الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche

Centre Universitaire Abdelhafid Boussouf - Mila
Institut des Mathématiques et Informatique
Département d'informatique

Mémoire préparé pour obtenir le diplôme de Master en Informatique

Option : Sciences et Technologies de l'Information et de la Communication (STIC)

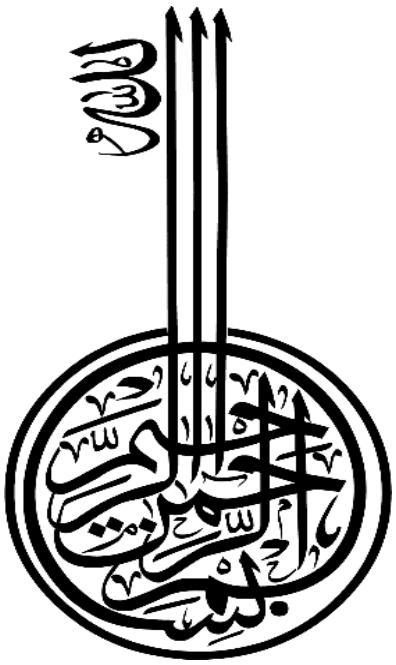
Thème :

Réingénierie des Applications Open Source à base de Microservices vers les Lignes de Produits Logiciels

Réalisé par :
Bouchareb Faten
Idri Wissal

Soutenue devant le jury :
Président : Dr. Hettab Abdelkamel
Encadré par Dr. Meghzili Said
Examinateur : Dr. Afri Faiza

Année Universitaire : 2024/2025


بسم الله الرحمن الرحيم

Remerciements

Avant tout, nous remercions Dieu de nous avoir donné la force et la patience nécessaires pour terminer ce mémoire.

Nous tenons à remercier chaleureusement notre encadreur, Dr. Saïd Meghzili, pour son écoute, ses conseils et son soutien tout au long de ce travail.

Nous remercions également l'ensemble des enseignants du Centre Universitaire Abdelhafid Boussouf - Mila pour leur accompagnement tout au long de notre parcours.

Nous adressons aussi nos remerciements aux membres du jury pour avoir accepté d'évaluer notre travail.

Enfin, un grand merci à nos familles et à nos proches pour leur soutien moral, leur compréhension et leurs encouragements, surtout dans les moments de doute.

Dédicas

With all our hearts, we dedicate this work to our families — the ones who stood by us with unwavering love, patience, and strength. Your faith in us gave us the courage to keep moving forward, even when we doubted ourselves.

To our friends, whose words of encouragement and moments of laughter helped us get through the hardest days — thank you for being our light.

And to every person who believed in us, supported us in silence, or simply reminded us that we were capable — this achievement is yours as much as it is ours.

تلخيص

في الآونة الأخيرة، تقوم الشركات والمؤسسات بنقل أنظمتها إلى بنية الميكروسيرفيس من أجل الاستفادة من المزايا التي توفرها. إلا أن هذه البنية لا تركز على تسيير التنوع في البرمجيات، مما يطرح مشكلة حقيقية. من ناحية أخرى، فإن خطوط منتجات البرمجيات (LPL) تركز أساسًا على تسيير التنوع في البرمجيات غير أنها لاتأخذ بعين الاعتبار الخصائص التقنية للأنظمة. لذلك، ومن أجل الاستفادة من مزايا هذا النهج، إلى جانب مزايا بنية الميكروسيرفيس، نقوم بإعادة هندسة تطبيقات التجارة الإلكترونية مفتوحة المصدر والتي بنيت باستخدام الميكروسيرفيس، وتحويلها إلى خطوط منتجات البرمجيات باستخدام منصة Mobioos Forge.

الكلمات المفتاحية: إعادة الهندسة، الميكروسيرفيس ، التنوع، خطوط منتجات البرمجيات، التجارة الإلكترونية، Mobioos Forge.

Résumé

Récemment, les entreprises et les organisations ont commencé à migrer leur système vers une architecture Microservices afin de bénéficier des avantages qu'elle offre. Cependant, cette architecture ne se focalise pas sur la gestion de la variabilité dans les logiciels, ce qui pose un véritable problème dans les applications. D'un autre côté, les lignes de produit logiciel (LPL) se concentrent sur la gestion de la variabilité mais ne prennent pas en considération les propriétés techniques des systèmes. Donc, pour profiter des avantages de cette approche et des avantages de l'architecture Microservices, nous migrons des applications Open Source e-commerce basées sur une architecture Microservices vers les lignes de produit logicielles en utilisant la plateforme Mobioos Forge.

Mots clés : Réingénierie, Microservices, Variabilité, Lignes de Produits Logiciels, E-commerce, Mobioos Forge.

Abstract

Recently, companies and organizations have started migrating their systems to a Microservices architecture to benefit from the advantages it offers. However, this architecture does not focus on managing variability in software, which poses a real problem in applications. On the other hand, software product lines (SPLs) focus on managing variability but do not take into account the technical properties of the systems. Therefore, to take advantage of this approach and the benefits of the Microservices architecture, we are migrating Open Source e-commerce applications based on a Microservices architecture to software product lines using the Mobioos Forge platform.

Keywords: Reengineering, Microservices, Variability, Software Product Lines, Migration, Mobioos Forge, E-commerce.

Table des matières

Table des figures

List des tableaux

Introduction générale

Ces dernières années, l'architecture Microservices est devenue de plus en plus populaire, que ce soit dans le domaine de la recherche ou dans l'industrie. Les Microservices sont de petits services autonomes qui communiquent ensemble en utilisant des protocoles légers. Ils sont indépendants les uns des autres, ce qui permet aux développeurs de choisir et de combiner librement différentes technologies. L'utilisation des Microservices promet plusieurs avantages, tels qu'une réduction des efforts de maintenance, une disponibilité accrue, une intégration plus facile de nouvelles fonctionnalités, une gestion optimisée de la scalabilité, ainsi qu'une réduction du délai de mise sur le marché. Actuellement, différentes organisations comme Netflix et Amazon ont adopté avec succès cette approche afin de moderniser leurs systèmes logiciels. Malgré les avantages techniques des Microservices, ils ne prennent pas en compte la variabilité fonctionnelle, c'est-à-dire la possibilité d'avoir des composants optionnels selon les besoins. Cela peut poser problème, par exemple si on supprime un Microservice, d'autres pourraient cesser de fonctionner. En effet, l'architecture Microservices se concentre surtout sur des propriétés non fonctionnelles et néglige la gestion de la variabilité dans les logiciels.

D'un autre côté, les lignes de produits logiciels (LPL) permettent de gérer cette variabilité. Plus précisément, une LPL permet de réutiliser systématiquement des fonctionnalités logicielles basées sur une plateforme configurable, permettant aux développeurs d'implémenter une famille de produits partageant une base commune, tout en personnalisant chaque produit selon des exigences spécifiques des clients. Pour cela, la plateforme comprend des options de configuration qui définissent quelles fonctionnalités sont variables. En pratique, les LPL ont montré qu'elles apportaient plusieurs bénéfices, tels que la réduction des coûts de développement et de maintenance, l'amélioration de la qualité logicielle ou une accélération du délai de mise sur le marché. Cependant, une LPL classique ne se concentre généralement pas sur l'amélioration des propriétés non fonctionnelles, comme le temps de réponse ou la scalabilité.

Donc, afin de tirer parti des points forts de l'architecture Microservices et des lignes de produit logiciels, nous nous intéressons dans ce travail à la réingénierie des applications open source vers les lignes de produit logiciels. Plus précisément, nous migrons deux applications open source e-commerce basées sur les Microservices vers une architecture de lignes de produits logiciels en utilisant la plateforme Mobioos Forge. L'objectif est de construire une solution adaptée aux exigences actuelles des systèmes logiciels modernes.

Ce mémoire est organisé en quatre chapitres :

Dans le premier chapitre, nous présentons tout d'abord les principes de l'architecture Microservices, suivis des avantages et des défis liés à cette approche. Ensuite, nous présentons les applications de type e-commerce. Enfin, nous exposons quelques exemples d'applications e-commerce basées sur des Microservices.

Dans le deuxième chapitre, nous présentons les concepts fondamentaux des lignes de produits logiciels (LPL), en mettant l'accent sur leur architecture ainsi que sur leur importance dans le développement logiciel. Ensuite, nous illustrerons cette approche à travers des exemples concrets d'entreprises ayant appliqué cette technique avec succès. Enfin, nous présentons la combinaison des Microservices avec l'ingénierie des lignes de produits logiciels (LPL).

Le troisième chapitre présente en détail le processus de migration de l'application open source *eShopOnContainers* vers une architecture basée sur les lignes de produits logiciels (LPL). En plus, il présente des statistiques des résultats obtenus lors de cette transition ainsi que quelques exemples de variantes dérivées.

Dans le quatrième chapitre, nous présentons la migration d'une autre application nommée *YAS,* en détaillant le processus suivi ainsi que les résultats obtenus. Le chapitre se termine par une comparaison entre les deux applications migrées.

CHAPITRE 1

Architecture Microservices et Applications E-Commerce

1.1 Introduction

Dans ce chapitre, nous allons présenter, dans un premier temps, les Microservices, leurs plateformes et leurs types existant, ainsi que leurs avantages et inconvénients. Ensuite, nous exposerons les applications e-commerce et leurs principes. Enfin, nous présenterons des exemples de des applications e-commerce basées sur des Microservices.

1.2 Architecture Microservices

La modernisation des applications dans le monde d'aujourd'hui implique souvent la migration vers des applications cloud natives, conçues sous forme de Microservices. L'architecture Microservices est une solution moderne et flexible, qui remplace le modèle de développement traditionnel de l'architecture monolithique.

1.2.1 Définition des microservices

Les microservices, plus précisément l'architecture orientée microservices (MSA), est la dernière tendance en matière de développement d'applications sur le cloud. Différentes définitions ont été proposées dont nous citons :

" *L'architecture microservice est une approche de développement d'une application unique sous forme d'une suite de petits services, chacun fonctionnant selon son propre processus et communiquant via des mécanismes légers, souvent une API de ressources HTTP. Ces services s'articulent autour de capacités métier et peuvent être déployés indépendamment par un système de déploiement entièrement automatisé. La gestion centralisée de ces services est*

minimale, et ils peuvent être écrits dans différents langages de programmation et utiliser différentes technologies de stockage de données" [1].

Les microservices [1-3] sont apparus comme une technologie qui vise à développer une application unique comme un ensemble de petits services qui exécutent leurs propres processus et communiquent via des mécanismes dits légers (par exemple, **gRPC**, **REST API**, bus d'événements). Chaque microservice est construit autour d'une fonction métier spécifique, est déployable indépendamment (via un déploiement automatisé), autonome en matière de données et peut être développé par de petites équipes indépendantes. Enfin, comme il s'agit de projets indépendants, chaque microservice peut être implémenté à l'aide de différents langages, frameworks et technologies, ce qui rend l'architecture orientée microservices agnostique [4].

1.2.2 Les microservices et les architectures monolithiques

Avant de vous lancer dans la conception d'applications avec l'architecture microservices, il est primordial de saisir les différences entre cette architecture et celle monolithique traditionnelle.

Le développement d'applications vise à répondre aux exigences métier et à appliquer la logique spécifique au domaine. Dans une structure monolithique, l'application complète est élaborée en tant qu'entité unique intégrant toute la logique d'entreprise. Dans une architecture basée sur les microservices, la logique d'entreprise est structurée en divers services faiblement interconnectés.

Figure 1.1 présente les architectures monolithiques et microservices [5] :

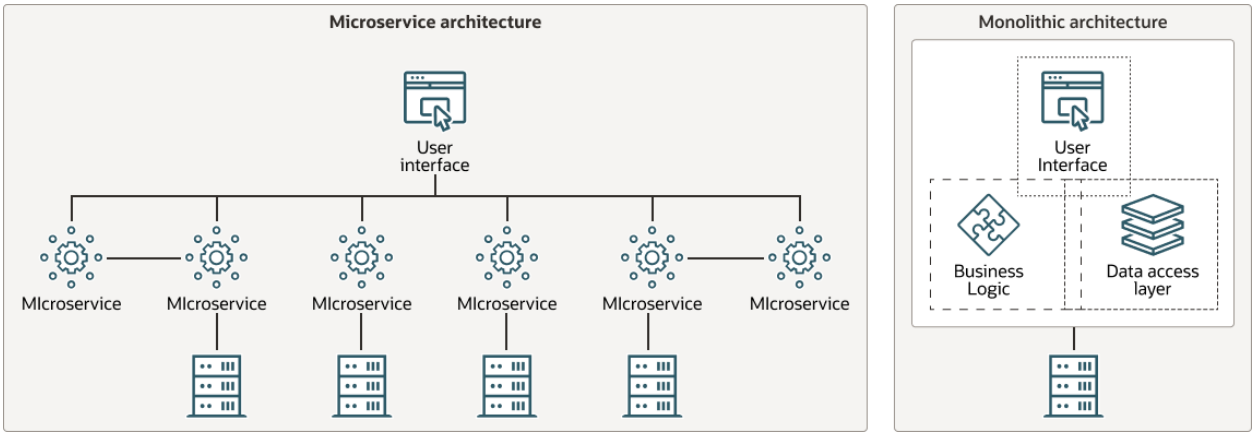


Figure 1.1 : Architectures Monolithiques et Microservices

Le tableau suivant résume les différences entre les microservices et les architectures monolithiques [5] :

Tableau 1.1: La différence entre les architectures Monolithique et Microservices

Caractéristiques	Architecture des microservices	Architecture monolithique
Conception de l'unité	L'application est composée de services faiblement couplés. Chaque service prend en charge une tâche métier unique.	L'ensemble de l'application est conçu, développé et déployé comme une seule unité.
Réutilisation des fonctionnalités	Les microservices définissent des API qui exposent leurs fonctionnalités à n'importe quel client. Ces clients peuvent 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 d'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 procédures internes.
Flexibilité technologique	Chaque microservice peut être développé à l'aide d'un langage de programmation et d'un framework qui conviennent le mieux au problème que le microservice est conçu pour résoudre.	Habituellement, l'application entière est écrite dans un seul langage de programmation.
Gestion des données	Décentralisé : chaque microservice peut utiliser sa propre base de données.	Centralisé : L'ensemble de l'application utilise une ou plusieurs bases de données.
Déploiement	Chaque microservice est déployé indépendamment, sans affecter les autres microservices de l'application.	Tout changement, aussi petit soit-il, nécessite le redéploiement et le redémarrage de l'ensemble de l'application.
Maintenabilité	Les microservices sont simples, ciblés et indépendants. L'application est donc plus facile à maintenir.	À mesure que la portée de l'application augmente, la maintenance du code devient plus complexe.
Élasticité	Les fonctionnalités de l'application sont réparties sur plusieurs services. En cas de défaillance d'un microservice, les fonctionnalités offertes par les autres microservices restent disponibles.	Une défaillance d'un composant peut affecter la disponibilité de l'ensemble de l'application.
Évolutivité	Chaque microservice peut être mis à l'échelle indépendamment des autres services.	L'ensemble de l'application doit être mis à l'échelle, même lorsque l'exigence commerciale est de mettre à l'échelle uniquement certaines parties de l'application.

1.2.3 Plateformes existantes

Il existe plusieurs plateformes pour le développement des microservices. Le tableau suivant en présente quelques-unes avec leurs principales caractéristiques [6] :

Tableau 1.2: Les différents Frameworks de développement de microservices

Framework	Description
Docker	Outil permettant d'exécuter des microservices dans des conteneurs légers et portables.
Spinnaker (Enterprise)	Plateforme open source avec divers outils pour la livraison continue.
Netflix OSS (Center)	Ensemble des outils open source permettant de construire et de déployer les services.
Spring (VMware)	Frameworks JAVA de développement de microservices.
Dropwizard	Framework largement utilisé pour développer des microservices en Java.
Vertx (Fox)	Boîte à outils pour la création des systèmes distribués réactifs sur la machine virtuelle Java.
Lagom (Odersky)	Framework open source pour la construction de systèmes de microservices réactifs en Java ou Scala.
OpenFaaS (Ellis)	Framework permettant aux développeurs de déployer facilement des fonctions et des microservices événementiels sur Kubernetes.
Azure Functions (Azure)	Outils de Microsoft de création d'une architecture de microservices.
Kubeless (framework)	Boîte à outils basée sur les ressources Kubernetes pour fournir une surveillance et un dépannage des services.
Goa (Design)	Framework pour la création de micro-services et d'API.
Kubernetes (Google)	Outils et ressources open source, permettant de gérer des applications conteneurisées sur plusieurs hôtes.
Drone (Drone.io)	Système de livraison continue basé sur la technologie des conteneurs.
Fuge (microservice shell)	Environnement d'exécution pour les développeurs des microservices.
Seneca (Framework)	Boîte à outils pour écrire des microservices et organiser la logique métier d'une application.
Bitbucket (Atlassian)	Environnement servant à planifier des projets, collaborer sur du code, le tester et le déployer.
Aws (Amazon)	Plateforme de création et de construction des microservices

1.2.4 Les types des Microservices

Le plus souvent, les Microservices sont divisés en catégories avec et sans état [7] :

- **Les microservices avec état** utilisent une base de données pour stocker les données, ce qui nécessite un espace de stockage supplémentaire. Cependant, ce type de microservice fonctionne de manière indépendante, car la base de données n'est pas partagée entre les différents microservices.
- **Les microservices sans état** ne nécessitent pas de stockage supplémentaire, car les données ne sont pas conservées. Elles n'existent qu'en réponse à une requête et ne peuvent pas être récupérées une fois la requête terminée.

1.2.5 Les avantages et les défis des microservices

Dans ce qui suit, nous présentons les avantages et les défis des Microservices.

1.2.5.1 Les avantages

L'architecture des microservices présente les avantages suivants [2] :

- **Il permet la diffusion et le déploiement continus d'applications complexes de grande taille** :

  Les équipes peuvent mettre à jour et déployer des parties de l'application de manière rapide et fiable, sans affecter l'ensemble du système.

- **Les services sont petits et faciles à entretenir :**

  Chaque service est relativement petit. Le code est plus facile à comprendre pour un développeur. Et chaque service démarre généralement beaucoup plus rapidement qu'un grand monolithe, ce qui rend également les développeurs plus productifs et accélère les déploiements.

- **Les services peuvent être déployés de façon indépendante :**

  Chaque service peut être déployé indépendamment des autres services. Si les développeurs responsables d'un service doivent déployer un changement local à ce service, ils n'ont pas besoin de coordonner avec d'autres développeurs. Ils peuvent déployer leurs modifications. Il est donc beaucoup plus facile de déployer des changements fréquemment en production.

- **Les services sont évolutifs de façon indépendante :**

Chaque service d'une architecture Microservices peut être mis à l'échelle indépendamment des autres services, chaque service peut être déployé sur le matériel le mieux adapté à ses besoins en ressources.

- **L'architecture Microservice permet aux équipes d'être autonomes** :

Vous pouvez structurer l'organisation de l'ingénierie comme un ensemble de petites équipes. Chaque équipe est seule responsable du développer, déployer et faire évoluer ses services indépendamment de toutes les autres équipes. En conséquence, la vitesse de développement est beaucoup plus élevée.

- **Il permet d'expérimenter et d'adopter facilement de nouvelles technologies :**

En principe, lors du développement d'un nouveau service, les développeurs sont libres de choisir le langage et les cadres qui conviennent le mieux à ce service.

- **Il a un meilleur isolement des défauts :**

L'architecture du microservice offre une meilleure isolation des pannes. Par exemple, une fuite de mémoire dans un service n'affecte que ce service. Les autres services continueront de traiter les demandes normalement. En comparaison, un seul élément d'une architecture monolithique qui se comporte mal détruira tout le système.

1.2.5.2 Les défis

Voici les principaux inconvénients et problèmes de l'architecture Microservice [2] :

- **Il est difficile de trouver le bon ensemble de services :**

L'un des défis de l'utilisation de l'architecture Microservice est qu'il n'existe pas d'algorithme concret et bien défini pour décomposer un système en services. Pour aggraver les choses, si vous décomposez un système de manière incorrecte, vous allez construire un monolithe distribué, un système composé de services qui doivent être déployés ensemble.

- **Les systèmes distribués sont complexes, ce qui rend le développement, les tests et le déploiement difficiles :**

Un autre problème lié à l'utilisation de l'architecture des microservices est que les développeurs doivent composer avec la complexité supplémentaire de la création d'un système distribué. Les services doivent utiliser un mécanisme de communication

interprocessus. C'est plus complexe qu'un simple appel de méthode. De plus, un service doit être conçu pour traiter les pannes partielles et gérer le fait que le service à distance soit indisponible ou présente une latence élevée.

- **Le déploiement de fonctionnalités qui couvrent plusieurs services nécessite une coordination minutieuse :**

Un autre défi lié à l'utilisation de l'architecture des microservices est que le déploiement de fonctionnalités qui couvrent plusieurs services nécessite une coordination minutieuse entre les différentes équipes de développement. Vous devez créer un plan de déploiement qui ordonne les déploiements de services en fonction des dépendances entre les services. C'est très différent d'une architecture monolithique, où vous pouvez facilement déployer des mises à jour sur plusieurs composants atomiques.

- **Il est difficile de décider quand adopter l'architecture Microservice :**

Un autre problème lié à l'utilisation de l'architecture Microservices est de décider à quel moment du cycle de vie de l'application vous devez utiliser cette architecture. Lors du développement de la première version d'une application, on ne rencontre pas les problèmes que cette architecture à résoudre. De plus, l'utilisation d'une architecture élaborée et distribuée ralentit le développement. L'utilisation de l'architecture Microservice rend beaucoup plus difficile l'itération rapide. Une startup devrait presque certainement commencer par une application monolithique.

1.3 Les applications E-Commerce

Dans cette partie, nous allons présenter les applications e-commerce, qui constituent le contexte de notre travail. Il est donc important de bien comprendre ce domaine. Dans ce qui suit, nous allons commencer par définir le domaine des applications e-commerce, ainsi que ses principes fondamentaux. Il est important de comprendre les avantages et les inconvénients de ce type d'applications. Nous terminerons cette section par des exemples concrets d'applications e-commerce, développées selon une architecture basée sur les Microservices.

1.3.1 Définition du e-commerce

Le commerce électronique, ou e-commerce, fait simplement référence à l'achat et à la vente de produits et de services sur Internet. Cependant, le terme est généralement utilisé pour décrire toutes les étapes et actions qu'un vendeur entreprend afin de vendre un produit directement à un consommateur. Le processus commence lorsqu'un client potentiel découvre le produit, l'achète et l'utilise jusqu'à ce que, idéalement, il devienne un client fidèle [8].

1.3.2 Principales fonctionnalités des applications E-Commerce

Les applications e-commerce offrent plusieurs fonctionnalités clés pour faciliter l'achat en ligne, de la recherche de produits jusqu'à la livraison. Parmi les plus importantes, on trouve [9]:

- **Liste des produits :** description détaillée, photos, prix et disponibilité du produit. Les fonctions de classification et de recherche facilitent la recherche.
- **Panier :** permet aux utilisateurs d'ajouter les produits qu'ils souhaitent acheter et de les visualiser avant de procéder au paiement.
- **Traitement des paiements :** intégrez diverses passerelles de paiement (cartes de crédit/débit, portefeuilles numériques, virements bancaires) pour faciliter les transactions sécurisées.
- **Compte utilisateur :** Fonctions d'inscription, de connexion et de gestion de profil. Historique des commandes, listes de souhaits et recommandations personnalisées.
- **Notifications :** notifications push pour informer les utilisateurs de l'état des commandes, des promotions et des nouveaux produits.

- **Avis et évaluations des clients :** aidez les autres à prendre des décisions éclairées en permettant aux clients de laisser des avis et d'évaluer les produits.
- **Options d'expédition et de livraison :** informations sur les méthodes d'expédition, les coûts et les délais de livraison. Suivi des commandes.
- **La fonctionnalité de retours :** est essentielle pour gérer les retours de produits et garantir la satisfaction des clients.

1.3.3 Avantages du l'E-Commerce

Le e-commerce offre plusieurs avantages comme [8] :

- **Faibles coûts de démarrage et d'exploitation :** démarrer une entreprise de commerce électronique est généralement beaucoup moins cher que démarrer un magazine .
- **Créer un magasin :** vous n'avez pas à payer de loyer ni à effectuer d'investissements immobiliers et vous pouvez commencer avec moins d'employés.
- **Flexibilité :** le commerce électronique est une solution plus flexible tant pour les entreprises que pour leurs clients, car les clients peuvent faire leurs achats à tout moment de la journée et depuis n'importe quel endroit.
- **Utilisez des données marketing :** votre boutique en ligne peut vous fournir de riches données marketing grâce à des analyses et des rapports, vous aidant à prendre de meilleures décisions et à promouvoir votre entreprise plus efficacement.
- **Large sélection de produits disponibles :** le commerce électronique offre aux clients une plus grande variété de produits, contrairement aux magasins traditionnels qui ne peuvent détenir qu'une certaine quantité de stocks dans l'espace physique qu'ils occupent.

1.3.4 Inconvénients de l'E-Commerce

Malgré ses avantages, le e-commerce présente aussi quelques limites, comme [8] :

- **Moins de relation avec les clients :** Parce qu'ils n'ont personne avec qui interagir au moment de faire un achat, les clients peuvent se sentir moins en relation avec votre entreprise. Les clients n'auront pas non plus l'occasion de toucher le produit avant de l'acheter pour s'assurer qu'il correspond à ce qu'ils veulent vraiment.

- **Plus de coordination pour gérer la logistique d'expédition :** Les clients de l'e-commerce peuvent se trouver n'importe où dans le monde et les dirigeants d'entreprise doivent donc s'assurer que leurs expéditions et leur réseau logistique permettront d'amener les produits jusqu'à leur destination en temps voulu. Par ailleurs, les clients doivent également attendre de recevoir leurs produits.
- **Service client minimal :** De nombreuses boutiques d'e-commerce ne disposent pas d'un représentant du service client disponible à tout moment pour répondre aux questions et résoudre les problèmes des clients.

1.3.5 Exemples d'applications e-commerce basées sur des Microservices

Dans cette partie, nous allons présenter quelques exemples d'applications e-commerce basées sur des Microservices. Nous décrirons brièvement chaque application, ainsi que son architecture et ses composants.

1.3.5.1 L'application E-shop

Pour illustrer concrètement l'architecture Microservices, on s'appuie ici sur un vrai projet open source disponible sur GitHub [10] :

Ce projet, appelé e-shop, est inspiré de *eShopOnContainers* (Microsoft), mais a été entièrement construit avec l'écosystème Spring Cloud. Il montre comment mettre en place une architecture cloud-native, modulaire et résiliente autour d'un cas réel d'application e-commerce.

On s'appuie ici sur le schéma d'architecture fourni pour décrire les composants visibles du système.

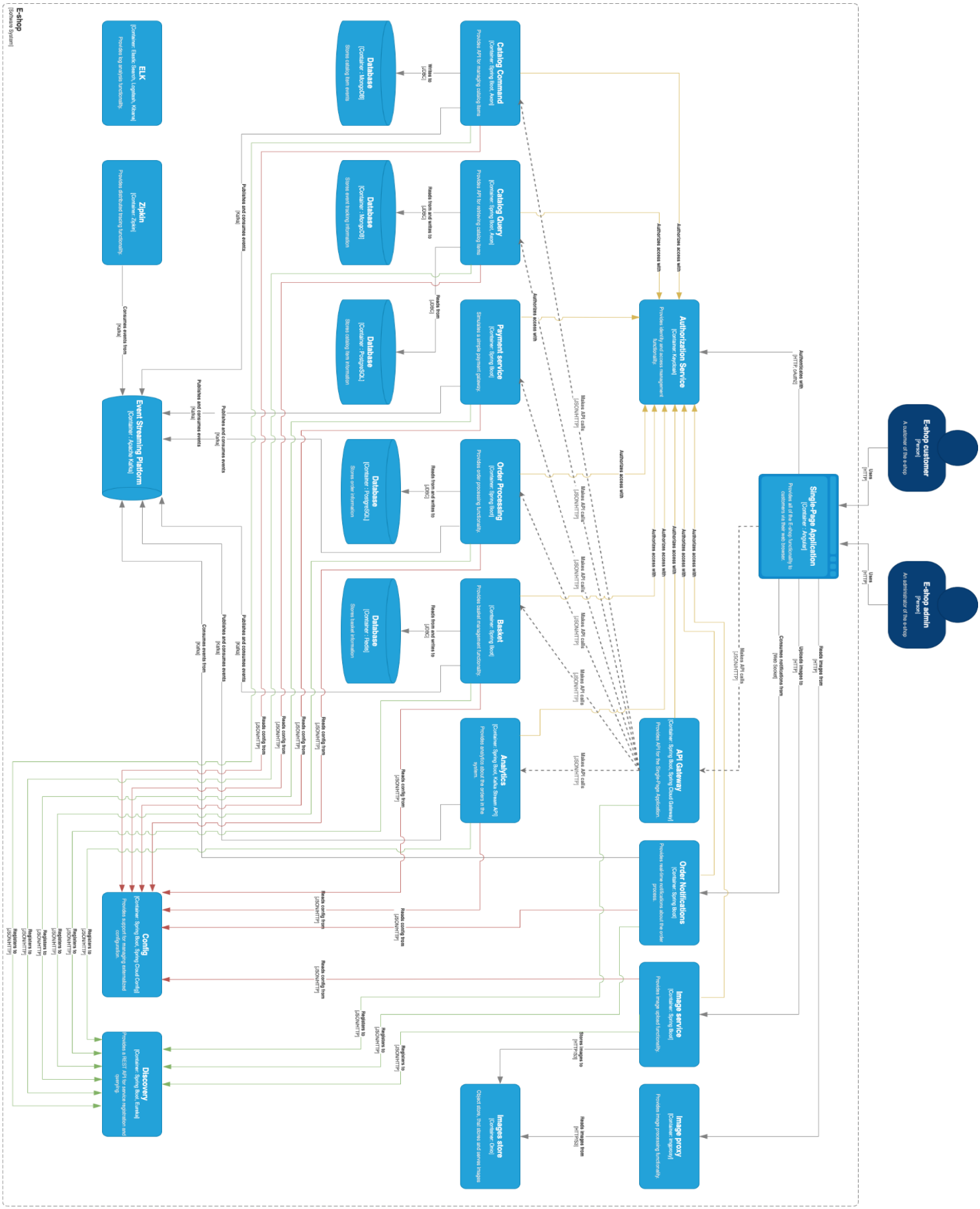


Figure 1.2 : Architecture de l'application e-shop

Dans ce qui suit, nous présentons les composants principaux de l'application *E-shop (Figure 1.2)*, organisée selon une architecture à base de Microservices :

Interface utilisateur : Single Page Application (SPA)

L'utilisateur interagit avec le système via une application web moderne de type SPA (Single Page Application) développée avec Angular. Toutes les requêtes partent de cette interface et passent par une API Gateway, qui joue le rôle de point d'entrée unique vers le système.

API Gateway

C'est le gardien de l'architecture. Elle s'occupe de :

- Rediriger les requêtes vers les bons services,
- Gérer la sécurité (authentification/autorisation),
- Appliquer des règles comme la limitation de débit ou la journalisation.

Elle centralise tout le trafic venant du client.

Identity Server

Ce composant est chargé de vérifier l'identité des utilisateurs et de gérer leurs droits d'accès. Il travaille main dans la main avec l'API Gateway pour protéger toutes les ressources exposées par les services métier.

Microservices métier

Chaque service est autonome, spécialisé dans une tâche précise et possède sa propre base de données. Dans le schéma, on retrouve :

- **Catalog Service :** gère les produits disponibles à la vente.
- **Basket Service :** permet aux utilisateurs d'ajouter ou retirer des articles de leur panier.
- **Ordering Service :** traite les commandes passées.
- **Payment Service :** simule le paiement des commandes.
- **Delivery Service :** gère les livraisons des commandes.

Cette séparation permet de développer, déployer et faire évoluer chaque service indépendamment.

Communication entre services

Les services échangent principalement de manière asynchrone, grâce à un message broker (RabbitMQ dans le projet). Cela améliore la tolérance aux pannes et la scalabilité.

Côté données, chaque service utilise la base la plus adaptée à son besoin (MongoDB, SQL Server, Redis...).

1.3.5.2 L'application eShopping

Pour illustrer les Microservices, on présente un autre cas réel disponible sur GitHub [11]. Ce projet, appelé *eShopping*, est une application complète de e-commerce construite autour de l'écosystème .NET Core. Il met en œuvre les meilleures pratiques d'une architecture microservices cloud-native orientée production, avec des technologies modernes comme Docker, Kubernetes, GRPC, IdentityServer, Redis, et bien d'autres. On s'appuie ici sur le schéma d'architecture du projet pour décrire les composants visibles du système.

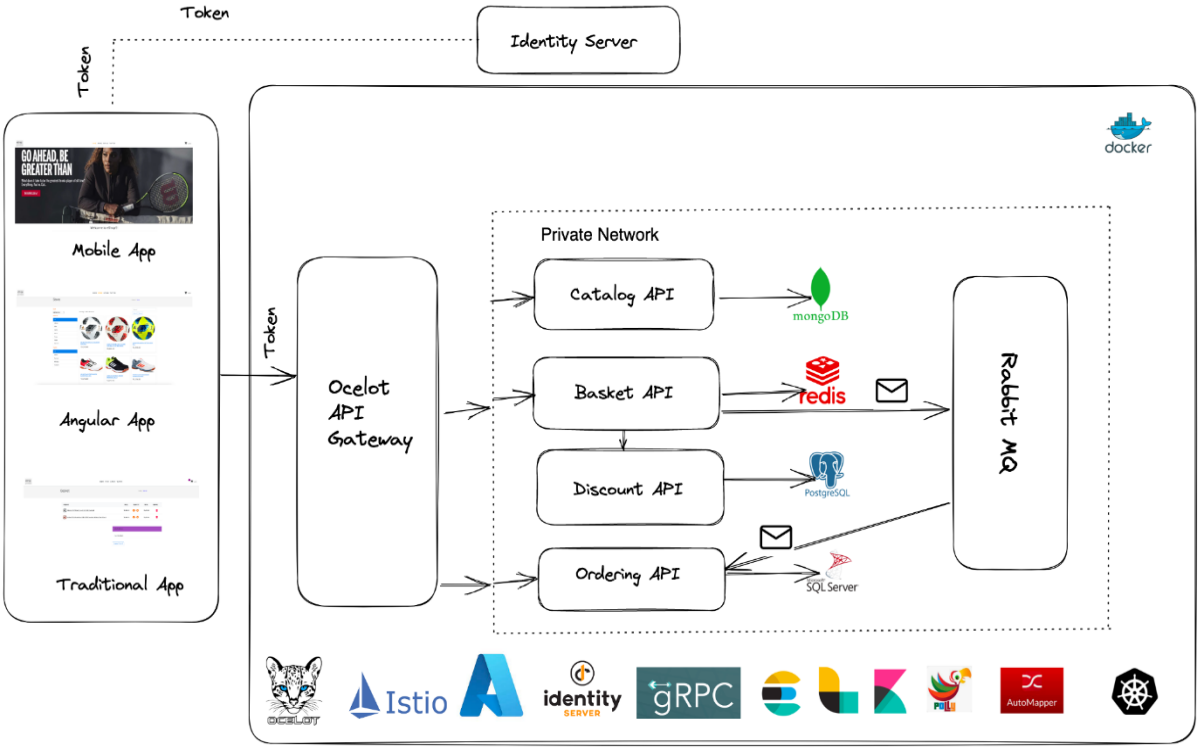


Figure 1.3 : Architecture de l'application eShopping

Dans ce qui suit, nous présentons les composants principaux de l'application *eShopping* :

▶ Interface utilisateur : Application Angular

L'utilisateur final interagit avec le système via une application Angular moderne. C'est une Single Page Application (SPA) qui consomme les services backend à travers l'API Gateway. Toutes les actions (ajouter au panier, commander, consulter les produits...) passent par ce point d'entrée unique.

▶ API Gateway – Ocelot

Elle sert de façade unique entre le client et les microservices. Elle joue plusieurs rôles clés :

- Redirection des requêtes vers les bons services backend (ex : panier, commande…)

- Authentification via IdentityServer
- Agrégation des réponses
- Application de règles de sécurité et de routage

Elle garantit une séparation claire entre frontend et backend tout en simplifiant la consommation des services.

▶ Identity Server

Le projet utilise IdentityServer4 comme serveur d'authentification et d'autorisation. Il sécurise l'accès aux API en émettant des tokens JWT. Chaque action de l'utilisateur (accès au panier, commande...) nécessite une vérification de l'identité via ce composant.

▶ Microservices métier

Chaque fonction de l'application est implémentée comme un microservice autonome, avec sa propre logique métier et sa propre base de données. Parmi eux :

- **Catalog.API :** gestion du catalogue de produits.
- **Basket.API :** gestion du panier de l'utilisateur.
- **Discount.API** : microservice optimisé en GRPC pour gérer les réductions.
- **Ordering.API :** traitement des commandes (stockage, gestion, confirmation).
- **Payment.API :** simulation du processus de paiement.

Tous les services sont déployés via Docker et orchestrés via Kubernetes, ce qui facilite l'évolutivité et la résilience.

▶ Communication entre services

L'interaction entre microservices est gérée de deux manières :

- **GRPC :** pour une communication performante (ex : service de réduction)
- **RabbitMQ :** pour une communication asynchrone et découplée via des événements, améliorant la scalabilité et la tolérance aux pannes.

1.4 Conclusion

Dans ce chapitre, nous avons présenté l'architecture Microservices, ainsi que les applications e-commerce. En plus, nous avons présenté quelques exemples d'applications e-commerce basées sur l'architecture Microservices.

Dans le chapitre suivant, nous introduirons l'approche des lignes de produits logiciels (LPL).

CHAPITRE 2

Les Lignes De Produits Logiciels (LPL)

2.1 Introduction

La complexité croissante des systèmes d'information rend leur développement toujours plus difficile. Pour répondre à ces défis, de nouvelles approches sont proposées afin d'augmenter la productivité et de réduire le temps de réalisation. Parmi elles, les lignes de produits logiciels (LPL) constituent une solution intéressante. Elles regroupent un ensemble de logiciels appartenant à un même domaine et qui prend en compte des facteurs de variation spécifiques à chaque produit.

Dans ce chapitre, tout d'abord, nous définissons les lignes de produits logiciels (LPL) et leurs concepts fondamentaux. Ensuite, nous présentons les principes de gestion associés à ces lignes de produits, l'ingénierie du domaine, l'ingénierie d'application et la plateforme Mobioos forge. Après, nous présentons les avantages liés à l'utilisation des LPL dans le développement logiciel. Par la suite, nous illustrons ces notions à travers des exemples concrets de lignes de produits logiciels. Enfin, nous explorons la combinaison des Microservices avec l'ingénierie des lignes de produits logiciels (LPL).

2.2 Définition des LPL

Il y a un consensus dans la littérature sur la définition d'une ligne de produits logiciels :

Elle est définie comme un ensemble de systèmes partageant des propriétés communes et répondant à des exigences spécifiques pour un domaine particulier [12].

La définition de Clements et Northrop est très souvent citée dans les travaux du domaine :

"A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way " [13].

2.3 Concepts fondamentaux

La définition ci-dessus de la ligne de produits caractérise un ensemble de produits qui partagent des caractéristiques communes (commonalité), mais aussi des différences (variabilité) spécifiques à chaque membre de la ligne. Nous présentons ici leurs définitions [12] :

- **Un domaine** est un secteur de métier ou de technologies ou des connaissances caractérisées par un ensemble de concepts et de terminologies que les utilisateurs de ce secteur peuvent comprendre.
- **La variabilité** englobe toutes les hypothèses permettant d'expliquer en quoi les produits, membres de la ligne de produits, diffèrent les uns des autres.
- **La commonalité** regroupe l'ensemble des hypothèses qui sont vraies pour tous les produits, membres de la ligne de produits.

2.4 Les principes de gestion des lignes de produits

L'ingénierie des lignes de produits logiciels, telle qu'elle est définie dans la littérature, distingue deux niveaux, illustrés par la Figure 2.1 : Ingénierie du domaine et Ingénierie d'application [12] :

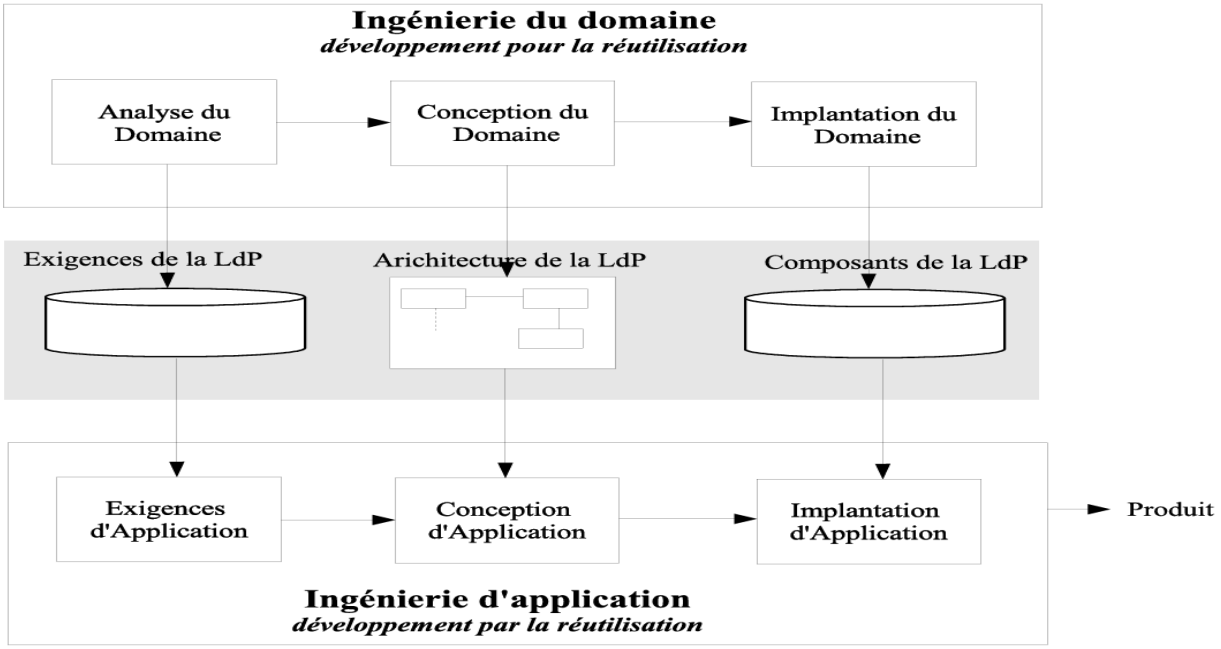


Figure 2.1: L'ingénierie de lignes de produits

2.5 Ingénierie du domaine

L'objectif de l'ingénierie du domaine est de déterminer avec précision la famille de logiciels représentée par la Ligne de Produits Logiciels (LPL) [13]. Nous distinguons un processus composé de trois activités : l'analyse, la conception et l'implantation du domaine [12].

▶ Analyse du domaine :

L'analyse du domaine a pour but de créer un modèle qui décrit le fonctionnement général du domaine concerné. Ce modèle contient des informations utiles pour définir le système et rédiger sa spécification. L'objectif est d'étudier le domaine de la ligne de produits afin d'en identifier les éléments communs et les différences entre les produits.

Il existe plusieurs méthodes pour l'analyse de domaine, dont la plus connue est FODA. Dans cette méthode, le domaine est représenté par un modèle de caractéristiques (appelé *Feature Model* ou FM), structuré sous forme d'un arbre. Chaque nœud de l'arbre correspond à une caractéristique du domaine, et les liens entre les nœuds montrent comment ces caractéristiques sont reliées entre eux. FODA distingue trois types de caractéristiques : les caractéristiques obligatoires, présentes dans tous les produits de la ligne, les caractéristiques optionnelles, qui apparaissent seulement dans certains produits, et les caractéristiques alternatives, parmi lesquelles il faut en choisir une ou plusieurs. Le modèle peut aussi inclure des contraintes supplémentaires entre les caractéristiques.

La Figure 2.2 montre un exemple de modèle de caractéristique FODA d'une ligne de produits de voitures [12]

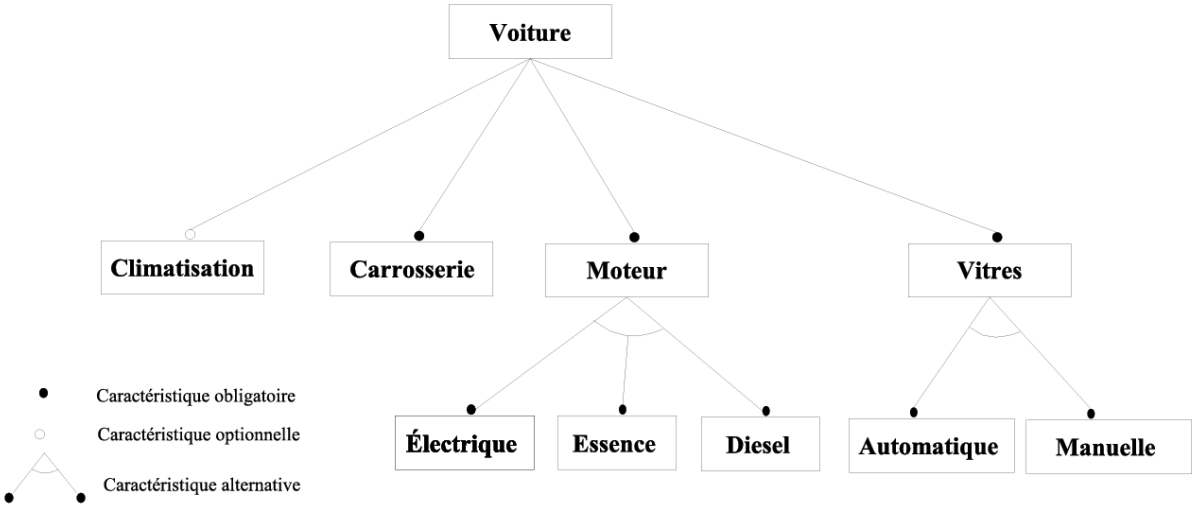


Figure 2.2 : Exemple d'un diagramme de caractéristiques FODA

- **La conception du domaine :**

Le but de la conception de domaine est de définir une architecture logicielle générique pour la ligne de produits. Cette architecture standard comprend un ensemble de composants, de connecteurs et de contraintes. Dans le contexte des lignes de produits, elle sert d'architecture de référence à partir de laquelle l'architecture de chaque produit est construite. Les éléments de variabilité identifiés lors de l'analyse du domaine doivent y être représentés de façon explicite.

- **L'implantation du domaine :**

Il s'agit de mettre en œuvre l'architecture générique définie lors de la conception du domaine, en la réalisant sous forme de composants réutilisables dans l'ingénierie d'application pour construire chaque produit.

2.6 Ingénierie d'application

L'ingénierie d'application consiste à utiliser les résultats de l'ingénierie de domaine pour la construction, également appelée dérivation, d'un produit particulier. Il s'agit d'un développement basé sur la réutilisation (voir la Figure 2.1). Comme mentionné ci-dessus, les résultats de l'ingénierie de domaine (modèles de caractéristiques, architecture générique et composants) intègrent de la variabilité. La dérivation d'un produit spécifique nécessite donc de prendre des décisions (ou faire des choix) liées à ces points de variation. La notion du modèle de décision est utilisée pour formaliser et enregistrer les décisions nécessaires à cette dérivation [12]. Dans la littérature deux activités essentielles liées à l'ingénierie de l'application [13] :

- La sélection des fonctionnalités du produit à réaliser, aussi appelée processus de configuration.
- La réalisation finale du produit à partir de la configuration définie et des éléments de code associés, constitue ce que l'on appelle le processus de réalisation. Ce processus est généralement automatisé et s'appuie sur l'ensemble des choix effectués lors du processus de configuration.
- **La notion de configuration** est utilisée pour représenter un produit d'une ligne de produits (LdP). Elle consiste en la sélection de caractéristiques compatibles avec les contraintes du modèle de caractéristiques (FM). Par exemple, dans le cas des voitures, une configuration peut inclure le choix des différentes options pour un modèle particulier [14].

2.7 Plateforme Mobioos forge

Mobioos Forge (MF) [15] est une extension de Visual Studio Code (VS Code), disponible gratuitement sur le MarketPlace de VSCode. Elle suit les principes de l'ingénierie des lignes de produits logiciels (SPL), mais propose une approche orientée vers la migration. L'idée est d'aider les développeurs à créer des lignes de produits logiciels (SPL) à partir du code source d'applications existantes, développées individuellement.

MF se décompose en quatre grandes activités :

- Spécification du modèle de fonctionnalités
- Mappage des fonctionnalités
- Détection des contraintes
- Génération des variantes

Dans la figure 2.3, nous allons présenter l'architecture de la plateforme Mobioos Forge (MF) [16] :

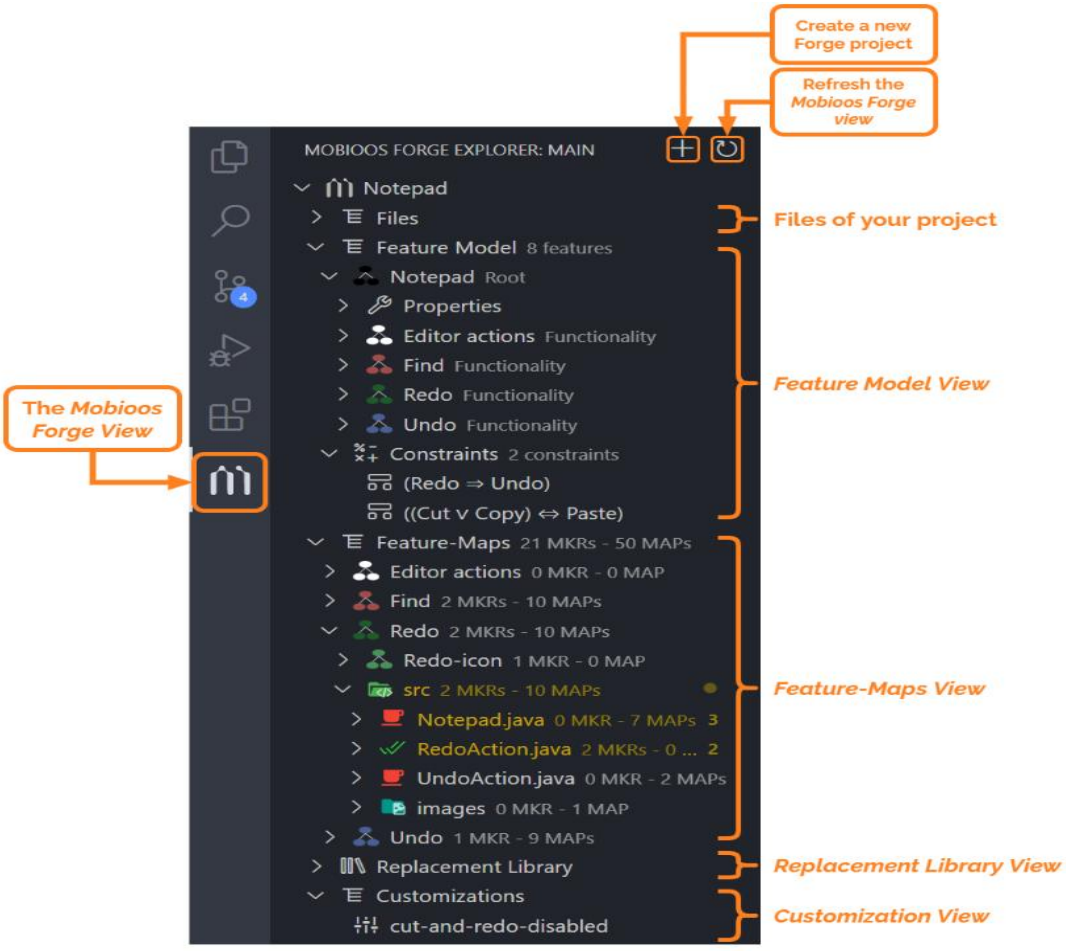


Figure 2.3 : Architecture de plateforme Mobioos Forge

2.8 Les avantages de l'utilisation des lignes de produits logiciels

Les LPL offre plusieurs avantages pour le développement et la gestion des applications [17] :

- **Qualité constante des produits**

Utilise des éléments de base communs à toutes les versions du produit, ce qui garantit une qualité constante et élevée.

- **Accélération du développement de produits**

Optimise les cycles de développement et accélère la mise sur le marché en partageant les ressources et en automatisant les différentes versions d'un produit.

- **Renforcement de l'innovation**

Augmente la capacité d'innovation des produits en intégrant le développement logiciel et matériel, en encourageant la collaboration et en permettant l'ajout rapide de nouvelles fonctionnalités.

- **Évolutivité**

Facilite l'expansion et l'ajout rapide de nouvelles variantes en gérant les configurations des produits de manière centralisée et en réutilisant efficacement les composants.

- **Efficacité optimisée grâce à la réutilisation des actifs**

Utilise des ressources techniques partagées, comme les spécifications, les modèles, le code et les tests, entre les différentes gammes de produits pour éviter les redondances et optimiser le développement.

- **Délai de mise sur le marché écourté**

Améliore la présence sur le marché et la compétitivité en réduisant les délais de lancement des nouvelles variantes, et offre un avantage concurrentiel grâce à une adaptation plus rapide.

2.9 Exemples des applications LPL

Dans ce qui suit, nous présentons trois exemples concrets d'applications LPL suivants : (1) lignes de produits automobiles, (2) l'application du Crocs, (3) l'application du Microsoft.

2.9.1 Lignes de produits automobiles

Chaque voiture qui sort d'une usine automobile moderne est unique chacune est adaptée aux besoins et aux souhaits d'un client particulier. Mais comment le client communique-t-il ses exigences ? Aujourd'hui, on peut utiliser des configurateurs de voitures en ligne pour cette tâche. Dans la Figure 2.4, nous montrons une capture d'écran du configurateur de BMW. Le client peut faire divers choix, notamment la série, la carrosserie, la couleur, la puissance, le prix, l'efficacité énergétique, et bien d'autres encore.

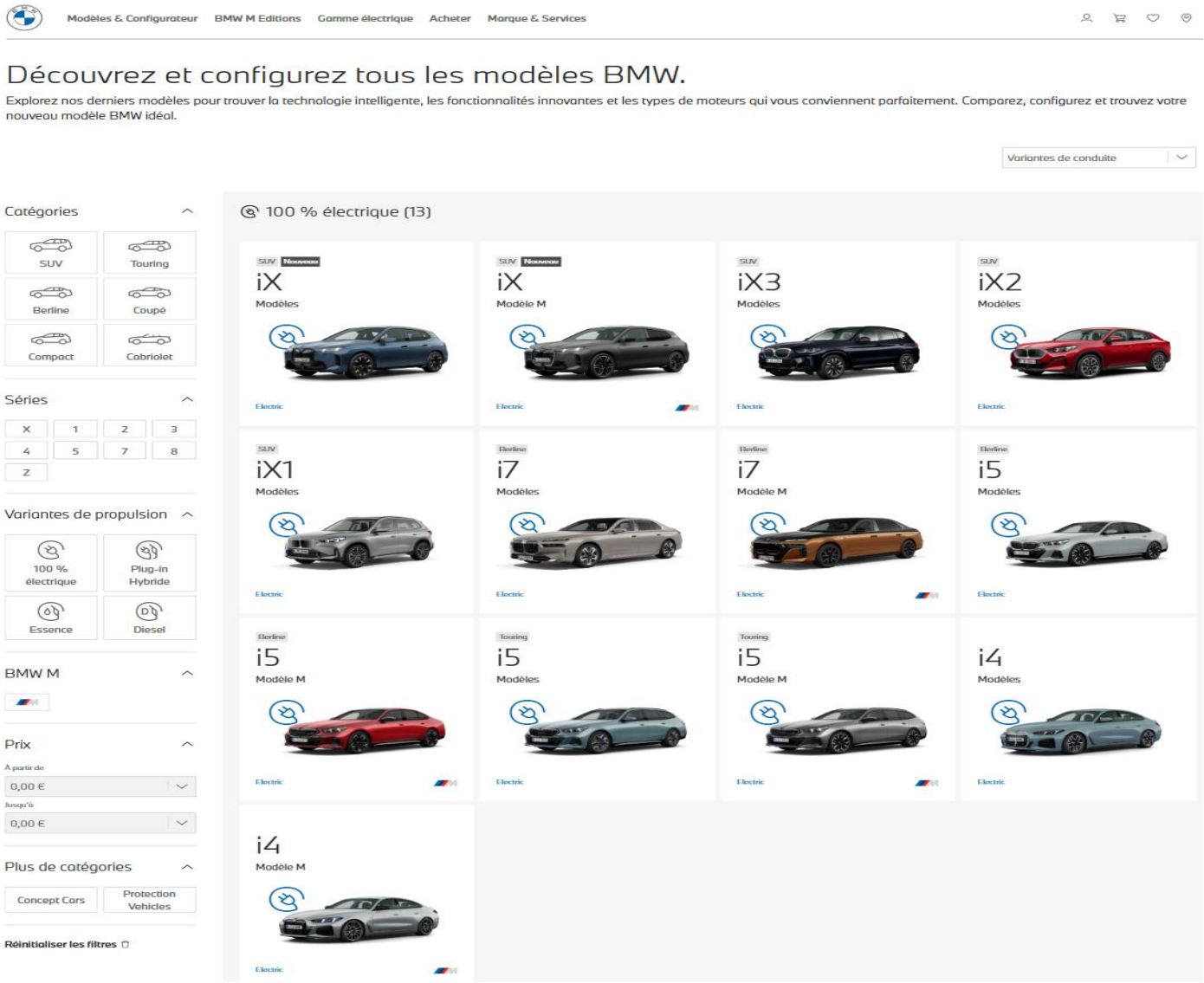


Figure 2.4: Le configurateur automobile de BMW

Notez que la Figure 2.4 ne montre qu'un petit extrait des choix qu'un client peut faire. En réalité, il existe des centaines, voire des milliers d'options possibles. Cet ensemble riche d'options de configuration donne lieu à un nombre astronomique de variantes de voitures possibles.

Il appartient donc au configurateur de guider le client à travers la configuration, en fournissant des retours, en masquant les choix invalides, etc. Il agit comme un assistant qui, en plusieurs étapes, propose des choix sous forme de cases à cocher et de boutons radio ; il existe même des curseurs permettant de choisir dans un spectre de valeurs possibles. Bien sûr, une fois la configuration définie, la voiture correspondante n'est pas conçue et construite à partir de zéro. Elle est plutôt assemblée automatiquement à partir de pièces réutilisables, en fonction des choix effectués par le client. Certains choix sont directement liés à des composants mécatroniques individuels (par exemple, le type de transmission), tandis que d'autres résultent de l'interaction entre plusieurs pièces (par exemple, l'efficacité énergétique) [18].

2.9.2 Crocs

Crocs est une marque connue pour ses sabots en mousse uniques, illustre également l'approche des lignes de produits, en diversifiant son offre pour répondre à une clientèle variée [20] :

- **Chaussures Crocs** : disponibles en plusieurs styles (sabots classiques, claquettes, sandales) et coloris, elles sont reconnues pour leur confort, leur légèreté et leur facilité d'entretien.
- **Chaussettes Crocs** : conçues pour accompagner les chaussures, elles se distinguent par leurs motifs ludiques qui renforcent l'identité jeune.
- **Breloques Jibbitz** : ces accessoires personnalisables s'insèrent dans les trous des sabots, permettant aux clients de personnaliser leur paire selon leurs goûts, et renforçant l'aspect modulaire et configurable du produit.
- **Talons Crocs** : une ligne plus formelle qui reprend les caractéristiques de confort propres à la marque, mais adaptée aux milieux professionnels grâce à un design plus soigné.

Figure 2.5 illustre la diversité des produits Crocs proposée sur leur site officiel [21] :

Figure 2.5: Aperçu des lignes de produits Crocs sur le site officiel

2.9.3 Microsoft

Microsoft, leader mondial dans les logiciels, le matériel et les services cloud, offre plusieurs lignes de produits adaptées à différents secteurs et besoins [20] :

- **Applications logicielles :** Microsoft est surtout connu pour ses produits logiciels tels que Microsoft Office (Word, Excel, PowerPoint), Microsoft Teams et Outlook , qui sont essentiels à la productivité et à la communication dans les environnements personnels et professionnels.
- **Systèmes d'exploitation :** Les systèmes d'exploitation de Microsoft incluent Windows 10 et Windows 11, ainsi que la série Windows Server , chacun répondant aux besoins des utilisateurs à domicile, des professionnels et des entreprises.
- **Appareils matériels :** La ligne de produits Microsoft comprend des produits tels que la Surface Pro (hybride tablette/ordinateur portable), le Surface Laptop et les consoles de jeu Xbox. Ces appareils sont conçus pour répondre aux besoins des utilisateurs particuliers et professionnels.
- **Services Cloud :** Azure est le service de cloud computing de Microsoft, qui fournit aux entreprises une large gamme de solutions, notamment le stockage, l'analyse et les applications basées sur le cloud, positionnant Microsoft comme un leader de la technologie cloud.

La Figure 2.6 illustre la diversité des produits Microsoft proposée sur leur site officiel [22] :

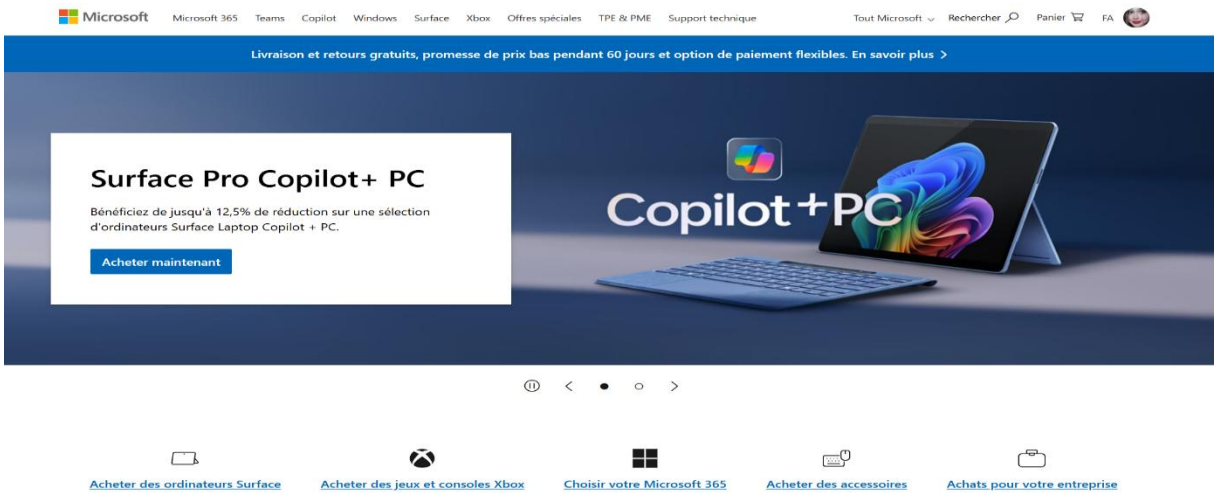


Figure 2.6: Aperçu des lignes de produits Microsoft sur le site officiel

2.10 Combinaison des Microservices avec l'Ingénierie des Lignes de Produits Logiciels (LPL)

L'architecture Microservices et les LPLs visent toutes deux pour objectif à rendre le développement logiciel plus flexible, modulaire et réutilisable. Les Microservices décomposent les applications en petits services autonomes, tandis qu'une LPL permet de gérer efficacement les variations entre différents produits à partir d'une base commune.

Les combiner permet de structurer les services Microservices comme des *features* dans une ligne de produits, afin de générer facilement différentes versions d'une même application, selon les besoins. Cela aide à gérer les évolutions, à réduire les duplications, et à mieux personnaliser les systèmes pour différents clients.

Dans notre travail, cette combinaison sera appliquée sur deux applications e-commerce basées sur les Microservices. Dans les chapitres 3 et 4, nous allons transformer ces applications en lignes de produits logicielles en utilisant l'outil Mobioos Forge, qui facilite la migration vers les LPLs en exploitant la modélisation des variantes et la génération automatique [23].

2.11 Conclusion

Dans ce chapitre, nous avons introduit le concept des lignes de produits logiciels (SPL), une approche stratégique et systématique permettant de développer efficacement une famille de produits logiciels. Plus précisément, nous avons exploré les concepts fondamentaux de variabilité et de commonalité, ainsi que les deux phases clés de cette ingénierie. À travers ces phases, une LPL favorise la réutilisation des actifs logiciels. Nous avons également mis en évidence des nombreux avantages offerts par cette approche. L'illustration par des exemples concrets tels que les constructeurs automobiles, les sandwicheries et l'entreprise Microsoft a permis de mieux comprendre l'application pratique des LPLs dans des domaines variés. Enfin, nous avons établi un lien entre l'ingénierie LPL et l'architecture Microservices. Cette combinaison ouvre la voie à une gestion encore plus souple des variantes de produits logiciels. Ce lien fera l'objet d'une mise en œuvre concrète dans les chapitres suivants à travers l'outil Mobioos Forge.

CHAPITRE 3

Migration de l'application eShopOnContainers vers Les lignes de produits logiciels

3.1 Introduction

Dans ce chapitre, nous présentons la migration de l'application open source *eShopOnContainers* vers une Ligne de Produits Logiciels (LPL) en utilisant *Mobioos Forge*. Cette application e-commerce est construite sur la base des Microservices et utilise plusieurs technologies modernes telles que *.NET 9, .NET Aspire, C#, ASP.NET Core* et *Docker.*

Dans ce qui suit, d'abord, nous présentons l'application eShopOnContainers. Ensuite, nous exposons le modèle de fonctionnalité et le mappage des fonctionnalités de cette application. Par la suite, nous donnons des exemples de variantes dérivées, et nous terminons ce chapitre par le calcul du temps de migration de cette application.

3.2 Présentation de l'application eShopOnContainers basée sur les microservices

eShopOnContainers [24] est une application e-commerce construite selon une architecture moderne basée sur les Microservices. Elle utilise de nombreuses technologies telles *que .NET 9, ASP.NET Core, C#, Blazor, gRPC, Docker, Redis, RabbitMQ, PostgreSQL, OpenAI, Playwright,* ainsi que l'environnement *.NET Aspire* pour gérer l'orchestration des services.
L'application *eShopOnContainers* est composée de plusieurs services métiers qui communiquent via des *API HTTP, gRPC* et un bus d'événements (*RabbitMQ*). Elle comprend aussi des interfaces clients (web et mobile) et des services d'administration et d'observabilité.

Nous présentons dans la figure 3.1 l'architecture de l'application *eShopOnContainers* [24] :

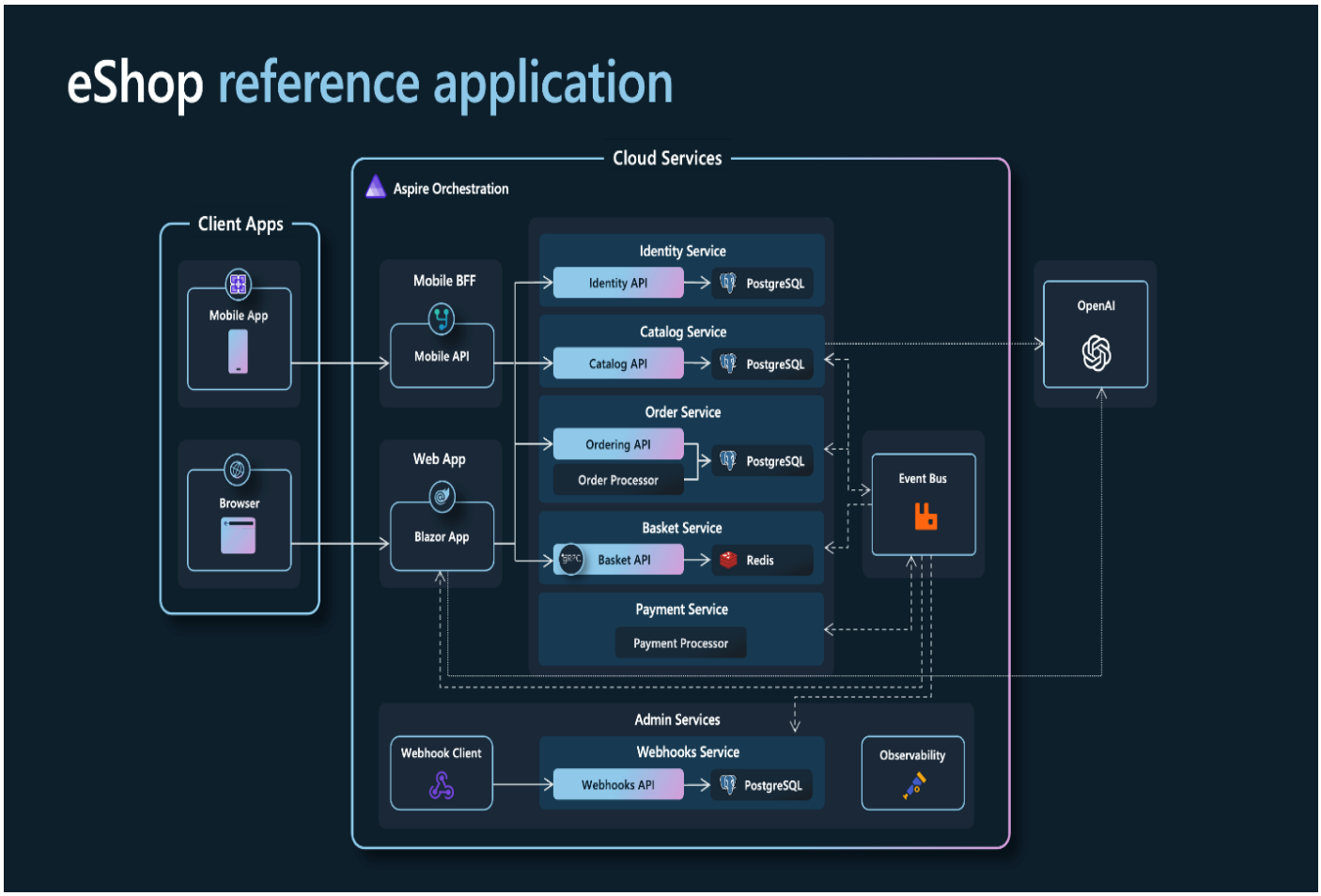


Figure 3.1 : Architecture Microservices de l'application eShopOnContainers

Voici les différents composants représentés dans cette architecture de l'application eShopOnContainers :

- **Interfaces utilisateur**
  - **Blazor Web App (WebApp) :** Interface web principale développée avec Blazor, offrant une expérience utilisateur moderne et interactive.
  - **Mobile App (MobileApp) :** Application mobile développée avec .NET MAUI, fournissant une version optimisée pour les terminaux mobiles.
- **Mobile BFF (Back-End For Front-End) :** Sert d'intermédiaire entre les applications mobiles et les autres microservices, en fournissant une API optimisée.
- **Identity Service (Login) :** Gère l'authentification et l'autorisation des utilisateurs. Il utilise une base de données **PostgreSQL** pour stocker les informations d'identités.
- **Catalog Service (Catalog) :** chargé de la gestion des produits du catalogue.

- **Order Service (Order) :** Gère tout le cycle de vie des commandes, y compris leur création, validation et suivi.
- **Basket Service (ShoppingBag) :** Permet aux utilisateurs d'ajouter, de modifier ou de supprimer des produits dans leur panier.
- **Payment Service (Payment) :** Prend en charge la logique de paiement, avec une simulation de connexion à des services de paiement externes.
- **Webhooks Service (WebhookClient) :** Envoie des requêtes HTTP à des systèmes tiers lorsque des événements spécifiques se produisent dans l'application.
- **OpenAI Service** : Microservice expérimental intégrant les API d'OpenAI pour enrichir l'expérience utilisateur avec des fonctionnalités basées sur l'intelligence artificielle.

Nous disposons ici de 13 conteneurs représentant les différents services de notre application *eShopOnContainers*, comme illustré dans l'image ci-dessous (figure 3.2) :

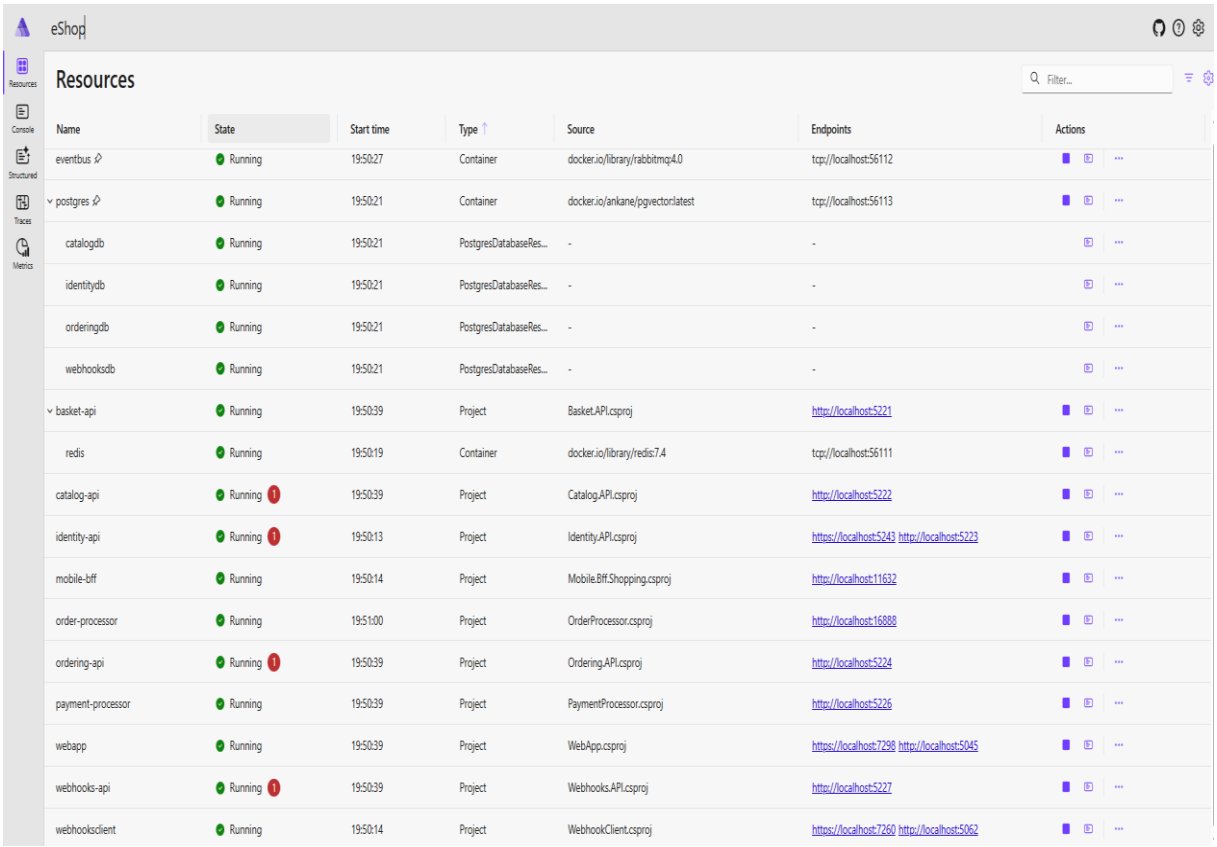


Figure 3.2 : Services de l'application eShopOnContainers dans .Net Aspire

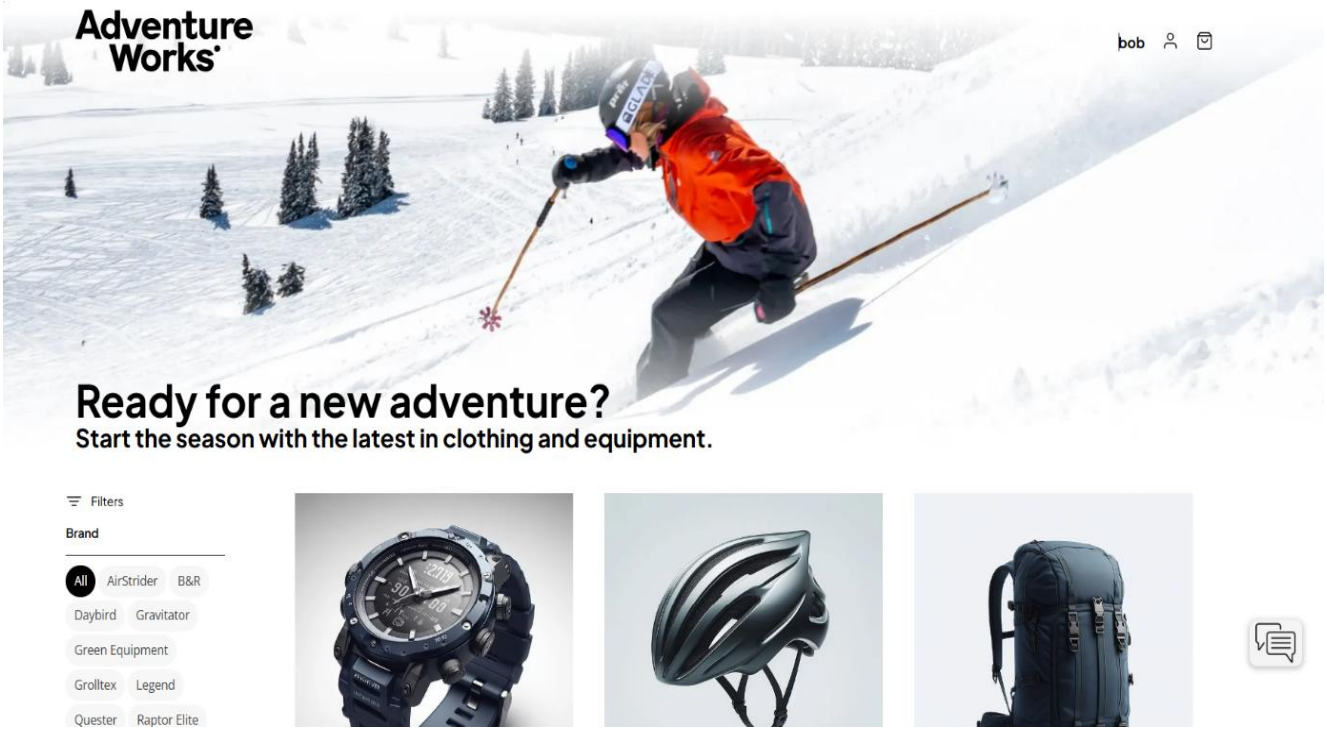


Figure 3.3: L'interface d'application web pour la page d'accueil

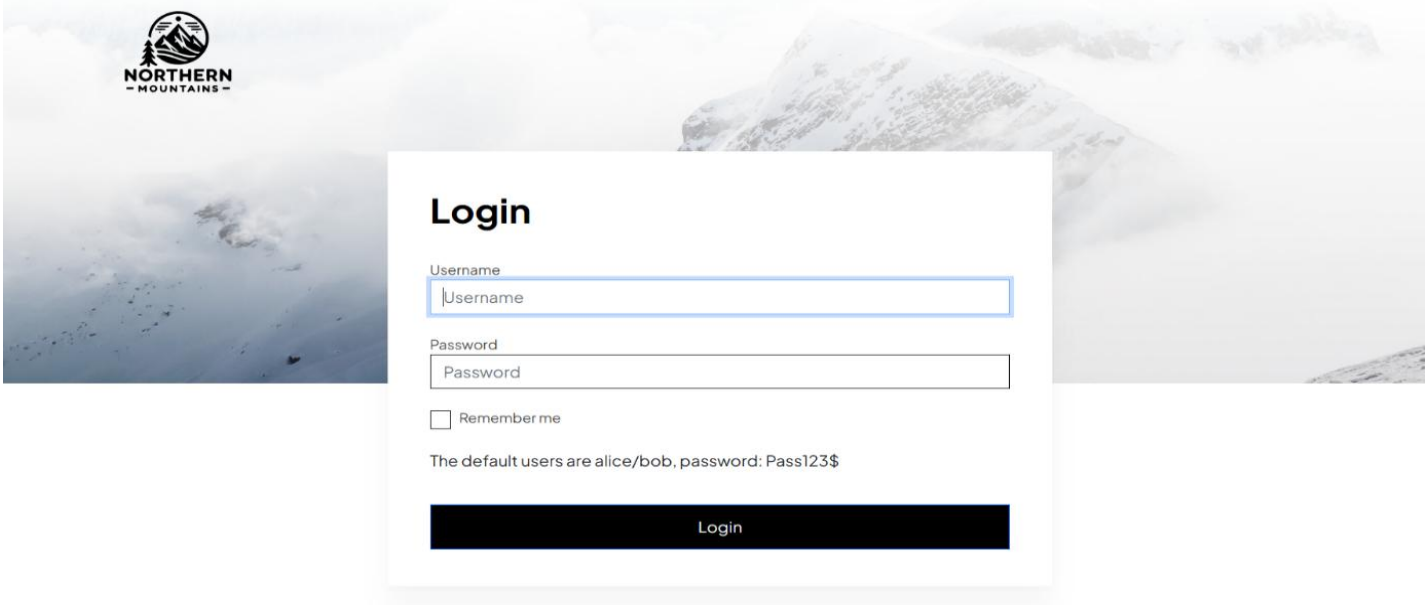


Figure 3.4 : L'interface d'application Login

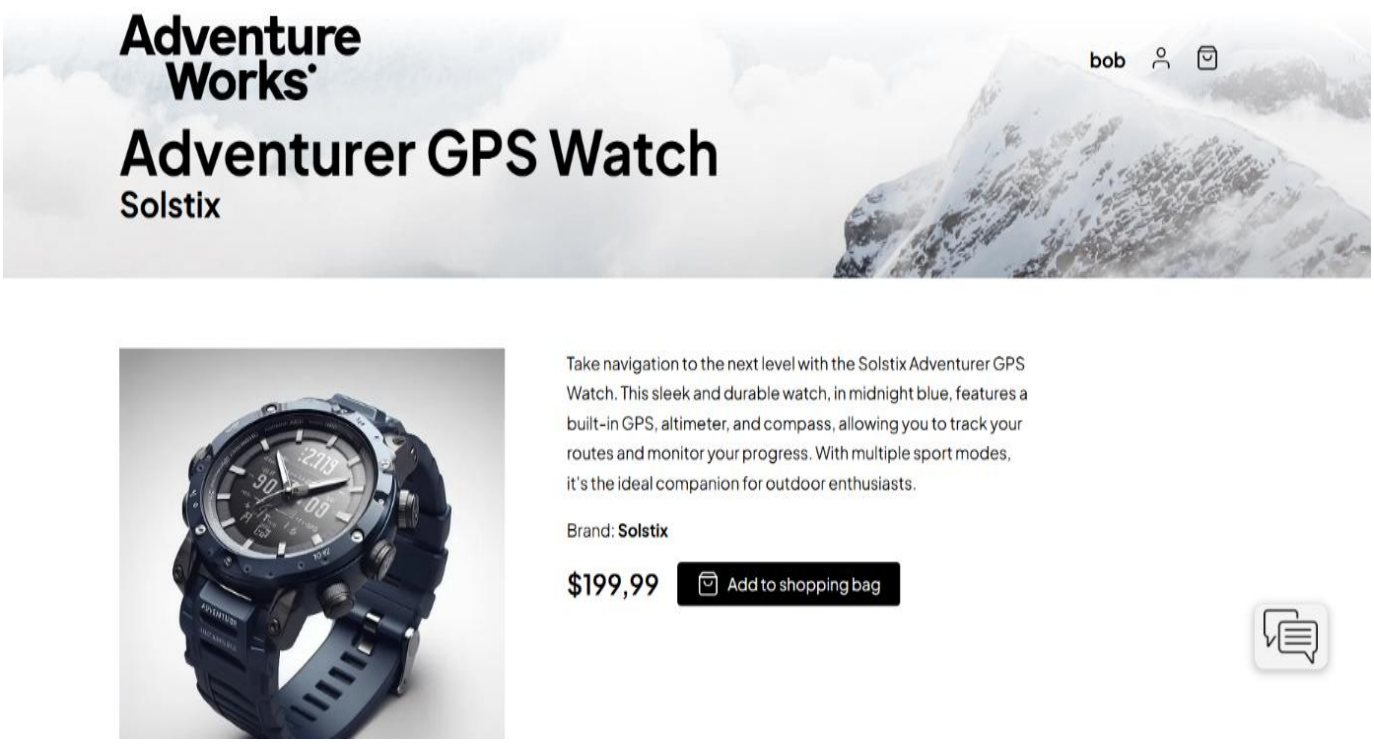


Figure 3.5 : L'interface d'application pour ajouter un produit au panier

Number	Date	Total	Status
1	03/05/2025 10:52:35 PM	$199,99	Paid

Figure 3.6 : L'interface de l'application pour suivi l'état des ordres

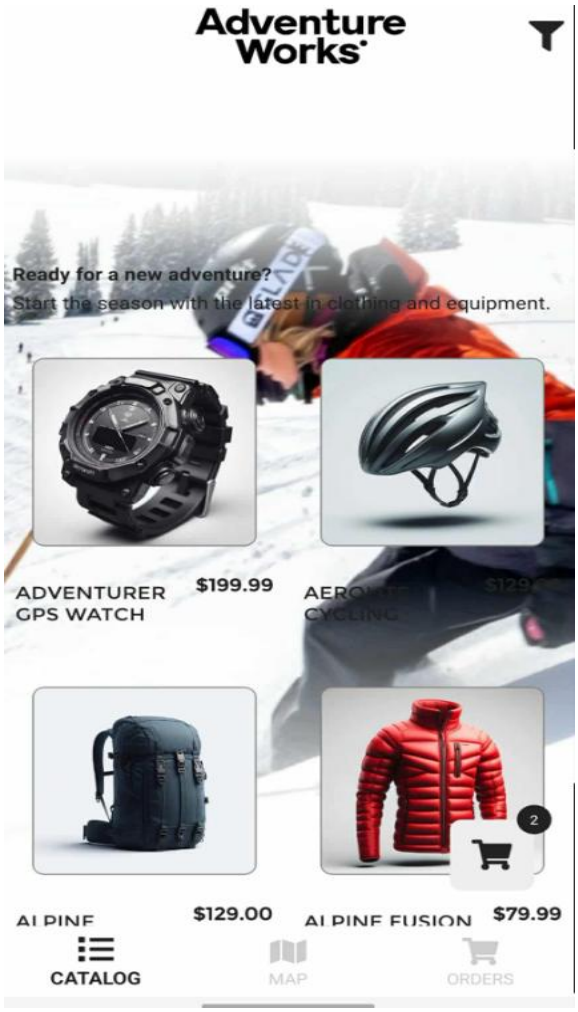


Figure 3.7 : L'interface de l'application mobile

3.3 Modèle de fonctionnalité

Dans cette partie, nous présentons la migration de l'application *eShopOnContainers* vers une LPL, ainsi que les résultats obtenus à l'aide de *Mobioos Forge*. Nous fournissons une vue détaillée du modèle de fonctionnalités conçu durant ce processus. L'application *eShopOnContainers* utilisée dans cette migration compte environ 58 KLOCs (exactement 57 916 lignes).

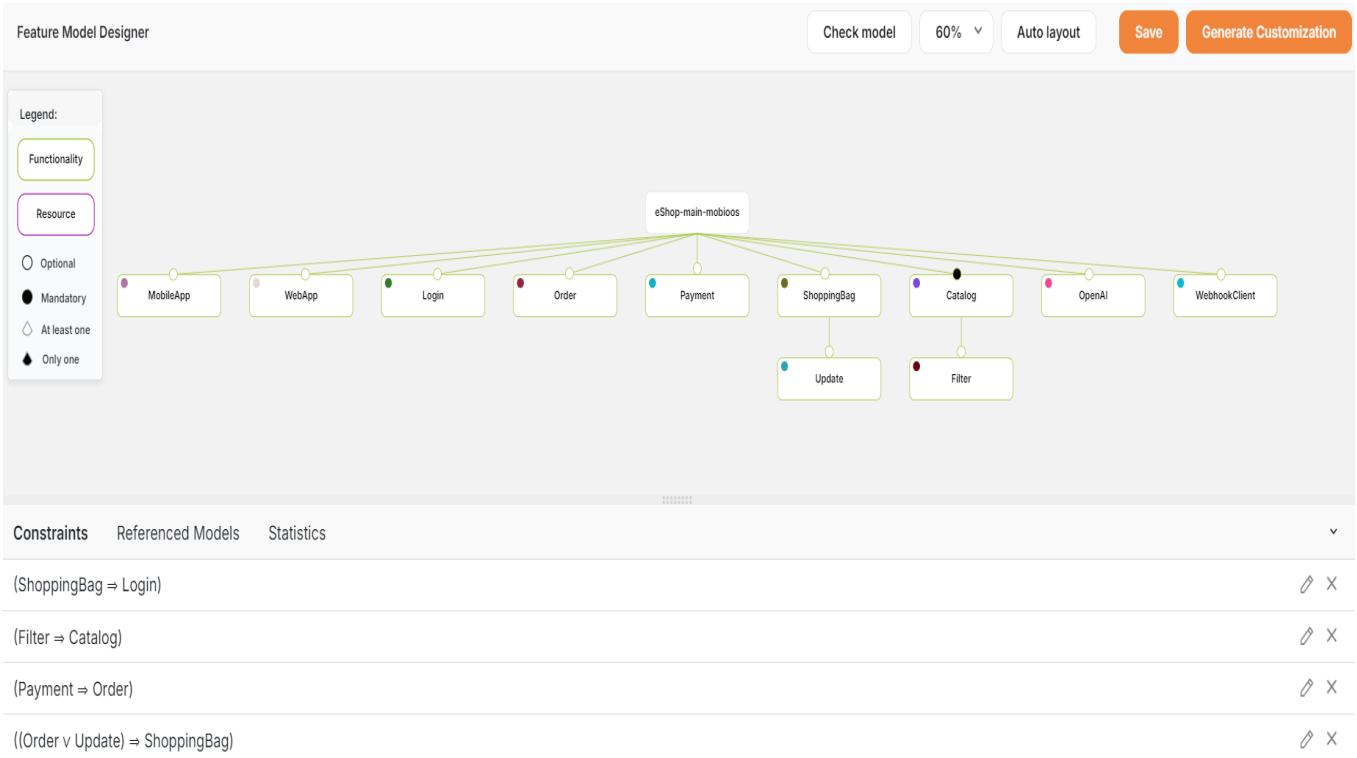


Figure 3.8 : Modèle de fonctionnalités pour l'application *eShopOncontainers*

La Figure 3.8 montre le modèle de fonctionnalités généré pour l'application *eShopOnContainers*. Ce modèle comprend onze fonctionnalités fonctionnelles, avec la fonctionnalité racine nommée *eshop-main-mobioos*. Il inclut une fonctionnalité obligatoire nommé *Catalog* ainsi que les fonctionnalités optionnelles suivantes : *MobileApp, WebApp, Login, Order, Payment, ShoppingBag, OpenAI et WebhookClient*. Certaines fonctionnalités possèdent des sous-fonctionnalités : *ShoppingBag* inclut *Update*, et *Catalog* contient *Filter*.

Le modèle contient également quatre contraintes entre les fonctionnalités, visibles en bas de la figure 3.8. Par exemple, la fonctionnalité *Payment* dépend de la fonctionnalité *Order*. Ces contraintes permettent d'assurer la cohérence les configurations générées.

3.4 Mappage des fonctionnalités de l'application eShopOnContainers

La Figure 3.9 illustre le mappage des fonctionnalités réalisé pour l'application *eShopOnContainers* à l'aide de Mobioos Forge. Cette étape consiste à relier chaque fonctionnalité identifiée dans le modèle à son implémentation dans le code source, à travers des marqueurs qui servent à localiser précisément les morceaux de code liés à une fonctionnalité.

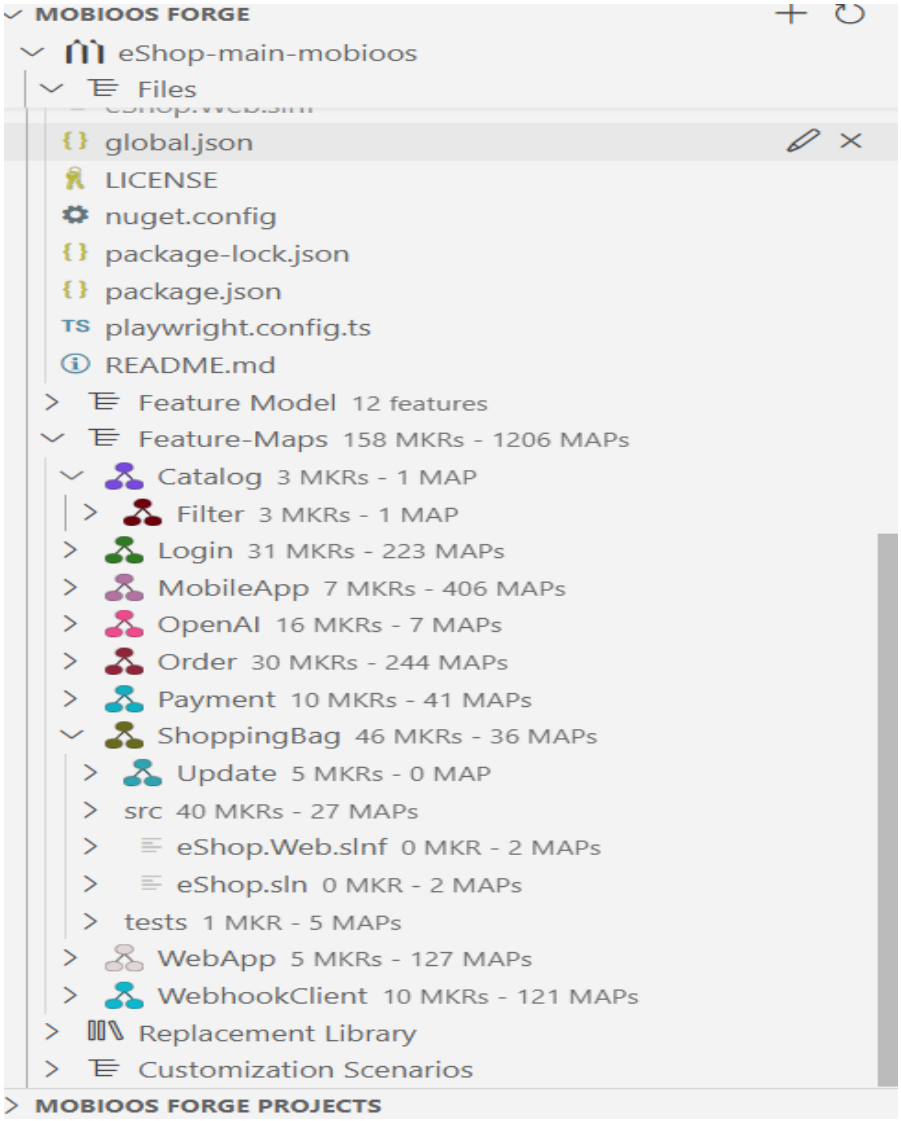


Figure 3.9 : Mappage des fonctionnalités de l'application eShopOnContainers dans Mobioos Forge

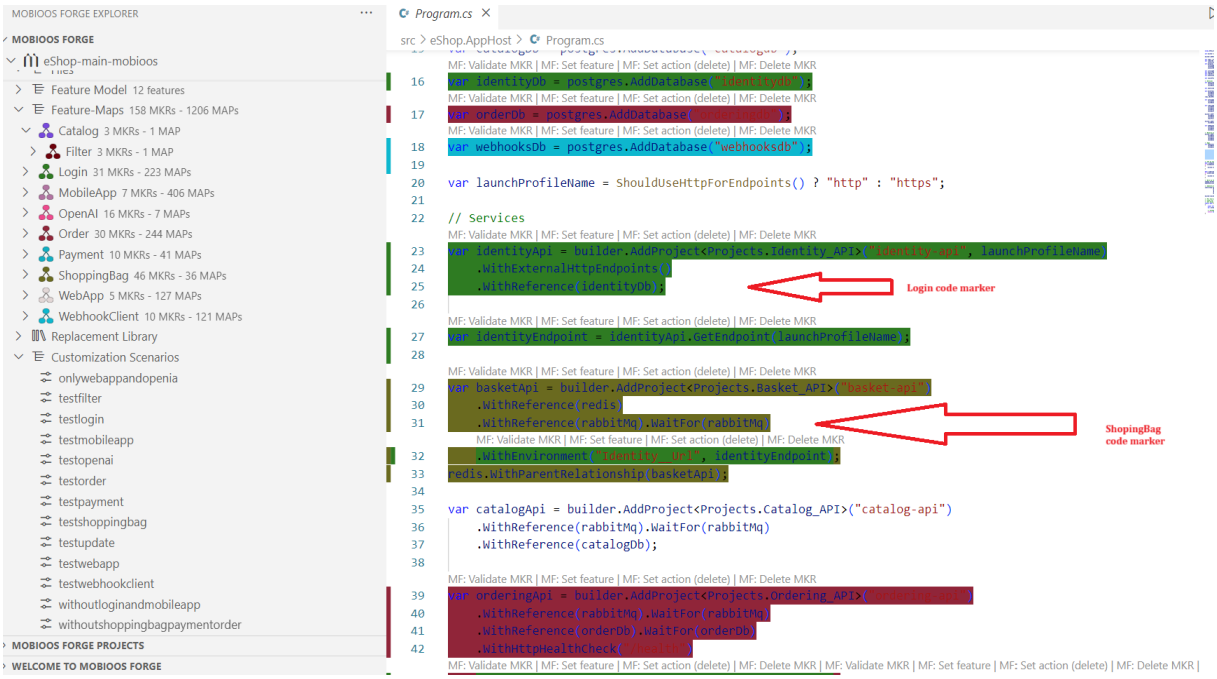


Figure 3.10 : Exemples des marqueurs de quelques fonctionnalités

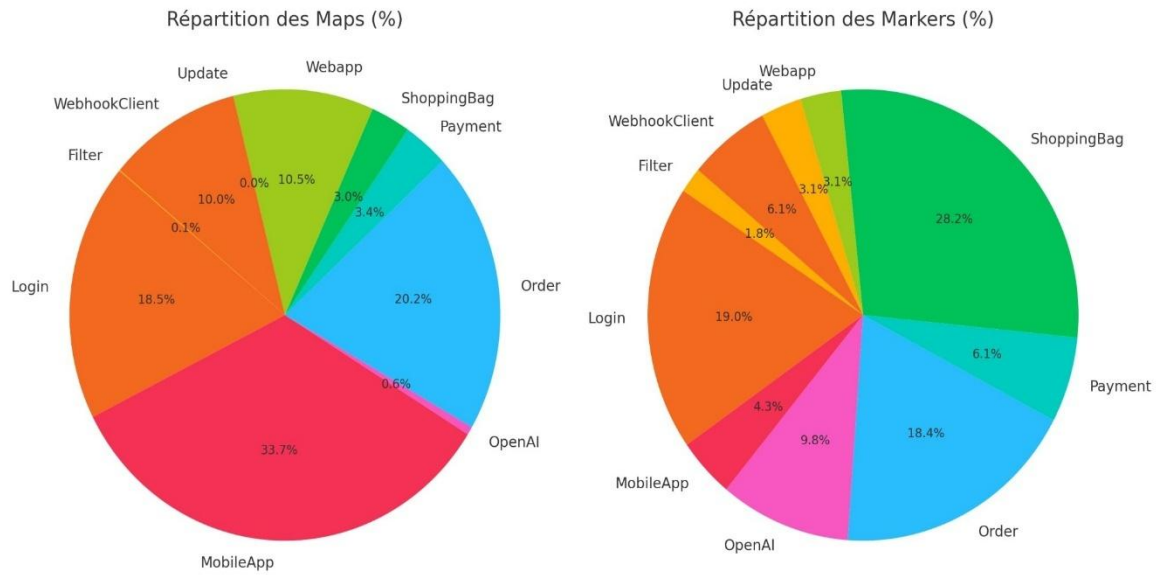


Figure 3.11 : Répartition des fonctionnalités selon le nombre des marqueurs et des maps

Dans notre cas, 158 marqueurs ont été ajoutés dans l'ensemble du projet, ce qui a permis de générer 1206 mappings. Cela signifie que chaque marqueur a été associé à plusieurs points du code, renforçant la traçabilité entre les fonctionnalités et leur implémentation.

Quand on observe la répartition, on remarque que la fonctionnalité *ShoppingBag* est celle qui a demandé le plus de travail de marquage avec 46 markers, ce qui représente près de 29% de

l'ensemble. Cela s'explique par le fait qu'elle est très intégrée dans différents modules de l'application. Viennent ensuite *Order* avec 30 marqueurs, *Login* avec 31, puis *OpenAI* (16 marqueurs) et *MobileApp* (7 marqueurs). D'un autre côté, des fonctionnalités plus simples ou plus isolées comme *Filter* n'ont nécessité que 3 marqueurs et 1 map, et aussi *Update* a été marquée par 5 marqueurs mais aucune map générée. Cela reste logique, vu leur nature ciblée et leur structure relativement stable, avec un impact limité sur l'ensemble du système.

Une analyse plus fine permet de comparer le nombre de maps générées par rapport au nombre des marqueurs pour chaque fonctionnalité. Pour certaines fonctionnalités comme *Login*, *MobileApp*, *OpenAI* et *WebhookClient*, le nombre de maps est significativement supérieur au nombre de marqueurs. Cela indique que chaque marqueur est exploité pour produire plusieurs adaptations ou scénarios. Par exemple, pour la fonctionnalité *WebhookClient*, on observe 10 marqueurs pour 121 maps, soit environ 12,1 maps par marqueur.

À l'inverse, pour d'autres fonctionnalités telles que *Filter*, *ShoppingBag* ou *Order*, le nombre de maps est inférieur ou équivalent au nombre des marqueurs. Cela signifie que les marqueurs ont été utilisés de manière plus ciblée, avec moins de variations associées. Par exemple, la fonctionnalité *ShoppingBag* présente 46 marqueurs pour seulement 36 maps.

Tableau 3.1 : Métriques de Fonctionnalités dans eShopOnContainers LPL

Feature	LOCS	Pourcentage de LOCS mappées	Fichiers impactés	Maps/Marker
Login	**26631**	**45.20%**	**124**	**7.19**
WebAPP	**1634**	**2.77%**	**40**	**25.4**
MobileAPP	**14917**	**25.31%**	**255**	**58**
Order	**5870**	**9.96%**	**150**	**8.13**
Payment	**382**	**0.64%**	**25**	**4.2**
OpenAI	**55**	**0.09%**	**14**	**0.43**
ShoppingBag	**1003**	**1.70%**	**28**	**0.78**
WebhookClient	**1217**	**2.06%**	**83**	**12.1**
Filter	**162**	**0.27%**	**4**	**0.33**
Update	**18**	**0.03%**	**1**	**0**
Catalog	**7027**	**11.92%**	**127**	**0**

Le Tableau 3.1 présente les métriques liées au mappage des fonctionnalités dans l'application eShopOnContainers LPL. Pour chaque fonctionnalité, on indique le nombre de *LOCs*, le pourcentage de *LOCs* mappées, le nombre de fichiers impactés, ainsi que le ratio Maps/Marker.

En analysant ce tableau, on observe que les fonctionnalités couvrent entre 0.03 % et 45.20 % du total des LOCs mappées. La fonctionnalité *Login* reste la plus dominante, avec 45.20 % des LOCs mappées réparties sur 358 fichiers, ce qui en fait la fonctionnalité la plus volumineuse et la plus intégrée du système. À l'inverse, les fonctionnalités *Update* (0.03 %), *OpenAI* (0.09 %) et *Filter* (0.27 %) présentent des taux très faibles de LOCs mappées, avec très peu de fichiers impactés (1 à 4 fichiers), ce qui suggère une présence minimale et probablement un couplage faible avec le reste du système.

En termes de fichiers impactés, les fonctionnalités *Login*, *MobileAPP* (269 fichiers) et *ShoppingBag* (235 fichiers) sont les plus répandues dans le système.

Concernant le ratio Maps/Marker, les fonctionnalités *MobileAPP* (58), *WebAPP* (25.4) et *WebhookClient* (12.1) se distinguent par une forte densité de mappage via des maps. À l'inverse, *Filter* (0.33), *OpenAI* (0.43) et *ShoppingBag* (0.78) présentent des ratios très bas, ce qui signifie que leur présence est surtout représentée par des marqueurs.

Enfin, bien que la fonctionnalité *Catalog* représente 11.92 % des LOCs mappées réparties sur 127 fichiers, elle est obligatoire dans toutes les variantes générées.

3.5 Exemples des variantes dérivées

Le Tableau 3.2 présente les métriques de quatre variantes générées à partir de l'application eShopOnContainers LPL, ainsi que de l'application originale. Chaque variante est définie par les fonctionnalités activées, avec le nombre total de LOCs générées et les LOCs supprimées.

Tableau 3.2 : Métriques de variantes générées de eShopOnContainers LPL

Variantes (fonctionnalités activées)	LOCS	LOCS supprimées
Toutes les fonctionnalités (application originale)	**58916**	**0**
Test 1 : Sans login	**32285**	**26631**
Test 2 : Sans login , mobileapp	**17368**	**41548**
Test 3 : Sans shoppingbag,order,payment	**51661**	**7255**
Test 4 : Seulement webapp,openai	**8716**	**50200**
Test 5 : Seulement webapp,login,filter	**34924**	**23992**

La variante Test 4

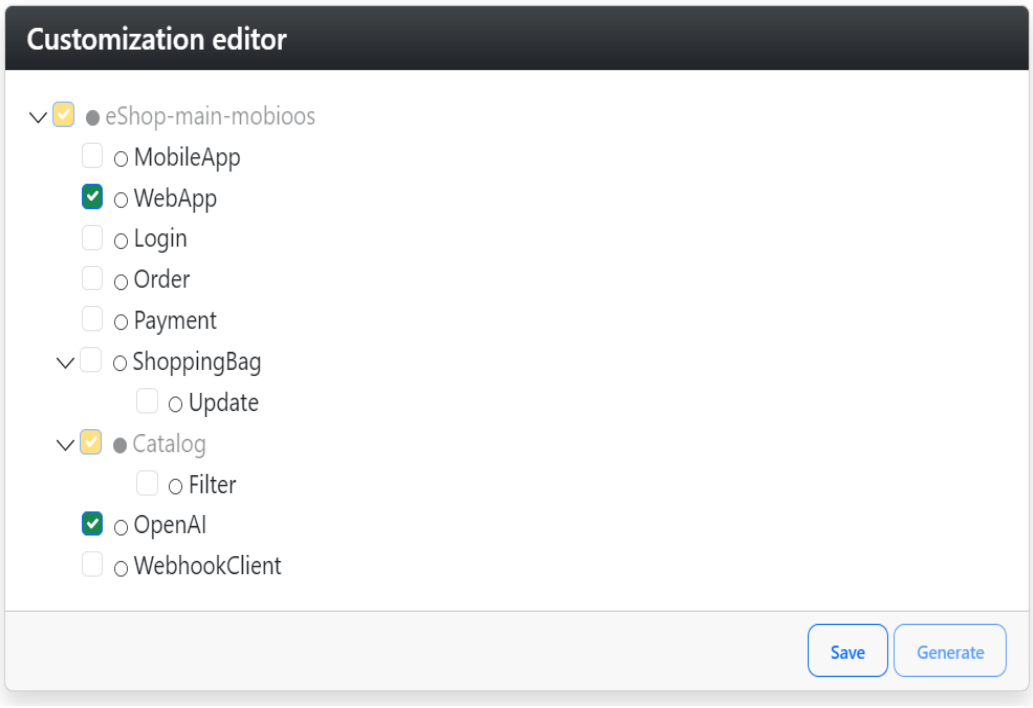


Figure 3.12 : Configuration de la variante 4

La Figure 3.12 montre la configuration de la variante 4 (sans : *MobileApp* , *Login*, *Order* , *Payment* , *ShoppingBag* , *Update* ,*Filter* et *WebhookClient*). Dans cette personnalisation, on peut voir que seules les fonctionnalités suivantes sont activées : *WebApp* et *OpenAI.*

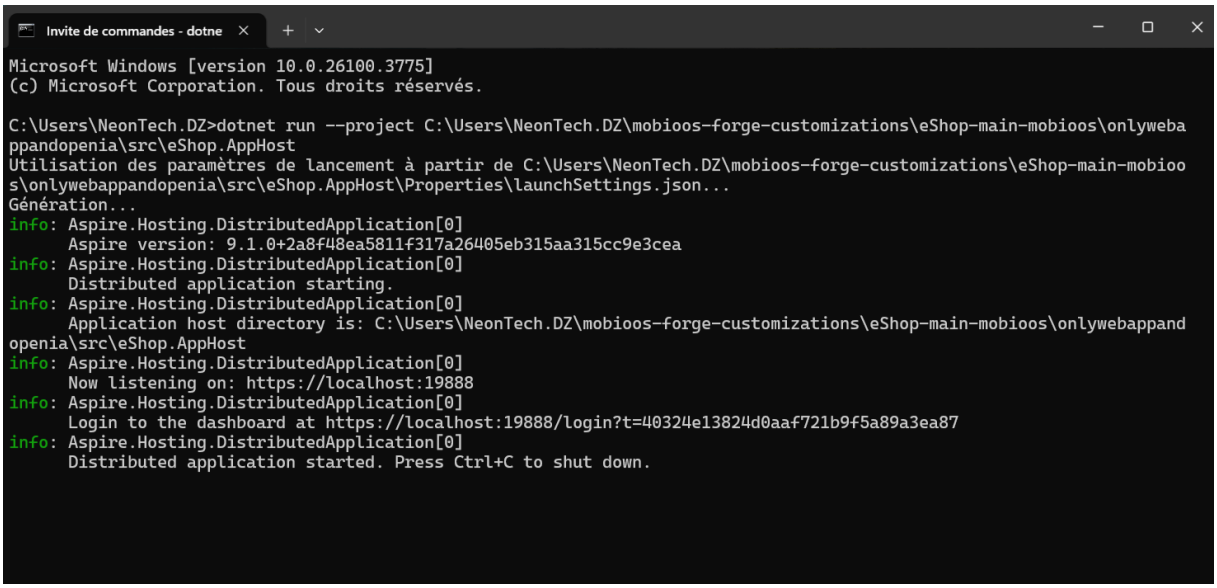


Figure 3.13 : Exécution la variante 4 en utilisant la commande *dotnet run*

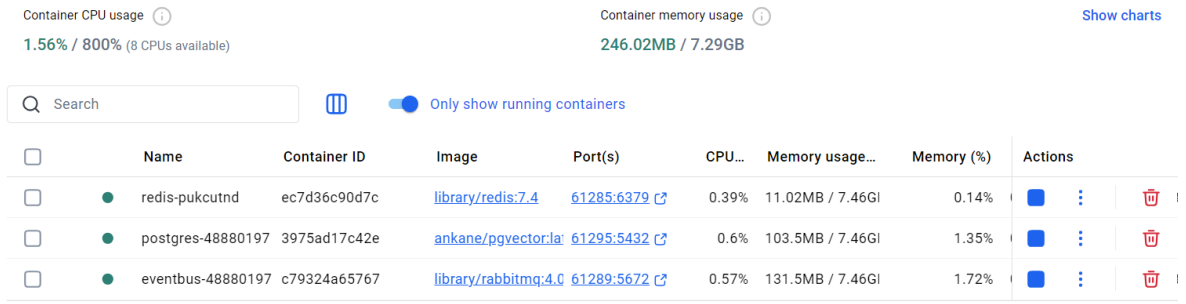


Figure 3.14 : Les conteneurs de la variante 4 dans Docker Desktop

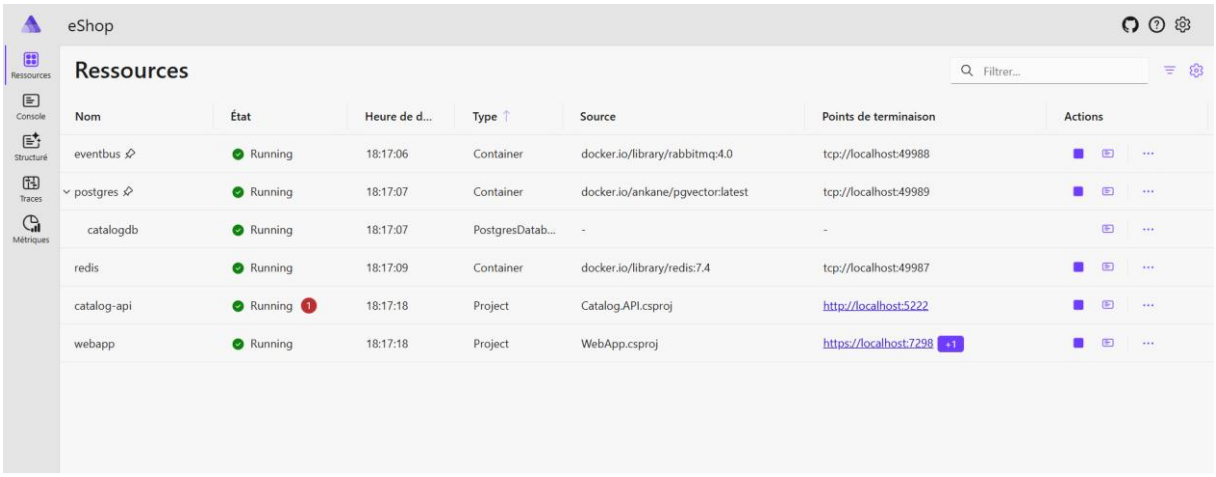


Figure 3.15 : Les services de la variante 4 dans .Net Aspire

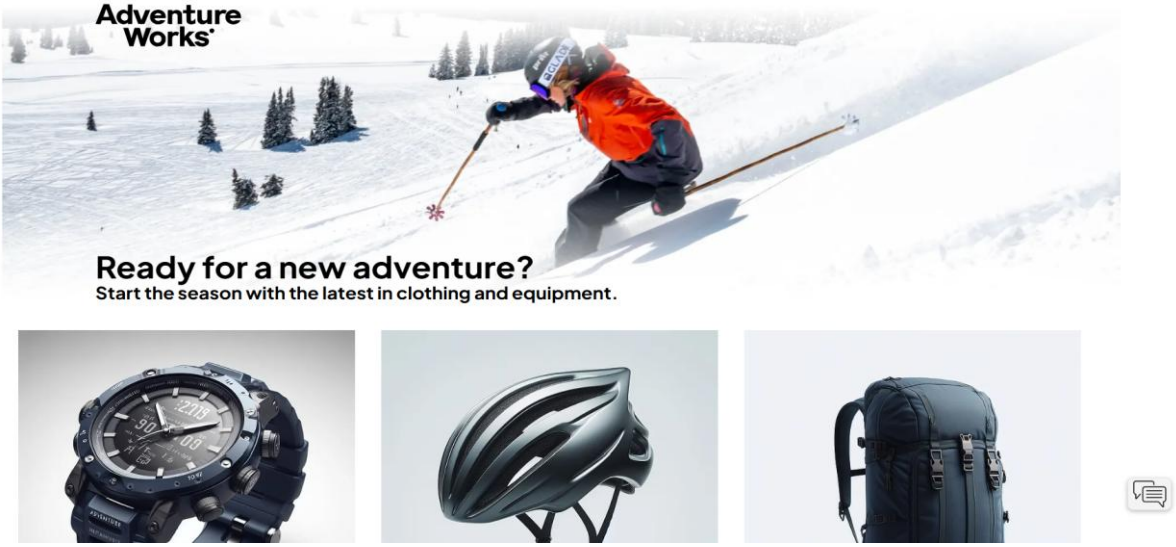


Figure 3.16 : Interface de la page d'accueil de la variante 4 générée

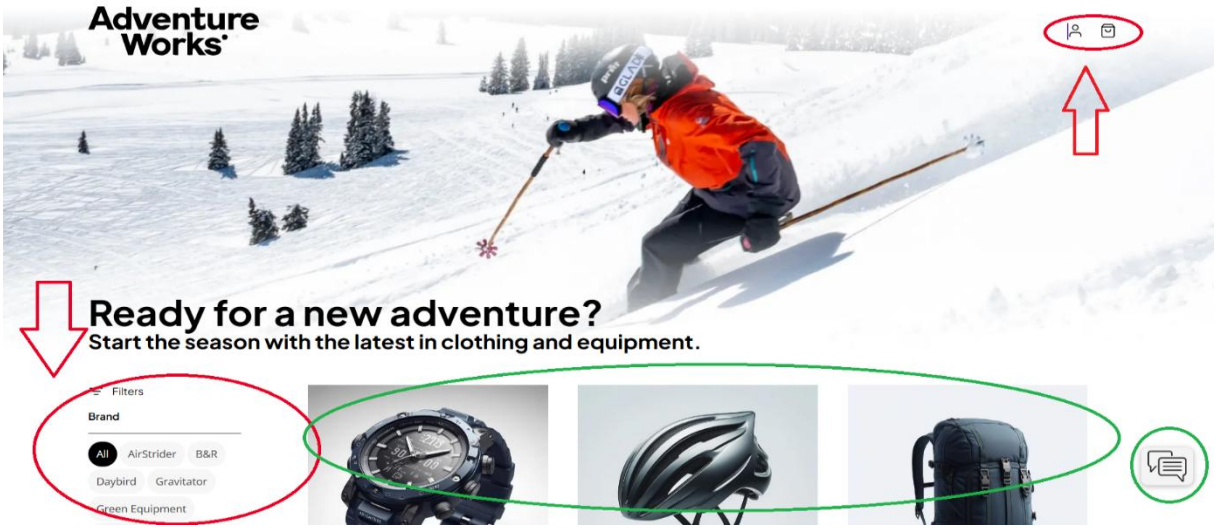


Figure 3.17 : Interface de la page d'accueil de l'application

Pour exécuter la variante 4 générée de l'application eShopOnContainers, nous lançons l'application depuis la ligne de commande en utilisant la commande *dotnet run*, comme illustré dans la figure 3.13. Cette commande permet de démarrer l'application à partir du chemin de projet spécifié. Une fois le lancement effectué, nous pouvons observer les journaux de démarrage dans le terminal, confirmant que l'application a été démarrée avec succès (figure 3.13).

En parallèle, les services nécessaires au bon fonctionnement de l'application, notamment *Redis, PostgreSQL et RabbitMQ*, sont exécutés dans *Docker* comme montré dans la figure 3.14.

Ensuite, nous pouvons accéder à l'espace de supervision de *.NET Aspire* via l'URL https://localhost:19888. La figure 3.15 montre cet espace où l'on visualise les différentes ressources en cours d'exécution. Parmi elles, on retrouve une ressource nommée *webapp* associée à l'adresse https://localhost:7298, qui permet d'accéder à l'interface utilisateur de l'application web.

La figure 3.16 présente l'interface de la page d'accueil de la variante 4 déployée sur notre serveur local. Dans cette version, nous remarquons que seules les fonctionnalités *WebApp* et *OpenIA* ont été conservées. La figure 3.17 montre l'interface de l'application Web originale. On y distingue, entourés en vert, les éléments qui ont été conservés dans la figure 3.17, à savoir le catalogue de produits ainsi que l'accès à *OpenIA* (icône de messagerie en bas à droite). En

revanche, les éléments entourés en rouge (le filtre de marques (*filter*), l'icône de connexion utilisateur (*login*) et le panier d'achat (*shoppingbag*) ont été supprimés dans la variante 4.

La variante Test 5

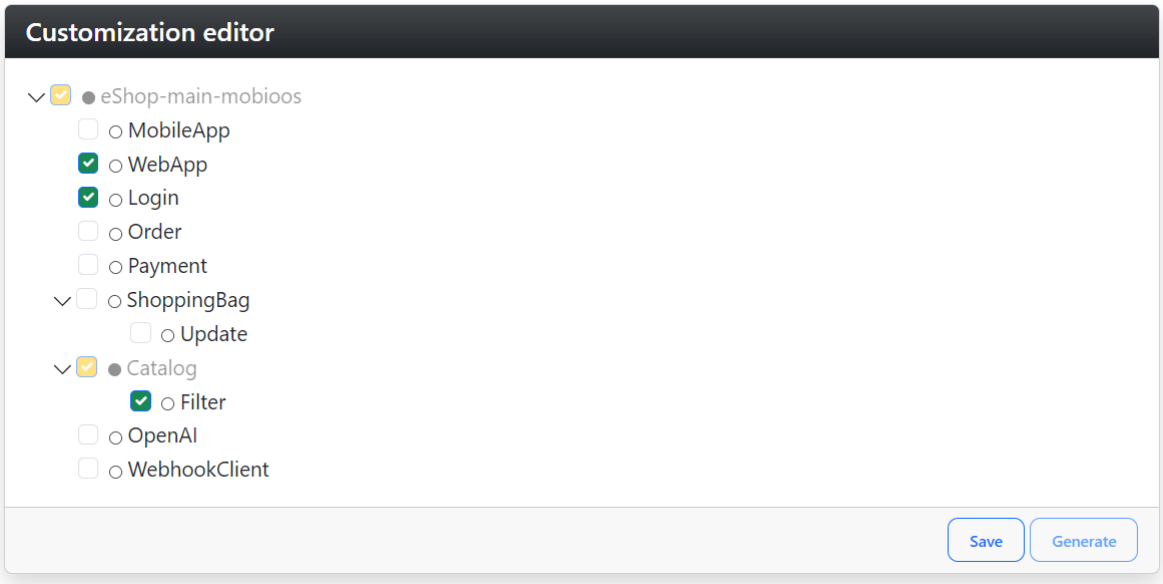


Figure 3.18 : Configuration de la variante 5

La figure 3.18 montre la configuration de la variante 5 (sans *MobileApp, Order, Payment, ShoppingBag, Update, OpenIA, WebhookClient*), on peut voir que seules les fonctionnalités suivantes sont activées : *WebApp* et *Login,Filter*.

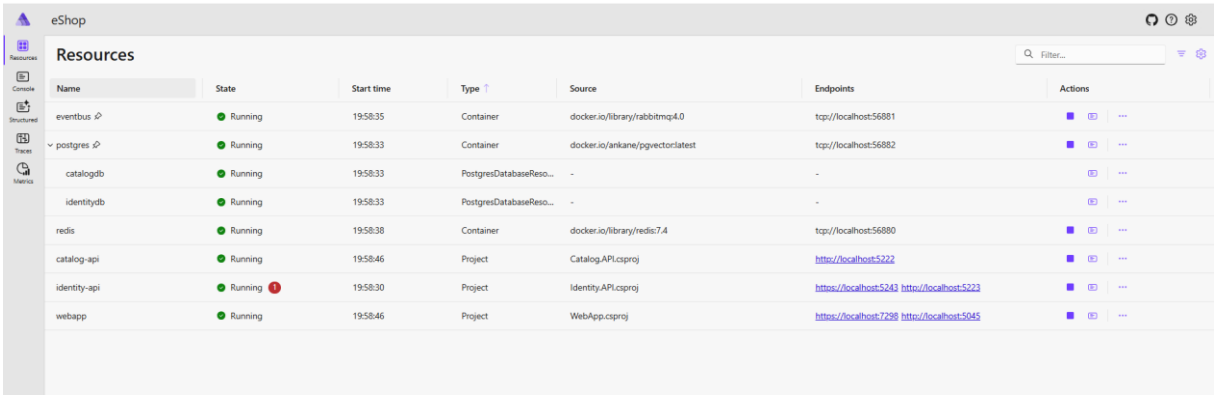


Figure 3.19 : Les services de la variante 5 dans .Net Aspire

La figure 3.20 présente l'interface de la page d'accueil de la variante 5, on voit que l'interface contient l'icône de connexion (*login*) ainsi que le *filtre* des produits (les éléments entourés en

vert), mais ne contient ni l'icône d'*OpenIA*, ni celle du panier d'achat (*shoppingbag*) (espace en rouge).

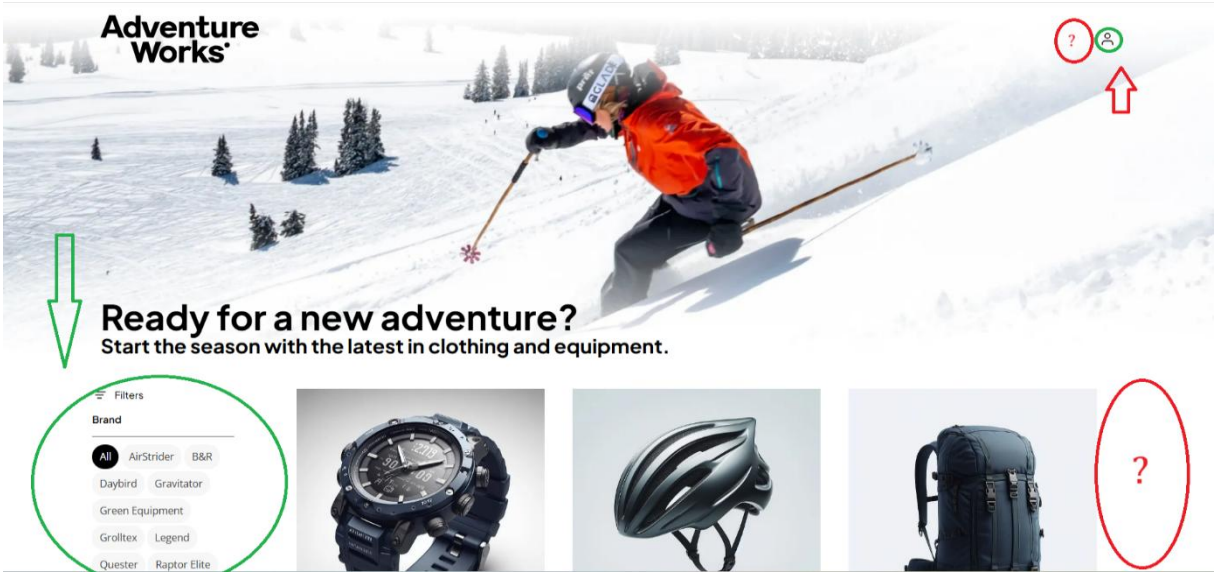


Figure 3.20 : Interface de la page d'accueil de la variante 5 généré

3.6 Calcul du temps de migration

La figure 3.21 illustre, à travers un histogramme, le temps nécessaire pour mapper chaque fonctionnalité de l'application, avec une courbe superposée représentant le nombre de maps validées manuellement.

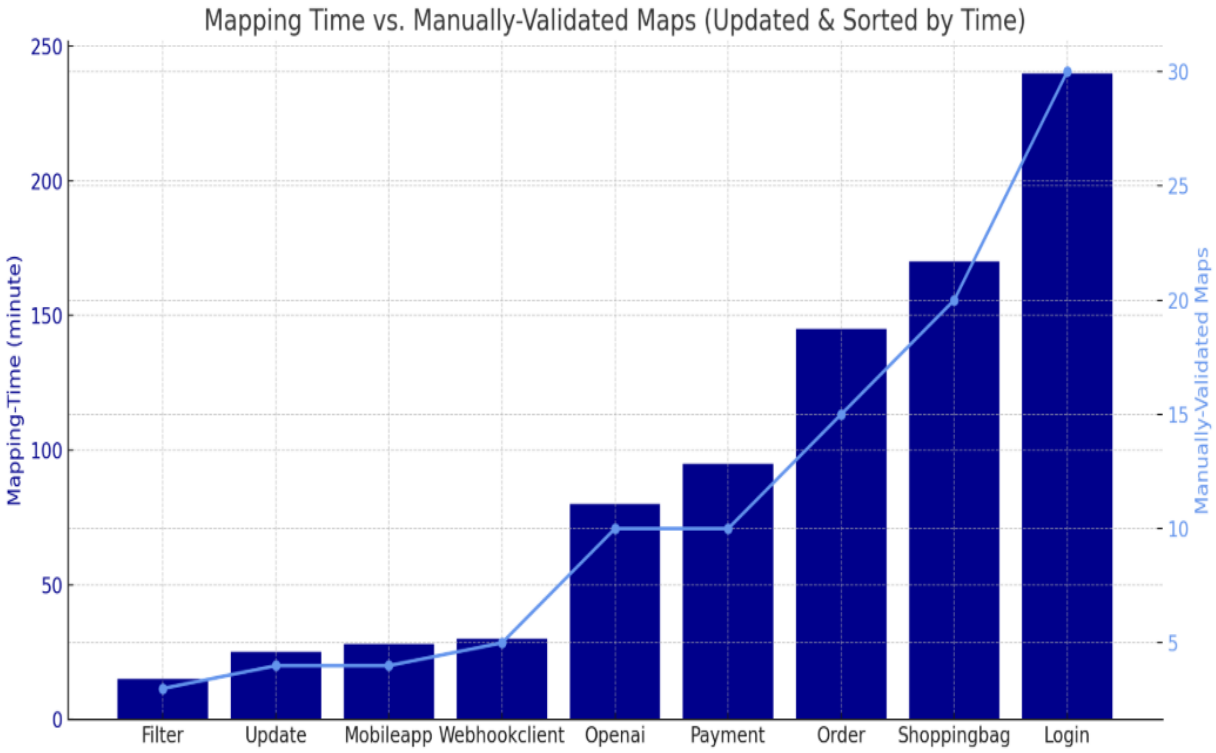


Figure 3.21 : Temps de mapping et nombre de maps validées manuellement de l'application eShopOnContainers LPL

On remarque que la majorité des fonctionnalités ont été mappées assez rapidement : *Filter*, *Update* et *Webhookclient* n'ont pris que 30 minutes ou moins, tandis que *Mobileapp* et *Openai* ont demandé un peu plus de temps, mais restent sous la barre des 1h30.

Par contre, certaines fonctionnalités se distinguent par des durées bien plus longues. *Order* et *Payment* nécessitent entre 1h30 et 2h30 de travail, tandis que *Shoppingbag* monte à près de 3 heures. La fonctionnalité *Login*, quant à elle, atteint les 4 heures de mapping « la plus longue durée observée ».

Ces cas correspondent justement à celles qui comptent le plus de maps validées manuellement, ce qui semble indiquer une relation directe entre la complexité du mapping et le volume de validations nécessaires.

Par exemple, *Login* cumule 30 maps, Shoppingbag en a 20, et Order en compte 15. Ce graphique met donc en lumière un lien logique : **plus une fonctionnalité nécessite de maps validées, plus son mapping est long, ce qui reflète probablement sa complexité fonctionnelle ou son importance dans l'application**.

3.7 Conclusion

Dans ce chapitre, nous avons présenté l'application *eShopOnContainers,* développée selon une architecture Microservices. Ensuite, nous avons introduit le modèle de fonctionnalité afin d'identifier les points de variabilité, puis réalisé le mappage des fonctionnalités entre le modèle et l'application. À partir de cela, nous avons généré quelques exemples de variantes dérivées, illustrant la transformation vers une ligne de produits logiciels. Enfin, nous avons calculé le temps de migration, pour mesurer l'effort requis et les bénéfices obtenus.

4

Migration de l'application YAS vers Les lignes de produits logiciels

4.1 Introduction

Dans ce chapitre, nous présentons la migration de l'application Microservices open source YAS vers une LPL. Cette application e-commerce utilise un ensemble de technologies actuelles telles que *Java*, *Spring Boot*, *Next.js*, *Keycloak, Kafka, PostgreSQL*, *Docker* et *Kubernetes*. En plus, elle offre une instrumentation complète via *OpenTelemetry*, *Grafana, Prometheus* et *Loki*.

Nous présenterons tout d'abord l'architecture de cette application, avant d'expliquer comment elle a été transformée en LPL à l'aide de la plateforme Mobioos Forge, en passant par la modélisation des fonctionnalités, l'identification des marqueurs et des maps, la génération d'exemples de variantes dérivées, l'évaluation du temps de migration, puis nous terminerons par une comparaison avec la migration de l'application eShop.

4.2 Présentation de l'application YAS

YAS (*Yet Another Shop*) [25] est une application e-commerce open source sur *GitHub*, conçue pour illustrer la mise en œuvre d'une architecture Microservices moderne. Elle intègre plusieurs technologies récentes telles que J*ava, Spring Boot, Next.js, Kafka, PostgreSQL, Docker, Kubernetes, Keycloak* pour l'authentification, ainsi que *OpenTelemetry, Prometheus, Grafana, Loki* et *Tempo* pour l'observabilité. L'architecture de L'application YAS repose sur une organisation modulaire composée de plusieurs Microservices métiers répartis par domaine fonctionnel. Ces services communiquent via des API REST et un bus d'événements basé sur Kafka. Le système intègre également une interface de gestion pour les administrateurs (*Backoffice*), une interface client (*Storefront*), et un ensemble de services techniques pour le monitoring, la recherche et l'intégration continue.

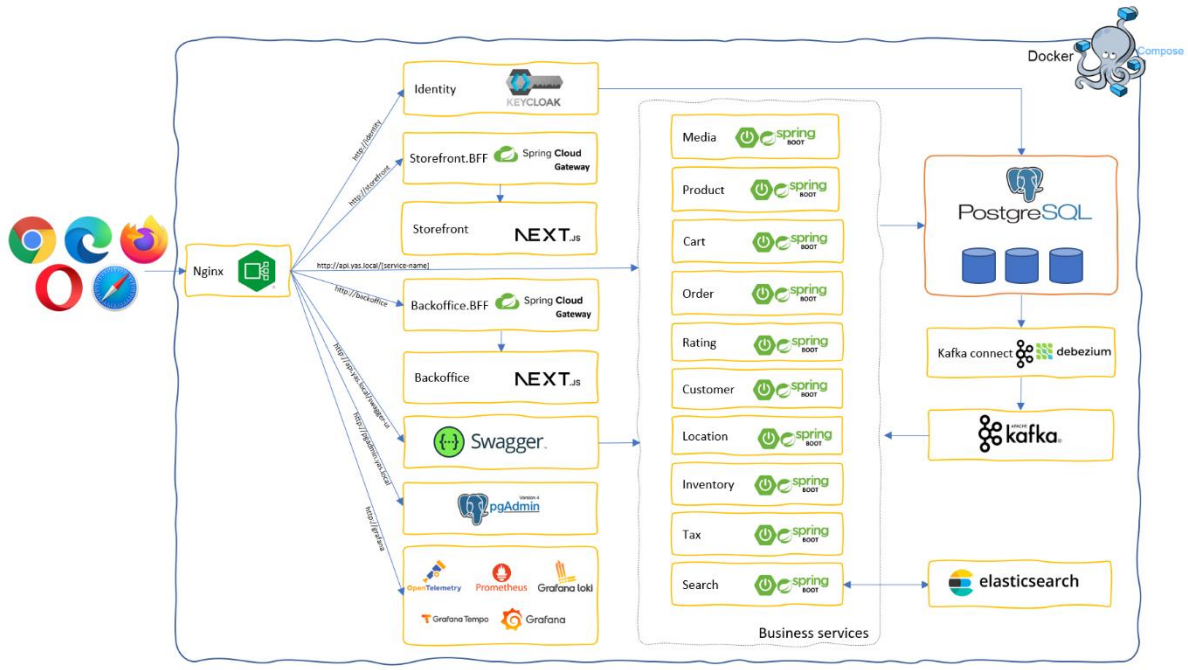


Figure 4.1 : Architecture basée sur les Microservices de l'application YAS

L'architecture globale de l'application YAS (figure 4.1) se compose des éléments suivants [25] :

Interface Utilisateur

- ***storefront :*** Interface web orientée client (Next.js) pour la navigation, l'achat de produits, la gestion du panier, etc.
- ***backoffice :*** Interface d'administration pour la gestion des produits, commandes, clients, promotions, etc.

Frontaux BFF (Backend For Frontend)

- ***storefront-bff :*** Sert de passerelle entre les microservices métiers et l'interface Storefront.
- ***backoffice-bff :*** Sert de passerelle entre les microservices métiers et l'interface Backoffice.

Services Métier

- ***produit*** : Gère les catalogues produits, leurs détails, catégories, attributs, etc.
- ***inventory :*** Gère les niveaux de stock, les emplacements d'entrepôt, les disponibilités.
- ***order :*** Gère le cycle de vie des commandes (création, modification, statut).
- ***cart :*** Gère le panier d'achat des utilisateurs.
- ***customer :*** Gère les informations des clients, leurs adresses, préférences, etc.
- ***payment :*** Gère les paiements (avec prise en charge de plusieurs moyens).
- ***payment-paypal*** : Service spécifique pour le paiement via PayPal.

- ***promotion :*** Gère les réductions, coupons et autres offres commerciales.
- ***rating :*** Gère les avis et évaluations des produits.
- ***search :*** Fournit des capacités de recherche (connecté à Elasticsearch).
- ***location :*** Gère les données géographiques (villes, pays, etc.).
- ***media :*** Gère les images et fichiers médias (ex. photos de produits).
- ***tax :*** Gère les calculs de taxes selon la localisation.

Infrastructure et Support

- ***identity :*** Gère l'authentification et l'autorisation des utilisateurs via Keycloak.
- ***webhook :*** Fournit une interface pour interagir avec des services externes via des webhooks.
- ***recommendation :*** Fournit des recommandations de produits basées sur les comportements des utilisateurs.

Observabilité et Monitoring

- ***tempo-data :*** Fournit le traçage distribué avec Tempo.
- ***grafana / loki / prometheus :*** Collecte et visualisation des métriques, logs et traces pour tous les services.
- ***OpenTelemetry :*** Utilisé pour instrumenter les services en traçabilité.

Outils et Scripts

- ***sampledata :*** Permet de précharger des données fictives (produits, etc.) pour les tests/démos.
- ***scripts/postman :*** Fichiers pour tester les API via Postman.
- ***automation-ui :*** Utilisé pour l'automatisation des tests d'interface utilisateur.

Communication et Intégration

- ***kafka/connects :*** Infrastructure de message basée sur Apache Kafka pour la communication entre services.
- ***nginx :*** Reverse proxy pour rediriger les requêtes vers les bons services.

Base de données

- ***PostegreSQL:*** système de gestion de base de données relationnelle.

Recherche

- ***ElasticSearch:*** moteur de recherche distribué. Les données des services peuvent y être indexées pour recherche rapide.

Nous disposons ici de 26 conteneurs représentant les différents services de notre application *eShopOnContainers*, comme illustré dans les figures (4.2-4.5) ci-dessous :

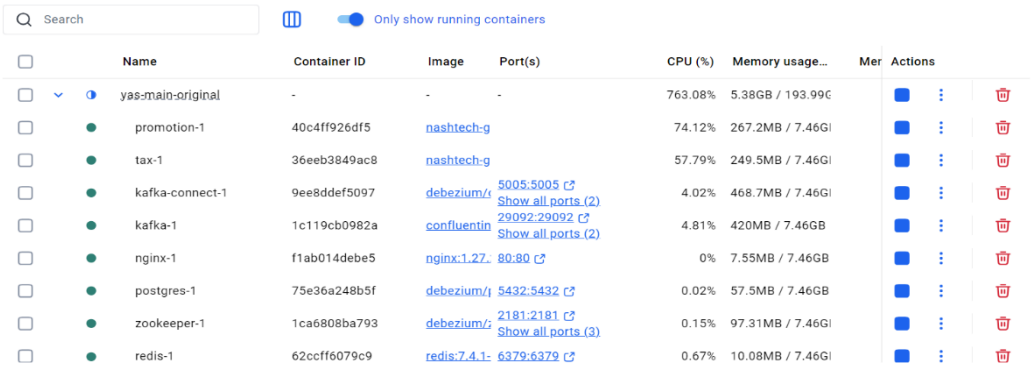


Figure 4.2 : Exécution des conteneurs de l'application YAS dans Docker (partie1)

Figure 4.3 : Exécution des conteneurs de l'application YAS dans Docker (partie2)

Figure 4.4 : Exécution des conteneurs de l'application YAS dans Docker (partie3)

Figure 4.5 : Exécution des conteneurs de l'application YAS dans Docker (partie4)

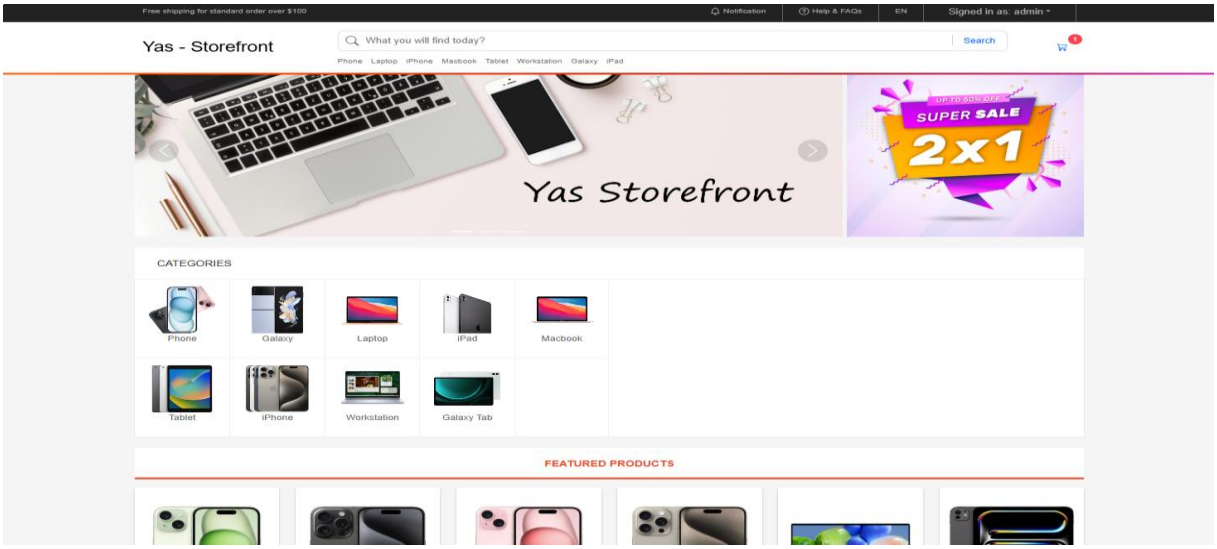


Figure 4.6 : L'interface utilisateur de la page d'accueil

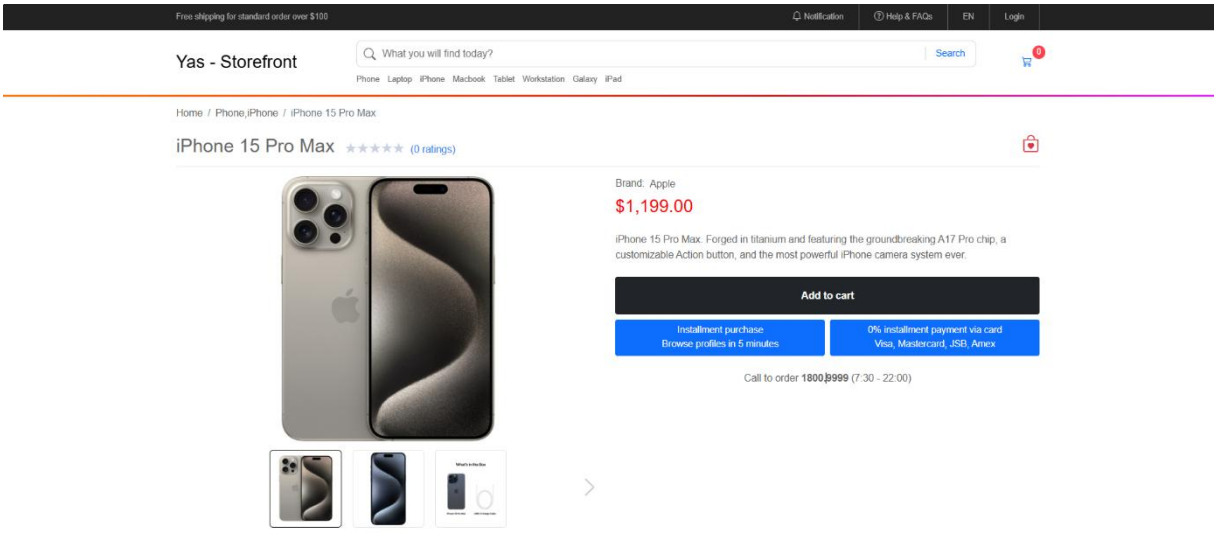


Figure 4.7 : L'interface utilisateur pour ajouter un produit

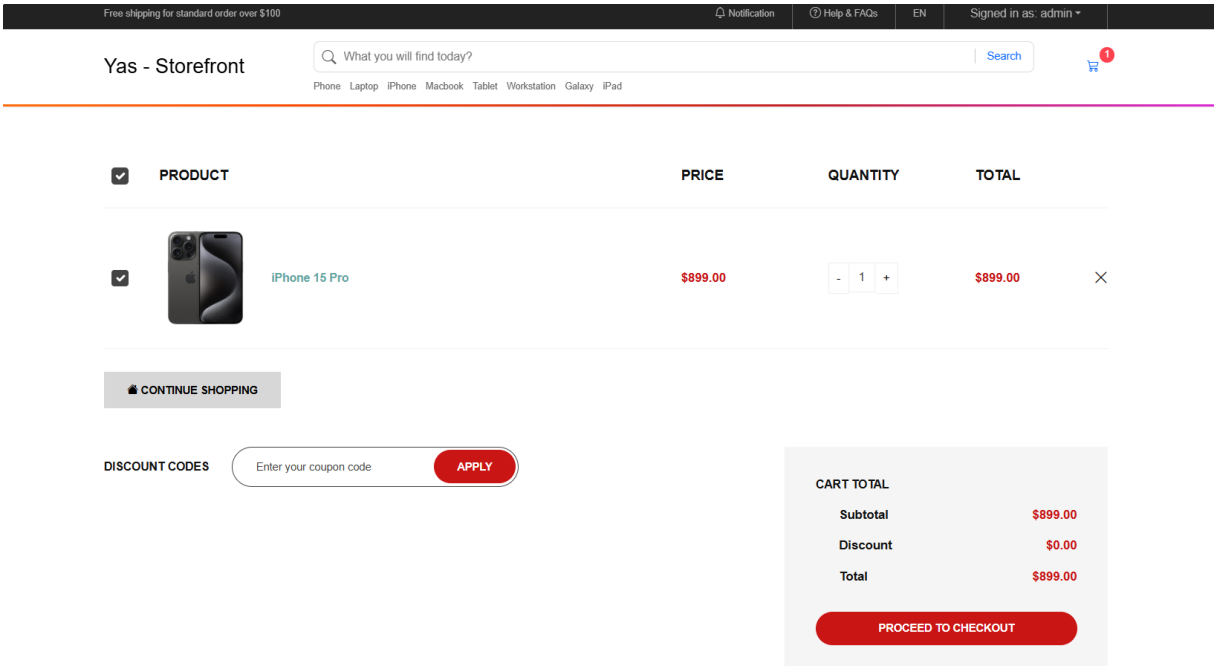


Figure 4.8 : L'interface utilisateur pour les produits ajoutés dans la basket

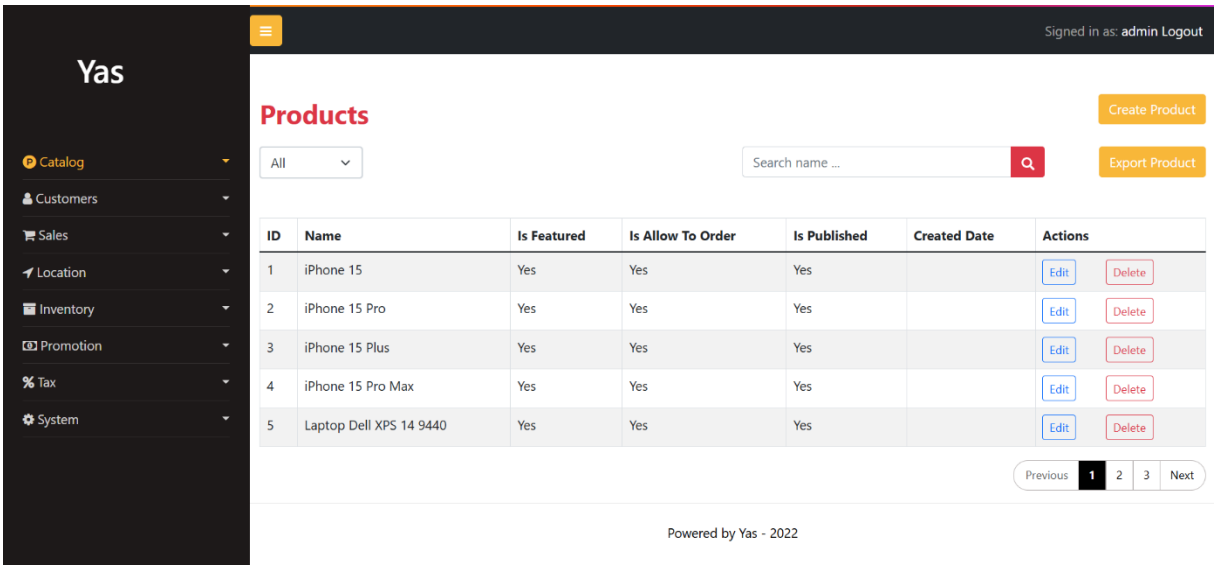


Figure 4.9 : L'interface administrateur

4.3 Modèle de fonctionnalités

Dans cette partie, nous présentons la migration de l'application YAS vers une LPL, ainsi que les résultats obtenus à l'aide de Mobioos Forge. Nous fournissons une vue détaillée du modèle de fonctionnalités conçu durant ce processus. L'application YAS utilisée dans cette migration compte environ 130 KLOCs (exactement 133 851 lignes).

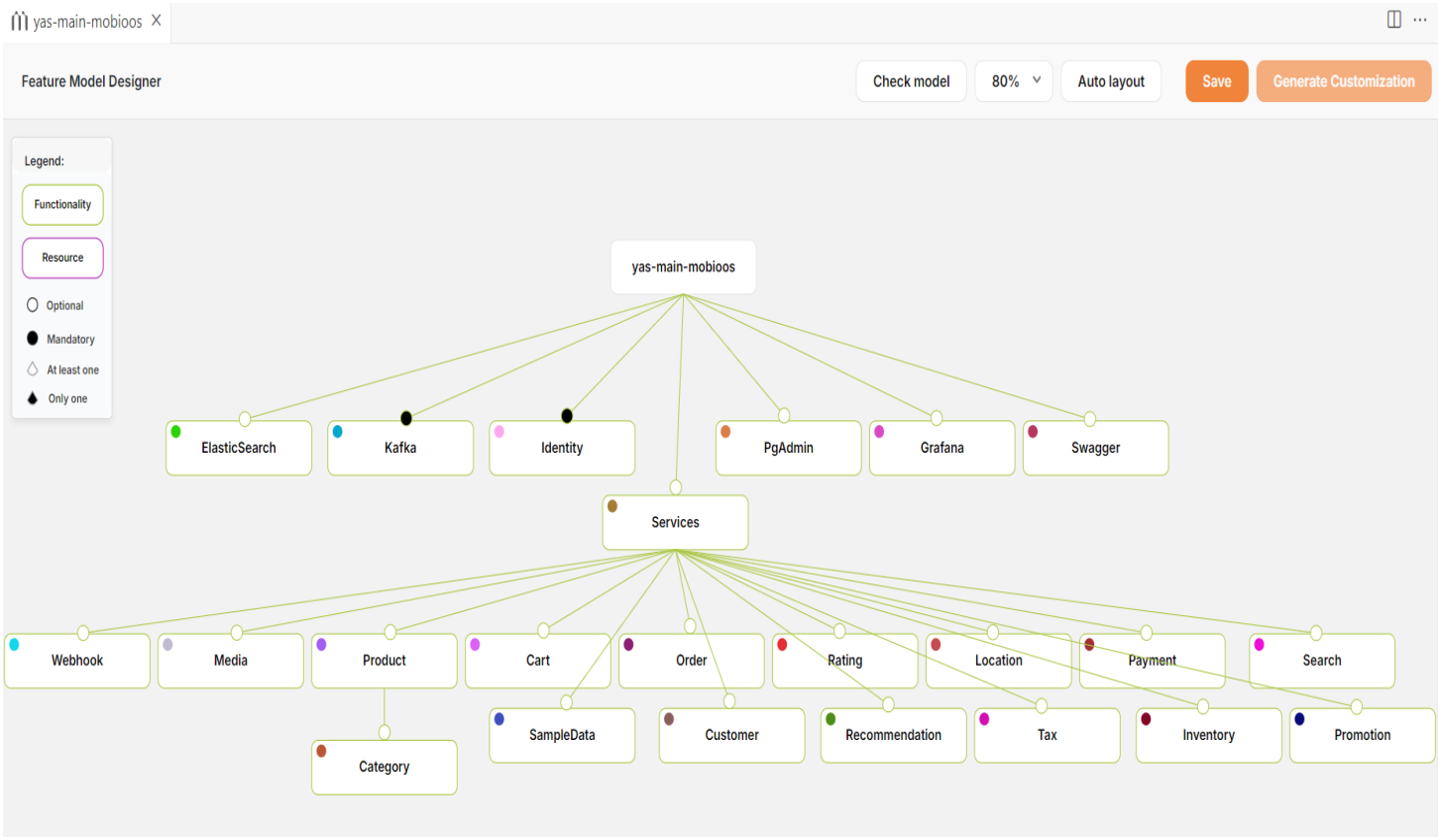


Figure 4.10 : Modèle de fonctionnalités pour l'application YAS

La Figure 4.10 montre le modèle de fonctionnalités généré pour l'application YAS. Ce modèle comprend vingt-trois fonctionnalités fonctionnelles, avec une fonctionnalité racine nommée *yas-main-mobioos*. Il inclut les deux fonctionnalités obligatoire suivantes : *Kafka*, *Identity* ainsi que les fonctionnalités optionnelles suivantes : *ElasticSearch, Swagger, PgAdmin, Grafana et Services* ce dernier possèdes les sous fonctionnalités suivantes : *Media , webhook, Category, Order, Payment, Rating, Cart, Location, Inventory, Tax, Promotion, Recommendation, SampleData, Search, Customer* et *Product* qui inclut elle-même une sous-fonctionnalité : *Category*.

Le modèle comprend également huit contraintes croisées entre les fonctionnalités, montrées dans la figure 4.11. Par exemple, la fonctionnalité *Tax* dépend de fonctionnalité *Location.* Ces contraintes permettent d'assurer la cohérence des configurations générées.

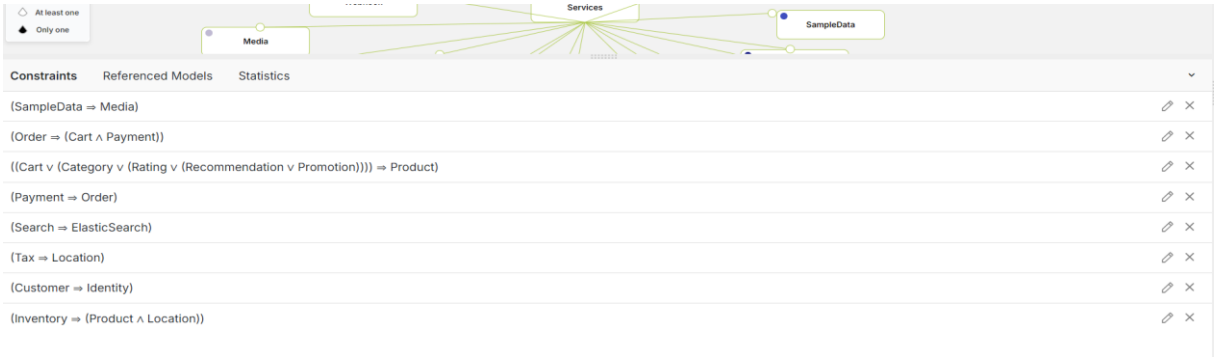


Figure 4.11 : Les contraintes du modèle de fonctionnalités pour l'application YAS

4.4 Mappage des fonctionnalités de l'application YAS

Figure 4.12 : Mappage des fonctionnalités de l'application YAS dans Mobioos Forge

Dans cette partie, nous présentons le mappage de fontionnalités de l'application YAS en utilisant Mobioos Forge (Figure 4.12). Cette étape consiste à relier chaque fonctionnalité identifiée dans le modèle à son implémentation dans le code source, à travers des marqueurs qui servent à localiser précisément les morceaux de code liés à une fonctionnalité donnée.

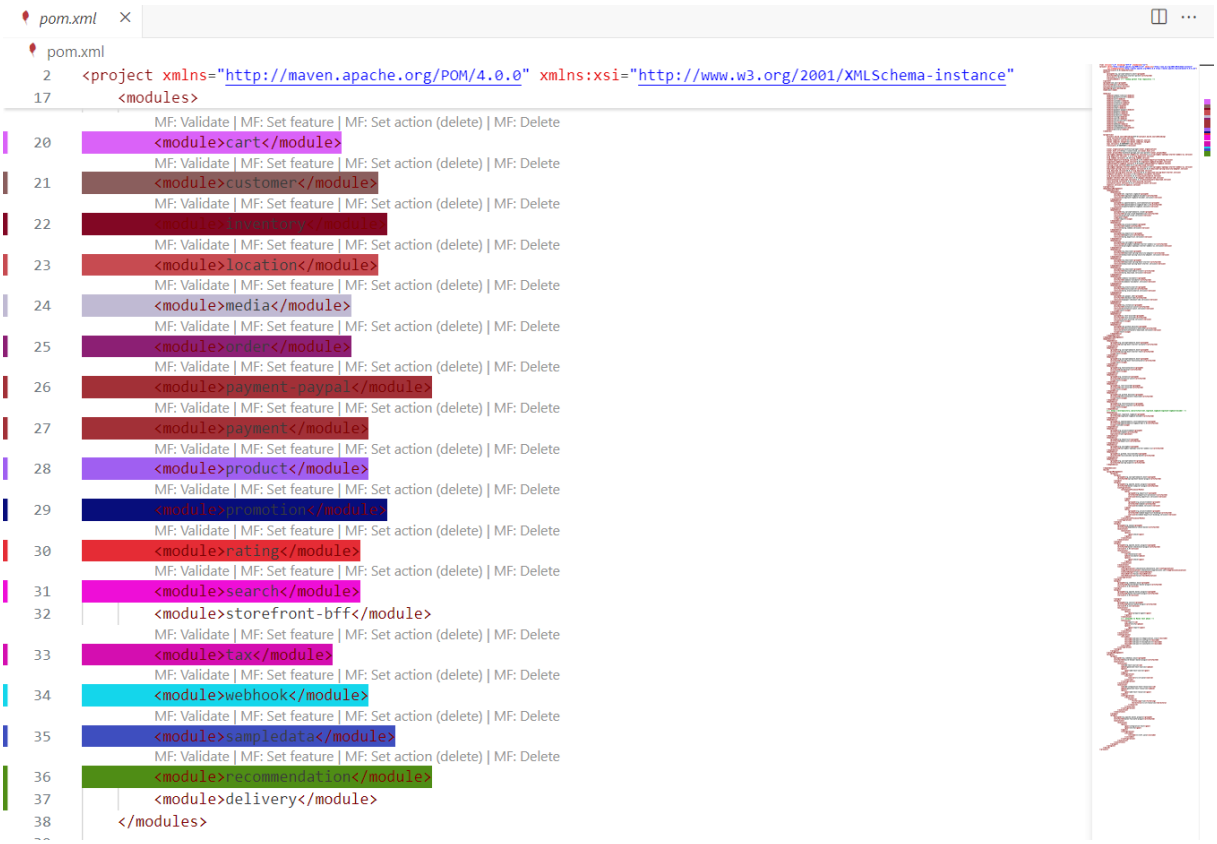


Figure 4.13 : Exemples des marqueurs de codes de quelques fonctionnalités

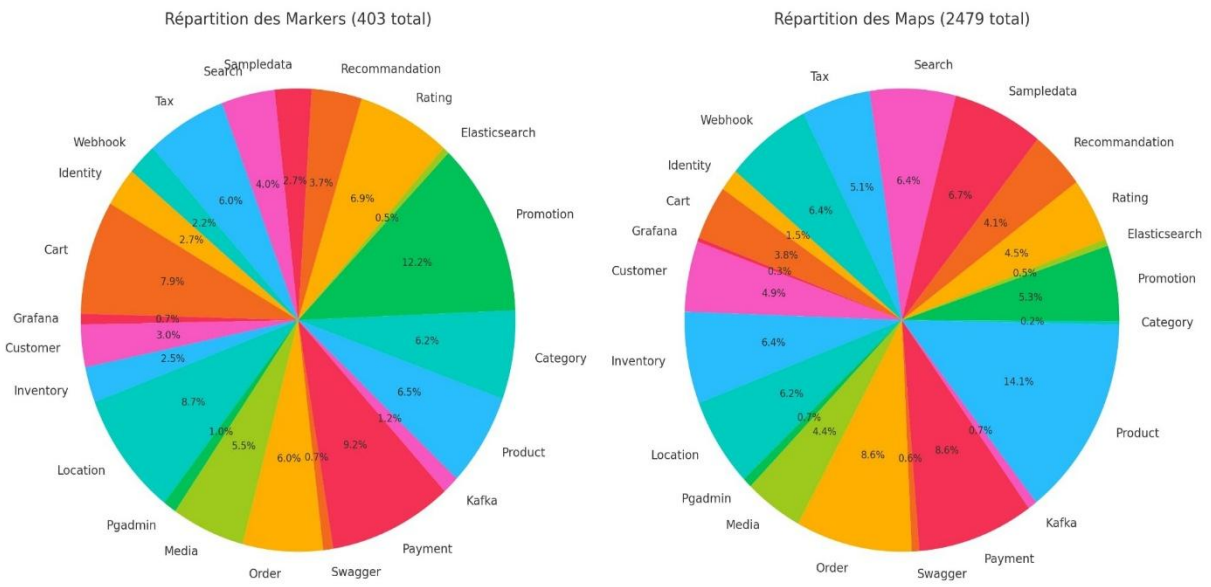


Figure 4.14 : Répartition des fonctionnalités selon le nombre des marqueurs et des maps

Dans notre cas, 403 marqueurs ont été ajoutés dans l'ensemble du projet, ce qui a permis de générer 2479 maps. Cela signifie que chaque marqueur a été associé à plusieurs points du code, renforçant ainsi la traçabilité entre les fonctionnalités et leur implémentation.

Lorsqu'on observe la répartition, on remarque que la fonctionnalité *Promotion* est celle qui a nécessité le plus de travail de marquage, avec environ 49 marqueurs, soit près de 12,2 % de l'ensemble. Cela s'explique par sa forte intégration dans différents modules de l'application. Viennent ensuite les fonctionnalités *Payment* (37 marqueurs), *Location* (35), *Cart* (32) et *Rating* (28). À l'inverse, des fonctionnalités plus simples ou plus isolées comme *Elasticsearch*, *Grafana* ou *Swagger* n'ont nécessité que 2 à 3 marqueurs chacune, ce qui reste logique compte tenu de leur nature ciblée et de leur impact limité sur le système global.

Une analyse plus précise permet de comparer le nombre de maps générés par rapport au nombre de marqueurs pour chaque fonctionnalité. Pour certaines fonctionnalités comme *Product*, *Sampledata*, *Search* ou *Webhook*, le nombre de maps est significativement supérieur au nombre de marqueurs. Cela indique que chaque marqueur est utilisé pour produire plusieurs adaptations ou scénarios. Par exemple, pour la fonctionnalité *Product*, on observe 41 marqueurs pour 355 maps, soit environ 14,5 maps par marqueur.

À l'inverse, pour la fonctionnalité *Category* (17 marqueurs pour 6 maps), le nombre de maps est inférieur à celui des marqueurs. Cela signifie que ces derniers ont été utilisés de manière plus ciblée, avec moins de variations associées.

Le Tableau 4.1 présente les métriques liées au mappage des fonctionnalités dans le modèle de fonctionnalités de l'application YAS. Pour chaque fonctionnalité, on indique le nombre de LOC, le pourcentage de LOC mappées, le nombre de fichiers impactés et le ratio Maps/Marker.

En analysant ce tableau, on observe que les fonctionnalités couvrent entre 0.05 % et 44.54 % des LOCs mappées. La fonctionnalité *Kafka* est la plus dominante avec 44,54 % des LOCs réparties sur 409 fichiers, ce qui reflète son rôle central dans l'application. À l'inverse, *ElasticSearch* (0.05 %), *Swagger* (0.19 %) et *Pgadmin* (0.22%) ont une couverture de code très faible, ce qui suggère une présence limitée dans le système.

En termes de fichiers impactés, les fonctionnalités *Kafka* (409 fichiers), *Product* (253) et *Order* (143) ainsi que *Customer* (143) sont les plus répandues, confirmant leur importance dans le cœur métier de l'application. À l'opposé, *ElasticSearch* (6 fichiers) et *Grafana* (6) restent marginales en termes de dispersion dans le code.

Tableau 4.1 : Métriques des fonctionnalités dans l'application YAS LPL

Feature	LOCS	% de LOCS mappées	Fichiers impactés	Maps/Marker
ElasticSearch	72	0.05%	6	6
Grafana	8017	6.03%	6	2.33
Identity	3085	2.32%	13	3.45
Kafka	59214	44.54%	409	3.4
Pgadmin	294	0.22%	10	4.25
Cart	2174	1.64%	53	2.93
Customer	2772	2.09%	143	10.16
Inventory	4101	2.98%	91	15.9
Location	6210	4.51%	93	4.4
Media	1550	1.13%	45	4.90
Order	6002	4.36%	143	8.91
Payment	2271	1.71%	114	5.78
Product	14719	11.07%	253	13.42
Category	1253	0.94%	13	0.24
Promotion	3890	2.93%	73	2.67
Rating	2271	1.71%	59	3.96
Recommendation	3897	2.93%	57	6.8
SampleData	1088	0.82%	27	15.18
Search	4791	3.60%	68	9.87
Tax	2867	2.16%	69	5.25
Webhook	2140	1.61%	72	17.66
Swagger	257	0.19%	8	4.66

Concernant le ratio Maps/Marker, certaines fonctionnalités comme *Webhook* (17.66), *Inventory* (15.9), *SampleData* (15.18) et *Product* (13.42) présentent une forte densité de mappage via des maps. À l'inverse, *Category* (0.24) et *Grafana* (2.33) présentent des ratios très faibles, ce qui signifie que leur présence repose davantage sur des markers que sur des blocs de code structurés.

4.5 Exemples des variantes dérivées

Le Tableau 4.2 présente les métriques de quatre variantes générées à partir de l'application YAS LPL ainsi que de l'application originale. Chaque variante est définie par les fonctionnalités activées ou désactivées, avec le nombre de LOCs générées et supprimées.

Tableau 4.2 : Métriques de variantes générées de l'application YAS LPL

Variantes (fonctionnalités activées et désactivées)	LOCS	LOCS supprimées
Toutes les fonctionnalités (application originale)	**133851**	**0**
Test 1 Sans : SampleData	**131510**	**2341**
Test 2 Sans: Rating, Recommendation, Inventory, Grafana, Customer.	**119666**	**14185**
Test 3 Sans : Promotion, Tax, Search, Swagger, Payment, ElasticSearch, Category.	**117516**	**16335**
Test 4 Seulement : Identity, Kafka	**62299**	**71552**

Dans cette partie, nous analysons certaines variantes présentées dans le tableau 4.2, en explorant les modifications apportées par rapport à l'application d'origine.

▸ La Variante Test 2 :

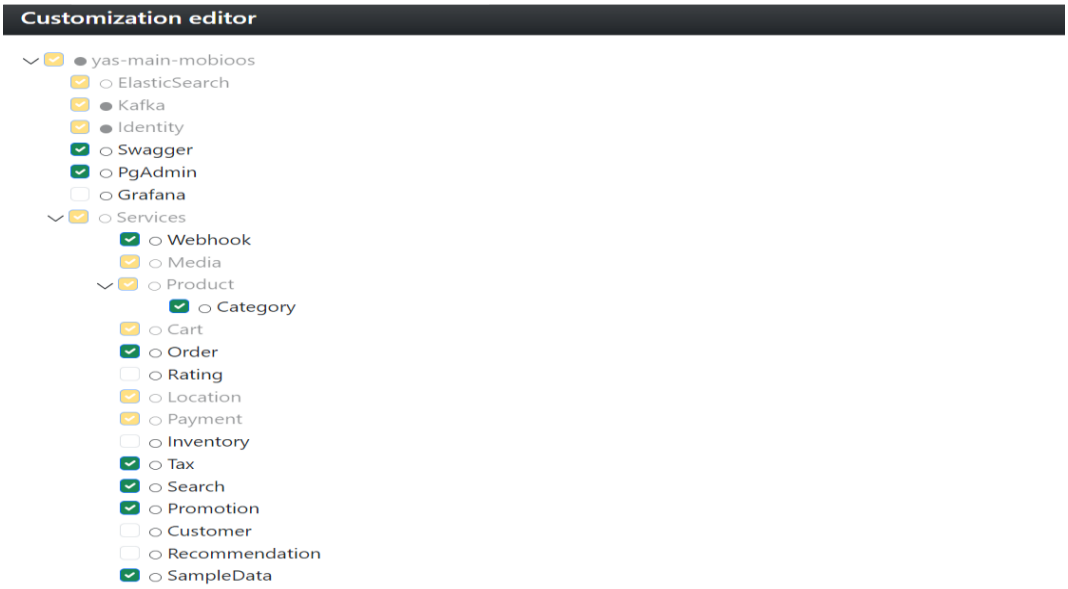


Figure 4.15 : Configuration de la variante Test 2

La **figure 4.15** montre la configuration de la variante test 2.

```
C:\Users\NeonTech.DZ\mobioos-forge-customizations\yas-main-mobioos\sans-recommendation-raiting-inventory-grafana>docker compose -f docker-compose.yml up -d
time="2025-06-01T14:16:08+01:00" level=warning msg="C:\\Users\\NeonTech.DZ\\mobioos-forge-customizations\\yas-main-mobioos\\sans-recommendation-raiting-inventory-grafana\\docker-compose.yml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
time="2025-06-01T14:16:09+01:00" level=warning msg="a network with name yas-network exists but was not created for project \"sans-recommendation-raiting-inventory-grafana\".\nSet `external: true` to use an existing network"
time="2025-06-01T14:16:09+01:00" level=warning msg="Found orphan containers ([sans-recommendation-raiting-inventory-grafana-inventory-1 sans-recommendation-raiting-inventory-grafana-customer-1]) for this project. If you removed or renamed this service in your compose file, you can run this command with the --remove-orphans flag to clean it up."
[+] Running 21/21
 ✔Container sans-recommendation-raiting-inventory-grafana-pgadmin-1           Started                    2.3s
 ✔Container sans-recommendation-raiting-inventory-grafana-postgres-1          Started                    2.8s
 ✔Container sans-recommendation-raiting-inventory-grafana-identity-1          Started                    2.2s
 ✔Container sans-recommendation-raiting-inventory-grafana-storefront-1        Started                    1.3s
 ✔Container sans-recommendation-raiting-inventory-grafana-cart-1              Started                    1.5s
 ✔Container sans-recommendation-raiting-inventory-grafana-product-1           Started                    1.7s
 ✔Container sans-recommendation-raiting-inventory-grafana-media-1             Started                    2.6s
 ✔Container sans-recommendation-raiting-inventory-grafana-location-1          Started                    1.6s
 ✔Container sans-recommendation-raiting-inventory-grafana-nginx-1             Started                    2.1s
 ✔Container sans-recommendation-raiting-inventory-grafana-zookeeper-1         Started                    2.0s
 ✔Container sans-recommendation-raiting-inventory-grafana-tax-1               Started                    2.5s
 ✔Container sans-recommendation-raiting-inventory-grafana-order-1             Started                    2.9s
 ✔Container sans-recommendation-raiting-inventory-grafana-payment-1           Started                    2.7s
 ✔Container sans-recommendation-raiting-inventory-grafana-kafka-1             Started                    3.1s
 ✔Container sans-recommendation-raiting-inventory-grafana-backoffice-1        Started                    2.4s
 ✔Container sans-recommendation-raiting-inventory-grafana-sampledata-1        Started                    1.5s
 ✔Container sans-recommendation-raiting-inventory-grafana-redis-1             Started                    1.6s
 ✔Container sans-recommendation-raiting-inventory-grafana-kafka-connect-1     Started                    4.0s
 ✔Container kafka-ui                                                          Started                    4.7s
 ✔Container sans-recommendation-raiting-inventory-grafana-storefront-nextjs-1 Started                    1.8s
 ✔Container sans-recommendation-raiting-inventory-grafana-backoffice-nextjs-1 Started                    2.1s

C:\Users\NeonTech.DZ\mobioos-forge-customizations\yas-main-mobioos\sans-recommendation-raiting-inventory-grafana>
```

Figure 4.16 : Exécution de la variante Test 2

La figure 4.16 présente l'exécution de la variante test 2 en utilisant la commande : *docker compose -f docker-compose.yml up -d*, qui permet de lancer les services dans des conteneurs Docker en arrière-plan.

On accéder à l'interface administrateur via le lien : http://backoffice/

On accéder à l'interface client via le lien : http://storefront/

La figure 4.17 présente l'interface de l'administrateur, où l'on remarque la disparition des menus *Inventory* et *Customers*.

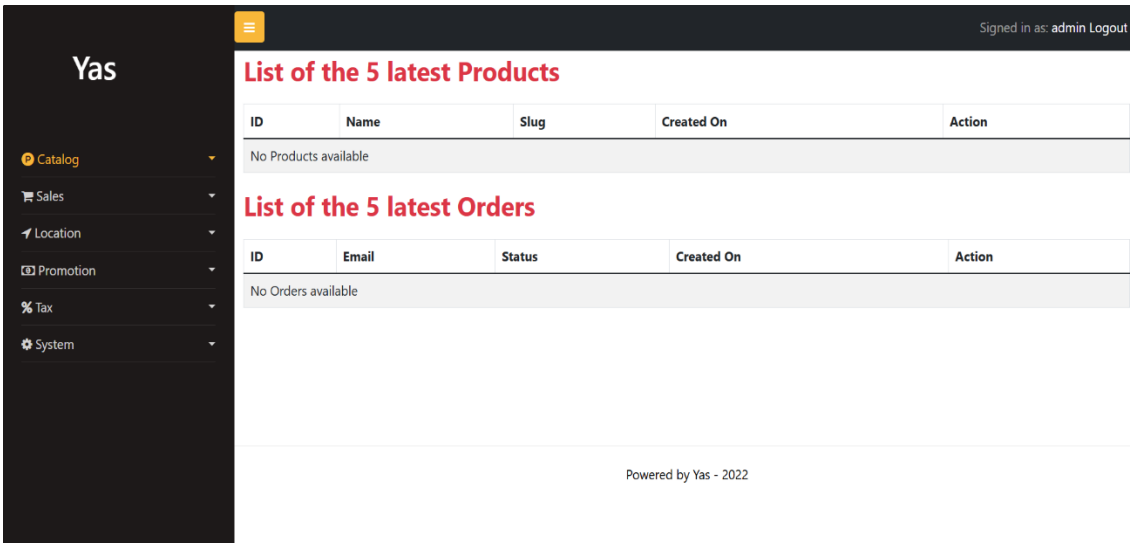


Figure 4.17 : L'interface administrateur de la variante test 2

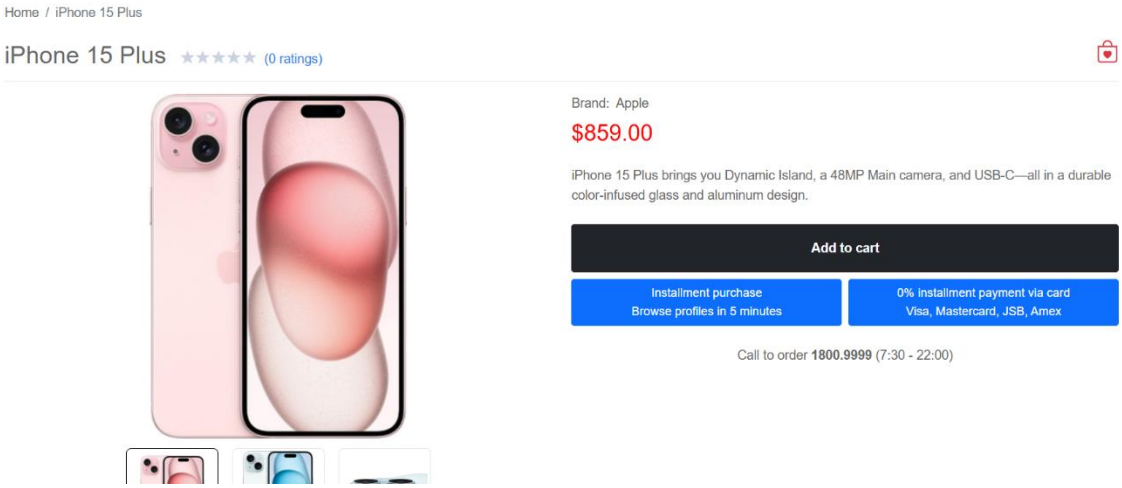


Figure 4.18 : L'interface pour ajouter un produit au panier dans l'application d'origine

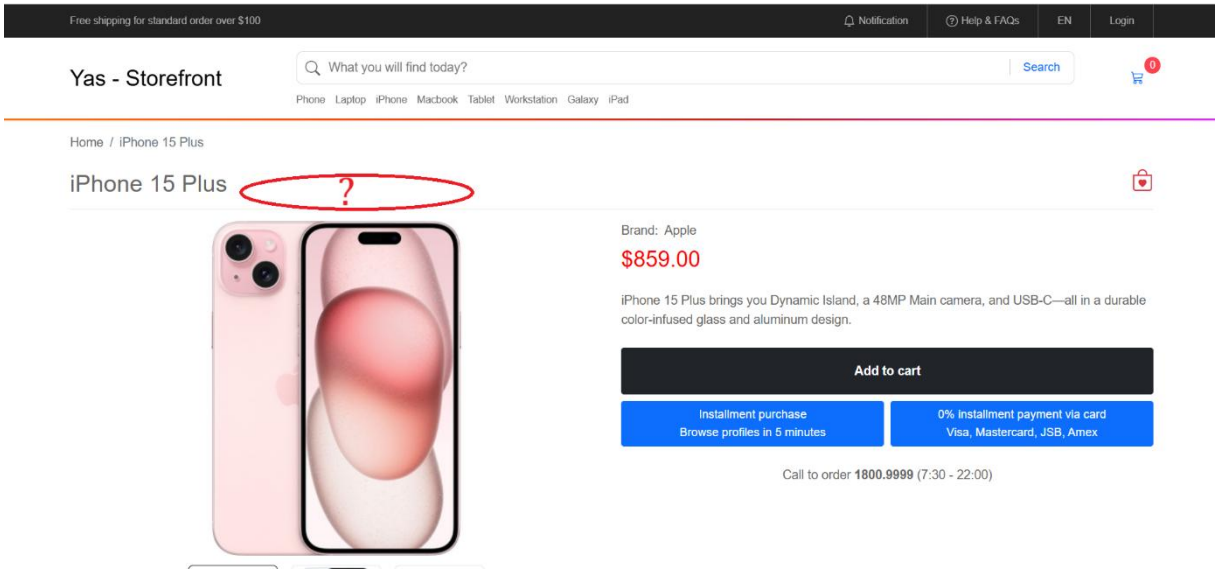


Figure 4.19 : L'interface pour ajouter un produit au panier dans la variante test 2

Selon la figure 4.19, on remarque la disparition de la fonctionnalité d'évaluation par étoiles (*ratings*).

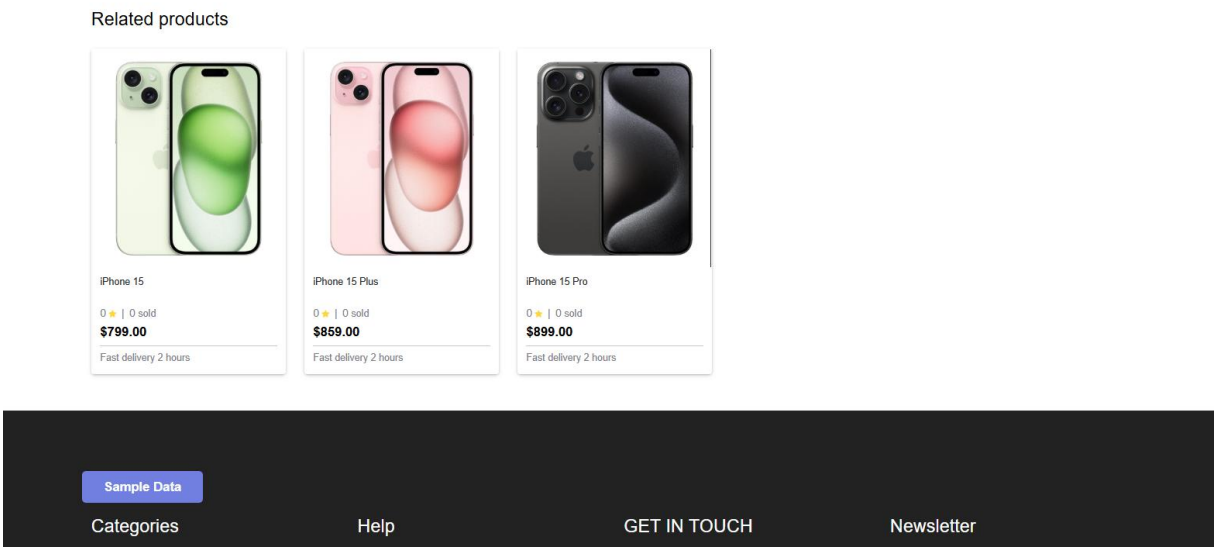


Figure 4.20 : L'interface utilisateur des produits recommandés de l'application d'origine

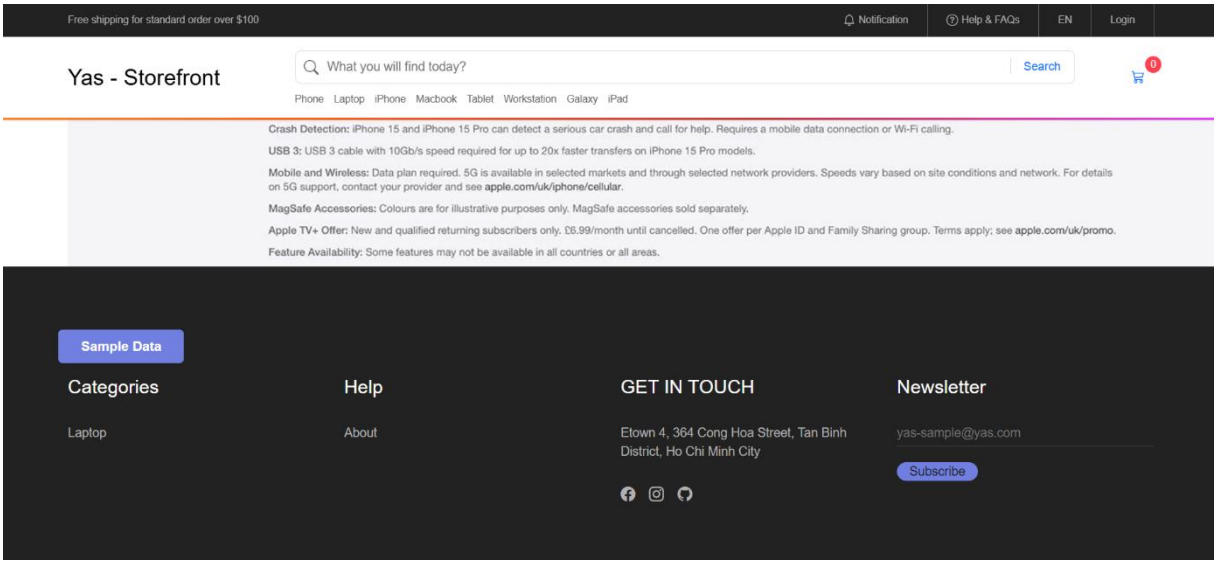


Figure 4.21 : L'interface utilisateur des produits recommandés de la variante test 2

Selon la figure 4.21, on remarque la disparition de la fonctionnalité des produits recommandés.

▸ La Variante Test 3 :

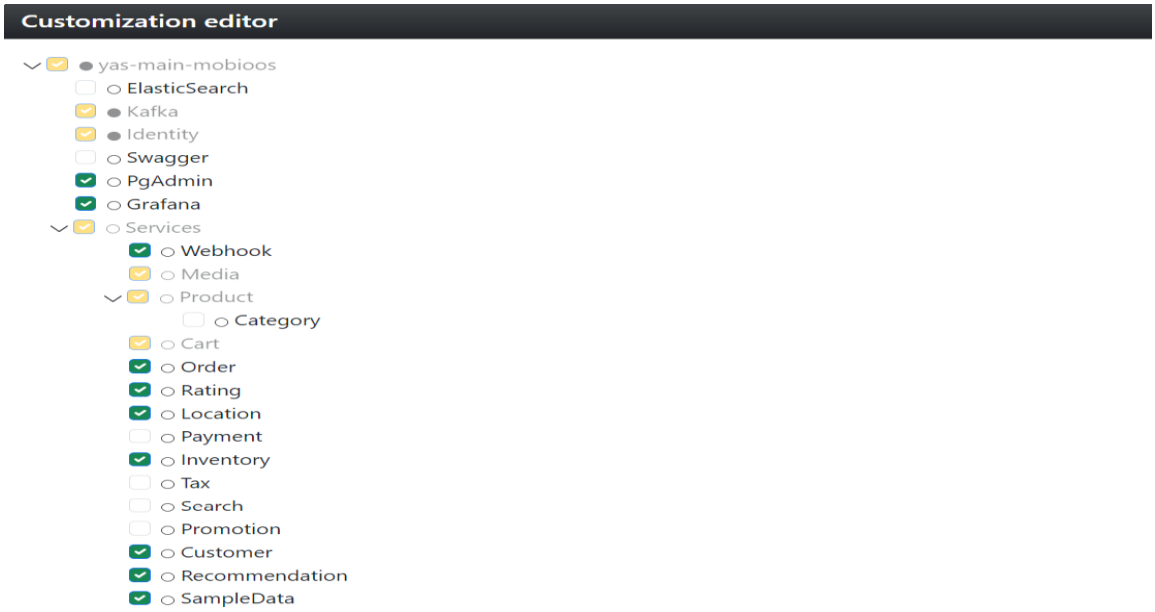


Figure 4.22 : Configuration de la variante Test 3

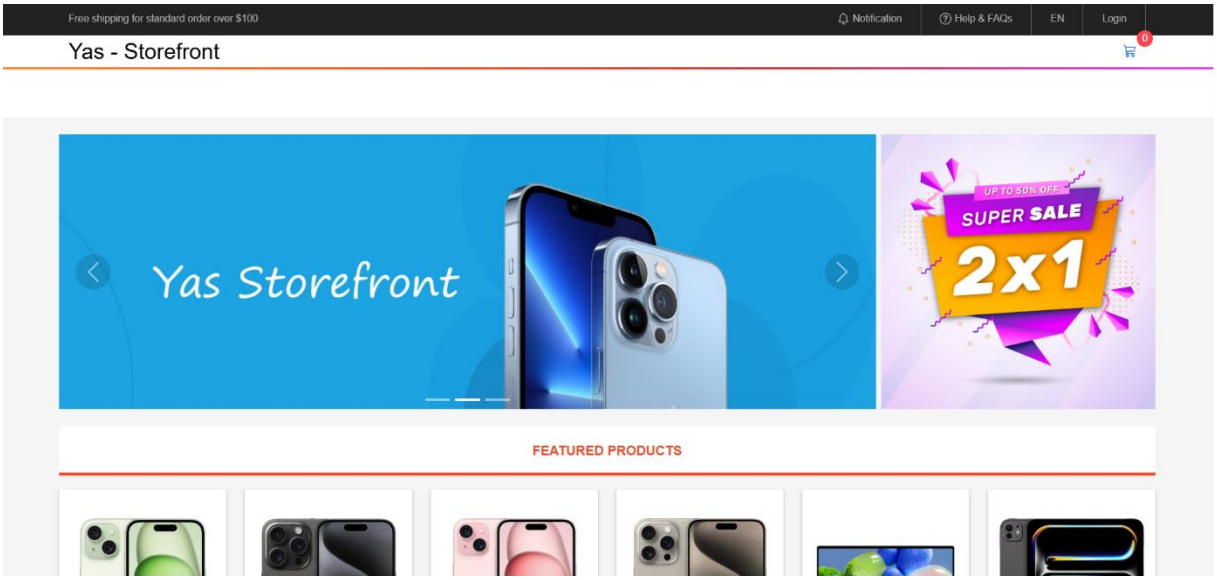


Figure 4.23 : L'interface de la page d'accueil de la variante test 3

Selon la figure 4.23, on remarque la disparition de la fonction de recherche ainsi que celle des catégories.

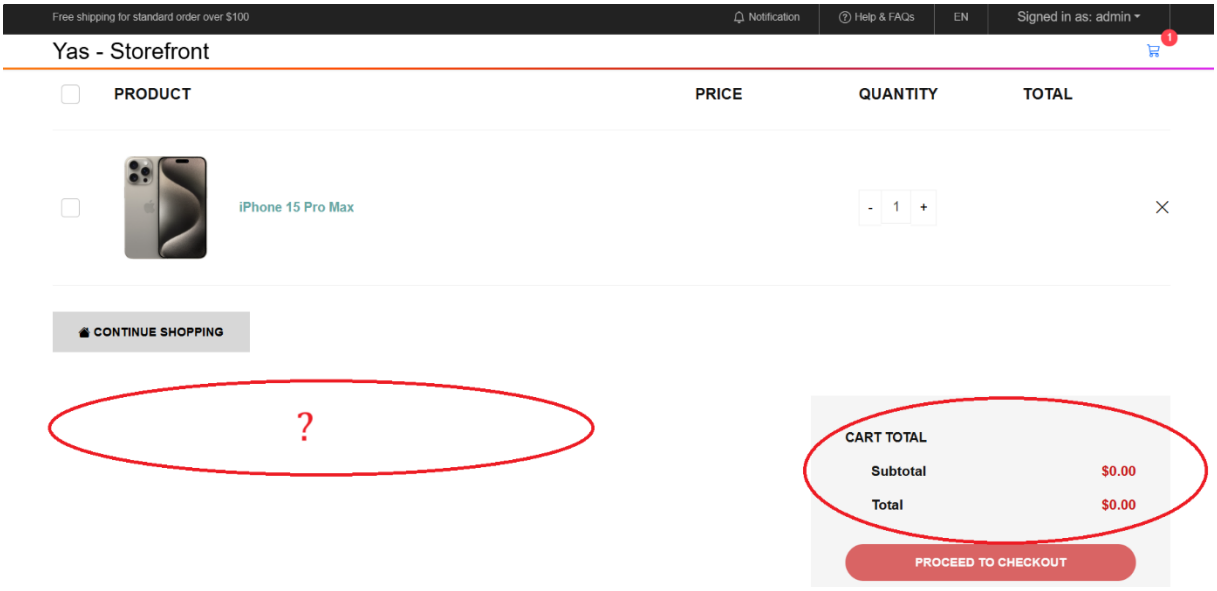


Figure 4.24 : Interface de la liste des produits ajoutés au panier pour la variante Test 3

Selon la figure 4.24, on remarque la disparition de la fonctionnalité de *promotion*.

La variante Test 4

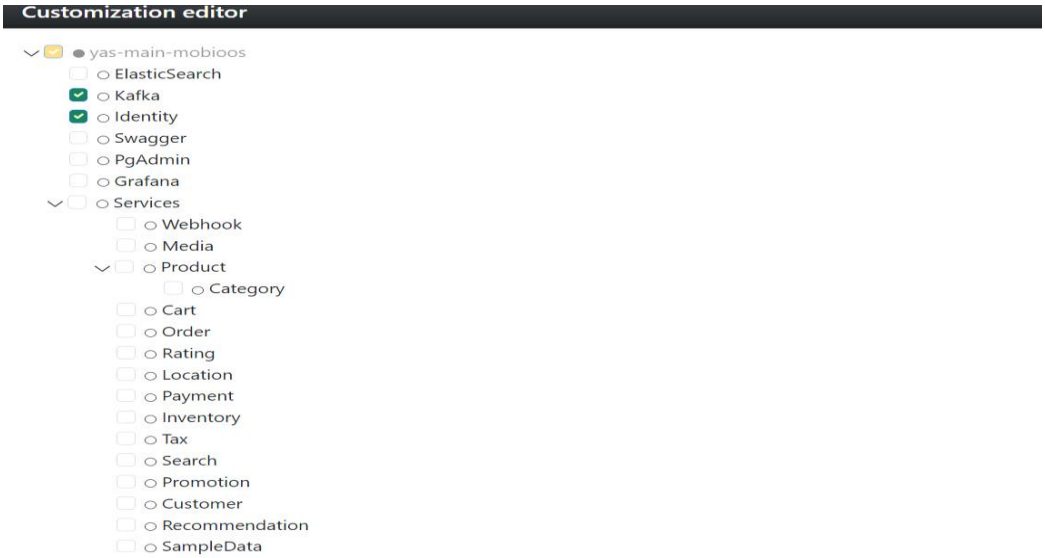


Figure 4.25 : Configuration de la variante Test 4

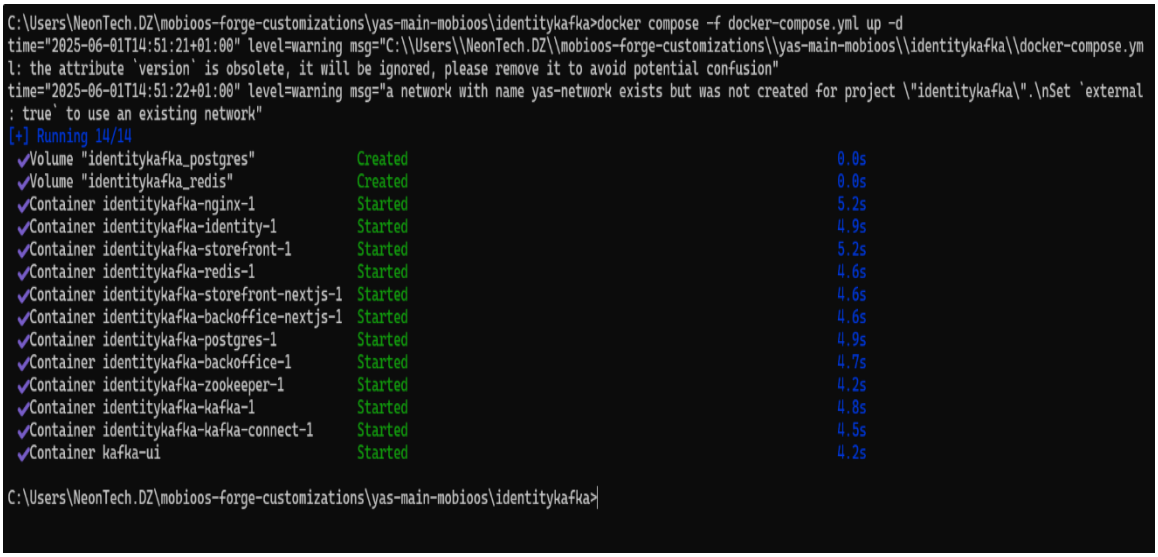


Figure 4.26 : Exécution de la variante Test 4

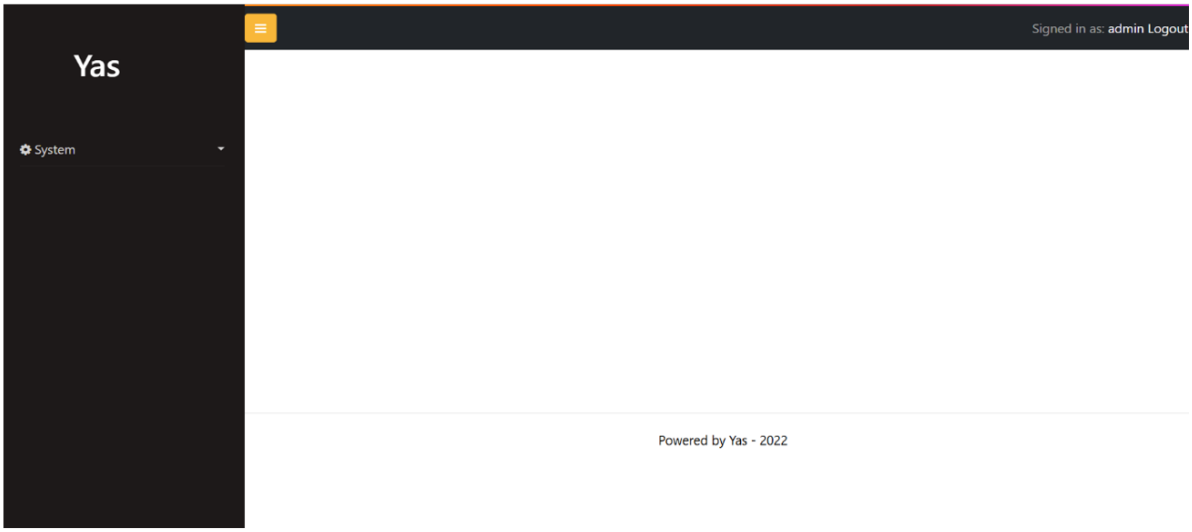


Figure 4.27 : L'interface administrateur de la variante test 4

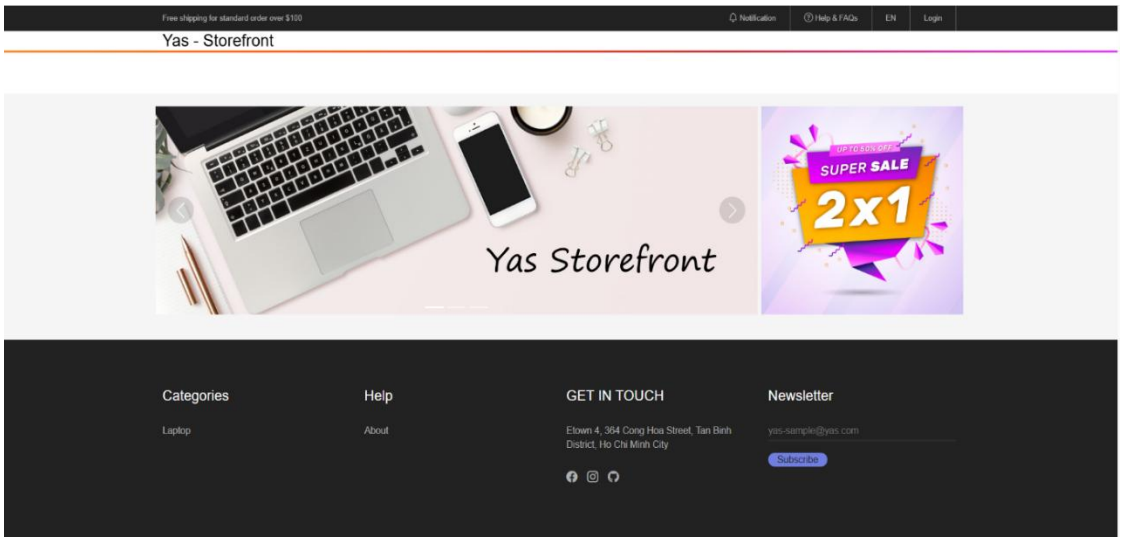


Figure 4.28 : L'interface utilisateur de la variante test 4

4.6 Calcul du temps de migration

La figure 4.29 illustre, à travers un graphique, le temps nécessaire pour mapper chaque fonctionnalité de l'application, représenté par des barres verticales, avec une courbe superposée indiquant le nombre de maps validées manuellement.

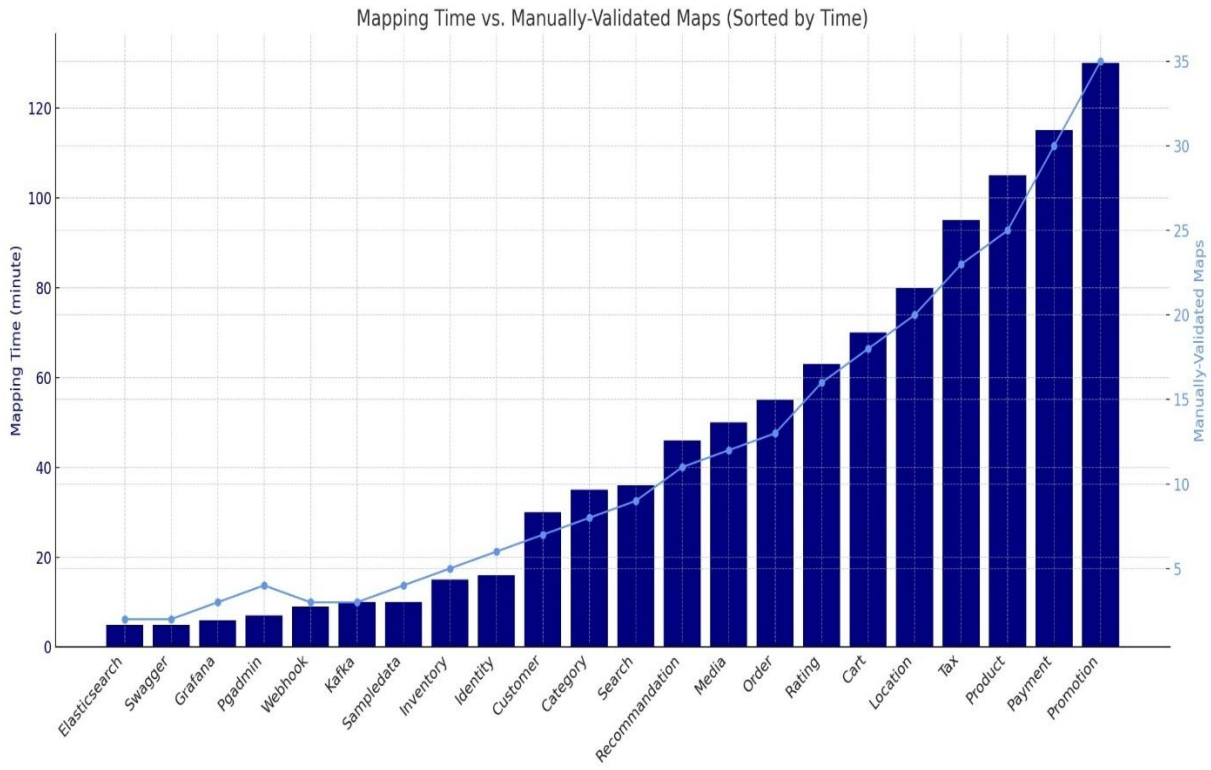


Figure 4.29 : Temps de mapping et nombre de maps validées manuellement de l'application

On remarque que la majorité des fonctionnalités ont été mappées assez rapidement : *Elasticsearch*, *Swagger*, *Webhook* ou encore *Grafana* n'ont pris que 10 minutes ou moins, tandis que des services comme *Kafka*, *Pgadmin* ou *Identity* ont demandé un peu plus de temps, mais restent sous la barre d'une heure

Par contre, certaines fonctionnalités se distinguent par des durées bien plus longues. *Product* et *Payment* nécessitent entre 1h30 et 2h, tandis que Promotion monte jusqu'à plus de 2h10.

On remarque que ces fonctionnalités sont également celles qui présentent le plus grand nombre des maps validées manuellement ce qui suggère qu'il y a une relation directe entre la complexité du mapping et le nombre de validations nécessaires. Par exemple, *Promotion* cumule 35 maps, *Payment* en a 30, et *Product* en compte 25.

Ce graphique met donc en lumière un lien logique : **plus une fonctionnalité nécessite de maps validées, plus son mapping est long, ce qui reflète probablement sa complexité fonctionnelle ou son importance dans l'application.**

4.7 Comparaison entre les applications eShopOnContainers et YAS

Afin d'évaluer l'efficacité de la migration vers les Lignes de Produits Logiciels (LPL), le tableau 4.3 présente une comparaison entre deux applications e-commerce basées sur des Microservices suivantes : **eShopOnContainers** et **YAS**. Cette comparaison tient compte de plusieurs critères techniques et fonctionnels, tels que le nombre de Microservices, les technologies utilisées, le nombre de LOCs, les fonctionnalités identifiées dans le modèle LPL.

Tableau 4.3 : Comparaison entre les deux applications migrées vers des Lignes de Produits Logiciels (LPL)

Nom de l'app	Technologies	MS	Protocoles de communication	LOCs	Fonctionnalités	Temps de Mapping (min)	Contraintes	Maps/ Marker
eShopOnContainers	.NET , Docker, Azure, ,RabbitMQ, AspNetCore, Redis, Blazor	8	HTTP/GRPC	57916	11	780 minutes	4	7.63
YAS	Java, Spring Boot, Docker, Next.js , Kafka, Kubernetes , Keycloak, PostgreSQL, ElasticSearch, OpenTelemetry Grafana, Loki, Prometheus, Tempo	22	HTTP	133851	23	993 minutes	8	6.15

Selon le tableau 4.3, on remarque que l'application **eShopOnContainers** utilise des technologies comme .NET, Docker et Azure. Elle est composée de 8 microservices et contient environ 58000 lignes de code. Lors de la migration vers une LPL, 11 fonctionnalités ont été identifiées, et le temps nécessaire pour faire le mapping était de 780 minutes. On a aussi relevé 4 contraintes pendant le processus de migration.

De son côté, **YAS** est plus grande et plus complexe. Elle utilise des technologies open-source comme Java, Spring Boot, Kafka, Kubernetes, etc. Elle contient 22 microservices, avec environ 134000 lignes de code. Le mapping vers une LPL a permis d'extraire 23 fonctionnalités, avec une durée d'environ 993 minutes. Il y avait aussi plus de contraintes (8 contraintes).

Enfin, le rapport **Maps/Marker** mesure l'équilibre entre les maps (fonctionnalités communes) et les markers (variantes). Ce ratio est un bon indicateur de la variabilité d'une ligne de produits. Plus le chiffre est petit, plus il y a de markers, donc plus de variantes à gérer. Dans notre cas, eShopOnContainers affiche 7,63, tandis que YAS est à 6,15. Cela montre clairement que YAS comporte davantage de variations fonctionnelles, ce qui rend sa modélisation plus complexe.

En résumé, **YAS est plus grande et plus riche en fonctionnalités**, mais elle demande plus d'effort pour être transformée en LPL. Cette comparaison nous aide à mieux comprendre les différences entre les projets, et l'impact de leur complexité sur le processus de migration.

4.8 Conclusion

Dans ce chapitre, nous avons présenté l'application YAS, développée selon une architecture à base de Microservices. Ensuite, nous avons introduit le modèle de fonctionnalité afin d'identifier les points de variabilité, puis exposé le mappage des fonctionnalités entre le modèle et l'application. À partir de cela, nous avons généré quelques exemples de variantes dérivées, illustrant la transformation vers une ligne de produits logiciels. Enfin, nous avons calculé le temps de migration de cette application. Enfin, nous avons réalisé une comparaison entre la migration de l'application eShopOnContainers et celle de YAS, afin d'analyser les différences en termes d'effort, de complexité et de résultats obtenus.

Conclusion générale

Dans ce mémoire, nous avons travaillé sur la migration des applications e-commerce open source basées sur des microservices vers une approche appelée Lignes de Produits Logiciels (LPL) en utilisant la plateforme *Mobioos Forge*. Plus précisément, nous avons commencé par étudier l'architecture Microservices, ainsi que le domaine des applications e-commerce, car c'est dans ce domaine que nous avons voulu appliquer notre approche. Ensuite, nous avons maîtrisé les concepts fondamentaux des Lignes de Produits Logiciels (LPL), ainsi que la plateforme Mobioos Forge. Notre contribution principale est la migration de deux applications open source e-commerce (*eShopOnContainers* et *YAS*), basées sur des microservices, vers une architecture LPL. La première étape de ce processus a nécessité une connaissance approfondie du domaine e-commerce, en particulier les applications à migrer. Ensuite, nous avons proposé un modèle de fonctionnalité pour chaque application et aussi lié chaque fonctionnalité avec son code dans l'application avec des marqueurs. Nous avons aussi montré des exemples de variantes dérivées et calculé le temps nécessaire pour chaque migration. Enfin, une analyse comparative a été effectuée entre les deux migrations. Donc, nous avons prouvé qu'il est possible de transformer une application à base de Microservices vers une LPL. Le résultat final permet de mieux gérer plusieurs variantes d'une même application tout en réduisant le travail répétitif. Même si notre méthode fonctionne, elle a nécessité beaucoup de travail manuel, Ce qui exige une grande expertise dans ce domaine.

Ce travail nous a permis d'apprendre beaucoup, il nous a permis de mieux comprendre les architectures modernes et d'améliorer notre méthode de développement des systèmes logiciels. Nous avons approfondi notre expertise professionnelle sur les approches modernes de développement, notamment les LPL et les Microservices.

Les perspectives pour l'avenir de ce travail impliquent la migration de d'autres applications, telles que *Vert.x Micro-shop*, *e-commerce-microservices*, *nodejs_microservice* Vers des LPLs d'une part et d'autre part la proposition d'une approche de migration automatique des Microservices vers les LPLs en utilisant l'intelligence artificielle, notamment à travers l'usage des LLMs (Large Language Models). Cette approche permettra de gagner du temps, de réduire les erreurs humaines et de faciliter le travail des développeurs.

Bibliographie

[1] Lewis, J., & Fowler, M. (2014). Microservices: a definition of this new architectural term. https://martinfowler.com/articles/microservices.html

[2] Richardson, C. (2019). Microservices Patterns: With Examples in Java. Manning Publications.

[3] Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, Today, and Tomorrow (pp. 195–216). Springer International Publishing, Cham .

[4] Zaragoza, P. (2022). Migration Dirigée par les Modèles d'Applications Monolithiques vers des Architectures à base de Micro-Services (Thèse de doctorat, Université de Montpellier). NNT: 2022UMONS051. https://tel.archives-ouvertes.fr/tel-04594297 .

[5] Oracle docs , Learn about architecting microservices-based applications on Oracle Cloud, URL: https://docs.oracle.com/en/solutions/learn-architect-microservice/index.html#GUID-8F74C81E-17E0-4697-98A7-C3345F7A75D7

[6] Daoud, M. T. (2021). Vers une approche d'identification automatique de microservices pour les besoins de migration de systèmes d'information (Thèse de doctorat, Université de Lyon). NNT: 2021LYSE1233. https://tel.archives-ouvertes.fr/tel-03663589.

[7] CodiLime, What is microservices architecture — types, tools, pros and cons, URL : https://codilime.com/blog/what-is-microservices-architecture/

[8] Intuit mailchimp , Qu'est-ce que l'e-commerce ? , URL : https://mailchimp.com/fr/marketing-glossary/e-commerce/

[9] Punub , What is an eCommerce application? , URL : https://www.pubnub.com/learn/glossary/what-is-an-ecommerce-application/

[10] Dimitrieski, H. e-shop – A Microservices-based E-commerce Application. GitHub repository. URL : https://github.com/hdimitrieski/e-shop

[11] Sahay, R. eShopping – Microservices Implementation Using .NET & Angular. GitHub repository. URL : https://github.com/rahulsahay19/eShopping

[12] Ziadi, T. (2004). Manipulation de lignes de produits en UML (Thèse de doctorat, Université de Rennes 1, IFSIC, école doctorale MATISSE, équipe IRISA-TRISKELL). No d'ordre : 3083.

[13] Urli, S. (2015). Processus flexible de configuration pour lignes de produits logiciels complexes (Thèse de doctorat, Université Nice Sophia Antipolis). NNT: 2015NICE4002 https://tel.archives-ouvertes.fr/tel-01134191v2 .

[14] Brandão, Anarosa & Boufedji, Dounia & Ziadi, Tewfik & Guessoum, Zahia. (2015). Vers une approche d'ingénierie multiagent à base de ligne de produits logiciels.

[15] K. Ghallab, T. Ziadi, and Z. Chalal, Migrating Individual Applications into Software Product Lines using the Mobioos Forge Platform, Asia Pacific Software Engineering Conference, 2023.

[16] MOBIOOS, Mobioos Forge. https://documentation.mobioos.ai/

[17] PTC , Ingénierie de lignes de produits, URL : https://www.ptc.com/fr/technologies/application-lifecycle-management/product-line-engineering-ple

[18] Apel, S., Batory, D., & Kästner, C. (2013). Feature-Oriented Software Product Lines. Springer.

[19] Bmw , Configurateur bmw , URL : https://www.bmw.fr/fr/configurateur.html

[20] young urban project , What is a Product Line? Examples, Importance and Strategies , URL: https://www.youngurbanproject.com/what-is-a-product-line/

[21]Crocs , Site officiel de Crocs France , URL : https://www.crocs.fr/

[22] Microsoft , Site officiel de Microsoft France , URL: https://www.microsoft.com/fr-fr/

[23] W. K. G. ASSUNÇÃO, J. KRÜGER et W. D. F. MENDONÇA, Variability Management Meets Microservices: Six Challenges of Re-engineering Microservice-Based Webshops, In Proceedings of the 24th International Systems and Software Product Line Conference (SPLC '20), ACM, Montréal, Canada, 2020. DOI: 10.1145/3382025.3414942.

[24] Microsoft. eShop – Sample Microservices-based .NET Application. GitHub repository. URL : https://github.com/dotnet/eShop

[25] NashTech Garage. YAS (Yet Another Shop) – A Microservices E-commerce Platform. GitHub repository. URL : https://github.com/nashtech-garage/yas