



## Ecole Nationale d'Ingénieurs de Tunis

Département Technologies de l'Information et de la Communication

## Projet de Fin d'Année II

**Mise en place d'une architecture Microservices pour la gestion de cinémas**

Réalisé par :

**Omar BOUAOUINA & Oussema LOUATI**

Classe :

**2ème année Génie Informatique Groupe 2**

Encadrés par :

**Mme Meriem KASSAR**

O.BOUAQUINA & O.LOUATI

## **Remerciements**

Nos remerciements s'adressent tout d'abord aux membres du Jury, qui nous font l'honneur d'évaluer notre travail de projet de fin d'année.

Nous tenons également à remercier notre encadrante, Madame Meriem Kassar, pour sa présence, son encadrement et ses recommandations dans la réalisation de ce projet de fin d'année.

Finalement, nous tenons à témoigner toute notre reconnaissance à toute personne ayant participé d'une manière ou d'une autre à la réalisation de ce projet.

## Résumé

Cette application a été conçue afin d'offrir aux "Cinéphiles" la meilleure expérience possible en matière de sélection de cinéma, de consultation de son programme, de réservation de tickets, etc. Le développement de cette application repose sur une architecture à base de micro-services et ce projet met en évidence son implémentation.

**Mots clés :** Microservices, Conteneurisation, Spring, Docker, etc.

## Abstract

This application was designed in order to offer to "Cinephiles" the best experience possible with cinema selection, browsing its program, booking tickets, etc. The development of this application is based on microservices architecture and this project focuses on its implementation.

**Key Words :** Microservices, Containerization, Spring, Docker, etc.

# Table des matières

Liste des figures	5
Liste des acronymes	7
Introduction Générale	9
<b>1 Présentation du projet</b>	<b>10</b>
Introduction . . . . .	10
1.1 Cadre du projet . . . . .	10
1.2 Objectifs du projet . . . . .	10
1.3 Étude préalable . . . . .	11
1.3.1 Analyse de l'existant . . . . .	11
1.3.2 Critique de l'existant . . . . .	11
1.3.3 Solution Proposée . . . . .	11
1.4 Démarche méthodologique . . . . .	11
Conclusion . . . . .	13
<b>2 Analyse &amp; Spécification des besoins</b>	<b>14</b>
Introduction . . . . .	14
2.1 Analyse des besoins . . . . .	14
2.1.1 Identification des acteurs . . . . .	14
2.1.2 Besoins fonctionnels . . . . .	15
2.1.3 Besoins non fonctionnels . . . . .	15
2.2 Spécification des besoins . . . . .	16
2.2.1 Diagrammes des cas d'utilisation . . . . .	16
2.2.1.1 Diagramme des cas d'utilisation de l'IT Guy . . . . .	16
2.2.1.2 Diagramme des cas d'utilisation du manager . . . . .	17
2.2.1.3 Diagramme des cas d'utilisation de l'utilisateur . . . . .	17
Conclusion . . . . .	18

## TABLE DES MATIÈRES

5

<b>3 Conception du projet</b>	<b>19</b>
Introduction . . . . .	19
3.1 Introduction aux Microservices . . . . .	19
3.1.1 Qu'est ce que les microservices ? . . . . .	19
3.1.2 Architecture de microservice . . . . .	20
3.1.3 Éléments de microservices . . . . .	20
3.1.4 S'orienter ou pas vers les microservices ? . . . . .	21
3.2 Introduction à la Conteneurization . . . . .	22
3.3 Conception de l'application . . . . .	23
3.3.1 Conception globale de l'application . . . . .	23
3.3.1.1 Architecture globale de l'application . . . . .	23
3.3.1.2 Architecture logicielle de l'application . . . . .	25
3.3.2 Conception détaillée de l'application . . . . .	25
3.3.2.1 Diagramme de classes . . . . .	25
3.3.2.2 Diagrammes de séquences . . . . .	25
Conclusion . . . . .	32
<b>4 Réalisation</b>	<b>34</b>
Introduction . . . . .	34
4.1 Environnement du travail . . . . .	34
4.1.1 Environnement matériel . . . . .	34
4.1.2 Environnement logiciel . . . . .	35
4.2 Technologies adoptées . . . . .	35
4.3 Démonstration du travail . . . . .	37
4.4 Difficultés rencontrées . . . . .	43
Conclusion . . . . .	44
<b>Conclusion Générale</b>	<b>45</b>
<b>Bibliographie</b>	<b>45</b>

# Table des figures

1.1	Cycle de vie du développement suivant l'architecture monolithique . . . . .	12
1.2	Cycle de vie du développement suivant l'architecture de microservice . . . . .	12
2.1	Diagramme des cas d'utilisation (IT Guy) . . . . .	16
2.2	Diagramme des cas d'utilisation (Manager) . . . . .	17
2.3	Diagramme des cas d'utilisation (Utilisateur) . . . . .	17
3.1	Comparaison entre l'architecture monolithique et l'architecture de micro-service . . . . .	20
3.2	Quelques éléments de microservices (stockage séparé des données, déploiement des microservices, etc) . . . . .	21
3.3	Architecture globale de l'application . . . . .	24
3.4	Diagramme de classes de l'application . . . . .	26
3.5	Diagramme de séquences "Gestion des films" . . . . .	27
3.6	Diagramme de séquences "Gestion des cinémas" . . . . .	27
3.7	Diagramme de séquences "Gestion des managers" . . . . .	28
3.8	Diagramme de séquences (IT Guy) . . . . .	29
3.9	Diagramme de séquences (Manager) . . . . .	30
3.10	Diagramme de séquences "Consultation" . . . . .	30
3.11	Diagramme de séquences "Réservation" . . . . .	31
3.12	Diagramme de séquences "Annuler Réservation" . . . . .	31
3.13	Diagramme de séquences système . . . . .	33
4.1	Netflix OSS dans l'architecture de microservice . . . . .	37
4.2	Microservice Spring Boot "Eureka Server" . . . . .	38
4.3	Interface Eureka . . . . .	38
4.4	Application Spring Boot "Films Microservice" comme un "Eureka Client" . .	39
4.5	"Films Microservice" inscrit sur Eureka . . . . .	39
4.6	Microservices inscrits sur Eureka . . . . .	40

*TABLE DES FIGURES*

7

4.7	Non efficacité de l'utilisation de la RAM par les microservices . . . . .	40
4.8	Base de données MYSQL et les microservices en tant que conteneurs Linux	41
4.9	Quelques interfaces pour les conteneurs avec Angular 7 . . . . .	42

O.BOUAOUINA & O.LOUATI

# Liste des acronymes

- API** Application Programming Interface
- CSS** Cascading Style Sheets
- HTTP** HyperText Transfer Protocol
- IT** Information Technology
- JSON** JavaScript Object Notation
- OSS** Open Source Software
- RAM** Random Access Memory
- REST** REpresentational State Transfer
- SDLC** Software Development Life Cycle
- SGBDR** Système de Gestion de Bases de Données Relationnelles
- SOA** Service-Oriented Architecture
- UML** Unified Modeling Language
- URI** Uniform Resource Identifier

# Introduction Générale

Le "changement" est fondamental dans le monde du développement des logiciels. Depuis de nombreuses années, des systèmes ont été construits et améliorés à chaque fois, grâce aux nouvelles technologies, aux nouveaux modèles d'architecture et aux "best practices" qui ont émergé au cours de ces années.

L'architecture de "Microservice" n'est que l'un de ces modèles d'architecture qui ont été conçus pour résoudre des problèmes complexes, ainsi que pour s'adapter à ces "changements" dans le domaine du développement logiciel.

Cette architecture a émergé d'un ensemble commun d'idéologies "DevOps" apparues dans des sociétés telles que Amazon, Netflix, Facebook, Google et bien d'autres, et, contrairement à l'architecture monolithique traditionnelle, l'avènement d'une nouvelle technologie ne sera plus un problème car l'application peut être décomposée en petits services indépendants, chacun d'eux pouvant adopter sa propre technologie et être isolé en cas d'erreur.

Bien que tout cela semble assez prometteur, les microservices sont confrontés à de nombreux défis, notamment en ce qui concerne la gestion de la configuration et la visibilité, c'est-à-dire l'identification, par exemple, de tout "bug" pouvant survenir dans un service bien particulier. Pour cela, des outils sont nécessaires pour le développement, l'automatisation et la surveillance. Nous nous basons sur le principe de la conteneurisation et plus précisément sur la dockerisation afin de développer un exemple de ce type d'architecture : une application permettant de gérer des cinémas.

Ce rapport comporte quatre chapitres. Le Chapitre 1 introduit le cadre et la problématique de notre projet. Le Chapitre 2 met en valeur l'étape d'analyse et de spécification des besoins. Le Chapitre 3 et 4 présentent l'ensemble des méthodes et des techniques élaborées ; le matériel, les outils et les technologies adoptés, ainsi que des illustrations de l'application elle-même et les différents défis et problèmes rencontrés.

# Chapitre 1

## Présentation du projet

### Introduction

Ce chapitre sera dédié à la présentation du cadre du projet, à une analyse de l'existant ainsi qu'à la présentation de la solution proposée.

#### 1.1 Cadre du projet

Ce projet, étant réalisé en binôme, est proposé pendant la deuxième semestre de la deuxième année du cycle d'ingénieur à l'École National d'ingénieurs de Tunis (ENIT), dans le cadre de l'élaboration d'un Projet de Fin d'Année (PFA). Ce dernier concerne le développement d'une application de gestion de cinémas en se basant sur une architecture micro-services.

#### 1.2 Objectifs du projet

Ce projet consiste à développer une architecture de micro-services à travers une application d'exploitation de salles de cinéma, qui facilitera à ses utilisateurs, les "Cinéphiles" tunisiens et tunisiennes, leur expérience dans un cinéma. Cette application permettrait le choix d'un cinéma particulier avec certains critères concernant la date, le film, etc ; de consulter son programme ; de réserver un ticket, ainsi qu'une place ; d'obtenir des informations (sur les places disponibles, sur les tarifs, sur les films, etc.) ; etc. Ce projet présente aussi une initiation au principe de la conteneurisation en utilisant le Docker.

## 1.3 Étude préalable

Dans cette partie, on s'intéressera à l'analyse de l'existant (spécification et critique) et à la définition de la solution proposée.

### 1.3.1 Analyse de l'existant

Après avoir effectué une recherche sur les solutions existantes qui offrent des services introduits précédemment, nous avons constaté que "Pathé Tunis City", qui est un multiplexe sur le site de Géant Tunis City et d'où le nom est celui de diverses entreprises françaises de l'industrie cinématographique, est la seule plateforme de gestion de cinéma en Tunisie.

### 1.3.2 Critique de l'existant

Dans cette partie on va présenter les limites de la solution existante :  
Tout d'abord, Pathé Tunis City n'est pas le seul cinéma tunisien, le 7ème art est à la hausse en Tunisie, tout comme le nombre de cinémas, mais cette application n'offre ses services qu'à ceux qui vont entrer dans ce cinéma. Ensuite, bien que nous ne soyons pas tout à fait sûrs, l'application semble être monolithique. Bien que cette architecture a sa part des avantages, elle présente aussi certains inconvénients, notamment la défaillance de l'application en cas de défaillance d'un composant, l'élévation de la maintenabilité, l'arrêt de tout le processus en cas de freinage d'une ligne de code, etc.

### 1.3.3 Solution Proposée

En tenant compte des limitations de la solution existante, nous avons poursuivi l'idée de réunir tous ces cinémas en une seule plateforme ou un "Cineplex", qui permet à son utilisateur de parcourir le catalogue d'un cinéma qu'il choisit, en fonction du gouvernerat, du prix du ticket, du genre de film, etc. tout en suivant une architecture de microservice pour, non seulement la mettre en place, mais encore pour assurer les avantages que le style architectural monolithique ne pouvait pas offrir.

## 1.4 Démarche méthodologique

Le développement d'une solution repose sur une méthodologie cohérente et un processus clairement défini allant du point A au point B. Dans le cas de notre projet, et en vue

de le mener à bien, nous avons suivi les étapes du processus SDLC (Software Development Life Cycle), le cycle de vie de développement logiciel, mais pas selon l'approche monolithique. Alors que cela commence avec les mêmes étapes d'identification des problèmes, la planification et la conception, c'est dans les étapes du codage, du test, de déploiement et de maintenance que nous nous trouvons divergés de l'approche monolithique classique (Voir figure 1.1) vers l'approche de microservices (Voir figure 1.2) : au lieu de coder, tester, déployer et maintenir le tout ensemble, chaque membre d'une équipe s'occupe d'effectuer ces tâches-là pour chaque microservice qui lui est associé.

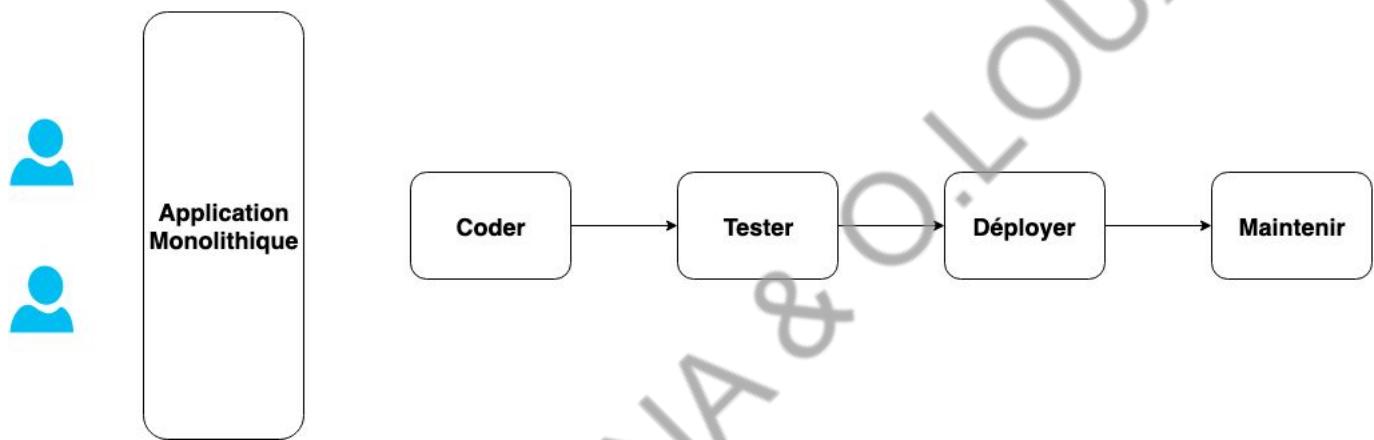


FIGURE 1.1 – Cycle de vie du développement suivant l'architecture monolithique

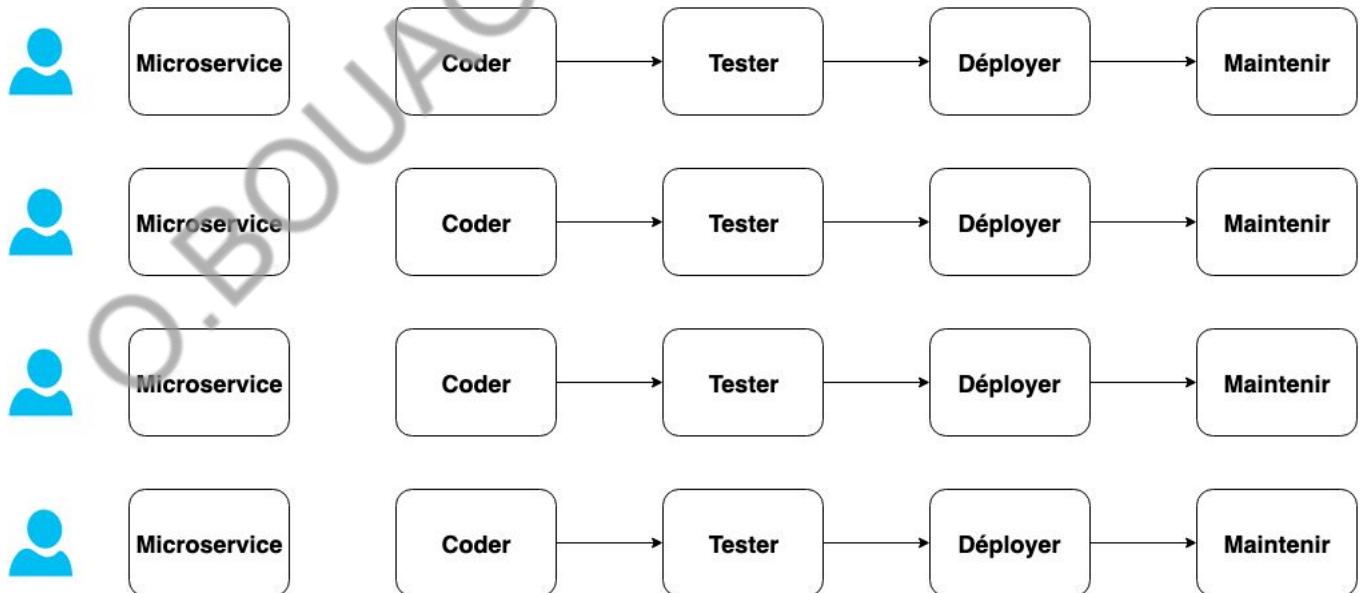


FIGURE 1.2 – Cycle de vie du développement suivant l'architecture de microservice

## Conclusion

Dans ce chapitre, nous avons décrit le contexte général de notre projet, notamment le cadre et les objectifs du projet et, ainsi que présenté l'état de l'art et de l'existant. Dans le deuxième chapitre, nous nous intéresserons à l'analyse et la spécification des besoins fonctionnels et non fonctionnels.

# Chapitre 2

## Analyse & Spécification des besoins

### Introduction

Après que nous ayons présenté notre projet de manière générale, nous entamons dans ce chapitre l'analyse et la spécification des besoins auxquels doit répondre notre application, d'une manière globale, pour ensuite donner une description plus détaillée du projet.

### 2.1 Analyse des besoins

L'application doit être opérationnelle, évolutive, conviviale et adaptative, c'est pourquoi elle doit satisfaire les exigences de ces utilisateurs. Nous présentons dans ce qui suit les différents acteurs de l'application ainsi que les besoins fonctionnels et non fonctionnels.

#### 2.1.1 Identification des acteurs

Les acteurs représentent les différents rôles joués par des entités externes (utilisateurs humains, dispositifs matériels ou autres systèmes) qui interagissent avec le système directement, en ayant un accès à ses services et ses données, chacun avec ses priviléges :

- **Utilisateur** : C'est l'acteur principal dans notre application. En tant que simple visiteur, que ce soit quelqu'un sans compte ou seulement non connecté, il peut consulter le programme d'un cinéma de son choix. En tant qu'un utilisateur connecté, il peut effectuer une réservation.
- **Manager du cinéma** : Ayant plus de priviléges, le manager peut effectuer des changements en termes de services, de tarifs et du programme du cinéma qui lui est associé.
- **IT Guy** : C'est la personne avec le plus de priviléges, il a accès aux informations

concernant les films et les cinémas ainsi qu'aux comptes des managers, il peut donc les créer, modifier, supprimer, etc.

### 2.1.2 Besoins fonctionnels

Étant une application web, elle doit assurer la sécurité et la fiabilité de l'accès à ses services. Ainsi, les utilisateurs peuvent réaliser différentes opérations dans les limites de leurs priviléges.

Les services diffèrent d'un acteur à un autre, c'est pourquoi une description des besoins spécifiques pour chacun d'eux est nécessaire, pour offrir un meilleur aperçu de l'application.

#### Utilisateur :

En tant que visiteur, l'utilisateur peut créer un compte, consulter le programme d'un cinéma (en ayant la possibilité d'utiliser des filtres de recherche) et savoir sur des informations. Une fois connecté à l'application avec son compte, il peut en plus faire une réservation d'un ou plusieurs tickets (et d'une ou des place disponibles s'il veut).

#### Manager du cinéma :

Le manager du cinéma peut, après avoir connecté avec le compte fourni par le IT Guy, effectuer des mises à jour du programme du cinéma qui lui est associé, des tarifs ainsi que des services du tel cinéma.

#### IT Guy :

Quant à l'IT Guy, il est le plus privilégié : il s'occupe de la gestion de l'application, que ce soit des informations relatives aux cinémas et aux films, ou des comptes des managers des cinémas.

### 2.1.3 Besoins non fonctionnels

Les besoins non fonctionnels représentent les exigences et les contraintes techniques pour lequel notre application doit répondre pour assurer son bon fonctionnement, qui sont :

#### L'ergonomie :

L'interface de l'application doit être représenté dans un cadre compréhensible et convivial auprès de l'utilisateur pour le bon fonctionnement.

#### La fiabilité :

Tout utilisateur doit s'authentifier avant d'accéder aux contenus principaux présentés par l'application.

### La rapidité :

Le temps de traitement de la commande doit être suffisamment court pour l'accès souhaitable par l'utilisateur.

### La flexibilité :

L'application doit être adaptable à tout ajout ou modification de modules.

## 2.2 Spécification des besoins

Pour mettre en évidence les principales fonctionnalités de l'application et les relations entre celles-ci et les utilisateurs, nous avons utilisé le langage de modélisation unifié, ou UML, pour créer des diagrammes des cas d'utilisation qui décrivent les attentes de chaque utilisateur.

### 2.2.1 Diagrammes des cas d'utilisation

Nous allons présenter, dans cette partie, les diagrammes des cas d'utilisation raffinés, pour chaque acteur.

#### 2.2.1.1 Diagramme des cas d'utilisation de l'IT Guy

La figure 2.1 représente le diagramme des cas d'utilisation de l'IT Guy :

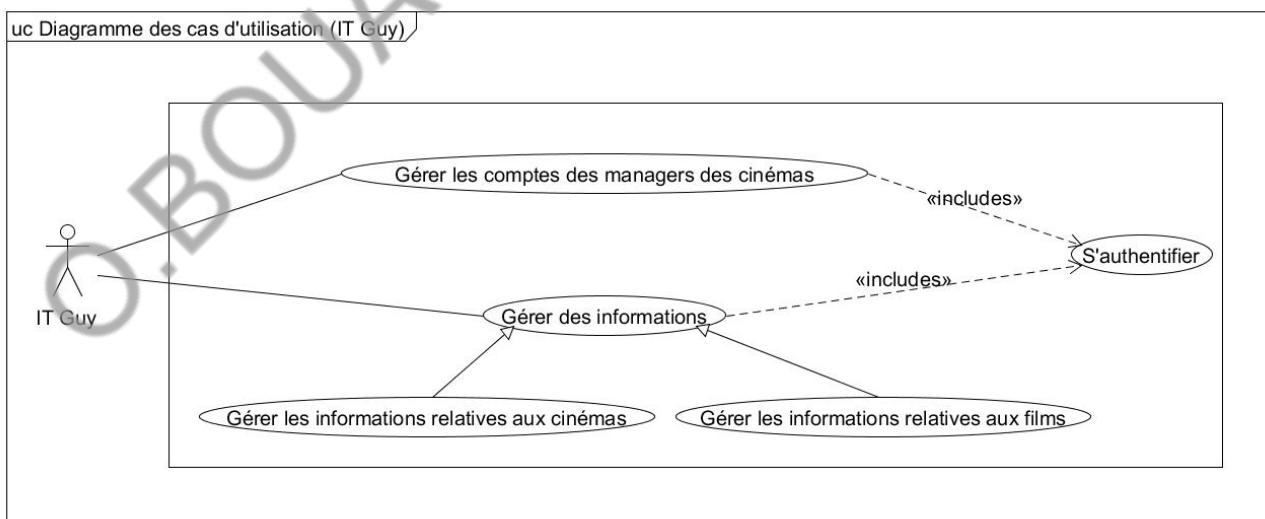


FIGURE 2.1 – Diagramme des cas d'utilisation (IT Guy)

### 2.2.1.2 Diagramme des cas d'utilisation du manager

La figure 2.2 représente le diagramme des cas d'utilisation du manager :

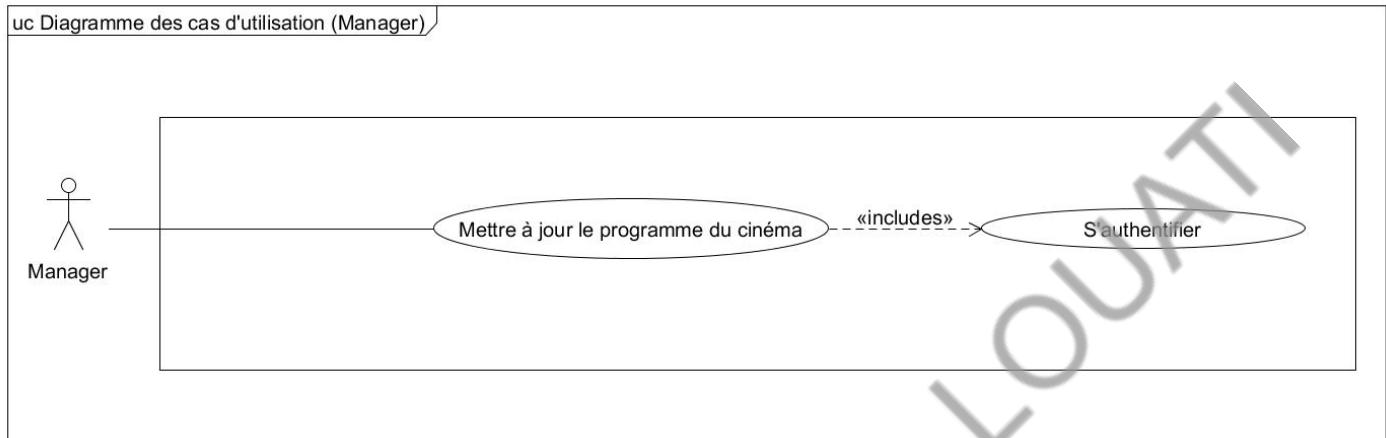


FIGURE 2.2 – Diagramme des cas d'utilisation (Manager)

### 2.2.1.3 Diagramme des cas d'utilisation de l'utilisateur

La figure 2.3 représente le diagramme des cas d'utilisation de l'utilisateur :

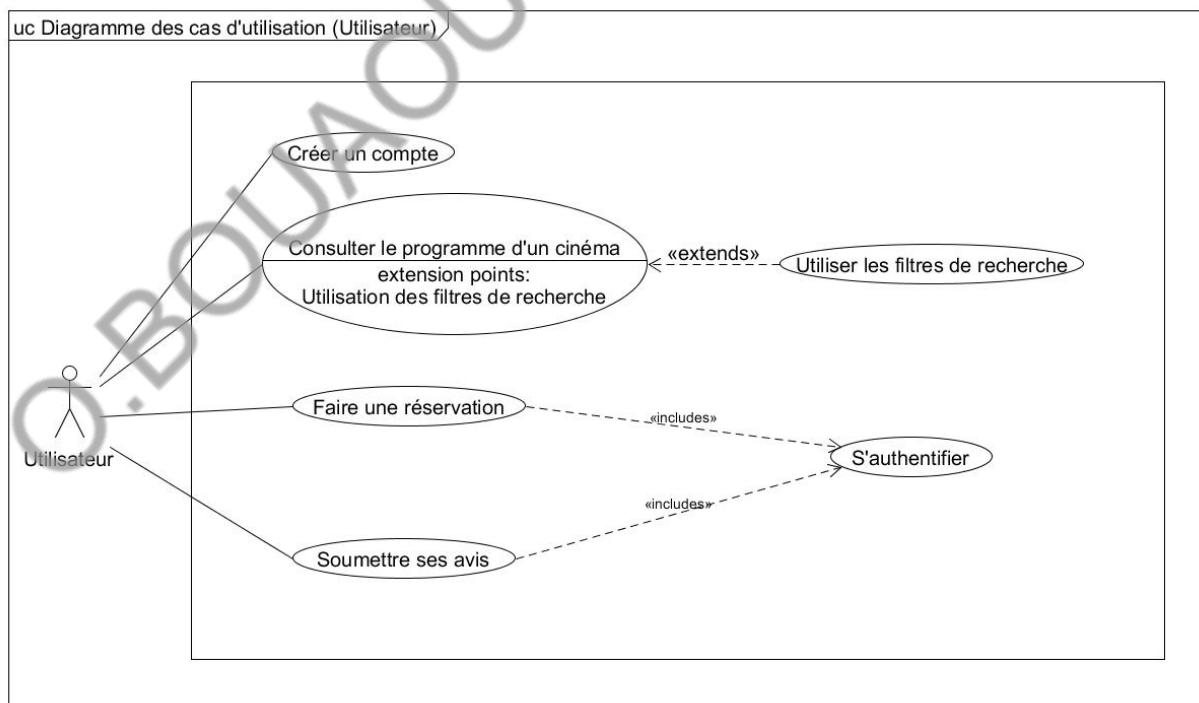


FIGURE 2.3 – Diagramme des cas d'utilisation (Utilisateur)

## Conclusion

Ce chapitre a été dédié à l'analyse et la spécification des besoins, ce qui nous a permis de mieux comprendre les objectifs de ce projet et nous a donné une vue d'ensemble sur les fonctionnalités fondamentales de l'application.

O.BOUAOUINA & O.LOUATI

# Chapitre 3

## Conception du projet

### Introduction

Après la présentation du projet ainsi que la spécification des besoins fonctionnelles et non fonctionnelles, nous entamons dans ce chapitre avec l'étape de la conception, qui nous permet de mettre en oeuvre les moyens, les méthodes ainsi que les techniques réalisées le long de ce projet.

### 3.1 Introduction aux Microservices

Cette section propose une introduction aux microservices, leur architecture et la motivation à leur utilisation.

#### 3.1.1 Qu'est ce que les microservices ?

Le terme "Microservices" décrit un style architectural de développement qui a connu une croissance de tendances récentes, et qui implique, contrairement à l'architecture monolithique où l'application est développée comme une seule unité, le développement d'une application sous la forme d'un certain nombre de petits services autonomes, chacun s'exécutant dans son propre processus et communiquant entre eux par le biais de mécanismes légers, ce qui montre le niveau d'indépendance des services, la portée limitée de chaque service ainsi que la décomposition de l'application en des services faiblement couplés (Voir figure 3.1), et c'est pour ça que la plupart des sites Web à grande échelle, notamment Netflix, Amazon et eBay, sont passés d'une architecture monolithique à une architecture de microservice.

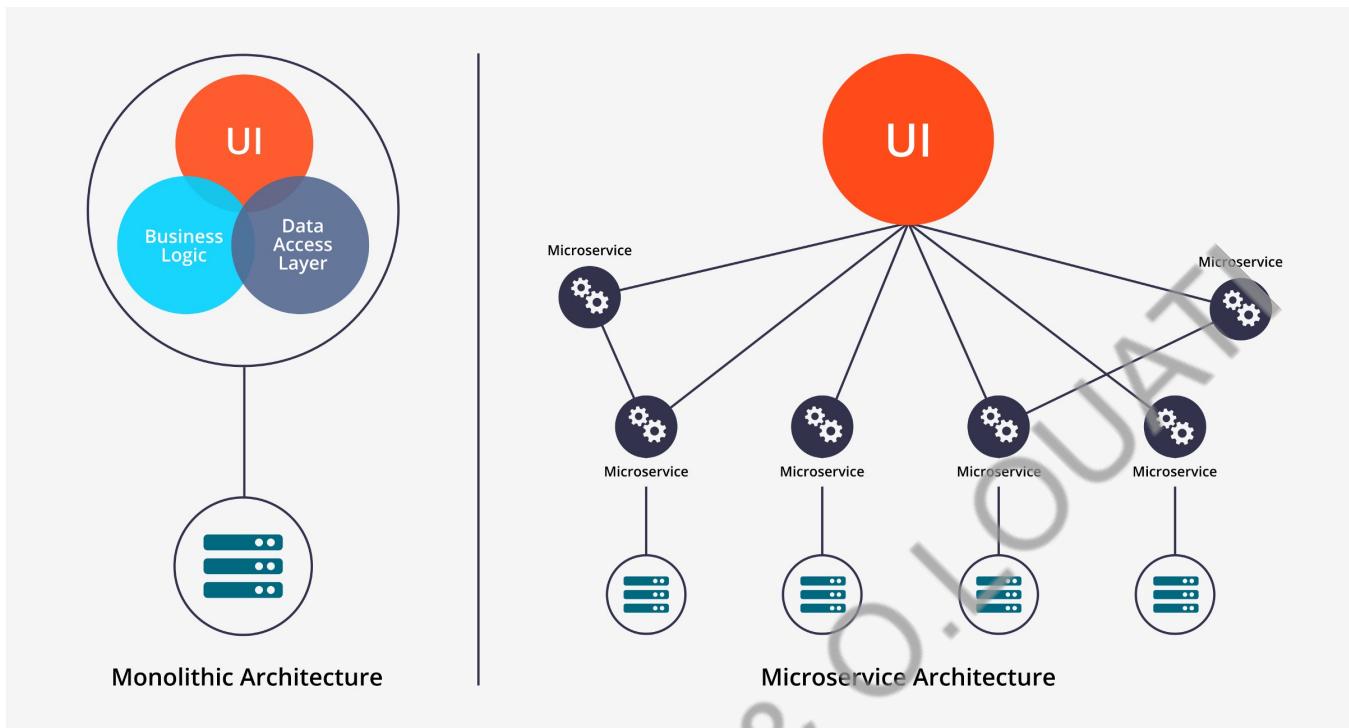


FIGURE 3.1 – Comparaison entre l’architecture monolithique et l’architecture de micro-service

### 3.1.2 Architecture de microservice

L’architecture de microservice est un ensemble de pratiques qui sont censées accroître la rapidité et l’efficacité de développement et de la gestion de solutions logicielles à grande échelle. Cet ensemble de pratique est agnostique par rapport à la technologie, ça veut dire qu’il n’existe pas une technologie ou un langage de programmation unique pour concevoir des microservices, mais plutôt presque tous les langages de programmation peuvent être utilisés pour créer des microservices. Il s’agit en fait d’appliquer des modèles architecturaux et des principes qui permettront d’obtenir cette architecture de microservice.

### 3.1.3 Éléments de microservices

La terminologie de microservice est riche et plein de concepts et de modèles de conception qui touchent la plupart des aspects techniques du système, depuis le stockage des données jusqu’à l’interface utilisateur, la sécurité, la surveillance, le déploiement, etc (Voir figure 3.2). Selon la complexité de l’architecture, pas tous les concepts des microservices pourraient être utilisés, mais nous pourrions le faire que la complexité de l’architecture se développer de plus en plus.

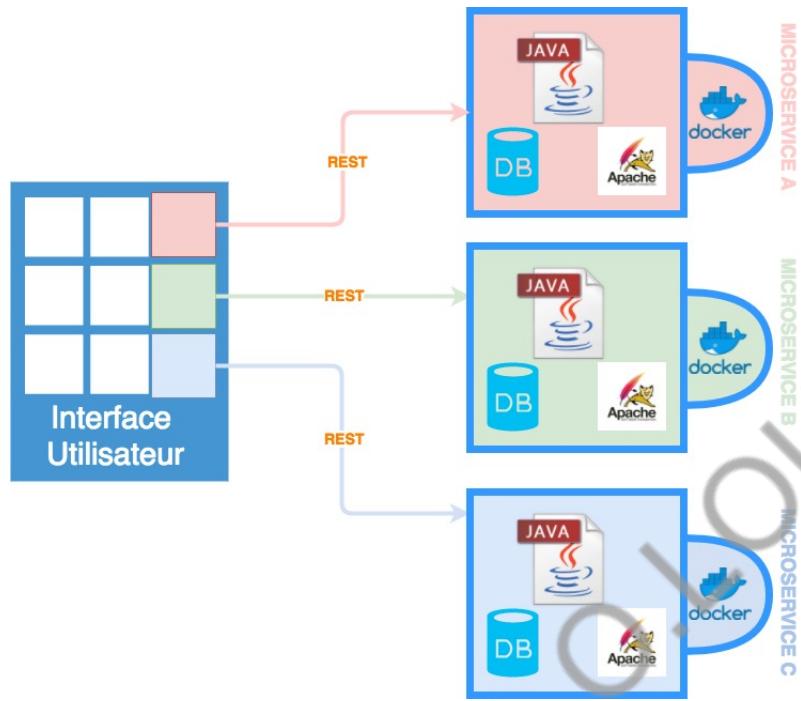


FIGURE 3.2 – Quelques éléments de microservices (stockage séparé des données, déploiement des microservices, etc)

### 3.1.4 S'orienter ou pas vers les microservices ?

Les microservices offre de nombreux avantages tels que :

- La livraison et le déploiement continu de grandes applications complexes grâce à une facilité de maintenance améliorée, vu chaque service étant relativement petit, il est donc plus facile à comprendre et à modifier, meilleure testabilité grâce à la taille réduite des services et donc à la rapidité de test, meilleure déployabilité puisque chaque service peut être déployé indépendamment, etc.
- La taille de chaque microservice, qui est relativement petite, ce qui permet aux développeurs de le comprendre plus facilement, aux IDEs de rendre rapidement ces développeurs plus productifs, et à l'application de démarrer rapidement et donc accélérer les déploiements.
- Meilleure isolation des fautes, par exemple, s'il y a une fuite de mémoire dans un service, seul ce service sera affecté, les autres services continueront à gérer les demandes, contrairement à une architecture monolithique dans laquelle un composant défectueux peut détruire le système entier.

- La liberté de choisir une nouvelle technologie lors du développement d'un nouveau service ou de la modification d'un service existant, éliminant ainsi tout engagement à long terme envers une technologie.

Mais, ils souffrent également d'un certain nombre d'inconvénients, parmi lesquels on peut citer :

- La difficulté de savoir décomposer proprement l'application en des services.
- Le fait de faire face à la complexité supplémentaire de la création d'un système distribué, étant donné que les développeurs doivent mettre en œuvre le mécanisme de communication inter-services et faire face aux défaillances partielles, la mise en œuvre de requêtes couvrant plusieurs services est plus difficile, les outils de développement ainsi que les IDEs sont axés sur la création d'applications monolithiques et ne fournissent pas de support explicite pour le développement d'applications distribuées, etc.
- La complexité du déploiement et, dans la phase de production, il y'a encore la complexité opérationnelle du déploiement ainsi que la gestion d'un système constitué de plusieurs services différents.
- L'augmentation de la consommation de mémoire, vu que cette architecture remplace un nombre N d'instances d'une application monolithique par NxM instances de services et, si chaque service fonctionne dans sa propre machine virtuelle, qui est dans la plupart des cas pour isoler les instances, alors il y'aura une surcharge.

## 3.2 Introduction à la Conteneurization

Passer à une nouvelle architecture d'application comme l'architecture de microservice n'est pas aussi simple qu'il y paraît, car vous devez apporter certains changements dans les anciennes pratiques et dans la manière de surveiller automatiser et exécuter une application basée sur des microservices, et plus important encore de choisir le bon environnement d'exécution dans lequel vous pouvez exécuter vos microservices.

L'utilisation des conteneurs est l'un des meilleurs choix pour exécuter une application basée sur des microservices. Puisque Les conteneurs sont des environnements d'exécution isolés et légers, ils ressemblent à des paquetages qui incluent tout ce qui est nécessaire pour exécuter votre service (code, dépendances, bibliothèques, fichiers binaires). Les conteneurs

sont donc par nature des environnements indépendants, ce qui constitue une solution pour la portabilité des applications.

Cette isolation des conteneurs se produit au niveau du système d'exploitation, ce qui signifie qu'une seule instance de système d'exploitation peut prendre en charge un certain nombre de conteneurs, chacun d'entre eux s'exécutant dans son propre environnement d'exécution. Avec ce type d'isolation, il est maintenant possible de placer plusieurs microservices sur un seul serveur.

Les applications basées sur des microservices ont des charges de travail très erratiques et ce qui rend les conteneurs parfaitement adaptés à l'exécution de microservices est leur nature légère. Les conteneurs sont plus efficaces à l'initialisation par exemple, ils démarrent généralement en secondes, voire en millisecondes.

Cette rapidité d'instanciation correspond parfaitement aux caractéristiques de charge de travail erratiques associées aux microservices. Si nous parlons de performance, les conteneurs sont l'environnement d'exécution évident pour les architectures microservices.

### 3.3 Conception de l'application

Après avoir donné une idée sur les microservices, la conteneurisation ainsi que l'automatisation, nous dédions cette section pour la conception de l'application. Nous démontrons, en premier lieu, la conception globale de cette application, à travers l'architecture adoptée, d'une manière globale ainsi que du côté logiciel, pour donner, en second lieu, la conception détaillée, à travers le diagramme de classes et les diagrammes de séquences.

#### 3.3.1 Conception globale de l'application

Pour assurer que l'application soit facile à améliorer, à évoluer, ainsi que soit adaptable à d'autres technologies, l'architecture de microservice était celle à utiliser.

##### 3.3.1.1 Architecture globale de l'application

Avec une architecture de microservice, nous pouvons diviser le domaine de l'application qui correspond à toute la couche métier en des sous-domaines, chaque sous-domaine correspondant à une partie de cette couche métier : ce sont les microservices. Cette architecture est démontrée dans la figure 3.3 :

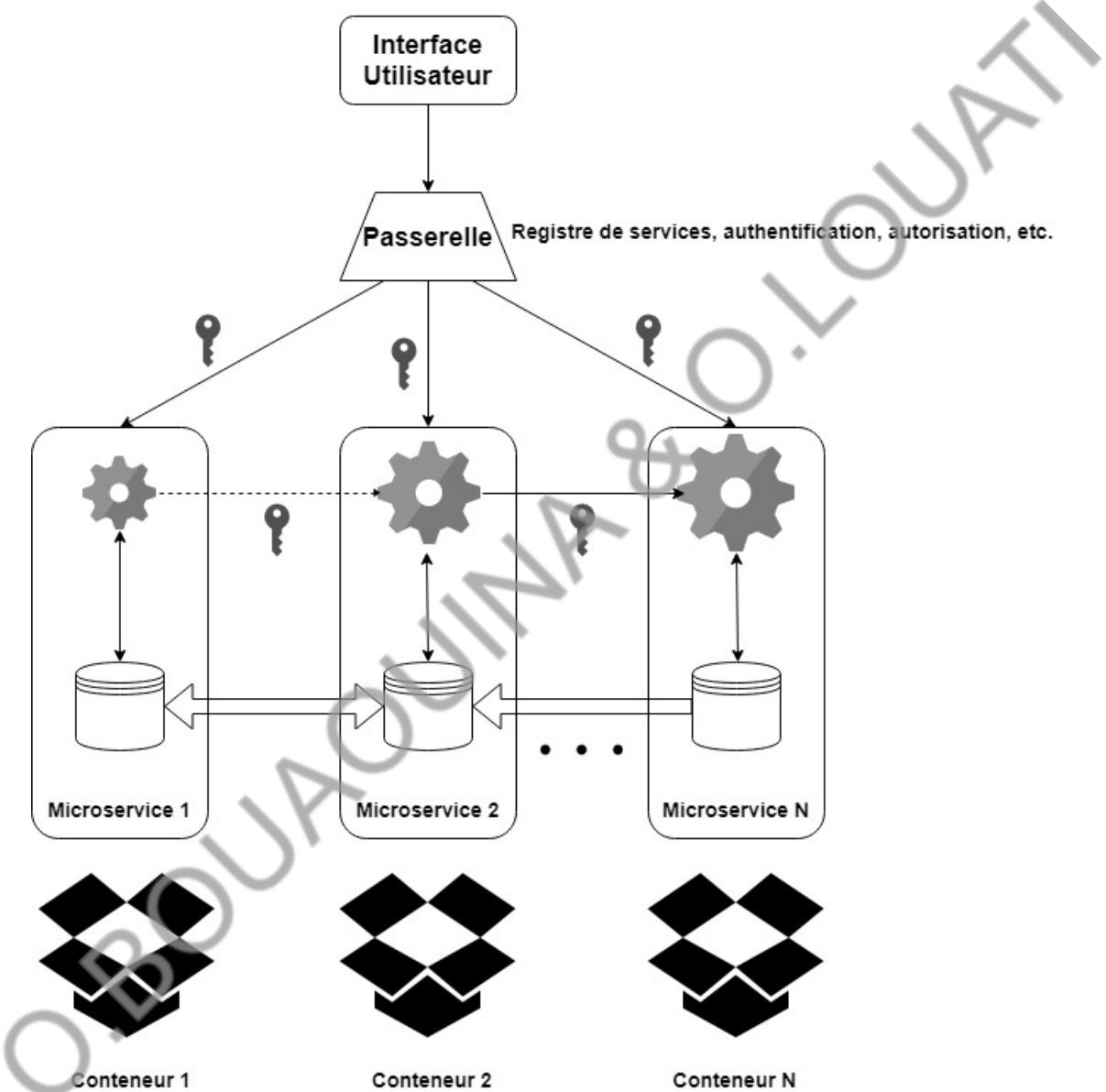


FIGURE 3.3 – Architecture globale de l'application

### 3.3.1.2 Architecture logicielle de l'application

Pareillement à l'architecture orientée services (SOA), nous avons implémenté les microservices comme des service RESTful, dont chaque microservice est accessible via un URI, en utilisant les méthodes standards du protocole HTTP (POST, GET, PUT et DELETE), et le format JSON (JavaScript Object Notation) pour la représentation des données.

### 3.3.2 Conception détaillée de l'application

Nous enchaînons maintenant avec la modélisation des différents composants de notre application, les différentes relations entre-eux, ainsi que les différentes interactions des acteurs avec l'application.

#### 3.3.2.1 Diagramme de classes

Nous représentons, à ce stade, les diverses entités qui constituent notre application, les relations entre celles-ci ainsi que leurs natures. Ce diagramme est important puisqu'il met en relief la manière dont nous avons traduit et concrétisé nos besoins en décomposant ces derniers en des éléments introduits lors du développement et du passage d'une solution imaginée à une solution informatique concrète (Voir figure Figure 3.4).

#### 3.3.2.2 Diagrammes de séquences

A travers les diagrammes de séquences, nous pouvons suivre les échanges entre les différents objets et acteurs du système en fonction du temps, ce qui se traduit par une vue dynamique des interactions entre ces derniers. Nous commençons par présenter les diagrammes de séquences détaillant les différentes interactions avant de réunir le tout en un diagramme de séquences système :

##### Diagramme de séquences "Gestion des films" :

Le diagramme de séquences de la figure 3.5 décrit les interactions possibles entre le IT Guy et les films : **Diagramme de séquences "Gestion des cinémas"** :

Le diagramme de séquences de la figure Figure 3.5 décrit les interactions possibles entre le IT Guy et les cinémas :

##### Diagramme de séquences "Gestion des managers" :

Le diagramme de séquences de la figure Figure 3.7 décrit les interactions possibles entre le IT Guy et les managers : **Diagramme de séquences (IT Guy)** :

## CHAPITRE 3. CONCEPTION DU PROJET

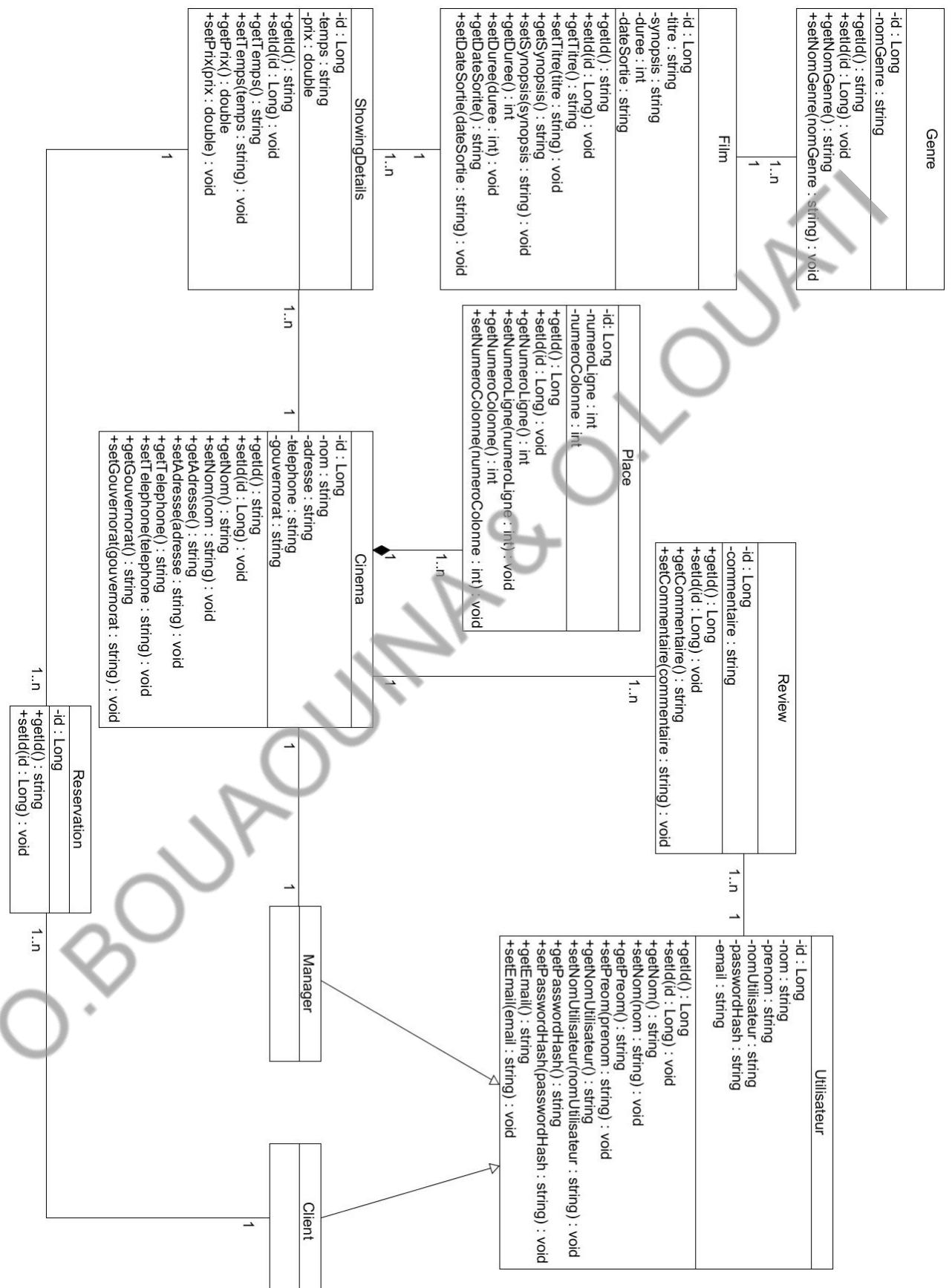


FIGURE 3.4 – Diagramme de classes de l’application

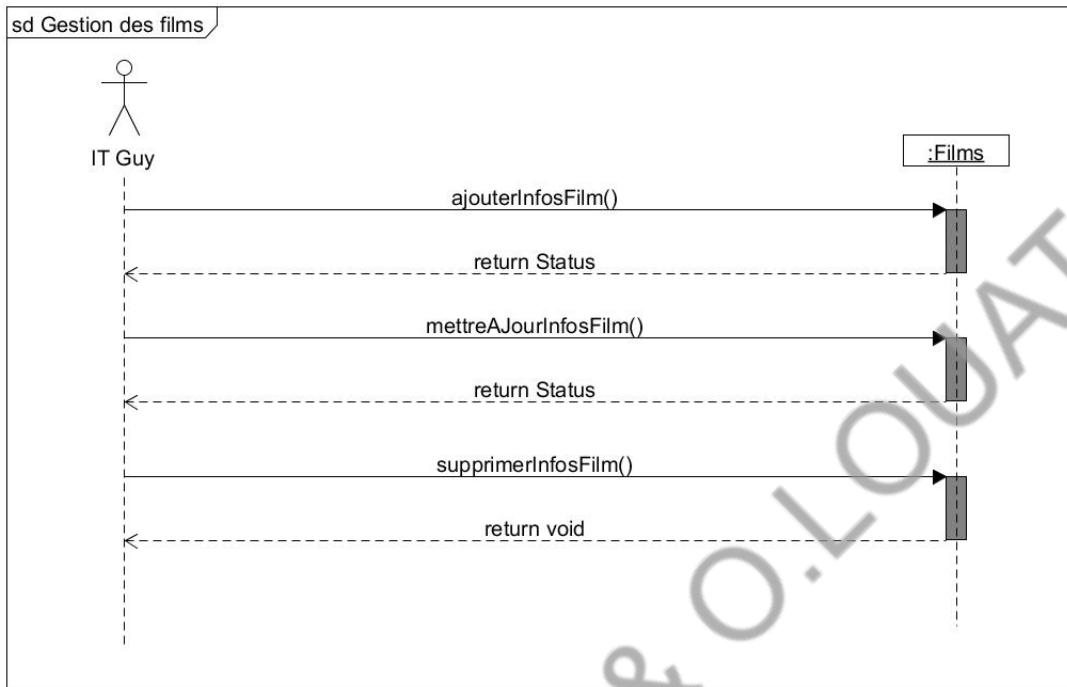


FIGURE 3.5 – Diagramme de séquences "Gestion des films"



FIGURE 3.6 – Diagramme de séquences "Gestion des cinémas"



FIGURE 3.7 – Diagramme de séquences "Gestion des managers"

Le diagramme de séquences de la figure 3.8 décrit les interactions possibles de l'IT Guy : **Diagramme de séquences (Manager)** :

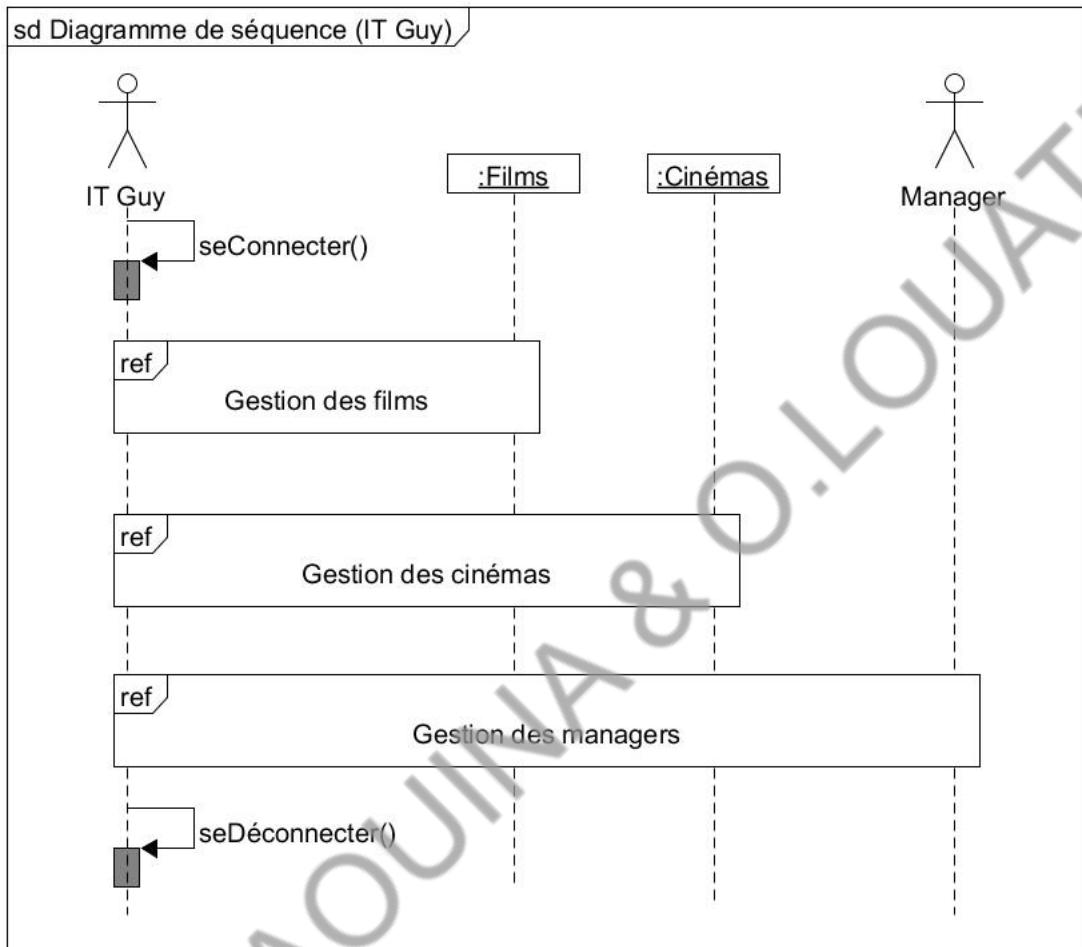


FIGURE 3.8 – Diagramme de séquences (IT Guy)

Le diagramme de séquences de la figure 3.9 décrit les différents scénarios du manager de cinéma :

#### Diagramme de séquences "Consultation" :

Le diagramme de séquences de la figure 3.10 décrit l'étape de consultation du programme des cinémas par un visiteur ou un client non connecté :

#### Diagramme de séquences "Réservation" :

Le diagramme de séquences de la figure 3.11 présente le scénario de réservation d'un ticket et d'une place ainsi que l'effectuation de paiement :

#### Diagramme de séquences "Annuler Réservation" :

Le diagramme de séquences de la figure 3.12 présente le scénario d'annulation d'une réservation :

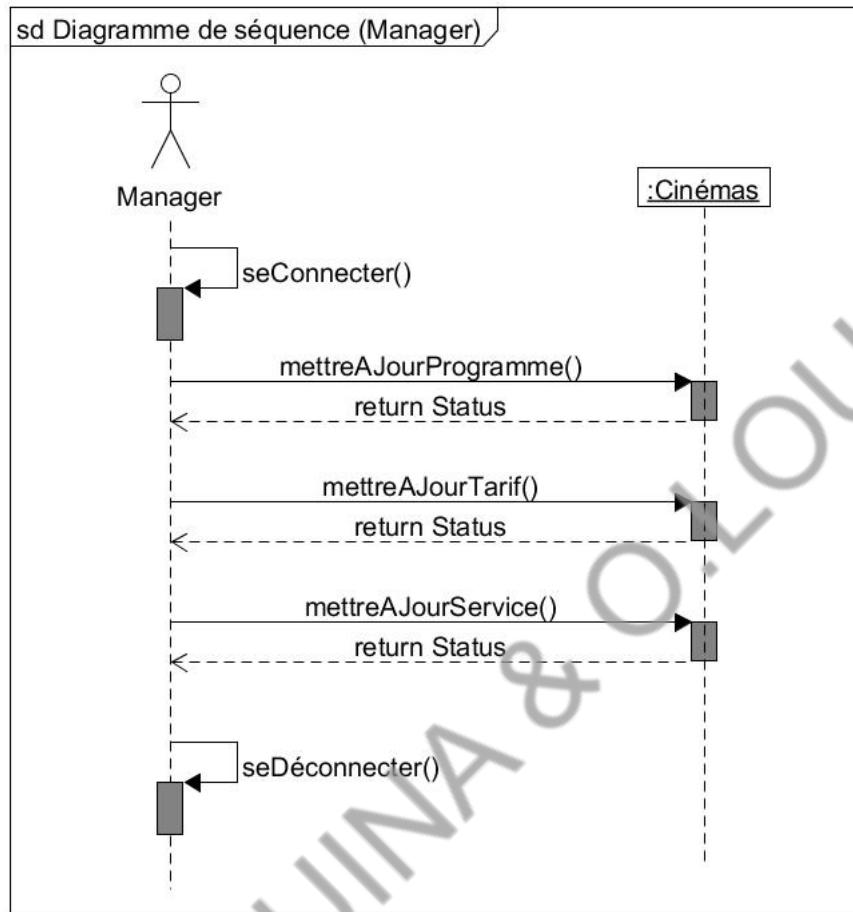


FIGURE 3.9 – Diagramme de séquences (Manager)

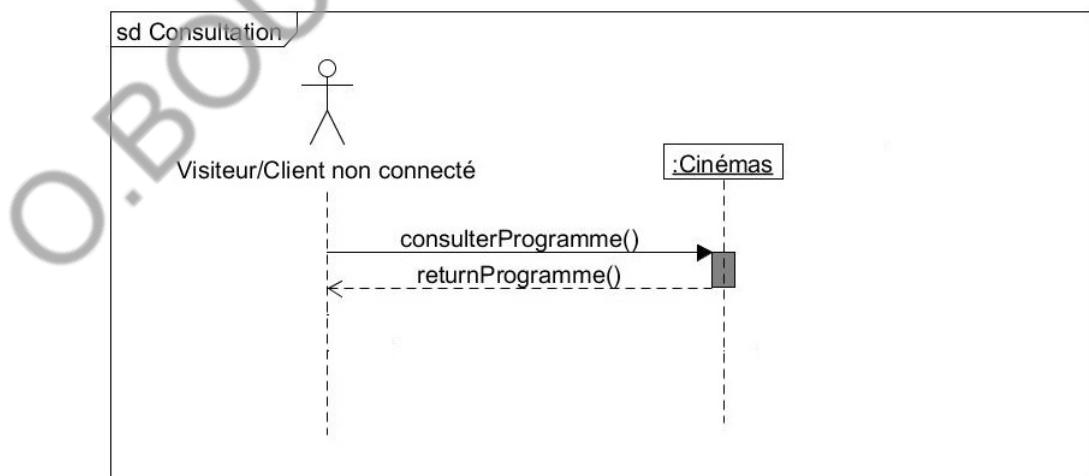


FIGURE 3.10 – Diagramme de séquences "Consultation"

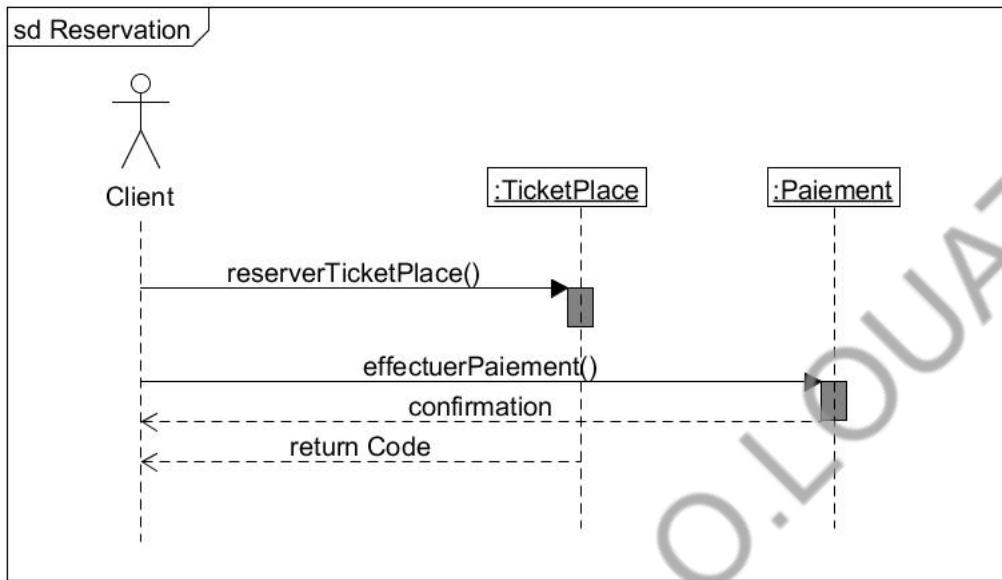


FIGURE 3.11 – Diagramme de séquences "Réservation"

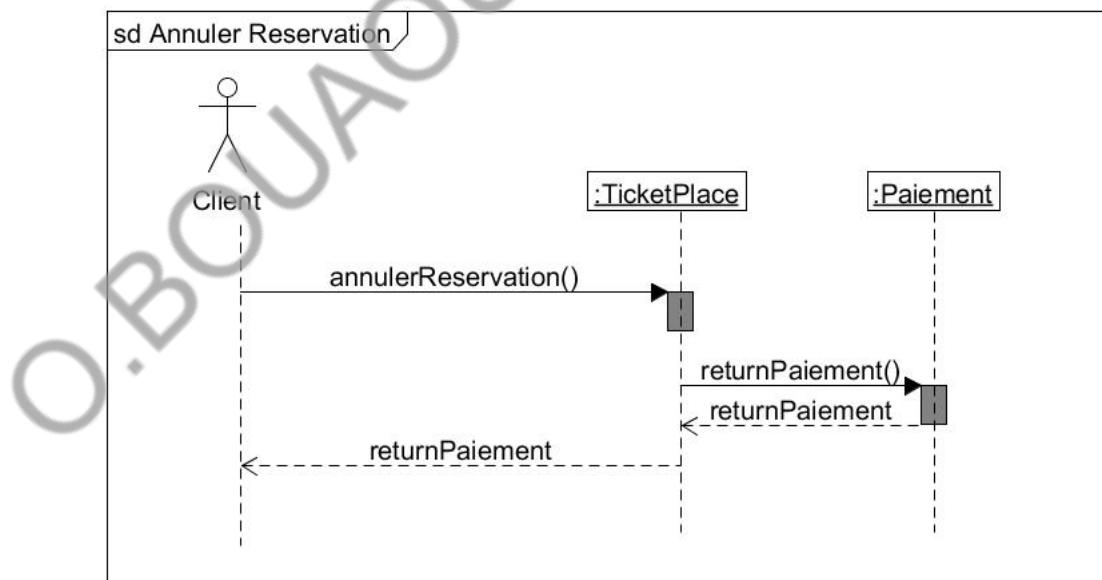


FIGURE 3.12 – Diagramme de séquences "Annuler Réservation"

**Diagramme de séquences système :**

Le diagramme de séquences de la figure 3.13 réunit tous les diagrammes que nous avons vus :

## Conclusion

Dans ce chapitre, nous avons présenté l'étude conceptuelle de notre projet, nous avons commencé par exposer l'architecture globale de notre application, en décrivant quels sont les microservices et comment les réaliser à l'aide de conteneurs. pour finir avec une conception détaillée en utilisant des diagrammes UML.

Toute cette conception prépare tout le nécessaire pour passer à la dernière étape qui consiste à réaliser l'application.

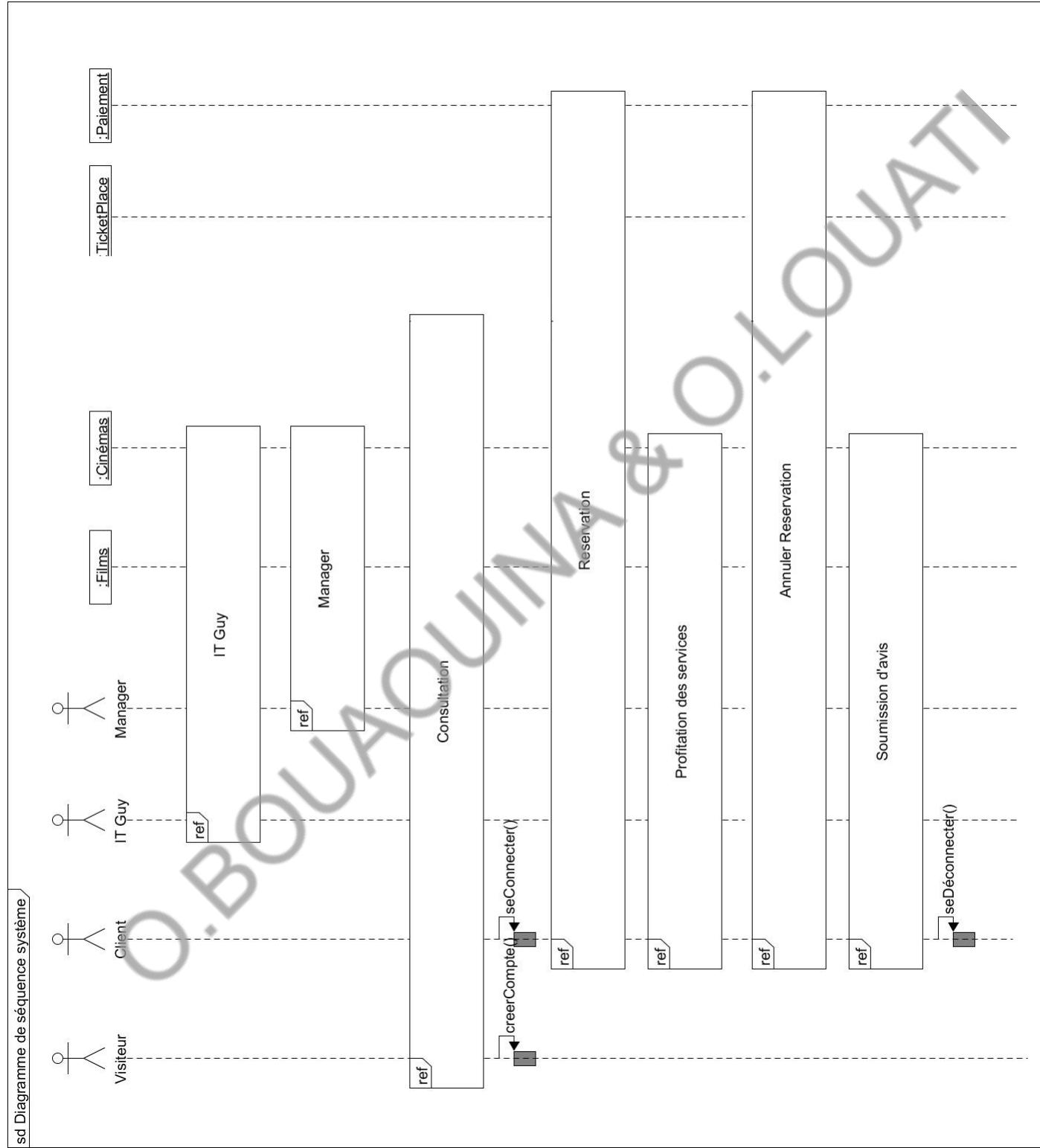


FIGURE 3.13 – Diagramme de séquences système

# Chapitre 4

## Réalisation

### Introduction

Avec le dernier chapitre, nous présentons les éléments et les outils utilisés pour la réalisation de notre projet, à savoir l'environnement du travail, matériel et logiciel, et les choix technologiques adoptés.

#### 4.1 Environnement du travail

Dans cette section, nous mettons en oeuvre l'ensemble des ressources matériels et logiciels qui constituent notre environnement de travail.

##### 4.1.1 Environnement matériel

Pour ce qui concerne l'environnement du travail matériel, il se résume à deux ordinateurs dont les caractéristiques sont les suivants :

###### Ordinateur 1 : Lenovo Y520

- Processeur : Intel Core i5-7300HQ
- Mémoire : 8GB
- Carte Graphique : NVIDIA GeForce GTX 1050 (Notebook)
- Système d'Exploitation : Windows 10

###### Ordinateur 2 : Lenovo Y530

- Processeur : Intel Core i5-8300H
- Mémoire : 8GB
- Carte Graphique : NVIDIA GeForce GTX 1050 (Notebook)
- Système d'Exploitation : Windows 10

### 4.1.2 Environnement logiciel

Nous entamons dans cette partie avec les différents logiciels auxquels nous avons eu recours le long du projet :

**Docker** : Docker est une plateforme de conteneurisation qui regroupe votre application et toutes ses dépendances, sous la forme d'un conteneur Docker, afin de garantir que votre application fonctionne de manière transparente dans n'importe quel environnement.

**IntelliJ IDEA** : C'est un environnement du développement intégré Java, conçu par JetBrains, et qu'on a préféré aux autres IDEs en raison de ses fonctionnalités supplémentaires telles que l'auto-complétion intelligente du code.

**MySQL** : C'est un SGBDR parmi les plus utilisés au monde, écrit en C, C++ et C#.

**Node.js** : C'est une plateforme open-source, qui est un environnement d'exécution JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge.

**Node.js Package Manager (npm)** : C'est le gestionnaire de paquets officiel de Node.js, faisant partie de la plateforme, qui les dépendances pour une application Node.js ainsi que l'installations des unes disponibles sur le dépôt npm.

**Overleaf** : C'est un outil collaboratif, en ligne, et en temps réel, pour la rédaction ainsi que la compilation en PDFs des programmes écrits en langage LaTeX.

**UMLet** : C'est un logiciel de dessin de diagrammes UML open-source et écrit en Java.

## 4.2 Technologies adoptées

Passons maintenant aux choix technologiques que nous avons opté pour, plus précisément les langages de programmation ainsi que les framework, qui sont :

**Java** :

Java est un langage de programmation polyvalent de haut niveau qui est couramment utilisé aujourd'hui pour créer des applications Web et mobiles. Nous avons choisi de travailler avec ce langage de programmation pour ses multiples avantages, parmi lesquels on peut citer :

- La simplicité, la sécurité et la robustesse.
- Le fait d'être orienté objet, indépendant de la plateforme, c'est à dire que le code Java que vous écrivez sur un système d'exploitation s'exécutera sur d'autres platesformes sans aucune modification, ainsi que multithread.
- Le fait d'être riche en des APIs, qui sont l'ensemble de commandes ou méthodes de communication entre diverses activités telles que la connexion à une base de données, la mise en réseau, les entrées sorties, les utilitaires, etc.

### Spring Boot :

Spring Boot est un projet construit sur le framework Spring. Il fournit un moyen plus simple et plus rapide d'installer, de configurer et d'exécuter des applications simples et Web. Vous pouvez commencer à utiliser des configurations minimales sans avoir besoin d'une configuration complète de la configuration Spring dès le début.

### Spring Cloud :

Spring Cloud est un ensemble d'outils fournissant des solutions à certains des modèles courants rencontrés lors de la création de systèmes distribués, parmi ces outils on peut citer :

- La gestion de la configuration
- La découverte de services
- Les "Circuit Breakers"
- Les sessions distribuées

### Netflix OSS :

Netflix OSS est un ensemble de framework et de bibliothèques que Netflix a écrit pour résoudre certains problèmes liés aux systèmes distribués. Des concepts tels que les modèles de découverte de service, d'équilibrage de charge, de tolérance aux pannes, ainsi que d'autres, sont extrêmement importants pour les systèmes distribués évolutifs, et Netflix apporte des solutions intéressantes à ces problèmes que nous avons eu recours à certaines de ces solutions (Voir figure 4.1), notamment :

- **ZUUL** : En tant que service de périphérie, Zuul est conçu pour permettre le routage, la surveillance, la résilience et la sécurité dynamiques.
- **Eureka** : Eureka Server contient les informations sur toutes les applications de service client. Chaque microservice s'enregistrera sur le serveur Eureka et celui-ci connaît toutes les applications client s'exécutant sur chaque port et adresse IP. Eureka Server est également connu sous le nom de "Discovery Server".

### Angular 7 :

Angular est un framework de Javascript open source et libre pour le développement des applications "Front end". On a choisi Angular 7 pour notre application en raison de ses multiples avantages, parmi lesquels on peut citer :

- La consistence du code
- La maintenabilité
- La modularité
- La prise d'erreurs précoce

### MySQL Workbench :

MySQL Workbench est une interface utilisateur graphique pour MySQL, fournissant la

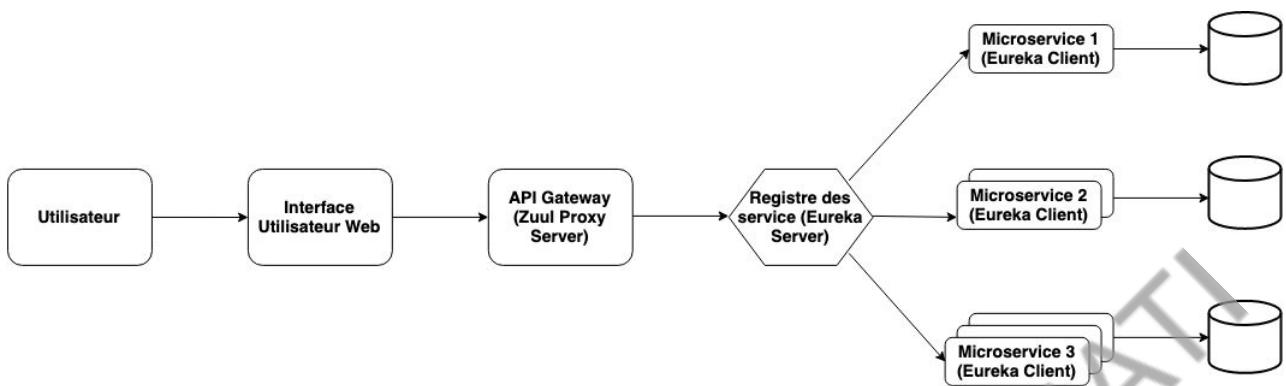


FIGURE 4.1 – Netflix OSS dans l’architecture de microservice

modélisation de données, le développement SQL et des outils d’administration complets pour l’administration des utilisateurs, la configuration du serveur, la sauvegarde, etc.

#### Docker Desktop :

Docker Desktop est la version de la plateforme Docker pour Windows 10.

#### Kitematic :

MySQL Workbench est une interface utilisateur graphique, permettant d’exécuter Docker ainsi que d’afficher et de contrôler vos conteneurs d’applications via l’interface.

#### HTML :

HTML est le langage de balisage standard pour la création de pages Web.

#### CSS :

CSS est un langage qui décrit le style d’un document HTML.

#### JSON :

JSON (JavaScript Object Notation) est un format léger pour stocker et transporter des données.

## 4.3 Démonstration du travail

“Eureka Server” est l’un des composants les plus importants de notre application. Il s’agit d’un service d’enregistrement auquel tous les microservices doivent s’inscrire. La Figure 4.2 représente la création du serveur Eureka, qui peut être visualisée via une interface graphique, comme illustré à la figure 4.3.

Maintenant, les autres microservices doivent être enregistrés dans Eureka également. Pour cela, ils doivent être des “Eureka client”. La figure 4.4 montre la configuration nécessaire pour que, par exemple, “Film-Microservice” devienne un client Eureka et y soit enregistré, comme indiqué à la figure 4.5.

```

package tn.enu.enit.eurekawservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@EnableEurekaServer
@SpringBootApplication
public class EurekaServiceApplication {

    public static void main(String[] args) { SpringApplication.run(EurekaServiceApplication.class, args); }

}

```

EurekaServiceApplication

```

eureka.client.registerWithEureka = false
eureka.client.fetchRegistry = false
server.port = 8761

```

FIGURE 4.2 – Microservice Spring Boot "Eureka Server"

**System Status**

Environment	test
Data center	default
Current time	2019-04-22T18:49:04 +0100
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

**DS Replicas**

localhost
-----------

**Instances currently registered with Eureka**

Application	AMIs	Availability Zones	Status
No instances available			

**General Info**

Name	Value
total-avail-memory	215mb
environment	test
num-of-cpus	4
current-memory-usage	57mb (26%)
server-upptime	00:00
registered-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
unavailable-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
available-replicas	

FIGURE 4.3 – Interface Eureka

### 4.3. DÉMONSTRATION DU TRAVAIL

39

```

1 package tn.nnu.enit.filmsmicroservice;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.context.annotation.Bean;
6 import org.springframework.web.client.RestTemplate;
7 import org.springframework.cloud.netflix.eureka.EnableEurekaClient;
8
9 @EnableEurekaClient
10 @SpringBootApplication
11 public class FilmsMicroserviceApplication {
12
13     public static void main(String[] args) { SpringApplication.run(FilmsMicroserviceApplication.class, args); }
14
15     @Bean
16     public RestTemplate restTemplate() { return new RestTemplate(); }
17
18 }
19
20
21
22 }

application.properties
1 spring.jpa.hibernate.ddl-auto=update
2 spring.datasource.url = jdbc:mysql://localhost:3306/films_microservice?zeroDateTimeBehavior=CONVERT_TO_NULL&serverTimezone=UTC&useSSL=false
3 spring.datasource.username=user
4 spring.datasource.password=user
5 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
6 server.port=8082
7
8 eureka.client.serviceUrl.defaultZone = http://localhost:8761/eureka
9 eureka.client.instance.preferIpAddress = true
10 spring.application.name = films-microservice

```

FIGURE 4.4 – Application Spring Boot "Films Microservice" comme un "Eureka Client"

Name	Value
total-avail-memory	215mb
environment	test
num-of-cpus	4
current-memory-usage	117mb (54%)
server-uptime	00:16
registered-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
unavailable-replicas	<a href="http://localhost:8761/eureka/">http://localhost:8761/eureka/</a>
available-replicas	

FIGURE 4.5 – "Films Microservice" inscrit sur Eureka

Pour pouvoir gérer les microservices, tous ces derniers doivent suivre la même procédure et devenir des clients Eureka, comme illustré à la figure 4.6.

The screenshot shows the Spring Eureka dashboard at [localhost:8761](http://localhost:8761). It displays the following sections:

- System Status:** Shows environment (test), data center (default), current time (2019-04-22T21:50:11 +0100), uptime (01:22), lease expiration enabled (true), renew threshold (11), and renew count (16).
- DS Replicas:** Shows a single instance for localhost.
- Instances currently registered with Eureka:** A table listing four services:
 

Application	AMIs	Availability Zones	Status
CINEMA-MICROSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.52:cinema-microservice:8081
FILMS-MICROSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.52:fils-microservice:8082
UTILISATEUR-MICROSERVICE	n/a (1)	(1)	UP (1) - 192.168.1.52:utilisateur-microservice:8090
ZUUL-GATEWAY	n/a (1)	(1)	UP (1) - 192.168.1.52:zuul-gateway:9091
- General Info:** A table of system metrics:
 

Name	Value
total-avail-memory	232mb
environment	test
num-of-cpus	4
current-memory-usage	168mb (72%)
server-upptime	01:22

FIGURE 4.6 – Microservices inscrits sur Eureka

Cette approche, consistant à exécuter chaque microservice, sur le 'localhost' et dans son propre onglet 'IntelliJ', s'est révélée peu efficace, car elle consomme facilement toute la RAM, ce qui complique tellement l'exécution de l'application (Voir figure 4.7).

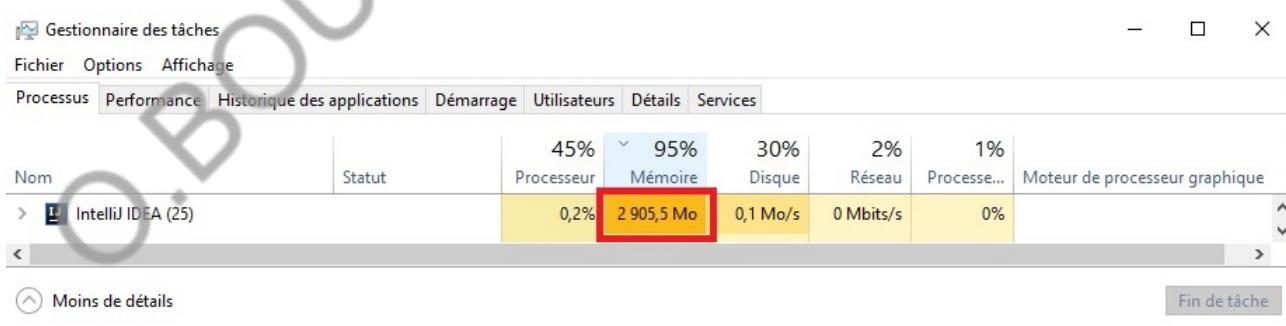


FIGURE 4.7 – Non efficacité de l'utilisation de la RAM par les microservices

Pour remédier à tout cela, nous avons eu recours aux conteneurs, où à partir de chaque microservice, nous avons créé un conteneur docker en configurant un 'Dockerfile', pour en faire des conteneurs Linux. Tous les microservices et même la base de données sont

### 4.3. DÉMONSTRATION DU TRAVAIL

41

maintenant des conteneurs. Ils fonctionneront à l'intérieur de leurs propres conteneurs, comme illustré à la figure 4.8.



FIGURE 4.8 – Base de données MYSQL et les microservices en tant que conteneurs Linux

Maintenant, toute cette logique derrière ces microservices et leurs interactions avec les bases de données est montrée à l'aide des interfaces créées par Angular 7, comme le montre la figure 4.9.

The figure displays three screenshots of a web application interface for a movie theater system named "Cineplex".

- Screenshot 1: Login Screen**  
A modal window titled "Login" is displayed. It contains two input fields: "Email" and "Password", both with placeholder text. Below the fields are two buttons: "Login" (blue) and "Sign up" (grey).
- Screenshot 2: Home Screen**  
The main application screen shows a sidebar with "Add Movie" and a navigation bar with "Frontend", "localhost:4200", "Home", "Films", and "Cinema". The "Films" section lists two movies with their posters:
  - Joker**: A dark green poster featuring a man's face.
  - Avengers: Endgame**: A purple poster featuring the复联 team.Each movie card includes a synopsis, duration, and release date. At the bottom of each card are "update" and "delete" buttons.
- Screenshot 3: Cinema Listings Screen**  
The "Cinema" section of the sidebar is selected. The main area displays a table of cinema locations:

Nom	Adresse	Telephone	Gouvernorat	Action
Cinéma Le Palace Tunis	Avenue Habib-Bourguiba, Tunis	29 847 167	tunis	Add showing Details Update Delete
Cinéma Le Palace Sousse	Avenue Habib Bourguiba	21 906 975	sousse	Add showing Details Update Delete

FIGURE 4.9 – Quelques interfaces pour les conteneurs avec Angular 7

## 4.4 Difficultés rencontrées

### De l'architecture monolithique aux microservices :

La migration d'une architecture monolithique vers une architecture microservice s'est révélée être un réel problème, car nous passons d'une approche de complexité du code à une approche de complexité des interactions, ce qui signifie que, dans une approche monolithique, le code était le seul élément complexe, mais maintenant, nous avons rencontré beaucoup de problèmes pour diviser l'application de manière cohérente en petits composants qui communiquent entre eux. Nous avons donc constaté une certaine redondance du code et de nombreux problèmes de gestion des transactions entre les bases de données de chaque composant.

### Sécurité :

L'architecture de microservices expose de nombreuses failles de sécurité. Nous avons donc dû effectuer beaucoup de configurations supplémentaires, telles que la configuration d'un API Gateway. Netflix Zuul est très populaire et est open-source, nous avons donc choisi de travailler avec.

### Les pratiques de DevOps :

Un autre problème était l'adoption des pratiques de DevOps, notamment en ce qui concerne la manipulation des conteneurs et de fichiers de configuration (YAML) des Docker.

### Authentification et Autorisation :

L'authentification et l'autorisation posent un tout autre problème, où chaque requête entre les microservices doit être authentifiée et autorisée. Cela s'avère plus complexe lorsque nous avons des microservices accédant l'un à l'autre, plusieurs utilisateurs ayant des rôles différents accédant aux microservices également. Le principal problème est que la logique d'autorisation et d'authentification ne doit être implémentée dans aucun service car, dans l'architecture de microservices, chaque service doit traiter un seul "Business Logic" (principe de la responsabilité unique des microservices). Par conséquent, si nous suivons l'approche monolithique et implantons l'authentification et l'autorisation dans chaque microservice, puisque sa logique doit être gérée dans chacun des microservices, nous nous trouverons dans une position où tous les microservices dépendent d'un certain code, ce qui affectera la flexibilité de cette architecture et c'est là que nous avons rencontré beaucoup de problèmes.

## Conclusion

Dans ce dernier chapitre, nous avons discuté les technologies que nous avons adoptées, nous avons aussi présenté l'environnement de travail matériel et logiciel, pour finir par exposer certains scénarios d'exécution à l'aide de quelques captures de notre application.

O.BOUAOUINA & O.LOUATI

# Conclusion Générale

Tout au long de la période de réalisation de ce projet, nous avons passé la majorité du temps à essayer de configurer une architecture microservices, à configurer des conteneurs et à travailler un peu sur l'automatisation du déploiement de ces services.

Dans ce rapport, nous avons procédé à une présentation générale de notre projet, dans laquelle nous avons précisé son objectif et la méthodologie de travail que nous avons suivi. Nous avons ensuite présenté l'analyse et la spécification des besoins ainsi de la détermination des différents intervenants dans notre application. Puis, nous sommes passés à la conception, où nous avons souligné l'architecture globale et détaillée de notre projet pour finir avec un dernier chapitre consacré aux différentes technologies utilisées dans le processus de création de l'application avec quelques captures d'écran montrant le résultat de ce travail.

Ce projet a été pour nous une véritable chance d'acquérir une nouvelle expérience, en particulier avec les nouvelles technologies et le nouvel aspect de la conception d'applications que nous avons suivies, et également de mettre en oeuvre les connaissances acquises au cours des deux années déjà passées à l'ENIT.

Notons que ce projet présente une implémentation assez basique. Parce que faire fonctionner l'ensemble des microservices comme une seule unité s'est avéré être une tâche difficile et qu'il nécessite beaucoup de travail et de configurations bien plus que la création du microservice lui-même. Mais cette application peut être grandement améliorée, de nouveaux microservices peuvent être ajoutés avec de nouvelles fonctionnalités et ce qui émane de la motivation d'une architecture de microservice. L'interface graphique peut être améliorée également pour une meilleure illustration, une des perspectives de notre projet.

# Bibliographie

- [1] Irakli Nadareishvili, Ronnie Mitra, Matt McLarty, Mike Amundsen  
Microservice Architecture: Aligning Principles, Practices, and Culture.  
2016.
- [2] Pethuru Raj, Jeeva S. Chelladhurai, Vinod Singh  
Learning Docker.  
2015.
- [3] Bhagwati Malav : Microservices vs Monolithic architecture.  
<https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4>  
16 Dec 2017
- [4] Sharma, Sourabh  
Mastering microservices with Java 9 : build domain-driven  
microservice-based applications with Spring, Spring Cloud, and Angular.  
2017.