

UNIVERSITÉ CHEIKH ANTA DIOP



FACULTÉ DES SCIENCES ET TECHNIQUES
DÉPARTEMENT DE MATHÉMATIQUES-INFORMATIQUE

Mémoire présenté pour l'obtention du diplôme de

Master en Informatique
Option : **Systèmes d'Information Répartis**

Par

KÉBA DEME

Sur le sujet

Architecture de microservices pour le développement d'applications cloud natives

Application au développement d'un logiciel de dématérialisation de la gestion scolaire

Soutenu le 13 Novembre 2021 devant le jury composé comme suit :

Président : Pr Abdourahmane Raimy , PROFESSEUR TITULAIRE UCAD
Examinateurs : Pr Aliou Boly, PROFESSEUR TITULAIRE UCAD
Dr Ndiouma Bame , MAÎTRE-ASSISTANT UCAD
Encadrant : Dr Djamal A N Seck , MAÎTRE DE CONFÉRENCES UCAD

Année universitaire 2019-2020

DEDICACES

Je dédie ce mémoire, événement marquant de ma vie, à tous ceux qui souhaitent me voir réussir.

REMERCIEMENTS

Ce travail est le fruit de la combinaison d'efforts de plusieurs personnes. Je remercie tout d'abord le tout-puissant qui, par sa grâce m'a permis d'arriver au bout de mes efforts en me donnant la santé, la force, le courage et en me faisant entourer des merveilleuses personnes dont je tiens à remercier.

Je remercie :

Mon encadreur, Dr Djamal Abdoul Nasser Seck pour son encadrement sans faille, son soutien moral, sa rigueur au travail, ses multiples conseils, ses orientations et sa disponibilité malgré ses multiples occupations ;

Les membres du jury qui me font le grand honneur d'évaluer ce travail ;

Mes chers parents, qui m'ont donné le meilleur de ce qu'ils ont, que Dieu nous les garde ;

Mon papa Djibril Gaye et ma tante Aissatou Touré pour leur soutien inconditionnel.

Mes Frères et sœurs pour leurs encouragements durant tout mon parcours ;

Mes camarades, amis et connaissances ;

Mes collègues de Gaindé 2000 et notre chef de projet Ndeye Khady Niang ;

Tous ceux qui de près ou de loin ont contribué à l'accomplissement de ce travail.

Table des matières

Introduction générale	1
1 Contexte et Problématique	2
1.1 Contexte	2
1.2 Diagnostic de l'existant: architecture monolithique	2
1.3 Objectifs	4
1.4 Solution proposée	4
2 Concepts fondamentaux	5
2.1 Concepts liés à l'architecture de microservices	5
2.1.1 Domain Driven Design(DDD)	5
2.1.2 Développement à base de composants	6
2.1.3 Polyglot persistence(Persistence polyglotte)	6
2.1.4 Polyglot programming(Programmation polyglotte)	6
2.2 Architecture de microservices	6
2.2.1 Définition de l'architecture de microservices	6
2.2.2 Caractéristiques de l'architecture de microservices	7
2.2.3 Communication inter-processus dans l'architecture de microservices	9
2.2.4 Définition des API dans une architecture de microservices	11
2.2.5 Découverte de services dans une architecture de microservices	11
2.2.6 Architecture orientée événement	14
2.2.7 Gestion des données dans une architecture de microservices	15
2.2.8 Gestion de la sécurité dans une architecture de microservices	16
2.2.9 Bilan sur l'architecture de microservices	16
2.3 Application cloud native	17
2.3.1 Définition	17
2.3.2 Avantages	18
2.4 Architecture d'une application en microservices	18

3 Analyse et conception de la solution	21
3.1 Méthodologie et approche de développement	21
3.1.1 Méthodologie SCRUM	21
3.1.2 L'approche DevOps	22
3.2 Analyse et spécification des besoins	24
3.2.1 Présentation de l'application G-School	24
3.2.2 Besoins fonctionnels	24
3.2.3 Besoins non fonctionnels	24
3.2.4 Identification des acteurs	25
3.2.5 Modélisation globale du diagramme de classe	25
3.2.6 Division de l'application G-School en microservices	26
4 Développement des microservices	28
4.1 Choix technologiques	28
4.1.1 Choix des frameworks de back-end	28
4.1.2 Choix des frameworks de front-end	30
4.1.3 Choix de message broker	32
4.2 Architecture technique	34
4.3 Conception et réalisation des microservices identifiés	35
4.3.1 Microservice Inscription-service	35
4.3.2 Microservice Cours-service	42
4.3.3 Microservice Classe-service	49
4.3.4 Microservice Evaluation-service	52
4.3.5 Microservice Utilisateur-service	57
5 Intégration des microservices	59
5.1 Service de découverte	59
5.2 L'API Gateway: Spring Cloud Gateway	61
5.3 Communication entre les microservices	61
5.3.1 Communication synchrone avec tolérance aux pannes	61
5.3.2 Communication asynchrone basée sur des messages	64
6 Déploiement Et Test de Charge	65
6.1 Déploiement des microservices avec Docker	65
6.1.1 Présentation de Docker	65
6.1.2 PRÉPARATION DES CONTENEURS DE DÉPLOIEMENT	66
6.1.3 Résultats du déploiement	68
6.2 Test de charge avec Gatling	70
6.2.1 Scénario à tester	70
6.2.2 Résultats obtenus après l'exécution du script de test de charge	70

TABLE DES MATIÈRES

iv

Conclusion générale et perspectives	72
Bibliographie	73

Table des figures

1.1	Architecture monolithique	3
1.2	Limites de l'architecture monolithique	3
2.1	Architecture monolithique vs architecture de microservices	7
2.2	Communication inter-processus	10
2.3	Interaction entre les services	11
2.4	Découverte de services	12
2.5	Découverte côté client	13
2.6	Découverte côté serveur	14
2.7	Composants d'une architecture en microservices	19
3.1	Pratique SCRUM: vue d'ensemble	22
3.2	Architecture de référence DevOPs	23
3.3	Diagramme de classe de G-School	26
3.4	Carte de mapping du diagramme de classe	27
4.1	Logo Quarkus	28
4.2	Logo Spring boot	28
4.3	Logo Vert.x	29
4.4	Logo Moleculer	29
4.5	Logo Micronaut	29
4.6	Tableau comparatif des frameworks Back-end	30
4.7	Logo VueJs boot	30
4.8	Logo Angualr	31
4.9	Logo React	31
4.10	Tableau comparatif des frameworks Front-end	31
4.11	Logo Apache Kafka	32
4.12	Logo React	32
4.13	Logo RabbitMQ	33
4.14	Tableau comparatif des logiciels de message broker	33
4.15	Architecture technique du logiciel G-school	34

4.16 Diagramme de cas d'utilisation du microservice Inscription-service	35
4.17 Diagramme de classe du microservice Inscription-service	36
4.18 Diagramme de séquence pour inscrire un élève	37
4.19 Architecture logicielle du microservice Inscription-service	37
4.20 Architecture technique du microservice Inscription-service	38
4.21 Page d'accueil du secrétaire de l'école	39
4.22 La liste de tous les élèves inscrits	39
4.23 Interface inscription 1	40
4.24 Interface inscription 2	40
4.25 Interface inscription 3	41
4.26 liste des élèves inscrits avec les boutons d'action	41
4.27 la fiche de l'élève	42
4.28 Diagramme de cas d'utilisation du microservice Cours-service	43
4.29 Diagramme de classe du microservice Cours-service	44
4.30 Diagramme de séquence pour noter les absents	44
4.31 Architecture technique du microservice Cours-service	45
4.32 La liste des professeurs	46
4.33 Ajouter un professeur	46
4.34 La liste des cours	47
4.35 Ajouter un cours 1	47
4.36 Ajouter un cours 2	48
4.37 Liste des absents d'une classe	48
4.38 Noter les absents d'un cours	49
4.39 Diagramme de cas d'utilisation du microservice Classe-service	50
4.40 Diagramme de classe du microservice Cours-service	50
4.41 Liste des classes	51
4.42 Liste des matières dispensées dans une classe	51
4.43 Liste des matières	52
4.44 Diagramme de cas d'utilisation du microservice Evaluation-service	53
4.45 Diagramme de classe du microservice Evaluation-service	53
4.46 Architecture technique du microservice Evaluation-service	54
4.47 Liste des évaluations d'une classe	55
4.48 Ajouter une évaluation avec le pop-up de confirmation	55
4.49 Liste des évaluations de la classe TL2B avec les actions	56
4.50 Interface pour ajouter les notes d'une évaluation	56
4.51 Liste des notes d'une évaluation	57
4.52 Diagramme de classe du microservice Utilisateur-service	57

4.53 Interface de connexion	58
5.1 Enregistrement des microservices de G-School dans l'annuaire	60
5.2 Dashboard Eureka	60
5.3 Fonctionnement de l'API Gateway	61
5.4 Communication synchrone entre le microservice de la gestion des inscriptions et celui de la gestion des classes	62
5.5 Diagramme d'état du disjoncteur de circuit	63
5.6 Hystrix Dashboard avec des appels distants réussis	63
5.7 Hystrix Dashboard avec des appels distants échoués	64
5.8 Communication asynchrone entre le microservice de la gestion des inscriptions et celui de la gestion des notifications	64
6.1 Architecture docker	66
6.2 Conteneurisation de G-school	66
6.3 Le fichier Dockerfile de notre service de découverte	67
6.4 Le fichier Docker compose des services de notre application	68
6.5 Le service de découverte en ligne	69
6.6 L'application frontend en ligne	69
6.7 Evolution du nombre d'utilisateur actif en fonction du temps	71
6.8 Distribution des temps de réponse des requêtes exécutées	71

Résumé

Dans ce travail de mémoire de master 2 en informatique, notre objectif était de faire une étude générale sur l'architecture de microservices ; celle-ci est définie comme une architecture « cloud native » par laquelle un système logiciel peut être réalisé sous la forme de petits services. Chacun de ces services peut être déployé indépendamment sur une plateforme différente et s'exécuter dans son propre processus tout en communiquant par le biais de mécanismes légers.

Dans ce mémoire, nous avons d'abord commencé par faire un diagnostic sur l'architecture monolithique. Au cours de ce diagnostic, nous avons pu déceler certaines limites des applications monolithes et parmi ces limites, on peut citer : les problèmes de performances, la tolérance aux pannes limites, les modifications coûteuses, la limite technologique, etc.

Ensuite, nous avons abordé tous les concepts de l'architecture de microservices.

Enfin, pour illustrer notre travail, nous avons utilisé ce style d'architecture pour réaliser une application « cloud native » qui vise à dématérialiser la gestion scolaire des établissements publics et privés. C'est une solution qui aide les gestionnaires de l'école à gérer de façon optimisée leur établissement. Cette application est déployée dans le cloud de « ionos » en utilisant la technologie de docker. Des tests de charges ont été aussi faits pour évaluer les performances de l'application et on a obtenu de très bon résultats.

Abstract

In this Master's thesis work in computer science, our objective was to make a general study of the microservices architecture; this is defined as a « cloud-native » architecture by which a software system can be realized as small services. Each of these services can be deployed independently on a different platform and run in its own process while communicating through lightweight mechanisms.

In this dissertation, we started by making a diagnosis of the monolithic architecture. During this diagnosis, we were able to identify some limitations of monolithic applications and among these limitations are: performance issues, borderline fault tolerance, costly modifications, technological limitation, etc. Next, we covered all the concepts of microservices architecture.

Finally, to illustrate our work, we have used this style of architecture to create a « cloud native » application that aims to dematerialize the school management of public and private schools. It is a solution that helps school managers to manage their school in an optimized way. This application is deployed in the « ionos » cloud using docker technology. Load tests were also done to evaluate the performance of the application and very good results were obtained.

Introduction générale

Depuis quelques années, les technologies de l'information et de la communication se présentent comme une nécessité dans tous les domaines de la société. Pour évoluer et accroître leurs efficacités, les organisations, quelles que soient leurs natures, ont besoin d'assimiler la culture de l'innovation portée par ces nouvelles technologies. Ainsi des applications informatiques sont utilisées dans presque tous les domaines pour résoudre des problèmes et offrir des services. Cependant, la plupart de ces solutions informatiques ont été bâties suivant une architecture monolithique. Cette dernière présente beaucoup de limites parmi lesquelles on peut citer: la limite technologique, les modifications coûteuses, la tolérance aux pannes limitée, les difficultés liées à la maintenabilité et la scalabilité. Ainsi une entreprise évoluant dans l'industrie de développement de logiciels devra orienter ses recherches et ses choix sur des techniques qui rendent ses solutions plus fiables et plus puissantes tout en répondant aux besoins évolutifs de ses clients. En particulier, elle doit choisir une bonne architecture d'application pour construire des logiciels robustes, scalables, maintenables et flexibles. En effet, une architecture d'application décrit les modèles et les techniques à utiliser pour concevoir et créer une application. Elle fournit en quelque sorte une feuille de route ainsi que les meilleures pratiques à suivre pour créer une application bien structurée. C'est dans cette optique que s'inscrit notre mémoire de fin d'étude qui consiste à faire une étude générale sur un nouveau style d'architecture appelé architecture de microservices. Pour un exemple de conception, nous réalisons dans ce présent mémoire une application de dématérialisation de la gestion scolaire basée sur l'architecture de microservices.

Ce présent rapport décrit les différentes étapes de notre travail et est structuré en cinq(5) chapitres articulés comme suit:

- Le premier chapitre concerne le contexte et la problématique. Nous y ferons aussi un diagnostic sur l'architecture existante.
- Le deuxième chapitre présente les concepts fondamentaux qui sont en rapport avec l'architecture de microservices.
- Le troisième chapitre est consacré à la conception du système. Nous spécifions d'abord les besoins de l'application. Ensuite nous identifierons les différents microservices de notre application en nous basant sur la modélisation. Et à la fin nous présenterons l'architecture d'une application en microservices.
- Le quatrième chapitre est consacré à la réalisation des microservices.
- Le cinquième chapitre sera consacré à l'intégration de nos différents microservices déjà réalisés au chapitre précédent.
- Dans le dernier chapitre nous ferons le déploiement de l'application avec docker et les tests de performance en utilisant gatling.

Chapitre 1

Contexte et Problématique

Dans ce premier chapitre introductif, nous nous intéresserons tout d'abord au contexte de notre projet. Nous ferons par la suite un diagnostic sur l'existant. Enfin, nous présenterons la solution proposée.

1.1 Contexte

Ces dernières années l'informatique est en train de prendre une tournure radicale avec les nouvelles technologies telles que le cloud computing, le big data, l'IoT Ainsi on a noté une forte migration des entreprises et du monde universitaire vers le cloud. Malgré les nombreux avantages offerts par le cloud, la plupart des architectures d'applications traditionnelles ne sont pas encore prêtes à les exploiter pleinement mais aussi leur adaptation à ce nouvel environnement est une tâche non triviale. Cela peut souvent conduire à faire table rase des développements passés, et à les reconstruire à partir de zéro. Conscient de l'utilité des logiciels qui deviennent de plus en plus important dans la course de l'innovation, les entreprises ne limitent plus leur gouvernance à la gestion des ressources matérielles et humaines mais elles essaient aussi de trouver des solutions pour accroître la vitesse, la flexibilité et la qualité du développement des applications tout en réduisant les risques. Ces derniers sont fortement dépendants de l'architecture logicielle adoptée. De ce fait, le changement de notre approche de la conception, du développement, du déploiement et de la gestion de nos applications devient impératif.

1.2 Diagnostic de l'existant: architecture monolithique

L'architecture monolithique est l'une des approches les plus utilisées pour le développement d'applications. Elle consiste à développer des applications en un seul bloc(war, jar, Ear, dll,...) avec une même technologie et déployée dans un serveur d'application. C'est une approche où tous les besoins fonctionnels de l'application sont centralisés. Parmi les anciennes applications d'entreprise, nombreuses sont celles basées sur des architectures monolithiques. Un logiciel monolithique est conçu pour être autonome ; ses composants sont interconnectés et interdépendants plutôt qu'associés de manière flexible comme dans le cas des programmes modulaires. Dans ce type d'architecture étroitement intégrée, chaque composant et ceux qui lui sont associés doivent être présents pour permettre l'exécution ou la compilation du code. L'avantage de l'architecture monolithique est sa simplicité de mise en place avec une unique méthode de déploiement, une unique architecture et une unique technologie. Cependant, le monolithe "pur" est à proscrire

car il crée une forte dépendance dans le code. Et cette dépendance peut à terme entraîner un coût important lorsqu'il s'agira de faire évoluer une fonctionnalité.

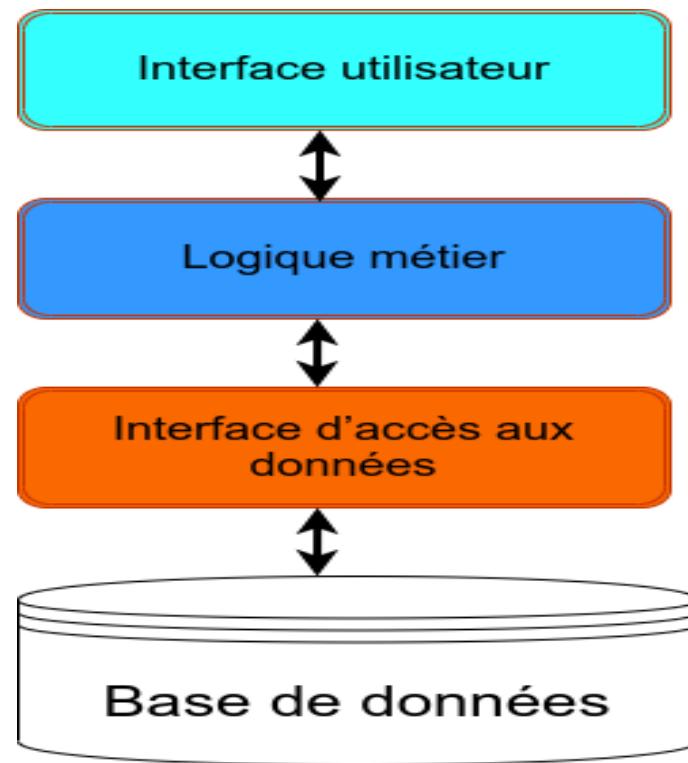


FIG. 1.1 – Architecture monolithique

Le tableau 1 ci-dessus récapitule les principales limites de l'architecture monolithique et leurs impacts sur une application.

Limite	Impact sur une application
Modification coûteuse	Chaque modification nécessite de tester les régressions et redéployer toute l'application
Limites technologiques	Une seule technologie est choisie pour développer toutes les fonctionnalités de l'application. Les développeurs de l'application doivent maîtriser les mêmes technologies
Tolérance aux pannes limitée	Si une fonctionnalité bloque le processus, les autres fonctionnalités ne seront pas disponibles
Livraison en bloc	Le client attend beaucoup de temps pour commencer à voir les premières versions de l'application
Maintenabilité du code	Quand les besoins fonctionnels augmentent, le code devient de moins à moins lisible et testable, la qualité et la productivité baissent et enfin la dette technique s'accumule
Scalabilité	Scalier une application monolithique entraîne souvent une augmentation de la consommation de ressources conséquentes de manière inutile

FIG. 1.2 – Limites de l'architecture monolithique

1.3 Objectifs

En se basant sur le diagnostic fait précédemment, nous avons prévu dans ce mémoire de faire une étude sur un autre type d'architecture qui permet d'apporter des solutions aux différentes limites de l'architecture monolithique. L'architecture que nous allons proposer dans ce mémoire devrait apporter des solutions aux limites des applications développées avec l'approche monolithique en privilégiant les tendances DevOps et les avantages offerts par le cloud.

1.4 Solution proposée

Pour remédier aux limites que présente l'architecture monolithique, nous proposons dans ce présent mémoire un nouveau style d'architecture appelé architecture de microservices. Nous ferons une étude sur les différents concepts liés à ce type d'architecture mais aussi nous décrirons les avantages que nous offre cette architecture et la méthodologie à suivre pour réussir à concevoir une application cloud native basés sur l'architecture de microservices. Pour illustrer notre travail, nous allons utiliser ce style d'architecture pour réaliser une application cloud native de dématérialisation de la gestion scolaire.

Chapitre 2

Concepts fondamentaux

Dans ce chapitre, nous nous concentrerons sur les concepts de base liés à notre travail. Nous présenterons d'abord les concepts liés à l'architecture des microservices. Puis nous décrirons par la suite l'architecture de microservices. Enfin nous expliquerons le concept d'application cloud native.

2.1 Concepts liés à l'architecture de microservices

2.1.1 Domain Driven Design(DDD)

Le développement logiciel désigne le processus consistant à bâtir des applications informatiques. La conception de logiciels est un art, et comme tout art elle ne peut pas être enseignée et apprise comme une science précise, au moyen de théorèmes et de formules. Nous pouvons découvrir des principes et des techniques utiles à appliquer tout au long du processus de création de logiciel, mais nous ne serons probablement jamais capables de fournir un chemin exact à suivre en partant du besoin du monde réel pour arriver jusqu'au module de code destiné à répondre à ce besoin.

DDD est une approche de développement de logiciels centrée sur le métier au travers de design patterns de conception, des modèles conceptuels. Il vise à accorder de l'importance au domaine métier. Selon Eric Evans, un modèle métier n'est pas un diagramme particulier. C'est une abstraction rigoureuse des connaissances d'expert du domaine. En effet, dans la plupart des logiciels, la logique métier qui est implémentée est ce qui constitue la plus grande valeur ajoutée puisque c'est cette logique qui rend le logiciel fonctionnel. L'approche DDD vise, dans un premier temps, à isoler un domaine métier. Un domaine métier riche comporte les caractéristiques suivantes:

- Il approfondit les règles métier spécifiques et il est en accord avec le modèle d'entreprise, avec la stratégie et les processus métier.
- Il doit être isolé des autres domaines métier et des autres couches de l'architecture de l'application.
- Le modèle doit être construit avec un couplage faible avec les autres couches de l'application
- Il doit être une couche abstraite et assez séparée pour être facilement maintenue, testée et versionnée.

- Le modèle doit être conçu avec le moins de dépendances possibles avec une technologie ou un framework.
- Le domaine métier ne doit pas comporter de détails d'implémentation de la persistance.

2.1.2 Développement à base de composants

L'ingénierie des logiciels basée composant fait une séparation des préoccupations du système en des entités clairement définies, appelées composants. Ces composants réutilisables sont définis et composés ensemble. La notion de composant vise à identifier une partie d'un système, autonome et bien délimitée. Même si l'ambition de départ est la réutilisation de composants pour concevoir un système par assemblage de ceux-ci, la notion de composant mérite d'être considérée comme un véritable élément d'architecture pour l'analyse et la conception de système d'information

2.1.3 Polyglot persistence(Persistence polyglotte)

La persistance polyglotte est un terme qui fait référence à l'utilisation de plusieurs technologies de stockage de données pour divers besoins de stockage de données dans une application ou dans des composants plus petits d'une application. En d'autres termes, on peut dire que c'est une solution hybride de gestion des données. Des besoins de stockage de données aussi variés pourraient survenir dans les deux cas, c'est-à-dire une entreprise avec plusieurs applications ou des composants singuliers d'une application devant stocker les données différemment.

2.1.4 Polyglot programming(Programmation polyglotte)

En 2006, Neal Ford a inventé le terme de programmation polyglotte , pour exprimer l'idée que les applications devraient être écrites dans un mélange de langages afin de tirer parti du fait que différents langages conviennent pour résoudre différents problèmes. Les applications complexes combinent différents types de problèmes, donc choisir le bon langage pour le travail peut être plus productif que d'essayer d'intégrer tous les aspects dans un seul langage.

2.2 Architecture de microservices

2.2.1 Définition de l'architectured e de microservices

Il existe un certain nombre de définitions des microservices. La plus connue reste celle de Martin Fowler qui dit: « Le style architectural de microservices est une approche permettant de développer une application unique sous la forme d'une suite logicielle intégrant plusieurs services. Ces services sont construits autour des capacités de l'entreprise et peuvent être déployés de façon indépendante. » Concrètement, l'architecture de microservices est une approche qui consiste à développer une seule application comme une suite de petits services, chacun fonctionnant selon son propre processus et communiquant avec des mécanismes légers, souvent une API de ressources HTTP. Ces services sont construits autour des compétences métiers et peuvent être déployés indépendamment. Ainsi, chacun peut être écrit dans un langage qui lui est propre et peut fonctionner (ou dysfonctionner) sans affecter les autres. Il existe un minimum de gestion centralisée de ces services, qui peuvent être écrits dans différents langages de programmation et utiliser différentes technologies de stockage des données.

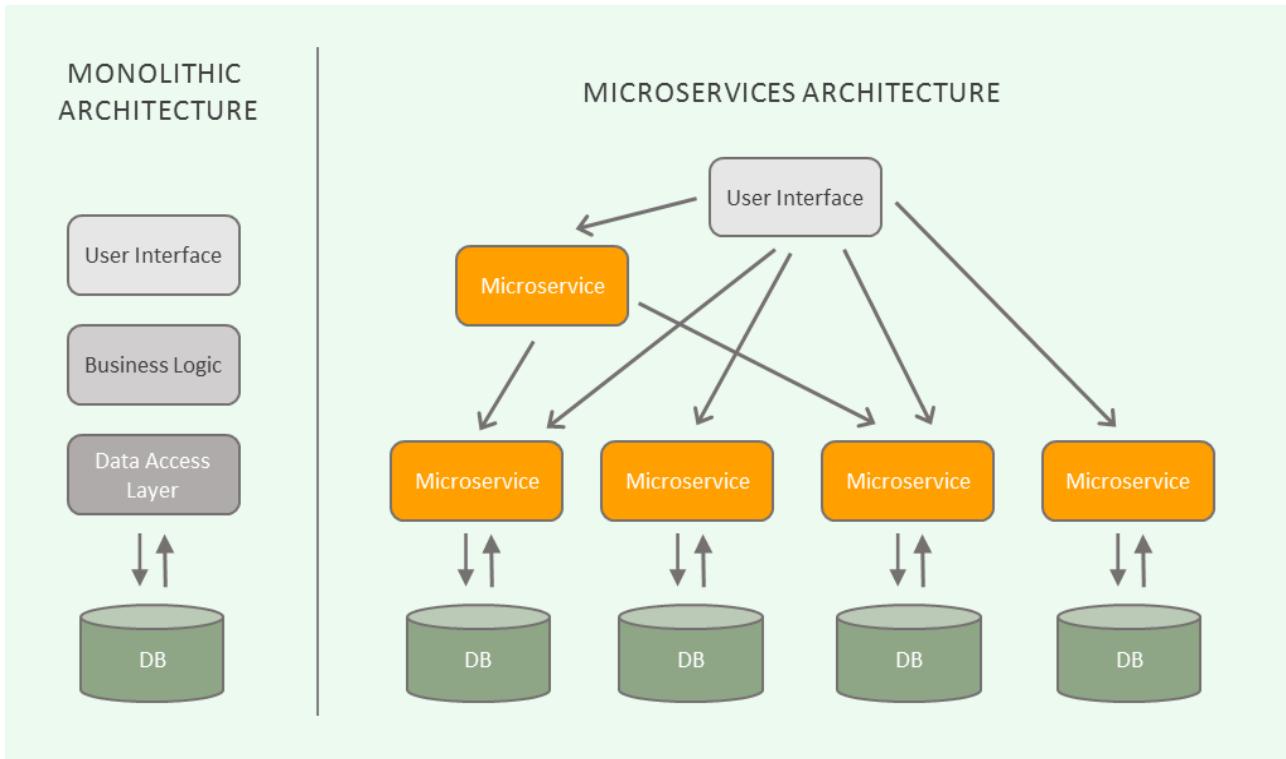


FIG. 2.1 – Architecture monolithique vs architecture de microservices[2]

2.2.2 Caractéristiques de l'architecture de microservices

D'après Martin Fowler[7], informaticien, conférencier et consultant britannique dans la conception de logiciels d'entreprise, l'architecture de microservices possède neuf(9) caractéristiques. Cependant il faut noter que toutes les architectures de microservices ne possèdent pas toutes les caractéristiques décrites par Martin Fowler.

2.2.2.1 Division en composant via les services

Hérité du génie logiciel à base de composant, un composant est une unité logiciel qui est indépendamment modifiable et remplaçable. Un logiciel développé avec l'architecture de microservices est décomposé en plusieurs services qui seront utilisés comme des composants. Chaque service est développé, testé et déployé séparément des autres. Un changement dans un service ne nécessite que le déploiement du service concerné. La seule relation entre les services est l'échange des données à travers les différents APIs qu'ils exposent. Ce qui permet d'éviter le couplage fort entre les composants.

2.2.2.2 Organisation autour des capacités métiers

Au début, les applications contenaient toutes les instructions nécessaires pour l'exécution des programmes. Ensuite les applications deviennent de plus en plus riches en fonctionnalités et par conséquent plus complexes. De ce fait, les développeurs ont commencé à diviser les applications selon les couches techniques. Ceci permet d'avoir une équipe d'interface utilisateur, une équipe de développement métier et une équipe de base de données. Cependant dans l'approche basée sur l'architecture de microservices, une application est décomposée en des services centrés sur des capacités métiers. Par conséquent, les équipes sont interfonctionnelles avec tous les niveaux

de compétences requises pour le développement: expérience utilisateur, base de données et gestion de projet.

2.2.2.3 Un Produit, pas un Projet

Ce concept peut être lié à la méthodologie Agile et l'approche DevOps présentés dans le premier chapitre. L'objectif est de livrer un morceau de logiciel qui est considéré comme terminé. Dans l'approche microservice, une équipe est responsable d'un produit tout au long de son cycle de vie. L'équipe de développement est le responsable du logiciel produit et doit suivre son comportement en production. Ceci renforce le lien entre les développeurs, les utilisateurs et les exploitants.

2.2.2.4 Points de terminaison intelligents et tuyaux stupides

Pour mieux comprendre ce concept, prenons le cas de l'Entreprise Service Bus(ESB), les produits ESB incluent souvent des fonctionnalités sophistiquées pour le routage des messages, la chorégraphie, la transformation et l'application des règles métier. Les applications utilisant les microservices favorisent l'utilisation de communications sans intelligence, qui ne font que transmettre les messages, alors que le service lui-même fait le reste. Les applications en micro-services visent à être aussi découplées et aussi cohérentes. Elles reçoivent une demande, appliquent la logique appropriée et produisent une réponse.

2.2.2.5 Gouvernance décentralisée

L'architecture de microservices favorise la gouvernance décentralisée. En général, la « gouvernance » signifie établir et appliquer la façon dont les personnes et les solutions travaillent ensemble pour atteindre les objectifs organisationnels. Dans l'architecture de microservices, les microservices sont conçus comme des services entièrement indépendants et découplés avec une variété de technologies et de plates-formes. Il n'est donc pas nécessaire de définir une norme commune pour la conception et le développement des services. Ainsi, nous pouvons résumer les capacités de gouvernance décentralisée des microservices comme suit:

- Dans l'architecture de microservices, il n'est pas nécessaire d'avoir une gouvernance centralisée au moment de la conception.
- Les microservices peuvent prendre leurs propres décisions concernant sa conception et sa mise en œuvre.
- L'architecture de microservices favorise le partage de services communs réutilisables.

La gouvernance décentralisée permet aux équipes de développement d'exploiter les avantages offerts par les différents langages de programmation. Les équipes qui créent des microservices préfèrent également une approche différente des normes. Plutôt que d'utiliser un ensemble de normes définies écrites quelque part sur papier, ils préfèrent l'idée de produire des outils utiles que d'autres développeurs peuvent utiliser pour résoudre des problèmes similaires à ceux auxquels ils sont confrontés.

2.2.2.6 Gestion décentralisée des données

Dans une architecture monolithique, l'application stocke les données dans une base de données unique et centralisée. Dans l'architecture de microservices, les fonctionnalités sont dispersées sur plusieurs microservices et, si nous utilisons la même base de données centralisée, alors les

microservices ne seront plus indépendants les uns des autres. L'architecture de microservices favorise l'approche de la persistance polyglotte qui admet l'utilisation de plusieurs bases de données. Par conséquent, chaque microservice doit avoir sa propre base de données.

2.2.2.7 Automatisation de l'infrastructure

Ces dernières années, les outils d'automatisation de l'infrastructure ont tellement évolué. L'évolution du cloud et des outils DevOps ont beaucoup facilité le déploiement et l'exploitation des microservices. Les équipes qui développent les microservices ont une vaste expérience en intégration continue et livraison continue. Elles ont à leur disposition des outils qui leur permettent d'utiliser les techniques d'automatisation des infrastructures.

2.2.2.8 Conception pour l'échec

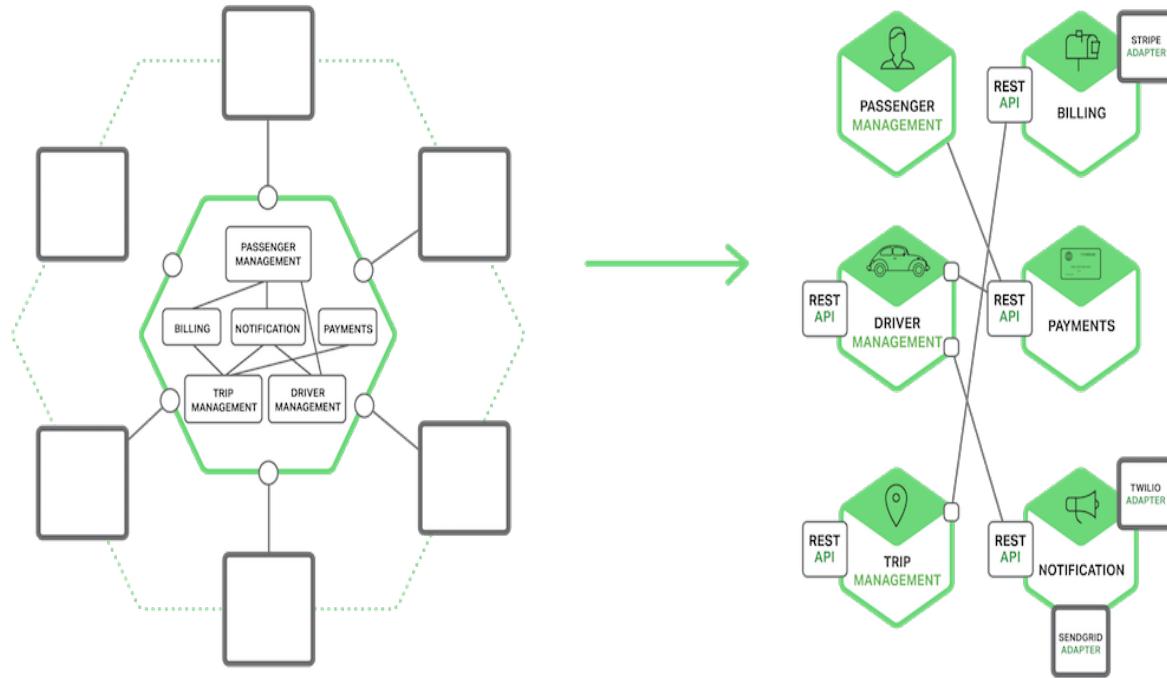
Les microservices sont conçus pour être tolérants aux pannes. La défaillance dans un service ne doit pas impacter les autres services de l'application. Étant donné que les services peuvent échouer à tout moment, il est important de pouvoir détecter rapidement les pannes et, si possible, de restaurer automatiquement le service. Les applications de microservices mettent beaucoup l'accent sur la surveillance en temps réel de l'application. La surveillance sémantique peut fournir un système d'alerte précoce en cas de problème qui incite les équipes de développement à effectuer un suivi et à enquêter.

2.2.2.9 Conception évolutive

Toutes les applications réussies évoluent avec le temps, que ce soit pour résoudre les bogues, ajouter de nouvelles fonctionnalités, introduire de nouvelles technologies ou améliorer l'évolutivité et la résilience des systèmes existants. Toutefois, si toutes les parties d'une application sont étroitement liées, l'introduction de modifications dans le système se révèle particulièrement ardue. Un changement dans une partie de l'application risque d'entraîner l'interruption d'une autre portion du système ou de se répercuter sur la totalité de la base de code. La propriété clé d'un microservice est la notion d'indépendance et d'évolutivité, ce qui implique que nous pouvons réécrire un microservice sans affecter les autres. En effet, il faut définir la modularité suivant le patron du changement. C'est-à-dire regrouper dans le même module les éléments qui changent en même temps.

2.2.3 Communication inter-processus dans l'architecture de micro-services

Dans une application monolithique, les composants s'appellent les uns les autres via des appels de méthode ou de fonction au niveau du langage. En revanche, les applications basées sur des microservices sont des systèmes distribués s'exécutant sur plusieurs machines. Chaque instance de service est généralement un processus. Par conséquent, comme le montre la figure ci-dessous, les services doivent interagir à l'aide d'un mécanisme de communication inter-processus (IPC).

FIG. 2.2 – *Communication inter-processus[3]*

2.2.3.1 Styles d'interaction

Avant de choisir un mécanisme IPC pour l'API du service, il est utile de considérer d'abord l'interaction entre le service et son client. Réfléchir d'abord à la méthode d'interaction vous aidera à vous concentrer sur vos besoins et à éviter de rester coincé dans les détails d'une technologie IPC particulière. Il existe de nombreux styles d'interactions client-service. Ils peuvent être classés selon deux dimensions.

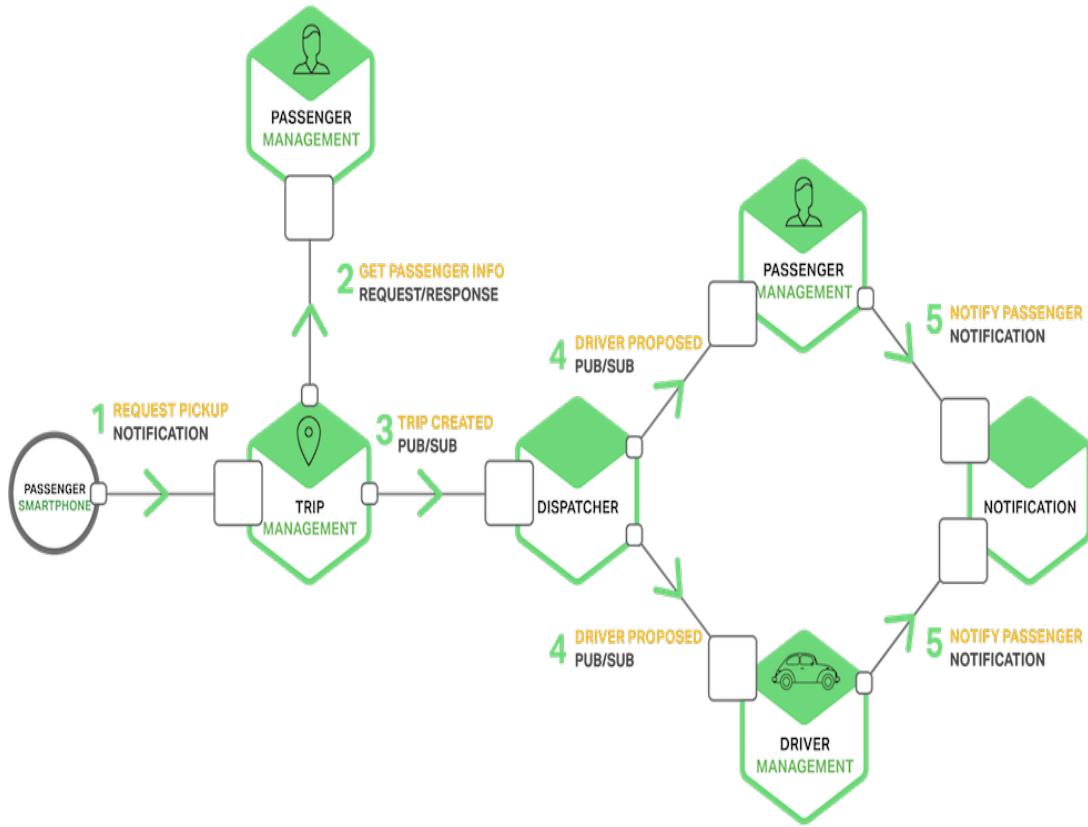
La première dimension est de savoir si l'interaction est un-à-un ou un-à-plusieurs:

- Un-à-un: Chaque demande client est traitée par exactement une instance de service.
- Un à plusieurs: Chaque demande est traitée par plusieurs instances de service.

La deuxième dimension est de savoir si l'interaction est synchrone ou asynchrone:

- Synchrone: Le client attend une réponse rapide du service et peut même bloquer pendant qu'il attend.
- Asynchrone: Le client ne bloque pas en attendant une réponse, et la réponse, le cas échéant, n'est pas nécessairement envoyée immédiatement.

La figure ci-dessous montre comment les services de l'application d'appel de taxi interagissent lorsque l'utilisateur demande un déplacement.

FIG. 2.3 – *Interaction entre les services*[3]

Dans l’architecture de microservices, les services utilisent une combinaison de notifications, de demande/réponse et de publication/abonnement. Par exemple, le smartphone du passager envoie une notification au service de gestion de voyage pour demander une prise en charge. Le service de gestion de voyage vérifie que le compte du passager est actif en utilisant la demande/réponse pour appeler le service passagers. Le service de gestion de voyage crée ensuite le voyage et utilise la publication/l’abonnement pour notifier d’autres services, y compris le répartiteur, qui localise un conducteur disponible. Maintenant que nous avons examiné les styles d’interaction, voyons comment définir les API.

2.2.4 Définition des API dans une architecture de microservices

L’API d’un service change invariablement au fil du temps. Dans une application monolithique, il est généralement simple de modifier l’API et de mettre à jour tous les appelants. Dans une application basée sur des microservices, c’est beaucoup plus difficile, même si tous les consommateurs de votre API sont d’autres services dans la même application. Vous ne pouvez généralement pas forcer tous les clients à se mettre à niveau en même temps que le service.

2.2.5 Découverte de services dans une architecture de microservices

Dans une architecture de microservices les services doivent se trouver et communiquer les uns avec les autres. C’est le service de découverte qui gère la façon d’enregistrement et de localisation des microservices déployés.

La découverte de services comprend trois composants: le fournisseur de services, le consommateur de services et le registre de services.

- Le fournisseur de services s'enregistre auprès du registre de services lorsqu'il entre dans le système et se désenregistre lorsqu'il quitte le système.
- Le consommateur de services obtient l'emplacement d'un fournisseur à partir du registre de services, puis le connecte au fournisseur de services.
- Le registre des services est une base de données qui contient les emplacements réseau des instances de service. Le registre des services doit être hautement disponible et à jour pour que les clients puissent accéder aux emplacements réseau obtenus à partir du registre des services. Un registre de services se compose d'un cluster de serveurs qui utilisent un protocole de réPLICATION pour maintenir la cohérence.

Pour envoyer une requête API à un service, le client ou la passerelle API doit connaître l'emplacement du service auquel il s'adresse. Étant donné que l'architecture de microservices est conçue pour prendre en charge plusieurs instances de chaque service, la découverte de services est étroitement liée à l'équilibrage de charge des demandes adressées aux instances de chaque microservice.

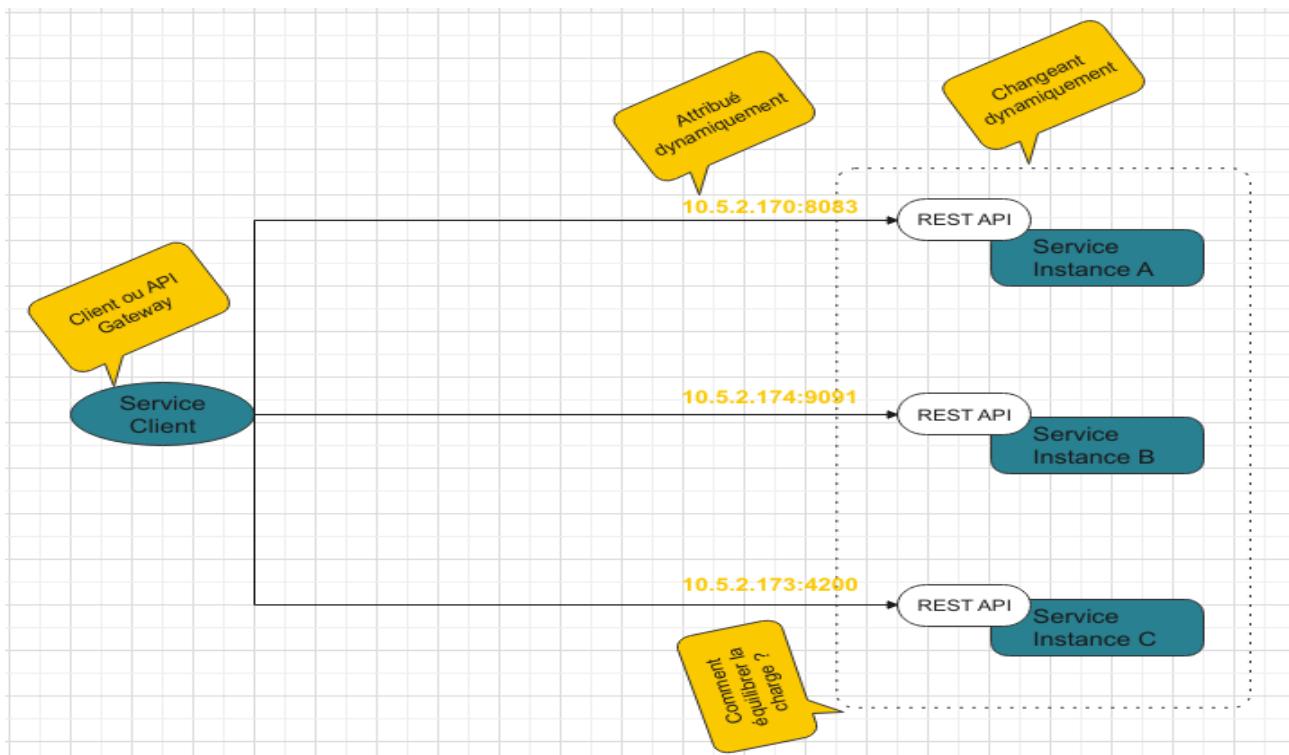
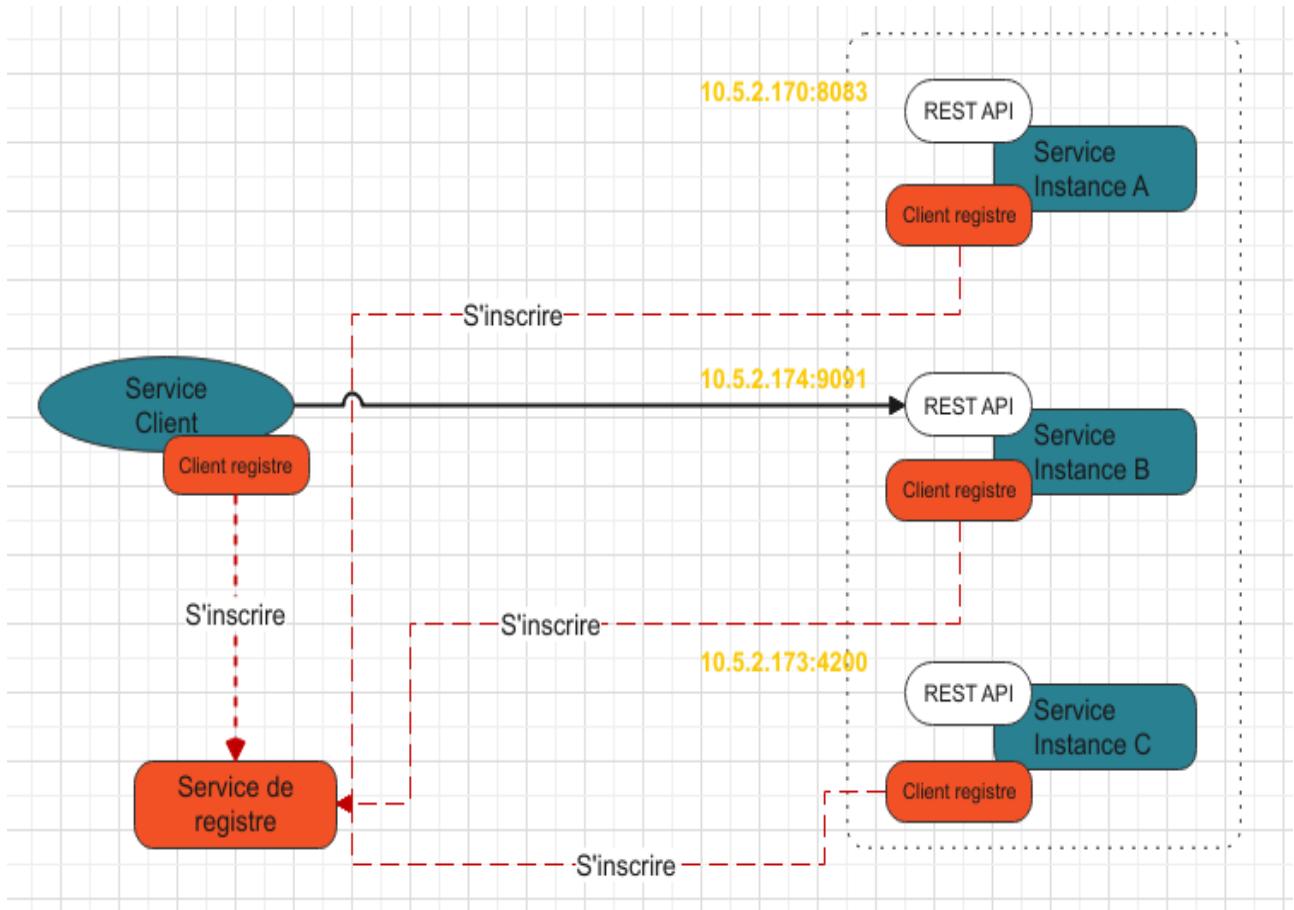


FIG. 2.4 – Découverte de services

Il existe deux modèles de découverte de services: côté client et côté serveur.

2.2.5.1 Découverte côté client

Avec la découverte côté client, le client demandeur ou la passerelle API est responsable de l'identification de l'emplacement de l'instance de service et du routage de la demande vers l'instance. Le client interroge d'abord le registre de service pour identifier l'emplacement des instances disponibles du service, puis détermine l'instance à utiliser. La logique d'équilibrage de charge peut être une simple méthode circulaire ou utiliser un système de pondération pour déterminer la meilleure instance à interroger à un moment donné.

FIG. 2.5 – *Découverte côté client*

2.2.5.2 Découverte côté serveur

Dans une découverte de service côté serveur, le client envoie une demande à un service via un équilibrEUR de charge. L'équilibrEUR de charge interroge le registre de service puis applique la logique d'équilibrage de charge pour acheminer chaque demande vers une instance de service disponible. À l'instar de l'enregistrement tiers, certains environnements de déploiement incluent cette fonctionnalité dans leurs offres de services, éliminant ainsi le besoin de configurer et de gérer d'autres composants. Dans ce cas, le client ou la passerelle API n'a besoin que de connaître l'emplacement du routeur.

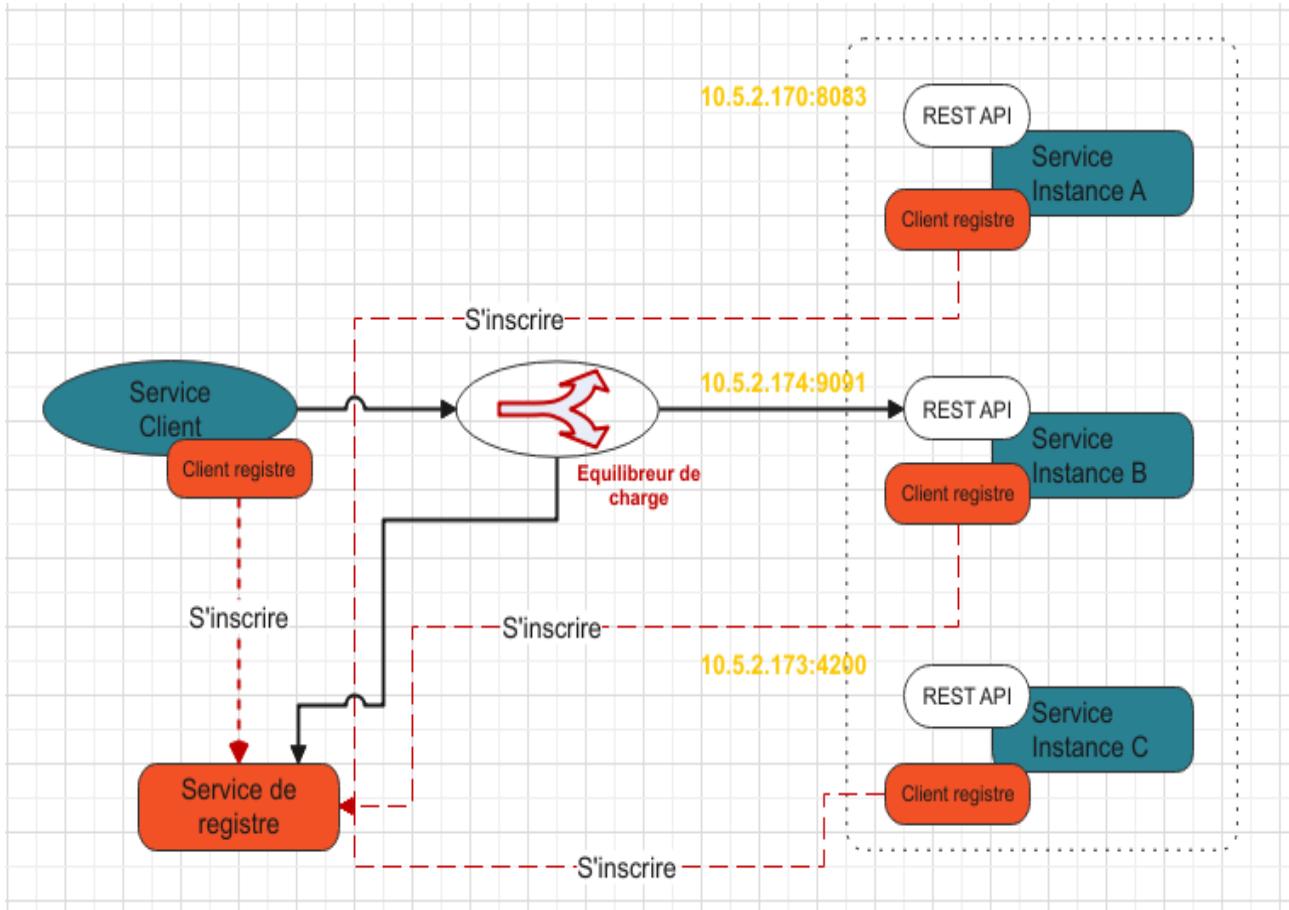


FIG. 2.6 – Découverte côté serveur

2.2.6 Architecture orientée événement

2.2.6.1 Définition

Event Driven Architecture (EDA) est un modèle d'architecture logicielle utilisé pour concevoir des applications. Dans un système événementiel, la capture, la communication, le traitement et la conservation des événements constituent la structure centrale de la solution. C'est ce qui distingue ce type de système du modèle traditionnel basé sur des requêtes. De nombreuses applications modernes sont conçues selon une architecture orientée événement. Vous pouvez utiliser n'importe quel langage de programmation pour créer des applications orientées événement, car l'architecture orientée événement est une méthode, pas un langage. L'architecture orientée événement permet un couplage faible. Par conséquent, il convient à l'architecture d'applications distribuées modernes.

2.2.6.2 Fonctionnement

Une architecture orientée événements implique des producteurs et des consommateurs d'événements. Un producteur d'événements détecte ou reconnaît un événement et le représente sous forme de message. Il ignore quels seront les consommateurs et les conséquences de chaque événement. Lorsqu'un événement a été détecté, il est transmis du producteur d'événements au consommateur via des canaux d'événement. Dans le canal d'événements, la plate-forme de traitement traitera ces événements de manière asynchrone. Les consommateurs d'événements doivent être

avertis lorsqu'un événement se produit. Ils peuvent traiter l'événement ou être seulement affectés par ce dernier. La plateforme de traitement des événements exécute la réponse adaptée à chaque événement et envoie l'activité en aval aux consommateurs concernés. Cette activité permet de visualiser le résultat d'un événement.

Apache Kafka est une plate-forme de diffusion de données distribuée largement utilisée pour le traitement des événements. Elle peut gérer la publication, le stockage et le traitement des flux d'événements en temps réel, ainsi que les abonnements à ces événements. Apache Kafka prend en charge de nombreux cas d'utilisation différents dans lesquels un débit élevé et une bonne évolutivité sont essentiels. En minimisant la nécessité d'une intégration point à point du partage de données dans certaines applications, la plate-forme peut réduire la latence à quelques millisecondes. Il existe d'autres types de middleware de gestionnaires d'événements qui peuvent être utilisés comme plates-formes de traitement d'événements.

2.2.7 Gestion des données dans une architecture de microservices

Une règle importante de l'architecture de microservices est que chaque microservice doit avoir des données et une logique dans son domaine. Au même titre qu'une application complète, chaque microservice doit posséder sa logique et ses données dans un cycle de vie autonome, avec un déploiement indépendant par microservice. Les microservices ont des besoins de gestion de données qui ne sont pas comparables à ceux des autres architectures logicielles. Pour réussir, vous aurez besoin d'un ensemble approprié de connaissances, de mécanismes et de principes de conception.

2.2.7.1 Comment créer des requêtes qui récupèrent des données à partir de plusieurs microservices ?

Ici le défi est de savoir comment implémenter des requêtes qui récupèrent des données à partir de plusieurs microservices, tout en évitant les communications bavardes avec les microservices à partir d'applications clientes distantes. Pour une agrégation simple de données à partir de plusieurs microservices possédant différentes bases de données, l'approche recommandée est un microservice d'agrégation appelé passerelle API. Cependant, vous devez être prudent lors de la mise en œuvre de ce modèle, car il peut être un point d'étranglement dans votre système, et il peut violer le principe de l'autonomie des microservices. Pour atténuer cette possibilité, vous pouvez avoir plusieurs passerelles API à granularité fine, chacune se concentrant sur un domaine d'activité du système.

2.2.7.2 Comment assurer la cohérence entre plusieurs microservices ?

Dans une architecture de microservices, les données appartenant à chaque microservice sont privées pour ce microservice et ne sont accessibles qu'à l'aide de son API. Par conséquent, l'un des défis actuels est de savoir comment mettre en œuvre des processus métier de bout en bout tout en maintenant la cohérence entre plusieurs microservices. La solution à ce problème est d'utiliser une cohérence éventuelle entre les microservices articulés par une communication événementielle et un système de publication et d'abonnement. Lors de l'utilisation d'une communication asynchrone pilotée par les événements, lorsqu'un microservice a besoin de l'attention d'autres microservices de son domaine (par exemple, une modification de prix dans un catalogue de produits d'un microservice), le microservice publie un événement d'intégration. D'autres microservices s'abonnent à des événements afin qu'ils puissent être reçus de manière

asynchrone. Lorsque cela se produit, les destinataires peuvent mettre à jour leurs propres entités de domaine, ce qui peut conduire à la publication de plusieurs événements d'intégration. Ce système de publication/abonnement est réalisé en utilisant une implémentation d'un bus d'événements. Le bus d'événements peut être conçu comme une abstraction ou une interface, avec l'API requises pour s'abonner ou se désabonner à des événements et pour publier des événements.

2.2.8 Gestion de la sécurité dans une architecture de microservices

Lorsqu'elle est correctement assemblée, une architecture de microservices permet l'inter-opération des applications entre divers services, parfois hébergés sur des plateformes différentes. Pour les microservices, la sécurité doit être en tête des préoccupations, car il n'existe aucun moyen de limiter le champ d'action des utilisateurs comme dans une application monolithique. Au lieu d'autoriser simplement l'accès direct à une application reposant sur des microservices, les équipes de développement doivent également sécuriser et gérer l'accès à chacun des services auxquels les utilisateurs ont affaire. Mais ceux-ci ne veulent pas avoir à authentifier chaque fois qu'ils passent d'un service à un autre. Les nouvelles approches en termes de sécurité optimisent la protection au sein d'une architecture distribuée, sans gêner l'expérience des utilisateurs. Dans la suite, nous présenterons les meilleures pratiques à appliquer pour sécuriser les microservices.

- **API Gateway(Passerelle API)** : Les API sont l'un des domaines les plus vulnérables des modèles d'architecture de microservices. Lors de la mise en place des meilleures pratiques de sécurité des microservices, la création de passerelle API est essentielle. Ceux-ci agissent comme un point d'entrée unique qui gère les demandes externes, aidant à bloquer l'accès direct d'un client aux microservices et empêchant les attaques potentielles d'acteurs malveillants.
- **Utiliser des jetons** : définissez des identités vérifiées, puis contrôlez l'accès aux services et aux ressources à l'aide des jetons qui leur sont attribués.
- **Utiliser des quotas et la limitation de requêtes** : définissez des quotas quant à la fréquence d'appels de votre API et suivez son utilisation dans l'historique. Une augmentation du nombre d'appels peut indiquer que l'API est utilisée de manière abusive. Il peut également s'agir d'une erreur de programmation, par exemple une boucle infinie qui ne cesse d'appeler l'API. Établissez des règles de limitation de requêtes pour protéger vos API des pics de trafic et des attaques par déni de service.

2.2.9 Bilan sur l'architecture de microservices

2.2.9.1 Avantages de l'architectures de microservices

- **Mise sur le marché plus rapide** : Comme les cycles de développement sont plus courts, l'architecture de microservices permet des déploiements et mises à jour plus agiles.
- **Haute évolutivité** : À mesure que la demande pour certains services augmente, vous pouvez étendre les déploiements sur plusieurs serveurs et infrastructures pour répondre à vos besoins.
- **Résilience** : Lorsqu'ils sont développés correctement, ces services indépendants n'ont aucun impact les uns sur les autres. Cela signifie que, lorsqu'un élément tombe en panne, l'ensemble de l'application ne cesse pas de fonctionner comme c'est le cas avec le modèle monolithique.

- **La réutilisation des microservices:** Dans le cas d'un service ne faisant que rendre disponible une API web, il est facile pour un autre service de la consommer, et ce quels que soient le langage, la plateforme, ou le système d'exploitation utilisé
- **Hétérogénéité de la technologie:** Avec un système composé de plusieurs services collaborant, nous pouvons décider d'utiliser différentes technologies à l'intérieur de chacun. Cela nous permet de choisir le bon outil pour chaque travail, plutôt que d'avoir à choisir une approche standardisée et standardisée, qui finit souvent par être le plus petit dénominateur commun.

2.2.9.2 Inconvénients de l'architecture de microservices

Bien que l'architecture des microservices ait ses avantages, elle reçoit encore quelques critiques et présente certains inconvénients. Nous pouvons lister les plus récurrents:

- **Plus de services équivaut à plus de ressources :** Plusieurs bases de données et la gestion des transactions peuvent être pénibles.
- **La communication entre les services est complexe :** Puisque tout est désormais un service indépendant, vous devez gérer avec soin les demandes transitant entre vos modules. Dans un tel scénario, les développeurs peuvent être obligés d'écrire du code supplémentaire pour éviter toute interruption. Au fil du temps, des complications surviendront lorsque les appels distants subissent une latence.
- **Les problèmes de débogage peuvent être plus difficiles :** Chaque service a son propre ensemble de journaux à parcourir.
- **Challengers de déploiement :** Le produit peut nécessiter une coordination entre plusieurs services, ce qui peut ne pas être aussi simple que de déployer un WAR dans un conteneur.

2.3 Application cloud native

2.3.1 Définition

Les apparences sont trompeuses. À première vue, l'expression « cloud native » pourrait signifier qu'une application soit déployée et qu'elle fonctionne dans le cloud. En réalité, cette expression fait référence à la manière dont des applications sont conçues, déployées et gérées. L'idée est d'exploiter les avantages du modèle dit de « cloud computing » pour accroître la vitesse, la flexibilité et la qualité du développement des applications tout en réduisant les risques. L'objectif global est d'améliorer la vitesse, la marge et, enfin, l'évolutivité.

- **Vitesse :** Les entreprises de toutes tailles voient désormais un avantage stratégique à pouvoir évoluer rapidement et mettre rapidement leurs idées sur le marché. Ce que nous voulons dire, c'est raccourcir le délai de mise en production d'une idée de plusieurs mois à quelques jours, voire quelques heures. Pour atteindre cet objectif, il faut notamment un changement culturel au sein de l'entreprise. Fondamentalement, une stratégie cloud native consiste à gérer les risques techniques. Dans le passé, notre façon habituelle d'éviter le danger consistait à nous déplacer lentement et prudemment. L'approche cloud native consiste à évoluer rapidement en prenant de petites actions réversibles et à faible risque.

- Marge : Dans le nouveau monde de l'infrastructure cloud, l'objectif stratégique est de ne payer pour des ressources supplémentaires qu'en cas de besoin, à mesure que de nouveaux clients se connectent. Les dépenses passent des CAPEX¹ initiaux aux OPEX².
- Évolutivité : À mesure que les entreprises se développent, il devient nécessaire de prendre en charge plus d'utilisateurs, dans plus d'emplacements, avec une plus large gamme d'appareils, tout en maintenant la réactivité, en gérant les coûts et en évitant les chutes.

Architecturée éventuellement sous forme de microservices, l'application cloud native peut être intégrée et déployée par de petites équipes de développement dédiées à une plate-forme, et permet de créer et livrer rapidement des fonctionnalités en phase avec les besoins des clients.

2.3.2 Avantages

Une application cloud native peut bénéficier de toutes les capacités du Cloud et des bénéfices associés. Par exemple, les services du cloud permettent plus facilement de concevoir des applications hautement disponibles : en cas de panne d'un équipement du DataCenter, un autre équipement de ce même DataCenter peut prendre le relais automatiquement ou bien la charge peut être immédiatement transférée vers un DataCenter d'une autre région. Et aussi en cas de pic d'utilisation, une application cloud native peut bénéficier de l'élasticité offerte par le cloud et utiliser davantage de ressources qui pourront être désactivées dès la fin du pic et un retour à la normale de l'utilisation.

2.4 Architecture d'une application en microservices

La figure ci-dessous présente les composants de base pour mettre en place une architecture de microservices.

1. CAPEX représente les dépenses pour des achats de biens ou de services qui seront utilisés pour améliorer les performances d'une entreprise à l'avenir.

2. OPEX représente les coûts qu'une entreprise engage pour la gestion de ses opérations quotidiennes.

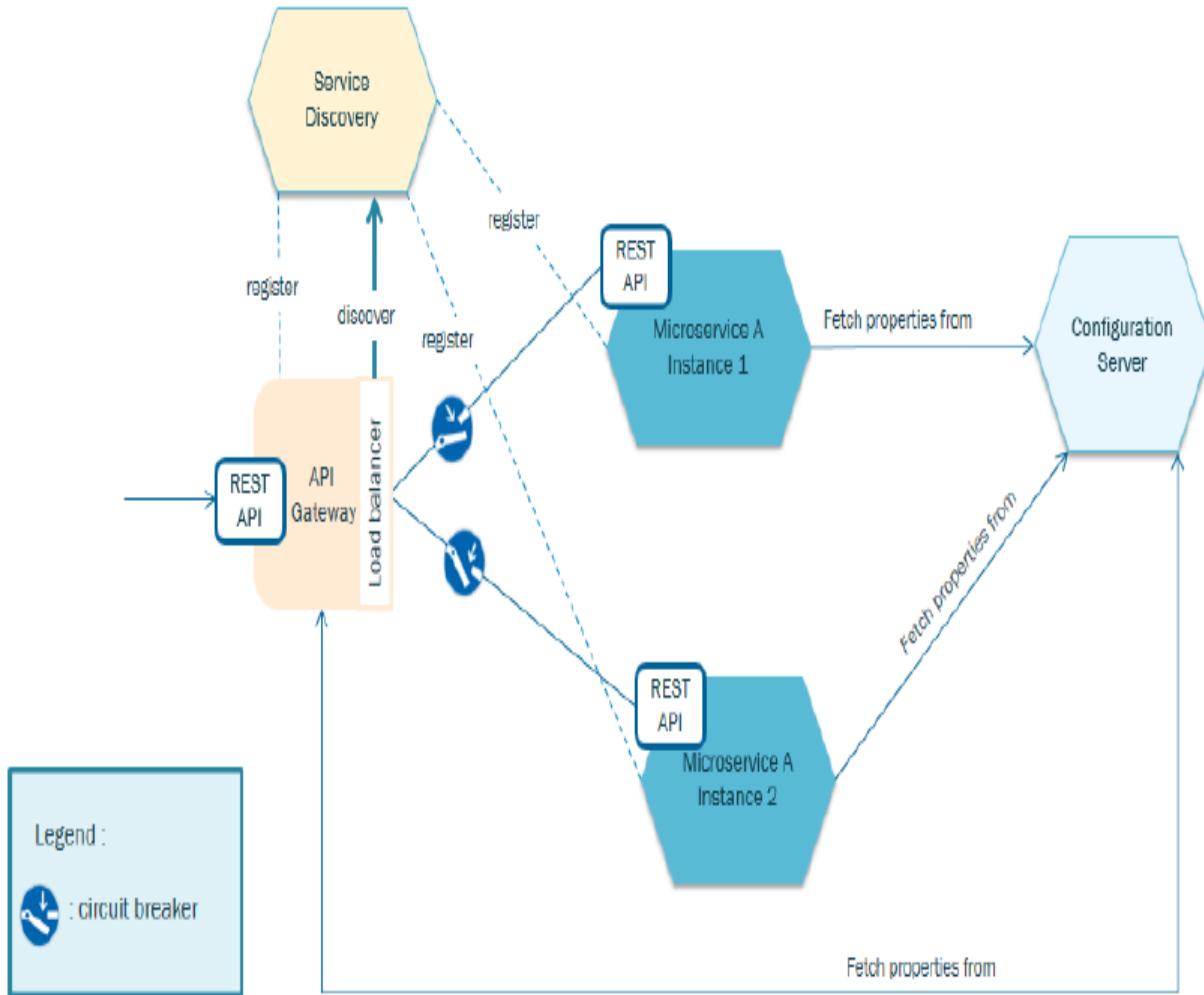


FIG. 2.7 – Composants d'une architecture de microservices

Elle se compose généralement des éléments suivants :

- **Un service de découverte** : Dans une architecture de microservices les services doivent se trouver et communiquer les uns avec les autres. C'est le service de découverte qui gère la façon d'enregistrement et de localisation des microservices déployés. L'utilité d'un tel service est de permettre de gérer la montée en charge des applications en donnant la possibilité aux clients de consulter le registre pour trouver les instances disponibles.
- **Un serveur de configuration** encore appelé “Config Server” qui permet de centraliser les fichiers des configurations de chaque microservice dans un simple dépôt Git. Ceci permet d'avoir une configuration partagée et évolutive indépendamment des applications. Au démarrage, chaque microservice récupère ses propriétés et sa configuration auprès du serveur de configuration.
- **Une passerelle** appelée encore “API Gateway”, qui présente le point d'entrée au système. Elle encapsule l'architecture du système interne et fournit des API adaptées pour chaque type de client. l'API Gateway encapsule un composant très important qui est l'équilibrer de charge, appelé “Load Balancer”. Il gère le routage et la répartition de la charge entre les instances des microservices disponibles. Pour avoir la liste des instances disponibles, le load balancer consulte le serveur d'enregistrement.
- **Un disjoncteur de circuit** “circuit breaker”, il nous aide à construire un système résilient et tolérant aux pannes. Il garantit une caractéristique très importante qui est la

conception pour l'échec, que nous avons déjà évoqué dans ce chapitre. Le disjoncteur de circuit permet d'éviter l'échec de tout le système en cas de défaillance d'un microservice.

Chapitre 3

Analyse et conception de la solution

Ce chapitre présente la méthodologie que nous avons adoptée pour la conception et réalisation de notre application. Au cours de ce chapitre, nous identifions les acteurs de notre système. Nous spécifions les besoins fonctionnels et non fonctionnels. Enfin nous faisons une modélisation globale de l'application.

3.1 Méthodologie et approche de développement

Pour le choix de la méthodologie de développement, il faut prendre en considération les spécificités de l'architecture utilisée et les intervenants du projet. En analysant ces facteurs, nous avons découpé le projet en cycles de développements itératifs et incrémentaux. A chaque itération, une partie du projet sera traitée. A la fin de chaque itération un microservice autonome va être livré. Pour se faire, nous avons choisi d'adopter la méthode Agile SCRUM pour le développement et l'approche DevOps pour la mise en production.

3.1.1 Méthodologie SCRUM

Scrum est la méthodologie la plus utilisée parmi les méthodes Agiles existantes. Le terme Scrum (qui signifie mêlée) apparaît pour la première fois en 1986 dans une publication de Hirotaka Takeuchi et Ikujiro Nonaka qui décrit une nouvelle approche plus rapide et flexible pour le développement de nouveaux produits. Elle s'appuie sur des tâches claires, classées par ordre de priorité, puis réalisées lors de phases opérationnelles récurrentes. La méthode Scrum s'appuie sur des Sprints qui sont des espaces temps assez courts, généralement entre 2 et 4 semaines. À la fin de chaque sprint, l'équipe présente ce qu'elle a ajouté au produit. L'expertise et les retours de l'équipe sont essentiels pour améliorer progressivement le projet en cours autant que sa productivité. Scrum ne dit pas comment réussir son logiciel, comment surmonter les obstacles, comment développer, comment spécifier, etc. Il se contente d'offrir un cadre de gestion de projet Agile: des rôles, un rythme itératif, des réunions précises et limitées dans le temps, des artefacts (product backlog, sprint backlog, graphique d'avancement) et des règles du jeu. Au sein de ce cadre méthodologique de gestion de projet, les acteurs ajustent empiriquement, au fil des itérations, leur propre méthode en fonction de leur contexte. On peut qualifier Scrum de simple, pragmatique, transparent et empirique. Scrum définit seulement 3 rôles :

- **Le Product Owner** qui porte la vision du produit à réaliser et travaille en interaction avec l'équipe de développement. Il s'agit généralement d'un expert du domaine métier du projet. Il définit les caractéristiques du produit à savoir les différentes fonctionnalités et

leurs valeurs métier. Le product owner décide de la date de livraison et peut accepter ou rejeter les résultats; Ajuste les fonctions et les priorités pour chaque sprint.

- **L'équipe de Développement** qui est chargée de transformer les besoins exprimés par le Product Owner en fonctionnalités utilisables. Elle est pluridisciplinaire et peut donc encapsuler d'autres rôles tels que développeur, architecte logiciel, DBA, analyste fonctionnel, graphiste/ergonome, ingénieur système. Elle doit livrer à chaque fin d'itération une nouvelle version de l'application enrichie de nouvelles fonctionnalités et respectant le niveau de qualité nécessaire pour être livré.
- **Le Scrum Master** autrement appelé maître de mêlée, a pour mission d'huiler les rouages et d'assister au mieux chacun dans ses tâches. Il veille au respect de la méthodologie Scrum, à la bonne communication et aide à surmonter les difficultés. Il a donc un rôle de coach à la fois auprès du Product Owner et auprès de l'équipe de développement. Il doit donc faire preuve de pédagogie. Il est également chargé de s'assurer que l'équipe de développement est pleinement productive. Généralement le candidat tout trouvé au rôle de Scrum Master est le chef de projet. Celui-ci devra cependant renoncer au style de management « commander et contrôler » pour adopter un mode de management participatif.

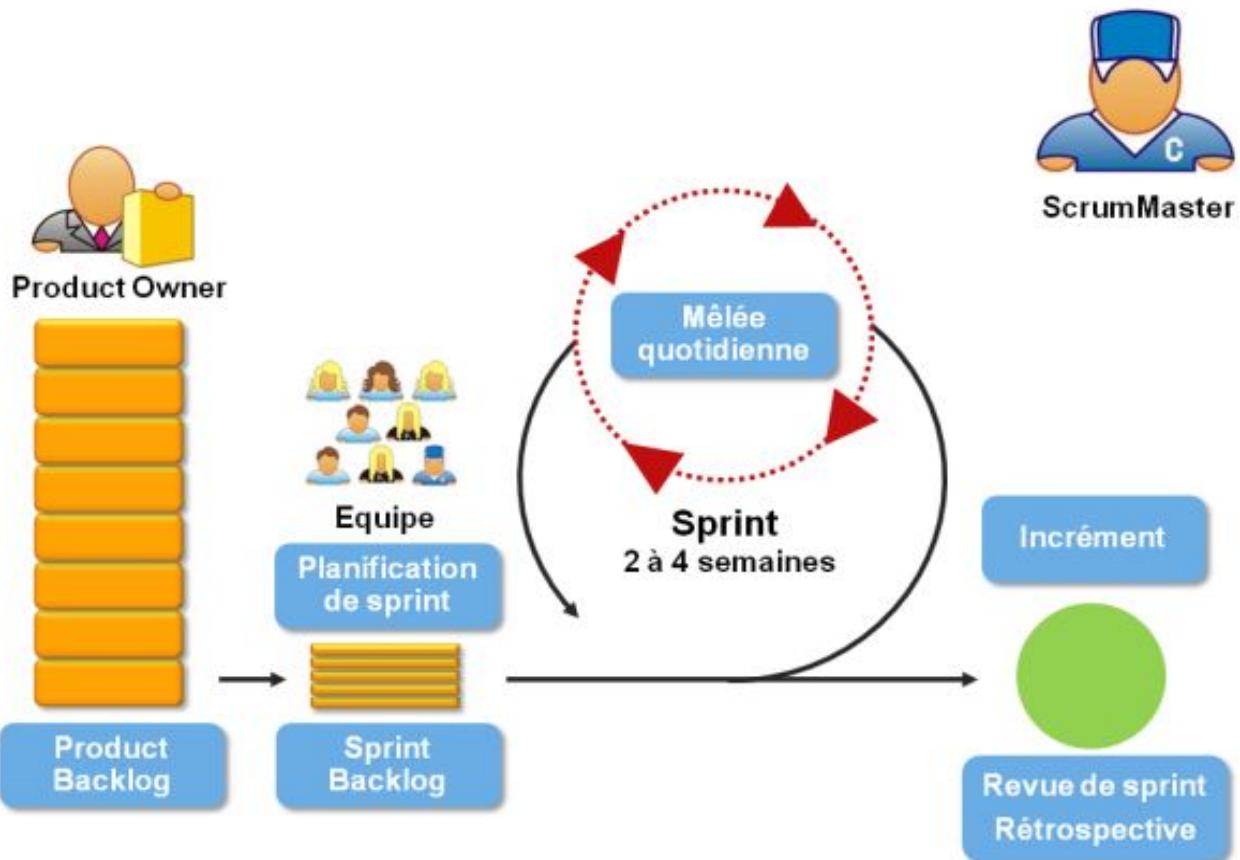


FIG. 3.1 – Pratique SCRUM: vue d'ensemble

3.1.2 L'approche DevOps

DevOps (un mot porte-manteau entre « Developpement » et « Operations ») est un ensemble de pratiques mettant en avant la communication, la collaboration et l'intégration entre les

développeurs logiciels (Dev) et les exploitants (Ops) d'une même plateforme. C'est une approche qui permet aux équipes de collaborer et d'accélérer le processus entre le développement et le déploiement, tout en continuant d'améliorer la qualité, la sécurité et la fiabilité. DevOps vise à créer une culture et un environnement dans lesquels la conception, les tests et la diffusion de logiciels peuvent être réalisés rapidement, fréquemment et efficacement. DevOps n'est pas seulement une méthodologie, c'est une véritable philosophie de travail.

Pour que l'approche DevOps soit efficace, vous devez commencer par mettre en place une culture et un état d'esprit de collaboration entre les développeurs et l'équipe chargée des opérations. En faisant preuve d'une organisation sans limite, mettez en place un environnement intégré où vous pouvez tester plusieurs fois le code logiciel et l'améliorer. Ensuite, mettez en œuvre un calendrier pour déployer le logiciel amélioré en continu. La qualité et la rapidité de livraison des nouveaux produits et services améliorent l'expérience utilisateur et la satisfaction de vos clients. À l'aide d'outils dotés d'algorithmes d'apprentissage automatique intégrés pour assurer une surveillance et une réactivité continues, les tâches (flux de travail) sont automatiquement réalisées sans intervention humaine.

Lorsque vous recueillez les analyses et les commentaires des clients, vous pouvez rapidement adapter ces informations à votre planification et au développement des futurs produits. Cela vous ramène de nouveau au début du cycle DevOps. Cependant, le développement collaboratif bénéficie cette fois des connaissances acquises et validées auprès des clients, et des débuts de l'optimisation.

En continuant à suivre la méthodologie DevOps, les sociétés se démarquent grâce à un écosystème bien adapté, comprenant des éléments interdépendants, des meilleures pratiques pour rationaliser le développement, ainsi que des normes établies pour maintenir un niveau élevé de qualité.

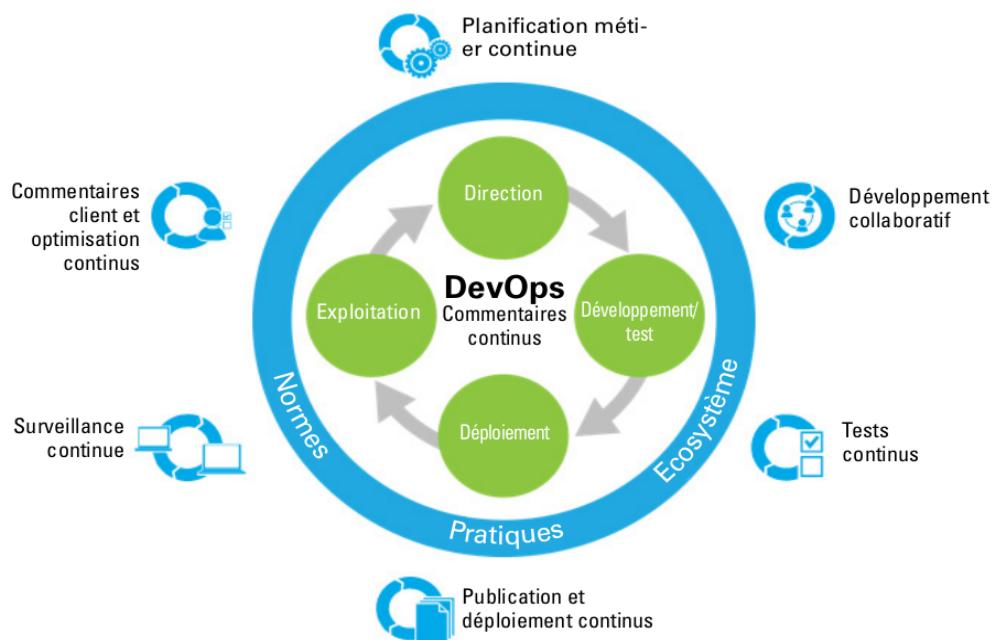


FIG. 3.2 – Architecture de référence DevOPs[6]

3.2 Analyse et spécification des besoins

3.2.1 Présentation de l'application G-School

G-School est une application cloud native qui vise à dématérialiser la gestion scolaire des établissements publics et privés. C'est une solution qui aide les gestionnaires de l'école à gérer de façon optimisée les opérations courantes telles que la gestion des inscriptions, la perception des frais de scolarité, la gestion des examens et notes d'examens, les absences, les devoirs, les notifications, les emplois du temps... et cela grâce à plusieurs services interconnectés. G-School propose une interface intuitive capable de gérer l'ensemble des processus de la vie scolaire, quels que soient le nombre des élèves, des enseignants et la pluralité des niveaux et des disciplines. Cette plateforme propose un ensemble de fonctionnalités pour une gestion complète de la vie scolaire.

3.2.2 Besoins fonctionnels

La spécification des besoins fonctionnels permet de cadrer le travail et de définir les différentes tâches à réaliser. Pour dématérialiser la gestion scolaire, l'application G-School doit satisfaire les fonctionnalités suivantes :

- Dossier scolaire: centraliser l'ensemble des informations scolaires des élèves dans un dossier consultable uniquement par les personnes habilitées.
- Suivi des absences et des retards: les absences, retards, motifs et justificatifs sont enregistrés pour suivre l'assiduité des élèves plus facilement. Une notification apparaît dans le tableau de bord de l'administrateur pour signaler toute absence de 2 journées dans la semaine. Une notification par sms est aussi envoyée au tuteur de l'élève lors d'une absence.
- Fiches élèves: en un seul clic, l'enseignant visualise l'ensemble des informations de ses élèves : date de naissance, nom, prénom, etc.
- Emplois du temps: suivi et diffusion aux personnes concernées des emplois du temps, des réunions, des conseils de classe, etc. Il vous est également possible d'utiliser l'aide à la conception des emplois du temps en indiquant les différentes contraintes (enseignants sur plusieurs classes, horaires de pause, etc.).
- Gestion des matières: ajoutez ou personnalisez les matières selon les spécialités de votre établissement.
- Gestion des évaluations et notes: les professeurs peuvent remplir les notes des élèves et à la fin du trimestre, le système génère automatiquement les bulletins de notes.
- Gestion des classes: permet la gestion des classes, affectation des matières, des professeurs et choix du niveaux de la classe
- Tableau de bord: visualisez en temps réel et en un clin d'œil tout ce qui se passe dans l'établissement et suivez facilement les évènements : absences, cours, événement, tâche, etc.
- Gestion des utilisateurs: permet de gérer tous les utilisateurs du système avec leurs prérogatives.

3.2.3 Besoins non fonctionnels

Les besoins non fonctionnels de l'application G-School sont :

- Tolérance aux pannes: G-School doit être capable de fonctionner même s'il existe des

microservices en défaillance.

- Mise à l'échelle: G-School doit permettre une mise à l'échelle ciblée, c'est-à-dire nous n'avons pas besoin de mettre à l'échelle toute l'application, mais seulement les microservices qui ont besoin d'y être.
- Maintenabilité: L'architecture doit permettre l'évolution et assurer l'extensibilité de l'application.
- Temps de réponse: G-School doit avoir un temps minimal pour rendre une vue dans le navigateur.
- Disponibilité: L'application doit être disponible 24heures/24 et 7jours/7.
- Sécurité: Seuls les utilisateurs autorisés ont accès à l'application.

3.2.4 Identification des acteurs

Pour ce système de gestion scolaire, nous avons identifiés les acteurs suivants:

1. Directeur d'école
2. Professeur
3. Secrétaire
4. Elève
5. Parent d'élève

3.2.5 Modélisation globale du diagramme de classe

Avant de découper notre application en microservices, nous allons faire en premier temps une modélisation globale du système. Le diagramme de classe de l'application G-School est représenté par la figure ci-dessous.

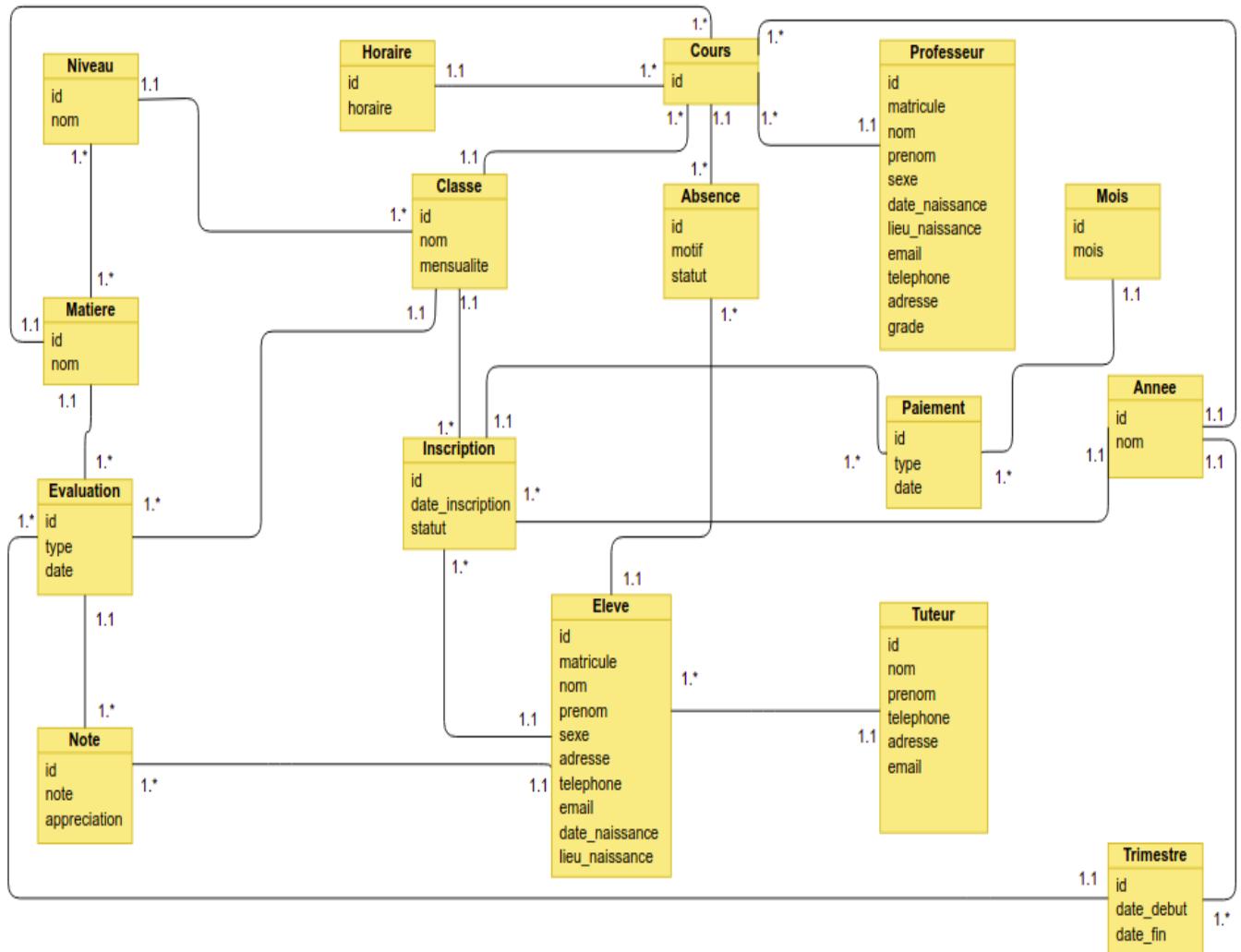


FIG. 3.3 – Diagramme de classe de G-School

3.2.6 Division de l'application G-School en microservices

Pour diviser notre application en microservices, nous définissons d'abord nos contextes bornés. Chaque contexte identifié représente un microservice à développer. Dans la figureci-dessous, nous présentons de manière simplifiée notre carte de mapping où chaque contexte est représenté par une couleur différente.

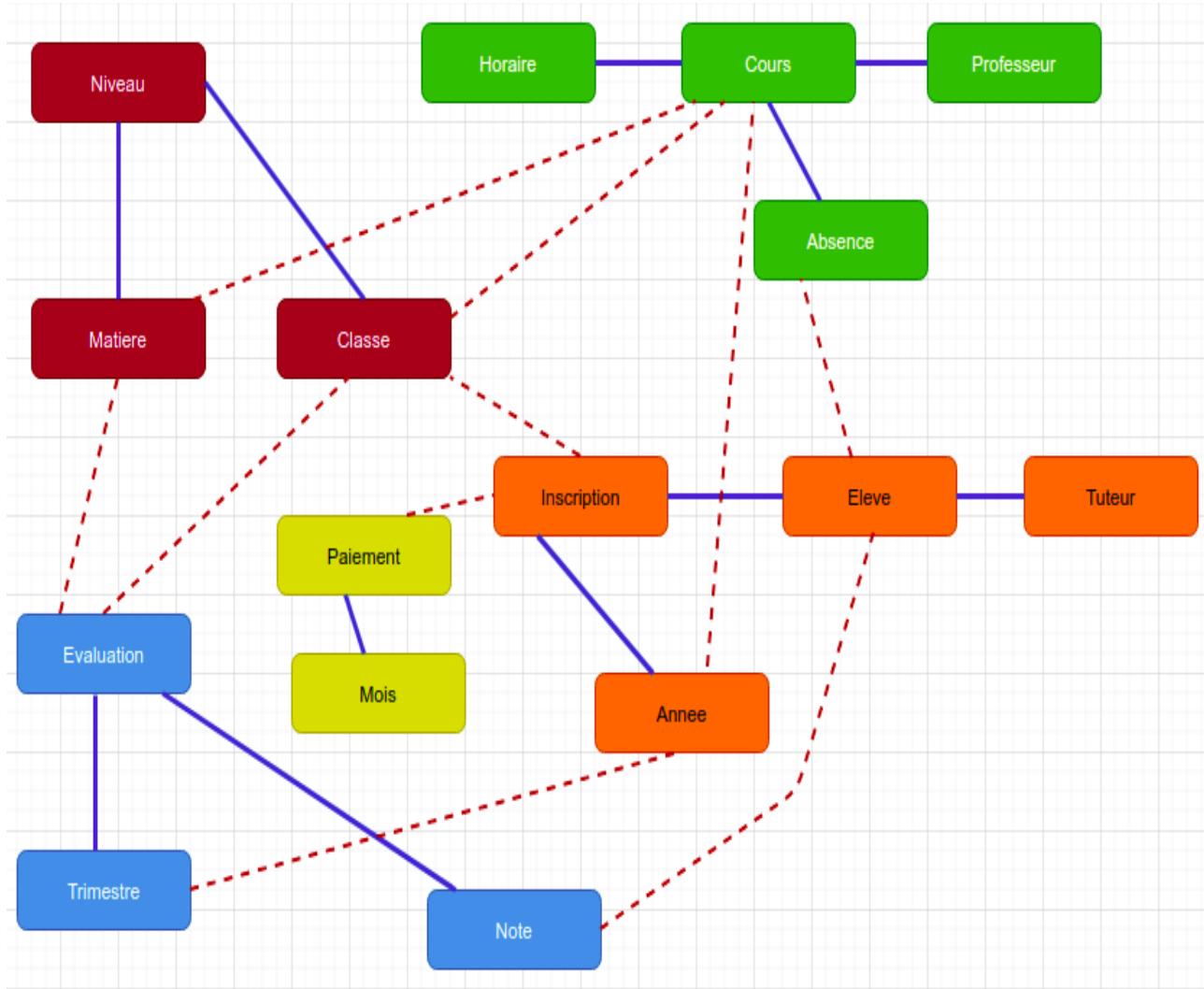


FIG. 3.4 – Carte de mapping du diagramme de classe

Ce schéma a été fait à l'aide du pattern bounded context¹. L'utilisation de ce pattern bounded context nous a permis d'identifier cinq(5) microservices plus le microservice de la gestion des notifications et celui de la gestion des utilisateurs et leurs priviléges.

Ces cinq(5) microservices sont:

1. Microservice inscription-service
2. Microservice cours-service
3. Microservice evaluation-service
4. Microservice classe-service
5. Microservice paiement-service

1. Bounded context est simplement la limite au sein d'un domaine où un modèle de domaine particulier s'applique

Chapitre 4

Développement des microservices

Dans ce chapitre, nous ferons d'abord le choix des technologies à utiliser pour la réalisation de l'application. Ensuite nous ferons la conception et réalisation de chaque microservice de notre application.

4.1 Choix technologiques

Dans une architecture microservice, chaque microservice peut être développé avec une technologie différente. Ici il ne s'agit pas de présenter les technologies à utiliser dans chaque microservice mais nous présentons les technologies globales que nous allons utiliser pour réaliser notre projet.

4.1.1 Choix des frameworks de back-end

La liste est longue mais nous allons juste exposer les frameworks en vogue.

1. Quarkus



FIG. 4.1 – Logo Quarkus

C'est un framework natif pour le cloud, conçu en premier par Red Hat pour l'écriture d'applications Java. Quarkus a été imaginé pour permettre le développement d'applications Java dites cloud-natives, ou “kubernetes native” pour reprendre leurs termes. L'objectif de Quarkus est de faire de Java une plate-forme leader dans Kubernetes et les environnements sans serveur tout en offrant aux développeurs un modèle de programmation réactif et impératif unifié pour répondre de manière optimale à un plus large éventail d'architectures d'applications distribuées.

2. Spring boot



FIG. 4.2 – Logo Spring boot

C'est un framework Java populaire pour l'écriture de microservices. Il fournit divers projets d'extension sous Spring Cloud pour créer des microservices à pile complète. Spring Boot permet de créer des systèmes à grande échelle en démarrant une architecture simple à partir d'un certain nombre de composants collaboratifs. Il peut être utilisé pour construire un système à petite ou grande échelle. Spring boot est très facile à intégrer à d'autres frameworks populaires également en raison de l'inversion de contrôle. Spring dispose de divers modules pour s'intégrer facilement aux bases de données populaires. Il fournit diverses fonctionnalités pour gérer les pannes dans le système distribué.

3. Vert.x



FIG. 4.3 – Logo *Vert.x*

C'est une boîte à outils de développement logiciel open source, réactive et polyglotte des développeurs d'Eclipse. La programmation réactive est un paradigme de programmation, associé à des flux asynchrones, qui répondent à tout changement ou événement. De même, Vert.x utilise un bus d'événements, pour communiquer avec différentes parties de l'application et transmet les événements, de manière asynchrone, aux gestionnaires lorsqu'ils sont disponibles. Nous l'appelons polyglotte en raison de sa prise en charge de plusieurs langages JVM et non JVM tels que Java, Groovy, Ruby, Python et JavaScript.

4. Moleculer



FIG. 4.4 – Logo *Moleculer*

C'est un framework de microservices intéressant. Comme NodeJS devient populaire, ce framework est préférable si vous êtes un développeur JavaScript. Moleculer est un framework de microservices rapide, moderne et puissant pour NodeJS. Il vous aide à créer des services efficaces, fiables et évolutifs.

Quelques caractéristiques principales :

- Prise en charge de l'architecture événementielle avec équilibrage
- Registre de services intégré et découverte de services dynamiques
- Requêtes et événements à charge équilibrée
- De nombreuses fonctionnalités de tolérance aux pannes
- Solution de mise en cache intégrée

5. Micronaut



FIG. 4.5 – Logo *Micronaut*

C'est un framework de microservices full stack moderne, basé sur JVM, conçu pour créer des applications de microservices modulaires et facilement testables.

Micronaut vise à fournir tous les outils nécessaires pour créer des applications de microservices complètes, notamment:

- Injection de dépendances et inversion de contrôle (IoC)
- Valeurs par défaut sensibles et configuration automatique
- Configuration et partage de configuration
- Découverte de service
- Routage HTTP
- Client HTTP avec équilibrage de charge côté client

	Quarkus	Spring boot	Vert.x	Moleculer	Micronaut
Langage utilisé	Java	Java	Polyglotte	Node.js	Java
Portabilité	Oui	Oui	Oui	Oui	Oui
Licence	Apache Software Licence 2.0	Apache Software Licence 2.0	Apache Software Licence 2.0	Licence MIT	Apache Software Licence 2.0

FIG. 4.6 – Tableau comparatif des frameworks Back-end

Pour ce projet, notre choix est porté sur spring avec son environnement spring cloud qui fournit des implémentations DiscoveryClient pour les registres populaires tels que Eureka, Consul, Zookeeper et même le système intégré de Kubernetes.

NB: Ce choix concerne seulement la réalisation du service de découverte, du serveur de configuration et la passerelle. Chaque microservice sera développé avec la technologie qui convient le mieux.

4.1.2 Choix des frameworks de front-end

1. VueJs



FIG. 4.7 – Logo VueJs

C'est un framework JavaScript utilisé pour créer des interfaces utilisateur pour les applications Web. Il a été créé par Evan You en 2014 et convient le mieux aux applications monopages, aux composants asynchrones, aux prototypes et au rendu côté serveur. Si vous êtes déjà familiarisé avec JavaScript, vous pouvez facilement utiliser Vue.js, car vous n'avez pas besoin de connaître des tonnes de JSX ou ES2016 pour commencer. La dernière version de Vue comprend l'API de composition, la modification globale de l'API de montage et des fragments. Certaines entreprises notables qui utilisent Vue.js sont Facebook pour leur marketing Newsfeed, le produit Portfolio d'Adobe et l'interface de Grammarly.

2. Angular



FIG. 4.8 – Logo Angular

C'est un framework populaire. Créé par Misko Hevery et Adam Abrono en 2009 en tant que projet parallèle, il est actuellement maintenu par Google. Sa version actuelle, publiée le 11 novembre 2020, s'appelle Angular 11 et est entièrement construite avec Typescript. Le framework propose un chargement différentiel, une syntaxe de chargement différé et des API de générateur et d'espace de travail. Avec Angular, vous pouvez utiliser du HTML pour vos modèles et lier le HTML pour exprimer clairement vos composants. Il vous aide également à réduire la quantité de code que vous devez écrire avec sa liaison de données et son injection de dépendances, le tout dans le navigateur. Certaines entreprises notables avec Angular dans leur pile technologique sont Google, Udemy et Amazon.

3. React



FIG. 4.9 – Logo React

C'est une bibliothèque frontale très populaire créée par un ingénieur logiciel Facebook, Jordan Walke en 2011. Elle est utilisée pour créer des interfaces utilisateur dynamiques et est très appréciée par les développeurs JavaScript. React a une communauté en ligne florissante, une excellente documentation et même des cours sur son site Web. Quelque temps après sa sortie, l'équipe React a créé React Native, un framework pour le développement mobile hybride. React rend les pages Web d'une manière qui les rend dynamiques et réactives à l'entrée d'un utilisateur. Il s'agit d'une bibliothèque JavaScript open source qui vous permet de créer d'impressionnantes interfaces côté utilisateur, rapides et conviviales pour le référencement. Certaines fonctionnalités de React.js incluent des composants réutilisables, une liaison de données unidirectionnelle et le DOM virtuel. Certaines applications notables créées avec React sont Facebook (page d'accueil), Instagram (géolocalisation et marquage).

	VueJs	Angular	React
Langage utilisé	TypeScript	TypeScript	Javascript
Scalabilité	Faible	Haute	Haute
Communauté des développeurs	Petite	Grande	Très grande
Licence	MIT	MIT	MIT
Performance	Haute	Moyen	Haute

FIG. 4.10 – Tableau comparatif des frameworks Fron-end

Pour notre projet, nous avons porté notre choix sur le framework angular qui dispose d'une documentation bien détaillée. Angular dispose aussi d'une grande communauté très disponible. C'est un framework qui met à la disposition des développeurs beaucoup de librairies faciles à utiliser.

4.1.3 Choix de message broker

Les files d'attente de messages sont l'épine dorsale de tout système distribué. Dans les systèmes avancés, une seule application ne peut être responsable de l'ensemble de l'opération. Au contraire, plusieurs applications sont interdépendantes pour exécuter leurs tâches et atteindre l'objectif du système dans son ensemble. Les applications individuelles reposent sur le transfert de données pour communiquer avec d'autres applications - et c'est là que le besoin de files d'attente de messages se fait sentir.

1. Apache Kafka



FIG. 4.11 – Logo Apache Kafka

Apache Kafka a été développé à l'origine par LinkedIn puis incubé à l'Apache Software Foundation. Il s'agit d'une plateforme de streaming de données open source écrite en Scala et Java. Bien qu'à l'origine conçu comme une file d'attente d'e-mails, le logiciel est rapidement devenu populaire en tant que plate-forme de diffusion d'événements, traitant actuellement des milliards d'événements chaque jour. Chaque fois que vous avez besoin d'une file d'attente de messages ou d'un système de courtier de messages, Kafka est un bon choix.

Caractéristiques:

- Tout comme un système de messagerie ou un système de mise en file d'attente de messages, Apache Kafka permet de publier et de s'abonner à des flux d'enregistrements.
- Les flux d'enregistrements sont stockés à l'aide d'une approche tolérante aux pannes.
- Il est évolutif car il permet le découplage des applications.
- Il offre un débit élevé pour gérer les transferts de données en temps réel.

2. ActiveMQ



FIG. 4.12 – Logo ActiveMQ

ActiveMQ est un service de messagerie open source populaire écrit en Java. Il a été créé à l'origine par LogicBlaze en 2004 en tant que courtier de messages open source, puis donné à Apache Software Foundation en 2007. Comme tous les autres courtiers de messages, il agit comme une plate-forme de communication entre plusieurs applications existant sur différents serveurs ou écrites dans différentes langues. Il implémente Java Message Service (JMS) et prend en charge plusieurs protocoles de messagerie, notamment AMQP

et MQTT. La dernière mise à jour a été publiée en juin 2020.

Caractéristiques:

- Plusieurs protocoles de connexion sont pris en charge.
- Les techniques de verrouillage au niveau des lignes de la base de données, le système de fichiers et d'autres modes sont utilisés pour la haute disponibilité.
- Outre l'authentification simple et l'authentification JAAS, ActiveMQ propose également une API pour les plug-ins d'authentification personnalisés.
- Outre la mise à l'échelle verticale, une fonctionnalité intégrée de mise à l'échelle horizontale, appelée Réseau de courtiers, est également prise en charge.
- Prend en charge plusieurs protocoles de transport, notamment STOMP, REST et OpenWire.

3. RabbitMQ



FIG. 4.13 – Logo RabbitMQ

RabbitMQ est un autre courtier de messages open source largement utilisé, employé par de nombreuses entreprises à travers le monde. Il est écrit en Erlang et a été conçu à l'origine pour le protocole AMQP (Advanced Message Queuing Protocol), mais a été mis à jour pour prendre en charge d'autres protocoles, notamment STOMP et MQTT. La dernière version 3.8.9 est sortie en octobre 2019.

Caractéristiques:

- Plusieurs techniques de messagerie sont prises en charge, notamment la messagerie pub-sub, point à point et demande-réponse. Vous pouvez également écrire le vôtre et le nourrir sous forme de plugin.
- Des modes de communication synchrones et asynchrones sont disponibles.
- Les accusés de réception garantissent un service fiable.
- La haute disponibilité est assurée par la réplication des files d'attente sur plusieurs nœuds d'un cluster, garantissant que les messages ne sont pas perdus même en cas de panne matérielle.
- L'interface utilisateur de gestion permet un contrôle et une surveillance conviviaux des fonctions du courtier de messages.

	Apache Kafka	RabbitMQ	ActiveMQ
Open source	oui	oui	oui
Écrit en	scala	Erlang	Java
Protocoles	Protocole binaire sur TCP.	AMQP, MQTT, HTTP, WebSockets	AMQP, AUTO, MQTT, REST, etc
Langages supportés	Java, PHP, Python, Go, Haskell, C#, Ruby, Node.js, OCaml, etc	Java, Python, PHP, Ruby, JavaScript,etc	Java, C, C++, Ruby, Perl, PHP,etc

FIG. 4.14 – Tableau comparatif des logiciels de message broker

Pour notre projet, nous avons porté notre choix sur apache kafka.

Pourquoi Apache Kafka ? Le principal avantage de l'utilisation de Kafka est qu'il peut utiliser des partitions pour paralléliser des sujets, et chaque partition peut être hébergée sur un ordinateur différent. Les consommateurs peuvent accéder aux sujets en parallèle. Vous pouvez également autoriser plusieurs utilisateurs à accéder à plusieurs partitions, ce qui facilite l'expansion. Cette fonctionnalité unique améliore considérablement le débit du traitement des files d'attente de messages par Kafka. Ses excellentes capacités d'extension lui permettent de gérer un grand nombre de charges de travail. Par exemple, LinkedIn utilise Kafka pour gérer 300 milliards d'événements utilisateur chaque jour.

4.2 Architecture technique

Après avoir fixé les choix techniques nécessaires pour concevoir une architecture en microservices, nous avons besoin d'avoir une image globale sur l'architecture cible et son fonctionnement. La figure ci-dessus présente l'architecture qui résume l'étude technique évoquée précédemment.

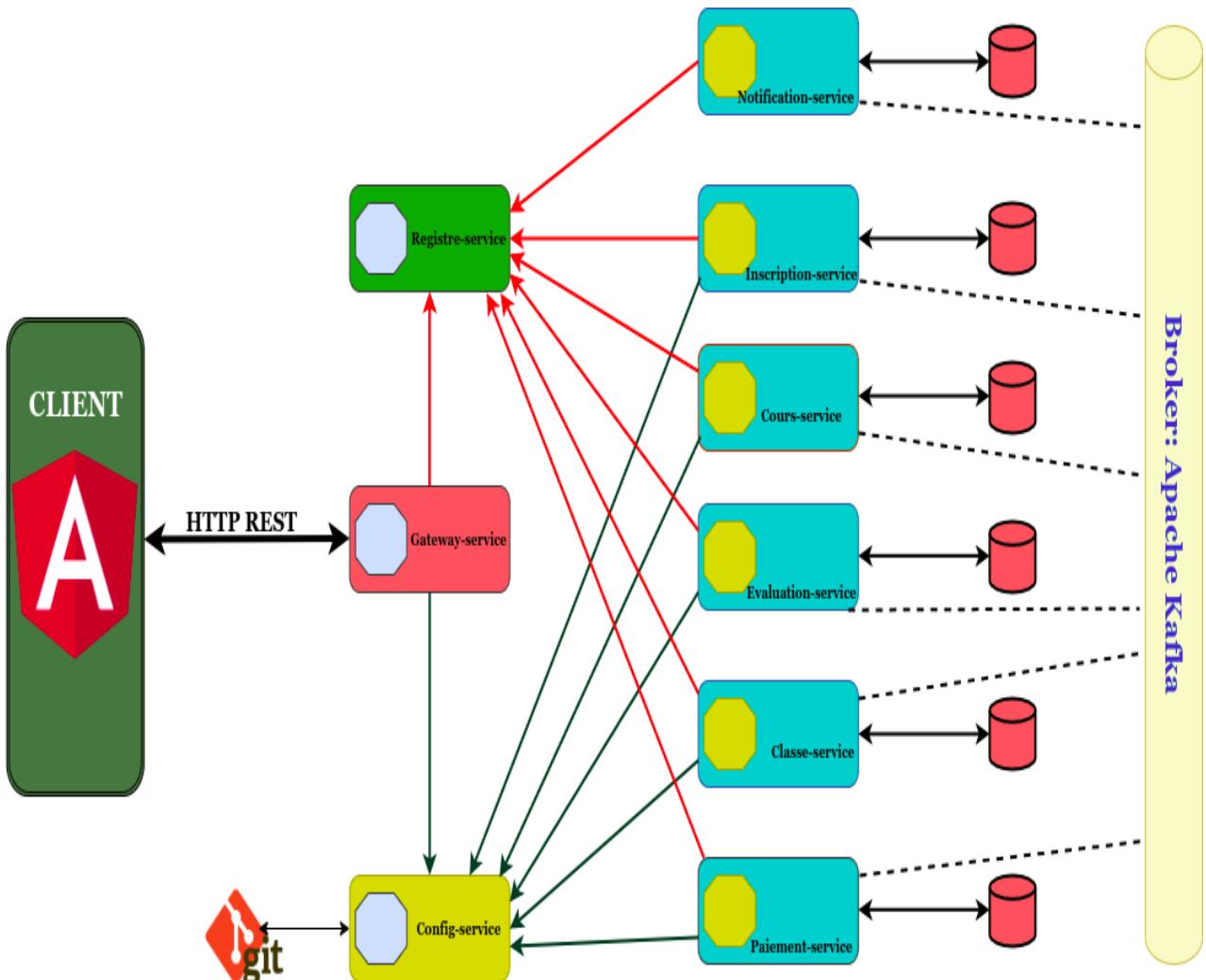


FIG. 4.15 – Architecture technique du logiciel G-school

4.3 Conception et réalisation des microservices identifiés

Dans cette partie, nous allons passer à la conception et aux réalisations des différents microservices de notre application. Pour chaque microservice, nous présenterons les besoins fonctionnels, le diagramme de classe, son architecture, le diagramme de séquence et son implémentation.

4.3.1 Microservice Inscription-service

4.3.1.1 Analyse fonctionnelle

Ce microservice est réalisé pour la gestion des inscriptions des élèves de l'établissement. Les fonctionnalités qui seront gérées dans ce microservice sont :

- les inscriptions pour une nouvelle année scolaire
- la gestion des informations liées aux élèves
- la gestion des informations des parents d'élèves

Le secrétaire de l'école est l'acteur qui a les prérogatives d'enregistrer une nouvelle inscription, de la modifier, de la supprimer et de consulter la liste des inscriptions existantes. Il a aussi le droit d'ajouter un élève, de le supprimer ou de modifier ses informations. Toutes ces opérations nécessitent une authentification.

La figure suivante résume les fonctionnalités attendues pour ce microservice.

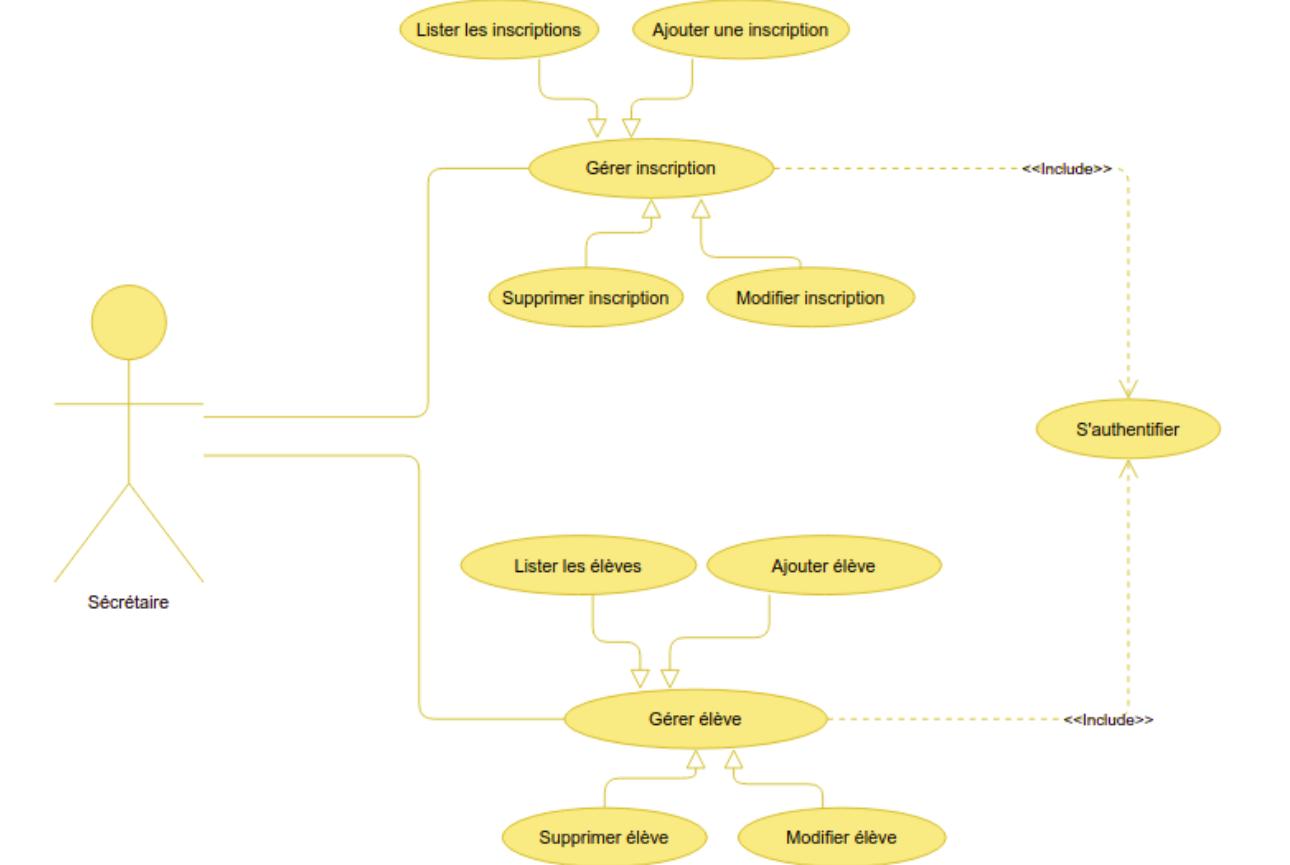


FIG. 4.16 – Diagramme de cas d'utilisation du microservice *Inscription-service*

4.3.1.2 Diagramme de classe

La figure ci-dessous illustre le diagramme de classe de notre microservice Inscription et les relations qui existent entre les classes.

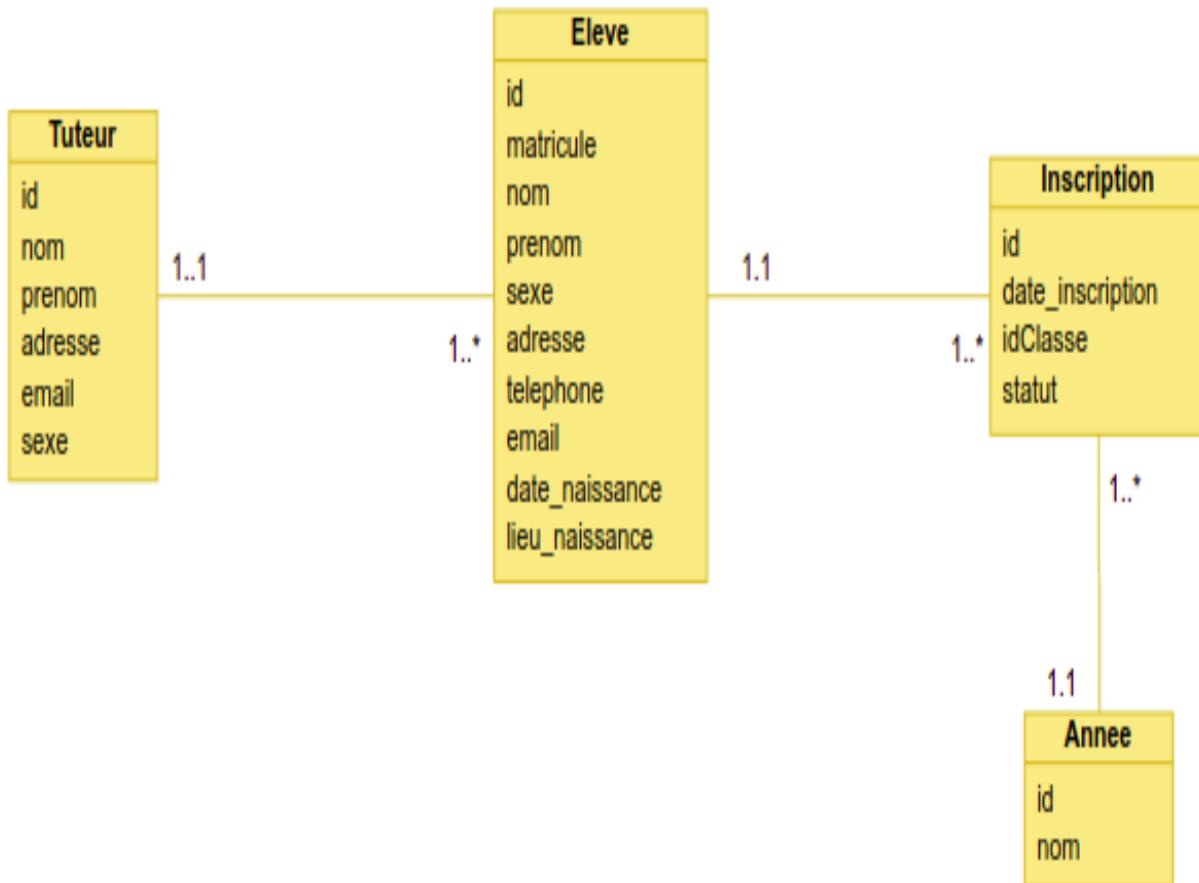


FIG. 4.17 – Diagramme de classe du microservice *Inscription-service*

4.3.1.3 Description du scénario d'inscription

Dans cette partie, nous allons présenter le diagramme de séquence qui décrit les différentes séquences à suivre pour inscrire un élève.

Ce diagramme est représenté par la figure suivante.

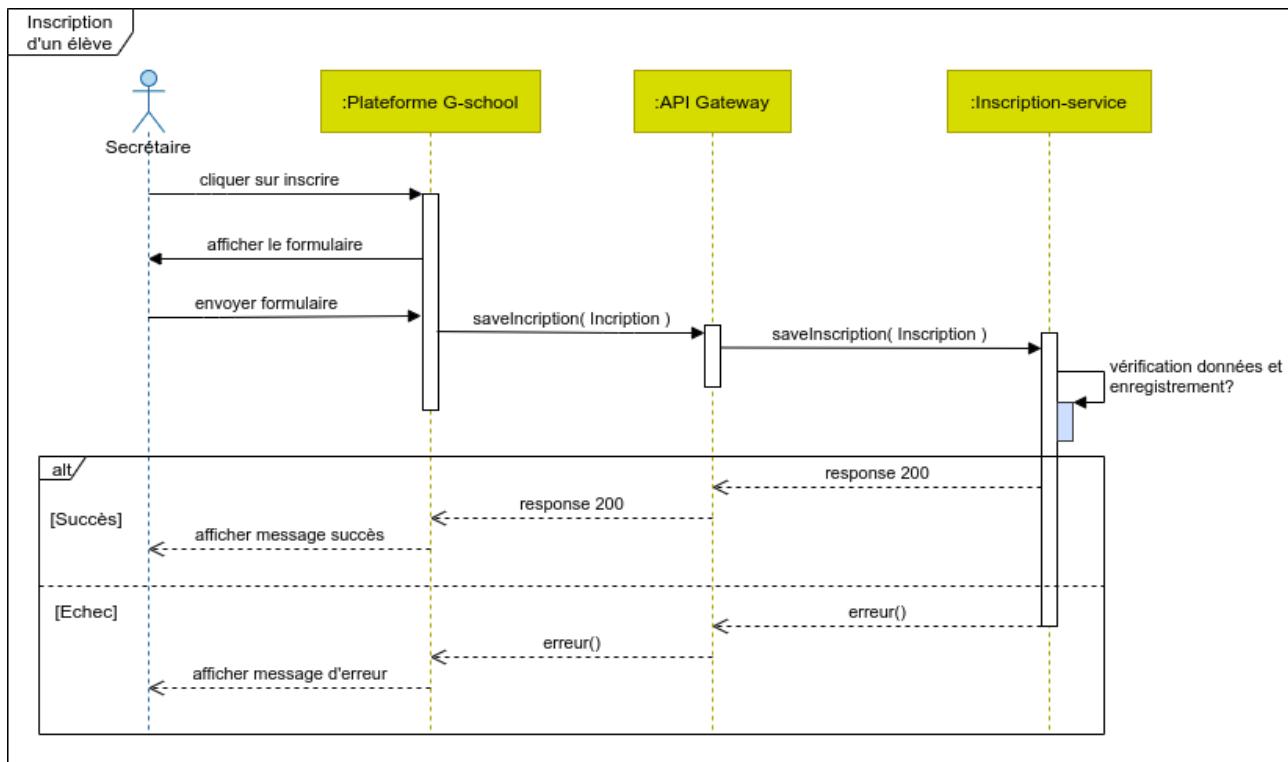


FIG. 4.18 – Diagramme de séquence pour inscrire un élève

4.3.1.4 Architecture logicielle du microservice Inscription-service

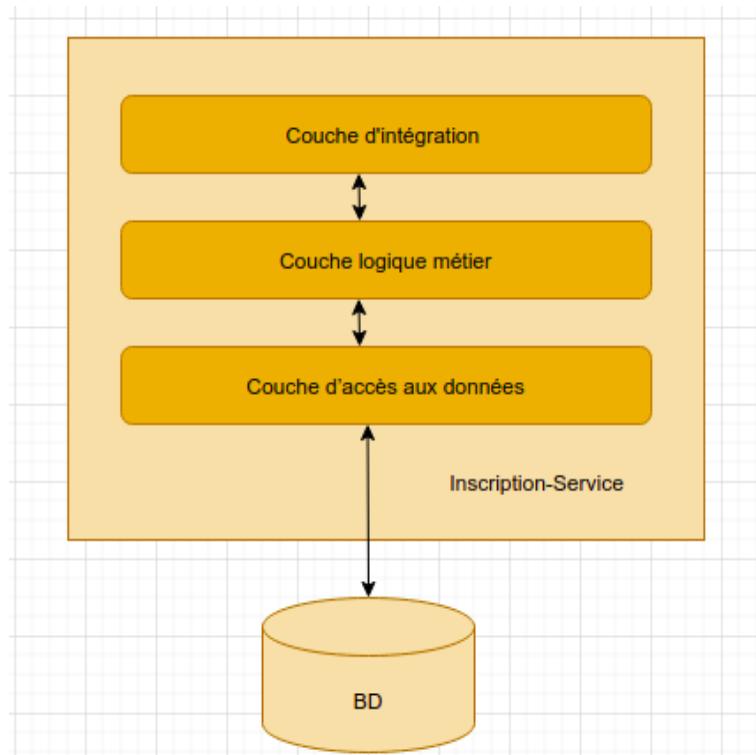


FIG. 4.19 – Architecture logicielle du microservice Inscription-service

La couche d'intégration : Elle est la responsable de la communication avec les autres microservices du système et elle délègue le traitement à la couche couche logique métier.

La couche logique métier : Elle est responsable du traitement de la logique métier.

La couche d'accès aux données : Elle est la responsable de la communication avec la base de données.

4.3.1.5 Architecture technique du microservice Inscription-service

La figure ci-dessous représente l'architecture technique du microservice Inscription-service.

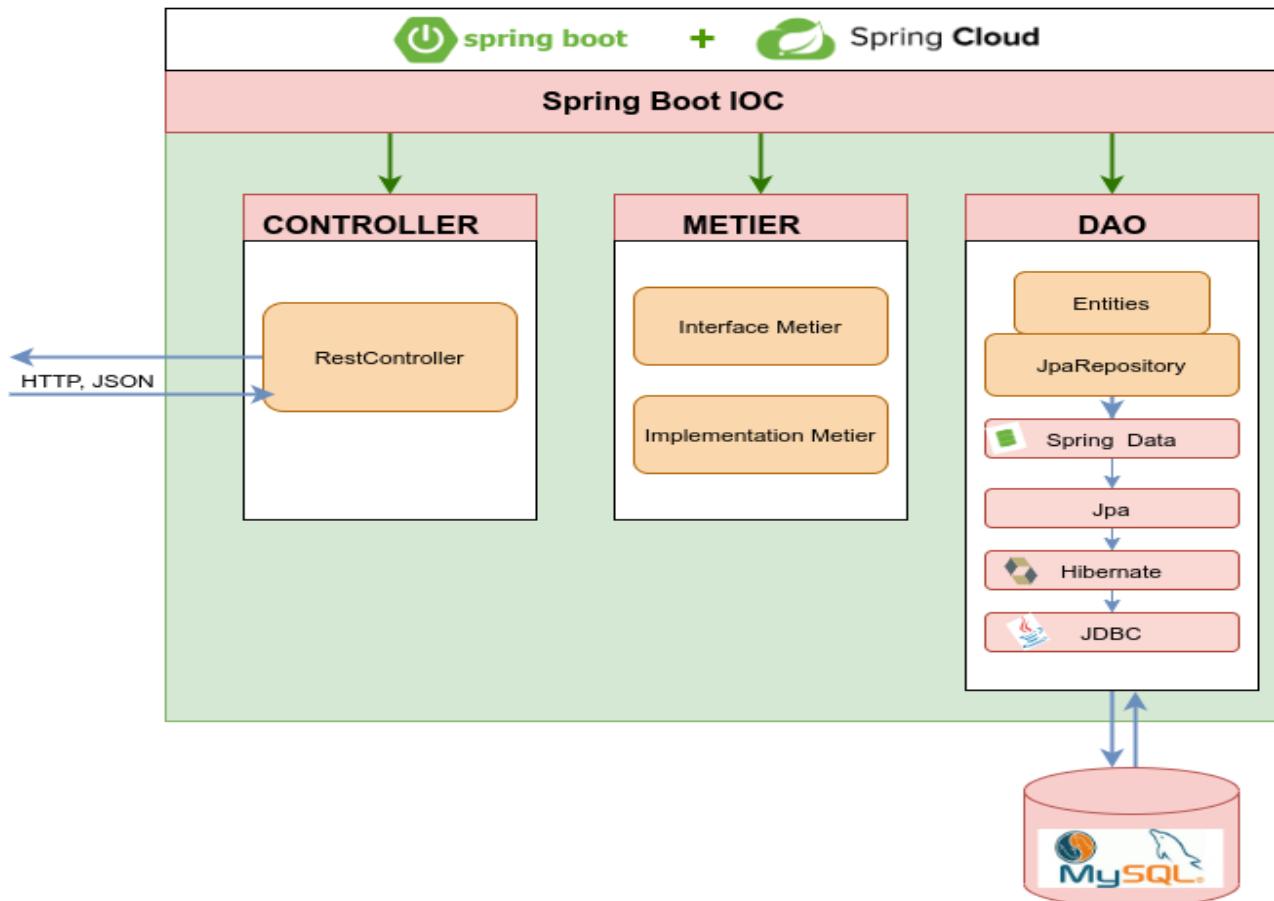


FIG. 4.20 – Architecture technique du microservice Inscription-service

4.3.1.6 Implémentation du microservice Inscription-service

Dans cette partie, nous exposerons les interfaces liées au microservice Inscription-service.



FIG. 4.21 – Page d'accueil du secrétaire de l'école

Toutes les opérations que le secrétaire peut faire sont illustrées dans les captures ci-dessous.

La liste des élèves inscrits					
Rechercher					
Matricule	Prénom	Nom	Adresse	Action	
20150757N	ABDOU	DEME	PIKINE	⋮	
201603T	ALBERT	EINSTEIN	GUÉDIAWAYE	⋮	
201603M	ALIOU	DIALLO	YOFF	⋮	
20140757N	AMINATA	DIALLO	PIKINE	⋮	
20140757N	ALIOU	DIALLO	PIKINE	⋮	
2017870A	AWA	DIAMÉ	GRAND YOFF	⋮	

Items per page: 6 1 - 6 of 11

FIG. 4.22 – La liste de tous les élèves inscrits

Le secrétaire peut cliquer sur le bouton « Incrire un élève » pour ajouter une nouvelle inscription. L'inscription peut concerner un nouvel élève ou bien un élève qui est déjà dans

l'établissement. Vous trouverez sur les trois(3) captures ci-dessous l'interface d'inscription d'un élève pour une année scolaire.

FICHE D'INSCRIPTION PEDAGOGIQUE

1 Infos Personnelles 2 Infos Pédagogiques 3 Infos Tuteur

Renouvellement ?

Prénom * khady

Nom * thior

Sexe * Féminin

Date de naissance * 10/7/2003

Lieu de naissance * rufisque

Adresse * rufisque

Téléphone * 774534033

Email kthior@crac.ed.sn

Suivant

FIG. 4.23 – Interface inscription 1

FICHE D'INSCRIPTION PEDAGOGIQUE

1 Infos Personnelles 2 Infos Pédagogiques 3 Infos Tuteur

Matricule * 202189TR

Statut * Normal

Niveau * Troisième

Classe * 3ème B4

Précédent Suivant

FIG. 4.24 – Interface inscription 2

FIG. 4.25 – Interface inscription 3

Pour chaque élève inscrit on peut consulter sa fiche.

Matricule	Prénom	Nom	Adresse	Action
2017870A	AWA	DIAMÉ	GRAND YOFF	⋮
20140757N	ALIOU	DIALLO	PIKINE	⋮
20140757N	ALIOU	DIALLO	PIKINE	⋮
2021075AT	ABIBOU	NDOYE	KAOLACK	⋮
202189TR	KHADY	THIOR	RUFISQUE	⋮

FIG. 4.26 – liste des élèves inscrits avec les boutons d'action

The screenshot shows the G-SCHOOL application's student profile page. At the top, there is a header with the logo 'G-SCHOOL' and navigation links for 'Année scolaire 2020-2021', 'Paramètres', and 'Déconnexion'. On the left, a sidebar menu includes 'Accueil', 'Tableau de bord', 'Registre des élèves' (with 'Elèves' selected), 'Scolarité', 'Registre Pédagogiques' (with 'Profs & Cours', 'Classes & Disciplines', 'Evaluations & Notes', and 'Emploi Du Temps' listed), and a user profile 'Deme Einstein | Ingénieur'. The main content area displays a student profile for 'Khady Thior' (Matricule: 202189TR, Status: Normal, Class: 3eme B4, Level: Troisième). Below the profile are tabs for 'ETAT CIVIL', 'TUTEUR', 'CONTACT', 'SCOLARITE', 'NOTES', and 'ABSENCES'. Under the 'ETAT CIVIL' tab, details are shown: First Name - Khady, Last Name - Thior, Sex - F, Date of Birth - 10/07/2003, Place of Birth - Rufisque, and Country - Sénégal.

FIG. 4.27 – la fiche de l'élève

4.3.2 Microservice Cours-service

4.3.2.1 Analyse fonctionnelle

Ce microservice est créé dans le but de permettre à l'administration de l'école de planifier les cours, gérer les professeurs et les absences. Pour notre application, c'est le secrétaire qui a les priviléges de planifier les cours et gérer les professeurs. Ces derniers ont aussi les priviléges d'ajouter, de supprimer et de modifier une absence.

NB: la plateforme est paramétrable donc ces priviléges peuvent être retirés ou alloués à un autre profil utilisant le système.

La figure suivante présente le diagramme de cas d'utilisation de ce microservice.

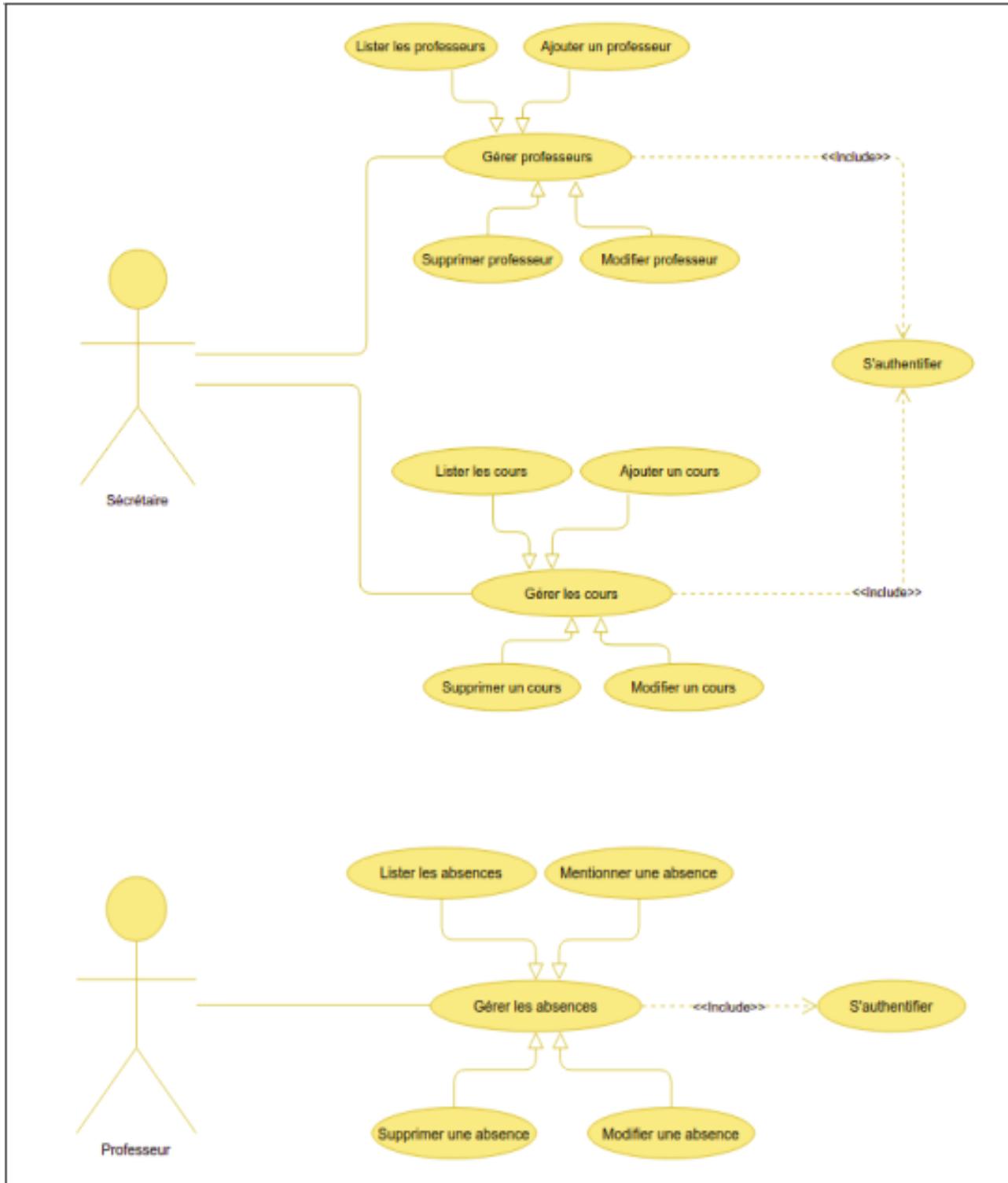


FIG. 4.28 – Diagramme de cas d'utilisation du microservice Cours-service

4.3.2.2 Diagramme de classe

La figure qui suit présente le diagramme de classe du microservice Cours-service.

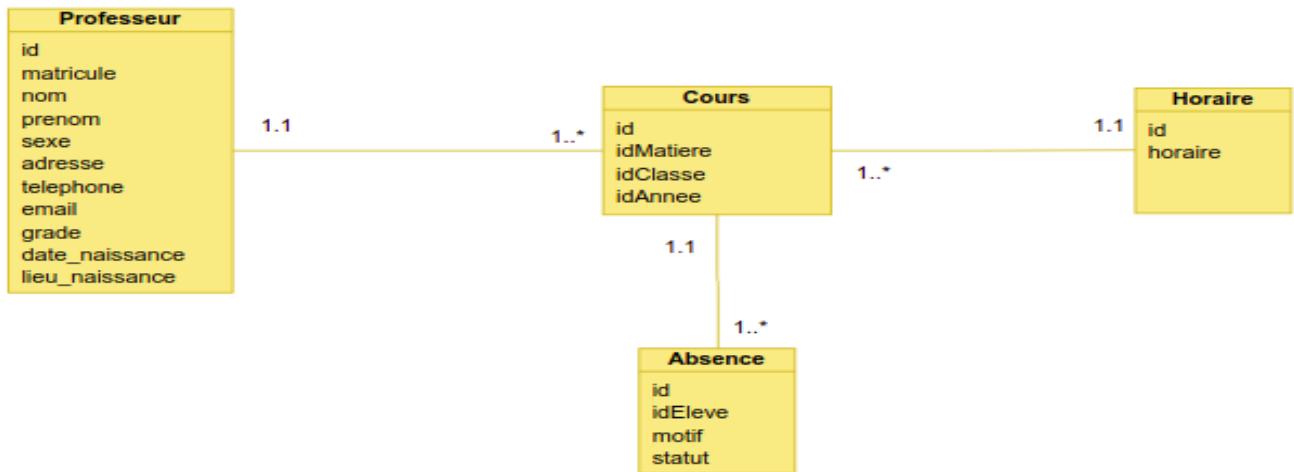


FIG. 4.29 – Diagramme de classe du microservice Cours-service

4.3.2.3 Elaboration du diagramme de séquence

Dans cette partie, nous allons présenter le diagramme de séquence pour la prise de la liste des absents lors d'un cours.

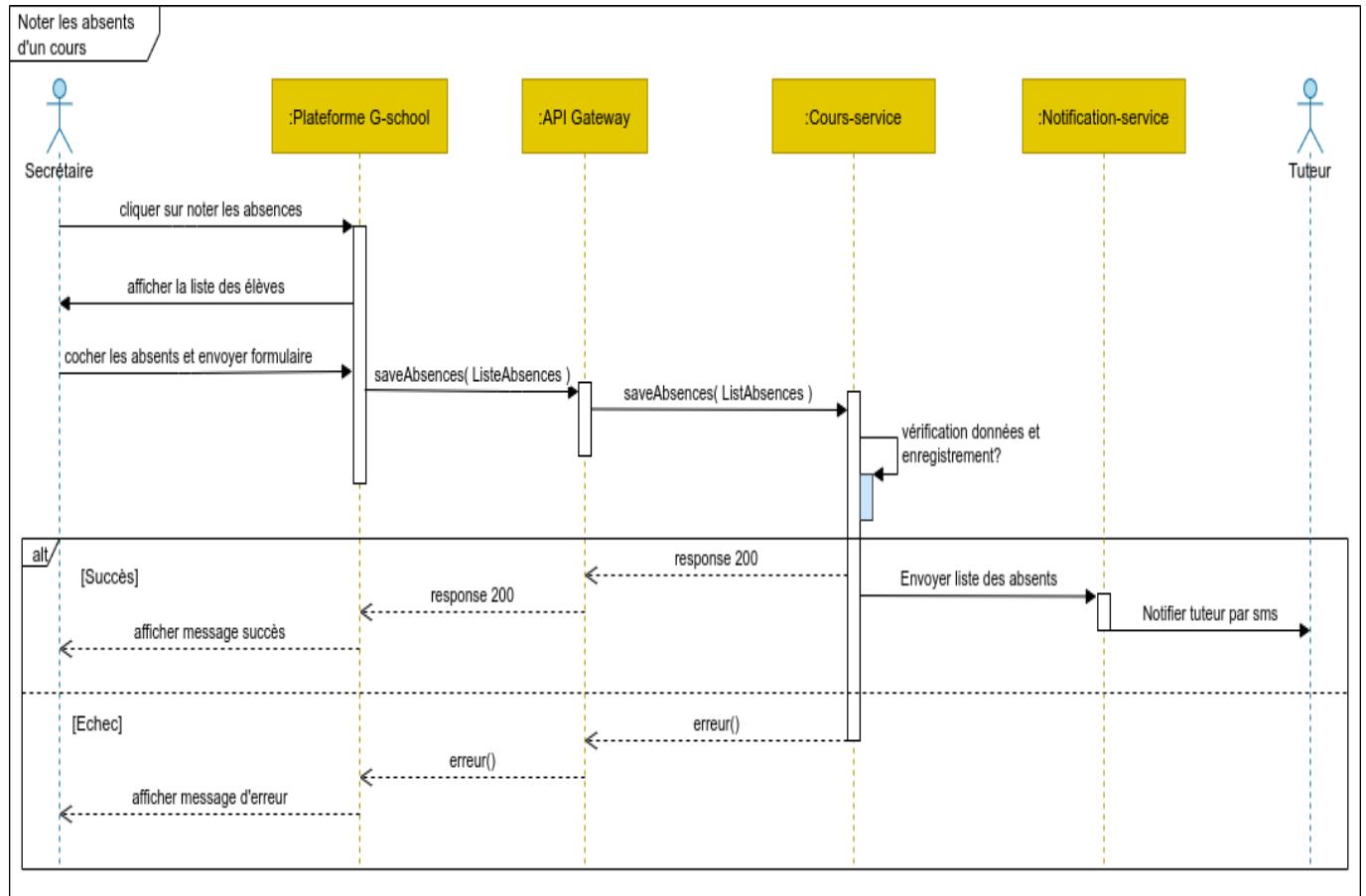


FIG. 4.30 – Diagramme de séquence pour noter les absents

4.3.2.4 Architecture technique du microservice Cours-service

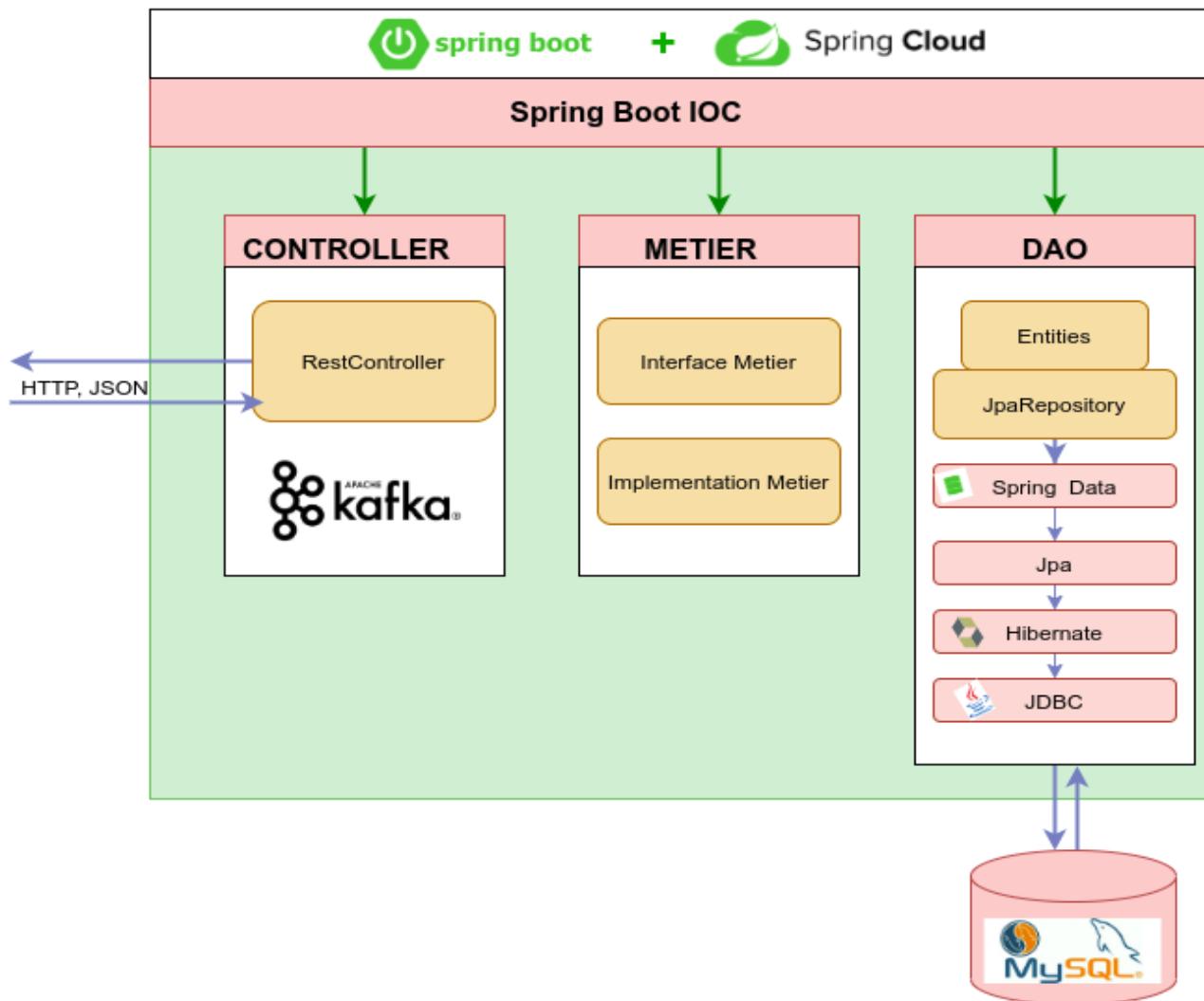


FIG. 4.31 – Architecture technique du microservice Cours-service

4.3.2.5 Réalisation du microservice Cours-service

Nous présentons dans cette partie les interfaces réalisées pour la gestion des professeurs, la planification des cours et la gestion des absences.

The screenshot shows the G-SCHOOL application's teacher management module. The left sidebar includes links for Accueil, Tableau de bord, Registre des élèves, Elèves, Absences, Scolarité, Prof & Cours (highlighted), Classes & Disciplines, Evaluations & Notes, and Emploi Du Temps. The main content area displays a table titled "Liste Des Professeurs" with columns: Matricule, Prénom, Nom, Téléphone, and Action. The table contains four rows of data. A "Nouveau Prof" button is located in the top right corner of the main content area.

Matricule	Prénom	Nom	Téléphone	Action
2016014H	AWA	DIANÉ	774534033	⋮
2017014N	BARHAM	GUEYE	774534033	⋮
2021AN179CR	ABIBOU	NDIAYE	774534033	⋮
2021AS112CR	ADAMA	SADIJI	774534033	⋮

FIG. 4.32 – La liste des professeurs

A partir de l'interface présenté dans FIG.4.32, le secrétaire ou le directeur peut visualiser l'ensemble des professeurs de l'établissement. Pour ajouter un nouveau professeur, on clique sur le bouton « Nouveau Prof » pour ouvrir le formulaire présenté dans l'interface ci-dessous.

The screenshot shows the "Nouveau Professeur" (New Teacher) form. The left sidebar is identical to FIG.4.32. The main form fields include: Prénom * (Aminata), Nom * (Diallo), Date Naissance * (2/3/1992), Lieu Naissance * (Diourbel), Adresse * (Patte d'oeie builders), Niveau d'étude * (Master 2), Téléphone * (774534033), and Email * (aminata@gmail.com). A "Ajouter" (Add) button is at the bottom right.

FIG. 4.33 – Ajouter un professeur

The screenshot shows the G-SCHOOL application's course management interface. On the left, a sidebar menu includes links for Accueil, Tableau de bord, Registre des élèves, Elèves, Absences, Scolarité, Prof & Cours (highlighted in blue), Classes & Disciplines, Evaluations & Notes, and Emploi Du Temps. The main content area is titled 'Cours' and displays a table of courses with columns for Cours (Subject), Jour (Day), Horaire (Time), Prof (Teacher), Classe (Class), and Action (More options). The table contains five entries:

Cours	Jour	Horaire	Prof	Classe	Action
ANGLAIS	JEUDI	8H-9H	AWA DIANÉ	TI2B	⋮
ANGLAIS	MARDI	8H-9H	AWA DIANÉ	6ème B	⋮
ANGLAIS	LUNDI	8H-9H	AWA DIANÉ	3eme M2	⋮
MATHS	MERCREDI	8H-9H	AWA DIANÉ	TI2B	⋮

At the bottom right of the table, there are pagination controls: 'Items per page: 4', '1 - 4 of 5', and navigation arrows.

FIG. 4.34 – La liste des cours

L’interface présentée dans FIG.4.34 ci-dessus visualise l’ensemble des cours dispensés dans l’établissement. On peut ajouter un nouveau cours en cliquant sur le bouton « Nouveau Cours » et renseigner le formulaire(voir FIG.4.35).

The screenshot shows the 'Nouveau Cours' (New Course) form. The sidebar menu is identical to FIG.4.34. The main form has a header 'Nouveau Cours' and contains four input fields:

- Matière *: dropdown menu set to 'Anglais'
- Classe *: dropdown menu set to 'TI2B'
- Jours *: dropdown menu set to 'JEUDI'
- Horaire: dropdown menu set to '10h-11h30' with a smiley face icon

Below these fields is a text input for 'Enseignant *' containing '2017014N - BARHAM GUEYE'. At the bottom are two buttons: 'Fermer' (Close) in red and 'Ajouter' (Add) in black.

FIG. 4.35 – Ajouter un cours

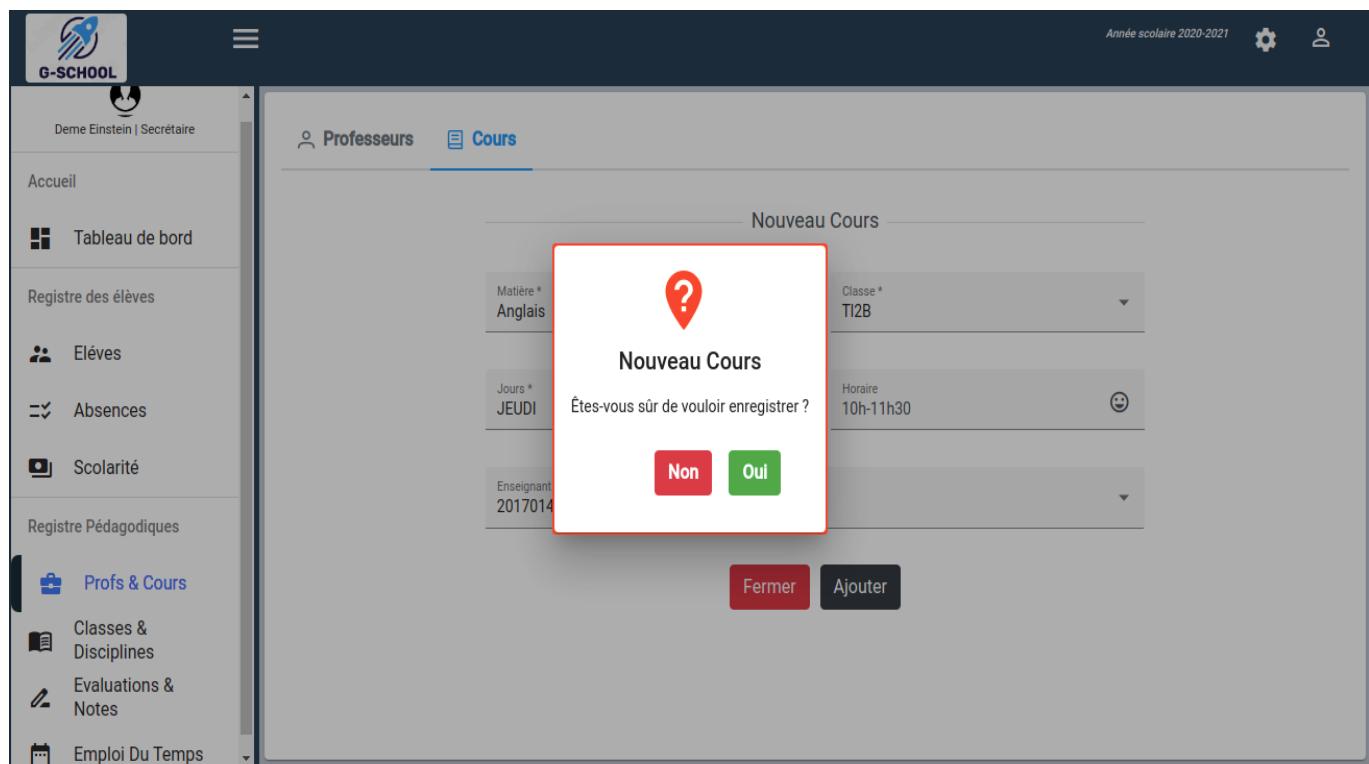


FIG. 4.36 – Confirmation ajouter un cours

The screenshot shows the 'Absences' (Absences) section of the G-SCHOOL application. The sidebar menu is identical to Fig. 4.36. The main area shows a list titled 'Liste des absents de la classe de TI2B' (List of absents for the TI2B class). It includes tabs for other classes: TI2B (selected), 3ème B4, TS2A, 3eme M2, 6ème B, and 6ème A. A button 'Saisir les absences' (Enter absences) is visible. The list table has columns: 'Elève' (Student), 'Cours' (Subject), 'Heure' (Time), 'Justifiée ?' (Justified?), and 'Action'. The data is as follows:

Elève	Cours	Heure	Justifiée ?	Action
ALBERT EINSTEIN	MATHS	8H-9H	✗	⋮
ABIBOU NDOYE	MATHS	8H-9H	✗	⋮
ALIOU DIALLO	ANGLAIS	8H-9H	✓	⋮
ALBERT EINSTEIN	ANGLAIS	8H-9H	✗	⋮

At the bottom, there are pagination controls: 'Items per page: 4', '1 - 4 of 12', and navigation arrows.

FIG. 4.37 – Liste des absents d'une classe

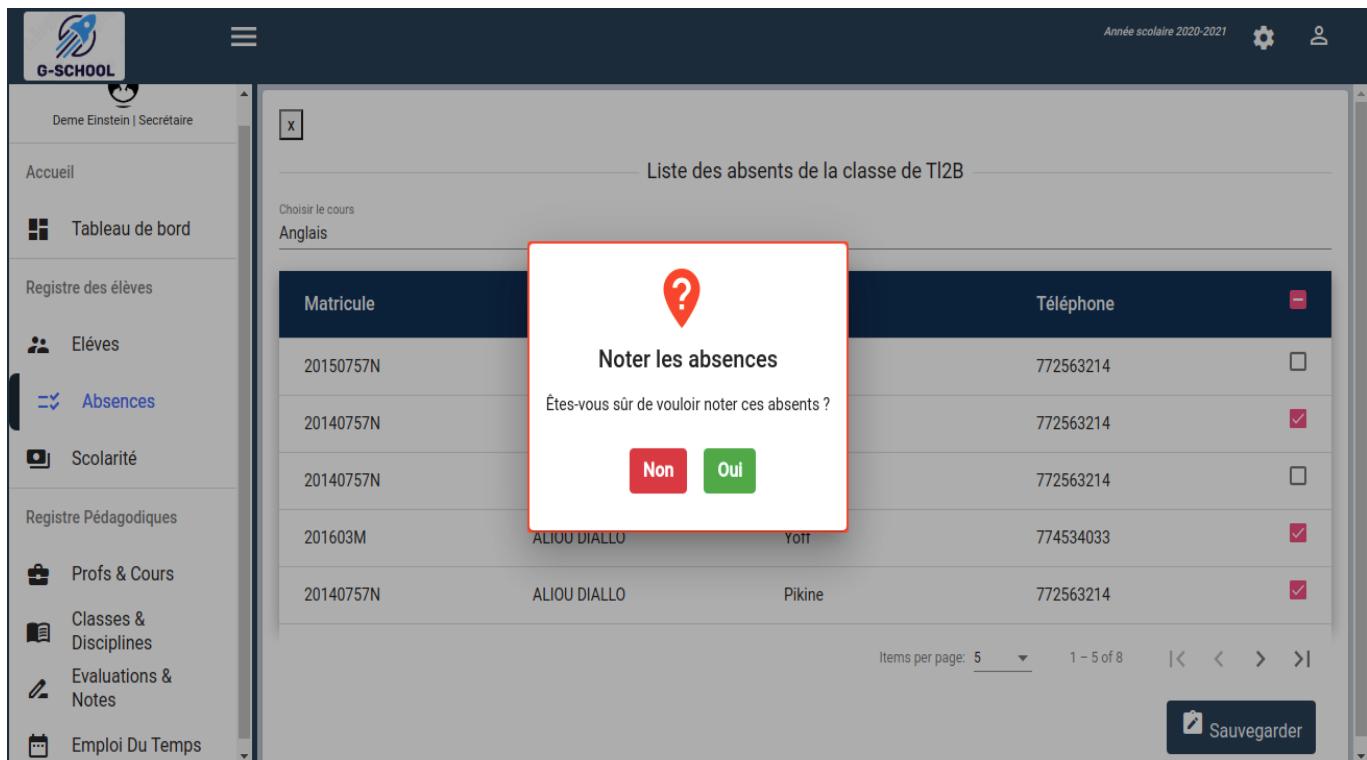


FIG. 4.38 – Noter les absents d'un cours

Pour renseigner la liste des absents, on doit d'abord choisir le cours concerné, cocher l'ensemble des élèves qui ne sont pas présents et enfin cliquer sur le bouton sauvegarder situé en bas pour enregistrer les absents dans la base de données.

4.3.3 Microservice Classe-service

4.3.3.1 Analyse fonctionnelle

Ce microservice est créé pour la gestion des classes. Dans la gestion des classes on a la gestion des matières dispensées et les niveaux d'études. Le secrétaire est désigné comme l'acteur ayant les priviléges pour faire ces différentes opérations mais le système est paramétrable donc les priviléges donnés au secrétaire peuvent être retirés ou alloués à un autre profil utilisant le système.

Le diagramme de cas d'utilisation de ce microservice est représenté par la figure suivante.



FIG. 4.39 – Diagramme de cas d'utilisation du microservice Classe-service

4.3.3.2 Diagramme de classe



FIG. 4.40 – Diagramme de classe du microservice Cours-service

4.3.3.3 Architecture logicielle et technique du microservice Classe-service

Ce microservice a les mêmes architectures (logicielle et technique) que le microservice Inscription-service(voir FIG. 4.19 et 4.20).

4.3.3.4 Réalisation du microservice Classe-service

Les interfaces de gestion des classes sont présentées par les trois(3) figures ci-dessous.

#	Classe	Niveau	Mensualité	Action
1	TL2B	TERMINALE	12500 fcfa	⋮
2	3ÈME B4	TROISIÈME	8500 fcfa	⋮
3	TS2A	TERMINALE	12500 fcfa	⋮

FIG. 4.41 – Liste des classes et le formulaire pour ajouter une nouvelle classe

Pour afficher les matières dispensées dans une classe, on clique sur les trois(3) points puis sur le bouton « Matières ». La figure 4.42 présente l'interface des matières dispensées à la classe de terminale L2B(TL2B).

#	Nom de la matière	Coefficient	Action
1	ANGLAIS	4	⋮ Modifier Supprimer
2	PHILOSOPHIE	6	
3	ESPAGNOL	5	

FIG. 4.42 – Liste des matières dispensées dans une classe

#	Nom de la matière	Action
1	ANGLAIS	⋮
2	MATHS	⋮
3	HISTOIRE & GÉOGRAPHIE	⋮
4	PHYSIQUE & CHIMIE	⋮

FIG. 4.43 – *Liste des matières dispensées dans l'établissement*

4.3.4 Microservice Evaluation-service

4.3.4.1 Analyse fonctionnelle

Ce microservice assure la gestion des évaluations. Les professeurs et le secrétaire sont les acteurs ayant les priviléges de faire les différentes opérations liées à la gestion des évaluations. Ces acteurs peuvent :

- planifier une évaluation(devoir ou composition)
- remplir les notes obtenues par les élèves lors d'un devoir ou d'une compostion
- générer les bulletins à la fin de chaque trimestre

Le diagramme de cas d'utilisation est présenté dans la figure suivant.



FIG. 4.44 – Diagramme de cas d'utilisation du microservice Evaluation-service

4.3.4.2 Diagramme de classe

La figure ci-dessous présente le diagramme de classe du microservice Evaluation-service.

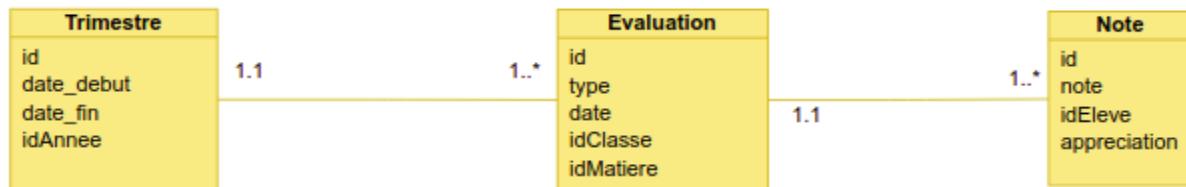


FIG. 4.45 – Diagramme de classe du microservice Evaluation-service

4.3.4.3 Architecture technique du microservice Evaluation-service

La figure suivante présente l'architecture technique du microservice Evaluation-service.

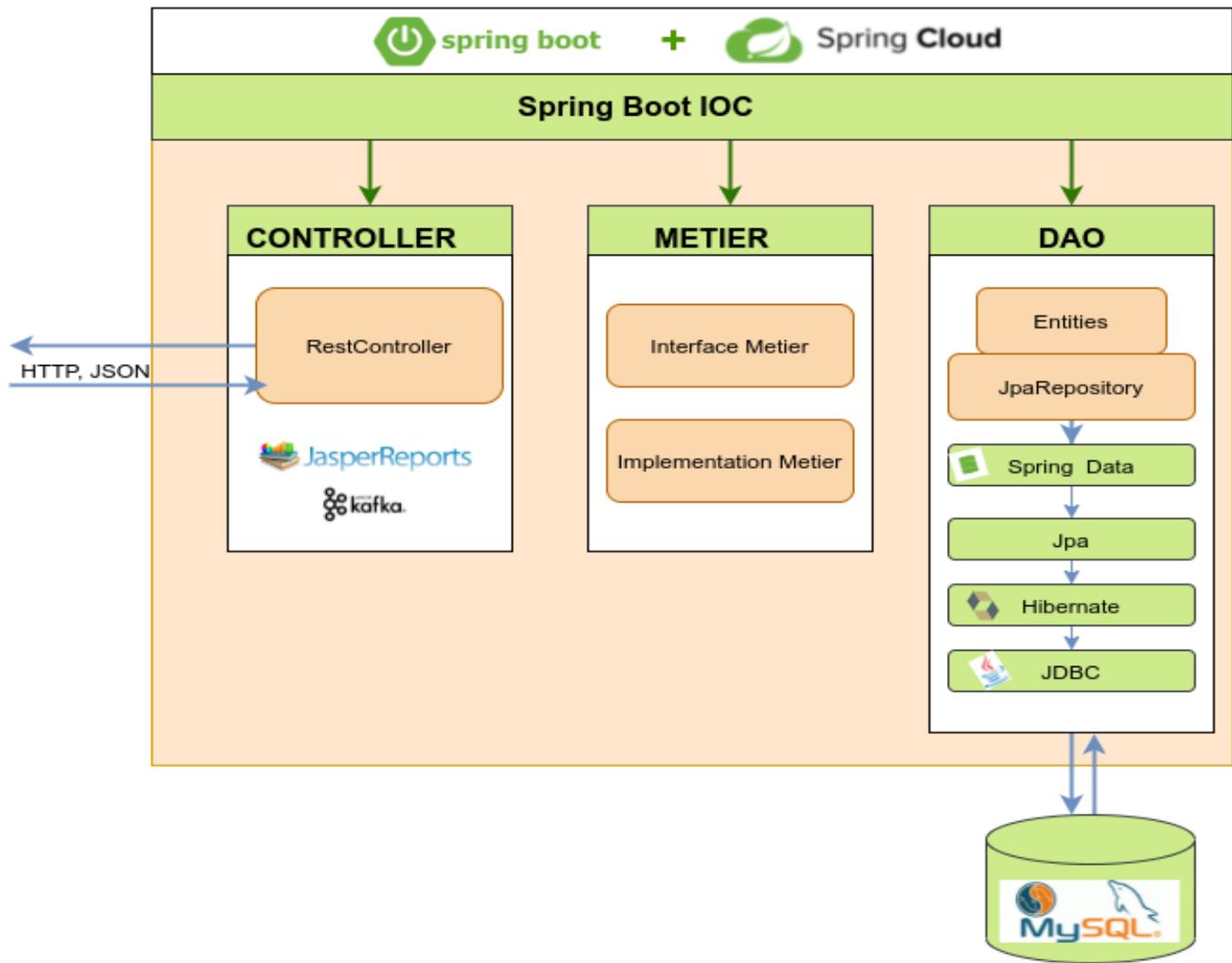


FIG. 4.46 – Architecture technique du microservice Evaluation-service

JasperReports est une bibliothèque de rapports open source qui permet aux utilisateurs de créer des rapports au pixel près qui peuvent être imprimés ou exportés dans de nombreux formats, notamment PDF, HTML et XLS. Elle est utilisée dans ce microservice pour la génération des bulletins de notes.

4.3.4.4 Implémentation du microservice Evaluation-service

Nous présentons dans cette partie quelques interfaces réalisées pour la gestion des évaluations. La figure suivante présente la liste des évaluations d'une classe.

The screenshot shows the G-SCHOOL application interface. On the left, a sidebar menu includes options like Accueil, Tableau de bord, and Evaluations & Notes. The main content area displays a table titled "Liste des évaluations de la classe TI2B". The table has columns for #, Matière, Type, Trimestre, Date, and Action. It lists four evaluations:

#	Matière	Type	Trimestre	Date	Action
1	PHYSIQUE & CHIMIE	DEVOIR	TRIMESTRE 2	02/03/2021	⋮
2	MATHS	COMPOSITION	TRIMESTRE 1	02/01/2021	⋮
3	PHILOSOPHIE	DEVOIR	TRIMESTRE 1	21/12/2020	⋮
4	ANGLAIS	DEVOIR	TRIMESTRE 1	26/11/2020	⋮

At the bottom right of the table, there are pagination controls: "Items per page: 4", "1 - 4 of 5", and navigation arrows.

FIG. 4.47 – Liste des évaluations de la classe TL2B

Pour programmer une évaluation, on clique sur le bouton « Evaluation » puis on renseigne le formulaire et cliquer sur le bouton « Ajouter ».

The screenshot shows the G-SCHOOL application interface. A confirmation dialog box is centered on the screen with the title "Nouvelle évaluation Pour TI2B". The dialog contains a question mark icon, the text "Nouvelle évaluation", and the message "Êtes-vous sûr de vouloir enregistrer?". At the bottom are two buttons: "Non" (red) and "Oui" (green). In the background, the main application window shows a table for adding a new evaluation. The table has fields for "Matière" (selected: Français), "Type" (selected: DEVOIR), "Trimestre" (selected: Trimestre 2), and a "Date" field. A large "Ajouter" button is at the bottom right of the table.

FIG. 4.48 – Ajouter une évaluation avec le pop-up de confirmation

Pour ajouter les notes d'une évaluation, on clique sur les trois(3) points dans la colonne « Action » puis sur le bouton « Notes »(cf fig. 4.49).

The screenshot shows the G-SCHOOL application's evaluation management module. On the left, a sidebar menu includes 'Accueil', 'Tableau de bord', 'Registre des élèves' (with 'Elèves' and 'Absences' sub-options), 'Registre Pédagogiques' (with 'Prof & Cours', 'Classes & Disciplines', 'Evaluations & Notes' (highlighted in blue), and 'Emploi Du Temps' sub-options). The main content area displays a table titled 'Liste des évaluations de la classe TI2B'. The table has columns for '#', 'Matière', 'Type', 'Trimestre', 'Date', and 'Action'. It lists four entries: 1. PHYSIQUE & CHIMIE (DEVOIR, TRIMESTRE 2, 02/03/2021), 2. MATHS (COMPOSITION, TRIMESTRE 1), 3. PHILOSOPHIE (DEVOIR, TRIMESTRE 1), and 4. ANGLAIS (DEVOIR, TRIMESTRE 1). A modal window on the right provides actions for each row: 'Notes', 'Modifier', and 'Supprimer'. Navigation tabs at the top include 'TI2B', '3ème B4', 'TS2A', '3eme M2', '6ème B', and '6ème A'. A search bar and a 'Rechercher' button are also present.

FIG. 4.49 – Liste des évaluations de la classe TL2B avec les actions

The screenshot shows the 'Evaluations & Notes' section of the G-SCHOOL application. The sidebar menu is identical to Fig. 4.49. The main content area is titled 'DEVOIR -- PHYSIQUE & CHIMIE' and shows a table titled 'Les notes de la classe de TI2B'. The table lists student grades for the 'PHYSIQUE & CHIMIE' assignment. Each row contains the student's name, their grade (Note), and a comment ('Appréciations'). The rows are for 'ABDOU DEME' (Note 12.5, 'Peut mieux faire'), 'AMINATA DIALLO' (Note 10, 'Passable'), 'ALIOU DIA' (Note 11, 'Peut mieux faire'), and 'ADAMA SAKHO' (Note 13, 'Abien'). At the bottom are buttons for 'Sauvegarder' (Save) and a '+' icon for adding new data.

FIG. 4.50 – Interface pour ajouter les notes d'une évaluation

The screenshot shows a web-based school management system interface. At the top, there's a header with the G-SCHOOL logo, a user profile for 'Deme Einstein | Secrétaire', and navigation links for 'Année scolaire 2020-2021', 'Paramètres', and 'Déconnexion'. On the left, a sidebar menu includes 'Accueil', 'Tableau de bord', 'Registre des élèves' (with 'Elèves' and 'Absences' sub-options), 'Registre Pédagogiques' (with 'Prof & Cours', 'Classes & Disciplines', 'Evaluations & Notes', and 'Emploi Du Temps' sub-options). The main content area is titled 'COMPOSITION – MATHS' and displays the message 'Les notes de la classe de TI2B'. It features a search bar and a 'Noter' button. Below is a table with columns: 'Matricule', 'Elève', 'Note', 'Appréciation', and 'Action'. The table contains three rows of data:

Matricule	Elève	Note	Appréciation	Action
20140757N	AMINATA DIALLO	15	BIEN	⋮
20150757N	ABDOU DEME	16	BIEN	⋮
20140757N	ALIOU DIA	11	PEUT MIEUX FAIRE	⋮

At the bottom right of the main content area, there are pagination controls: 'Items per page: 4', '1 - 3 of 3', and navigation arrows.

FIG. 4.51 – Liste des notes d'une évaluation

4.3.5 Microservice Utilisateur-service

4.3.5.1 Description

Ce microservice est créée pour assurer la gestion des utilisateurs. Il facilite la création et la gestion des utilisateurs et de leurs rôles, des autorisations accordées par catégorie et des priviléges. La figure ci-dessous présente le diagramme de classe de ce microservice.



FIG. 4.52 – Diagramme de classe du microservice Utilisateur-service

4.3.5.2 Interfaces

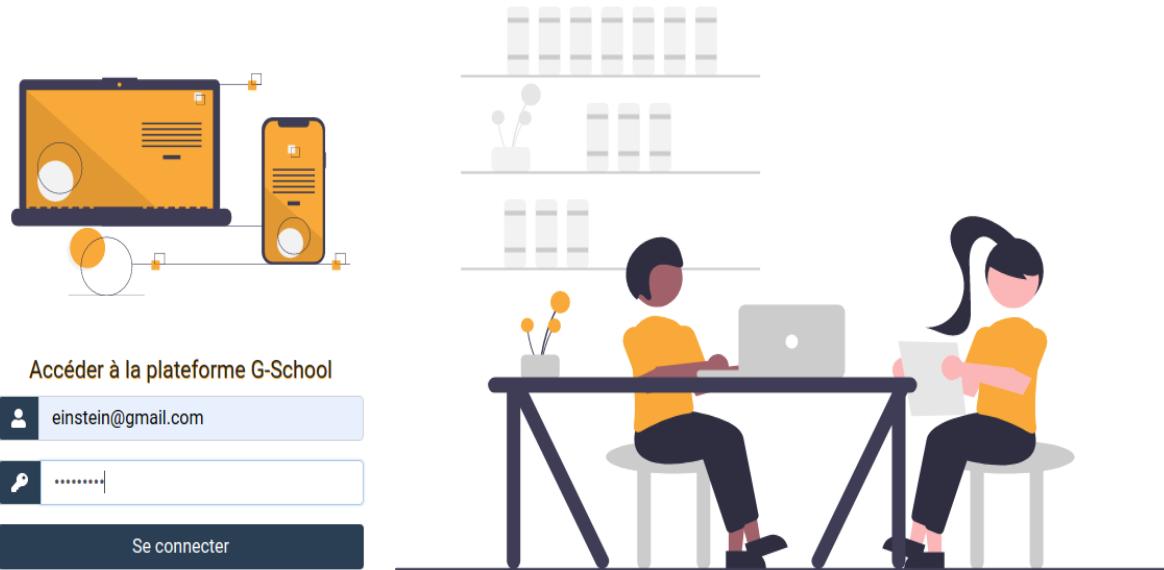


FIG. 4.53 – Interface de connexion

Chapitre 5

Intégration des microservices

Dans l'architecture de microservices, chaque service est une unité livrable indépendante. Chaque microservice peut être déployée pour répondre parfaitement à des fonctionnalités spécifiques. Par contre, nous avons besoins d'une plateforme cohérente qui apparait à l'utilisateur comme un composant unique et qui répond à nos différents besoins. Pour cela nous devons faire l'intégration des différents microservice. Ce chapitre présente la phase d'intégration des microservices déjà développés.

5.1 Service de découverte

Dans l'architecture des microservices, les services doivent se trouver et communiquer entre eux. C'est le service de découverte qui gère l'enregistrement et la localisation des microservices déployés. Le registre de service est un élément clé de la découverte de service. Il s'agit d'une base de données contenant les emplacements réseau des instances de service. Un registre de services doit être hautement disponible et à jour. Les clients peuvent mettre en cache les emplacements réseau obtenus à partir du registre de service. Cependant, ces informations finissent par devenir obsolètes et les clients deviennent incapables de découvrir les instances de service. Par conséquent, un registre de service se compose d'un cluster de serveurs qui utilise un protocole de réPLICATION pour maintenir la cohérence.

Dans notre architecture nous avons opté pour le service de découverte **Eureka** de la stack Netflix. Eureka fournit une API REST pour l'enregistrement et l'interrogation des instances de service. Une instance de service enregistre son emplacement réseau à l'aide d'une requête POST. Toutes les 30 secondes, elle doit actualiser son enregistrement à l'aide d'une requête PUT. Un client peut récupérer les instances de service enregistrées à l'aide d'une requête GET. Eureka a un dashboard qui nous permet de visualiser toutes les instances de nos microservices(voir FIG.5.2).

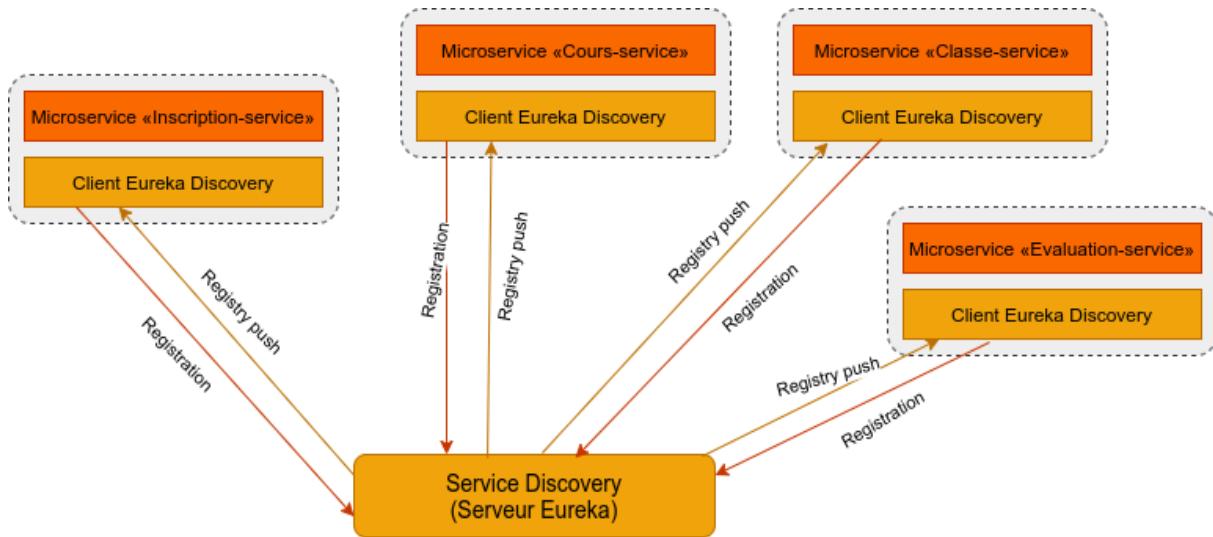


FIG. 5.1 – Enregistrement des microservices de G-School dans l’annuaire

System Status

Environment	N/A
Data center	N/A
Current time	2021-06-30T20:13:09 +0000
Uptime	00:07
Lease expiration enabled	false
Renews threshold	10
Renews (last min)	10

EMERGENCY! EUREKA MAY BE INCORRECTLY CLAIMING INSTANCES ARE UP WHEN THEY'RE NOT. RENEWALS ARE LESSER THAN THRESHOLD AND HENCE THE INSTANCES ARE NOT BEING EXPIRED JUST TO BE SAFE.

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
CLASSE-SERVICE	n/a (1)	(1)	UP (1) - localhost
EVALUATION-SERVICE	n/a (1)	(1)	UP (1) - localhost
PROXY-SERVER	n/a (1)	(1)	UP (1) - 192.168.1.14:proxy-server:9090
SERVICE-COURS	n/a (1)	(1)	UP (1) - 192.168.1.14:service-cours:8082
SERVICE-INSCRIPTION	n/a (1)	(1)	UP (1) - 192.168.1.14:service-inscription:8081

General Info

Name	Value
total-avail-memory	331mb
num-of-cpus	4
current-memory-usage	164mb (49%)
server-upptime	00:07
registered-replicas	http://localhost:8761/eureka/
unavailable-replicas	http://localhost:8761/eureka/ ,
available-replicas	

Instance Info

Name	Value
ipAddr	192.168.1.14
status	UP

FIG. 5.2 – Dashboard Eureka

5.2 L'API Gateway: Spring Cloud Gateway

L'API Gateway encapsule l'architecture du système interne et fournit une API adaptée à chaque client. Elle peut avoir d'autres responsabilités telles que l'authentification, la surveillance, l'équilibrage de charge, la mise en cache. Elle est responsable du routage des demandes, de la composition et de la traduction du protocole. Toutes les demandes des clients passent d'abord par la passerelle API. Elle achemine ensuite les demandes vers le microservice approprié.

Pour ce projet, nous avons utilisé **Spring Cloud Gateway** qui représente le point d'entrée à notre application. Elle nous permet d'assurer l'équilibrage de charge et la tolérance aux pannes.

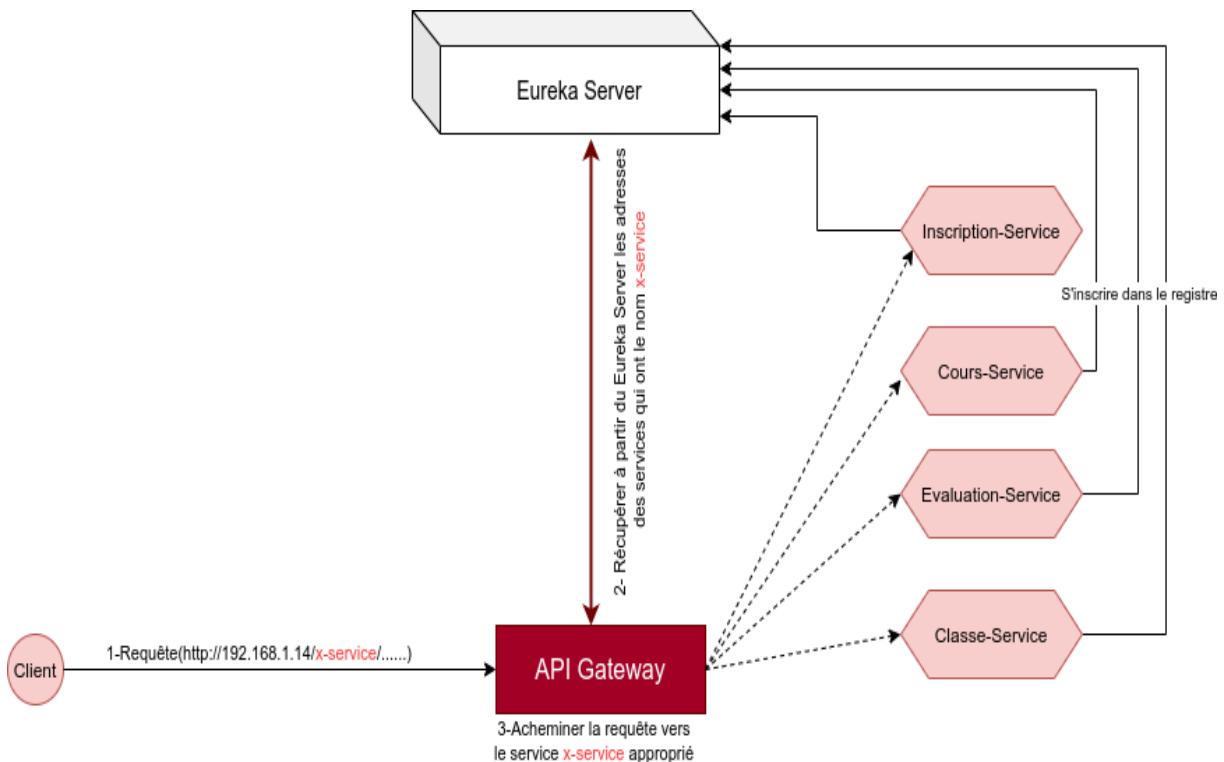


FIG. 5.3 – Fonctionnement de l'API Gateway

5.3 Communication entre les microservices

Dans l'architecture de microservices, chaque microservice gère ses propres données et ne les partage pas directement avec les autres services. Ces derniers sont autonomes mais ils ont besoins d'échanger des données pour réaliser certaines fonctionnalités. C'est pourquoi nous avons opté de réaliser des intercommunications synchrones via des protocoles ouverts et asynchrones via un service de file d'attente de messages.

5.3.1 Communication synchrone avec tolérance aux pannes

Pour assurer la communication synchrone entre nos différents microservices, nous avons utilisé **Spring Netflix Feign Client**. Avec ce dernier, chaque client peut agir simultanément en tant

que serveur, pour répliquer son statut à un pair connecté. En d'autres termes, un client récupère une liste de tous les pairs connectés d'un registre de services et fait toutes les autres demandes à tout autre service via un algorithme d'équilibrage de charge.

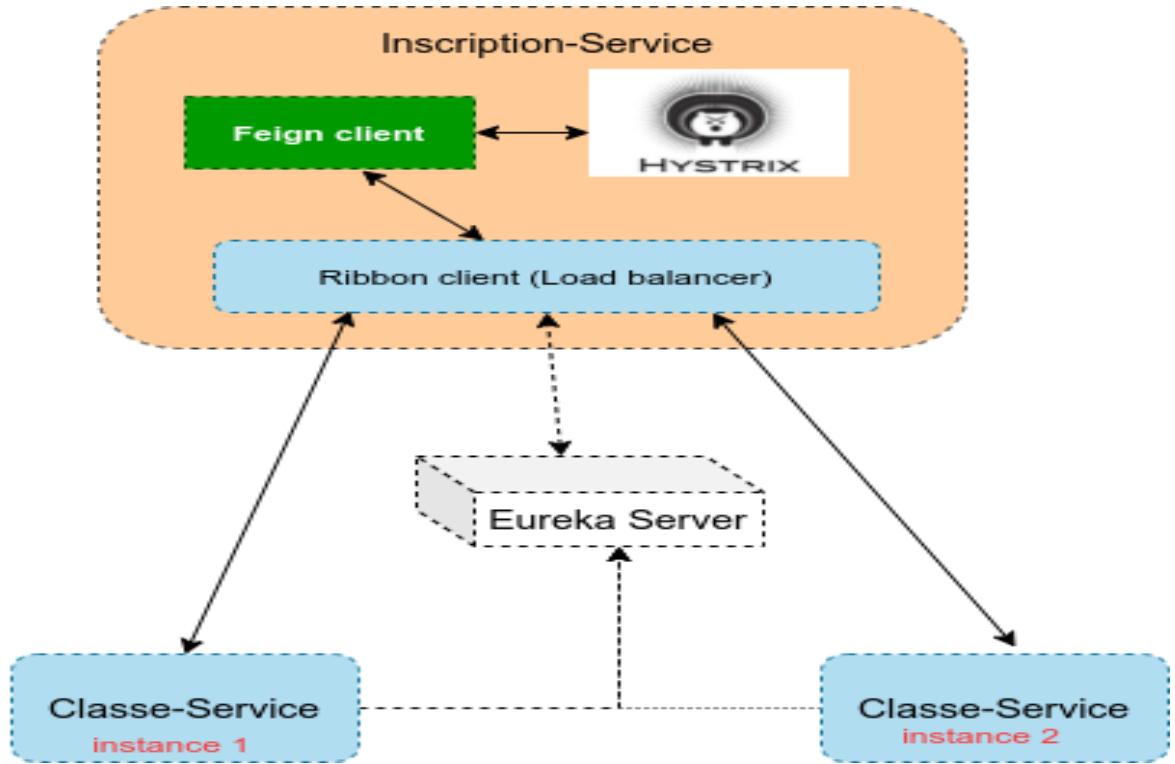


FIG. 5.4 – Communication synchrone entre le microservice de la gestion des inscriptions et celui de la gestion des classes

L'approche de communication synchrone présente certains inconvénients, tels que des délais d'attente et un couplage fort. Par exemple , le service de gestion des inscriptions(Inscription-service) doit attendre la réponse du service de gestion des classes(Classe-service), et le couplage fort signifie que le microservice Inscription-service ne peut pas fonctionner sans que le microservice Classe-service ne soit disponible. Pour éviter ce couplage , nous avons utilisé un disjoncteur de circuit(circuit breaker) avec la bibliothèque **Hystrix**, qui nous permet d'utiliser des replis au cas où le service ne serait pas disponible au moment de l'appel.

Hystrix surveille les échecs d'une méthode et si les échecs atteignent un seuil, il ouvre le circuit afin que les appels suivants échouent automatiquement. Tant que le circuit est ouvert, il redirige les appels vers une méthode de secours spécifiée.

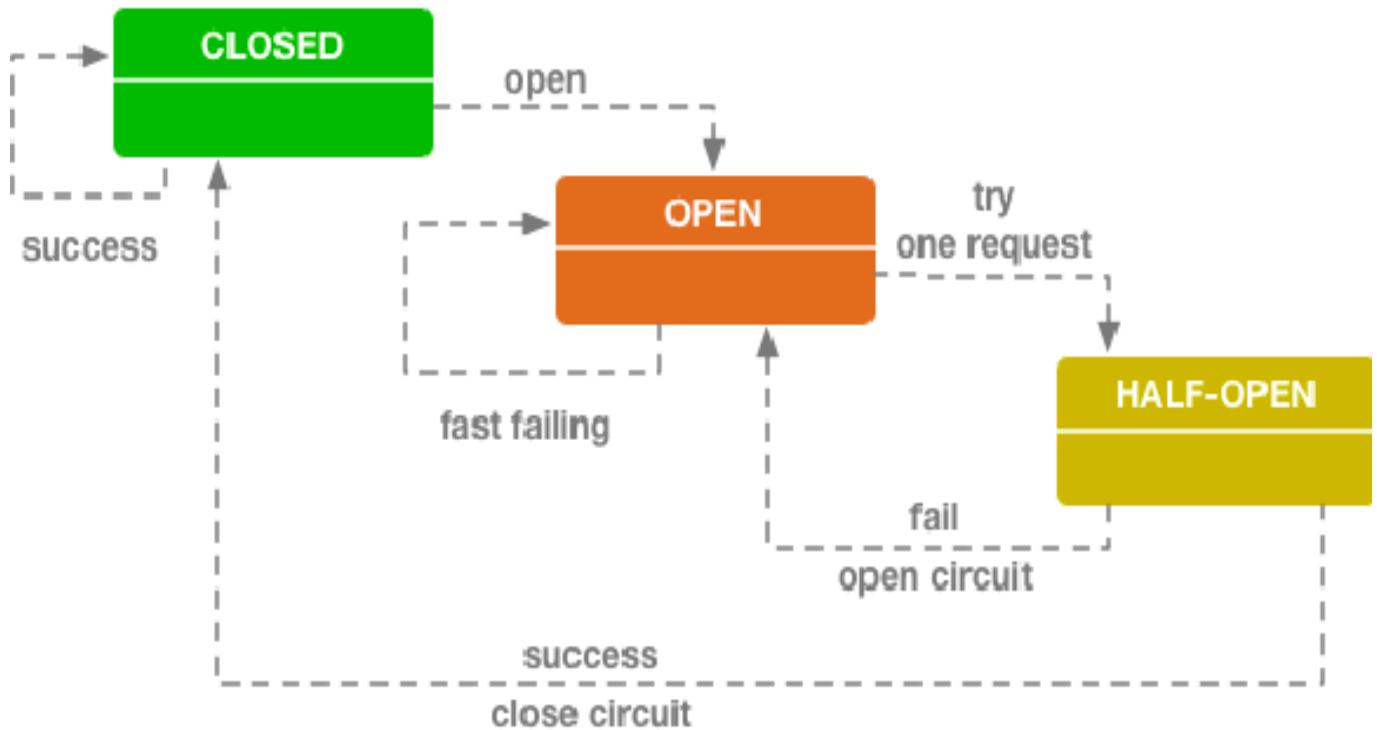


FIG. 5.5 – Diagramme d'état du disjoncteur de circuit

Dans les figures suivantes, on présente les captures du dashboard hystrix. Elles présentent des appels distants du microservice de la gestion des cours vers celui de la gestion des classes(Classe-service).

La figure 5.6 est capturée du dasboard hystrix lorsque le microservice « Classe-service » répond à toutes les demandes provenant du service « Cours-service ». Dans ce cas le circuit breaker est à l'état « **close** » car les appels distants ont réussi.

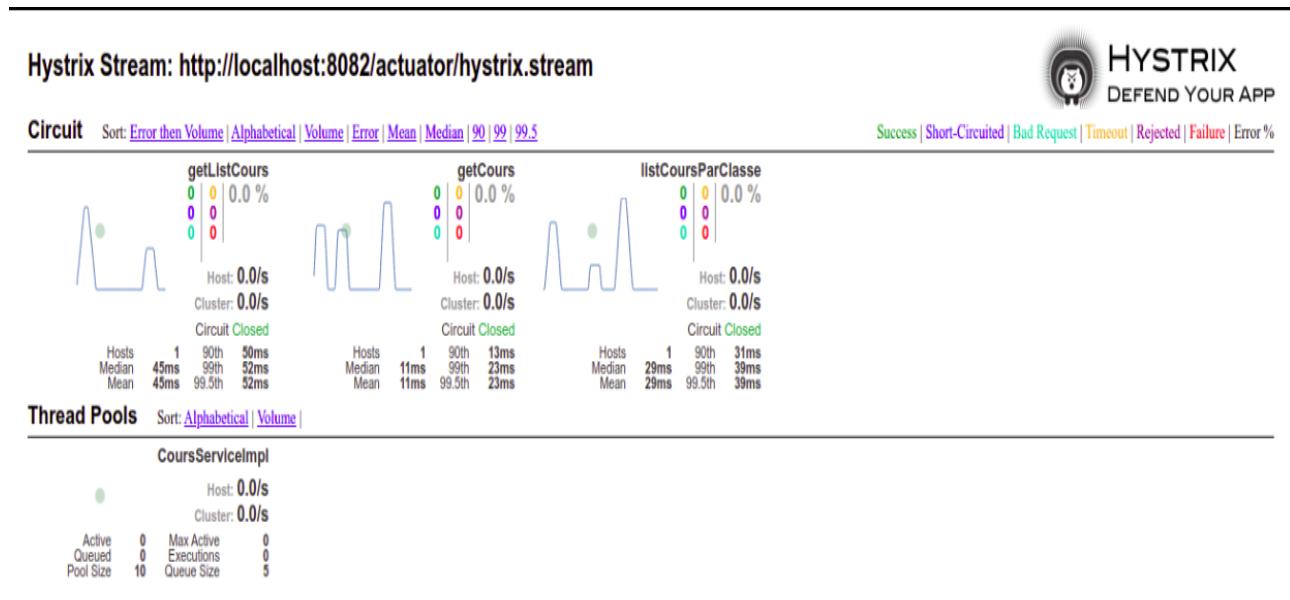
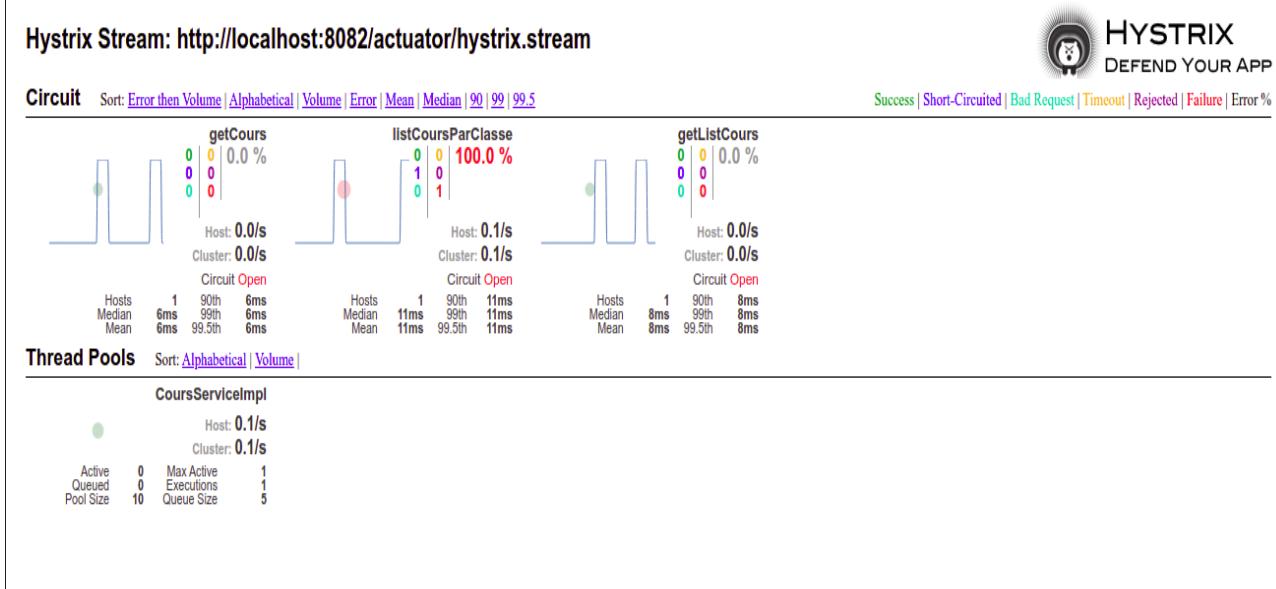


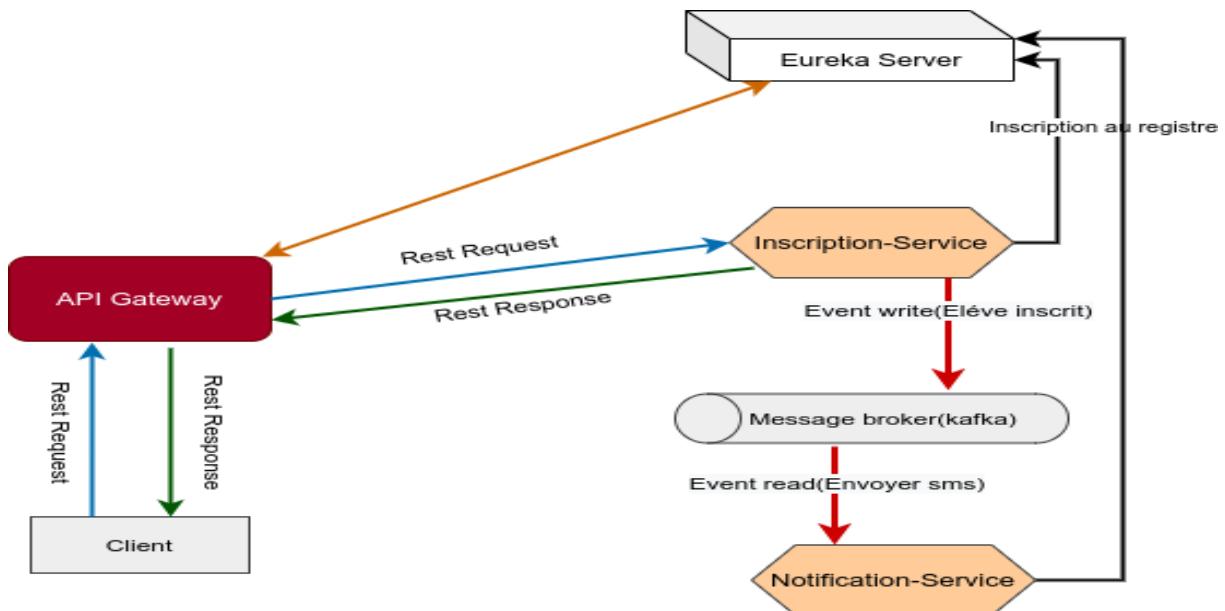
FIG. 5.6 – Hystrix Dashboard avec des appels distants réussis

La figure 5.7 présente le dashboard hystrix lorsque les appels distants du service « Cours-service » vers le service « Classe-service » ont échoué. Le circuit breaker est à l'état « **open** ».

FIG. 5.7 – *Hystrix Dashboard avec des appels distants échoués*

5.3.2 Communication asynchrone basée sur des messages

La communication asynchrone est essentielle lors de la propagation des modifications entre plusieurs microservices. Lors de l'utilisation de la messagerie, les processus communiquent en échangeant des messages de manière asynchrone. Un client fait une commande ou une requête à un service en lui envoyant un message. Si le service doit répondre, il renvoie un message différent au client. Puisqu'il s'agit d'une communication basée sur un message, le client suppose que la réponse ne sera pas reçue immédiatement et qu'il peut n'y avoir aucune réponse. Pour ce mémoire, une approche centrée sur **kafka** est utilisée pour assurer la messagerie asynchrone entre nos différents microservices.

FIG. 5.8 – *Communication asynchrone entre le microservice de la gestion des inscriptions et celui de la gestion des notifications*

Chapitre 6

Déploiement Et Test de Charge

Ce chapitre explique la phase de déploiement de l'application que nous avons développée pour illustrer notre étude sur l'architecture de microservices. Dans ce chapitre, nous allons aussi évaluer les performances de notre application. Nous décrirons les différentes étapes à suivre pour le déploiement de l'application. Et nous ferons un test de charge afin d'évaluer les performances de l'application.

6.1 Déploiement des microservices avec Docker

6.1.1 Présentation de Docker

Docker est une plate-forme logicielle qui vous permet de concevoir, tester et déployer rapidement des applications. Docker intègre le logiciel dans des unités standardisées appelées conteneurs, qui rassemblent tout ce dont il a besoin pour fonctionner, y compris les bibliothèques, les outils système, le code et l'environnement d'exécution. Avec Docker, vous pouvez facilement déployer et mettre à l'échelle des applications dans n'importe quel environnement et être sûr que votre code s'exécutera correctement.

6.1.1.1 Fonctionnement de Docker

La technologie Docker utilise le noyau Linux et des fonctions de ce noyau, telles que les groupes de contrôle et les espaces de noms, pour séparer les processus afin qu'ils puissent s'exécuter de façon indépendante. Cette indépendance reflète l'objectif des conteneurs : exécuter plusieurs processus et applications séparément les uns des autres afin d'optimiser l'utilisation de votre infrastructure tout en bénéficiant du même niveau de sécurité que celui des systèmes distincts.

6.1.1.2 Pourquoi utiliser Docker

Docker vous permet d'envoyer du code plus rapidement, de standardiser les opérations de vos applications, de migrer facilement le code et de faire des économies en améliorant l'utilisation des ressources. Avec Docker, vous pouvez obtenir un projet qui peut s'exécuter de manière fiable n'importe où. Avec sa syntaxe simple, Docker vous donne un contrôle total. Grâce à Docker, il est facile de concevoir et d'exécuter des architectures de microservices distribués, de déployer votre code avec des pipelines standardisés d'intégration et de diffusion continues, de développer des systèmes de traitement des données hautement scalables.

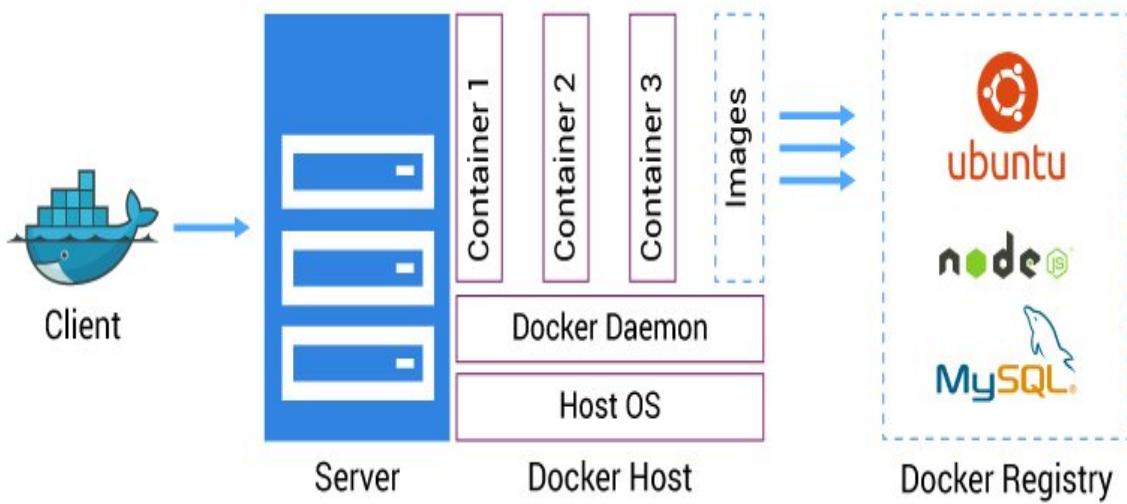


FIG. 6.1 – Architecture docker

6.1.2 PRÉPARATION DES CONTENEURS DE DÉPLOIEMENT

Pour notre application, nous utilisons Docker compose pour démarrer et arrêter plusieurs conteneurs en même temps, tout en définissant les liens entre eux.

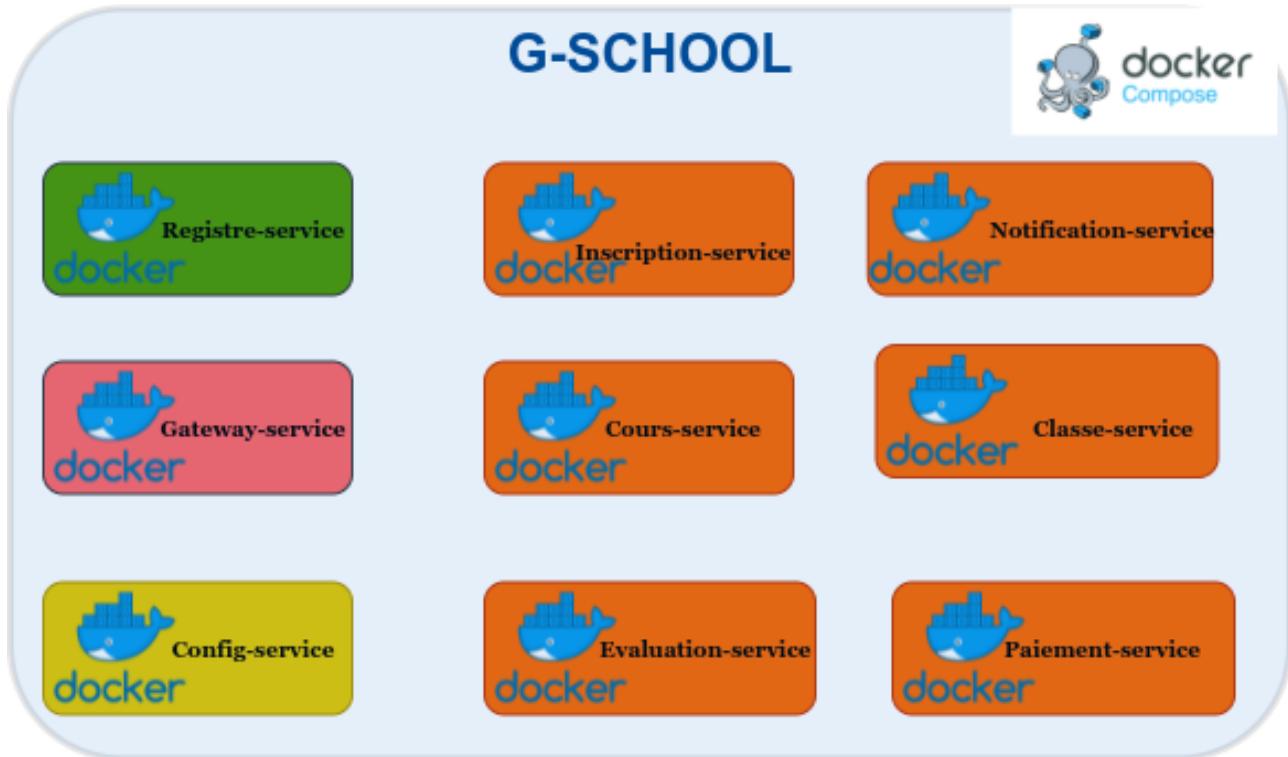
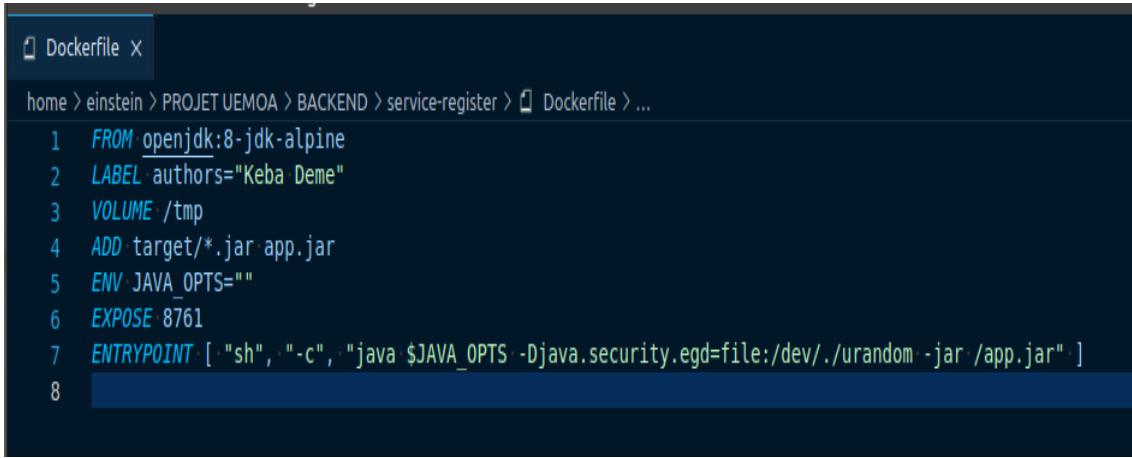


FIG. 6.2 – Conteneurisation de G-school

Les différentes étapes à suivre pour déployer notre application avec docker en utilisant docker compose sont :

1. Pour chaque service, nous allons créer l'image Docker dans laquelle nous allons installer toutes les dépendances de notre projet. Pour cela, nous allons créer un fichier

nommé Dockerfile. Dans ce fichier Dockerfile, vous allez trouverez l'ensemble de la recette décrivant l'image Docker dont vous avez besoin pour votre projet.



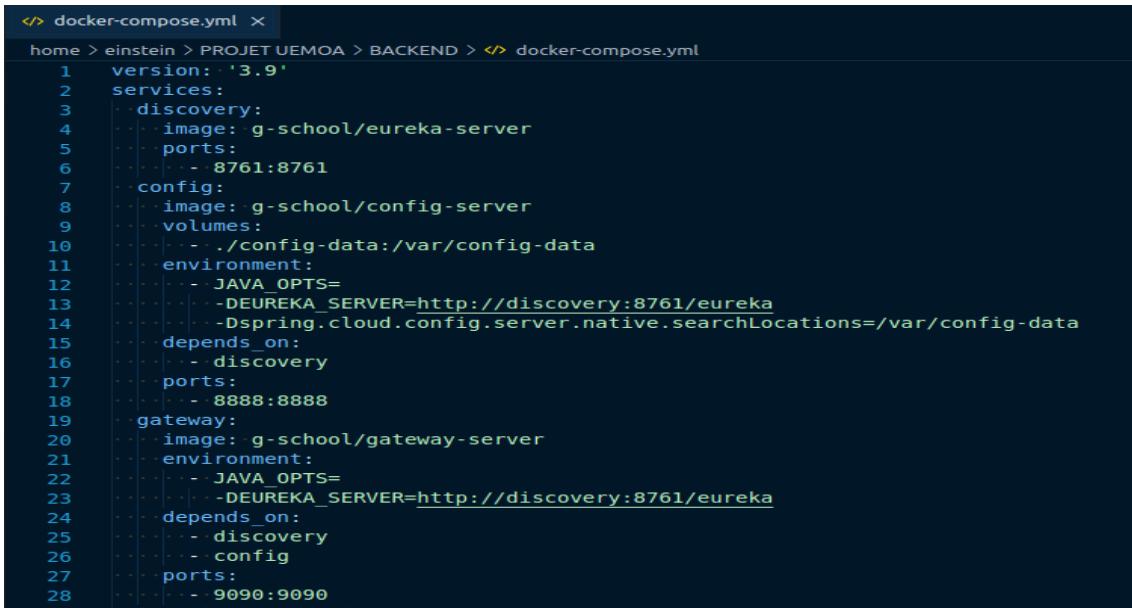
```

Dockerfile X
home > einstein > PROJET UEMOA > BACKEND > service-register > Dockerfile > ...
1 FROM openjdk:8-jdk-alpine
2 LABEL authors="Keba Deme"
3 VOLUME /tmp
4 ADD target/*.jar app.jar
5 ENV JAVA_OPTS=""
6 EXPOSE 8761
7 ENTRYPOINT [ "sh", "-c", "java $JAVA_OPTS -Djava.security.egd=file:/dev/./urandom -jar /app.jar" ]
8

```

FIG. 6.3 – Le fichier Dockerfile de notre service de découverte

- Créer un fichier docker-compose.yml pour définir les services qui composent l'application. Docker Compose est un outil qui permet de décrire (dans un fichier YAML) et gérer (en ligne de commande) plusieurs conteneurs comme un ensemble de services inter-connectés.



```

docker-compose.yml X
home > einstein > PROJET UEMOA > BACKEND > docker-compose.yml
1 version: '3.9'
2 services:
3   discovery:
4     image: g-school/eureka-server
5     ports:
6       - 8761:8761
7     config:
8       image: g-school/config-server
9       volumes:
10      - ./config-data:/var/config-data
11     environment:
12       - JAVA_OPTS=
13       - DEUREKA_SERVER=http://discovery:8761/eureka
14       - Dspring.cloud.config.server.native.searchLocations=/var/config-data
15     depends_on:
16       - discovery
17     ports:
18       - 8888:8888
19   gateway:
20     image: g-school/gateway-server
21     environment:
22       - JAVA_OPTS=
23       - DEUREKA_SERVER=http://discovery:8761/eureka
24     depends_on:
25       - discovery
26       - config
27     ports:
28       - 9090:9090

```

```

home > einstein > PROJET UEMOA > BACKEND > docker-compose.yml
  29   . inscription-service:
  30     . . . image: g-school/inscription-service
  31     . . . environment:
  32     . . . . - JAVA_OPTS=
  33     . . . . . -DEUREKA_SERVER=http://discovery:8761/eureka
  34     . . . depends_on:
  35     . . . . - discovery
  36     . . . . - config
  37     . . . ports:
  38     . . . . - 8081:8081
  39   . cours-service:
  40     . . . image: g-school/cours-service
  41     . . . environment:
  42     . . . . - JAVA_OPTS=
  43     . . . . . -DEUREKA_SERVER=http://discovery:8761/eureka
  44     . . . depends_on:
  45     . . . . - discovery
  46     . . . . - config
  47     . . . ports:
  48     . . . . - 8082:8082
  49   . evaluation-service:
  50     . . . image: g-school/inscription-service
  51     . . . environment:
  52     . . . . - JAVA_OPTS=
  53     . . . . . -DEUREKA_SERVER=http://discovery:8761/eureka
  54     . . . depends_on:
  55     . . . . - discovery
  56     . . . . - config
  57     . . . ports:
  58     . . . . - 8083:8083

home > einstein > PROJET UEMOA > BACKEND > docker-compose.yml
  59   . classe-service:
  60     . . . image: g-school/classe-service
  61     . . . environment:
  62     . . . . - JAVA_OPTS=
  63     . . . . . -DEUREKA_SERVER=http://discovery:8761/eureka
  64     . . . depends_on:
  65     . . . . - discovery
  66     . . . . - config
  67     . . . ports:
  68     . . . . - 8084:8084
  69   . paiement-service:
  70     . . . image: g-school/paiement-service
  71     . . . environment:
  72     . . . . - JAVA_OPTS=
  73     . . . . . -DEUREKA_SERVER=http://discovery:8761/eureka
  74     . . . depends_on:
  75     . . . . - discovery
  76     . . . . - config
  77     . . . ports:
  78     . . . . - 8085:8085
  79   . notification-service:
  80     . . . image: g-school/notification-service
  81     . . . ports:
  82     . . . . - 8086:8086

```

FIG. 6.4 – Le fichier Docker compose des services de notre application

6.1.3 Résultats du déploiement

Les conteneurs de l'application sont déployés dans le cloud de « ionos » dont les caractéristiques du serveur sont : ubuntu 20.04 avec 2 Go de RAM, 80 Go SSD de stockage et un processeur core i7.

Les applications sont accessibles à partir des liens suivants :

- Le service de découverte : <http://74.208.217.250:8761/>
- L'application frontend : <http://74.208.217.250/g-school>

← → C Non sécurisé | 74.208.217.250:8761

Applications Gmail YouTube Maps Traduire How to Deploy... PolyglottePer... Rapport pfe-R... Microservices Liste de lecture

spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	N/A	Current time	2021-11-02T00:22:01 +0000
Data center	N/A	Uptime	34 days 04:29
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	16

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
EVALUATION-SERVICE	n/a (1)	(1)	UP (1) - linux:evaluation-service:8084
PROXY-SERVER	n/a (1)	(1)	UP (1) - linux:proxy-server:9090
SERVICE-COURS	n/a (1)	(1)	UP (1) - linux:service-cours:8082

FIG. 6.5 – Le service de découverte en ligne

← → C Non sécurisé | 74.208.217.250/g-school/login

Applications Gmail YouTube Maps Traduire How to Deploy... PolyglottePer... Rapport pfe-R... Microservices Liste de lecture

Accéder à la plateforme G-School

Nom d'utilisateur

Mot de passe

Se connecter

FIG. 6.6 – L'application frontend en ligne

6.2 Test de charge avec Gatling

Pour valider la tenue de charge pour un certain nombre d'utilisateur, nous allons effectuer des tests de charges sur notre application. L'objectif est de tester la robustesse de notre application développée avec l'architecture de microservices et de vérifier si elle fournit des temps de réponses convenables aux requêtes web des clients tout en mesurant les réserves systèmes dont dispose le serveur.

6.2.1 Scénario à tester

Pour ce scénario, nous allons tester les performances de notre application avec un pic d'utilisation de 1000 utilisateurs simultanés qui sollicitent le service de la gestion des inscriptions. L'utilisateur va:

- Accéder à la page de connexion
- Saisir son nom d'utilisateur et mot de passe pour se connecter
- Une fois connecter, cliquer sur le menu « Élèves » pour afficher la liste des élèves déjà inscrits
- Cliquer sur le bouton « Incrire un élève » pour afficher le formulaire d'inscription
- Renseigner tous les champs du formulaire et cliquer sur sauvegarder pour enregistrer l'inscription

Durée d'exécution du script = 42 minutes.

6.2.2 Résultats obtenus après l'exécution du script de test de charge

Après test, nous avons obtenu les résultats suivants :

- **93 %** des requêtes sont traitées avec un temps de réponse inférieur à **800 ms**
- **2 %** des requêtes ont un temps de réponse compris entre **800 ms et 1200 ms**
- **3 %** des requêtes sont traitées avec un temps de réponse supérieur à **1200 ms** et **2 %** parmi les requêtes exécutées ont échouées

Les graphes suivants donnent plus de détails sur les statistiques.

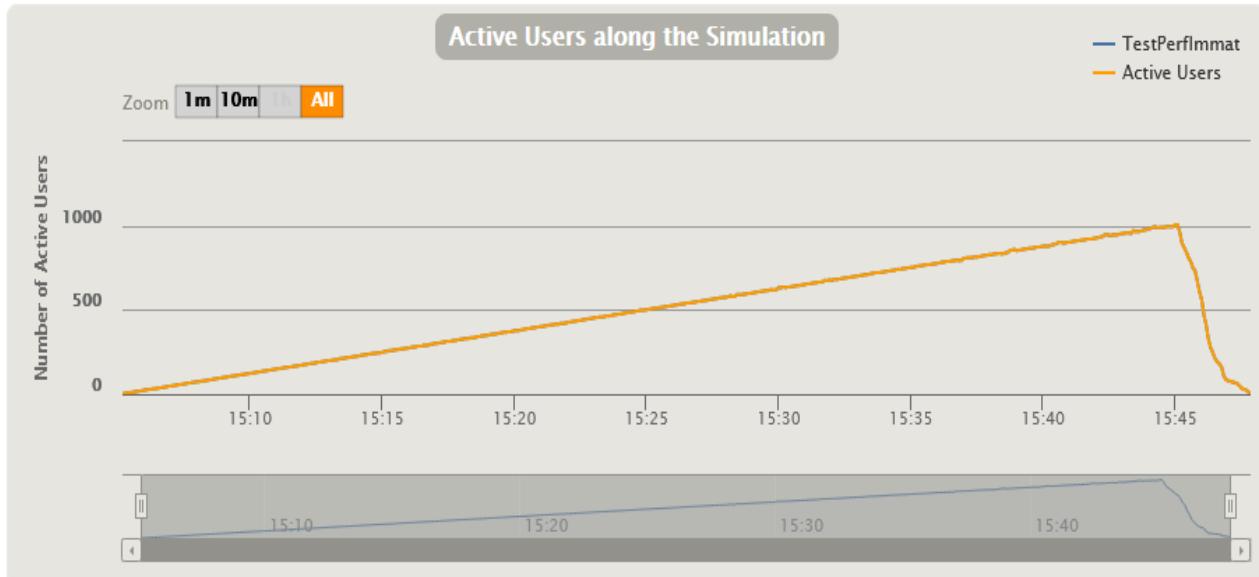


FIG. 6.7 – Evolution du nombre d'utilisateurs actifs en fonction du temps

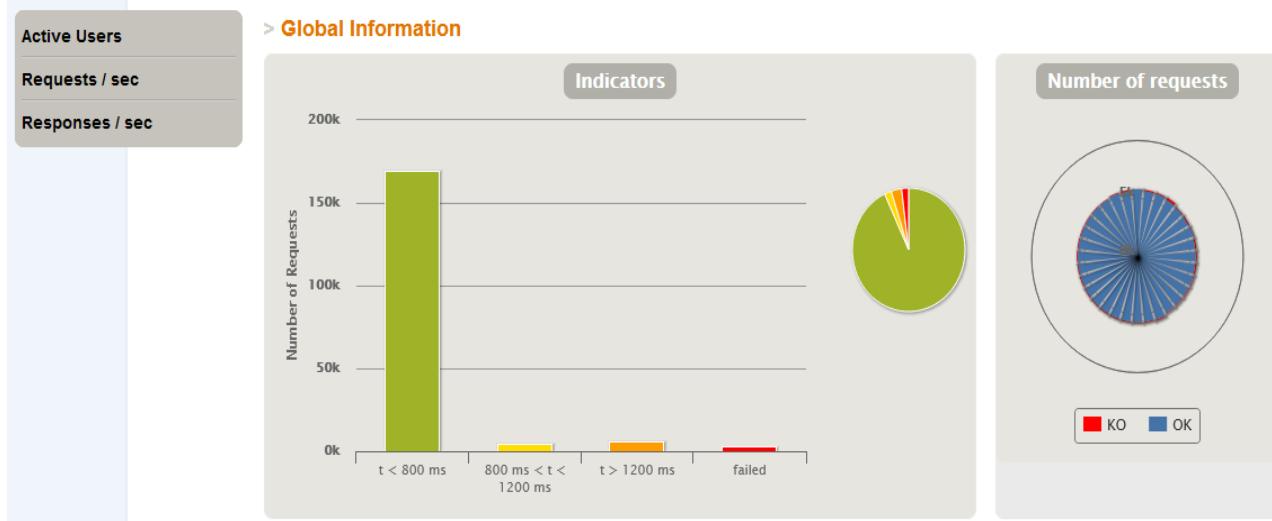


FIG. 6.8 – Distribution des temps de réponse des requêtes exécutées

D'après les résultats obtenus après test, nous pouvons conclure que notre application a de très bonnes performances et peut bel et bien supporter la montée en charge. Plus de 90 % de nos requêtes ont été exécutées en moins d'une seconde.

Conclusion générale

Dans ce mémoire, nous nous sommes particulièrement intéressés à une étude générale sur l'architecture de microservices et son utilisation pour la réalisation des applications cloud natives.

Au cours de cette étude, nous avons d'abord fait un diagnostic sur l'architecture monolithique afin d'identifier ses limites. Par la suite, nous avons étudié les concepts fondamentaux de l'architecture de microservices. Enfin nous avons abouti à la conception et réalisation d'une application de dématérialisation de la gestion scolaire basée sur l'architecture de microservices.

Au terme de ce travail, nous avons remarqué que, avec l'évolution technologique dans ces dernières années, adopter l'architecture de microservices pour la mise en œuvre d'applications d'entreprise n'est plus un sujet de prospective mais bien une réalité surtout que les géants du web la trouvent efficace et fiable.

Ce travail de recherche nous a permis de découvrir tous les concepts liés à l'architecture de microservices et les bonnes pratiques à utiliser pour réaliser des applications basées sur ce style d'architecture.

Bibliographie

- [1] H. A. Anelis Pereira-Vale, Gaston Marquez and E. B. Fernandez. Security mechanisms used in microservices-based systems: A systematic mapping. <https://ieeexplore.ieee.org/abstract/document/9073967>, 2019. [Consulté le 26-Décembre-2020].
- [2] A. BLOG. Introduction to microservices. <https://laptrinhx.com/introduction-to-microservices-3908596879/>, 2018. [Consulté le 16-Janvier-2021].
- [3] C. R. d'Eventuate. Building microservices: Inter-process communication in a microservices architecture. <https://dzone.com/storage/temp/5302608-1.png>, 2015. [Consulté le 20-Février-2021].
- [4] A. P.-V. G. M. H. A. E. B. Fernandez. Security mechanisms used in microservices-based systems: A systematic mapping. <https://ieeexplore.ieee.org/abstract/document/9073967>, 2019. [Consulté le 20-Juin-2020].
- [5] N. S. Gill. Microservices architecture and design patterns. <https://www.xenonstack.com/insights/microservices>, 2018. [Consulté le 11-Mars-2021].
- [6] I. John Wiley Sons. Devops pour les nuls ®, 2e Édition limitée ibm. www.wiley.com., 2015. [Consulté le 14-Mars-2021].
- [7] J. L. M. Fowler. A definition of this new architectural term. <http://martinfowler.com/articles/microservices.html>, 2014. [Consulté le 12-Juin-2020].