Architecture Microservices

Rapport de projet

Auteurs :

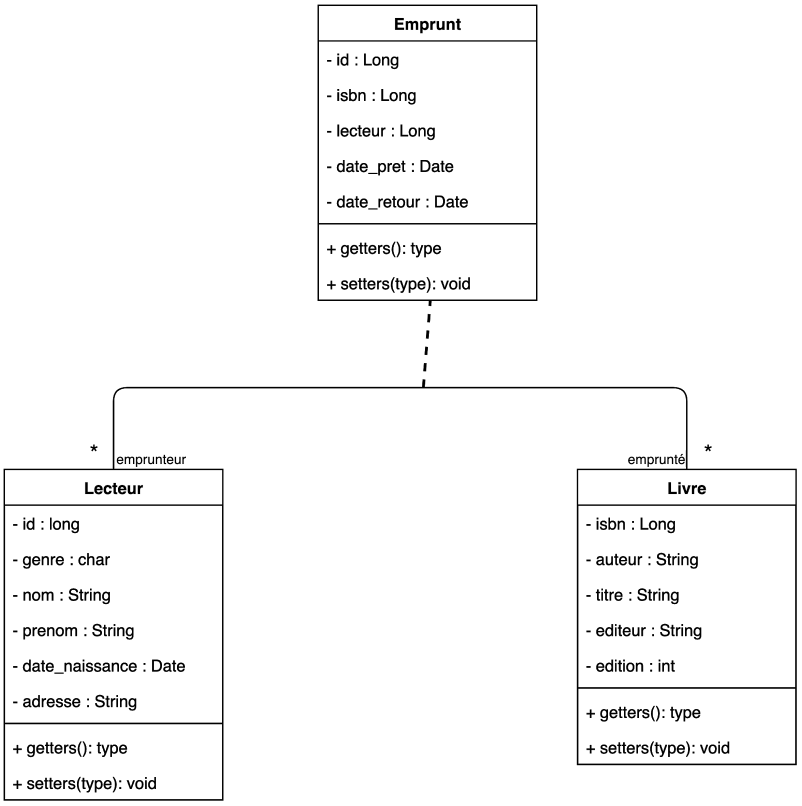
- Mickaël Peeren
- Quentin Sauvage

1. Instructions de compilation et d'exécution

Les instructions de compilation et d'exécution de notre application se trouvent dans le fichier README.md de notre repository Github disponible à l'adresse : https://github.com/Deeplygends/microservices.

2. Documentation technique

a. Diagramme de classes "métier"



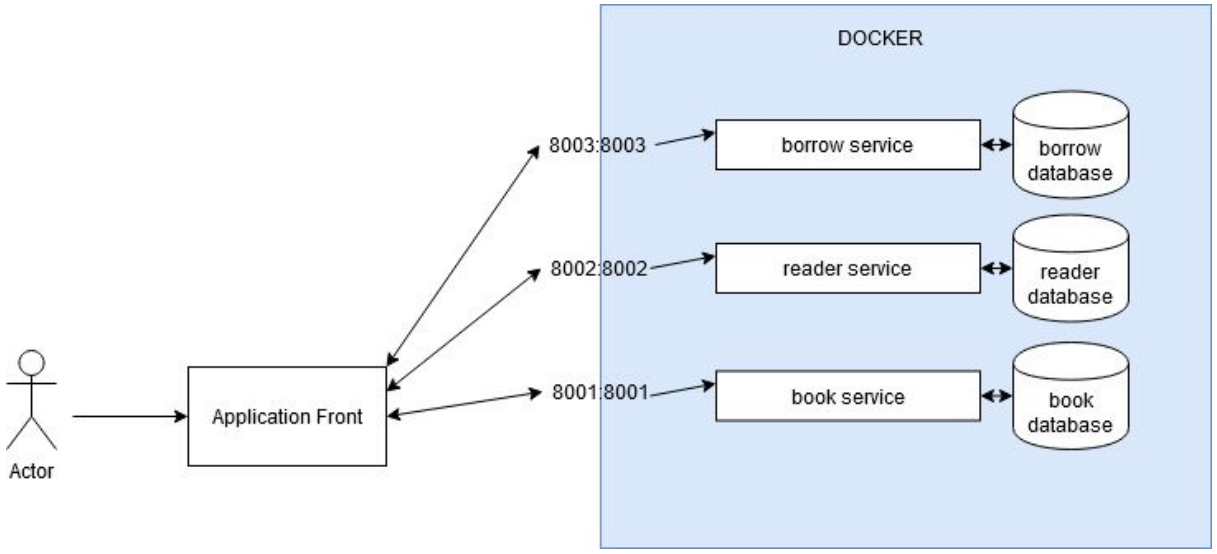
b. Schéma d'architecture

Notre application est composé d'une architecture en microservices développés en Java mise en oeuvre grâce au framework Spring Boot. Nous avons également mis en place une partie "frontend" de l'application afin de pouvoir la manipuler.

Nous avons fait le choix de découper l'application en 3 services afin de maximiser la mise en oeuvre de cette architecture. Cependant, la simplicité de ce système aurait pu mené à une architecture tout à fait fonctionnelle en 2 services.

Ainsi, la partie frontend est développée en JQuery. Cette partie pointe, à travers trois ports différents (8001, 8002 et 8003) sur trois services gérés par Docker (bookService, readerService et borrowService). Chacun de ces services manipule les données qui lui est propre dans une base de donnée h2 indépendante des autres services. De plus, aucun service ne peut communiquer directement avec un autre, ils doivent donc forcément passer par l'interface client. Ce choix est en parti basé sur le principe d'indépendance des services qui définit cette architecture. En effet, si bookService était dépendant de borrowService, alors toute panne de borrowService serait répercutée sur bookService.

Notre architecture est illustrée ci-après :



c. Choix techniques

Nous avons choisi d'utiliser d'un serveur WAMP pour exécuter le frontend de l'application. Nous avons choisi d'utiliser cela afin que notre API frontend s'exécute sur le domaine http://localhost, ce qui permet de partager les ressources provenant de différentes origines. En effet, en exécutant nos fichiers d'application frontend tel quel, ceux-ci ne sont pas portés par le domaine http://localhost ce qui les empêche d'accéder aux ressources fournies par nos services qui eux sont hébergés sur le domaine http://localhost grâce aux containers Docker. Ce mécanisme de partage des ressources entre des modules aux origines différentes est nommé CORS pour Cross Origin Resource Sharing.

Pour le service des emprunts, nous n'avons pas implémenté de gestion métier des emprunts en cours, du nombre d'exemplaires, etc. Le service permet donc d'emprunter autant de fois que nécessaire un même livre. De plus, nous gérons la suppression en cascade lors de la suppression d'un livre ou d'un lecteur enregistré du côté frontend de l'application dans un souci d'indépendance entre services. En outre, il n'est possible de modifier un emprunt seulement pour lui affecter une date de retour qui sera alors la date du jour ou l'on déclare le livre retourné.

3. Bilan du projet

a. Bilan technique

Au cours de la réalisation de ce projet, nous avons rencontré quelques points de difficultés :

- Lien d'une base persistante (SQLServer) dans Docker : problème de droits d'accès à certains fichiers d'exécution à travers Docker sur Windows, nous avons alors préféré migrer vers une base de données volatile h2.
- Parsing d'une Date JQuery vers JPA : en requêtant l'API, nous recevions une valeur null pour les attributs de type Date qui était insérée dans la base de données à la place de la date envoyée via JSON par l'interface. Pour pallier ce problème, nous avons fait en sorte de transmettre vers JPA une date au format String que notre application Java se chargeait ensuite de convertir en Date pour l'insérer dans la base de données.
- CORS : comme nous l'avons expliqué dans le chapitre précédent, nous avons parfois eu des soucis à récupérer certaines ressources depuis nos propres services lorsque ceux-ci étaient exécutés avec Docker. Nous avons trouvé la solution d'utiliser Wamp afin d'exécuter la partie frontend de notre application et donc utiliser le même domaine pour l'intégralité de notre application, ce qui efface les éventuelles complications liées au CORS.
- Windows Docker, documentation en général : nous avons dans un premier temps éprouver une grande difficulté à configurer Docker sur Windows. Cela est dû au fait que la documentation concernant les technologies que nous avons utilisées est parfois particulièrement vaste et que les configurations nécessitent une précision accrue. Il est donc difficile de trouver les instructions qui correspondent parfaitement à nos besoins et à nos restrictions.

Réussites :

- Nous sommes parvenus à mettre en place et configurer Docker sur Windows et sur Mac OS.
- Nous avons réussi à développer et fournir une application fonctionnelle basée sur le framework Spring et exécutable sur toute plateforme.

b. Bilan personnel

D'un point de vue personnel, nous avons trouvé l'aspect devops, c'est-à-dire gérant la partie frontend et la partie backend de l'application, particulièrement motivante. En effet, cela nous a permis de suivre le fonctionnement de notre système de A à Z et donc de développer une attitude quasi omnisciente la concernant, ce qui est gratifiant.
D'autre part, nous avons été amenés à utiliser une architecture et un framework populaires et reconnus par la société des développeurs informatiques, ce qui fait de nous des techniciens désormais compétents sur ces technologies importantes, notamment avec la modernisation par le cloud des systèmes informatiques.

c. Pistes d’amélioration

Dans les pistes d’amélioration, nous avons noté qu’il serait possible d’utiliser Kubernetes afin d’automatiser le déploiement et la gestion de notre application conteneurisée. D’après notre expérience suivant la tentative d’utilisation de cet outil, nous avons déterminé qu’il était nécessaire de disposer d’une connaissance avancée et stable des technologies Docker et Kubernetes pour mener à bien son implémentation au coeur d’un projet. Dans le cadre de celui-ci, nous n’avons pas été en mesure de disposer d’une automatisation des déploiements de nos conteneurs avec Kubernetes.

Nous avons également tenté d’utiliser les outils fournis par Amazon Web Service dans le cadre du développement de notre application mais nous avons préféré réaliser celui-ci localement sur nos ordinateurs en vue de notre manque d’expérience sur les technologies d’AWS.