ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

22/12/2025

Gestion distribuée Produits & Commandes

J2EE

Réaliser par :

- EL KHAYATI Mohamed
- MOUDID Mouad

5IIR11

Remerciements

Nous tenons tout d’abord à exprimer notre sincère gratitude à notre professeur M. HASBI pour son encadrement, ses précieux conseils et son accompagnement tout au long de ce projet. Son expertise et sa pédagogie ont été déterminantes pour la réussite de cette réalisation.

Nous remercions également l’établissement et l’équipe pédagogique pour nous avoir offert l’opportunité de travailler sur un projet concret intégrant des technologies modernes et des architectures distribuées.

Enfin, nous nous témoignons mutuellement, Mouad MOUDID et Mohamed EL KHAYATI, notre reconnaissance pour le travail d’équipe, la collaboration et l’engagement dont nous avons fait preuve durant tout le développement de ce projet.

Résumé

Ce projet consiste en la conception et le développement d'une application de gestion de produits et de commandes basée sur une architecture microservices avec Spring Cloud et un frontend moderne en Next.js. L'objectif principal était de mettre en pratique les principes des systèmes distribués : scalabilité, résilience, découverte de services et gestion centralisée de la configuration.

Le backend est structuré autour de plusieurs services indépendants : un serveur de configuration (Config Server), un serveur de découverte (Eureka Server), une API Gateway pour le routage, et deux microservices métier (Produits et Commandes). Le frontend, développé avec Next.js et TypeScript, offre une interface utilisateur réactive et intuitive permettant de gérer les produits et les commandes de manière fluide.

Les fonctionnalités clés incluent la gestion CRUD des produits et commandes, l'intégrité référentielle entre les entités, la tolérance aux pannes via Resilience4J, ainsi qu'une documentation API automatique avec SpringDoc OpenAPI. L'ensemble démontre une maîtrise technique des écosystèmes Java/Spring et React/Next.js dans un contexte professionnel.

Abstract

This project involves the design and development of a product and order management application based on a microservices architecture using Spring Cloud and a modern Next.js frontend. The main objective was to apply distributed systems principles: scalability, resilience, service discovery, and centralized configuration management.

The backend is structured around several independent services: a configuration server (Config Server), a discovery server (Eureka Server), an API Gateway for routing, and two business microservices (Products and Orders). The frontend, developed with Next.js and TypeScript, provides a responsive and intuitive user interface for seamless management of products and orders.

Key features include CRUD operations for products and orders, referential integrity between entities, fault tolerance via Resilience4J, and automatic API documentation with SpringDoc OpenAPI. The project demonstrates technical mastery of the Java/Spring and React/Next.js ecosystems in a professional context.

Table des figures

Table de matière

Introduction Générale

Au cours des dernières années, les systèmes d'information ont connu une évolution majeure, portée par la transformation numérique, l'augmentation du volume des données et la nécessité de fournir des applications toujours plus performantes, disponibles et évolutives. Les architectures logicielles traditionnelles, basées sur des applications monolithiques, montrent aujourd'hui leurs limites face à ces nouvelles exigences, notamment en termes de maintenance, de scalabilité et de tolérance aux pannes.

Dans ce contexte, les architectures microservices se sont imposées comme une alternative moderne et efficace. Elles reposent sur le découpage d'une application en plusieurs services indépendants, chacun responsable d'une fonctionnalité bien définie et communiquant avec les autres services via des interfaces normalisées, généralement des API REST. Cette approche permet une meilleure modularité, un déploiement indépendant des services et une résilience accrue du système global.

Le framework Spring Boot, associé à Spring Cloud, offre un ensemble d'outils puissants facilitant la mise en œuvre des architectures microservices. Il permet notamment la gestion centralisée des configurations, la découverte automatique des services, la mise en place d'une passerelle d'API (API Gateway) et l'intégration de mécanismes de tolérance aux pannes tels que les circuit breakers. Ces fonctionnalités sont devenues indispensables dans le développement d'applications distribuées modernes.

Par ailleurs, le développement d'interfaces utilisateurs a également évolué vers des solutions plus dynamiques et performantes. Les frameworks frontend modernes comme React et Next.js permettent de concevoir des interfaces riches, responsives et optimisées, offrant une expérience utilisateur améliorée. L'intégration d'un frontend moderne avec une architecture backend distribuée constitue aujourd'hui un enjeu majeur dans la conception d'applications complètes et professionnelles.

C'est dans cette optique que s'inscrit le présent projet, réalisé dans le cadre du module J2EE. Il consiste à concevoir et développer une application de gestion de produits et de commandes, basée sur une architecture microservices utilisant Spring Cloud pour le backend et Next.js pour le frontend. Le projet vise à mettre en pratique les concepts théoriques étudiés, tout en respectant les bonnes pratiques de conception, de développement et de documentation des applications distribuées.

L'objectif principal de ce travail est de proposer une architecture scalable, maintenable et résiliente, capable de gérer efficacement les interactions entre plusieurs services indépendants. Plus spécifiquement, il s'agit de mettre en œuvre un serveur de configuration centralisée, un service de découverte, une API Gateway, ainsi que des mécanismes de tolérance aux pannes. En parallèle, une interface utilisateur moderne est développée afin de permettre une interaction fluide avec le système.

Ce rapport présente l'ensemble des étapes de réalisation du projet, depuis l'étude du contexte et de l'état de l'art, jusqu'à la conception, l'implémentation et l'évaluation de la solution proposée. Il est structuré en quatre chapitres principaux. Le premier chapitre aborde le contexte

général du projet et les objectifs visés. Le deuxième chapitre est consacré à l'état de l'art, présentant les concepts et technologies liés aux architectures microservices et aux frameworks utilisés. Le troisième chapitre traite la conception et la modélisation de l'application, tandis que le quatrième chapitre décrit en détail la réalisation et l'implémentation de la solution développée. Enfin, une conclusion générale vient synthétiser les résultats obtenus et proposer des perspectives d'évolution.

Chapitre 1 : Contexte général du projet

1 Contexte académique et pédagogique

Ce projet s'inscrit dans le cadre du module J2EE, dispensé au sein de la filière 5IIR-11, durant l'année universitaire 2025–2026. Il a pour objectif principal de permettre aux étudiants de mettre en pratique les concepts théoriques étudiés en cours, notamment ceux liés au développement d'applications d'entreprise, aux architectures distribuées et aux bonnes pratiques de conception logicielle.

À travers ce projet, nous avons été amenés à concevoir une application complète, allant du backend au frontend, en utilisant des technologies largement adoptées dans le monde professionnel. Cette approche pédagogique vise à renforcer nos compétences techniques, notre capacité d'analyse ainsi que notre autonomie dans la réalisation de projets informatiques complexes.

2 Évolution des architectures logicielles

Les architectures logicielles ont connu une évolution significative au cours des dernières décennies. Les premières applications étaient majoritairement basées sur des architectures monolithiques, dans lesquelles l'ensemble des fonctionnalités était regroupé au sein d'une seule et même application. Bien que simples à développer au départ, ces architectures deviennent rapidement difficiles à maintenir, à faire évoluer et à déployer lorsque la taille de l'application augmente.

Avec l'essor des systèmes distribués et des besoins croissants en scalabilité et en disponibilité, les architectures microservices ont émergé comme une solution adaptée. Elles reposent sur le découpage de l'application en plusieurs services indépendants, chacun responsable d'un domaine fonctionnel précis. Cette approche facilite la maintenance, permet des déploiements indépendants et améliore la résilience globale du système.

3 Problématique du projet

La mise en œuvre d'une architecture microservices soulève plusieurs défis techniques et organisationnels. Parmi les principales problématiques rencontrées, on peut citer :

- La gestion centralisée des configurations des différents services
- La découverte dynamique des services disponibles
- La communication fiable entre les microservices
- La gestion des pannes et la tolérance aux défaillances
- L'intégration d'une interface utilisateur moderne avec un backend distribué

La problématique centrale de ce projet peut donc être formulée comme suit :

Comment concevoir et implémenter une architecture microservices fiable, évolutive et résiliente, intégrant un backend distribué et un frontend moderne, tout en assurant une communication fluide entre les différents composants du système ?

4 Objectifs du projet

Afin de répondre à cette problématique, plusieurs objectifs ont été définis.

Objectif général

L'objectif principal du projet est de concevoir et développer une application de gestion de produits et de commandes basée sur une architecture microservices, en utilisant Spring Boot, Spring Cloud et un frontend moderne développé avec Next.js.

Objectifs spécifiques

- Mettre en place une architecture microservices conforme aux bonnes pratiques
- Implémenter un serveur de configuration centralisée (Config Server)
- Mettre en œuvre un service de découverte (Eureka Server)
- Déployer une API Gateway comme point d'entrée unique
- Assurer la résilience du système à l'aide de mécanismes de circuit breaker
- Développer une interface utilisateur moderne, intuitive et responsive
- Documenter automatiquement les API REST
- Tester et valider le bon fonctionnement de l'ensemble du système

5 Périmètre et limites du projet

Le périmètre fonctionnel du projet se limite à la gestion de deux entités principales :

- Les produits
- Les commandes

Chaque fonctionnalité est implémentée sous forme de microservice indépendant. Le projet est développé dans un environnement académique, ce qui implique certaines limites, notamment :

- L'utilisation d'une base de données en mémoire (H2)
- L'absence de mécanismes avancés de sécurité (authentification et autorisation)
- Un déploiement réalisé localement, sans infrastructure cloud réelle

Malgré ces limites, l'architecture mise en place constitue une base solide et extensible, pouvant être facilement enrichie et adaptée à un contexte de production.

6 Méthodologie adoptée

Pour la réalisation de ce projet, nous avons adopté une méthodologie progressive, basée sur les étapes suivantes :

1. Étude des concepts liés aux architectures microservices
2. Analyse des besoins fonctionnels et techniques
3. Conception de l'architecture globale du système
4. Implémentation des microservices et des composants Spring Cloud
5. Développement du frontend avec Next.js
6. Tests et validation de l'application

Cette méthodologie nous a permis de structurer efficacement notre travail et d’assurer la cohérence entre les différentes phases du projet.

Chapitre 2 : État de l'art

1 Architectures monolithiques

Les architectures monolithiques constituent l'un des premiers modèles utilisés pour le développement d'applications logicielles. Dans ce type d'architecture, l'ensemble des fonctionnalités de l'application (interface utilisateur, logique métier et accès aux données) est regroupé au sein d'une seule et même application. Ce modèle présente l'avantage d'être simple à concevoir, à déployer et à tester dans les premières phases de développement.

Cependant, à mesure que l'application évolue, l'architecture monolithique montre plusieurs limites. La maintenance devient complexe, les mises à jour nécessitent souvent le redéploiement complet de l'application et la scalabilité est difficile à gérer. De plus, une défaillance dans un seul composant peut impacter l'ensemble du système, ce qui réduit la disponibilité globale de l'application.

2 Architectures microservices

Face aux limites des architectures monolithiques, les architectures microservices ont émergé comme une solution moderne et flexible. Elles reposent sur le découpage de l'application en plusieurs services indépendants, chacun étant responsable d'une fonctionnalité spécifique. Chaque microservice est autonome, dispose de sa propre base de données et communique avec les autres services via des protocoles légers, généralement HTTP/REST.

Les principaux avantages des microservices sont la scalabilité horizontale, la facilité de maintenance, la possibilité de déployer chaque service indépendamment et une meilleure tolérance aux pannes. Toutefois, cette architecture introduit également de nouveaux défis, tels que la gestion de la communication inter-services, la supervision, la sécurité et la gestion des configurations distribuées.

3 Spring Boot

Spring Boot est un framework Java largement utilisé pour le développement d'applications d'entreprise. Il simplifie considérablement la création d'applications Spring grâce à une configuration automatique et à une approche opinionated. Spring Boot permet de développer rapidement des microservices autonomes, embarquant leur propre serveur applicatif.

Grâce à son écosystème riche et à son intégration native avec d'autres projets Spring, Spring Boot constitue une base solide pour la mise en œuvre d'architectures microservices performantes et maintenables.

4 Spring Cloud

Spring Cloud est une suite de projets fournissant des solutions aux problématiques courantes des systèmes distribués. Il offre des outils permettant de gérer la configuration centralisée, la découverte de services, le routage des requêtes et la tolérance aux pannes.

Dans le cadre de ce projet, Spring Cloud joue un rôle central en facilitant l'orchestration des différents microservices et en assurant une communication fiable et dynamique entre eux.

5 Configuration centralisée avec Spring Cloud Config

La gestion des configurations constitue un enjeu majeur dans les architectures microservices. Spring Cloud Config permet de centraliser les fichiers de configuration dans un dépôt distant, généralement un dépôt Git. Chaque microservice peut ainsi récupérer dynamiquement sa configuration au démarrage ou lors d'un rafraîchissement.

Cette approche garantit la cohérence des configurations, facilite leur mise à jour et réduit les risques d'erreurs liés à la duplication des fichiers de configuration.

6 Découverte des services avec Eureka

Dans une architecture microservices, les services peuvent être dynamiquement ajoutés ou supprimés. Le mécanisme de découverte de services est donc essentiel. Eureka Server, fourni par Spring Cloud Netflix, permet aux microservices de s'enregistrer automatiquement et de découvrir les autres services disponibles.

Grâce à Eureka, les adresses des services ne sont plus codées en dur, ce qui améliore la flexibilité et la résilience du système.

7 API Gateway

L'API Gateway constitue le point d'entrée unique de l'application. Elle permet de centraliser le routage des requêtes, la sécurité, la gestion des erreurs et le load balancing. Spring Cloud Gateway est une solution moderne et performante permettant de mettre en place une passerelle d'API basée sur des routes dynamiques.

Dans le cadre de ce projet, l'API Gateway joue un rôle essentiel en assurant la communication entre le frontend et les différents microservices backend.

8 Résilience et tolérance aux pannes

La résilience est un aspect fondamental des systèmes distribués. Les pannes étant inévitables, il est nécessaire de mettre en place des mécanismes permettant de limiter leur impact. Le pattern Circuit Breaker est l'un des mécanismes les plus utilisés pour gérer les défaillances des services.

Resilience4J est une bibliothèque légère qui permet d'implémenter des circuit breakers, des timeouts et des fallbacks. Elle contribue à améliorer la stabilité et la disponibilité globale de l'application.

9 Frontend moderne avec React et Next.js

Le développement des interfaces utilisateurs a évolué vers des frameworks JavaScript modernes permettant de créer des applications dynamiques et performantes. React est une bibliothèque populaire permettant de construire des interfaces basées sur des composants réutilisables.

Next.js, basé sur React, apporte des fonctionnalités avancées telles que le rendu côté serveur (SSR), le routage intégré et l'optimisation des performances. Son utilisation dans ce projet permet d'offrir une expérience utilisateur fluide et responsive, tout en facilitant l'intégration avec une architecture backend distribuée.

10 Synthèse et positionnement du projet

À travers cet état de l'art, nous avons présenté les concepts et technologies clés liés aux architectures microservices et au développement frontend moderne. Le projet s'inscrit dans cette dynamique en combinant les outils fournis par Spring Cloud pour le backend et Next.js pour le frontend.

Cette approche permet de proposer une solution complète, évolutive et conforme aux standards actuels du développement d'applications distribuées, tout en répondant aux objectifs pédagogiques du module J2EE.

Chapitre 3 : Conception et modélisation de l'application

1 Analyse des besoins fonctionnels

La première étape de la conception consiste à analyser les besoins fonctionnels de l'application. Le système développé vise principalement la gestion de produits et de commandes au sein d'une architecture microservices. Les fonctionnalités principales identifiées sont les suivantes :

- Gestion des produits (création, consultation, mise à jour et suppression)
- Gestion des commandes avec référence aux produits
- Consultation des listes de produits et de commandes
- Calcul automatique des montants des commandes
- Affichage des données via une interface utilisateur moderne

Ces besoins constituent le socle fonctionnel de l'application et ont guidé les choix de conception adoptés.

2 Identification des acteurs

L'application comporte un acteur principal :

- Utilisateur : il interagit avec l'application via l'interface frontend afin de gérer les produits et les commandes.

Dans une version future, d'autres acteurs pourraient être ajoutés, tels qu'un administrateur ou un gestionnaire, avec des droits spécifiques.

3 Architecture globale de l'application

L'architecture globale de l'application repose sur le modèle microservices. Elle est composée des éléments suivants :

- Un Config Server pour la centralisation des configurations
- Un Eureka Server pour la découverte des services
- Une API Gateway servant de point d'entrée unique
- Un microservice Produits
- Un microservice Commandes
- Une application frontend Next.js

Cette architecture permet un découplage total entre les services et facilite leur évolution indépendante.

Architecture microservices Spring Cloud :

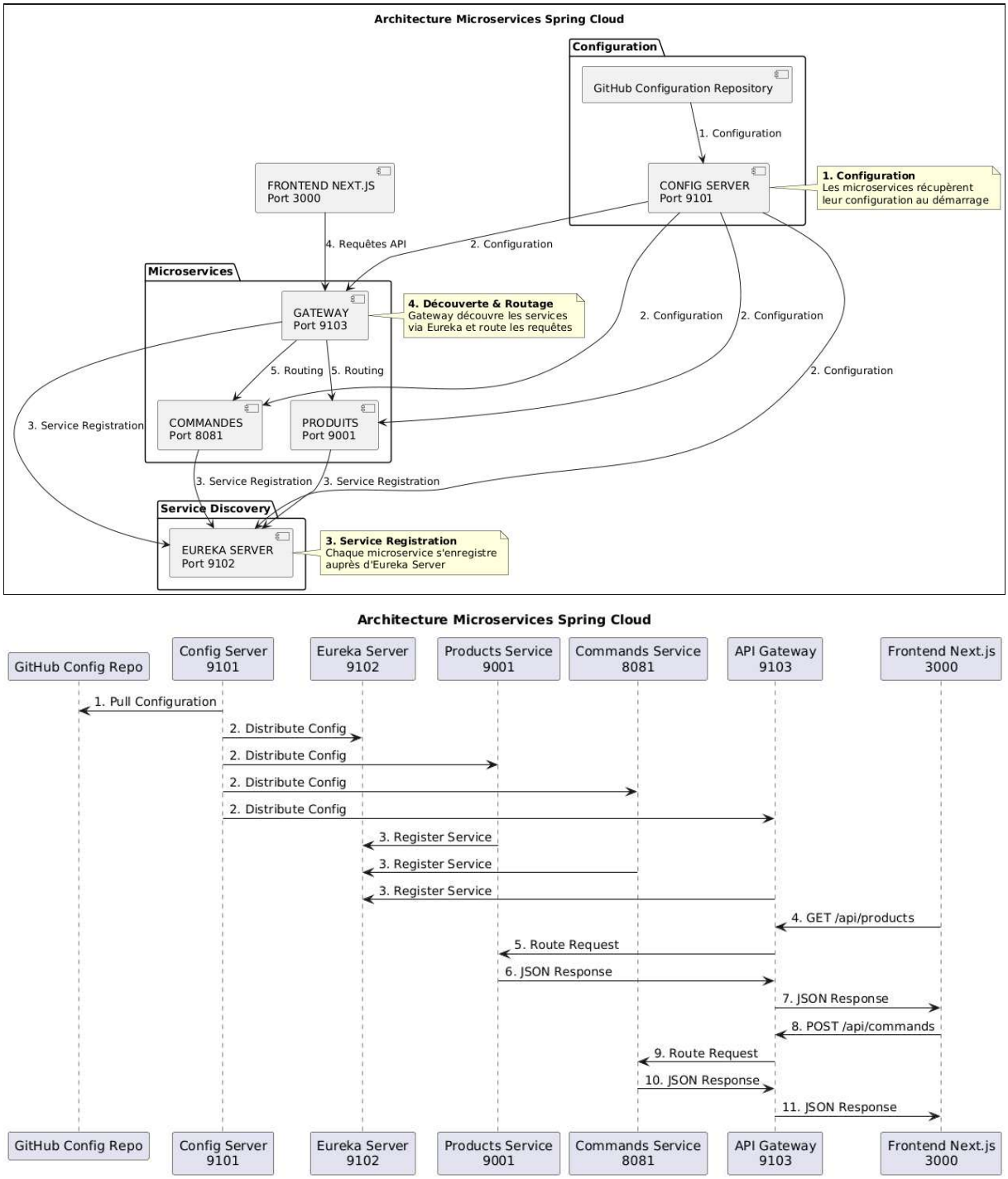

Figure 1:Architecture microservices Spring Cloud

4 Découpage en microservices

Le découpage fonctionnel a conduit à la création de deux microservices principaux :

4.1 Microservice Produits

Ce microservice est responsable de la gestion des produits. Il offre des fonctionnalités CRUD complètes et expose des API REST permettant :

- La création de produits
- La consultation des produits
- La mise à jour des informations
- La suppression de produits

4.2 Microservice Commandes

Ce microservice gère les commandes. Il est chargé de :

- Créer des commandes en référence à des produits
- Calculer le montant total des commandes
- Consulter l'historique des commandes
- Supprimer des commandes

Chaque microservice possède sa propre base de données, garantissant ainsi l'indépendance et l'autonomie des services.

5 Modélisation des données

5.1 Entité Produit

L'entité Produit est définie par les attributs suivants :

- Identifiant unique
- Nom du produit
- Prix
- Quantité disponible

5.2 Entité Commande

L'entité Commande est définie par :

- Identifiant unique
- Date de commande
- Montant total
- Identifiant du produit associé

La relation entre les entités est de type référence, une commande étant associée à un produit via son identifiant.

6 Conception des API REST

Les microservices exposent leurs fonctionnalités via des API REST respectant les principes RESTful. Les principales opérations sont basées sur les méthodes HTTP suivantes :

- GET : récupération des données
- POST : création de nouvelles ressources
- PUT : mise à jour des ressources existantes
- DELETE : suppression des ressources

Cette approche garantit une communication standardisée et interopérable entre les différents composants du système.

7 Diagrammes UML

7.1 Diagramme de classes

Le diagramme de classes met en évidence les entités principales du système ainsi que leurs attributs. Il permet de visualiser la structure des données et les relations existantes entre les entités Produit et Commande.

7.2 Diagramme de composants

Le diagramme de composants illustre l'architecture globale de l'application et les interactions entre les différents microservices, la Gateway, le Config Server et le frontend.

7.3 Diagramme de séquence

Le diagramme de séquence représente le déroulement des échanges lors de la création d'une commande, depuis l'appel effectué par le frontend jusqu'au traitement effectué par les microservices backend.

8 Sécurité et communication inter-services

Dans le cadre de ce projet, la communication inter-services est réalisée via des appels REST transitant par l'API Gateway. La sécurité n'a pas été implémentée dans cette version académique, mais l'architecture a été conçue de manière à permettre l'ajout ultérieur de mécanismes d'authentification et d'autorisation, tels que OAuth2 ou JWT.

9 Synthèse de la conception

La conception et la modélisation de l'application ont permis de définir une architecture claire, modulaire et extensible. Le découpage en microservices, la définition des entités et la conception des API REST constituent une base solide pour la phase de réalisation.

Ce travail de conception garantit la cohérence globale du système et facilite sa maintenance ainsi que son évolution future.

Chapitre 4 : Réalisation de l'application

1 Environnement de développement

La réalisation de l'application a été effectuée dans un environnement de développement adapté aux architectures distribuées. Les outils et technologies utilisés sont les suivants :

- Langage backend : Java 17

- Framework backend : Spring Boot et Spring Cloud
- Gestion des dépendances : Maven
- Base de données : H2 (en mémoire)
- Frontend : Next.js avec React et TypeScript
- Outils de développement : IntelliJ IDEA, Visual Studio Code
- Gestion de versions : Git et GitHub

Cet environnement nous a permis de développer, tester et valider efficacement chaque composant du système.

2 Interfaces de l'application

2.1 Architecture Technique du Système

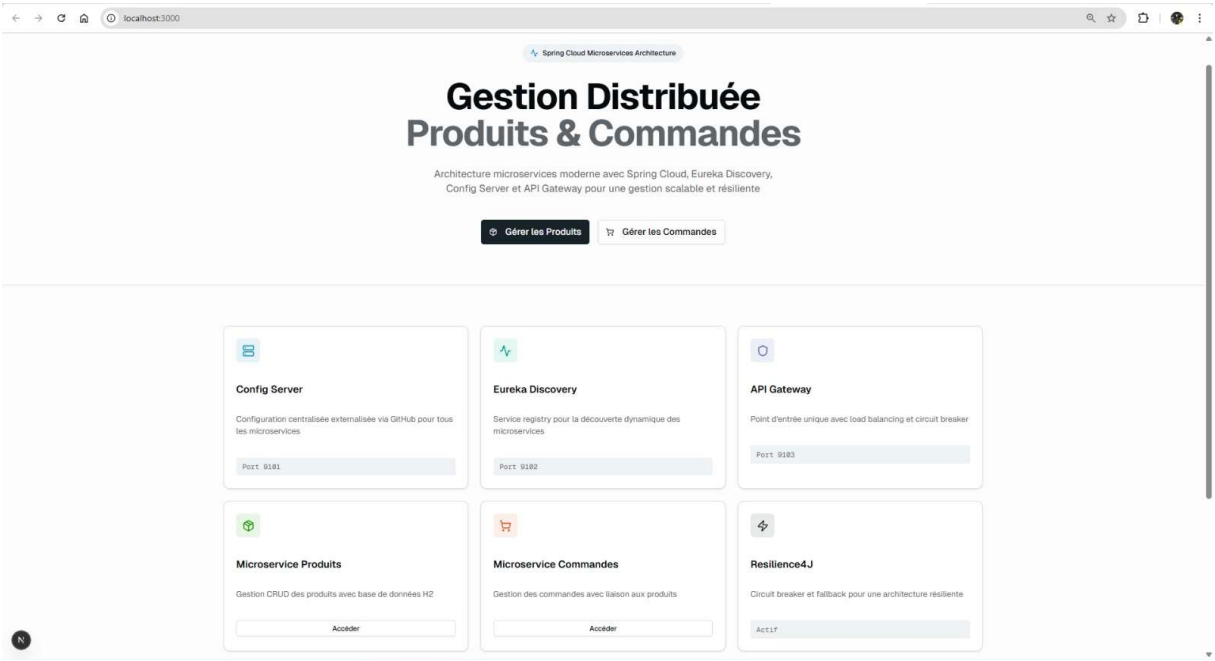

Figure 2:Schéma d'Architecture Microservices

Diagramme illustrant l'architecture distribuée complète du projet avec Spring Cloud. On y voit les différents services interconnectés : Config Server (port 9101) pour la configuration centralisée, Eureka Discovery Server (port 9102) pour la découverte des services, le Microservice Produits (port 9001) et le Microservice Commandes (port 8081), l'API Gateway (port 9103) comme point d'entrée unique, et l'application Frontend Next.js (port 3000). L'architecture démontre une mise en œuvre professionnelle des patterns microservices modernes.

2.2 Interface Vide de Gestion des Produits

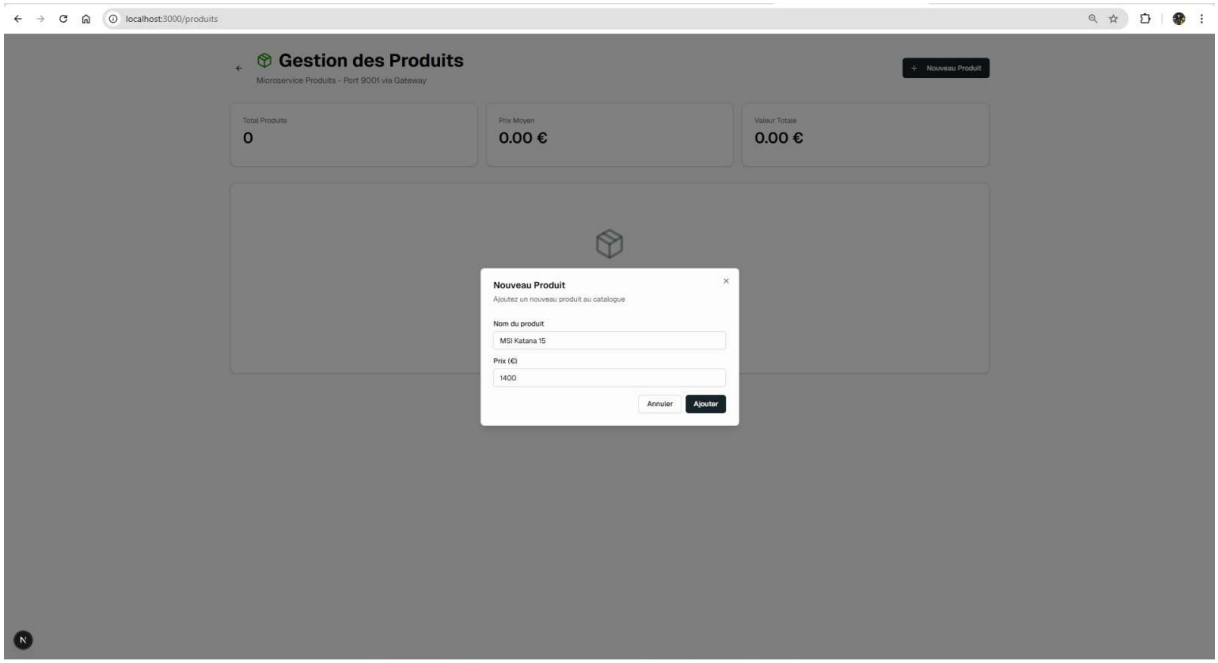

Figure 3:Tableau de Bord Produits (État Initial)

Capture de l'interface de gestion des produits avant l'ajout de données. L'écran montre un état vide avec 0 produit, 0.00 € de valeur totale et moyenne. Un formulaire de création de nouveau produit est visible avec les champs "Nom du produit" et "Prix". L'en-tête indique l'accès via le Gateway (port 8000). Cette image représente l'état initial du système.

2.3 Modification d'un Produit Existant

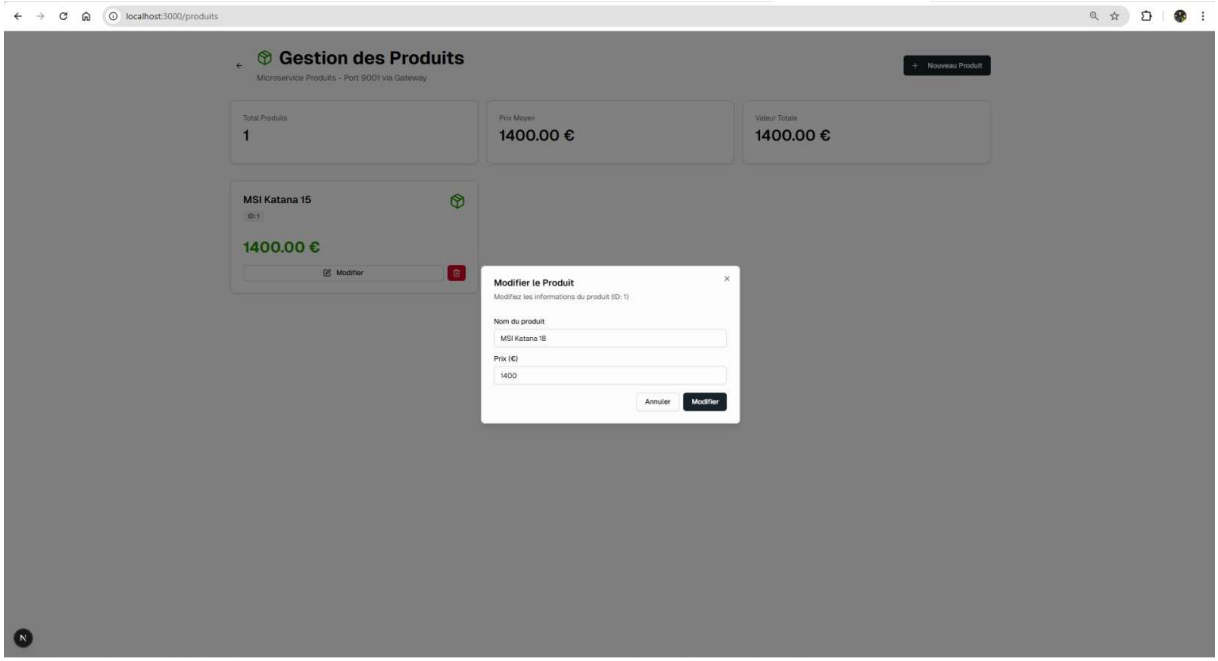

Figure 4:Écran d'Édition de Produit

Interface permettant la modification d'un produit existant. Le produit "MSI Katana 18" (ID: 1) est affiché avec son prix actuel de 1400.00 €. Un formulaire pré-rempli permet de modifier le nom et le prix du produit. Les boutons "Modifier" et "Annuler" sont disponibles. Cette capture démontre la fonctionnalité de mise à jour (Update) du CRUD.

2.4 Liste Complète des Produits

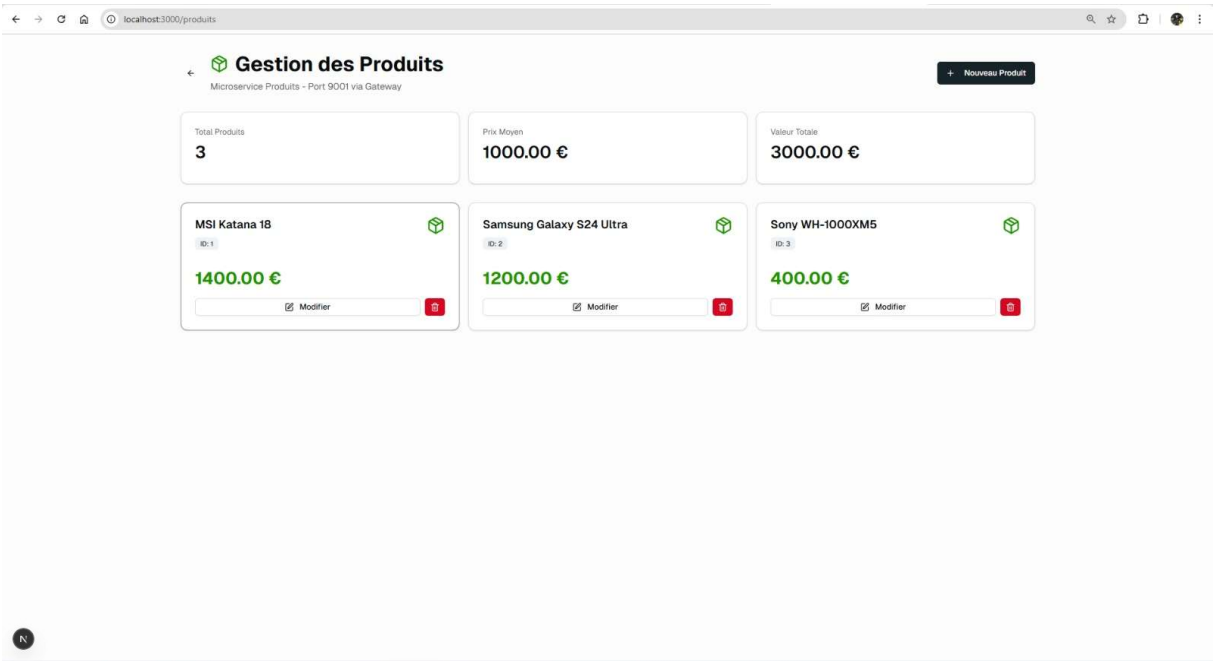

Figure 5:Catalogue des Produits avec Statistiques

Vue principale affichant la liste complète des trois produits : MSI Katana 18 (1400.00 €), Samsung Galaxy S24 Ultra (1200.00 €), et Sony WH-1000XM5 (400.00 €). Les statistiques sont calculées automatiquement : prix moyen (1000.00 €) et valeur totale (3000.00 €). Chaque produit dispose d'un bouton "Modifier" et un bouton "+ Nouveau Produit" permet d'ajouter d'autres articles.

2.5 Création d'une Nouvelle Commande

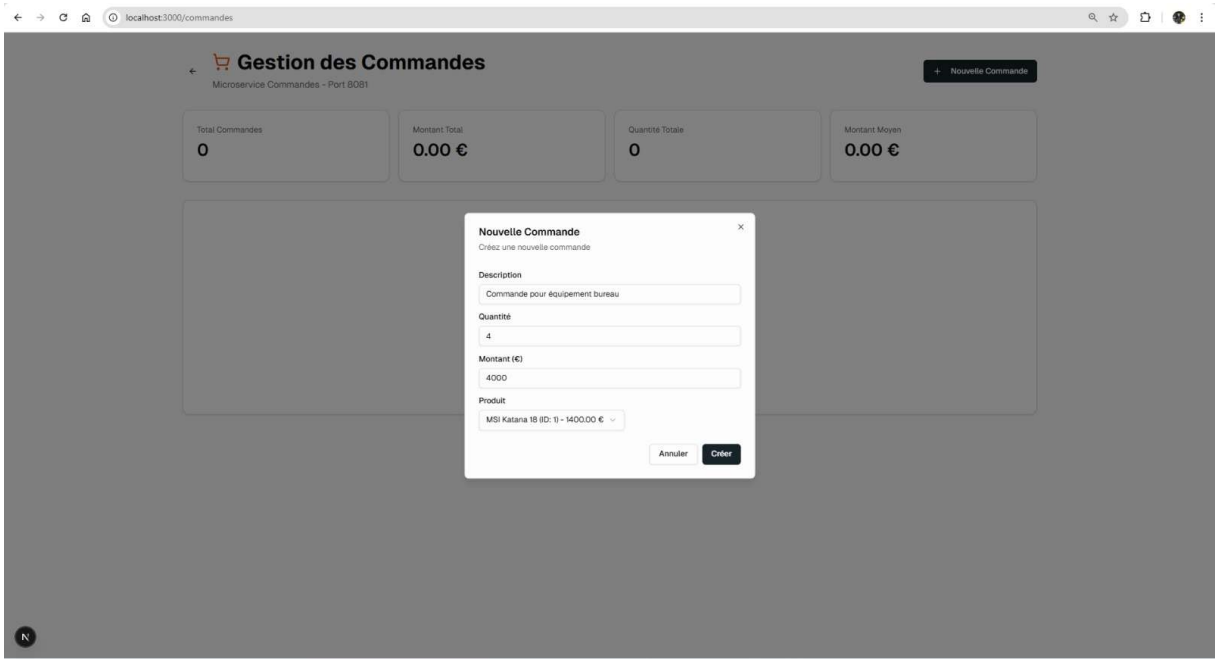

Figure 6:Formulaire de Création de Commande

Interface de création d'une nouvelle commande dans le microservice Commandes (port 8081). Le formulaire comprend les champs : Description, Quantité, Montant, et Produit (avec liste déroulante des produits existants). À ce stade, aucune commande n'existe encore (0 commande, 0.00 € total). Cette image montre l'intégration entre les microservices via la sélection de produits.

2.6 Tableau de Bord des Commandes

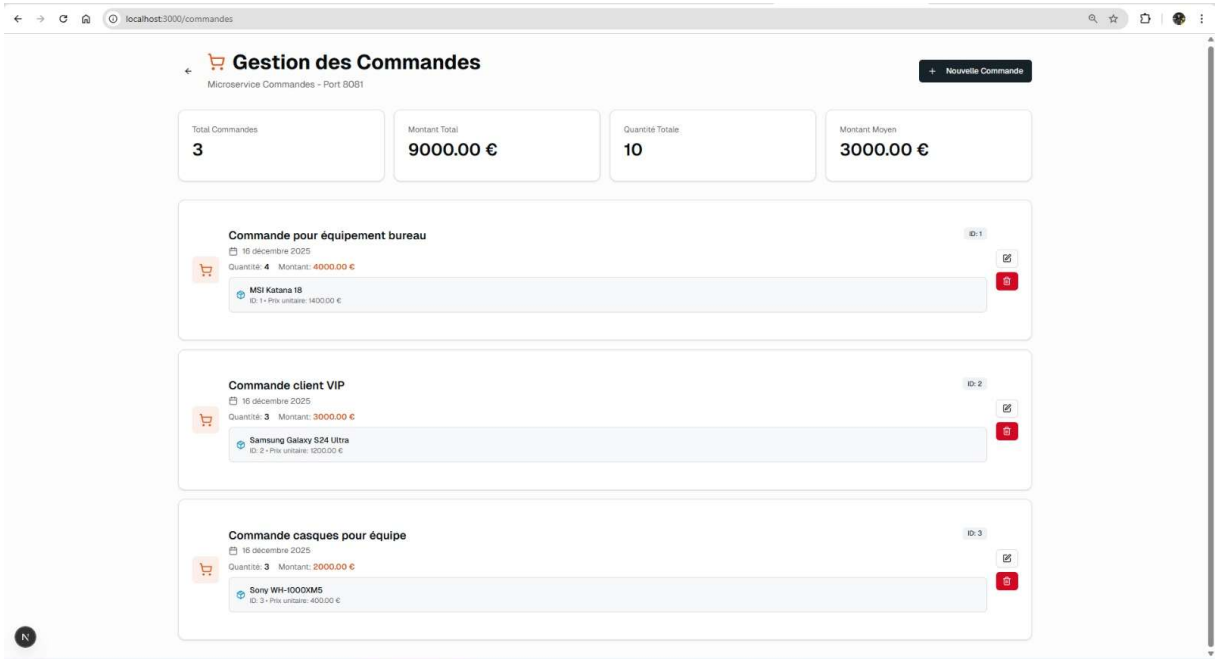

Figure 7:Visualisation des Commandes avec Métriques

Affichage des trois commandes créées, chacune liée à un produit spécifique. Les statistiques globales sont présentées : 3 commandes totales, montant total de 9000.00 €, quantité totale de 10 articles, et montant moyen de 3000.00 €. Cette capture illustre la relation entre produits et commandes ainsi que les calculs automatiques.

2.7 Confirmation de Suppression

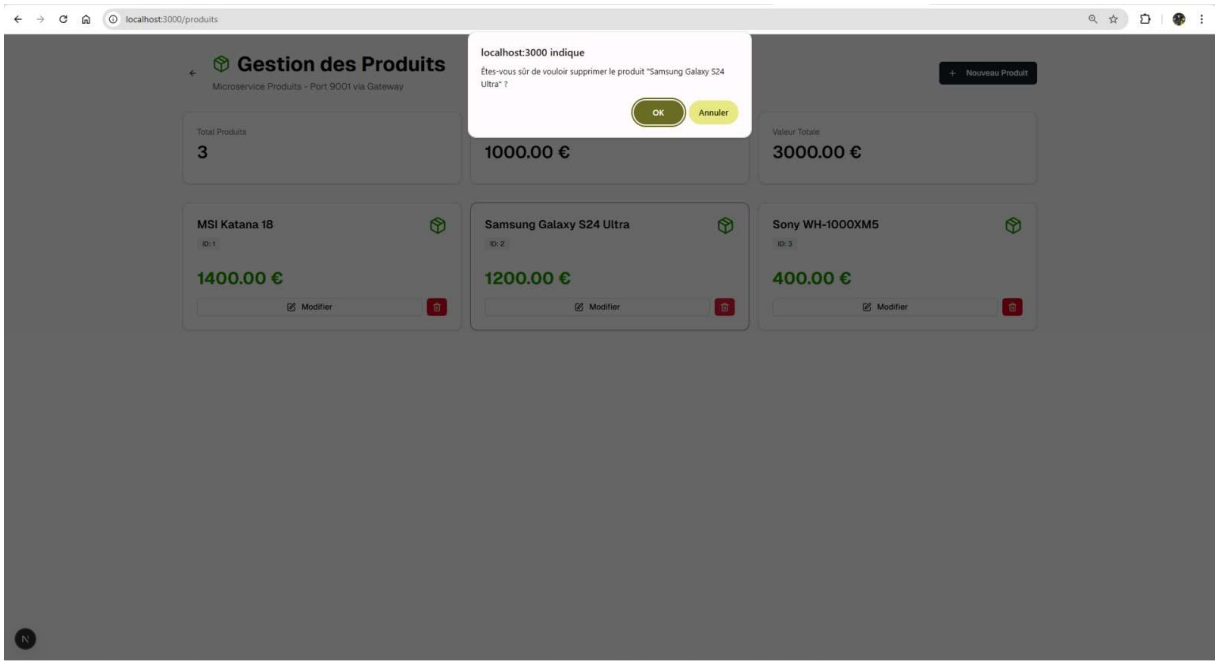

Figure 8:Dialogue de Confirmation Avant Suppression

Boîte de dialogue de confirmation apparaissant avant la suppression d'un produit. Le produit "Samsung Galaxy S24 Ultra" est sélectionné. Un message demande confirmation à l'utilisateur avec les boutons "OK" (confirmer) et "Annuler". Cette fonctionnalité prévient les suppressions accidentelles.

2.8 Contrainte d'Intégrité Référentielle

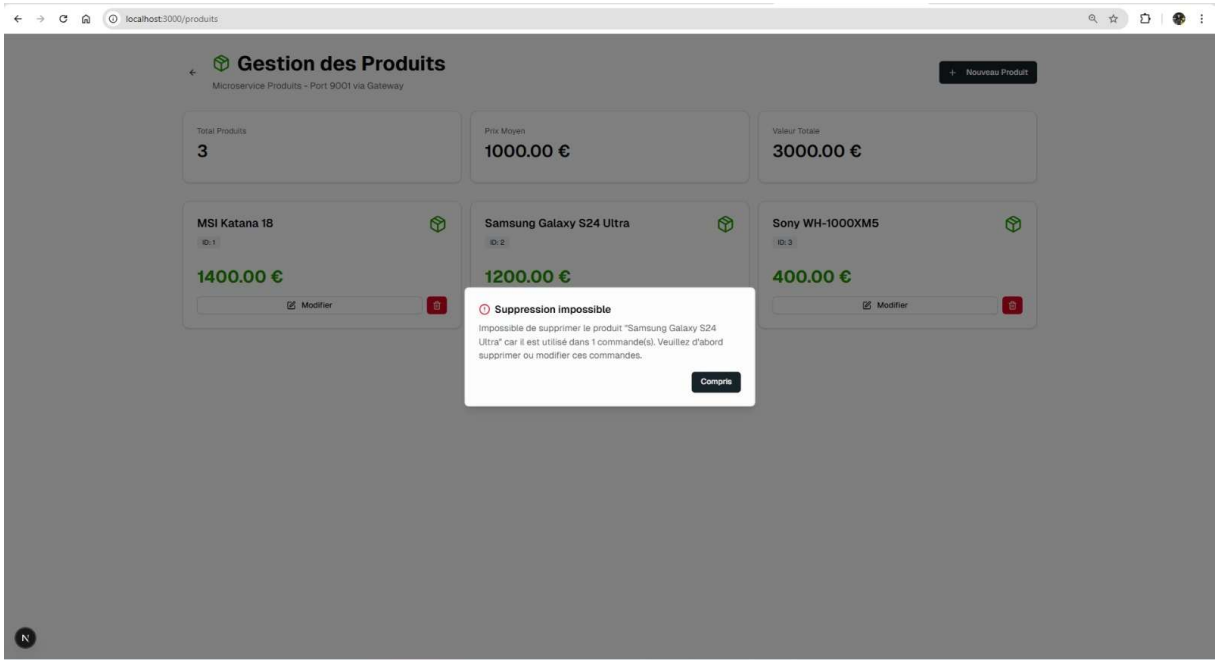

Figure 9:Message d'Erreur - Suppression Impossible

Capture montrant un message d'erreur lorsqu'un utilisateur tente de supprimer un produit utilisé dans une commande. Le message indique : "Impossible de supprimer le produit 'Samsung Galaxy S24 Ultra' car il est utilisé dans une/des commande(s)." Cette fonctionnalité garantit l'intégrité des données en appliquant des contraintes référentielles entre les entités.

3 Mise en place du serveur de configuration (Config Server)

Le Config Server a été implémenté afin de centraliser les fichiers de configuration des différents microservices. Il est connecté à un dépôt GitHub contenant les fichiers de configuration spécifiques à chaque service.

Au démarrage, chaque microservice récupère automatiquement sa configuration depuis le Config Server, ce qui garantit la cohérence et facilite la maintenance du système. Cette approche permet également de modifier les configurations sans avoir à recompiler les services.

4 Implémentation du service de découverte (Eureka Server)

Le Eureka Server a été mis en place pour assurer la découverte dynamique des microservices. Chaque service backend s’enregistre automatiquement auprès du serveur Eureka lors de son démarrage.

Grâce à ce mécanisme, les microservices peuvent se localiser entre eux sans utiliser d'adresses statiques, ce qui améliore la flexibilité et la résilience de l'architecture.

5 Mise en œuvre de l'API Gateway

L'API Gateway constitue le point d'entrée unique de l'application. Elle a été implémentée à l'aide de Spring Cloud Gateway et permet :

- Le routage dynamique des requêtes vers les microservices
- La gestion centralisée des points d'accès
- L'intégration du load balancing
- La mise en place de mécanismes de résilience

Le frontend communique exclusivement avec l'API Gateway, ce qui permet de masquer la complexité du backend.

6 Développement du microservice Produits

Le microservice Produits a été développé avec Spring Boot et expose des API REST permettant la gestion complète des produits. Il prend en charge les opérations CRUD et utilise une base de données H2 pour stocker les informations relatives aux produits.

Les endpoints sont documentés automatiquement à l'aide de SpringDoc OpenAPI, ce qui facilite la compréhension et l'utilisation de l'API.

7 Développement du microservice Commandes

Le microservice Commandes gère les opérations liées aux commandes. Il permet de créer des commandes associées à des produits, de consulter l'historique des commandes et de supprimer des commandes existantes.

Un Health Indicator personnalisé a été implémenté afin de surveiller l'état du service. Ce mécanisme facilite la supervision et l'intégration avec les outils de monitoring.

8 Implémentation de la résilience avec Resilience4J

Afin d'assurer la tolérance aux pannes, des mécanismes de circuit breaker ont été mis en place à l'aide de la bibliothèque Resilience4J. En cas de défaillance d'un microservice, le circuit breaker intercepte les appels et déclenche un mécanisme de fallback.

Cette approche permet d'éviter les effets de cascade et d'améliorer la stabilité globale du système, même en cas d'indisponibilité partielle.

9 Développement du frontend avec Next.js

Le frontend de l'application a été développé avec Next.js, un framework basé sur React. Il offre une interface utilisateur moderne, responsive et intuitive.

Les principales fonctionnalités du frontend incluent :

- Un tableau de bord global
- La gestion des produits
- La gestion des commandes
- Des formulaires interactifs
- Un design responsive adapté aux différents écrans

L'utilisation de TypeScript permet de renforcer la fiabilité du code et de réduire les erreurs.

10 Intégration frontend–backend

L'intégration entre le frontend et le backend est réalisée via des appels HTTP vers l'API Gateway. Cette approche garantit une séparation claire des responsabilités et facilite l'évolution indépendante des deux parties.

Les services backend retournent des réponses au format JSON, qui sont ensuite traitées et affichées par le frontend.

11 Tests et validation

Plusieurs types de tests ont été réalisés afin de valider le bon fonctionnement de l'application :

- Tests unitaires des services backend
- Tests d'intégration pour vérifier la communication entre les microservices
- Tests fonctionnels via l'interface frontend
- Tests de résilience en simulant l'arrêt d'un service

Ces tests ont permis de confirmer la stabilité et la fiabilité de la solution développée.

12 Synthèse de la réalisation

La phase de réalisation a permis de concrétiser la conception définie précédemment. L'ensemble des composants a été développé et intégré avec succès, aboutissant à une application fonctionnelle et cohérente.

Cette étape a également permis de renforcer nos compétences pratiques en développement d'applications distribuées et en intégration frontend/backend.

Conclusion Générale

Ce projet a permis de mettre en œuvre une architecture microservices complète dans le cadre du module J2EE, en combinant les technologies Spring Boot, Spring Cloud et un frontend moderne développé avec Next.js. L'objectif principal était de concevoir et de réaliser une application de gestion de produits et de commandes répondant aux exigences de modularité, de scalabilité et de résilience des applications distribuées modernes.

À travers ce travail, nous avons pu appliquer concrètement les concepts théoriques étudiés, notamment le découpage en microservices, la configuration centralisée, la découverte dynamique des services et l'utilisation d'une API Gateway comme point d'entrée unique. L'intégration de mécanismes de tolérance aux pannes, tels que les circuit breakers, a permis d'améliorer la robustesse et la disponibilité globale du système.

Sur le plan technique, l'architecture mise en place se distingue par son découplage, sa maintenabilité et sa facilité d'évolution. Chaque microservice est indépendant, dispose de sa propre logique métier et peut être déployé séparément. Le frontend Next.js offre une interface utilisateur moderne, responsive et intuitive, assurant une interaction fluide avec le backend distribué via l'API Gateway.

D'un point de vue pédagogique, ce projet a constitué une expérience enrichissante, nous permettant de renforcer nos compétences en développement d'applications d'entreprise, en architectures distribuées et en intégration frontend/backend. Il nous a également permis d'acquérir une meilleure compréhension des défis liés aux systèmes microservices, notamment en matière de communication inter-services, de gestion des configurations et de résilience.

Malgré les résultats satisfaisants obtenus, certaines limites subsistent, notamment l'utilisation d'une base de données en mémoire, l'absence de mécanismes avancés de sécurité et le déploiement local de l'application. Ces limites ouvrent la voie à plusieurs perspectives d'évolution, telles que l'intégration de solutions de sécurité basées sur OAuth2 ou JWT, l'ajout d'outils de monitoring et de logging, ainsi que la conteneurisation de l'application à l'aide de Docker et Kubernetes.

En conclusion, ce projet constitue une base solide pour des développements futurs plus complexes et démontre notre capacité à concevoir et implémenter une solution logicielle complète, conforme aux standards actuels du développement d'applications distribuées. Il répond pleinement aux objectifs fixés et s'inscrit dans une démarche professionnelle et pédagogique valorisante.