

# Rapport projet PFA 2e année

FILIÈRE

## Génie Logiciel

---

# Application e-commerce avec architecture microservice

---

*Réalisé par :*

BRAIKAT Fatima-Ezzahra  
EL KARATI Meryem

*Encadré par :*

Pr. Mahmoud Nassar

Année Universitaire 2023-2024

# Remerciements

Avant d'explorer plus en détail notre expérience, il apparaît approprié de commencer par des remerciements, à ceux qui nous ont enseignées de précieuses leçons tout au long de ce projet, ainsi qu'envers ceux qui ont généreusement contribué à faire de cette période un moment extrêmement bénéfique et enrichissant.

Nous tenons à exprimer nos plus sincères remerciements à **Pr. Mahmoud Nassar**, pour avoir gracieusement accepté d'encadrer notre projet de fin d'année et avoir apporté toute l'aide nécessaire pour mener à bien ce projet.

Nous remercions également toute l'équipe pédagogique de l'École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS) pour avoir assuré une formation de haute qualité et, plus particulièrement, à notre chef de filière **Pr. Bouchra EL ASRI**, ainsi qu'à la responsable de gestion des projets PFA **Pr. Bouchra BERRADA** pour nous avoir permises d'entreprendre cette opportunité de projet si intéressante. Veuillez trouver ici le témoignage de notre respect le plus profond.

Que tous ceux qui ont contribué de près ou de loin à l'aboutissement de ce projet trouvent l'expression de mes remerciements les plus sincères.

# Résumé

Le présent document synthétise notre travail effectué au titre du projet de fin de 2ème année, qui s'intitule « Mise en place d'une application e-commerce avec architecture microservice ».

Ce projet a pour mission la conception et l'implémentation d'une application de commerce électronique, en exploitant une architecture microservice.

Pendant le développement des différentes briques du projet, nous nous sommes principalement appuyés sur concepts de bases et les patterns des microservices.

---

**Mots clés :** Microservices, Développement Web, API REST, Patterns.

---

# Abstract

The present document synthesizes our work carried out as part of the second-year final project, titled "Implementation of an E-commerce Application with Microservice Architecture."

The mission of this project is to design and implement an e-commerce application utilizing a microservice architecture.

During the development of the various components of the project, we primarily relied on the fundamental concepts and patterns of microservices.

---

**Keywords :** Microservices, Web Development, REST API, Patterns.

---

# Liste des abréviations

**API** *Application Programming Interface*

**AMQP** *Advanced Message Queuing Protocol*

**DDD** *Domain-Driven Design*

**JWT** *JSON Web Token*

# Table des figures

2.1	Le diagramme de classes . . . . .	7
3.1	Architecture à trois niveaux . . . . .	9
3.2	Encapsulation des composantes d'un service . . . . .	10
3.3	Patterns de microservices . . . . .	12
3.4	Le Cube de Scalabilité . . . . .	12
3.5	Décomposition de services . . . . .	14
3.6	Base de données par service . . . . .	15
3.7	Registre de services . . . . .	15
3.8	Passerelle API . . . . .	16
3.9	Traçage distribué . . . . .	17
3.10	File d'attente de messages . . . . .	18
3.11	Architecture e-commerce . . . . .	18
4.1	DCU du service Client . . . . .	21
4.2	Diagramme de classe du service Client . . . . .	22
4.3	DCU du service Product . . . . .	22
4.4	Diagramme de classes du service Product . . . . .	23
4.5	DCU du service Order . . . . .	23
4.6	Diagramme de classes du service Order . . . . .	24
4.7	DCU du service Notification . . . . .	24
4.8	Diagramme de classe du service Notification . . . . .	25
4.9	Diagramme de classes du service Payment . . . . .	25
4.10	Diagramme de séquence du service Eureka . . . . .	26
4.11	Diagramme de séquences du service Gateway . . . . .	27
4.12	DCU du service Auth . . . . .	27
4.13	Diagramme de classes du service Auth . . . . .	28
4.14	Diagramme de séquences pour l'inscription . . . . .	28
4.15	Diagramme de séquences pour l'authentification . . . . .	29
5.1	Logos de Spring Boot & Spring Cloud . . . . .	31
5.2	ReactJs . . . . .	31
5.3	Serveur Eureka . . . . .	32
5.4	Schéma éclaircissant la fonction d'un server registry . . . . .	32
5.5	Spring Cloud Gateway . . . . .	33
5.6	Routing de services dans API Gateway . . . . .	33
5.7	Spring Security & JWT . . . . .	34
5.8	Exemple d'un token JWT . . . . .	34
5.9	Service Zipkin . . . . .	35

## Table des figures

---

5.10 Feign Client . . . . .	36
5.11 Fonctionnement de RabbitMQ . . . . .	36
5.12 Interface de RabbitMQ . . . . .	37
5.13 Page d'accueil . . . . .	38
5.14 Modale de Login . . . . .	38
5.15 Page du catalogue de produits . . . . .	39
5.16 Page des détails d'un produit . . . . .	39
5.17 Modale de confirmation d'achat . . . . .	40
5.18 Accès à la page d'admin pour l'administrateur . . . . .	40
5.19 Page de l'administrateur . . . . .	41
5.20 Page de gestion des clients . . . . .	41
5.21 Page de gestion des produits . . . . .	42
5.22 Page de gestion des catégories . . . . .	42

# Table des matières

<b>Remerciements</b> . . . . .	<b>II</b>
<b>Résumé</b> . . . . .	<b>III</b>
<b>Abstract</b> . . . . .	<b>IV</b>
<b>Introduction générale</b> . . . . .	<b>1</b>
<b>1 Présentation générale du projet</b> . . . . .	<b>2</b>
1.1 Introduction . . . . .	2
1.2 Contexte et motivation . . . . .	3
1.3 Objectifs du projet . . . . .	3
1.4 Conclusion . . . . .	4
<b>2 Analyse des besoins et conception</b> . . . . .	<b>5</b>
2.1 Introduction . . . . .	5
2.2 Analyse fonctionnelle . . . . .	6
2.3 Analyse non fonctionnelle . . . . .	6
2.4 Utilisateurs cibles . . . . .	6
2.5 Diagramme de classe - architecture monolithique	7
2.6 Conclusion . . . . .	7
<b>3 Architecture microservices</b> . . . . .	<b>8</b>
3.1 Introduction . . . . .	8
3.2 Introduction aux microservices . . . . .	9
3.2.1 Architecture monolithique . . . . .	9
3.2.2 De la Monolithique vers les Microservices . . . . .	10
3.2.3 Avantages et inconvénients des microservices . . . . .	11
3.3 Concepts fondamentaux en microservices . . . . .	12
3.3.1 Scalabilité . . . . .	12
3.3.2 Décomposition . . . . .	13
3.3.3 Base de données par service : . . . . .	14
3.3.4 Le registre de service : . . . . .	15
3.3.5 Passerelle d'API . . . . .	16
3.3.6 Le traçage distribué . . . . .	16
3.3.7 La file d'attente de messages . . . . .	17
3.4 Architecture finale . . . . .	18
3.5 Conclusion . . . . .	19

## Table des matières

---

<b>4 Conception de chaque microservice . . . . .</b>	<b>20</b>
4.1 Introduction . . . . .	20
4.2 Microservices fonctionnels . . . . .	21
4.2.1 Client-Service . . . . .	21
4.2.2 Product-Service . . . . .	22
4.2.3 Order-Service . . . . .	23
4.2.4 Notification-Service . . . . .	24
4.2.5 Payment-Service . . . . .	25
4.3 Microservices d'infrastructure . . . . .	25
4.3.1 Eureka-Server . . . . .	26
4.3.2 Gateway-Service . . . . .	26
4.3.3 Auth-Service . . . . .	27
4.4 Conclusion . . . . .	29
<b>5 Réalisation . . . . .</b>	<b>30</b>
5.1 Introduction . . . . .	30
5.2 Frameworks . . . . .	31
5.2.1 Spring boot & Spring cloud . . . . .	31
5.2.2 ReactJs . . . . .	31
5.3 Pour la conception microservices . . . . .	32
5.3.1 Registre de services : Eureka . . . . .	32
5.3.2 API Gateway : Spring Cloud Gateway . . . . .	33
5.3.3 Sécurité : Spring Security & Tokens JWT . . . . .	34
5.3.4 Traçage distribué : Sleuth & Zipkin . . . . .	35
5.3.5 Communication entre services . . . . .	35
5.4 Interfaces . . . . .	37
5.5 Conclusion . . . . .	42
<b>Conclusion et perspectives . . . . .</b>	<b>43</b>

# Introduction générale

Dans le cadre de notre deuxième année en génie logiciel à l'Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS), nous réalisons notre projet de fin d'année qui nous permettra de mettre en pratique tous les concepts et notions acquis lors de l'année. Ce projet nous permettra d'apprendre et de maîtriser de nouvelles compétences, au travers d'un cahier de charges ayant pour objectif la conception et le développement d'une application e-commerce en exploitant l'architecture de microservices.

Ce rapport trace les différentes étapes suivies pour la réalisation de projet. Il est divisé en 4 parties :

- **Partie 1:** Présentation générale du projet.
- **Partie 2:** Analyse des besoins et conception.
- **Partie 3:** Architecture en microservices.
- **Partie 4:** Réalisation.

# **Chapitre 1**

## **Présentation générale du projet**

### **1.1 Introduction**

Dans ce chapitre, nous nous intéresserons à la description de notre projet. Nous présenterons le contexte ainsi que la motivation derrière le choix de ce sujet. Nous détaillerons par la suite les principaux objectifs du projet, pour enfin aboutir à une conclusion.

## **1.2 Contexte et motivation**

Depuis les années 2000 et l'essor d'Internet, le mode de consommation à travers le monde a subi une transformation profonde. Le e-commerce s'est développé de manière exponentielle, avec une multiplication des détaillants et des commerçants en ligne.

Ordinateur, mobile, tablette, ... Les canaux de ventes se multiplient et le commerce s'invite dans notre vie quotidienne, offrant aux consommateurs une expérience d'achat sans contrainte de temps ni de lieu.

Ceci a entraîné une grande croissance dans le secteur du commerce électronique, avec de plus en plus d'entreprises investissant dans leurs plate-forme de vente en ligne. La concurrence s'intensifie alors que les entreprises cherchent à capturer une part du marché en expansion et à répondre aux besoins changeants des consommateurs. Pour rester compétitives, les entreprises affinent leurs stratégies de commerce électronique, en se concentrant sur des éléments tels que l'expérience utilisateur, la personnalisation, la logistique efficace et le service clientèle de qualité. Cette évolution rapide du paysage du commerce électronique présente à la fois des opportunités et des défis pour les entreprises, les incitant à innover et à s'adapter rapidement pour réussir sur ce marché en constante évolution. [1]

Avec une augmentation continue du nombre d'utilisateurs et de transactions, beaucoup d'entreprises ont remarqué que leurs systèmes ont commencé à rencontrer plusieurs problèmes de performance et d'évolutivité, devenant plus complexes et difficiles à maintenir à mesure que les applications grossissent. Il était devenu clair que ce style d'architecture traditionnel n'était plus suffisant pour supporter la croissance considérable que subit ce secteur.

C'est là où interviennent les microservices. Introduits en 2014 par Lewis et Fowler dans leur fameux article [2], l'architecture en microservices apportent une approche décentralisée, en découplant les services et divisant les applications en plus petits composants indépendants flexibles et faciles au déploiement et à la maintenance, poussant de nombreuses entreprises à migrer vers cette architecture. [3]

## **1.3 Objectifs du projet**

Notre projet vise à employer les microservices dans le contexte d'une application e-commerce. Parmi ses objectifs :

- Bien exploiter les concepts derrière les microservices dans le cadre de l'e-commerce, notamment la modularité, la scalabilité, la flexibilité et la facilité de déploiement des applications ;
- Passer de la conception monolithique vers les microservices ;
- Assurer une expérience utilisateur cohérente. Les services déployés et gérés indépendamment doivent fonctionner de manière transparente ensemble, sans que les clients ne perçoivent de différences ou de discontinuités entre les différentes fonctionnalités.

## **1.4 Conclusion**

Ceci conclut notre vue en plan du sujet en question. Nous pourrons donc entamer ce projet par une étude plus détaillée de l'architecture de microservices et des composantes du système à développer.

# **Chapitre 2**

## **Analyse des besoins et conception**

### **2.1 Introduction**

Le chapitre d'analyse et de conception représente une étape fondamentale dans le processus de développement de notre projet d'application e-commerce. Avant de plonger dans la réalisation concrète de notre application, il est crucial de mener une analyse approfondie des besoins et des exigences du projet, ainsi que de concevoir une architecture logicielle adaptée pour répondre à ces besoins. Dans cette section, nous examinerons en détail les différentes étapes de l'analyse et de la conception de notre application e-commerce, en mettant en lumière les méthodologies utilisées, les modèles de conception envisagés et les choix architecturaux pris pour garantir le succès de notre projet.

## 2.2 Analyse fonctionnelle

Les besoins fonctionnels principaux de notre application e-commerce pourraient inclure :

- Tout visiteur peut visualiser le catalogue de produits proposés.
- Tout visiteur de la plateforme peut s'inscrire pour devenir un client.
- Tout client peut s'authentifier.
- Un client authentifié peut faire des commandes.
- Un administrateur peut se connecter au tableau de bord.
- Un administrateur peut gérer les produits, catégories et clients.

## 2.3 Analyse non fonctionnelle

L'analyse non fonctionnelle de l'application "Let's Shop" pourrait comprendre :

- **Performance** : Mesure du temps de réponse des pages, capacité à gérer un grand nombre d'utilisateurs simultanés.
- **Sécurité** : Protocoles de sécurité utilisés (inscriptions, authentification, autorisation).
- **Fiabilité** : Disponibilité du système, gestion des pannes.
- **Extensibilité** : Facilité d'ajout de nouvelles fonctionnalités et de nouveaux services.
- **Maintenabilité** : Facilité de maintenance et de mise à jour du code et des micro-services.

## 2.4 Utilisateurs cibles

Les utilisateurs cibles de l'application "Let's Shop" sont :

- **Clients finaux** : Personnes souhaitant acheter des produits en ligne.
- **Administrateurs** : Personnes responsables de la gestion des produits, des commandes et des utilisateurs.
- **Visiteurs** : Clients potentiels

## 2.5 Diagramme de classe - architecture monolithique

Le diagramme de classe ci-dessous est le résultat de notre analyse fonctionnelle pour une première approche monolithique où tout est organisé en une seule place.

L'entité "User" représente les comptes des utilisateurs de la plateforme, avec chaque utilisateur ayant un ou plusieurs rôles définis par l'entité "Role". Un "Client" peut avoir un compte utilisateur. A tout moment, un client peut décider d'acheter un produit avec une quantité spécifique, l'opération sera donc enregistrée en tant que "Order" qui référence les entités "Product" (Contenant les informations sur les produits proposés) et le client lui-même. Enfin, chaque produit appartient à une catégorie spécifique définie dans "Category". Chaque client peut avoir une à plusieurs cartes de paiements, et une commande peut contenir plusieurs transactions (paiement, remboursement, ...)

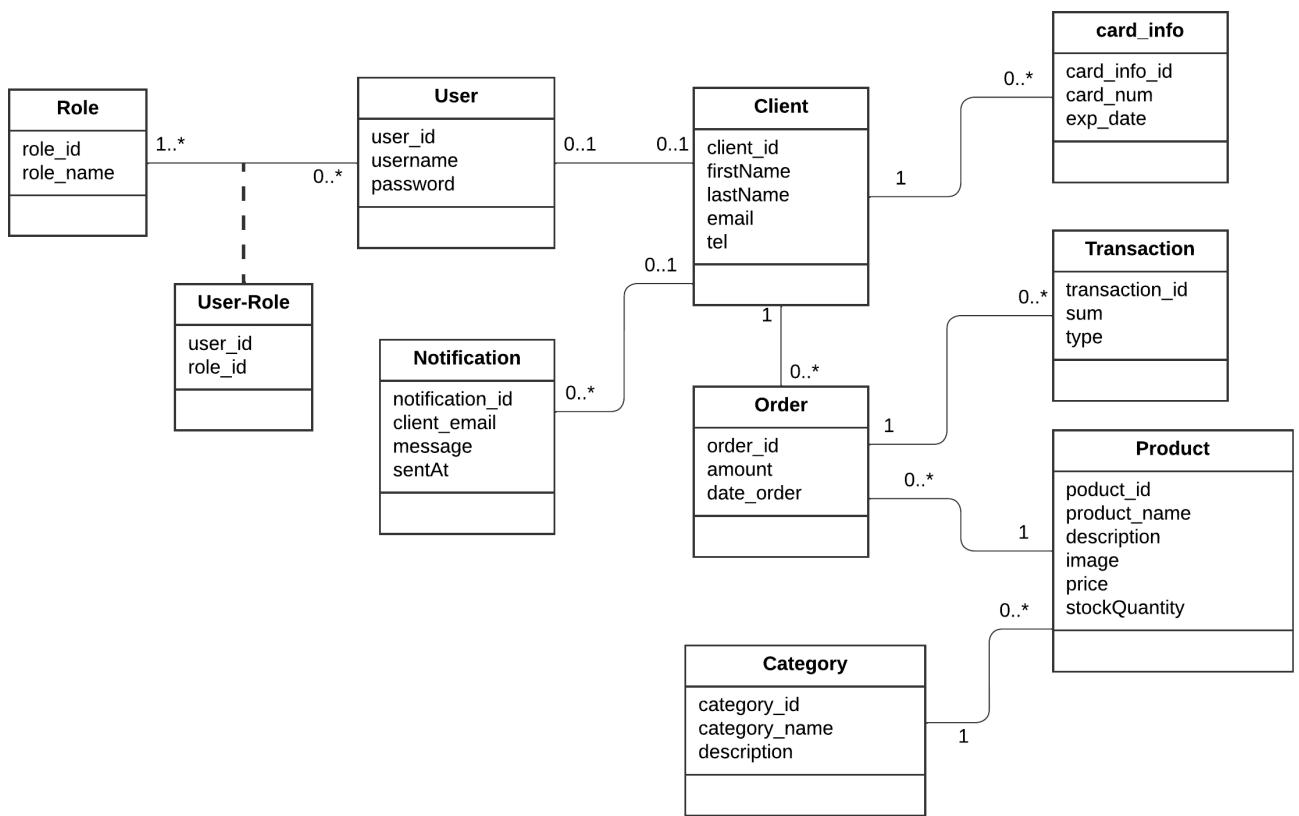


FIG. 2.1 : Le diagramme de classes

## 2.6 Conclusion

En conclusion, l'analyse des besoins pour l'application "Let's Shop" a mis en lumière les exigences fonctionnelles et non fonctionnelles essentielles à son développement. Les besoins fonctionnels et non fonctionnels ainsi que les utilisateurs cibles ont été identifiés.

# Chapitre 3

## Architecture microservices

### 3.1 Introduction

Le chapitre consacré à l'étude des systèmes microservices constitue une étape essentielle dans notre projet de développement d'application e-commerce. Il vise à explorer en profondeur les principes, les avantages et les défis des systèmes basés sur des microservices, tout en évaluant leur pertinence et leur applicabilité pour notre projet. En examinant les concepts fondamentaux, les bonnes pratiques et les études de cas pertinents, nous serons mieux équipés pour prendre des décisions éclairées concernant l'architecture de notre application e-commerce et pour naviguer efficacement dans la mise en œuvre de cette approche innovante.

## 3.2 Introduction aux microservices

Les microservices sont des services indépendamment déployables modélisés autour d'un domaine métier. Ils communiquent entre eux via des réseaux, et en tant que choix d'architecture, offrent de nombreuses options pour résoudre les problèmes auxquels nous pourrions être confrontés. Il en découle qu'une architecture de microservices est basée sur plusieurs services collaborant entre eux. C'est un type d'architecture orientée services (Service Oriented Architecture - SOA), avec chaque service communiquant via des réseaux, ce qui en fait une forme de **système distribué**.

### 3.2.1 Architecture monolithique

Avant de parler des microservices, il faut commencer par introduire l'architecture monolithique, pour ensuite mieux comprendre l'intérêt de choisir les microservices.

Un *monolithe* est une application où toutes ses parties sont regroupées et déployées en tant qu'unité unique, généralement dans un seul processus. Ainsi, tous les aspects de l'application et son code sont implémentés en tant que composants interdépendants et exécutés dans le même processus. Il existe en fait trois types de systèmes monolithiques : le système à processus unique (single-process system), le monolithe distribué (distributed monolith) et les systèmes boîte noire de tiers (third-party black-box systems).

En prenant l'exemple d'un type d'architecture très connu et simple à concevoir ; **l'architecture à trois niveaux** (three-tiered architecture). C'est un modèle d'architecture qui divise une application en trois couches distinctes :

- La couche présentation
- La couche logique ou fonctionnelle
- La couche données

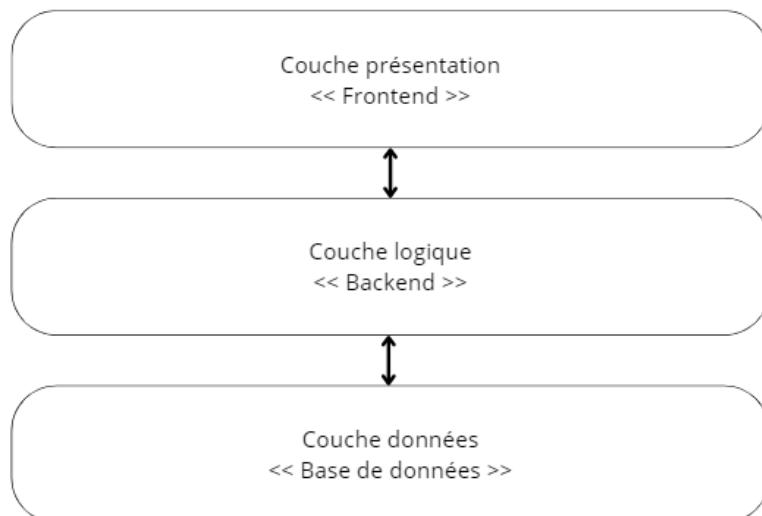


FIG. 3.1 : Architecture à trois niveaux

En appliquant le style monolithique, toutes les composantes du code pour chacune des deux couches Frontend et Backend sont capsulées dans leur propre unité à déployer. Seulement, si on désire ajouter une fonctionnalité qui affectera les trois couches, il faudra appliquer du changement dans chacune des ces couches, tout en assurant le fonctionnement correct de l'application en perpétuité.

Dans notre cas, si on dispose d'une application pour e-commerce, et qu'on désire par exemple ajouter une fonctionnalité pour que le client puisse ajouter des produits à ses "préférés", il faudra inclure une nouvelle section dans l'interface UI, ajouter les fonctions métier niveau Backend, et même ajuster la base de données pour stocker les informations associées à cette fonctionnalité.

### 3.2.2 De la Monolithique vers les Microservices

Une méthode alternative qui pourrait améliorer l'organisation du code serait si on le divisait selon les fonctionnalités métier. Comme le montre la figure ci-dessous, chaque service aurait sa propre architecture à trois niveaux, avec sa propre interface UI, ses fonctions métiers et ses données concernées. De cette manière, l'ajout d'une fonctionnalité concernant seulement le client, comme par exemple ses produits préférés, impacterait seulement le service client, et le déploiement des changements peut se faire uniquement au niveau du client, au lieu de toute l'application.

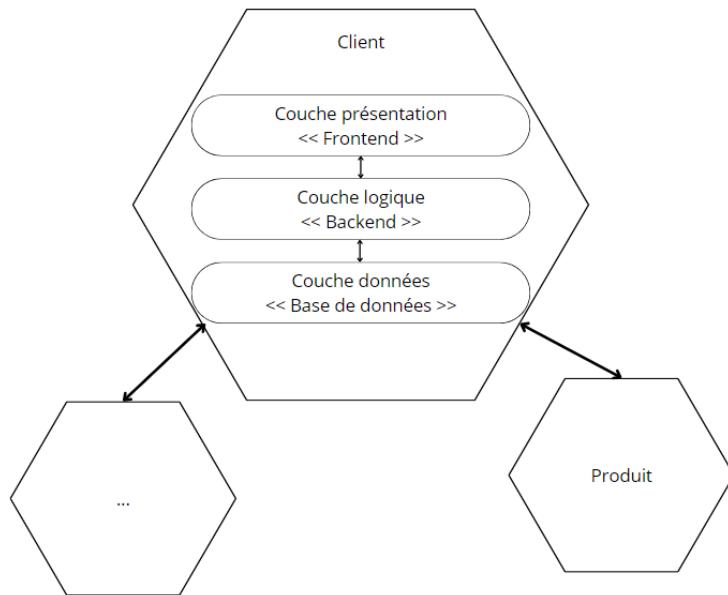


FIG. 3.2 : Encapsulation des composantes d'un service

### 3.2.3 Avantages et inconvénients des microservices

L'approche de microservices apporte plusieurs **avantages** :

- En divisant l'application en plusieurs unités ou services autonomes et évolutifs, ceci nous donne plus de liberté pour améliorer l'échelle et la robustesse du système.
- Il est possible de combiner plusieurs différentes technologies selon le besoin spécifique pour chaque service sans avoir à se soucier de conflits, vu que chaque service possède ses propres dépendances et technologies isolées des autres.
- On a une énorme flexibilité, et une plus grande facilité quant à la manière dont nous pouvons résoudre les problèmes, vu que nous pouvons concentrer notre attention sur une seule partie du code pour détecter les failles.

Cependant, cette approche ne reste pas sans ses propres failles. En effet, il existe, quand même, un ensemble d'**inconvénients** engendrés par la décomposition du système :

- L'un des principaux défis qu'on peut mentionner est le moyen de communication entre ces services : les **réseaux**. Surtout dans les systèmes très complexes ou très distribués, les appels réseaux peuvent introduire des latences et des goulets d'étranglement.
- Avec des données réparties entre plusieurs microservices, maintenir la **cohérence des données**, leur synchronisation, mais aussi assurer les propriétés ACID peuvent représenter un défi.
- Assurer l'intégration et la compatibilité entre les différents microservices peut être difficile, surtout si chaque microservice est développé et déployé de manière indépendante. Des stratégies d'interopérabilité et de communication bien définies sont nécessaires.

### 3.3 Concepts fondamentaux en microservices

Lors de la conception de nos différents microservices, il est essentiel de prendre en compte certains aspects cruciaux afin de garantir une conception optimale.

Voici un schéma représentant quelques motifs ou patterns utilisés souvent dans un contexte de microservices.

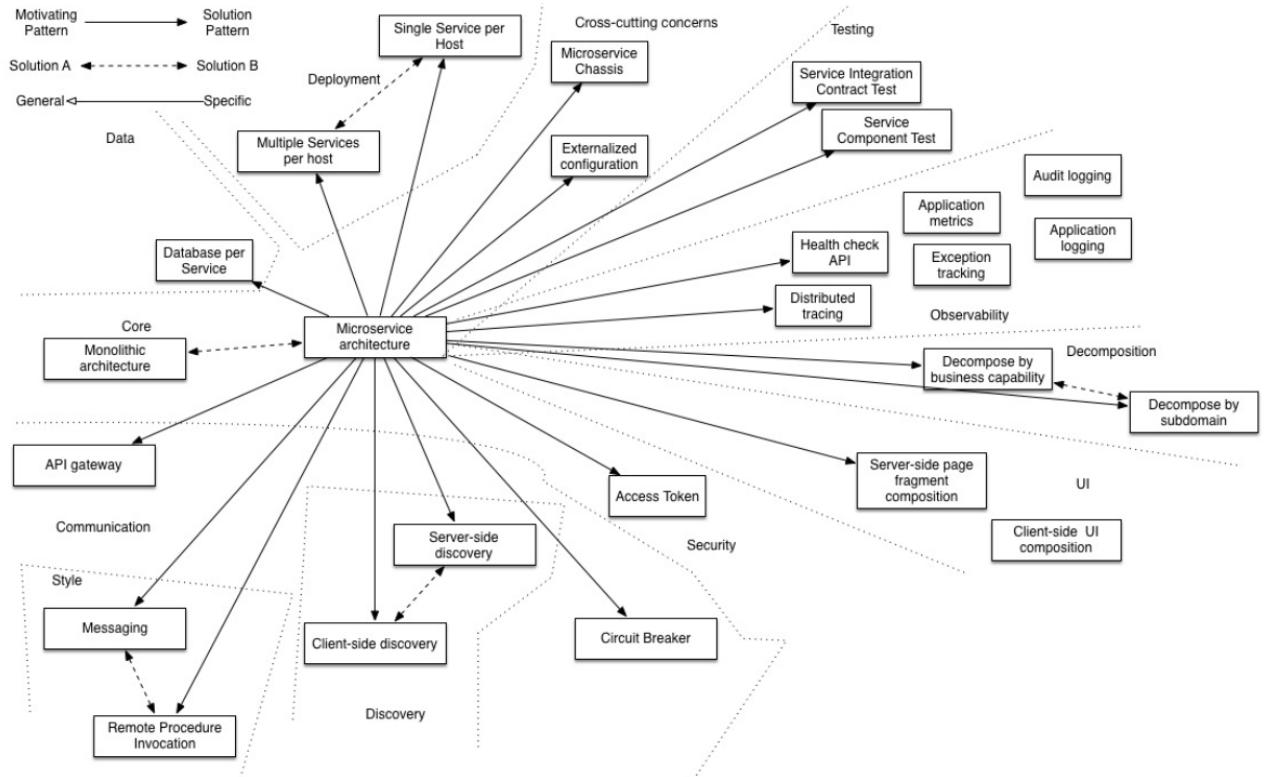


FIG. 3.3 : Patterns de microservices

#### 3.3.1 Scalabilité

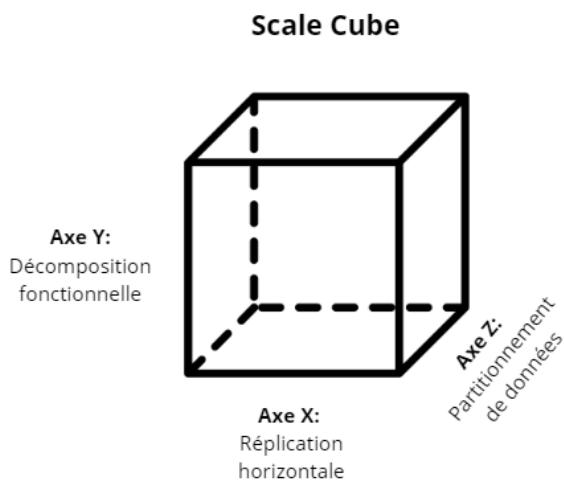


FIG. 3.4 : Le Cube de Scalabilité

Comme représenté ci-dessus, le "Scale Cube" (ou "Cube de Scalabilité") est un modèle conceptuel développé par Michael Fisher et Martin Abbott pour décrire les différentes dimensions de la scalabilité d'un système. Ce modèle propose trois axes de scalabilité, représentés sous forme de cube, d'où son nom. Chaque axe correspond à une dimension spécifique de la scalabilité :

- **Axe X : Scalabilité horizontale**

Lorsque la demande des utilisateurs augmente, le système doit être capable de faire évoluer automatiquement le nombre d'instances de microservices pour répondre à cette demande croissante en dupliquant les composants autant que nécessaire. Il existe des techniques de *Load Balancing* qui assurent cela.

- **Axe Y : Scalabilité verticale**

Il faut assurer la décomposition de l'application en différents modules fonctionnels ou fonctionnalités, chacun étant responsable d'une partie spécifique du système. Chaque service est conçu pour être hautement spécialisé et peut être mis à l'échelle de manière indépendante des autres services. Cela permet une meilleure isolation des fonctionnalités et une évolutivité plus fine.

- **Axe Z : Partitionnement des données** Partitionnement ou fragmentation des données de l'application afin de répartir la charge de travail sur plusieurs bases de données. Cela peut être réalisé en partitionnant les données en fonction de certains critères ou en utilisant des bases de données spécifiques à chaque microservice. Ceci permet de réduire les goulets d'étranglement et d'améliorer les performances de l'application.

### 3.3.2 Décomposition

Il existe plusieurs patterns de décomposition en microservices qui peuvent aider à structurer et à diviser une application en composants plus petits et indépendants. Vu que notre projet a pour but de développer une simple application e-commerce, nous avons opté pour l'usage de deux de ces patterns :

#### 1. Décomposition par capacité métier :

Chaque microservice correspond à une fonction métier distincte et autonome.

- **Auth Service** : Gestion des inscriptions et authentification des utilisateurs ainsi que la gestion de leurs rôles. Maintient la sécurité d'accès aux microservices.
- **Product Service** : Gestion des produits, leur création, mise à jour, et la consultation du catalogue de produits.
- **Client Service** : Gestion des informations de clients.
- **Order Service** : Gestion des commandes, leur suivi, et l'historique.
- **Payment Service** : Traitement des paiements et transactions.

## 2. Décomposition par sous-domaine :

En utilisant les principes de Domain-Driven Design (DDD), la décomposition en microservices peut également être vue comme une division basée sur les sous-domaines de notre domaine métier. Chaque microservice est responsable d'un sous-domaine spécifique, ce qui permet de gérer de manière isolée les aspects fonctionnels distincts de l'application e-commerce.

En plus des services déjà mentionnées, nous disposerons aussi d'un service **notification** où seront traitées des messages à chaque fois qu'une opération importante est effectuée au sein de la plateforme.

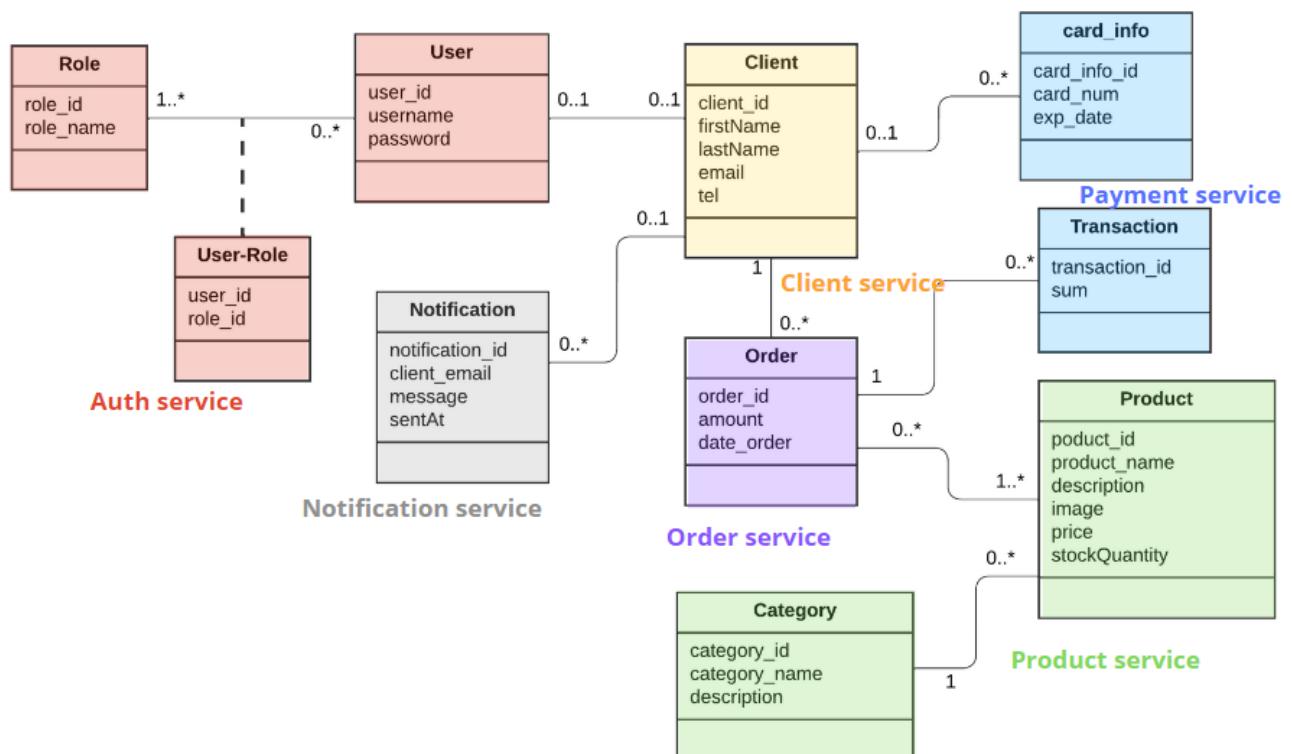


FIG. 3.5 : Décomposition de services

### 3.3.3 Base de données par service :

Chaque microservice doit avoir sa propre base de données dédiée pour être découpé des autres microservices. Il est responsable de sa propre persistance des données et ne partage pas sa base de données avec d'autres microservices. Cela favorise l'indépendance et l'isolation des microservices, ce qui facilite le déploiement, la mise à l'échelle et la maintenance.

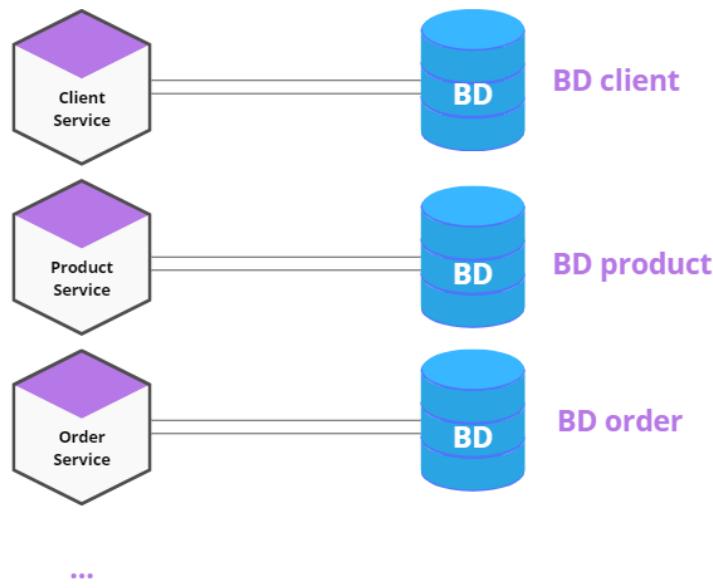


FIG. 3.6 : Base de données par service

### 3.3.4 Le registre de service :

Élément clé de l'architecture de microservices permettant la **découverte (discovery)** des services. Il s'agit d'une base de données qui contient les emplacements des microservices disponibles dans le système. Lorsqu'un nouveau microservice est déployé, il doit d'abord s'inscrire auprès de ce registre de service en envoyant son adresse IP et son numéro de port.

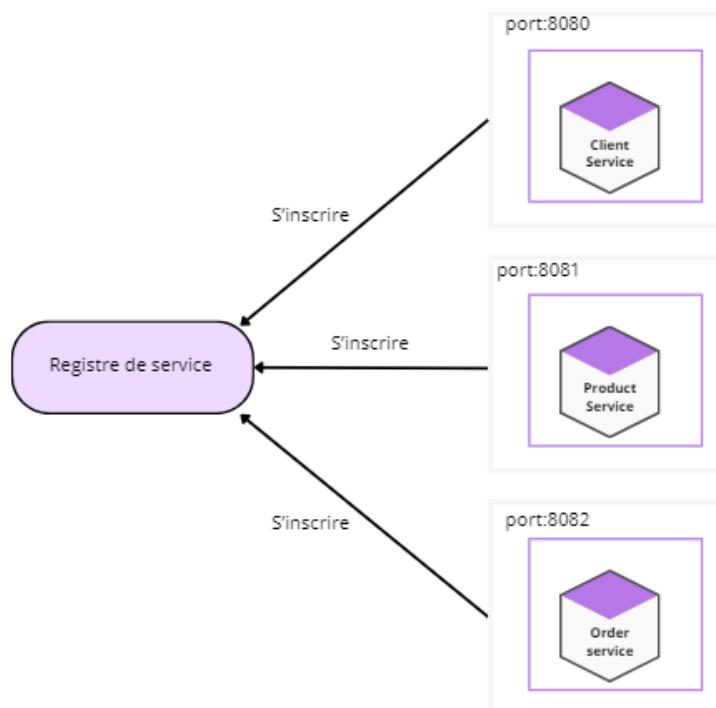


FIG. 3.7 : Registre de services

Nous pouvons ainsi découvrir l'emplacement des services en interrogeant le registre, ce qui permet d'adapter dynamiquement les appels en fonction des changements dans le déploiement des services.

Il peut être utilisé également par des mécanismes d'équilibrage de charge (Load balancing) pour distribuer les requêtes de manière uniforme entre les instances de services disponibles, optimisant ainsi l'utilisation des ressources et améliorant la performance.

### 3.3.5 Passerelle d'API

Un **API Gateway** ou passerelle d'API se positionne entre un client et une collection de services back-end. C'est un patron de microservices qui agit comme un proxy inversé, il intercepte toutes les requêtes entrante et rassemble les différents services requis pour y répondre et renvoie le résultat souhaité.

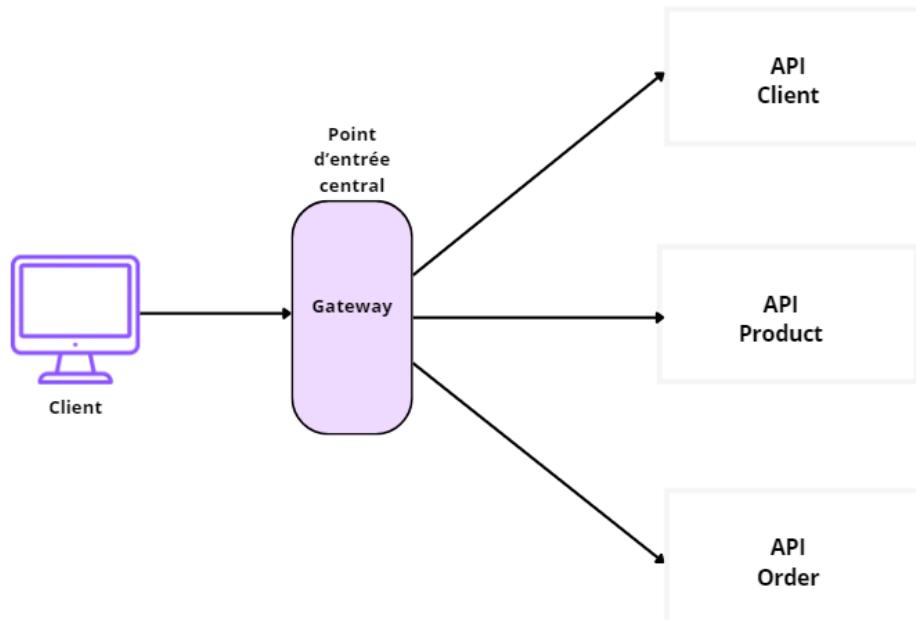


FIG. 3.8 : Passerelle API

Au sein des entreprises qui adoptent une approche DevOps, les développeurs utilisent des microservices pour créer et déployer des applications de façon itérative et accélérée. Les API constituent la méthode de communication la plus courante entre les microservices. Ainsi, à mesure que les API se complexifient et que leur usage augmente, les passerelles d'API se montrent de plus en plus utiles.

### 3.3.6 Le traçage distribué

Le traçage distribué ou distributed tracing, est une technique utilisée pour surveiller et suivre les appels entre les différents composants d'une architecture de microservices. Cette méthode permet de comprendre et d'analyser le chemin parcouru par

une requête à travers divers services, facilitant ainsi le diagnostic des problèmes de performance, des erreurs et des goulots d'étranglement.

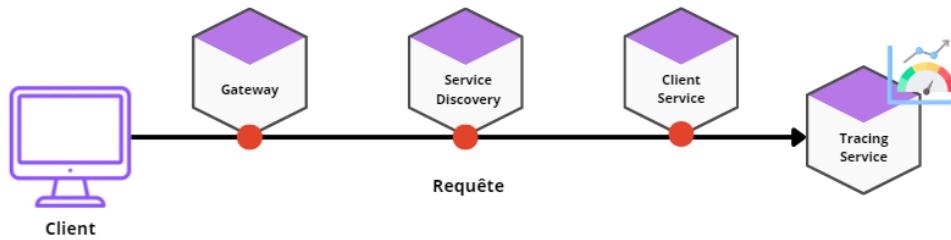


FIG. 3.9 : Traçage distribué

Quelques concepts du traçage distribué :

- **Trace** : l'ensemble des opérations effectuées par une requête unique à travers les différents services. Elle capture le chemin complet suivi par la requête, du début à la fin.
- **Span** : unité de travail dans une trace. Il représente une opération unique effectuée par un service. Chaque span contient des informations telles que le début et la fin de l'opération, les métadonnées associées, et parfois des logs.
- **Context de trace** : Il s'agit des informations de suivi (par exemple, un identifiant unique pour la trace et les spans) qui sont propagées avec la requête d'un service à l'autre. Cela permet de relier les spans entre eux pour former une trace cohérente.
- **Instrumentation** : Pour mettre en place le traçage distribué, il faut instrumenter les microservices. Cela implique de modifier le code des services pour ajouter des points de traçage (création et propagation des spans) ou d'utiliser des bibliothèques et des frameworks qui fournissent cette fonctionnalité de manière automatique.
- **Outil de visualisation et d'analyse** : Les traces collectées sont envoyées à un système de stockage et d'analyse, où elles peuvent être visualisées. Des outils comme Jaeger, Zipkin, ou encore des solutions intégrées dans des plateformes comme AWS X-Ray, sont couramment utilisés pour cela. Ces outils permettent de voir les relations entre les services, le temps de réponse, et de diagnostiquer les problèmes.

### 3.3.7 La file d'attente de messages

Une file d'attente de messages (**message queue**) est un mécanisme qui permet aux systèmes distribués de communiquer de manière **asynchrone**. Les messages sont envoyés par les producteurs (microservices émetteurs, producers) et stockés dans la file d'attente jusqu'à ce qu'ils soient récupérés par les consommateurs (microservices récepteurs, consumers). Cela permet de découpler les composants du système,

facilitant ainsi la scalabilité et la résilience. Mais aussi assurant qu'aucun service ne se bloque en attente de réponses après chaque envoi de requête.



FIG. 3.10 : File d'attente de messages

### 3.4 Architecture finale

Après avoir découvert les différents piliers de notre architecture de microservices, il est temps de rassembler le tout dans un schéma complet et explicatif qui illustre l'architecture générale de notre application **Let's Shop**.

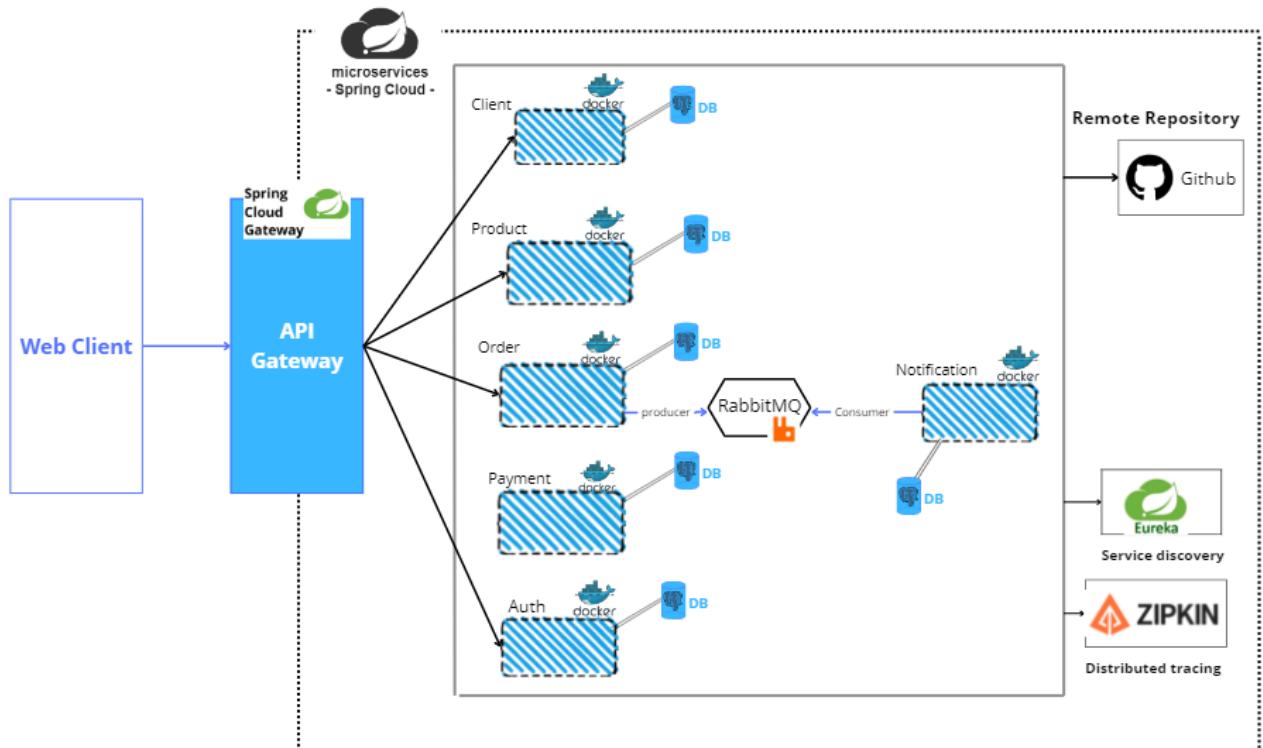


FIG. 3.11 : Architecture e-commerce

### **3.5 Conclusion**

En conclusion, la conception de l'architecture microservices pour l'application "Let's Shop" représente une avancée significative vers une solution plus flexible et évolutive. En comparaison avec l'approche monolithique, les microservices offrent une meilleure gestion des fonctionnalités métier, permettant une modification ciblée et indépendante des différentes parties de l'application. Cette approche favorise également l'utilisation de technologies diverses adaptées à chaque service, sans conflits potentiels.

# **Chapitre 4**

## **Conception de chaque microservice**

### **4.1 Introduction**

Dans cette section, nous détaillons la conception de chaque microservice composant notre application. Nous distinguons les microservices fonctionnels, qui gèrent la logique métier, des microservices d'infrastructure, qui fournissent des fonctionnalités de support essentielles au bon fonctionnement de l'architecture globale.

## 4.2 Microservices fonctionnels

Nous désignons les microservices fonctionnels ou métier comme Product, Order, Client, Notification et Payment. Leur rôle principal est de gérer la logique métier de l'application. Chacun de ces services est autonome et responsable d'un domaine fonctionnel précis.

### 4.2.1 Client-Service

Dans ce service, nous gérons les informations et les interactions relatives aux clients, comme l'enregistrement, la mise à jour des données et la consultation des informations clients.

D'après le diagramme de cas d'utilisation suivant, tout administrateur peut consulter tous les clients existants.

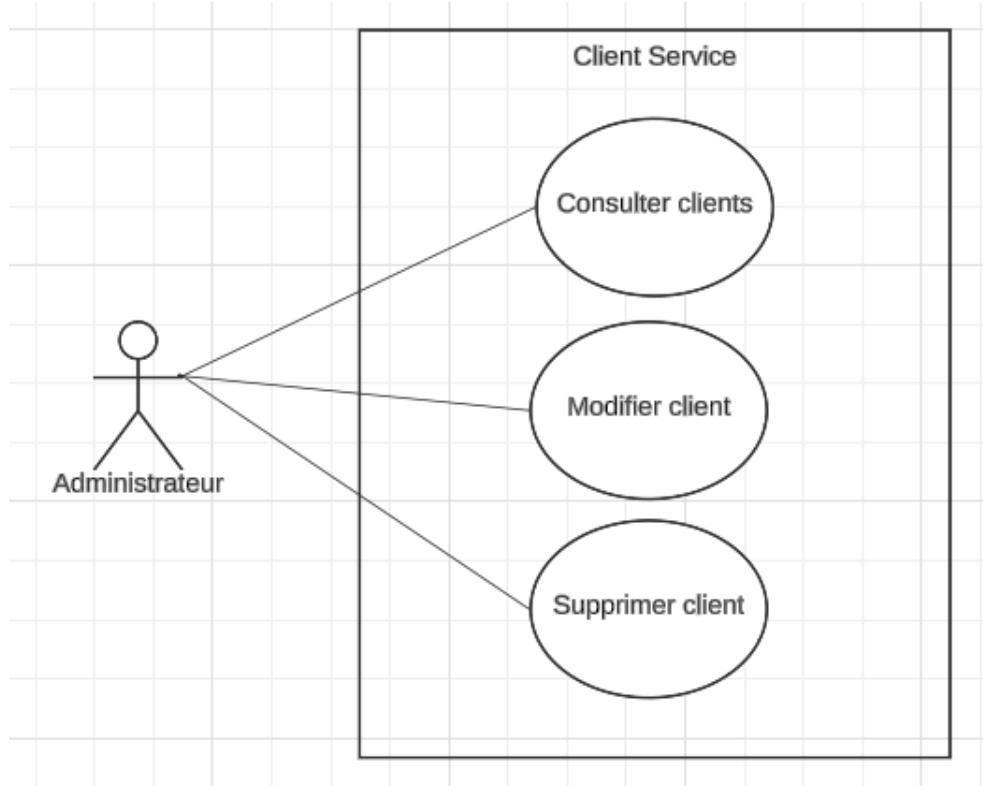


FIG. 4.1 : DCU du service Client

La classe Client référence la classe User, permettant aux clients d'avoir un compte pour se connecter à la plateforme et pouvoir entreprendre des achats.

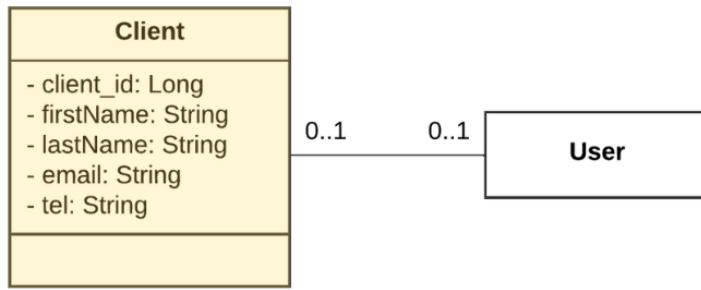


FIG. 4.2 : Diagramme de classe du service Client

#### 4.2.2 Product-Service

Le service Product est responsable de la gestion des produits disponibles sur la plateforme. Cela inclut l'ajout de nouveaux produits, la mise à jour des informations de produit existant, la suppression de produits et la consultation des détails des produits.

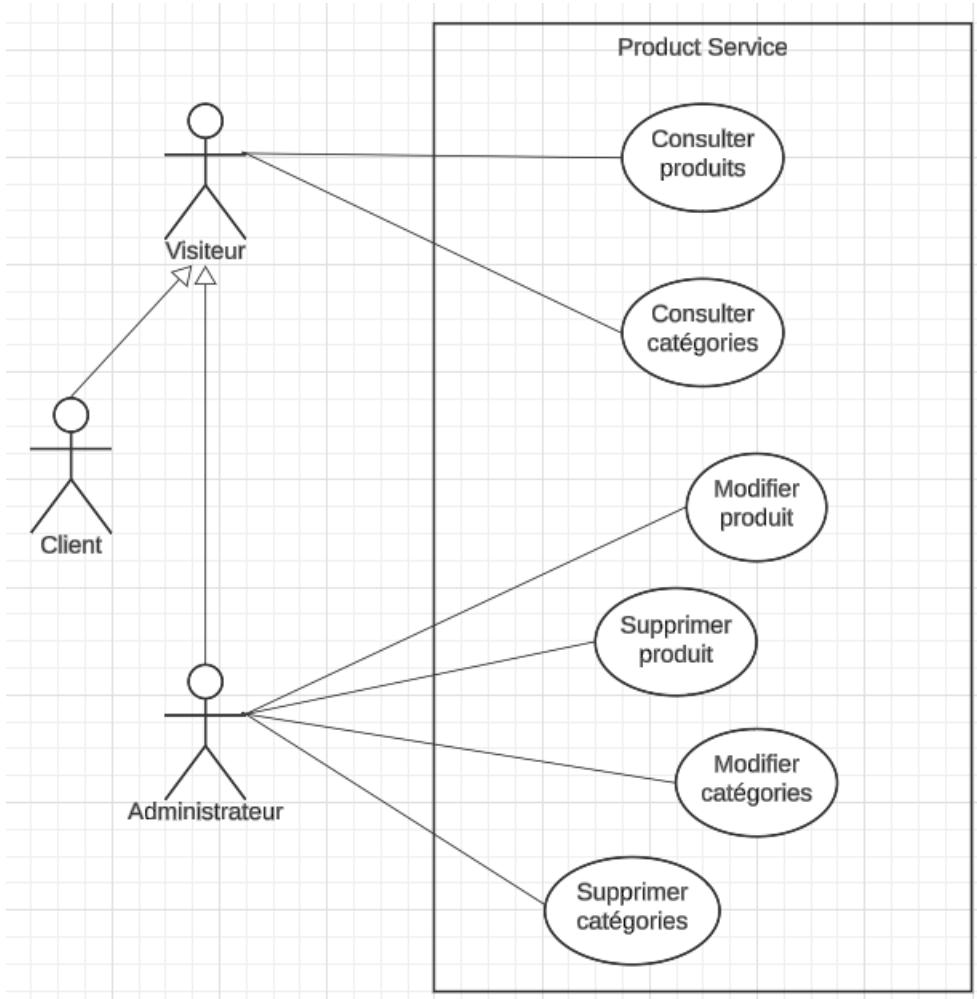


FIG. 4.3 : DCU du service Product

La classe Product contient des détails tels que son nom, description, catégorie, une

image du produit, son prix unitaire et la quantité stockée. Les différentes catégories sont représentées par l'entité Category.

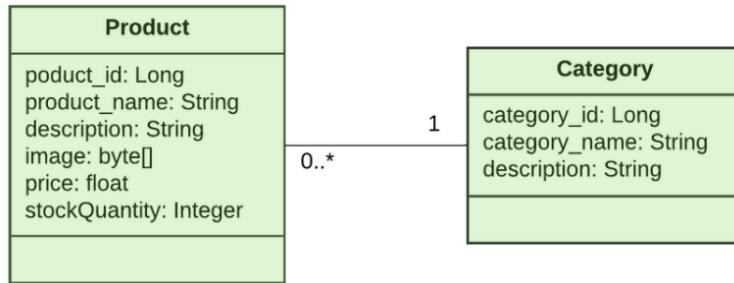


FIG. 4.4 : Diagramme de classes du service Product

### 4.2.3 Order-Service

Le service Order gère les commandes passées par les clients. Cela inclut la création de nouvelles commandes, la mise à jour des commandes existantes, et la consultation des détails des commandes. Il devrait aussi pouvoir stocker l'évènement de la commande dans un nouveau message envoyé vers le service notification.

Le diagramme de cas d'utilisation montre que les clients peuvent créer et suivre leurs commandes, tandis que les administrateurs peuvent gérer et consulter toutes les commandes.

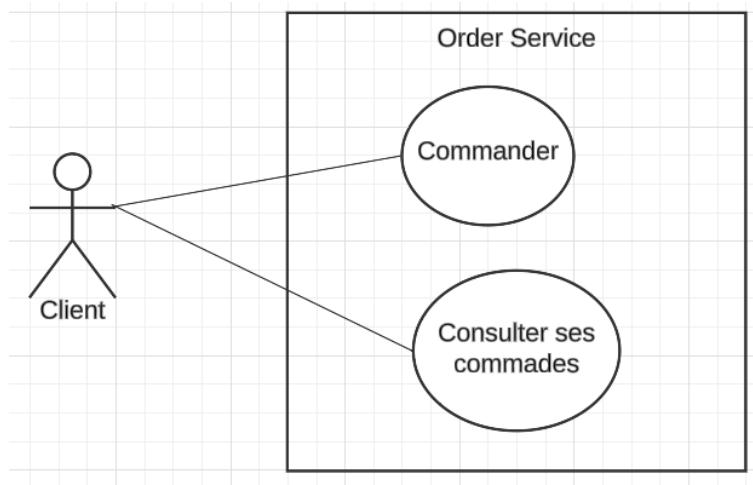


FIG. 4.5 : DCU du service Order

La classe Order référence le client qui a fait la commande, ainsi que le produit acheté. Les opérations associées incluent la création, la mise à jour, et l'annulation des commandes, ainsi que la gestion de l'état des commandes.

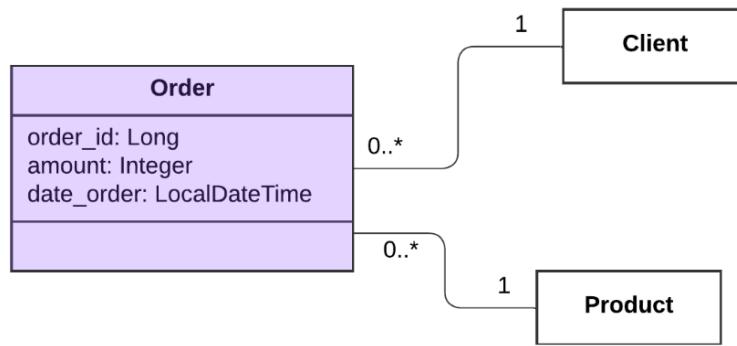


FIG. 4.6 : Diagramme de classes du service Order

#### 4.2.4 Notification-Service

Le service Notification est responsable du stockage de notifications destinées aux clients. Cela peut inclure l'envoi de notifications par email ou SMS lors de certains événements, comme la création d'une commande ou la mise à jour du statut d'une commande.

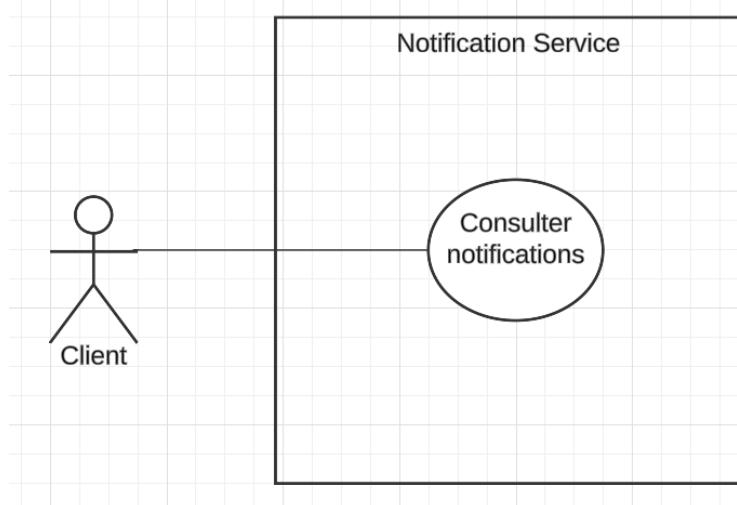


FIG. 4.7 : DCU du service Notification

La classe Notification référence un client et contient le message en question ainsi que le timestamp.

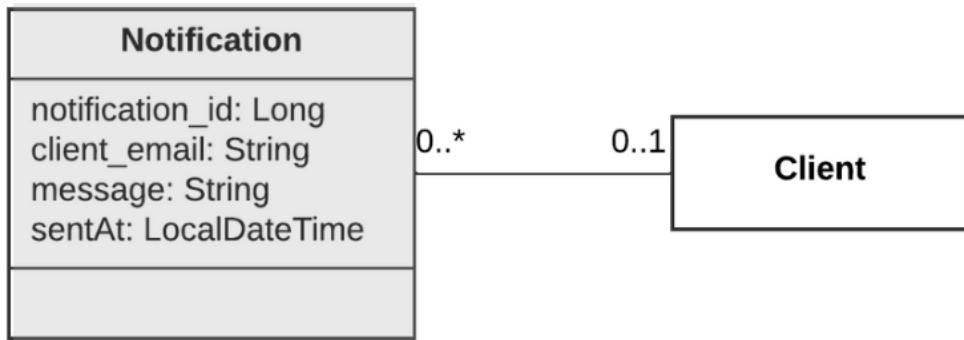


FIG. 4.8 : Diagramme de classe du service Notification

#### 4.2.5 Payment-Service

Le service Payment gère les transactions financières entre les clients et la plateforme. Cela peut inclure le traitement des paiements, la gestion des remboursements, et la consultation de l'historique des transactions.

La classe Transaction contient les détails de chaque transaction, tels que l'identifiant de la commande et le montant. La classe Carte de Paiement contient les informations de la carte de paiement, telles que le numéro de carte, la date d'expiration, le titulaire de la carte, etc.

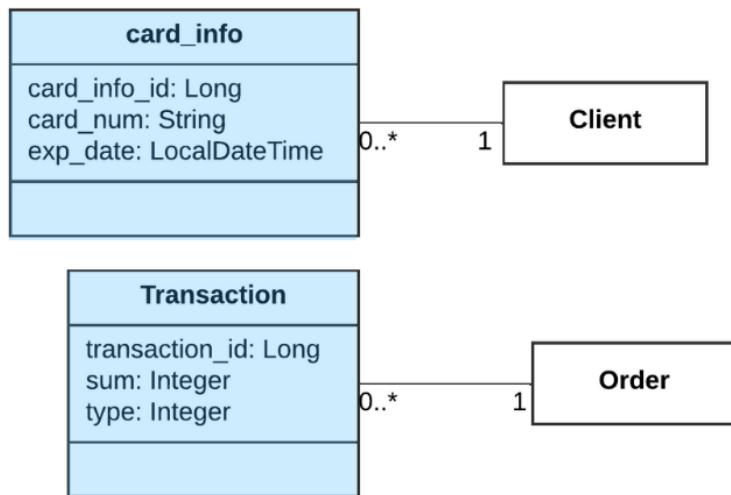


FIG. 4.9 : Diagramme de classes du service Payment

### 4.3 Microservices d'infrastructure

Le rôle des microservices d'infrastructure est de fournir des fonctionnalités de support et d'infrastructure nécessaires pour faire fonctionner l'ensemble de l'architecture de microservices de manière efficace et robuste.

### 4.3.1 Eureka-Server

Le service Eureka est un service de découverte qui permet aux microservices de s'enregistrer et de découvrir les instances d'autres microservices. Cela permet une gestion dynamique des adresses réseau et assure une communication fluide entre les microservices.

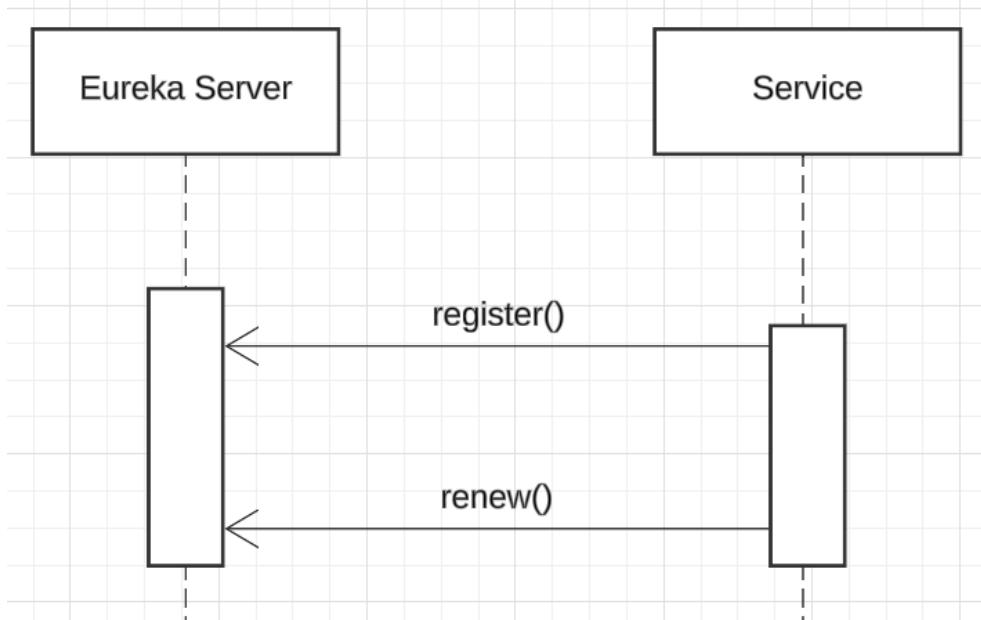


FIG. 4.10 : Diagramme de séquence du service Eureka

### 4.3.2 Gateway-Service

Le service Gateway sert de point d'entrée unique pour toutes les requêtes des clients vers les microservices. Il gère des tâches telles que l'authentification, la répartition de charge, la transformation des protocoles, et l'agrégation des réponses.

Chaque nouvelle requête est envoyée vers le gateway d'abord (de port 8222 par exemple), et c'est le gateway qui renvoie la requête au service destination approprié, comme c'est démontré dans le diagramme de séquences suivant :

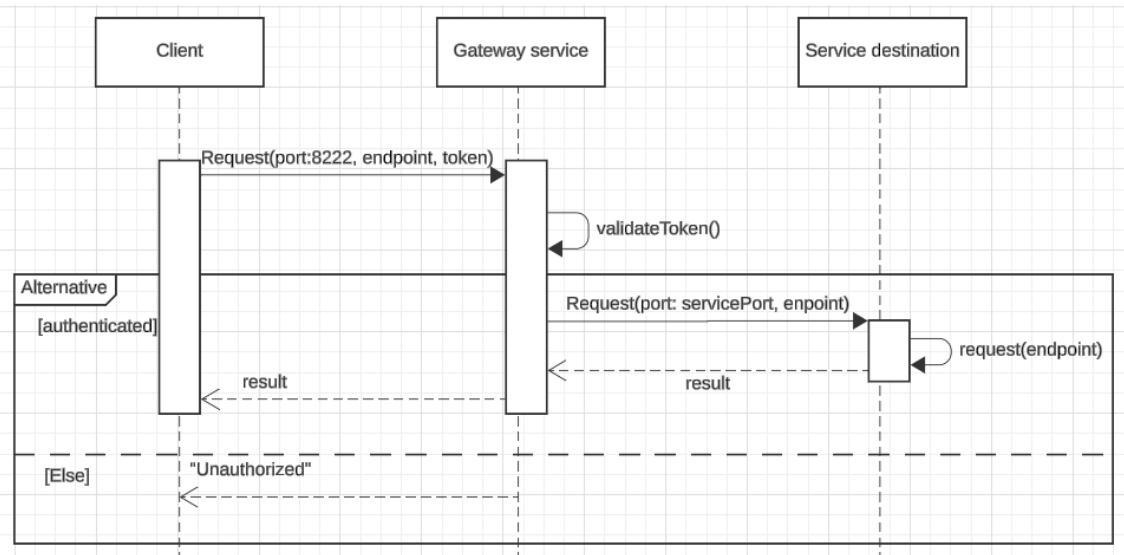


FIG. 4.11 : Diagramme de séquences du service Gateway

### 4.3.3 Auth-Service

Le service Auth gère l'authentification et l'autorisation des utilisateurs. Il vérifie les informations d'identification fournies par les utilisateurs, génère et valide les tokens d'authentification, et applique les politiques de sécurité.

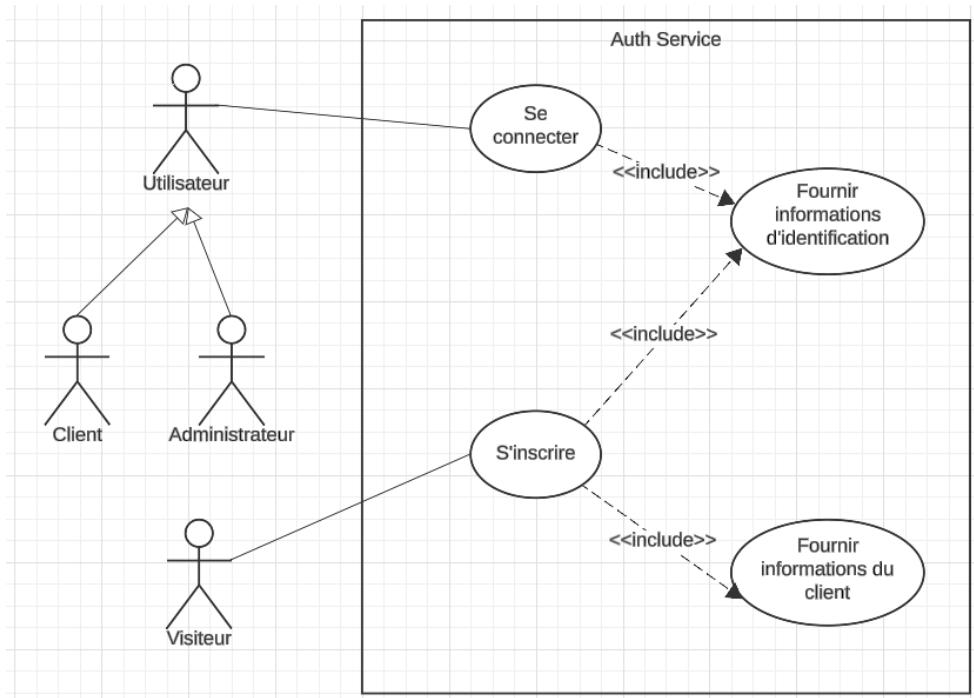


FIG. 4.12 : DCU du service Auth

La classe User contient les informations d'authentifications qui vont permettre aux utilisateurs de la plateforme de se connecter. Un utilisateur peut avoir plusieurs

rôles, et un rôle peut être fourni à plusieurs utilisateurs, d'où la relation Many to Many (n,n) créant une entité intermédiaire pour gérer cette relation.

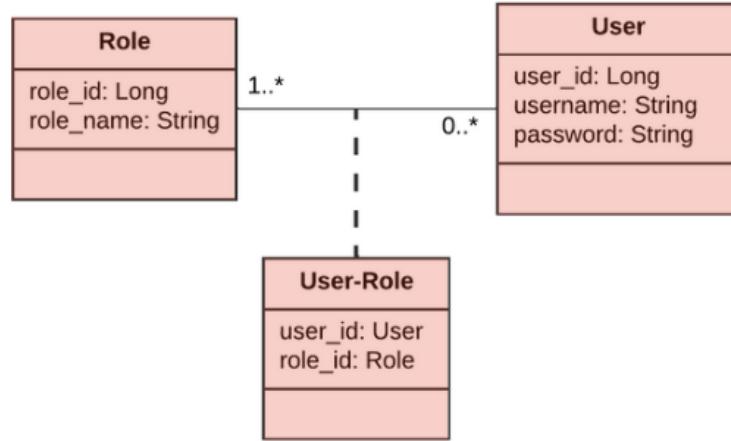


FIG. 4.13 : Diagramme de classes du service Auth

### Authentification : Modèle d'accès par Jeton

Le **Token Access Pattern** (ou modèle d'accès par jeton) est une approche utilisée dans les architectures de microservices pour gérer l'authentification et l'autorisation des utilisateurs. Il implique l'utilisation de jetons (tokens) pour vérifier l'identité et les permissions des utilisateurs lors de chaque interaction avec les microservices.

Pour l'**inscription**, on envoie les informations d'utilisateurs (`username` et `password`) en plus des informations de client personnelles (`firstName`, `lastName`, `email`, `tel`), cette requête est transférée par le service Gateway vers le service Auth qui enregistre les informations de `User` et `Client`. Voici le diagramme de séquences établi pour l'inscription :

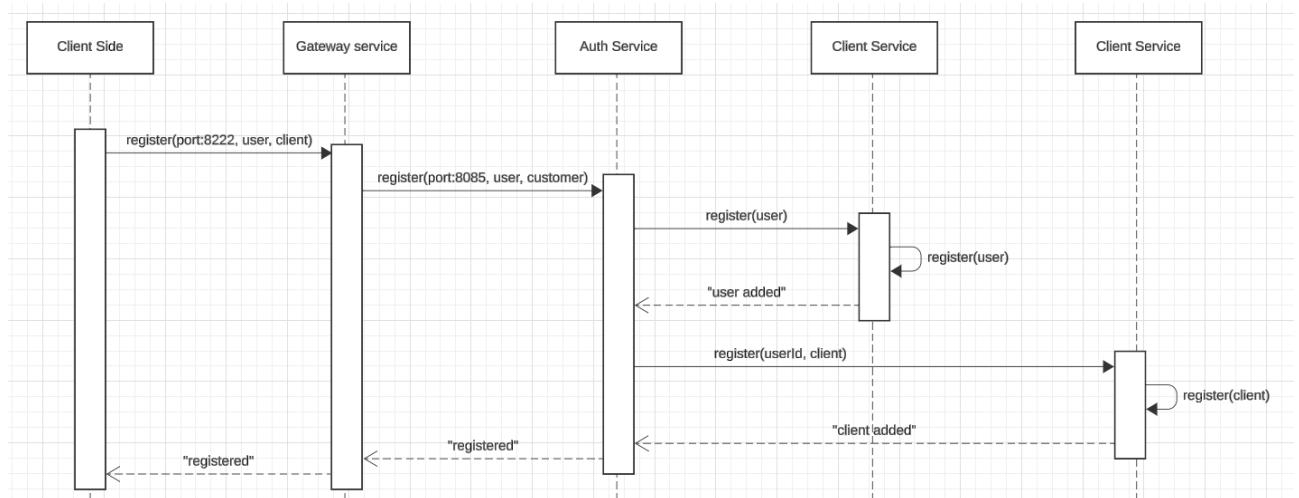


FIG. 4.14 : Diagramme de séquences pour l'inscription

Pour l'**authentification**, on envoie les informations d'identification (`username` et `password`), cette requête est transférée par le service Gateway vers le service Auth

qui vérifie la validité de ces informations, si elles sont valides, il génère un token qu'il renvoie, sinon, il renvoie un message "Not Authorized".

Voici le diagramme de séquences établi pour l'authentification :

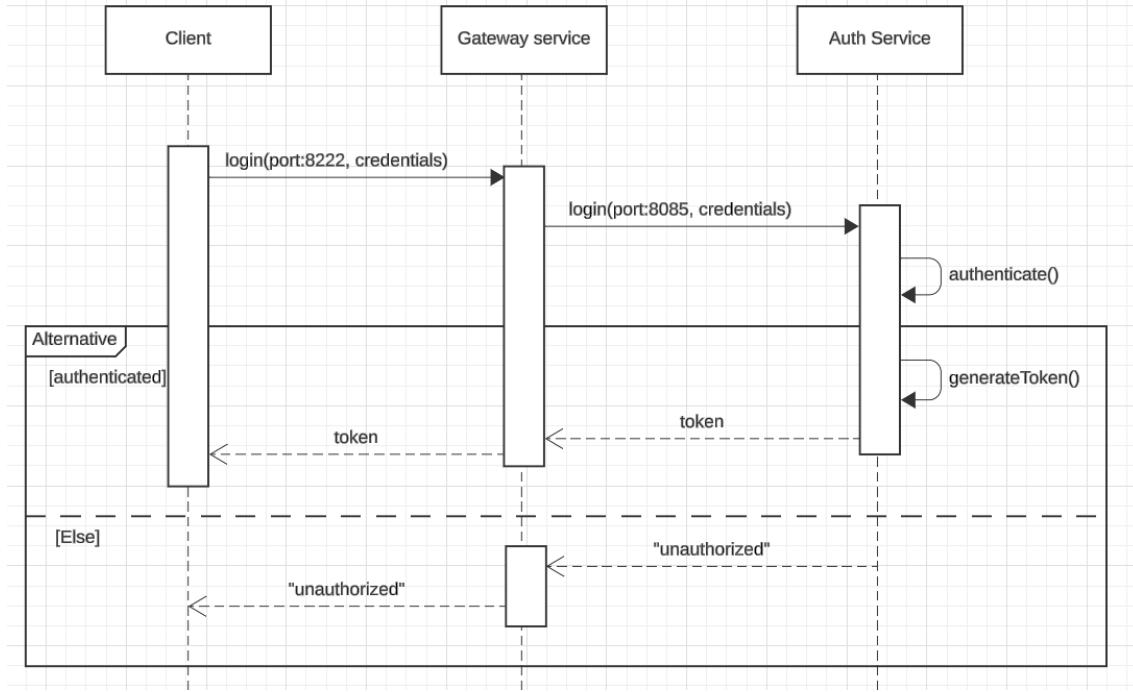


FIG. 4.15 : Diagramme de séquences pour l'authentification

## 4.4 Conclusion

Ainsi se résume la partie Conception de chaque microservices où nous nous sommes concentrés sur la conception et modélisation de chaque service. nous pouvons donc enfin passer à la réalisation.

# **Chapitre 5**

## **Réalisation**

### **5.1 Introduction**

Après avoir présenté notre projet, fait une étude des systèmes micro-services et abordé l'analyse et conception, il est temps maintenant de concrétiser cela en abordant la partie réalisation et en mettant en œuvre les concepts et compétences acquises au cours de notre formation. Cette phase de réalisation est l'occasion de relever des défis techniques, de prendre des décisions architecturales et de faire face à des problèmes concrets, tout en travaillant en équipe pour atteindre nos objectifs communs. Dans ce chapitre, nous détaillerons les différentes étapes de mise en œuvre de notre application e-commerce, en mettant en lumière les choix technologiques, les méthodologies de développement et les défis rencontrés tout au long du processus.

## 5.2 Frameworks

### 5.2.1 Spring boot & Spring cloud



FIG. 5.1 : Logos de Spring Boot & Spring Cloud

**Spring Boot** est un framework Java pour le développement d'applications backend, idéal pour les microservices grâce à sa simplicité de configuration et ses fonctionnalités intégrées de gestion des dépendances et de sécurité. Il permet de développer et déployer rapidement des services robustes et évolutifs, facilitant la gestion des transactions et des données pour les applications e-commerce.

**Spring Cloud** est un framework de microservices léger, conçu pour faciliter le développement et le déploiement d'applications distribuées et de microservices. Il s'intègre étroitement avec le framework Spring Boot, permettant ainsi de tirer parti des fonctionnalités puissantes de Spring tout en simplifiant la complexité inhérente aux architectures microservices.

### 5.2.2 ReactJs

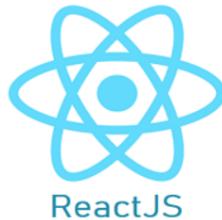


FIG. 5.2 : ReactJs

ReactJS, une bibliothèque JavaScript populaire, permettant de construire des interfaces utilisateur interactives grâce à son approche déclarative et à la composition de composants. Utilisée pour notre application e-commerce, elle offre une expérience utilisateur fluide et intuitive, gérant efficacement les mises à jour en temps réel pour des fonctionnalités dynamiques comme la navigation entre les pages et la gestion du panier d'achat.

Ainsi, en combinant Spring Boot pour le backend et ReactJS pour le frontend, nous adoptons une approche moderne et modulaire pour le développement de notre application e-commerce. Cette combinaison de technologies nous offre la flexibilité nécessaire pour concevoir une architecture scalable et hautement performante, tout en fournissant une expérience utilisateur immersive et intuitive.

## 5.3 Pour la conception microservices

A part les frameworks utilisés pour la programmation, nous avons utilisé, dans ce projet, plusieurs autres technologies pour mettre en place notre conception microservices. Pour chaque service, nous créerons un sous-module qui représentera un service indépendant remplissant ses fonctionnalités.

### 5.3.1 Registre de services : Eureka



FIG. 5.3 : Serveur Eureka

**Eureka** est un service de registre développé par Netflix et utilisé dans les architectures microservices pour la découverte de services. Dans un environnement microservices, les instances de services peuvent changer fréquemment en raison de la mise à l'échelle dynamique, des pannes et des déploiements. Eureka résout ce problème en agissant comme un annuaire central où les services peuvent s'enregistrer et découvrir d'autres services. Les instances de services enregistrent leurs adresses auprès d'Eureka, et les clients peuvent ensuite interroger Eureka pour obtenir les adresses actuelles des services disponibles, facilitant ainsi la communication et l'équilibrage de charge entre les microservices.

Un exemple d'utilisation pour notre projet :

 A screenshot of a web browser window. The address bar shows "localhost:8761". The page title is "DS Replicas". Below the title, a section titled "Instances currently registered with Eureka" contains a table with two rows of data. The table has columns: Application, AMIs, Availability Zones, and Status. The first row for "ORDER" shows "n/a (1)" in all columns, with "Status" being "UP (1) - localhost:order:8081". The second row for "PRODUCT" also shows "n/a (1)" in all columns, with "Status" being "UP (1) - localhost:product:8082". At the bottom of the page, there is a section titled "General Info".
 

Application	AMIs	Availability Zones	Status
ORDER	n/a (1)	(1)	UP (1) - localhost:order:8081
PRODUCT	n/a (1)	(1)	UP (1) - localhost:product:8082

FIG. 5.4 : Schéma éclaircissant la fonction d'un server registry

Pour pouvoir utiliser Eureka, il suffit d'ajouter la dépendance : **spring-cloud-starter-netflix-eureka-client** au pom.xml du module Eureka-server.

### 5.3.2 API Gateway : Spring Cloud Gateway



FIG. 5.5 : Spring Cloud Gateway

Spring Cloud Gateway est une API Gateway fournissant une solution flexible et performante pour gérer le routage et la gestion des API dans les architectures microservices.

Ceci nous permet de définir toutes les routes vers les microservices dans le fichier de configuration gateway, et avoir toutes les requêtes client dirigée vers seulement le point d'entrée centrale de l'API Gateway sur le port 8222.

```
routes:
- id: client
  uri: http://localhost:8080
  predicates:
    - Path=/api/client/**
  filters:
    - AuthFilter

- id: order
  uri: http://localhost:8081
  predicates:
    - Path=/api/order/**
  filters:
    - AuthFilter

- id: product
  uri: http://localhost:8082
  predicates:
    - Path=/api/product/**
  filters:
    - AuthFilter

- id: category
```

FIG. 5.6 : Routing de services dans API Gateway

La technologie utilisée dans ce cas est la dépendance **spring-cloud-starter-gateway**.

### 5.3.3 Sécurité : Spring Security & Tokens JWT



FIG. 5.7 : Spring Security & JWT

**Spring Security** est un puissant framework de sécurité pour les applications basées sur Java, offrant des fonctionnalités d'authentification, d'autorisation, de protection contre les attaques de type CSRF (Cross-Site Request Forgery), et bien plus encore. Il est hautement personnalisable et s'intègre parfaitement avec d'autres composants de l'écosystème Spring.

Un **JWT** (JSON Web Token) est un standard ouvert qui définit une méthode compacte et autonome pour transmettre de manière sécurisée des informations entre des parties en tant qu'objet JSON. Ces informations peuvent être vérifiées et confiées grâce à une signature numérique.

Un jeton JWT est composé de trois parties distinctes, qui sont séparées par des points (.) :

- (1) **Header (En-tête)** : Le header contient généralement deux parties : le type de jeton (typ) et l'algorithme de hachage (alg) utilisé pour signer le jeton.
- (2) **Payload (Charge utile)** : Contient les informations ou les "déclarations" du jeton. Il peut s'agir de données d'utilisateur (username, rôles).
- (3) **Signature** : Elle est créée en prenant le header et le payload encodés en Base64URL, et une clé secrète. Ces éléments sont ensuite combinés et signés à l'aide de l'algorithme de hachage spécifié dans le header. La signature garantit que le jeton n'a pas été modifié lors de son transport et permet de vérifier l'authenticité du jeton.

**Encoded**

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJtZXJ5MzIiLCJyb2xlcjI6W3sibmFtZSI6ImNsawWVudCJ9XSwiaWF0IjoxNTE2MjM5MDIyLCJleHAiOiE1MTYzMzkzMjJ9.NimuViC8fppIW25SVTlunNxEGtM-mdg4AIed9e8URC0
```

**Decoded**

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE	
{           "alg": "HS256",           "typ": "JWT"         }	
PAYLOAD: DATA	
{           "sub": "merry32",           "roles": [             {"name": "client"}           ],           "iat": 151623902,           "exp": 151623932         }	
VERIFY SIGNATURE	
HMACSHA256(           base64UrlEncode(header) + "." +           base64UrlEncode(payload),           your_256_bit_secret         )	

FIG. 5.8 : Exemple d'un token JWT

### 5.3.4 Traçage distribué : Sleuth & Zipkin

**Zipkin** est un système de traçage distribué conçu pour aider à surveiller et à dépanner les applications microservices. Il permet de collecter des données sur les requêtes qui transitent entre différents services, ce qui aide à identifier les goulets d'étranglement et à analyser les performances du système. En centralisant ces traces, Zipkin offre une vue d'ensemble des interactions entre les services, facilitant ainsi le diagnostic des problèmes et l'amélioration des performances globales de l'application. Sa compatibilité avec divers frameworks et sa facilité d'intégration en font un outil précieux pour les équipes de développement cherchant à maintenir des architectures complexes.

**Spring Cloud Sleuth** est un outil de traçage distribué qui aide à suivre les requêtes à travers les microservices. Il ajoute des identifiants uniques aux logs afin de corrélérer les logs générés par différentes applications et services impliqués dans le traitement d'une requête unique.

Lorsque Zipkin est utilisé avec Sleuth, il permet de centraliser et de visualiser les données de traçage, facilitant ainsi le débogage et la surveillance des applications microservices.

En faisant une requête basique, par exemple récupérer tous les clients, la requête entière sera tracée avec chaque microservice par où elle est passée.

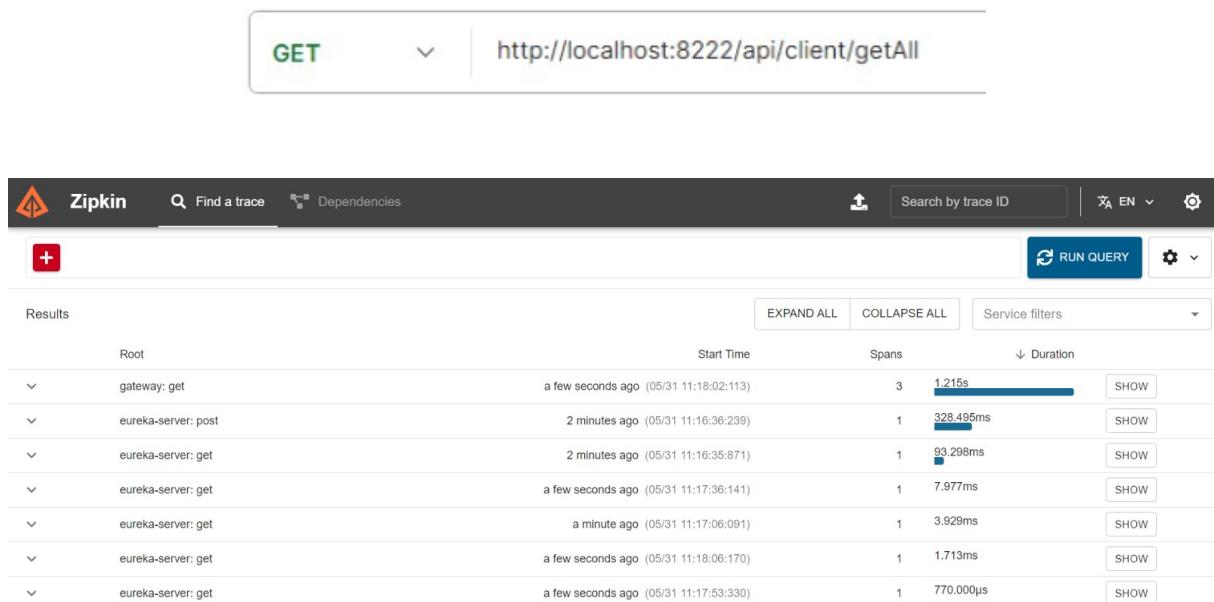


FIG. 5.9 : Service Zipkin

### 5.3.5 Communication entre services

#### 5.3.5.1 Rest Template

RestTemplate est une classe fournie par Spring pour simplifier la communication avec des services RESTful. Elle offre des méthodes pratiques pour envoyer des re-

quêtes HTTP et recevoir des réponses, permettant ainsi aux applications Spring de consommer des API externes de manière simple et efficace.

Cependant, Rest Template nécessite de gérer manuellement les requêtes HTTP et les mappages de réponses, ce qui peut rendre le code verbeux et difficile à maintenir, surtout pour des interactions complexes.

### 5.3.5.2 Feign Client



FIG. 5.10 : Feign Client

Feign Client est une bibliothèque Java pour simplifier les appels aux services REST dans les architectures microservices. Il permet de définir des clients HTTP de manière déclarative en utilisant des interfaces Java annotées. Plutôt que d'écrire manuellement du code pour gérer les requêtes et réponses HTTP, Feign Client génère automatiquement ce code en se basant sur les annotations fournies. Cette approche réduit considérablement le code boilerplate et améliore la lisibilité et la maintenabilité du code. De plus, Feign Client offre ainsi une solution robuste et flexible pour la communication inter-services dans une architecture microservices.

### 5.3.5.3 File d'attente de messages : RabbitMQ

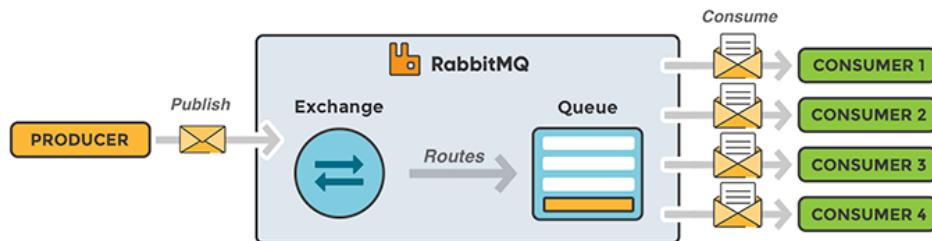


FIG. 5.11 : Fonctionnement de RabbitMQ

RabbitMQ est un système de messagerie open-source robuste, conçu pour faciliter la communication entre applications distribuées. En tant que broker de messages, RabbitMQ permet aux applications de publier, stocker et consommer des messages de manière fiable, même dans des environnements à haute charge.

Il utilise le protocole **AMQP** (Advanced Message Queuing Protocol) pour assurer la transmission des messages de manière **asynchrone** entre producteurs et consommateurs. Grâce à sa flexibilité et à ses fonctionnalités avancées telles que la gestion des files d'attente, la réplication des messages et la gestion des erreurs, RabbitMQ est largement utilisé dans les architectures de microservices pour garantir une communication fiable et efficace entre les services.

Pour intégrer cette technologie dans notre projet, il faut créer un nouveau module amqp, pour y mettre les configurations de RabbitMQProducer ainsi que la dépendance *spring-boot-starter-amqp*.

Ensuite, nous avons récupéré l'image Docker de RabbitMQ pour l'exécuter dans un conteneur dédié.

The screenshot shows the RabbitMQ Management Interface at the URL `localhost:15672/#`. The interface is a web-based dashboard for managing a RabbitMQ cluster. At the top, it displays the version `RabbitMQ 3.9.11 Erlang 24.2`. The main navigation bar includes links for Overview, Connections, Channels, Exchanges, Queues, and Admin. The Overview page provides a summary of system health and resource usage. It includes sections for Totals (Queued messages last minute, Currently idle, Message rates last minute, Currently idle, Global counts), Nodes (listing a single node `rabbit@hc3972d20bc69` with various metrics like File descriptors, Socket descriptors, Erlang processes, Memory, Disk space, and Uptime), and links for Churn statistics, Ports and contexts, Export definitions, and Import definitions. At the bottom, there are links for HTTP API, Server Docs, Tutorials, Community Support, Community Slack, Commercial Support, Plugins, GitHub, and Changelog.

FIG. 5.12 : Interface de RabbitMQ

## 5.4 Interfaces

Notre application **Let's Shop** est une application web e-commerce se basant sur des microservices, dont les différentes interfaces sont les suivantes :

Chaque visiteur de la plateforme peut accéder à la page d'accueil de **Let's Shop** :

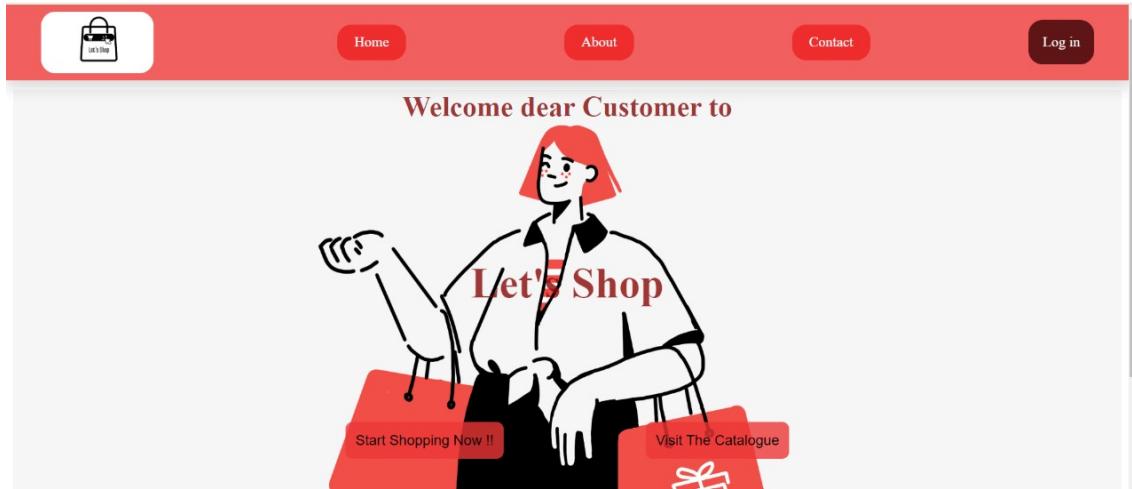


FIG. 5.13 : Page d'accueil

Après cela, l'utilisateur peut s'authentifier à travers la modale d'authentification ci-dessous. S'il est nouveau à notre application, il peut créer un compte en cliquant sur "Sign Up".

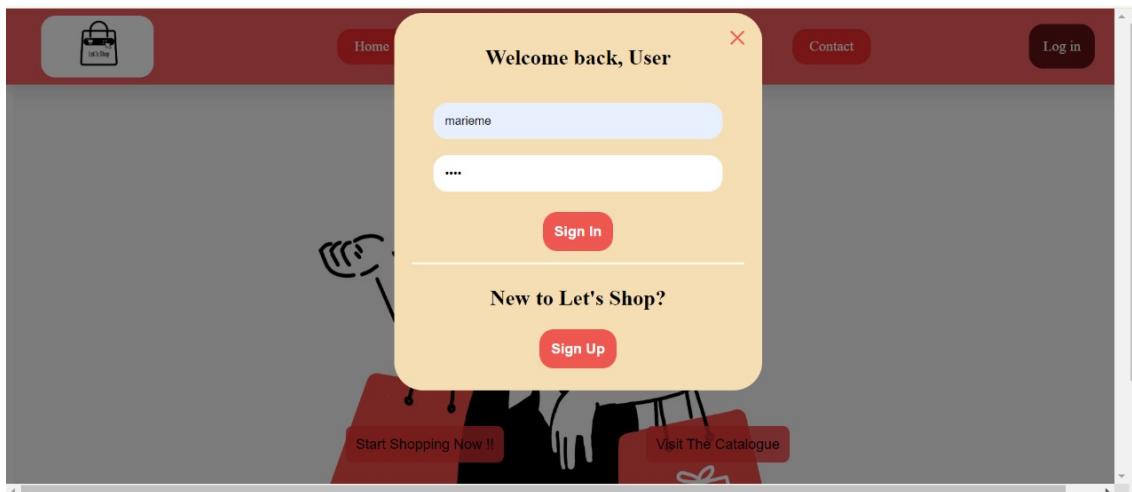


FIG. 5.14 : Modale de Login

Tout visiteur a accès à la page de catalogue où il peut consulter tous les produits proposés :

## Chapitre 5. Réalisation

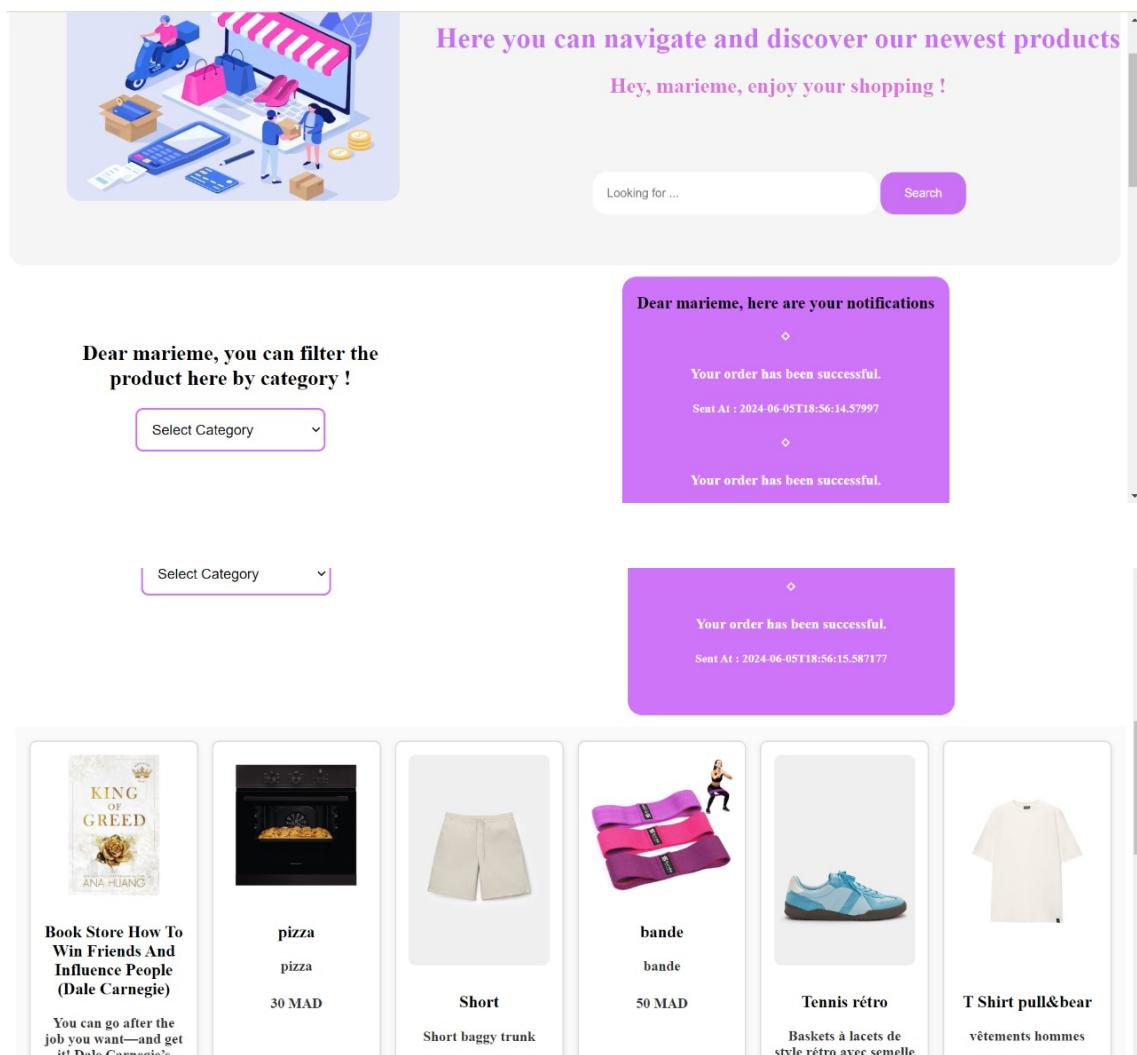


FIG. 5.15 : Page du catalogue de produits

En cliquant sur l'un des produits, on peut voir les détails et informations sur ce produit et décider si l'on veut le commander en appuyant sur le bouton "Order Now!".

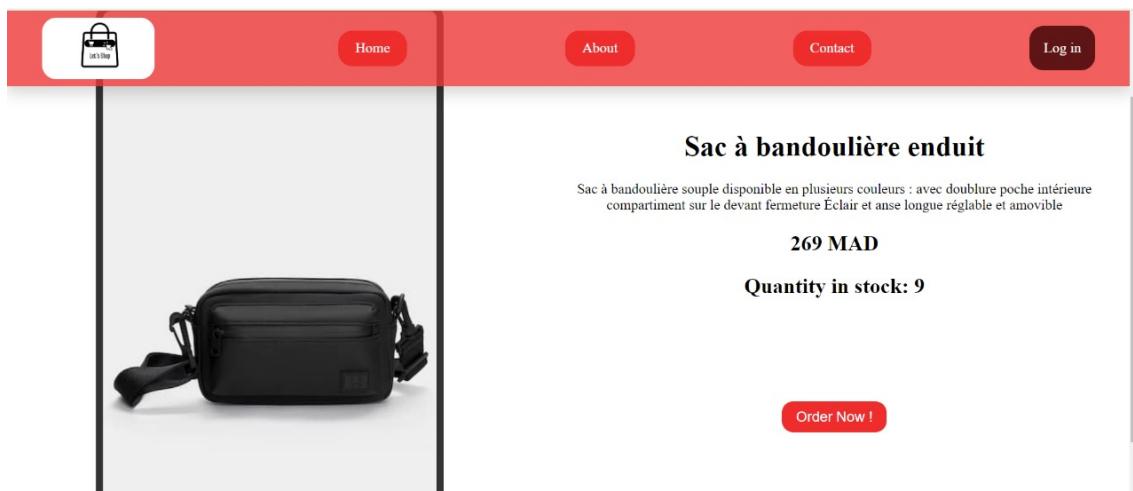


FIG. 5.16 : Page des détails d'un produit

En cliquant sur "Order Now!", on a accès à une page pour confirmer l'achat et spécifier la quantité :

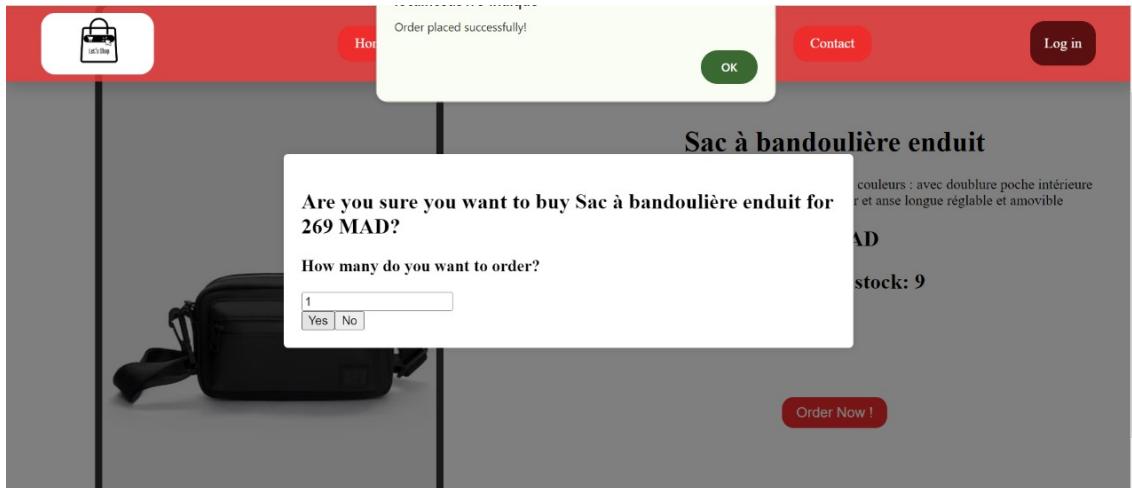


FIG. 5.17 : Modale de confirmation d'achat

Interface de l'administrateur : Si l'utilisateur authentifié possède le rôle "admin", il peut accéder à cette interface :

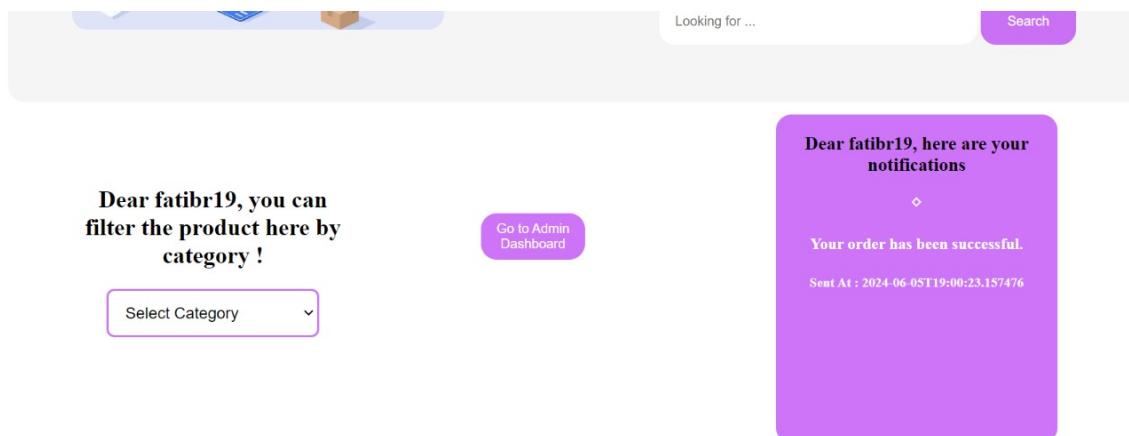


FIG. 5.18 : Accès à la page d'admin pour l'administrateur

## Chapitre 5. Réalisation

---

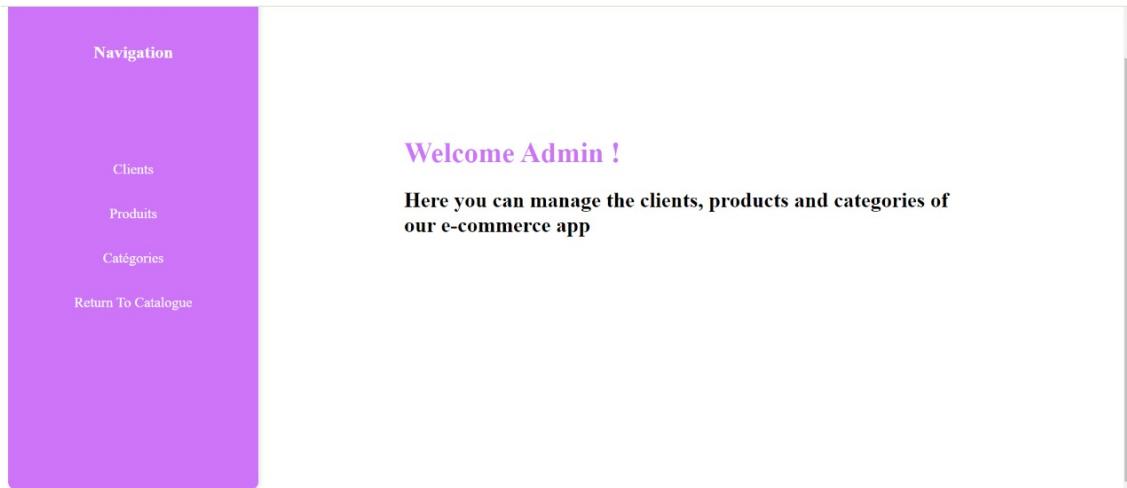


FIG. 5.19 : Page de l'administrateur

Après cela, l'admin de **Let's Shop** peut gérer les Clients, Produits et Catégories, en cliquant sur l'espace correspondant sur la Sidebar.

En premier lieu, vient la page des clients, où l'admin peut modifier, ajouter ou supprimer un client.

A screenshot of the client management page. The sidebar on the left is identical to Figure 5.19. The main content displays a table of users with the following columns: Prénom, Nom, Email, Téléphone, Id Utilisateur, and Actions. The table contains six rows of data. Each row includes "Modifier" and "Supprimer" buttons in the Actions column.

Prénom	Nom	Email	Téléphone	Id Utilisateur	Actions
Sara	Braik	sara@gmail.com	0601020304	3	<button>Modifier</button> <button>Supprimer</button>
Fati	Braikat	fati@gmail.com	0612345678	1	<button>Modifier</button> <button>Supprimer</button>
Mery	El Karati	mery@gmail.com	0634567812	2	<button>Modifier</button> <button>Supprimer</button>
Hiba	Mekkaoui	hiba@gmail.com	0634562167	4	<button>Modifier</button> <button>Supprimer</button>
Doaa	Atatri	doaa@gmail.com	0789762453	5	<button>Modifier</button> <button>Supprimer</button>
Doaa	Atatri	dodo.doe@example.com	1234567890	6	<button>Modifier</button> <button>Supprimer</button>

FIG. 5.20 : Page de gestion des clients

Après, il peut faire la même chose avec les produits (modification, ajout et suppression d'un produit).

## Chapitre 5. Réalisation

---

Nom	Description	Catégorie	Image	Prix	Quantité en stock	Actions
Book Store How To Win Friends And Influence People (Dale Carnegie)	You can go after the job you want—and get it! Dale Carnegie's rock-solid, time-tested advice has carried countless people up the ladder of success in their business and personal lives.	Books		58		<button>Modifier</button> <button>Supprimer</button>
pizza	pizza	Food		30	1	<button>Modifier</button> <button>Supprimer</button>
Short	Short baggy trunk	Clothes		299	7	<button>Modifier</button> <button>Supprimer</button>
						<button>Modifier</button>

FIG. 5.21 : Page de gestion des produits

Enfin, il peut modifier, ajouter ou supprimer une catégorie sur la page correspondante, figurant ci-dessous.

Nom	Description	Actions
Clothes	Clothes	<button>Modifier</button> <button>Supprimer</button>
Cosmetics	Cosmetics	<button>Modifier</button> <button>Supprimer</button>
Books	Books	<button>Modifier</button> <button>Supprimer</button>
HouseHold Appliances	HouseHold Appliances	<button>Modifier</button> <button>Supprimer</button>
Sport	Sport equipments	<button>Modifier</button> <button>Supprimer</button>
Food	Food	<button>Modifier</button> <button>Supprimer</button>

FIG. 5.22 : Page de gestion des catégories

## 5.5 Conclusion

En conclusion, le chapitre sur la réalisation de notre projet d’application e-commerce marque une transition essentielle de la théorie à la pratique. En mettant en œuvre les concepts et compétences acquises, nous avons relevé des défis techniques et pris des décisions architecturales. Grâce à l’utilisation de technologies telles que Spring Boot et ReactJS, nous avons créé une architecture robuste et une expérience utilisateur immersive. Ce chapitre démontre notre engagement à transformer nos idées en réalité fonctionnelle, tout en nous rapprochant de notre objectif de livrer une application e-commerce de qualité.

# Conclusion et perspectives

## Conclusion générale

En guise de conclusion, à travers ce projet, nous avons pu entamer le développement d'une application web e-commerce basée sur les microservices, nous permettant d'apprendre plusieurs nouvelles notions et concepts sur l'architecture distribuée.

Tout d'abord, on a commencé par une présentation générale où nous avons expliqué le contexte et les objectifs de ce travail.

Ensuite vient un chapitre dédié à l'analyse des besoins et la conception où nous avons fait un tour sur l'analyse fonctionnelle, non fonctionnelle, les utilisateurs cibles de notre application, le diagramme de cas d'utilisation, pour enfin exposer la version monolithique de notre diagramme de classes.

Le 3ème chapitre tourne autour de l'architecture microservices où on a essayé de l'introduire, parler de ses différences par rapport à son homologue monolithique (avantages & inconvénients), on a exposé aussi les concepts fondamentaux dans le domaine des microservices, en présentant les patterns sur lesquels se base cette architecture. Ceci sans oublier d'évoquer plusieurs autres aspects théoriques se rapportant aux microservices (scalabilité, décomposition), les concepts clés de cette architecture (BD par services, registre de services, passerelle d'API, traçage distribué ainsi que l'utilisation de file d'attente de messages) pour qu'en fin de ce chapitre, nous présentons l'architecture finale englobant tous ces concepts et théories.

Ensuite, pour la partie "Réalisation", nous avons défini les technologies avec lesquelles nous avons travailler ce projet pour enfin donner quelques unes des interfaces dont dispose notre application web "Let's Shop".

En fin de cette conclusion générale et en guise de perspectives, nous pouvons citer le déploiement dans un cluster Kubernetes, l'ajout de la partie CI/CD et l'ajout des tests unitaires et d'intégrité. Les points cités ci-dessus étaient parmi les objectifs que nous avons tracés pour notre projet, cependant, et en raison de manque de temps, nous n'avons pas pu les atteindre, mais certainement, on essaiera de le compléter dans les plus proches délais, pour aboutir à un produit final complet et fonctionnel.

# Bibliographie

- [1] C.CHAKRAVARTHI et al. “Microservices In E-Commerce : An Approach To Avoid Monolithic Architecture”. In : *International Journal of Advanced Science and Technology* 29.8 (2020), p. 4221-4228.
- [2] James LEWIS et Martin FOWLER. *Microservices : A Definition of this New Architectural Term*. 2014. URL : <https://martinfowler.com/articles/microservices.html>.
- [3] Jacopo SOLDANI, Damian Andrew TAMBURRI et Willem-Jan Van Den HEUVEL. “The Pains and Gains of Micro services : A Systematic Grey Literature Review”. In : *Journal of systems and software* 146 (Sept. 2018). DOI : [10.1016/j.jss.2018.09.082](https://doi.org/10.1016/j.jss.2018.09.082).