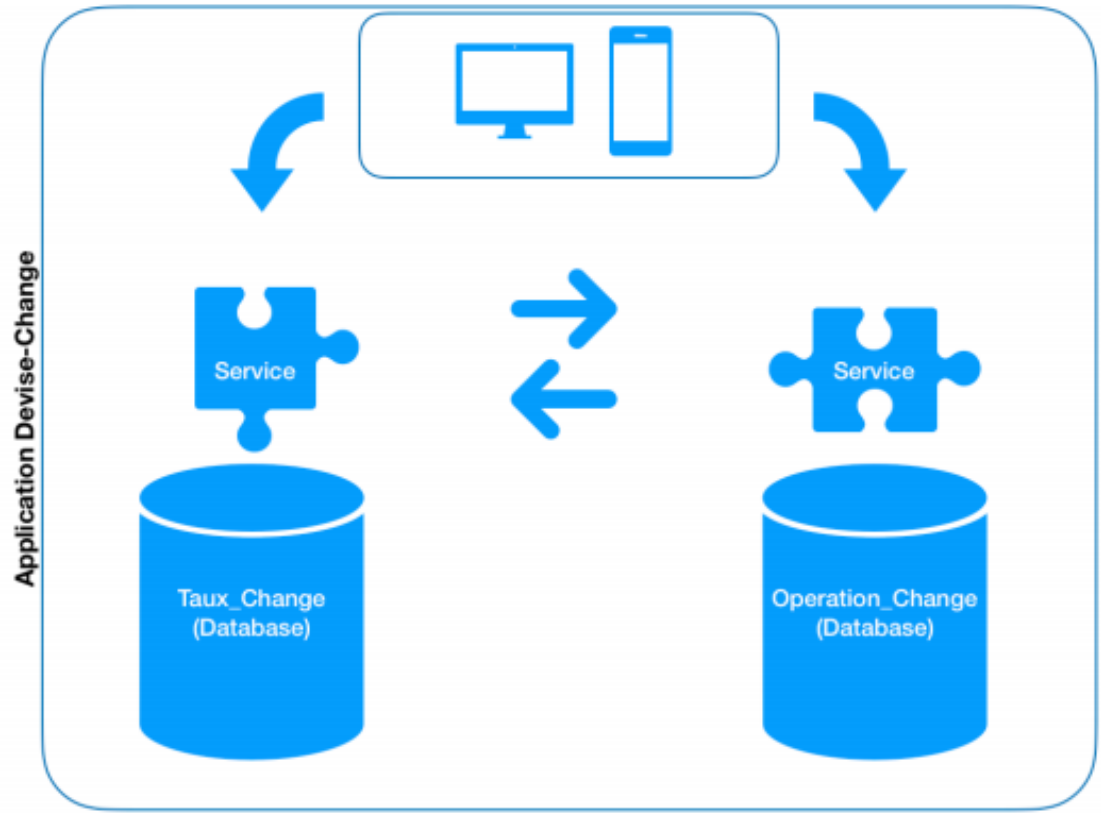
Le coeur du projet a été réalisé grâce à springboot, permettant de définir les APIs REST et le JPA de persistence.

Pour le gestion des requêtes HTTP j'ai utilisé PostMan.

Je n'ai pas géré de docker étant donné que par mail vous m'avez dit de ne pas m'en occuper.

L'ensemble du projet est stocké sur gitHub à l'adresse : https://github.com/Antoine2312/ProjetArchiMicroServices.

Schéma d’Architecture



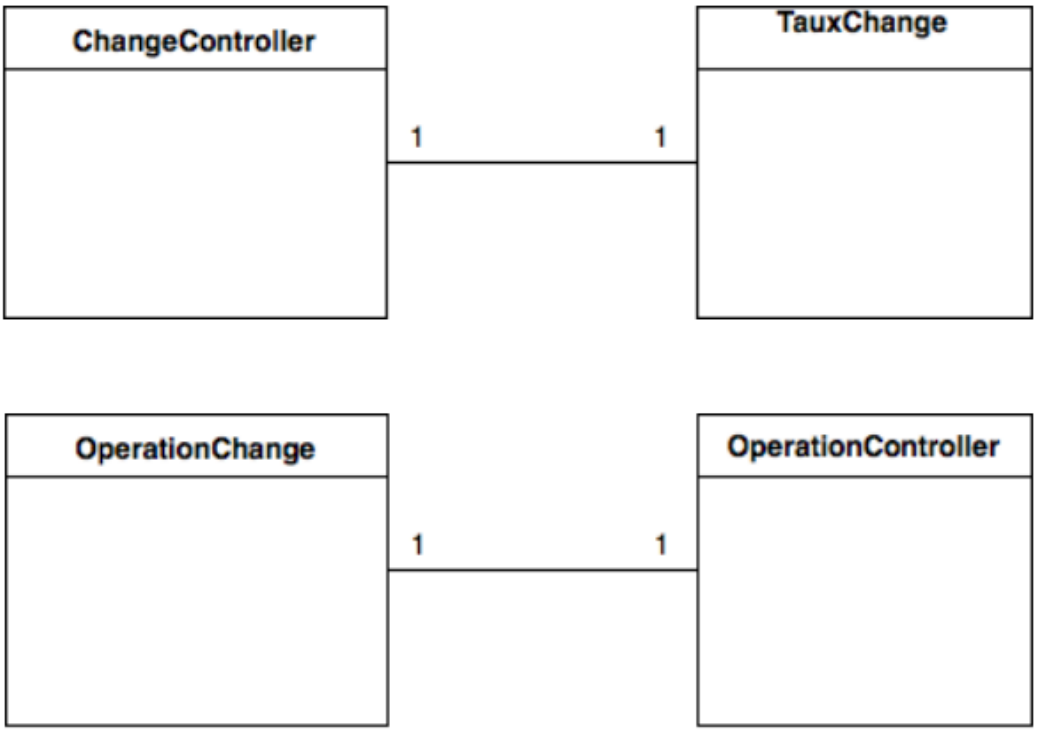
L’architecture se compose de deux microservices stockés dans deux répertoire différents (demo et operation) dans le répertoire github.

Chaque service est indépendant et gère ses propres ressources à commencer par la base de donnée.

Un docker aurait permis d’isoler encore davantage ces deux services en créant deux containers regroupants chacun leurs dépendances.

Suite à des difficultés que j’expliquerai plus tard je n’ai pas fait communiquer ces deux services.

Diagramme de classes



Ces deux classes correspondant aux deux microservices utilisent des Controller pour gérer l'API REST.

Méthodes disponibles

Pour le Taux de Change :

-/TauxChanges (GET) affiche toutes les instances

-/TauxCHange/{source},{dest} (GET) affiche une instance en fonction de sa clé primaire (devise_source, devise_destination)

-/CreerTC (POST) permet de créer une instance avec en body de requête les attributs de l'instance

-/SupprimerTC{source},{dest} (DELETE) supprime l'instance ayant pour clé primaire (devise_source, devise_destination)

On a ainsi toutes les opérations CRUD sachant que la modification peut être directement effectuée manuellement via H2

Pour l'Opération de Change :

-/OperationChanges (GET) affiche toutes les instances

-/OpérationChange/{source},{dest} (GET) affiche une instance en fonction de sa clé primaire (devise_source, devise_destination)

-/CreerOC (POST) permet de créer une instance avec en body de requête les attributs de l'instance

-/SupprimerOC{source},{dest} (DELETE) supprime l'instance ayant pour clé primaire (devise_source, devise_destination)

On a ainsi toutes les opérations CRUD sachant que la modification peut être directement effectuée manuellement via H2

Bilan

J’ai rencontré beaucoup de difficultés à réaliser ce projet. En effet pour commencer je n’étais familier avec aucun des outils utilisés.

De plus j’ai réalisé ce projet seul et dans un contexte de rattrapages, à savoir qu’aucun autre groupe ne réalisait ce projet en même temps que moi. Je n’ai donc pas pu recevoir d’aide. Les binômes de la première session ayant oublié les subtilités du projet et supprimé toutes leurs données relatives.

C’est pourquoi au moment du rendu je ne suis pas sûr que mon micro service tourne correctement. En effet sur mon poste de travil je n’ai pas réussi à faire communiquer IntelliJ et H2. Ce n’est pourtant pas faute d’avoir cherché. Le problème étant que l’installation d’IntelliJ et d’H2(faites et refaites à maintes reprises) n’a créé aucun raccourci ou executable pour lancer l’application. De ce fait il fallait que je lance une commande .sh dans le répertoire bin de ces deux outils pour les lancer. Linconvénient étant de monipoliser le terminal. Je ne pouvais donc pas lancer les deux outils en même temps. Toutes mes tentatives pour me connecter à H2 avec des url simples (par exemple localhost:8000/h2-console/) ont été un échec. C’est pourquoi je n’ai pas pu tester mon code et prendre en main l’API finale.

Heureusement grâce à postman et une implémentation static de base de donnée dans une classe, que j’ai supprimée lors de l’ajout de JPA au projet, me permettait de créer mes propres DAO et vérifier que les requetes généres par postman fonctionnaient. C’est pourquoi j’ai tout de même réalisé un code.

Le fait est que sans H2 je n’ai du coup pas cherché à faire communiquer mes deux microservices, ne sachant pas comment vérifier que les opérations se dérouleraient correctement.

Je retiens toutefois de ce projet une nouvelle approche de développement. Bien que la mise en place pour quelqu’un comme moi, ne connaissant pas ces outils, a été laborieuse, je reconnais que ces outils sont extrememement pratiques. Je préfère maintenant IntelliJ à Eclipse, Maven bien que relativement transparent est appréciable car il permet de faire compiler IntelliJ, Postman est très intéressant et c’est probablement l’outil que j’ai préféré de part sa simplicité d’usage et le temps qu’il fait gagner en développement de requetes http. J’ai comme regret de n’avoir pas pu manipuler h2 car je trouve son concept de mémoire cache très intéressant.

Malgré les difficultés rencontrées je ressors donc enrichi par ce projet et avec un nouvelle conception de la réalisation d’application, moi qui effectue une mission de maîtrise d’ouvrage, et j’espère pouvoir travailler à nouveau sur ce type d’application.