République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université AMO de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique

Mémoire de Master

en Informatique

Spécialité : *Ingénierie des systèmes d'information et logicielles*

Thème

Architecture Monolithique vs Architecture Microservises

Encadré par

— MAHFOUD Zohra

Réalisé par

— CHABANI Nour El-Houda

— SAFIA Thanina

Examiné par

— AMAD Mourad

— CHOUIREF Zahira

— HACINE GHERBI Ahcine

2022/2023

Remerciement

Tout d'abord, nous remercions ALLAH le tout-puissant d'avoir nous donner le courage, la volonté et la patience de mener à terme le présent travail.

On tient à remercier nos très chers parents que nul remerciement, aucun mot ne pourrait exprimer à leur juste valeur la gratitude et l'amour qu'on vous porte. On met vos mains, le fruit de longue de longs mois de distance d'amour de votre tendresse, de longs jours d'apprentissage. Votre éducation, votre soutien, votre encouragement nous ont toujours de la force pour et prospérer dans la vie. Chaque ligne de cette thèse chaque mot et chaque lettre vous la reconnaissance, le respect, l'estime et le merci d' nos parents que Dieu vous garde.

Un grand merci à notre encadreuse, Mme. Mahfoud Zohra, qui nous a apporté un soutien continu, des conseils avisés et s'est toujours montrée disponible pour répondre à nos besoins. Son engagement envers notre réussite a été inestimable. Nos remerciements vont également à Monsieur Badis Lyes pour son aide précieuse tout au long de notre projet. Ses connaissances et son accompagnement ont grandement contribué à la qualité de notre travail.

Enfin, nous tenons à exprimer notre gratitude envers les membres du jury pour nous avoir offert l'opportunité de présenter notre travail. Leurs commentaires constructifs ont été extrêmement utiles pour améliorer notre mémoire de fin d'études.

Dédicace

Nous dédions ce projet :

À nos parents,
Qui ont toujours cru en nous, et qui nous ont soutenus tout au long de notre parcours.

À nos familles et amies,
Qui ont toujours été là pour nous, et qui nous ont fait rire, même dans les moments les plus difficiles.

À nos professeurs,
Qui nous ont transmis leur passion pour l'apprentissage, et qui nous ont aidés à devenir les personnes que nous sommes aujourd'hui.
Et à tous ceux qui ont contribué à la réalisation de ce projet.

Ce projet est le fruit de l'amour, du soutien et de l'amitié qui nous entourent, et nous vous en sommes profondément reconnaissants.

Résumé

Au fil des dernières années, les microservices sont une approche prometteuse pour concevoir des systèmes logiciels modulaires et scalables. Ils offrent une modularité qui facilite la scalabilité, la flexibilité du déploiement et la résilience grâce au principe du couplage faible des services.

Dans le cadre de ce travail, nous avons réalisé une étude comparative entre les architectures monolithiques et les architectures microservices. Les tests ont été effectués sur un réseau social que nous avons développé dans le cadre de ce projet.

Les tests de performance ont mis en évidence des différences entre les deux architectures. À 50 utilisateurs, l'architecture monolithique offre des temps de réponse plus courts, tandis que les microservices se distinguent par 100 utilisateurs en termes de temps de réponse, de débit et de stabilité, mettant en lumière la scalabilité des architectures microservices.

Mots clés : Architecture logicielle, Microservices, réseaux sociaux, monolithique, performance, analyse comparative.

Abstract

In recent years, microservices have become a promising approach to designing modular and scalable software systems. They offer modularity that facilitates scalability, deployment flexibility, and resilience through the principle of loose coupling between services.

As part of this work, we conducted a comparative study of monolithic and microservice architectures. The tests were performed on a social network that we developed as part of this project.

Performance tests have revealed differences between the two architectures. At 50 users, the monolithic architecture offers shorter response times, while microservices stand out at 100 users in terms of response time, throughput, and stability, highlighting the scalability of microservice architectures.

Key words : Software architecture, Microservices, social networks, monolithic, performance, benchmarking.

ملخص

في السنوات الأخيرة، أصبحت هندسة الخدمات المصغرة نهجًا واعدًا لتصميم أنظمة برمجية قابلة للتوسع وقابلة للتكيف. فهي توفر قابلية للتوسع سهلة التنفيذ، ومرونة في النشر، وموثوقية من خلال مبدأ ارتباط الخدمات بشكل ضعيف.

في هذا العمل، قمنا بإجراء دراسة مقارنة بين هندسات الخدمات المتجانسة الموحدة وهندسات الخدمات المصغرة. تم إجراء الاختبارات على شبكة اجتماعية قمنا بتطويرها في إطار هذا المشروع.

أظهرت اختبارات الأداء اختلافات بين المعمارتين. عند ٥٠ مستخدمًا، توفر هندسة الخدمات المتجانسة الموحدة أوقات استجابة أقصر، بينما تتميز هندسة الخدمات المصغرة بـ ١٠٠ مستخدم في مجالات أوقات الاستجابة والمعدل والثبات، مما يسلط الضوء على قابلية توسع هندسات الخدمات المصغرة.

الكلمات المفتاحية : هندسة البرمجيات، الخدمات المصغرة، الشبكات الاجتماعية، الخدمات المتجانسة الموحدة، الأداء، التحليل المقارن.

Table des matières

Table des figures

Liste des tableaux

Liste des abréviations

API	Application Programming Interface
IHM	Interface Homme Machine
UI	User Interface
DB	Database
API	Application Programming Interface
SOA	Service-Oriented Architecture
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
HTTP	Hypertext Transfer Protocol
AMQP	Advanced Message Queuing Protocol
XML	eXtensible Markup Language
JSON	JavaScript Object Notation
ESB	Enterprise Service Bus
MSA	Microservices Architecture
VM	Virtual Machine
IP	Internet Protocol
DNS	Domain Name System
IDE	Integrated Development Environment
RAM	Random Access Memory
CPU	Central Processing Unit
KVM	Kernel-based Virtual Machine
EC2	Elastic Compute Cloud
PC	Personal Computer

LTS	Long-Term Support
DDR4	Double Data Rate 4
SSD	Solid-State Drives
CL14	CAS Latency 14
HD	Hard Drive
gRPC	Google Remote Procedure Call
UP	Unified Process
UML	Unified Modeling Language

Introduction générale

Avec l'expansion constante du monde, de plus en plus de personnes ont désormais accès aux ordinateurs et à l'Internet. Cela impose une exigence d'accessibilité et de fiabilité continue pour les systèmes informatiques. Face à cette demande croissante, de nombreux éditeurs de logiciels se sont engagés dans la recherche de nouvelles architectures capables de résoudre ces défis.

L'architecture logicielle constitue le socle fondamental d'un logiciel, englobant les exigences techniques et opérationnelles. Elle vise à optimiser chaque aspect d'une application, soulignant ainsi l'importance cruciale du choix d'une architecture appropriée dès les premières phases du développement logiciel. Il existe plusieurs types d'architectures logicielles. L'architecture monolithique est le modèle classique d'architectures. Cependant, une problématique se pose lorsqu'une entreprise grandit : l'architecture monolithique peut devenir inefficace et difficile à gérer. C'est pourquoi les architectures microservices ont gagné en popularité ces dernières années grâce à ses plusieurs avantages, à savoir : agilité, scalabilité et facilité de déploiement. Cette architecture consiste à décomposer une application en différents services autonomes et interconnectés, offrant ainsi une solution aux limitations des architectures monolithiques.

Dans ce contexte, ce projet de fin d'études présente les différentes architectures logicielles, en mettant l'accent sur les architectures microservices, en se focalisant sur leurs avantages, leurs limites et les meilleures pratiques pour les implémenter. Nous étudierons également les défis auxquels sont confrontées les organisations lors de l'adoption de ces architectures. Ainsi, cette étude contribuera à mieux comprendre les enjeux et les opportunités des architectures logicielles basées sur les microservices, pour les entreprises et les

développeurs qui souhaitent adopter cette approche innovante et prometteuse.

Dans le deuxième chapitre, nous nous concentrerons sur l'analyse comparative entre l'architecture monolithique et l'architecture microservices, en examinant les articles de recherche qui ont étudié leurs performances, leurs avantages et leurs inconvénients. Il est important de souligner que chaque architecture a ses propres forces et faiblesses, et qu'elles trouvent leur place optimale dans des contextes spécifiques. Nous explorerons donc les situations dans lesquelles l'approche monolithique ou l'approche microservices se révèlent les plus performantes, permettant ainsi de guider les décisions architecturales en fonction des besoins et des contraintes du système à développer.

Le troisième chapitre est consacré à l'étude des réseaux sociaux en général et nous décrivant précisément les fonctionnalités spécifiques de notre application de réseau social dédiée aux tests présentes dans le prochain chapitre. La modélisation de fonctionnalité de notre application est effectuée à l'aide du diagramme de cas d'utilisation du langage UML.

Dans le dernier chapitre, nous présentons le développement et l'évaluation du réseau social que nous avons développée en deux versions : une version monolithique et une version microservices. Nous présentons ainsi les outils utilisés pour son développement et nous exposons également quelques interfaces de l'application. Ensuite, nous décrivant les tests de performance lancés et nous discutons les résultats des tests.

Chapitre 1

Les architectures logicielles et les microservices

1.1 Introduction

Dans ce premier chapitre, nous décrivons plusieurs types d'architectures logicielles, à savoir l'architecture monolithique, l'architecture orientée services et l'architecture des microservices. Nous présentons leurs différentes caractéristiques, ainsi que leurs avantages et inconvénients.

1.2 Architecture monolithique

Le terme monolithique signifie : « qui forme un ensemble rigide, homogène », « constitué d'un seul bloc » [1].

Une architecture monolithique est le modèle unifie traditionnel pour la conception d'un logiciel d'entreprise. Dans une architecture monolithique, toutes les fonctionnalités sont encapsulées dans une seule application, ses modules ne peuvent donc pas être exécutés indépendamment. Les applications monolithiques ont été conçues pour traiter de multiples tâches connexes. Il s'agit généralement d'applications complexes qui englobent plusieurs fonctions étroitement couplées [2].

Une application monolithique est souvent associée à une base de données (DB), ainsi qu'à une interface utilisateur côté client (UI) et une application côté serveur. Comme le

montre la figure 1.1 ci-dessous .

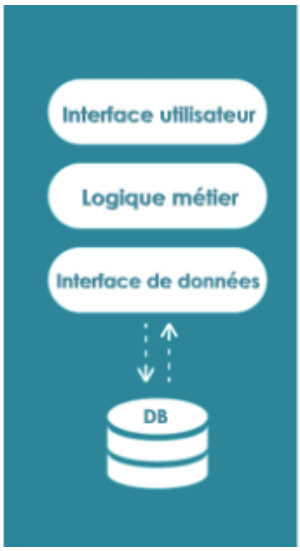

FIGURE 1.1 – Architecture monolithique.

La figure 1.2 ci-dessous présente un exemple d'application monolithique qui fournit une logique métier pour un site de commerce électronique, accessible via diverses API (desktop ou mobile ...).

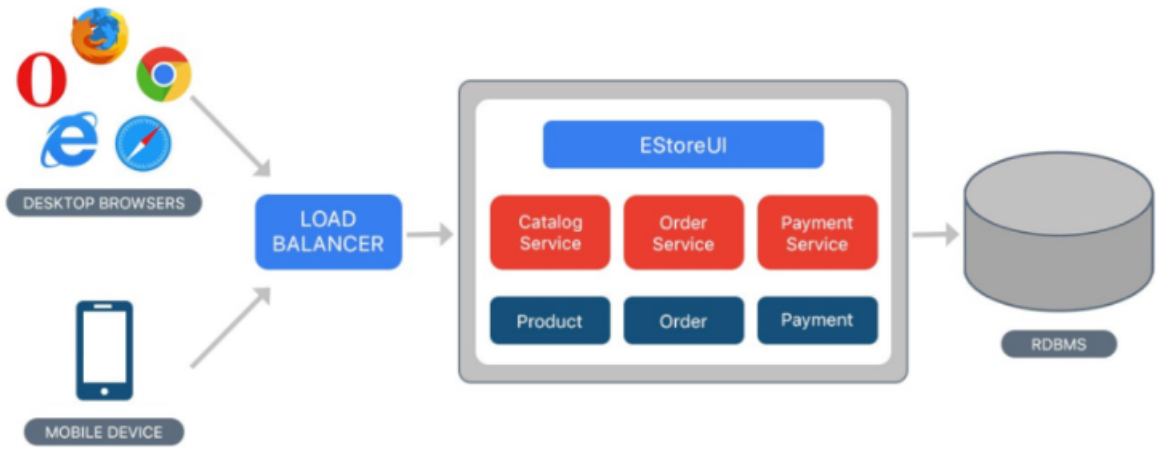

FIGURE 1.2 – Exemple d'une application monolithique de e-commerce.

1.2.1 Avantages d'architecture monolithique

Au début des architectures monolithiques, lorsque l'application était relativement petite, cette architecture présentait de nombreux avantages : ([2] [3])

— Simple à développer : les IDE et autres outils de développement se concentrent sur la création d'une seule application construit d'un seul bloc de code, elle est plus facile à développer.

— Facile à tester. Par exemple, en implémentant des tests de bout en lançant simplement l'application et en testant l'interface utilisateur à l'aide des outils appropriés.

— Facilité de déploiement : Un seul fichier exécutable ou répertoire simplifie le déploiement.

— Simple à mettre à l'échelle : En pouvant faire évoluer l'application en exécutant plusieurs instances de l'application derrière un équilibreur de charge.

— C'est une bonne idée de commencer un projet avec ce type d'architecture car cela permet d'explorer à la fois la complexité du système et les limites de ses composants.

1.2.2 Les limites de l'architecture monolithique

Une fois que l'application deviendra grande et que l'équipe grandit, cette architecture peut présenter des problèmes qui ont de plus en plus importants :
— La correction des bugs finit éventuellement par entraîner de nouveaux bugs.
— La taille de l'application peut ralentir le temps de démarrage.
([2], [4], [5]).

Limites	Impact sur la plateforme
Limites technologiques.	◦ Le choix des technologies est déterminé avant le début du développement de l'application. ◦ Une seule technologie est utilisée pour le développement de l'application.
Mise à l'échelle couteuse.	◦ La mise à l'échelle d'une partie qui a besoin de plus de ressources requiert toute l'application.

Modification couteuse.	◦ Une modification dans une petite partie de l'application requiert une reconstruction et un redéploiement entier.
Tolérance aux pannes limitée.	◦ Un dysfonctionnement sur une partie du système impacte tout le système.
Agilité limitée.	◦ Agilité réduite de l'équipe et fréquence de livraison limitée à cause du couplage fort entre les composants. ◦ Effort important de montée en compétences pour un nouvel arrivant sur des composants fortement couplés. ◦ Nécessité de l'intervention de plusieurs personnes pour chaque modification.
Coût des tests élevé.	◦ Durée des tests automatiques longue.
Problème de fiabilité.	◦ Un bogue dans n'importe quel module peut potentiellement faire échouer l'ensemble du processus.
Maintenance complexe.	◦ L'application est trop volumineuse et complexe donc la maintenance deviendra très compliquée.

TABLE 1.1 – Limites des architectures monolithiques.

1.3 Architectures services

Pour clarifier l'architecture orientée services ou l'architecture de microservices, nous devons définir le terme "service".

Un service est une fonction bien définie et indépendante des autres services. Les services sont des composants qui sont liés entre eux pour former des applications orientées services ou microservices.

1.3.1 Architectures Orientées Services

Les architectures orientés services (SOA) sont développées pour surmonter les inconvénients de l'architecture monolithique. Ce type d'architecture est considéré comme un modèle d'interaction distribuée entre les applications.

L'architecture orientée services consiste à décomposer l'application en plusieurs composants ou modules appelés "services", qui sont faiblement couplés entre eux. Ces services sont des programmes autonomes pouvant être développés avec n'importe quelle technologie, sans tenir compte des technologies utilisées par les autres services.

FIGURE 1.3 – L'architecture orientée services.

Les services développés peuvent communiquer entre eux via des API telles que REST, SOAP, GraphQL, etc. L'API des services est généralement configurée avec HTTP, mais il est également possible d'utiliser d'autres systèmes pour transférer des messages tels que AMQP, RabbitMQ, etc. Cette architecture utilise des formats d'échange par exemple XML ou JSON et une couche d'interface interopérable appelée les services web [6].

Le bus de service d'entreprise (ESB) fournit une API qui peut être utilisée pour développer des services et garantir que ces services puissent interagir de manière fiable entre eux. Techniquement, l'ESB est une infrastructure de messagerie qui effectue la conversion de protocole, la transformation de format de message, le routage, l'acceptation et la livraison de messages provenant de divers services et applications connectés à l'ESB [7].

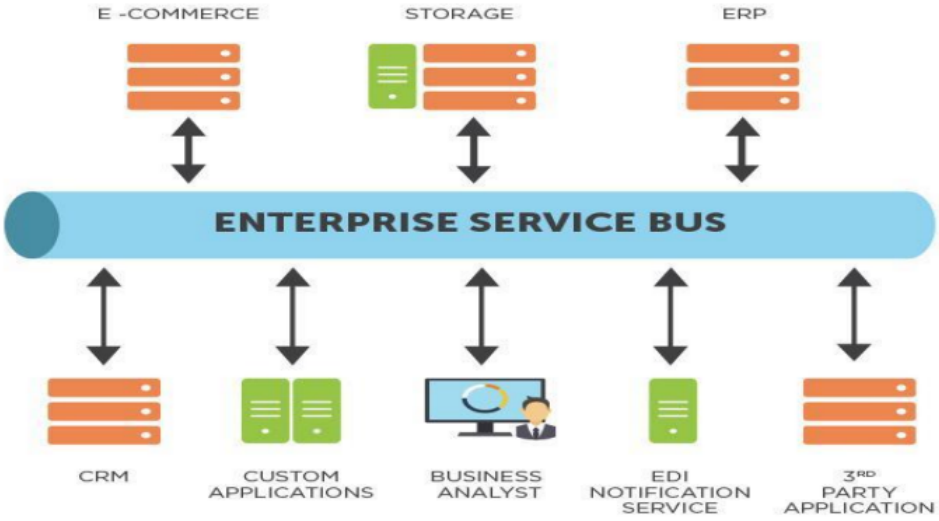

FIGURE 1.4 – L' ESB.

1.3.2 Architecture microservice

Afin d'éviter les problèmes associés aux applications monolithiques et de tirer parti de certains avantages de l'architecture SOA, le modèle d'architecture Microservices est apparu comme un sous-ensemble léger de l'architecture SOA. Ce modèle est basé sur une philosophie de développement agile et s'appuie sur des années d'expérience en génie logiciel et en conception de systèmes [8]. Il est utilisé par des entreprises telles qu'Amazon, Netflix, LinkedIn et Uber pour prendre en charge et mettre à l'échelle leurs applications et produits. L'idée est de diviser l'application en un ensemble de petits services interconnectés plutôt que de créer une seule application monolithique.

L'image 1.5 suivante montre la migration de l'architecture monolithique vers une architecture microservices. [9]

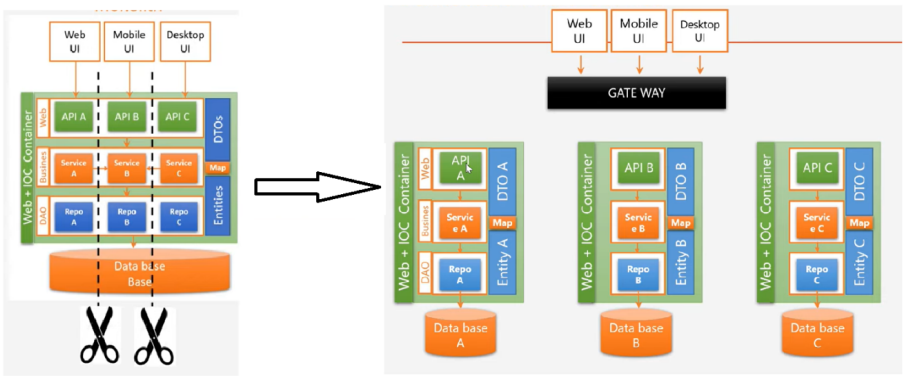

FIGURE 1.5 – Migration d'une architecture monolithique vers microservice.

L'architecture microservices (MSA) représente l'approche consistant à développer une application unique sous la forme d'un ensemble des petits services faiblement couplés.

La différence majeure entre SOA et les microservices réside dans la façon dont les services sont conçus et organisés. SOA encourage souvent la réutilisation des services à travers différentes applications, tandis que les microservices se concentrent davantage sur la conception de services spécifiques à une application. De plus, les microservices ont tendance à être plus petits et plus spécialisés que les services SOA [2].En résumé, l'architecture orientée services est une approche plus large pour la conception de systèmes distribués, tandis que l'architecture microservices est une approche plus spécifique et axée sur une application.

Chaque service à sa propre logique métier et sa propre base de données. Les différentes opérations de mises à jour, tests, déploiement et de mise à l'échelle concerne un seul service [10].

La figure 1.6 ci-dessous illustre un exemple d'application utilisant l'architecture microservices, fournissant une logique métier de e-commerce similaire à celle de la version monolithique précédente.

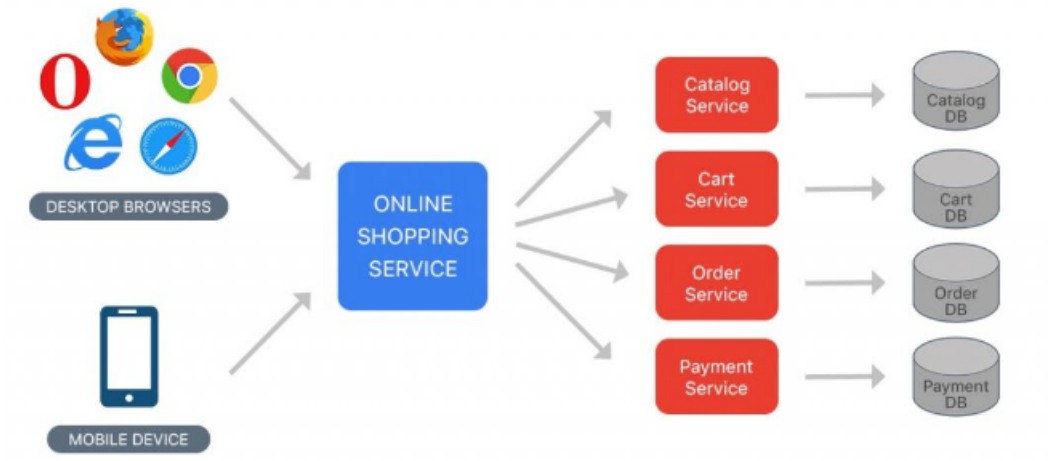

FIGURE 1.6 – Architecture de microservices (exemple pour une application de e-commerce).

Une architecture microservices est composée de (Voire la figure 1.9) :

A. Api gateway

Une MSA permet de répondre à différents types de clients via divers interfaces à travers des navigateurs Web en utilisant différents appareils intelligents tels que les laptots, smartphones, tabltes, etc. [11].

L'API Gateway est un service qui résout la problématique d'avoir des clients de natures différentes, elle est le seul point d'accès pour un microservice qui donne accès à de nombreuses APIs.

Elle permet de : publier plusieurs API, chacune dédiée à un groupe de clients différents. Cette passerelle possède d'autres fonctionnalités telles que la vérification des autorisations, l'équilibrage de charge, la mise en cache et la surveillance, l'authentification, la transformation de données et la transformation de protocole [11] [12] [13].

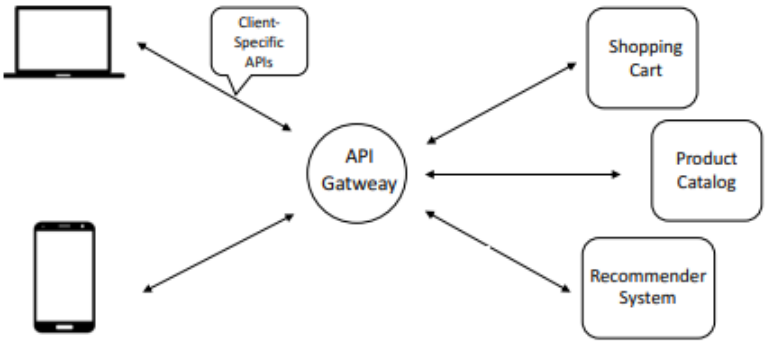

FIGURE 1.7 – Api gateway.

Le point d'entrée d'une architecture de microservices est généralement une API gateway, qui permet de fournir un accès centralisé et sécurisé aux différents services de l'application. En plus de la gateway, d'autres services tels que le serveur de configuration, le service de découverte ou encore l'équilibrage de charge peuvent être intégrés pour assurer le bon fonctionnement de l'ensemble du système.

B. Un serveur de configuration

C'est un serveur central qui fournit la configuration de chaque microservice. Ce serveur connecte au service de découverte pour récupérer les fichiers de configuration dans un dépôt (des référentiels) accessibles aux microservices.

Il permet de changer la configuration de l'application sans redéployer le code. Comme une architecture de microservices comporte de nombreux services et que leur redéploiement va être coûteux [14].

C. Service de découverte

En pratique, l'emplacement d'un microservice peut ne pas être connu de manière statique au moment de la conception. En effet, les microservices peuvent être déployés dans un système basé sur le cloud où les services peuvent être répliqués et déplacés pendant l'exécution, et généralement plusieurs instances du même microservice s'exécutent dans différents conteneurs/VM virtualisés. Par conséquent, un participant à un MSA a besoin d'employer un mécanisme de découverte de service. Ceci est généralement réalisé avec la même idée adoptée pour les SOA, c'est-à-dire en utilisant un services registery.

Le Service Registry est un composant qui permet aux autres services de récupérer des informations de connexion sur les différents services d'une application (comme leur nom, adresse IP et port DNS). Les microservices communiquent avec le registre pour publier leur emplacement (ou celui des autres services). Ainsi, la communication entre les microservices doit être définie dynamiquement pour que les clients puissent communiquer efficacement avec les microservices correspondants, en basculant automatiquement entre les différentes instances disponibles.Il est intéressant de faire la distinction entre les modèles côté client et côté serveur [13] [15].

D. Le modèle de découverte côté client

Ici, les clients interrogent le service registery, sélectionnent une instance, et lancent une demande directement [15].

E. Le modèle de découverte côté serveur

Dans ce modèle, un client fait une demande via un équilibreur de charge qui interroge le registre et transmet la demande à une instance disponible [15].

F. Équilibreur de charge

Pour être scalable, une application doit être capable de répartir la charge sur un service individuel parmi ses nombreuses instances. C'est le devoir d'un équilibreur de charge, et dans ce cas, il devrait obtenir les instances disponibles à partir du composant Service Discovery [14].

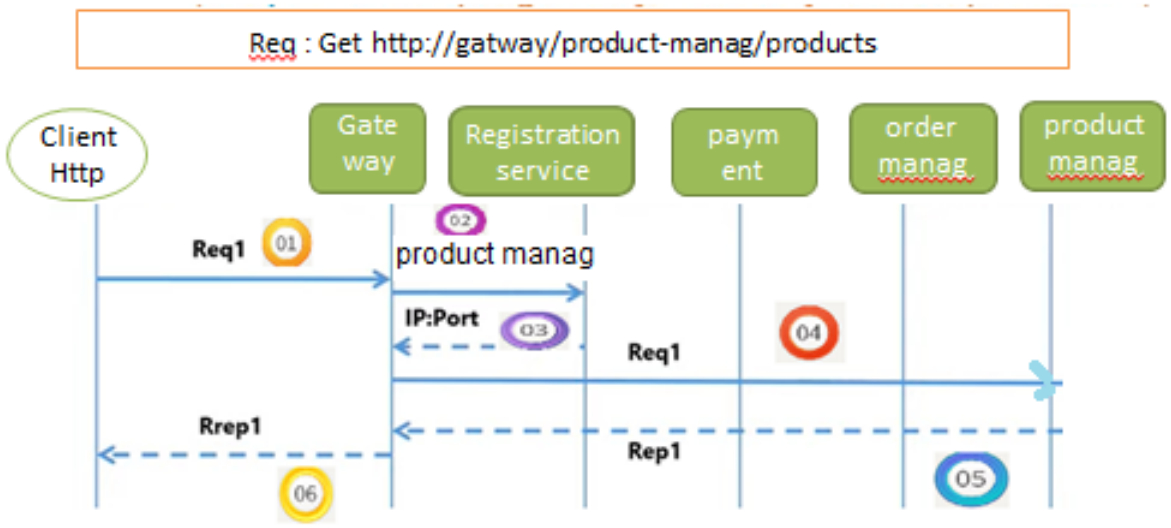

FIGURE 1.8 – Résumé de comment sa marche l'architecture microservice.

Lorsqu'un utilisateur souhaite accéder à un service spécifique, sa requête est acheminée via la passerelle API. La passerelle communique ensuite avec registery service pour obtenir l'emplacement dynamique de chaque instance de microservice. Registery service contacte le microservice concerné pour vérifier sa disponibilité et transmet son emplacement (souvent l'adresse IP ou le nom de domaine et le port) à la passerelle. Ensuite, la passerelle envoie la requête au microservice approprié, qui renvoie une réponse [15].
[9]

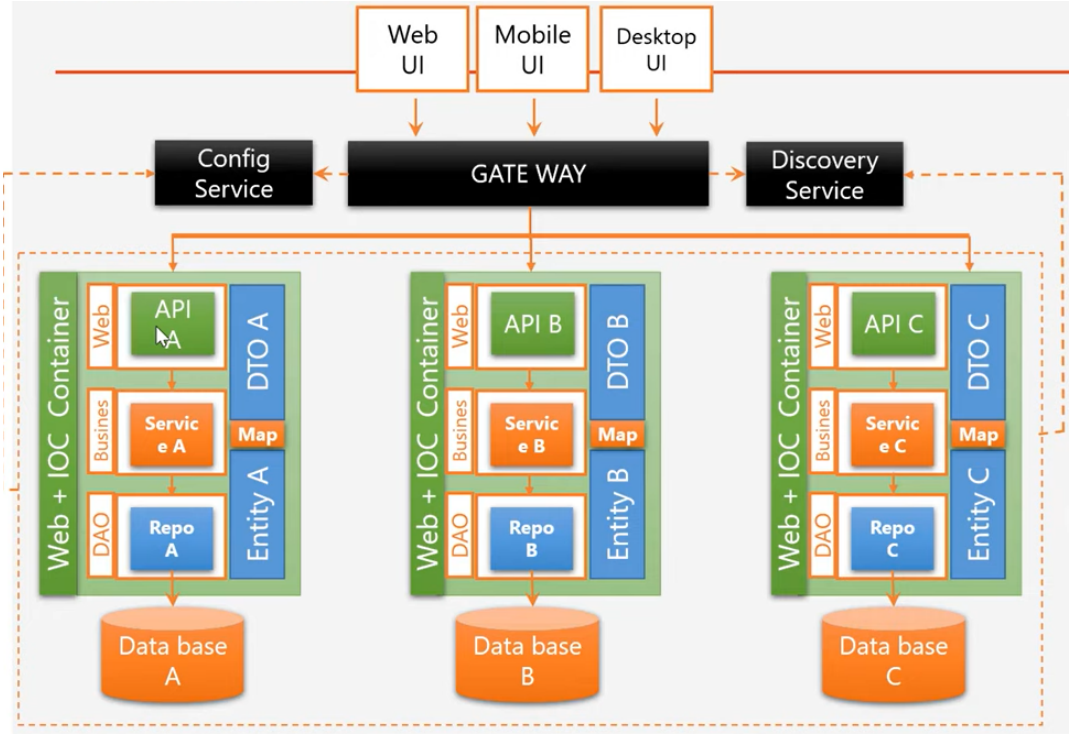

FIGURE 1.9 – Architecture microservices et ses composant.

1.3.3 Avantage de l'architecture microservice

L'architecture des microservices est devenue extrêmement populaire en raison de divers avantages, tels que :

—**Déploiement indépendant :** Les microservices étant des unités individuelles, ils permettent de déployer indépendamment des fonctionnalités individuelles de manière rapide et facile, et ouvre de nouveaux modèles pour améliorer l'échelle et la robustesse des systèmes.

—**Meilleure maintenabilité :** Chaque service est relativement petit, ce qui permet de se concentrer sur une seule partie, facilitant ainsi la compréhension et la modification.

—**Amélioration de l'isolation des pannes :** Les services peuvent s'exécuter en parallèle. Ainsi, si un service perd de la mémoire, seul ce service est affecté, ce qui rend plus efficace la poursuite du traitement des requêtes sans interrompre les autres services. En pouvant demander à plusieurs développeurs de résoudre le problème [16].

—**L'isolation des processus :** peut également modifier les choix technologiques.

Mélanger et associer différents langages de programmation, styles de programmation, plates-formes de déploiement ou bases de données pour trouver la bonne combinaison [16].

—Dans les grandes entreprises, différents services peuvent appartenir à différentes équipes. Ces services peuvent être réutilisés dans toute l'entreprise [17].

—**Meilleure testabilité :** Les services sont plus petits et plus rapides pour tester de nouvelles fonctionnalités et revenir en arrière en cas de problème. Cela facilite la mise à jour du code et accélère la commercialisation des nouvelles fonctionnalités.

—**Les temps de démarrage** et un IDE plus rapides rendent les développeurs plus productifs et plus satisfaits, et accélère les déploiements.

—**Temps de développement réduit :** Le développement distribué permet d'éditer et de développer plusieurs microservices simultanément. Cela accélère le développement d'applications car plusieurs développeurs peuvent intervenir sur le projet en même temps sans se déranger et nuire au travail de l'autre [17].

Caractéristiques des microservices

A. Autonomie

Les microservices sont autonomes et peuvent être utilisés dans divers contextes, ce qui les rend plus portables. En outre, ils peuvent être implémentés dans des langages de programmation différents et utiliser des technologies différentes, ce qui permet d'avoir des composants hétérogènes. Les services n'ont pas besoin de partager leur code ou leur implémentation avec d'autres services, car les composants individuels communiquent via des API clairement définies [18].

B. Spécialisation

Chaque service est conçu pour un ensemble de fonctions et axé sur la résolution d'un problème spécifique. Au fur et à mesure que les développeurs contribuent plus de code aux services au fil du temps, et que les services deviennent plus complexes, ils peuvent se fragmenter en services plus petits [18].

C. Conçue pour les entreprises

Les architectures de microservices sont généralement structurées autour d'objectifs et de capacités métier. Une architecture de microservices s'appuie sur des groupes interfonctionnels où différentes équipes de développement ont des objectifs spécifiques, par opposition aux stratégies de croissance monolithiques traditionnelles. Chaque groupe fabrique des produits particuliers basés sur ses propres services qui communiquent via un bus de messagerie.

D. Résistant aux pannes

Étant donné qu'un grand nombre de services différents interagissent entre eux, il est possible qu'un service tombe en panne à un moment donné. Dans de tels cas, il est important que l'utilisateur puisse quitter le système sans perturbation tout en permettant aux services voisins de continuer à fonctionner normalement. L'architecture microservices est conçue pour gérer les défaillances, en limitant le risque de dysfonctionnement et en fournissant des mécanismes de récupération en cas de défaillance.

E. Distribution des microservices

La distribution des microservices nécessite que chaque service soit un processus système autonome, qui peut être isolé sur une machine dédiée ou déployé dans un conteneur. Les microservices communiquent entre eux via des protocoles et services web, que ce soit avec ou sans modèle d'acteur. Cette isolation permet d'assurer l'indépendance et la flexibilité de chaque microservice, ainsi qu'une meilleure résilience en cas de panne.

Les objectifs de l'architecture microservices

–Maximiser l'autonomie des différentes équipes

Chaque équipe peut se concentrer sur son propre microservice et le développer de manière autonome sans avoir à se soucier des autres éléments du système. Cela peut

également accélérer le développement en permettant aux équipes de travailler simultanément sur différentes parties du système sans être bloquées les unes par les autres.

—**Fournir de la flexibilité sans compromettre la cohérence**

Il faut donner aux équipes la liberté, dont elles ont besoin pour développer leur partie, mais il faut suivre un ensemble de règles standard auxquelles tout le monde peut se référer. En plus des avantages de développement, il facilite également les tests (tests automatisés ou unitaires) des parties de code, c'est-à-dire les modifications et les adaptations.

—**Isoler les problèmes**

L'isolation des microservices permet de limiter l'impact des problèmes et de faciliter l'identification des causes, grâce à la séparation des données et de l'exploitation. En effet, due à la nature particulière des microservices, en théorie l'identification des problèmes devrait être plus facile puisqu'on peut identifier et pointer le service responsable.

1.3.4 Inconvénients des microservices

Certes, aucune technologie n'est une solution miracle, et l'architecture des microservices présente un certain nombre d'inconvénients et de problèmes :

—Déterminer le bon ensemble de services est un défi car il est essentiel de trouver un équilibre entre le nombre de microservices et leur taille. Si les microservices sont trop nombreux et trop petits, cela peut entraîner une surcharge de communication et des problèmes de latence.

—**Coûts d'infrastructure exponentiels :** chaque nouveau microservice peut impliquer ses propres coûts pour la suite de tests, l'infrastructure d'hébergement, les outils de surveillance, et bien plus encore. Cela peut rendre la maintenance et la mise à l'échelle du système très coûteuses.

—**Complexité du déploiement :** En production, il y a également la complexité opérationnelle du déploiement et de la gestion d'un système composé de nombreux services différents.

—**Développement tentaculaire :** par rapport à une architecture monolithique, les microservices ajoutent de la complexité, car plusieurs équipes créent plusieurs services dans plusieurs emplacements. Si elle n'est pas bien gérée, cette multiplication peut ralentir le développement et dégrader les performances opérationnelles.

—**Frais organisationnels supplémentaires :** Les équipes doivent ajouter un autre niveau de communication et de collaboration pour coordonner les mises à jour et les interfaces. Cela peut entraîner des coûts organisationnels supplémentaires et ralentir le développement.

1.3.5 Les défis des microservices

A. Risques de sécurité

Le grand nombre d'API, de ports et de composants exposés peut rendre les déploiements vulnérables à diverses cybermenaces. Les pare-feu traditionnels peuvent ne pas être suffisants pour fournir une sécurité adéquate [8].

B. La performance

La performance est définie comme un attribut de la qualité du logiciel qui inclut le comportement du système logiciel. Cependant, l'architecture microservices requiert une interaction continue entre les services qui composent l'application. Chaque service doit utiliser l'interface de communication entre les services pour effectuer des opérations transactionnelles, ce qui peut affecter négativement les performances en raison de la communication à travers le réseau.

C. Débogage

Les microservices sont des systèmes distribués simultanés qui sont complexes et dynamiques. Pour déboguer ces systèmes, le moyen le plus efficace est de suivre et de visualiser leur exécution. Toutefois, cela peut être plus difficile que pour les systèmes distribués concurrents traditionnels en raison de leur complexité accrue [8].

D. La cohérence des données

La flexibilité et la scalabilité offertes par les systèmes de microservices comportent un défi majeur : la cohérence des données. Dans un système monolithique, les données sont stockées de manière centralisée dans une seule base de données, tandis que dans une architecture de microservices, elles sont réparties sur différentes bases de données de microservices. Il est donc difficile d'assurer la cohérence transactionnelle entre plusieurs bases de données. C'est un problème majeur qui doit être résolu pour garantir l'intégrité des données dans le système [19].

E. La complexité des tests

La complexité des tests peut être un défi dans l'architecture de microservices. Les microservices doivent être testés ensemble, ce qui peut entraîner des blocages de test si un microservice échoue. De plus, le nombre élevé de services et d'interfaces nécessite des tests indépendants pour chaque côté de l'interface, ce qui peut être difficile à gérer.

F. Les microservices peuvent être coûteux

Les appels réseau effectués par les API ne sont pas gratuits et les coûts peuvent s'élever à des coûts énormes. De plus, le coût de l'effort du développeur impliqué dans la rupture d'un monolithe peut créer une dépense autrement inutile.

Meilleures pratiques pour la mise en œuvre d'une architecture de microservices.

La mise en place d'une architecture de microservices peut être complexe, mais en suivant certaines meilleures pratiques, il est possible de surmonter certains défis associés. Voici quelques recommandations :

— Évaluer l'adaptation de l'architecture microservices au projet avant de l'implémenter. Avant d'utiliser l'architecture microservices, il sera préférable de décider si l'architecture conviendra au projet ou non.

— Concevoir des services faiblement couplés pour favoriser la flexibilité.

— Utiliser des API et des événements pour permettre une communication efficace entre les services.
— Utiliser des machines virtuelles pour fournir un environnement de développement cohérent sur tous les systèmes peut aider les développeurs à éviter tout problème inutile pouvant survenir en raison de la variation des performances au sein de différents systèmes.
— Utiliser une base de données distincte pour chaque microservice pour personnaliser les exigences de stockage.
— Isoler chaque microservice pour faciliter le déploiement et la coordination.
— Utiliser des conteneurs pour une gestion plus efficace des ressources et une plus grande flexibilité.
— Mettre en place un système de journalisation et de surveillance centralisé pour faciliter la gestion des erreurs et l'analyse des causes profondes.

Comment démarrer avec une architecture de microservices dans une entreprise ?

L'architecture microservices offre plusieurs avantages et est un choix judicieux pour de nombreux projets. Mais par où commencer ? Une approche courante consiste à commencer par un système monolithique et à passer progressivement à une architecture de microservices en découpant le système en fonctionnalités distinctes. Il est important d'avoir une structure de conception fonctionnelle pour les microservices dès le départ. De plus, il est essentiel de déployer et d'héberger les différents composants de manière indépendante, d'adopter des options de gestion des données spécifiques aux services et de choisir la meilleure technologie disponible. Enfin, il est recommandé de centraliser les opérations pour une meilleure gestion et une coordination efficace entre les différentes équipes.

Comparaison entre les architectures microservices et monolithique

Caractéristiques	Architecture des microservices	Architecture monolithique
Conception de l'unité	L'application consiste en des services faiblement couplés, chacun étant dédié à une	l'ensemble de l'application est développé, conçu et déployé en tant qu'une

	tâche métier spécifique.	seule entité.
Réutilisation de la fonctionnalité	Les microservices définissent des API qui exposent leurs fonctionnalités à n'importe quel client. Les clients pourraient même être d'autres applications.	La possibilité de réutiliser les fonctionnalités entre les applications est limitée.
Communication au sein de l'application	Pour communiquer entre eux, les microservices d'une application utilisent le modèle de communication requête-réponse. L'implémentation typique utilise des appels API REST basés sur le protocole HTTP.	Les procédures internes(appels de fonctions)facilitent la communication entre les composants de l'application. Il n'est pas nécessaire de limiter le nombre d'appels de procédure interne.
Flexibilité technologique	Chaque microservice peut être développé à l'aide d'un langage et d'un cadre de programmation qui conviennent le mieux au problème que le microservice est conçu pour résoudre.	Habituellement, l'ensemble de l'application est développé en utilisant un seul langage de programmation.
Gestion des données	Décentralisé : chaque microservice peut utiliser sa propre base de données. Base de données séparée.	Centralisé : toute l'application utilise une ou plusieurs bases de données. Base de données partagée.
Déploiement	Chaque microservice est déployé indépendamment, sans affecter les autres microservices de l'application. Peut être fait rapidement.	Toute modification apportée à l'application, même petite, nécessite le redéploiement et le redémarrage de l'ensemble de l'application. Prend plus de temps.
Maintainability	Les microservices sont simples, ciblés et indépendants. L'application est donc plus facile	À mesure que la portée de l'application augmente, le maintiendu code devient

	à entretenir.	plus complexe.
Résilience	La fonctionnalité de l'application est répartie entre plusieurs services. Si un microservice échoue, la fonctionnalité offerte par les autres microservices continue d'être disponible.	Une défaillance dans n'importe quel composant pourrait affecter la disponibilité de l'ensemble de l'application.
Scalabilité	Chaque microservice peut être échelonné indépendamment des autres services. Faciles à apporter modifications.	Toute l'application doit être mise à l'échelle, même si l'exigence commerciale ne concerne que certaines parties de l'application. Difficile de faire des changements.

TABLE 1.2 – Comparaison entre l'architecture monolithique et microservice.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les différentes architectures logicielles disponibles, en mettant l'accent sur les inconvénients des architectures monolithiques traditionnelles et la solution qui est offerte par les microservices. Nous avons mis en évidence les avantages significatifs des microservices, tout en notant qu'ils ont également des inconvénients à prendre en compte.

Chapitre 2

Comparaison entre Monolithique et Microservices

2.1 Introduction

Plusieurs études de recherches ont été consacrés à l'étude des différents aspects des performances des microservices, notamment en comparant l'efficacité des applications microservices et les applications monolithiques.

Dans ce chapitre, nous définissons d'abord les principaux critères de comparaison, ensuite à partir de certains articles, nous présentons une étude des travaux liés à la comparaison entre l'architecture microservice et l'architecture monolithe. Enfin, nous terminerons par les résultats comparatifs des articles des deux architectures.

2.2 Critères de comparaison

Afin de comparer les deux architectures monolithiques et microservices, deux versions de l'application sont développées une avec l'architecture monolithique et une autre avec l'architecture microservices. Des tests de performance sont lances et plusieurs paramètres peuvent être considérés pour comparer les résultats, tels que : ([20] [21] [22])

- **Le temps de réponse :** est le temps qu'il faut à un client pour recevoir une réponse du serveur pour une demande d'un service spécifique, c'est-à-dire le temps entre l'envoi d'une requête et la réception de la réponse correspondante .

- **Débit :** est le nombre de requêtes qu'une application peut traiter par seconde.
- **Taux d'erreur :** c'est la différence entre les valeurs approximatives et les valeurs exactes, exprimée en pourcentage de la valeur exacte. La formule du pourcentage d'erreur est utilisée pour calculer ce taux.
- **Utilisation de la mémoire(RAM) :** L'utilisation de la RAM est une mesure qui représente le pourcentage de mémoire vive actuellement utilisée par un système informatique pour exécuter des programmes et stocker des données en temps réel.
- **Utilisation du processeur (CPU) :** est une mesure de performance qui représente le pourcentage de temps pendant lequel le processeur est occupé à exécuter des tâches, par rapport au temps total disponible.
- **La vitesse d'écriture sur disque :** est une mesure de la performance d'un système de stockage qui indique la quantité de données qu'un disque peut écrire en une seconde. Cette mesure est généralement exprimée en mégaoctets par seconde (Mo/s).
- **La vitesse de lecture sur disque** est une mesure de la vitesse à laquelle un disque dur ou un autre périphérique de stockage peut lire des données à partir du disque.

  Les performances du disque sont utilisées pour évaluer l'efficacité des opérations de lecture et d'écriture séquentielles sur le disque.
- **La réception et la transmission du réseau :** mesurent le transfert de données.

2.3 Travaux connexes

"Performance characteristics between monolithic and microservice-based systems"

R.Flygare et A. Holmqvist [23] ont utilisés quatre architectures : une architecture microservice sur une machine puis sur deux machines, une architecture monolithique fonctionnant à l'intérieur d'un conteneur sur une machine et une autre fonctionnant directement sur une seule machine.
Les tests sont lancés avec 1 seul utilisateur puis 10, 100 et 1000 utilisateurs simultanément. Ils ont comparé la latence, le débit réussi et mesurent l'utilisation de la RAM et du processeur pour les applications monolithiques et microservices. Trois tests sont lancés : archi-

tecture monolithique versus architecture microservice (RQ1), monolithique conteneurisé versus baremetal monolithique (RQ2), microservice sur une machine versus un cluster d'ordinateurs (RQ3).
Les résultats des tests montrent que :

— Le système monolithique a des temps de réponse plus courts et un débit plus élevé, tout en utilisant moins de RAM et de CPU qu'un système de microservices.
— La conteneurisation du système avec Docker améliorait les performances en termes de RAM, de CPU, de débit et de latence.
— Lors de l'utilisation des clusters dans une architecture microservice, la latence reste la même et d'autre part le débit augmente, mais consomme plus de ressources (CPU, RAM) par rapport à l'utilisation d'un seul ordinateur.

"The comparison of microservice and monolithic architecture"

Dans l'article "The comparison of microservice and monolithic architecture " [24], Konrad Gos et Wojciech Zabierowski ont développé deux applications de gestion d'une agence de location des voitures.

Le test inclut des requêtes HTTP GET et POST pour les deux architectures. L'architecture microservice a été testée en utilisant trois variantes - sans réplication, deux réplications et quatre réplications.

Les auteurs ont comparé les temps de réponse des applications monolithiques et microservices. Les résultats des tests montrent que l'architecture de microservice est plus efficace en cas de grande quantité de requêtes.

"From Monolithic Systems to Microservices : A Comparative Study of Performance"

Freddy Tapia et al. Dans l'article "From Monolithic Systems to Microservices : A Comparative Study of Performance" [22] ont fait une analyse approfondie des microservices et de l'application monolithique. Cette dernière était hébergée sur des machines virtuelles KVM, tandis que les microservices sur le cluster EC2. Des mesures de performances telles

que la consommation du processeur, la vitesse de lecture et d'écriture du disque, l'utilisation de la mémoire, ainsi que la réception et la transmission de données sur le réseau sont évaluées.

Selon l'étude, l'architecture des microservices est plus performante en termes d'utilisation des ressources matérielles (disque, mémoire), de réduction des coûts et de productivité accrue. Cependant, elle entraîne une consommation de CPU plus élevée par rapport à l'architecture monolithique.

"Performance analysis of Monolithic and Microservice architectures – Containers technology"

Une étude similaire a été réalisée dans l'article " Performance analysis of Monolithic and Microservice architectures – Containers technology". Alexis Saransig et Freddy Tapia montrent que la consommation de mémoire, de processeur et de réseau est plus élevée lors de l'utilisation de microservices par rapport à une architecture monolithique [25].

"A Comparative Review of Microservices and Monolithic Architectures"

L'article "A Comparative Review of Microservices and Monolithic Architectures" [20] de Omar Al-Debagy et Peter Martinek est une revue comparative des architectures microservices et monolithiques. Les auteurs comparent les architectures monolithiques et basées sur les microservices, en passant en revue leurs avantages, leurs inconvénients, leur performance, leur maintenance, leur déploiement, leur gestion de données, leur sécurité et leurs coûts.

Selon l'étude, les architectures basées sur les microservices offrent plusieurs avantages par rapport aux architectures monolithiques, notamment en termes de flexibilité et de scalabilité, mais nécessite une planification et une conception minutieuses, ainsi qu'une mise en œuvre et une gestion rigoureuses.

"Monolithic vs. Microservice Architecture : A Performance and Scalability Evaluation"

L'article "Monolithic vs. Microservice Architecture : A Performance and Scalability Evaluation" [26] de Grzegorz Blinowski , ANNA OJDOWSKA, et ADAM PRZYBYŁEK

compare les performances et la scalabilité des architectures monolithiques et des architectures des microservices. Les auteurs ont créé un système de vente en ligne en utilisant les deux approches architecturales et ont effectué des tests de charge pour mesurer la vitesse de traitement des requêtes et la consommation de ressources. Les résultats ont montré que l'architecture microservices était supérieure en termes de vitesse et de capacité de mise à l'échelle, mais qu'elle nécessitait plus de temps et de compétences pour sa mise en place et sa maintenance que l'architecture monolithique.

2.4 Comparaison des articles sur l'architecture des microservices et l'architecture monolithique

Le tableau 2.1 compare les différents articles cités dans le paragraphe précédent :

Propositions	Contexte des expériences			Résultats des tests
	Application Développé	Caractéristique de matériel utilise	Les tests	
[23]	/	Le matériel utiliser 2x Intel NUC5i7RYH avec les spécifications suivantes : 16 Go de RAM, processeur Intel Core i7-5557U et Ubuntu Server 16.04 en tant que système d'exploitation hôte. Pour les tests, les autres machines exécutant Windows 10 pro avec un processeur	Architecture monolithique versus architecture microservice (RQ1). Monolithique conteneurisé versus baremetal monolithique (RQ2). Microservice sur une machine versus un cluster d'ordinateurs(RQ3).	L'architecture monolithique requiert moins de ressources en termes de RAM et de CPU par rapport à un système de microservices. L'usage des conteneurs permet d'améliorer les performances en matière de consommation de RAM, de CPU, de débit et de latence.

		Intel Core i54690K et 16 Go de RAM.		
[24]	Application gestion d'une agence de location des voitures	Les tests ont été effectués sur un PC avec un système d'exploitation Ubuntu 18.04.2 LTS. La station de travail avait les paramètres suivants : Processeur - Intel i9-9900K 3,6 GHz-8 cores, 16 threads. Mémoire RAM - G.SKILL 32 Go DDR4 3200 MHz CL14. Disque dur - Samsung SSD 840 EVO 120 Go.	L'application est testée en utilisant trois variantes sans réplication, deux réplications et quatre réplications. Deux types de tests ont été réalisés 300 000 requêtes et 30 000 requêtes.	L'architecture de microservices est plus performante pour traiter un grand nombre de requêtes.
[22]	Application représente la conception de base pour la gestion de forums, de chats, de comment-aires ou de notificat-ions.	/	Le premier scénario génère des requêtes GET via HTTP vers les points d'extrémité de chaque service pour générer les bases de données et les données. Le deuxième scénario génère des requêtes GET via	L'architecture de microservices est plus performante en termes d'utilisation de la vitesse d'écriture sur disque, de consommation de mémoire, ainsi que de réception et de transmission de

			HTTP vers les points d'extrémité de chaque service pour sélectionner les informations.	données sur le réseau. Elle entraîne une consommation de CPU plus élevée par rapport à l'architecture monolithique.
[25]	Application représente un design de base pour la gestion de forums, de chats, de commentaires ou de notifications dans la partie Backend.	Processeur Intel Core i7 2,7 GHz, 16 Go de mémoire, HD disque dur et réseau virtuel. En ce qui concerne le logiciel, le système d'exploitation de base utilisé est Ubuntu 16.04, une version LTS pour ordinateur de bureau.	Deux scénarios ont été mis en place : le premier avec une application monolithique sur KVM, et le deuxième avec une application à base de microservices sur des conteneurs. Deux cas d'étude ont été générés. Le premier avec un nombre réduit de requêtes (273) à l'application et le deuxième cas avec un plus grand nombre de requêtes (1053).	L'architecture Monolithique, associée au KVM, présente des performances favorables en termes de CPU, de RAM et de réseau. Quant aux Microservices, ils se révèlent efficaces en termes de performance des ressources, d'application et de disque.
[20]	Application JHipster qui a été élaborée	JMeter installé sur un client distant connecté au serveur via	Le premier scénario de test est un test de charge qui évalue l'impact de	Pour les charges de travail normales, les performances sont

	comprenait trois services (JHipster Registry, l'applicat ion micro-service qui fournira les capacités de backend en exposant l'API et la passerelle	Ethernet, avec Docker pour exécuter les applications sur des conteneurs. Le serveur utilisé avait une mémoire de 16 Go et un CPU de 2,60 GHz.	l'augmentation du nombre d'utilisateurs sur le débit et le temps de réponse de l'application. Il commence avec 100 threads et augmente progressivement jusqu'à 7000 avec une montée en puissance de 2 minutes à chaque étape. Le deuxième scénario de test est un test de concurrence qui mesure la capacité du système à gérer les demandes de tous les services en même temps. Il commence avec 100 demandes pour chaque service et augmente progressivement jusqu'à 1000. Le troisième scénario a testé	similaires, mais les applications monolithiques ont une légère avance pour les faibles charges de travail. Les applications monolithiques ont une meilleure capacité de traitement des demandes et sont donc recommandées lorsque la rapidité de traitement est un objectif clé. Les microservices utilisant Consul comme technologie de découverte de services ont montré de meilleures performances que ceux utilisant Eureka, avec une amélioration de 4% en termes de débit.

			l'endurance du système avec 10000 requêtes et différentes technologies de découverte de services telles que Consul et Eureka.	
[26]	Azure Spring Cloud et Azure App Service(version Java de l'applic ation).	Un ordinateur PC fonctionnant sous Microsoft Windows version 10 Enterprise avec les paramètres matériels suivants : Intel(R) Core(TM) i7-9850H CPU 2, 60 GHz, six cœurs physiques et 12 cœurs logiques, et 32 Go de RAM (Dell Precision 7540). Apache JMeter6. Java - mis en œuvre en Java 84 avec le framework Spring Boot 2.3.0 .NET - mis en œuvre en C# version 8 avec	Deux scénarios d'essai ont été établis (environnement (Local/Cloud))selon la configuration de charge : -le nombre total de requêtes. -le nombre de threads qui seront utilisés pour exécuter le test, ce qui simule le nombre d'utilisateurs accédant simultanément à l'application. -la méthode de vérification des résultats.	-Environnement local : Les performances des applications à microservices ont été détériorées par une surcharge de communication entre la passerelle API et le service backend. Le système monolithique a traité plus de requêtes que les applications à microservices pour le service City non intensif en calcul. La plateforme Java est plus performante que .NET pour les services intensifs en calcul sur des

		le framework ASP.NET Core 3.1.		machines puissantes, mais moins performante sur des matériels à faible capacité de calcul pour les services moins intensifs. -Environnement Cloud : Les microservices et les versions monolithiques ayant les mêmes ressources CPU offrent des performances similaires.

TABLE 2.1 – Comparaison des articles de recherche sur l'architecture des microservices et l'architecture monolithique.

2.5 Benchmark pour les microservices

Un ensemble varié de benchmarks a été développé spécifiquement dans le but de tester et évaluer les microservices. Le tableau ci-dessus 2.2 offre une vue d'ensemble exhaustive des caractéristiques essentielles de chaque benchmark, permettant ainsi de prendre des décisions éclairées en fonction des besoins spécifiques d'évaluation et de comparaison des microservices [27].

	µBench	DeathStar Bench	TeaStore	Online Boutique	Bookinfo	Sock Shop	JPetStore	PetClinic
Objectifs	Évaluat	Évaluat	Évaluat	Évaluat	Évaluat	Évalua	Évaluat	Évaluat

	ion compar ative	ion compara tive, Démo	ion compara tive, Démo	ion compara tive, Démo	ion compara tive, Démo	tion compar ative, Démo	ion compara tive, Démo	ion compara tive, Démo
Nombre de applicati ons	Configu rable	3	1	1	1	1	1	1
Nombre de microser vices	Configu rable	19, 31, 33	6	11	4	14	4	5
Appeler une méthode	REST, gRPC	REST	REST	gRPC	gRPC	REST	REST	REST
Modèle de travail	Configu rable	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'appl icatio n	Dépend ant de l'applic ation	Dépend ant de l'applic ation
Maillage de services	Configu rable	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'applic ation	Dépend ant de l'appl icatio n	Dépend ant de l'applic ation	Dépend ant de l'applic ation
Modèle d'enver gure	Stochas tique, basé sur les traces	Orienté utilisat eur	Orienté utilisat eur	Orienté utilisat eur	Orienté utilisat eur	Orienté utilisa teur	Orienté utilisat eur	Orienté utilisat eur
Langage s	Python (autres	C++, Java,	Java	Node.js, Python,	Python, Java,	Java, Go,	Java	Java

	avec contene ur auxilia ire)	Node.js, etc.		Go, etc.	Ruby, Node.js	Node.js		
Platefor mes	Kubern etes	Kubernet es, Docker compose, Openshift	Kuberne tes, Docker compose	Kubern etes	Kuberne tes	Kubern etes, Docker compos e	Kubernet es	Kubern etes, Cloud- Found ry
Perform ance exporté e	Métriq ue, Traces	Métrique, Traces	Métriqu e, Traces	Métriqu e, Traces	Métriqu e, Traces	Métriq ue	Métrique, Traces	Métriqu e

TABLE 2.2 – Comparaison avec les applications de Benchmarking de microservices les plus populaires.

2.6 Conclusion

En conclusion de ce chapitre, nous avons analysé les principaux critères de comparaison des performances entre les microservices et les applications monolithiques. Nous avons approfondi notre étude en examinant les travaux de recherche existants dans ce domaine, qui ont mis en lumière les avantages et les inconvénients de chaque architecture. Enfin, nous avons présenté les résultats clés des articles étudiés, offrant ainsi une vue d'ensemble des différences observées entre ces deux approches. Les travaux analysés montrent que chaque architecture a ses propres avantages et inconvénients, et qu'elles trouvent leurs places optimales dans des contextes spécifiques.

Chapitre 3

Cas d'étude : Réseaux sociaux

3.1 Introduction

Ce chapitre est consacré à la présentation des réseaux sociaux que nous allons utiliser pour réaliser nos expériences de comparaison entre les architectures monolithiques et les architectures microservices. Nous avons choisi les réseaux sociaux comme ils occupent une place croissante dans notre vie quotidienne grâce à leurs nombreux avantages, leur facilité d'utilisation et les fonctionnalités attrayantes qu'ils offrent aux utilisateurs.

Nous commençons le chapitre par définir les réseaux sociaux, leurs fonctionnalités ainsi que leurs différentes applications, tout en mettant en évidence leurs avantages et leurs inconvénients. Ensuite, nous présenterons les fonctionnalités spécifiques de notre application ensuite nous modélisant l'application avant de conclure.

3.2 Réseaux sociaux

3.2.1 Définition

La notion "réseau social" (social network en Anglais) a été introduite dans le domaine de la sociologie par J. Arundel Barnes, un anthropologue britannique, qui l'a utilisée pour décrire les ensembles d'individus et les relations qu'ils entretiennent les uns avec les autres au sein d'une communauté [28].

Un réseau social est un ensemble d'entités (individus ou organisations) reliées entre elles par divers types de relations, telles que les relations familiales, amicales ou professionnelles [29]. Cet ensemble peut être organisé (tel qu'une entreprise) ou non (comme un réseau d'amis) [28].

Dans le domaine des technologies, un réseau social consiste en un service permettant de regrouper diverses personnes afin de créer un échange sur un sujet particulier ou non [30].

Un réseau social est un site Internet en ligne ou une application mobile où les utilisateurs peuvent créer leur profil et interagir avec d'autres personnes en partageant du contenu, des idées, des informations, des intérêts et des activités. Ces plateformes permettent à des millions de personnes de se connecter et de communiquer virtuellement, en publiant et partageant différents types de contenus tels que des textes, des photos, des vidéos, etc., indépendamment de leur situation géographique [29].

3.2.2 Fonctionnalité d'un réseau social

Les fonctionnalités d'un réseau social varient selon la plateforme et les besoins des utilisateurs, nous citons ci-après quelques fonctionnalités courantes [29] [31] :

Création de profil

Tout réseau social traditionnel doit permettre à chaque utilisateur de créer son propre profil en remplissant une fiche d'identité. Cette dernière doit contenir des informations telles qu'un identifiant (une adresse e-mail ou un numéro de téléphone), un mot de passe et un pseudonyme. Ensuite, l'utilisateur peut compléter son profil en y ajoutant des informations personnelles comme son nom, sa photo de profil, sa ville de résidence, sa date de naissance, etc.

Publication

Après avoir créé leur profil, les utilisateurs peuvent contribuer aux réseaux sociaux en publiant divers types d'informations et de contenus sur leur profil personnel ou sur

des pages de groupes. Ils peuvent partager des publications qui pourraient intéresser les autres utilisateurs.

Navigation

La navigation sur un réseau social permet à l'utilisateur d'explorer la plateforme, de découvrir de nouveaux contenus, de consulter son fil d'actualité, de lire et d'interagir avec des publications.

Interactions avec les autres utilisateurs

Les fonctionnalités de commentaires et de réactions permettent aux utilisateurs de réagir aux publications d'autres utilisateurs en laissant des commentaires, des mentions "j'aime", ou d'autres types de réactions. Il est également possible pour les utilisateurs de partager une publication d'un autre membre sur leur propre profil.

Connexion aux autres utilisateurs

Les utilisateurs ont la possibilité de rechercher d'autres utilisateurs ou des groupes et de se connecter avec eux en envoyant des demandes d'amitié ou de connexion.

Chat et messagerie

Les utilisateurs peuvent communiquer en temps réel avec leurs amis ou d'autres utilisateurs via des chats ou des messageries intégrées à la plateforme.

Jeux

Les jeux en ligne est l'une des fonctionnalités les plus anciennes dans les réseaux sociaux avec Friendster. Les jeux sur les réseaux sociaux sont des applications de jeu en ligne qui permettent aux utilisateurs de jouer en mode individuel ou en groupe avec leurs amis ou d'autres joueurs connectés sur la même plateforme sociale.

Événements

Les utilisateurs peuvent créer et partager des événements en ligne et inviter des amis à y participer.

3.2.3 Applications des réseaux sociaux

Les réseaux sociaux ne sont pas seulement utilisés pour la communication et le divertissement, mais leur utilisation s'est également étendue à de nombreux autres aspects [29] [31].

Communication

Les réseaux sociaux permettent aux utilisateurs de communiquer et d'interagir avec d'autres personnes, qu'elles soient des amis, des membres de la famille ou des inconnus partageant des intérêts communs.

Information

Les réseaux sociaux offrent au grand public un outil facile et puissant de diffusion d'information, comme ils peuvent être une source importante d'informations pour les utilisateurs, notamment pour les nouvelles, les événements et les tendances.

Marketing

Les entreprises utilisent les réseaux sociaux pour promouvoir leurs produits et services, atteindre un public plus large et interagir avec leurs clients.

Analyse des réseaux sociaux

La recherche en analyse des réseaux sociaux est un domaine émergent qui vise à étudier les tendances sociologiques, économiques ou politiques à partir des données publiées par les utilisateurs sur les réseaux sociaux.

Recrutement

Les réseaux sociaux sont devenus une ressource importante pour les entreprises en matière de recrutement, leur permettant de rechercher des candidats potentiels, d'évaluer leur adéquation culturelle et leurs compétences. Ils offrent de nombreux avantages tels que la transparence du marché de l'emploi, la diffusion rapide et à grande échelle des offres d'emploi, et ce, à moindre coût.

Éducation

Les enseignants peuvent utiliser les réseaux sociaux pour interagir avec leurs étudiants, partager des ressources et créer des groupes d'étude.

3.2.4 Types de réseaux sociaux

Réseaux généralistes/ Réseaux spécialisés [29]

Les réseaux sociaux généralistes sont des plateformes populaires qui offrent la plupart des fonctionnalités communes des réseaux sociaux

Les réseaux spécialisés sont centrés sur des thématiques spécifiques et peuvent inclure plusieurs sous-catégories telles que les réseaux communautaires, professionnels, académiques ou de partage de contenus.

Réseaux de relations /réseaux de contenu

Les réseaux de relations mettent l'accent sur les liens et les interactions entre les utilisateurs. Ils permettent aux individus de se connecter et de créer des relations virtuelles avec d'autres utilisateurs.

Les réseaux de contenu se concentrent sur le partage et la diffusion de contenu tel que des vidéos, des images, des articles, des podcasts, etc. Ces réseaux sociaux sont centrés autour du contenu lui-même plutôt que sur les relations entre les utilisateurs.

Classification à base des objectifs

Selon le chercheur Thelwall [32], les réseaux sociaux peuvent être classés en fonction de leurs objectifs distincts, à savoir la socialisation, le réseautage ou la navigation.

3.2.5 Avantages et inconvénients des réseaux sociaux

Les réseaux sociaux sont devenus une partie intégrante de notre vie quotidienne, offrant de nombreux avantages tels que [29] [33] :

La connectivité : Les réseaux sociaux permettent aux utilisateurs de se connecter avec des personnes du monde entier et de créer des liens avec des amis, des membres

de la famille, des collègues, etc. Ils peuvent également aider les utilisateurs à renforcer leur réseau professionnel.

Les médias sociaux peuvent être utiles pour augmenter le trafic vers un site e-commerce, par exemple, et pour améliorer la relation client en général, et trouver de nouveaux clients et partenaires.

La communication : Les réseaux sociaux offrent des moyens de communication rapides et efficaces pour échanger des informations, des nouvelles et des idées.

Facilité d'utilisation et d'administration : Les réseaux sociaux ont simplifié la publication de divers formats d'informations pour le grand public sans nécessiter de recourir à un hébergeur ou à un spécialiste en informatique. Cela a rendu leur utilisation et leur gestion plus facile pour les utilisateurs.

La liberté d'expression sur les réseaux sociaux permet aux utilisateurs de s'exprimer librement et de partager leurs opinions et idées. Cela favorise le débat, l'échange d'informations et la mobilisation autour de causes sociales.

Cependant, les réseaux sociaux ont également présentant certains inconvénients :

La dépendance : Les utilisateurs peuvent devenir dépendants des réseaux sociaux, ce qui peut nuire à leur santé mentale et physique.

Crédibilité des informations : Les informations publiées sur les réseaux sociaux peuvent manquer de fiabilité, car il est facile de partager rapidement des informations erronées ou des avis non fondés (fake news), sans nécessairement être un expert dans le domaine.

Le cyberharcèlement : Les réseaux sociaux peuvent être utilisés pour harceler, intimider ou menacer les utilisateurs, ce qui peut avoir des conséquences graves sur leur bien-être émotionnel.

3.2.6 Exemples de réseaux sociaux

Facebook : La plus grande plateforme de réseau social au monde, qui permet aux utilisateurs de se connecter avec des amis, de partager du contenu, de rejoindre des groupes et de suivre des pages d'intérêt.

YouTube : Une plateforme de partage de vidéos où les utilisateurs peuvent télécharger,

regarder, commenter et partager des vidéos sur une large gamme de sujets.

WhatsApp : Une application de messagerie instantanée qui permet aux utilisateurs d'envoyer des messages, des appels vocaux et vidéo, et de partager des fichiers avec leurs contacts.

Twitter : Un réseau social de microblogging où les utilisateurs peuvent publier des messages courts appelés "tweets" et suivre d'autres utilisateurs pour rester informés des actualités, des tendances et des discussions en cours.

LinkedIn : Un réseau social professionnel qui permet aux utilisateurs de créer des profils professionnels, de se connecter avec d'autres professionnels, de rechercher des opportunités d'emploi et de développer leur réseau professionnel.

La figure 3.1 ci-dessous [34] présente les réseaux sociaux les plus populaires dans le monde, classés par le nombre d'utilisateurs actifs mensuels, tel que mesuré en janvier 2023.

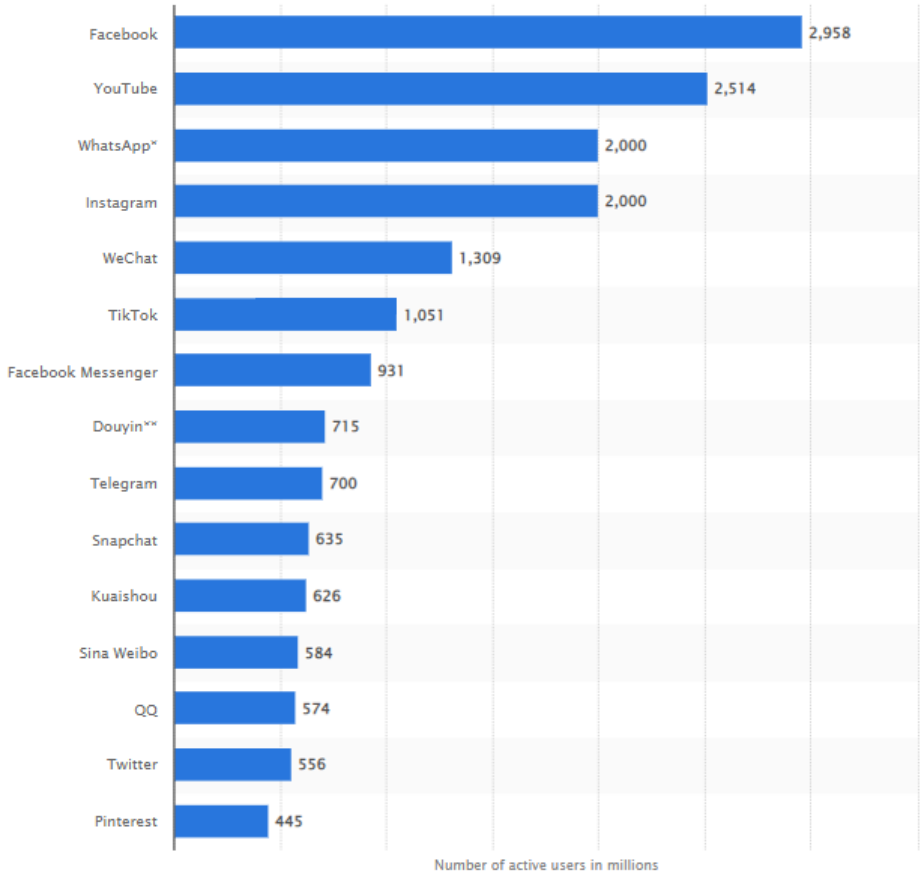

FIGURE 3.1 – Classification des réseaux sociaux selon le nombre des utilisateurs.

3.2.7 Réseau social développé pour réaliser nos tests

Nous allons décrire par la suite le Réseau social que nous avons développé pour réaliser nos tests de comparaison entre les deux architectures monolithique et microservices.

Fonctionnalités de notre application

Un service de création de profil permettant aux utilisateurs de remplir une fiche d'identité comprenant des informations personnelles telles que le nom d'utilisateur, l'adresse e-mail et un mot de passe.

Un service de publication où les utilisateurs peuvent partager du contenu et interagir en laissant des commentaires et des mentions "j'aime".

Un service de messagerie permettant aux utilisateurs de communiquer avec d'autres utilisateurs.

3.3 Modélisation

Nous avons adopté une approche de développement méthodique pour concevoir notre application, en respectant un processus UP (Unified Process).

Le Processus Unifié est principalement un processus de développement logiciel, qui se concentre sur les activités essentielles pour convertir les exigences d'un utilisateur en un système logiciel fonctionnel [35].

L'une des étapes essentielles de ce processus a été la modélisation, où nous avons utilisé les diagrammes d'UML pour représenter les différentes perspectives et fonctionnalités de notre système de manière claire et structurée.

UML (Unified Modeling Language) est un langage de modélisation graphique utilisé pour visualiser, spécifier, construire et documenter les systèmes logiciels. Il fournit des diagrammes et des notations standardisées pour représenter les différentes parties d'un système et les relations entre elles.

3.3.1 Diagramme de cas d'utilisation

Le diagramme de cas d'utilisation est un outil de modélisation graphique pour décrire les interactions entre les utilisateurs et un système informatique, en montrant les actions possibles des utilisateurs et les réponses du système. Cela permet de comprendre les besoins des utilisateurs et les exigences fonctionnelles.

Le réseau social que nous avons créé à un seul acteur c'est l'utilisateur qui est chargé de gérer les publications, d'interagir avec le contenu et de communiquer avec les autres utilisateurs.

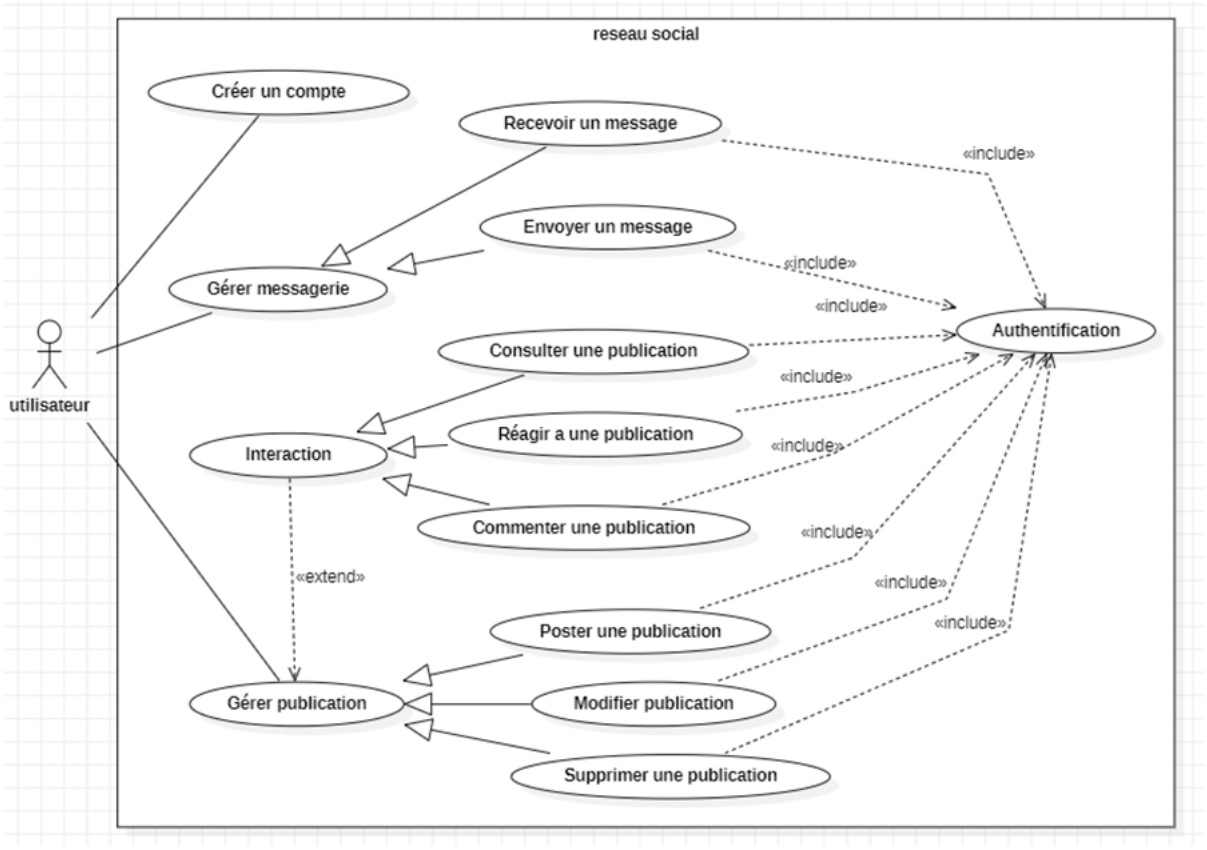

FIGURE 3.2 – Diagramme de cas d'utilisation du réseau développé.

3.4 Conclusion

Ce chapitre explore les réseaux sociaux et présente en détail le réseau social que nous avons développé. Nous avons analysé les différentes caractéristiques et fonctionnalités des réseaux sociaux, ainsi que le rôle des utilisateurs au sein de notre plateforme. Nous

avons également examiné les interactions et les possibilités de communication entre les utilisateurs. Ce chapitre constitue une base solide pour notre étude comparative entre les architectures monolithiques et les architectures microservices dans le contexte des réseaux sociaux.

Chapitre 4

Implémentation, Test et résultats

4.1 Introduction

Ce chapitre présente le développement et l'évaluation du réseau social que nous avons développée en version monolithique et en version microservices. Nous débutons le chapitre en présentant brièvement l'application, puis nous expliquons les outils et les langages utilisés pour son développement. Ensuite, nous présentons quelques interfaces développées. Ensuite, nous exposons les tests de comparaison entre la version monolithique et la version microservices. Enfin, nous clôturons par une conclusion.

4.2 Application développée

Afin d'effectuer nos tests de comparaison, Nous avons développé deux version d'un réseau social : l'une avec une architecture monolithique, tandis que l'autre repose sur une architecture basée sur les microservices, avec les mêmes interfaces.

4.2.1 L'application que nous avons développée

La version microservices 4.1 est composée principalement de trois services : le service de connexion ("login"), le profil et la messagerie. Tous ces services sont mis en œuvre à l'aide de Spring Boot.

Pour le service de connexion, nous avons mis en place un mécanisme sécurisé permettant aux utilisateurs de s'authentifier, en utilisant firebase.

Le service de profil offre une expérience conviviale et personnalisée. Les utilisateurs ont la possibilité de créer et de mettre à jour leur profil en partageant des statuts et des publications. De plus, ils peuvent interagir avec les publications d'autres utilisateurs en fonction de leurs intérêts et préférences.

Quant au service de messagerie, nous avons exploité les fonctionnalités avancées de Spring Boot pour fournir une plateforme de communication instantanée et sécurisée. L'utilisation de Spring WebSockets nous a permis de garantir des communications rapides et bidirectionnelles.

Nous avons utilisé MySQL comme base de données pour stocker les données de manière distribuée et scalable.

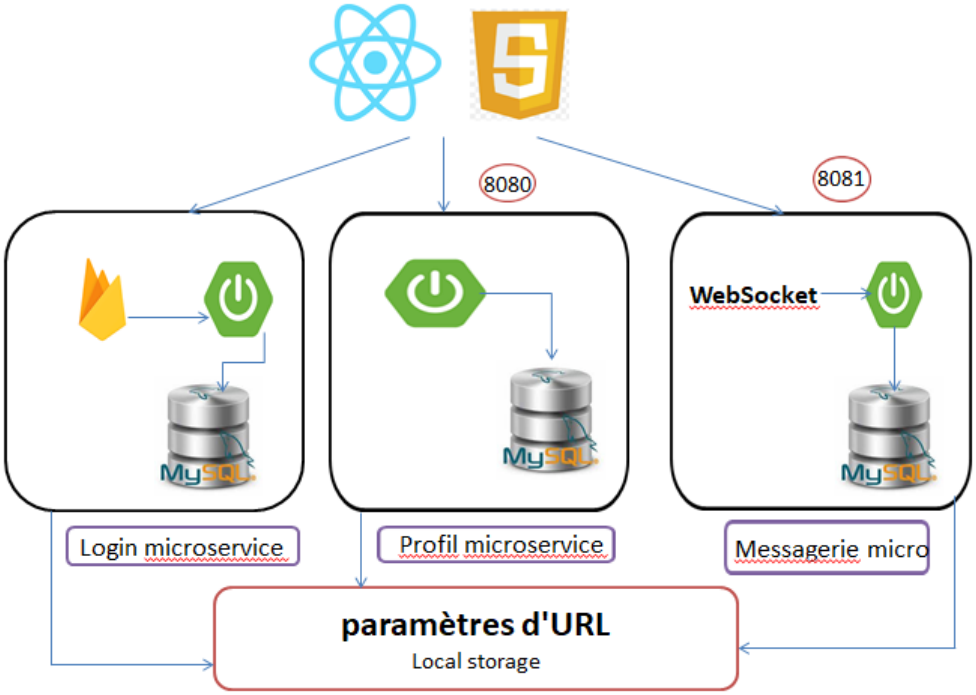

FIGURE 4.1 – Architecture de l'application.

Nous avons partagé notre code sur GitHub via le lien https://github.com/Thaninasaf/social-media .

4.3 Les outils

Lors du développement de notre application, nous avons utilisé une combinaison d'outils puissants et modernes pour garantir une mise en œuvre efficace et de qualité. Dans la suite, nous allons présenter brièvement les principaux outils que nous avons utilisés :

Un ordinateur HP Laptop 15-dw0xxx Intel(R) Core(TM) i5-8265U avec les caractéristiques suivantes :

- CPU @1.60 GHz 1.80GHz.
- 8.00 Go.
- Un système d'exploitation Windows 10 de 64 bits.

Visual Studio Code

Visual Studio Code est un éditeur de code léger, polyvalent et gratuit, développé par Microsoft. Il prend en charge de nombreux langages de programmation. Doté de fonctionnalités avancées telles que la coloration syntaxique, l'achèvement automatique du code, le débogage intégré et la gestion des contrôles de version, il est largement utilisé par les développeurs. Avec son interface conviviale et la possibilité d'installer des extensions pour étendre ses fonctionnalités, Visual Studio Code facilite le développement de logiciels dans différents langages de programmation [36].

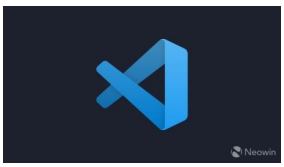

FIGURE 4.2 – Visual Studio Code.

Spring Tool Suite (STS)

C'est un environnement de développement intégré (IDE) conçu spécifiquement pour le développement d'applications basées sur Spring Framework. Il offre des fonctionnalités avancées telles que la génération automatique de code, la navigation facilitée, le débogage intégré et la gestion des dépendances. STS simplifie le processus de développement en fournissant des outils spécifiques à Spring, ce qui permet aux développeurs de créer des applications Spring de manière plus efficace et productive [37].

FIGURE 4.3 – Spring Tool Suite (STS).

4.3.1 Les langages utilisés

Frontend

Html : Hypertext Markup Language(HTML) est le langage de balisage standard utilisé pour créer des pages web. Il permet de structurer et de présenter le contenu d'une page en utilisant des balises et des éléments prédéfinis. C'est le langage fondamental pour la création de sites web et constitue la base sur laquelle repose l'ensemble du contenu et de l'interactivité d'une page web [38].

FIGURE 4.4 – HTML.

CSS : (Cascading Style Sheets) : est un langage de style qui permet de définir l'apparence et la mise en forme des pages web. Il agit en complément du langage HTML en fournissant des règles et des instructions pour styliser les éléments HTML [39].

FIGURE 4.5 – CSS.

JavaScript : JavaScript est un langage de programmation largement utilisé pour ajouter des fonctionnalités interactives aux pages web. Il permet de manipuler et de modifier le contenu HTML, d'effectuer des requêtes HTTP asynchrones, d'effectuer des animations et bien plus encore. JavaScript s'exécute directement dans le navigateur web, ce qui permet de créer des expériences interactives en temps réel sans avoir besoin de recharger la page. En résumé, JavaScript est un langage polyvalent qui améliore l'interactivité et la dynamique des sites web [40].

FIGURE 4.6 – JavaScript.

React JS : React est une bibliothèque JavaScript pour la création d'interfaces utilisateur interactives. Elle simplifie le processus de développement d'applications web dynamiques et performantes en utilisant des composants réutilisables et en optimisant le Virtual DOM [41].

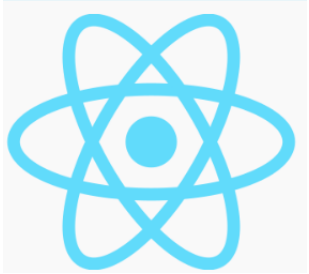

FIGURE 4.7 – React JS.

Backend

Java : Nous avons développé notre application en utilisant Java, un langage de programmation polyvalent et puissant orienté objet qui permet de créer des applications complètes, ainsi que des petits modules d'application intégrables dans une page Web [42].

FIGURE 4.8 – Java.

4.3.2 framework

Spring boot

Pour le développement de notre application, nous avons utilisé Spring Boot, un framework Java puissant et populaire. Java Spring Boot est un outil open source qui facilite l'utilisation d'infrastructures Java pour créer des microservices et des applications web en fournissant des fonctionnalités prêtes à l'emploi et des configurations par défaut intelligentes [43].

FIGURE 4.9 – Spring bootS.

4.3.3 WebSocket

WebSocket est un protocole de communication en temps réel entre navigateur et serveur. Il permet une transmission instantanée et bidirectionnelle des données. Utilisé dans

les applications nécessitant une communication en temps réel, comme les chats ou les mises à jour en direct. WebSocket offre une alternative rapide et réactive aux méthodes de communication traditionnelles. En bref, c'est un protocole pour la communication en temps réel entre navigateur et serveur [44].

4.3.4 Firebase

Firebase est une plateforme de développement d'applications mobiles et web, fournie par Google. Elle offre un ensemble d'outils et de services cloud permettant de développer, déployer et gérer des applications de manière plus facile et plus rapide [45].

Firebase offre une variété de fonctionnalités prêtes à l'emploi, dont nous avons utilisé l'authentification des utilisateurs et le stockage de fichiers. Ces fonctionnalités intégrées nous ont permis de sécuriser l'accès aux utilisateurs et de gérer efficacement les fichiers dans notre application.

FIGURE 4.10 – Firebase.

4.4 Interfaces

Les interfaces que nous avons développées ont été soigneusement élaborées afin de fournir une expérience utilisateur optimale. Nous présentons ci-dessous un aperçu de certaines de ces interfaces :

1. **Interface de connexion ("login")**

   - Cette interface permet aux utilisateurs de saisir leurs identifiants (email et un mot de passe) pour s'authentifier de manière sécurisée et accéder à l'application.

   - L'interface est conçue de manière à être simple et intuitive, avec des champs clairement étiquetés et des instructions précises.

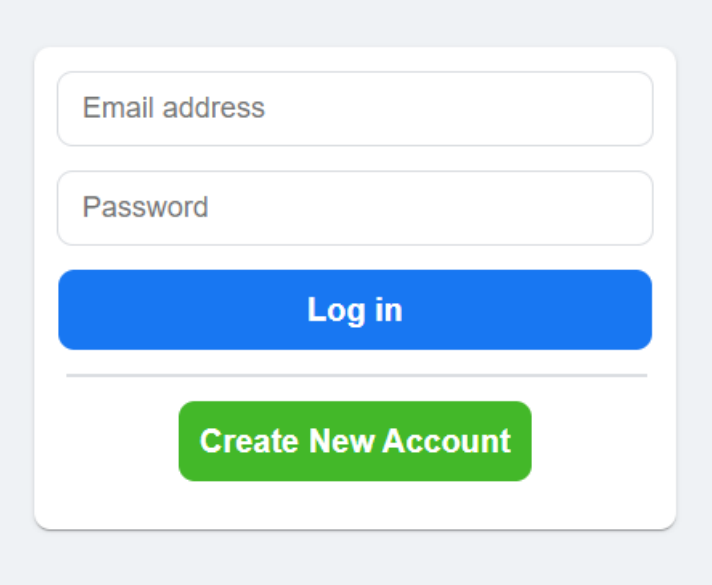

FIGURE 4.11 – Interface de connexion ("login").

Interface de l'inscription

Cette interface offre aux utilisateurs la possibilité de créer un compte personnalisé en fournissant simplement leur nom, leur adresse e-mail et leur mot de passe. En remplissant ces informations essentielles, les utilisateurs peuvent s'inscrire rapidement et facilement :

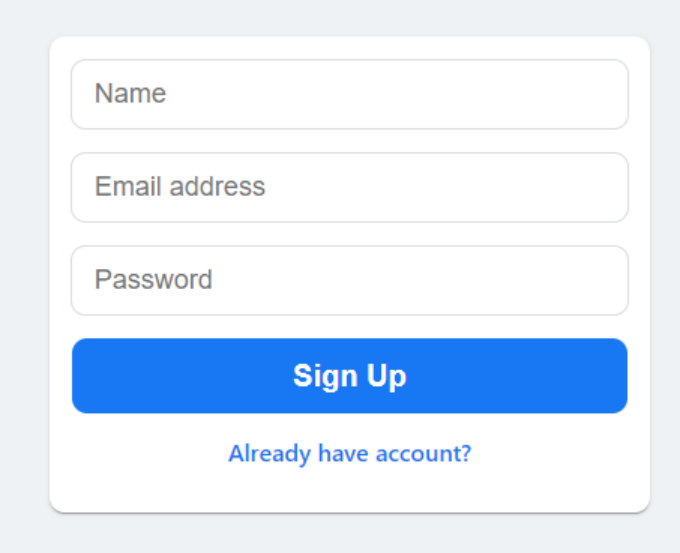

FIGURE 4.12 – Interface de l'inscription.

2. **Interface de profil**

- Cette interface permet aux utilisateurs de créer et de gérer leur profil personnel.

- L'interface peut être divisée en sections ou onglets clairs pour organiser les différentes informations et rendre la navigation plus fluide.

- Un microservice de profil a été mis en place, permettant à un utilisateur de créer des publications.

- Une fonctionnalité a été implémentée pour permettre aux utilisateurs d'interagir avec d'autres publications en utilisant la mention 'j'aime'.

- De plus, la possibilité de laisser des commentaires sur les publications a été ajoutée."

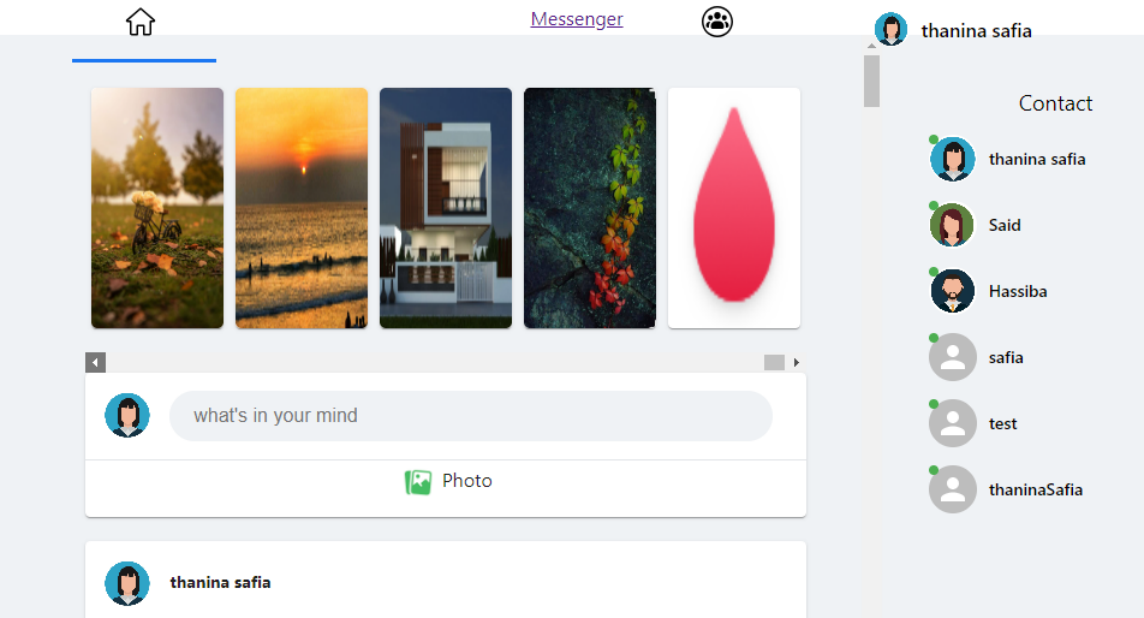

FIGURE 4.13 – Interface de profil utilisateur.

Interface statuts

Notre interface propose une fonctionnalité attrayante pour les utilisateurs : la possibilité de partager des statuts. En cliquant sur l'icône "+", les utilisateurs peuvent créer et publier leurs propres statuts.

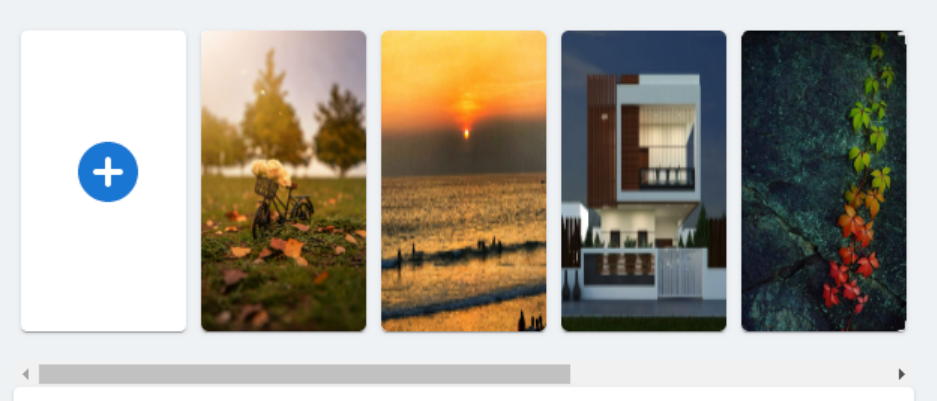

FIGURE 4.14 – Interface de profil utilisateur(Interface statuts).

Interface poster une publication

L'interface de notre application offre la possibilité à l'utilisateur de poster facilement une publication. En cliquant sur l'option "photo" :

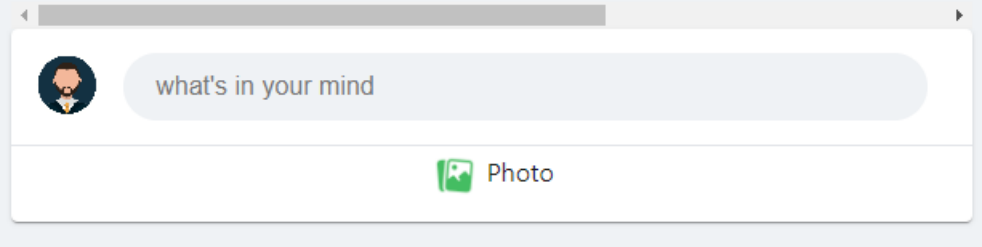

FIGURE 4.15 – Interface de profil utilisateur(Interface poster une publication).

L'utilisateur peut choisir une image à inclure dans sa publication, ainsi qu'un texte associé. Cette fonctionnalité permet à l'utilisateur d'exprimer ses idées et de partager du contenu visuel avec les autres utilisateurs de manière simple et intuitive.

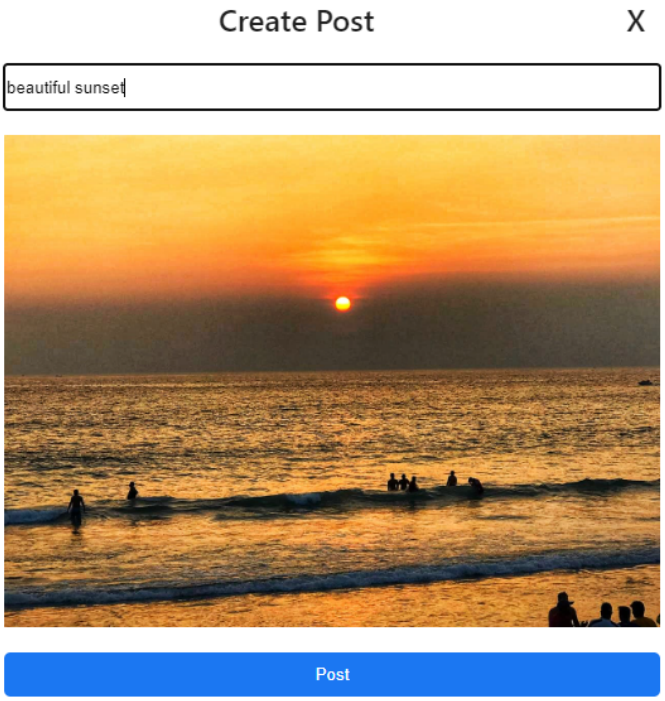

FIGURE 4.16 – La création d'une publication.

Voici un exemple concret d'un post réalisé dans notre application :

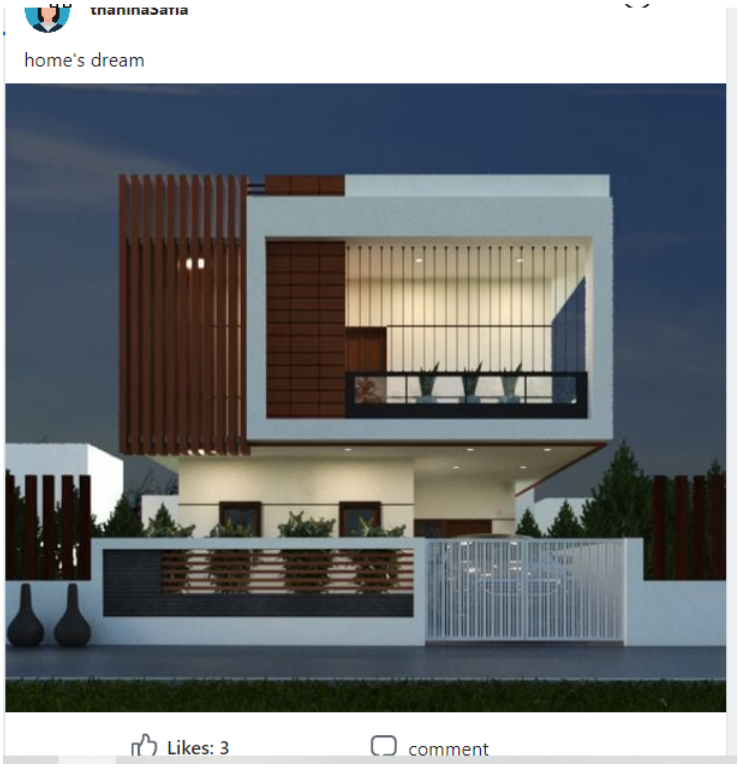

FIGURE 4.17 – Poster la publication.

Interface pour la publication d'un commentaire

L'interface de notre application propose une fonctionnalité permettant aux utilisateurs d'exprimer leurs idées en publiant des commentaires. En cliquant sur l'option "What's in your mind", les utilisateurs peuvent partager leurs réflexions, opinions ou réactions sur une discussion ou un sujet donné.

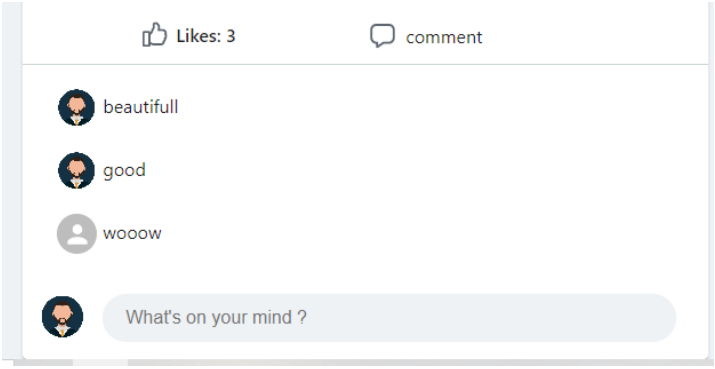

FIGURE 4.18 – Interface pour la publication d'un commentaire.

3. **Interface de messagerie**

- Cette interface permet aux utilisateurs de communiquer de manière instantanée et efficace.

- L'interface peut afficher des notifications en temps réel pour informer les utilisateurs des nouveaux messages entrants.

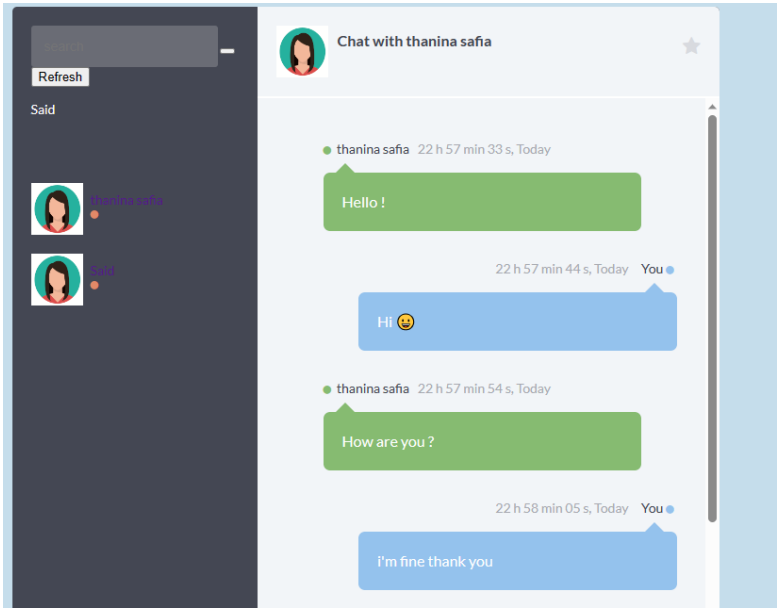

FIGURE 4.19 – Interface de messagerie.

4.5 Test et résultats

Cette partie expose les tests de comparaison entre la version monolithique et la version basée sur l'architecture microservices de notre réseau social.

Nous avons utilisé JMeter [46] pour réaliser nos tests de performances. JMeter est un outil open source spécialisé dans les tests de charge et la simulation de trafic réseau, il permet d'évalué les performances et les temps de réponse de l'application. Nos tests ont permis ainsi d'analyser la réaction de chaque version du reseau social face à différentes charges de travail. Pour ce faire, nous avons effectué des tests avec JMeter en simulant des utilisateurs, 50 pour le premier test et 100 pour le deuxième test. Les durées de ces tests étaient respectivement d'environ 50 secondes et de 1 minute 30 secondes, avec une répartition uniforme des requêtes.

Nous avons lancé cinq types de requêtes pour interroger les différents services de notre réseau, les requêtes sont : getComment, getPost, getStatus, getUser et statusSavepost.

4.5.1 Tests pour 50 utilisateurs

FIGURE 4.20 – Temps de réponse en fonction du temps

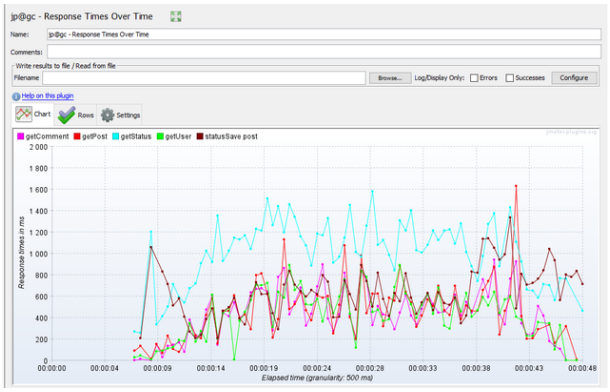

FIGURE 4.21 – (a) Réseau social microservices

FIGURE 4.22 – (b) Réseau social monolithique

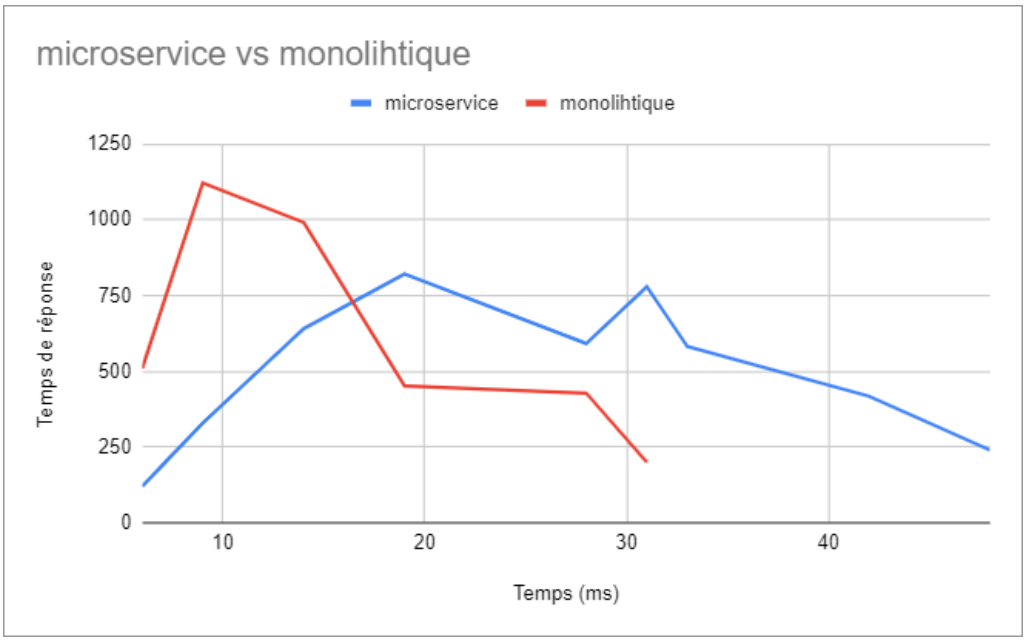

FIGURE 4.23 – c) Réseau social microservices vs réseau social monolithique

La figure 4.20 représente le temps de réponse en fonction du temps pour 50 utilisateurs. La figure 4.21 concerne la version basée sur l'architecture microservices, tandis que la figure 4.22 porte sur la version monolithique. La figure 4.23 résume les informations des figures 4.21 et 4.22 pour une meilleure lisibilité.

L'analyse des différentes figures montre que le temps de réponse de l'architecture microservices est meilleur dans l'intervalle [0, 19] ms, tandis que le temps de réponse de l'architecture monolithique est meilleur dans l'intervalle [19, 32] ms.

Les microservices maintiennent une stabilité temporelle avec de faibles variations entre les échantillons, affichant principalement des valeurs comprises entre 500 et 800 ms, avec un pic à 822 ms. L'architecture monolithique présente des performances ponctuelles supérieures. Par exemple, à partir de 19 ms, le temps de réponse de l'architecture monolithique est de 452 ms, tandis que celui des microservices est de 822 ms. Cependant, par la suite, les temps de réponse de l'architecture monolithique se stabilisent à des niveaux proches de ceux des microservices.

En conclusion, bien que l'architecture monolithique puisse occasionnellement afficher des performances supérieures à certains moments, l'architecture microservices offre généralement des temps de réponse plus stables et plus rapides pour une utilisation continue.

FIGURE 4.24 – Débit en fonction du temps

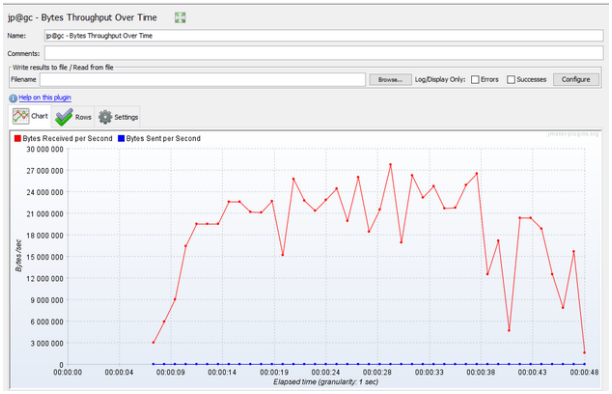

FIGURE 4.25 – (a) Réseau social microservices

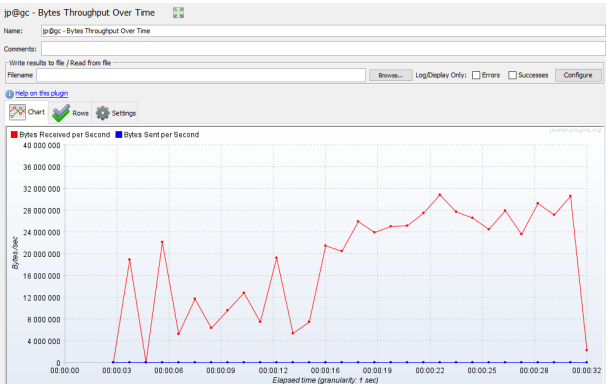

FIGURE 4.26 – (b) Réseau social monolithique

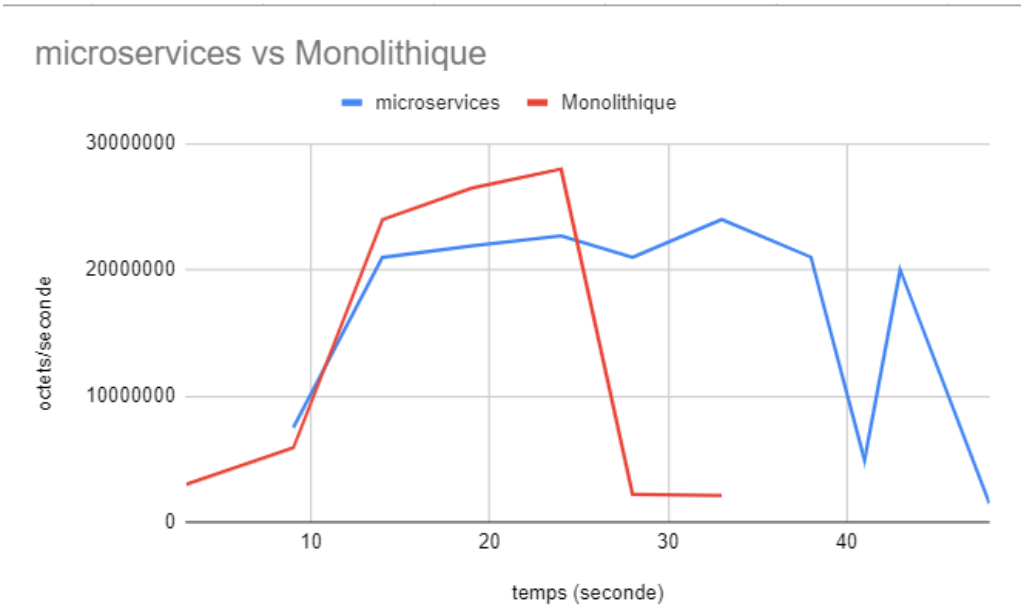

FIGURE 4.27 – c) Réseau social microservices vs réseau social monolithique

La figure 4.24 représente le Débit en fonction du temps pour 50 utilisateurs. La figure 4.25 concerne la version basée sur l'architecture microservices, tandis que la figure 4.26 porte sur la version monolithique. La figure 4.27 résume les informations des figures 4.25 et 4.26 pour une meilleure lisibilité.

Les figures montrent que l'architecture monolithique a affiché une performance de débit supérieure à l'architecture microservices. L'architecture monolithique a atteint un débit maximal de 28 000 000 octets/seconde en 24 secondes, tandis que les microservices ont

atteint 24 000 000 octets/seconde en 33 secondes. Cependant, le débit du monolithe monolithique à partir de 19 ms.

Ces données indiquent qu'en général, l'architecture monolithique surpasse l'architecture microservices en termes de débit en fonction du temps.

FIGURE 4.28 – Temps de réponse par rapport aux threads

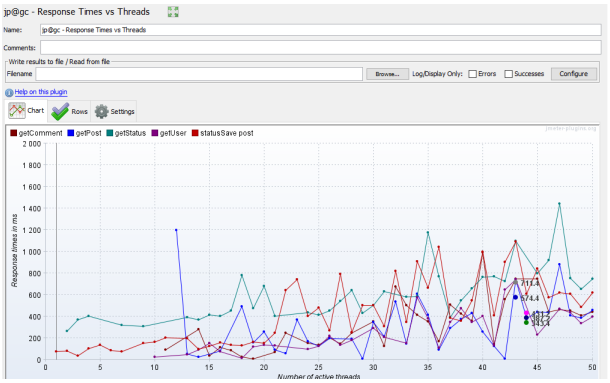

FIGURE 4.29 – (a) Réseau social microservices

FIGURE 4.30 – (b) Réseau social monolithique

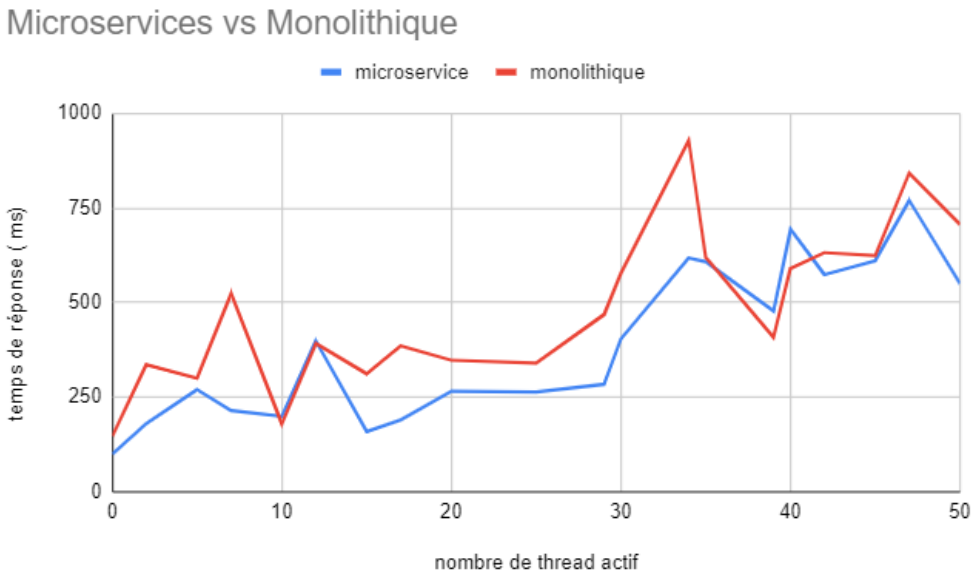

FIGURE 4.31 – c) Réseau social microservices vs réseau social monolithique

La figure 4.28 représente le temps de réponse par rapport aux threads en fonction du temps pour 50 utilisateurs. La figure 4.29 concerne la version basée sur l'architecture microservices, tandis que la figure 4.30 porte sur la version monolithique. La figure 4.31 résume les informations des figures 4.29 et 4.30 pour une meilleure lisibilité.

Pour l'architecture microservices, nous constatons une progression des temps de réponse avec des fluctuations mineures, atteignant un maximum de 770 millisecondes avec 47 threads actifs. En revanche, l'architecture monolithique présente des temps de réponse généralement supérieurs, bien que moins sujets aux fluctuations, atteignant un maximum de 928 millisecondes avec 34 threads actifs. Ces résultats suggèrent que l'architecture microservices offre une stabilité relative dans les temps de réponse avec une montée en charge, tandis que l'architecture monolithique peut connaître des pics de temps de réponse plus élevés. Le choix final dépendra des exigences spécifiques de l'application en matière de performances.

4.5.2 Tests pour 100 utilisateurs

FIGURE 4.32 – Temps de réponse en fonction du temps

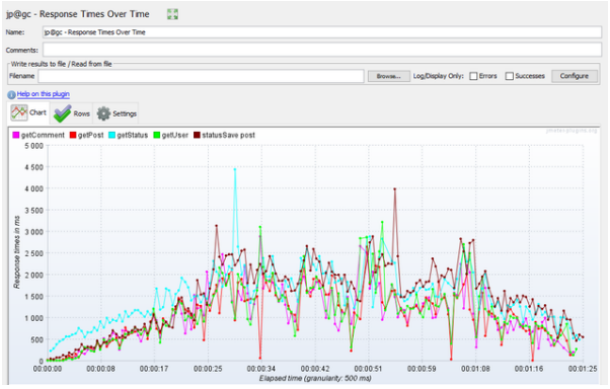

FIGURE 4.33 – (a) Réseau social microservices

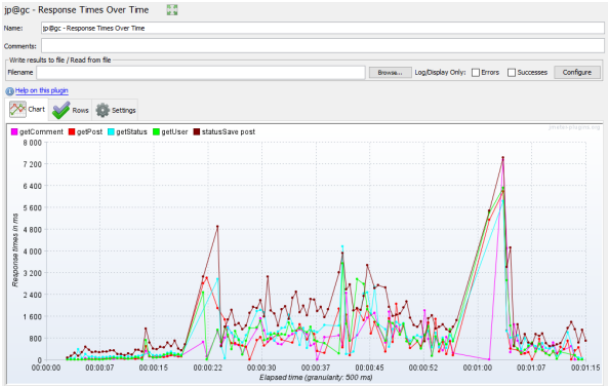

FIGURE 4.34 – (b) Réseau social monolithique

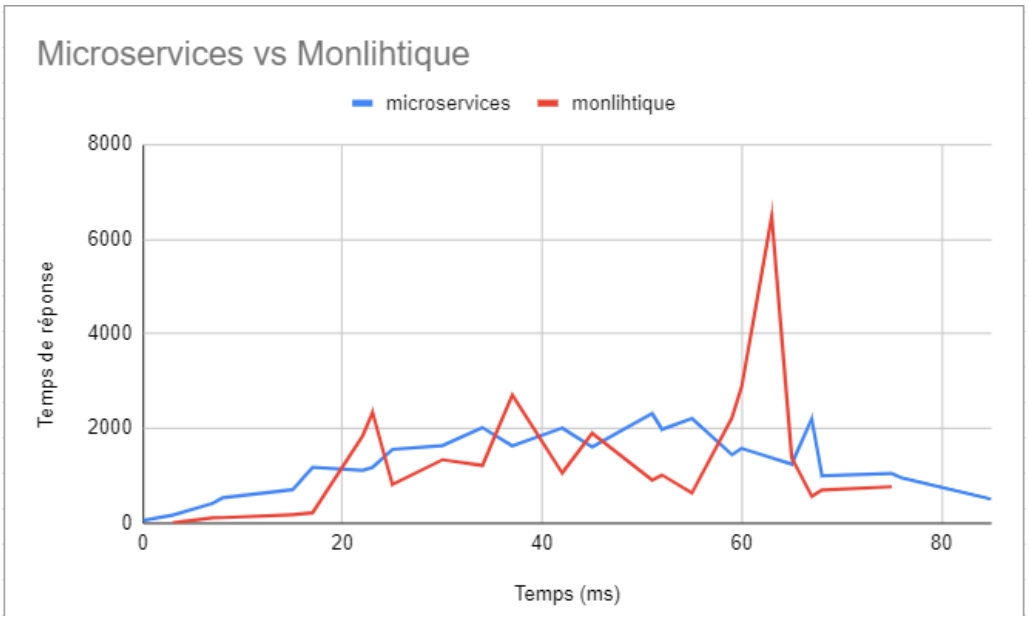

FIGURE 4.35 – c) Réseau social microservices vs réseau social monolithique

La figure 4.32 représente le temps de réponse en fonction du temps pour 100 utilisateurs. La figure 4.33 concerne la version basée sur l'architecture microservices, tandis que la figure 4.34 porte sur la version monolithique. La figure 4.35 résume les informations des figures 4.33 et 4.34 pour une meilleure lisibilité.

Les figures montrent que l'architecture microservices se distingue par une stabilité accrue des temps de réponse et une performance généralement meilleure, avec un temps de réponse maximal de 2320 ms. En revanche, l'architecture monolithique présente des temps de réponse proches de ceux des microservices, mais elle affiche des fluctuations plus marquées et des temps de réponse maximaux bien plus élevés, atteignant un pic à 6480 ms. Ainsi, il est clairement démontré que, pour cette charge utilisateur, l'architecture microservices offre une meilleure expérience en termes de stabilité du temps de réponse en fonction du temps.

FIGURE 4.36 – Débit en fonction du temps

FIGURE 4.37 – (a) Réseau social microservices

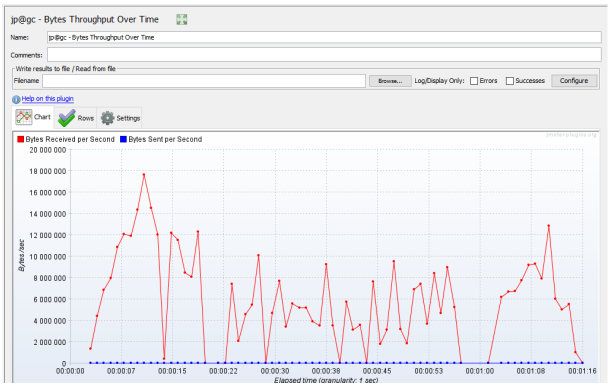

FIGURE 4.38 – (b) Réseau social monolithique

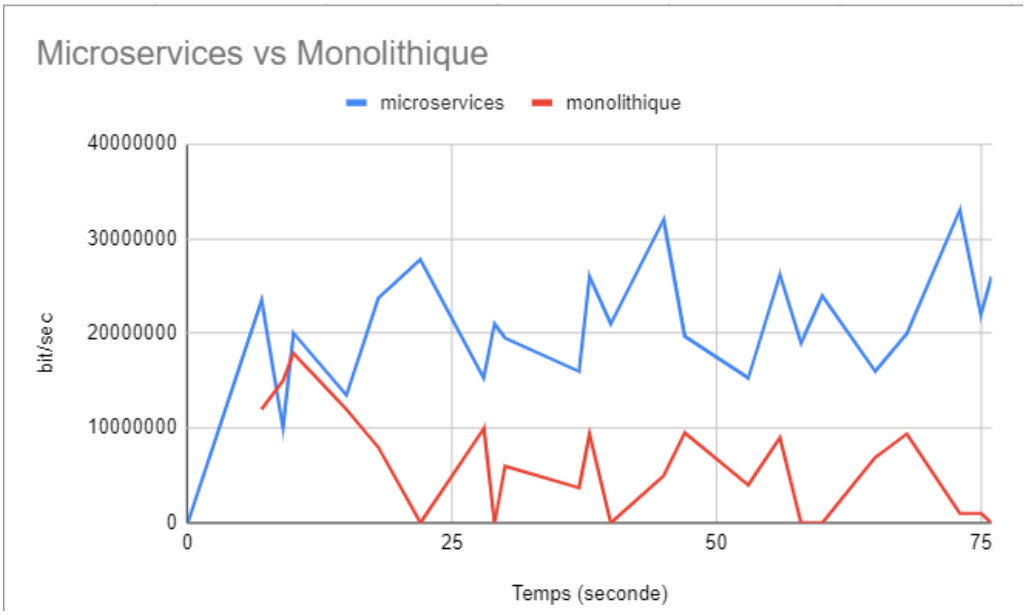

FIGURE 4.39 – c) Réseau social microservices vs réseau social monolithique

La figure 4.36 représente le Débit en fonction du temps pour 100 utilisateurs. La figure 4.37 concerne la version basée sur l'architecture microservices, tandis que la figure 4.38 porte sur la version monolithique. La figure 4.39 résume les informations des figures 4.37 et 4.38 pour une meilleure lisibilité.

Les figures démontrent que les microservices maintiennent des niveaux de débit élevés, atteignant jusqu'à 33 000 000 octets/seconde, ils présentent également des fluctuations

marquées. En revanche, l'architecture monolithique affiche un débit initial légèrement inférieur, avec des périodes d'inactivité (nul).

Ces données confirment que, pour cette charge utilisateur, l'architecture microservices offre des performances de débit supérieures, tout en étant plus scalable que l'architecture monolithique.

FIGURE 4.40 – Temps de réponse par rapport aux threads

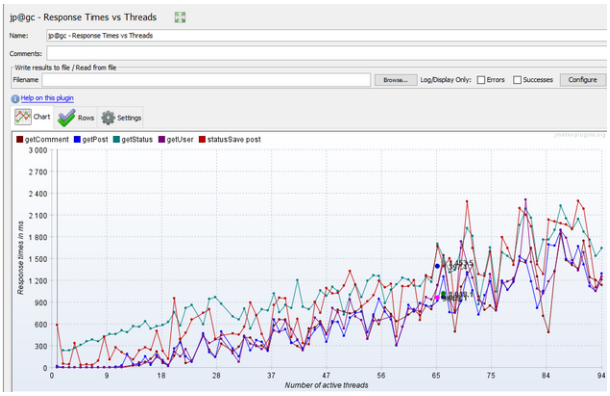

FIGURE 4.41 – (a) Réseau social microservices

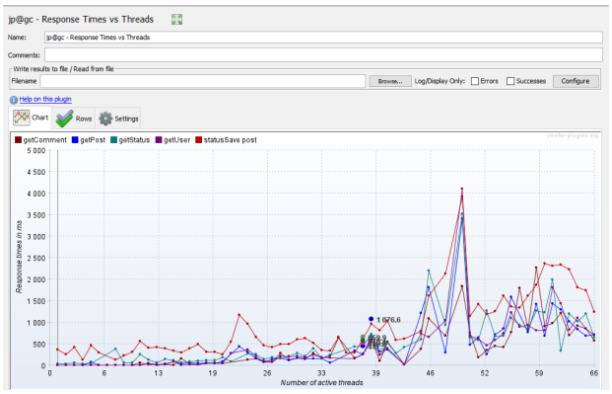

FIGURE 4.42 – (b) Réseau social monolithique

La figure 4.40 représente le temps de réponse par rapport aux threads en fonction du temps pour 100 utilisateurs. La figure 4.41 concerne la version basée sur l'architecture microservices, tandis que la figure 4.42 porte sur la version monolithique. La figure 4.43 résume les informations des figures 4.41 et 4.42 pour une meilleure lisibilité.

Les figures montrent que l'architecture microservices présente une gestion plus efficace de la charge. Les temps de réponse augmentent progressivement, atteignant un pic de 1920 ms, mais restent généralement stables. En revanche, l'architecture monolithique connaît une augmentation abrupte avec un pic remarquable de 3360 ms pour seulement 49 threads actifs.

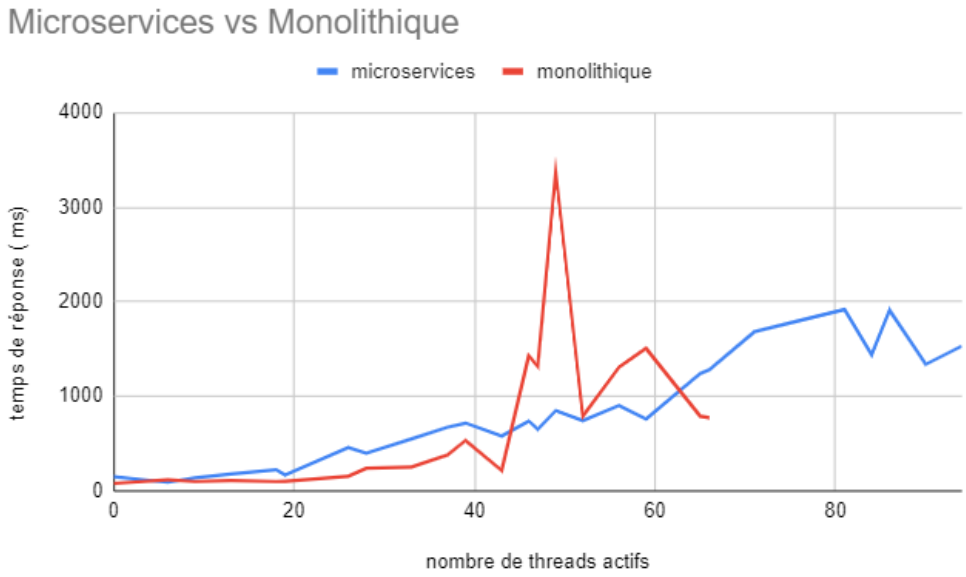

FIGURE 4.43 – c) Réseau social microservices vs réseau social monolithique

En conclusion, pour cette configuration, l'architecture microservices semble être la meilleure option en termes de temps de réponse par rapport au nombre de threads.

4.5.3 Résultats des Tests

Après une analyse minutieuse des résultats des tests pour différentes charges utilisateur, y compris les scénarios de "Temps de réponse en fonction du temps", de "Débit en fonction du temps" et de "Temps de réponse par rapport aux threads" avec 50 et 100 utilisateurs, nous avons conclu que les performances des deux architectures varient. Pour une charge de 50 utilisateurs, l'architecture monolithique se distingue par des temps de réponse plus courts et un meilleur débit. En revanche, pour une charge de 100 utilisateurs, les microservices affichent de meilleures performances en termes de temps de réponse, de débit et de stabilité, tandis que l'architecture monolithique présente moins de performances, en plus de fluctuations au fil du temps et de périodes d'inactivité. Cela montre la scalabilite des architectures micoservices.

4.6 Conclusion

Ce chapitre a porté sur le développement et la mise en œuvre de deux versions de notre réseau social : une basée sur l'architecture monolithique et l'autre sur l'architecture

microservices. Les tests de comparaison ont été exécutés à l'aide de JMeter, simulant des charges de 50 et 100 utilisateurs. Une répartition équitable des requêtes a été mise en place pour interroger les divers services des applications.

Les résultats des tests ont révélé des performances variables des deux architectures. Dans l'ensemble, Pour une charge de 50 utilisateurs, l'architecture monolithique a montré des temps de réponse plus courts et un meilleur débit. En revanche, pour une charge de 100 utilisateurs, les microservices affichent de meilleures performances en termes de temps de réponse, de débit et de stabilité, tandis que l'architecture monolithique présente moins de performances, en plus de fluctuations au fil du temps et de périodes d'inactivité.

Ces résultats soulignent la nécessité de prendre en considération les besoins spécifiques de l'application lors du choix entre une architecture monolithique et une architecture microservices. Les microservices se révèlent plus adaptés à une scalabilité efficace, tandis que le monolithique peut être meilleur dans d'autres situations. En fin de compte, le choix de l'architecture dépendra des exigences et des objectifs de l'application à développer.

Conclusion générale et perspectives

Dans cette thèse de Master, nous avons étudié les architectures logicielles, à savoir l'architecture monolithique et celle basée sur les microservices.

L'architecture monolithique est le modèle classique où toutes les fonctionnalités de l'application sont encapsulées dans un seul code. Bien que ce modèle présente de nombreux avantages, les applications monolithiques peuvent rencontrer des difficultés à mesure que l'application devient plus complexe et que l'équipe de développement s'élargit, cela a conduit à l'émergence de l'architecture microservices.

Ce dernier type l'architecture, est basée sur l'idée de diviser l'application en un ensemble de petits services faiblement couplées plutôt que de créer une seule application monolithique. Cela permet une meilleure scalabilté et facilite le déploiement indépendant de chaque service.

Plusieurs recherches récentes comparent les architectures monolithiques et les architectures microservices. Nous avons analysé ces recherches et exposé les principaux critères de comparaison entre les deux modèles architecturales.

Dans le cadre de ce travail, nous avons choisi de développer un réseau social afin d'effectuer nos tests de comparaison. Cette décision découle de l'importance des réseaux sociaux dans notre vie quotidienne, tant sur le plan personnel que professionnel. Ce type d'application propose des plateformes qui facilitent la communication, le partage d'informations, l'interaction avec d'autres utilisateurs et la création de communautés en ligne.

Le réseau social a été développé en deux versions, une version monolithique et une autre basée sur les microservices.

Nous avons soumis les deux applications à une série de tests exhaustifs afin d'évaluer leurs performances. Nous avons couvert différents aspects, tels que les temps de réponse, le débit, la scalabilité et la stabilité sous différentes charges.

Nos analyses des résultats des tests de performance ont révélé des différences de performance entre les deux architectures. À une charge de 50 utilisateurs, l'architecture monolithique se caractérise par des temps de réponse plus courts et un débit supérieur. Cependant, lorsque la charge augmente à 100 utilisateurs, les microservices affichent de meilleures performances en termes de temps de réponse, de débit et de stabilité, tandis que l'architecture monolithique présente des différences de performance, accompagnées de fluctuations temporelles et de périodes d'inactivité. Ces conclusions mettent en évidence la capacité d'adaptation des architectures microservices. Il est essentiel de noter que le choix entre une architecture microservices et une architecture monolithique ne se limite pas à une décision binaire. Chacune de ces approches comporte ses avantages et inconvénients, et le choix de l'architecture la mieux adaptée dépendra des caractéristiques spécifiques du projet.

En ce qui concerne les futures directions de ce travail, nous suggérons : i) d'étendre nos tests à des serveurs plus puissants, et ii) d'envisager l'utilisation d'un benchmark dédié aux microservices afin de comparer les deux applications développées.

Bibliographie

[1] https ://dictionnaire.lerobert.com/definition/monolithique. Accessed : [10-03-2023].

[2] P. Francisco, M. Gaston, and A. Hernan. Migrating from monolithic architecture to microservices : A rapid review. In *IEEE Conference Proceedings*, page 17, 2019.

[3] C. Richardson. *Microservices Patterns : With examples in Java*. Manning, 2018.

[4] https ://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59. Accessed : [12-03-2023].

[5] https ://microservices.io/patterns/monolithic.html. Accessed : [04-04-2023].

[6] J. Hurwitz, C. Baroudi, R. Bloor, and M. Kaufman. *what is soa*, pages 5–9. 2006.

[7] A. Goel. Enterprise integration eai vs. soa vs. esb. White Paper 87, Infosys Technologies, 2006.

[8] J. Ghofrani and D. Lübke. Challenges of microservices architecture : A survey on the state of the practice. In *ZEUS*, pages 1–8, 2018.

[9] https ://www.youtube.com/watch ?v=yygitxf3hla. Accessed : [29-04-2023].

[10] https ://www.atlassian.com/fr/microservices/microservices-architecture/microservices-vs-monolith. Accessed : [29-04-2023].

[11] J. T. Zhao, S. Y. Jing, and L. Z. Jiang. Management of api gateway based on microservice architecture. In *Journal of Physics : Conference Series*, volume 1087, page 032032, 2018, September.

[12] E. Caron and Z. Houmani. Architecture microservices pilotée par les données. In *CAL 2018-Conférence francophone sur les Architectures Logicielles*, pages 1–6, 2018, June.

[13] F. Montesi and J. Weber. Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv :1609.05830*, 2016.

[14] A. Balalaie, A. Heydarnoori, and P. Jamshidi. Migrating to cloud-native architectures using microservices : an experience report. In *Advances in Service-Oriented and Cloud Computing : Workshops of ESOCC 2015*, volume 4, pages 201–215. Springer International Publishing, 2016.

[15] D. Taibi, V. Lenarduzzi, and C. Pahl. Architectural patterns for microservices : a systematic mapping study. In *CLOSER 2018 : Proceedings of the 8th International Conference on Cloud Computing and Services Science*, pages 79–92, 2018.

[16] S. Newman. *Monolith to Microservices : Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, 2019.

[17] https ://programmerfriend.com/monolith-vs-microservices/. Accessed : [21-05-2023].

[18] https ://aws.amazon.com/fr/microservices/. Accessed : [21-05-2023].

[19] G. Liu, B. Huang, Z. Liang, M. Qin, H. Zhou, and Z. Li. Microservices : architecture, container, and challenges. In *2020 IEEE 20th international conference on software quality, reliability and security companion (QRS-C)*, pages 629–635. IEEE, 2020, December.

[20] O. Al-Debagy and P. Martinek. A comparative review of microservices and monolithic architectures. In *2018 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2018, November.

[21] C. Auer and B. Weber. Monolithic or microservices ? the question of choice for modernizing legacy systems. *IEEE Software*, 38(2) :76–83, 2021.

[22] F. Tapia, M. Á. Mora, W. Fuertes, H. Aules, E. Flores, and T. Toulkeridis. From monolithic systems to microservices : A comparative study of performance. *Applied sciences*, 10(17) :5797, 2020.

[23] R. Flygare and A. Holmqvist. Performance characteristics between monolithic and microservice-based systems. *arXiv preprint arXiv :1708.06785*, 2017.

[24] K. Gos and W. Zabierowski. The comparison of microservice and monolithic architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*, pages 150–153. IEEE, 2020.

[25] A. Saransig and F. Tapia. Performance analysis of monolithic and micro service architectures–containers technology. *In Trends and Applications in Software Engineering : Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*, pages 270–279, 2019.

[26] G. Blinowski, A. Ojdowska, and A. Przybyłek. Monolithic vs. microservice architecture : A performance and scalability evaluation. *IEEE Access*, 10 :20357–20374, 2022.

[27] Andrea Detti, Luca Funari, and Laure Petrucci. bench : an open-source factory of benchmark microservice applications. *IEEE Transactions on Parallel and Distributed Systems*, 2023.

[28] M. Forse. Définir et analyser les réseaux sociaux : les enjeux de l'analyse structurale. *Informations sociales*, (3) :10–19, 2008.

[29] H. Badis. Contribution aux performances des réseaux de capteurs sans fil, 2021.

[30] https ://www.futura-sciences.com/tech/definitions/informatique-reseau-social-10255/. Accessed : [10-06-2023].

[31] H. Benhacine and L. Hammadi. Les réseaux sociaux : outils d'optimisation des stratégies de recrutement. In *Proceedings of the International Conference on Recruitment Strategies*, s.d.

[32] H. Diouani, A. Graa, and M. Chahidi. Les médias sociaux et leur utilisation dans les établissements hôteliers : Une approche qualitative. In *Proceedings of the International Conference on Social Media in Hospitality*, s.d.

[33] B. Žáková. Le français dans les réseaux sociaux. *Journal of Social Linguistics*, 2022.

[34] https ://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Accessed : [10-06-2023].

[35] I. Jacobson, G. Booch, and J. Rumbaugh. The unified process. *IEEE Software*, 16(3) :96, 1999.

[36] https ://code.visualstudio.com/docs. Accessed : [13-07-2023].

[37] https ://spring.io/tools. Accessed : [13-07-2023].

[38] https ://developer.mozilla.org/fr/docs/web/html. Accessed : [13-07-2023].

[39] https ://glossaire.infowebmaster.fr/css/. Accessed : [13-07-2023].

[40] https ://developer.mozilla.org/fr/docs/learn/javascript/first_steps/what_is_javascript. Accessed : [13-07-2023].

[41] https ://fr.legacy.reactjs.org/docs/getting-started.html. Accessed : [13-07-2023].

[42] https ://www.lemagit.fr/definition/java. Accessed : [13-07-2023].

[43] https ://azure.microsoft.com/fr-fr/resources/cloud-computing-dictionary/what-is-java-spring-boot. Accessed : [13-07-2023].

[44] https ://fr.javascript.info/websocket. Accessed : [13-07-2023].

[45] https ://firebase.google.com/. Accessed : [13-07-2023].

[46] https ://jmeter.apache.org/. Accessed : [01-08-2023].