

**Département Mathématiques et Informatiques**

**FI-Génie du Logiciel et des Systèmes Informatiques Distribués**

“**TP Gestion de catalogue  
Avec l’architecture distribuées Dot Net**

**NAJI khadija**

**Elément : Architecture distribuées Dot Net**

**GLSID, Semestre 5**

Sommaire :

[**Introduction** 2](#_Toc61561972)

[**1-** **Énoncé des problèmes à réaliser** 2](#_Toc61561973)

[**2-** **Architecture et conception en fonction des cas** 2](#_Toc61561974)

[**3-** **Code source des classes, interfaces et fichiers de configuration** 4](#_Toc61561975)

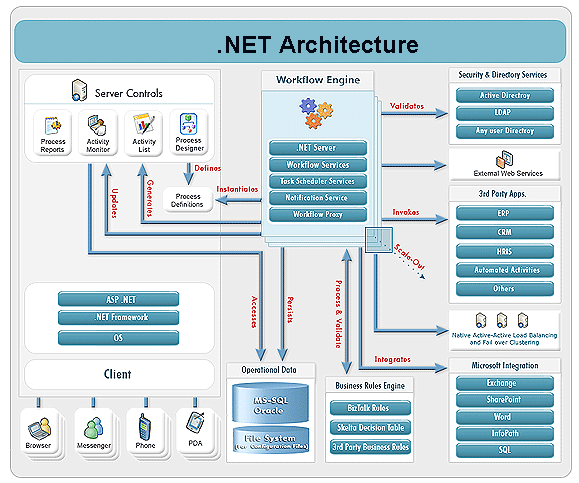
[**4-** **Captures d'écrans relatives à l'exécution** 15](#_Toc61561976)

[**Conclusion** 26](#_Toc61561977)

# **Introduction**

Microsoft .NET est un terme général qui désigne un certain nombre de technologies lancées par Microsoft. Prises dans leur ensemble, ces technologies représentent les changements les plus importants apportées à la plate-forme de développement Microsoft depuis le passage du développement.

La plate-forme Microsoft .NET est une solution complète pour développer, déployer et exécuter des Applications de tous types, y compris des Services Web. Fondée sur des standards de l'industrie (HTTP, XML, SOAP, WSDL), la plate-forme .NET est un moyen simple et puissant d'implémenter la coopération des services logiciels entre eux, quelle que soit leur localisation, leur implémentation technique, qu'ils soient internes ou externes, existants ou à inventer. La plate-forme Microsoft .NET repose sur le système d'exploitation Windows et est composée du Framework .NET qui est le socle de développement.



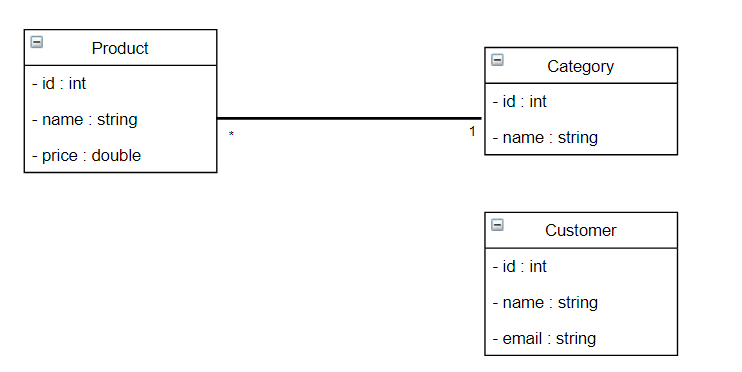
L'objectif de .NET est d'assurer l'interopérabilité entre les différentes machines reliées sur Internet. Il s'agit de pouvoir rassembler l'information, et de profiter des services de l'Internet sur plusieurs types de supports. .NET veut être la plate-forme de référence pour le développement de la prochaine génération d'applications : les services web XML. Une partie est entièrement consacrée à la mise en œuvre de cette technologie sous .NET.

1. **Énoncé des problèmes à réaliser**

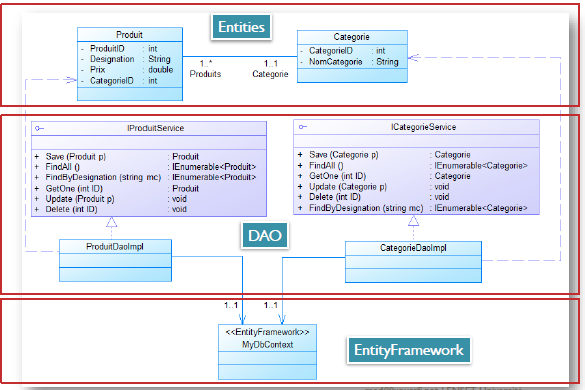
On souhaite créer une application Web Distribuée basée sur les micro-services qui permet de gérer la facturation des produits concernant des clients. La partie backend de l’application est basée sur deux micro-services. Un micro-service qui permet de gérer des clients et produits appartenant à des catégories qui sera développé en Dot.Net Core et un micro-service qui permet gérer la facturation développée avec Spring Boot. Les deux applications communiquent d’une manière synchrone avec REST et Asynchrone à travers un Broker KAFKA. La partie Frontend web de l’application sera développée en Angular.

1. Créer les entités Client, Produit et Catégorie dont la persistance sera gérée par Entity Framework.
2. Créer des API REST qui permettent de gérer les clients, les produits et les catégories.
3. Tester les API Rest en utilisant un Client REST.
4. Créer une application Angular qui permet de gérer les clients, les produits et les catégories.
5. Mette en place un Broker KAFKA.
6. Modifier l’application pour qu’à chaque opération d’ajout de modification ou de suppression, un évènement doit être envoyé vers un Topic KAFKA
7. Créer un Consommateur KAFKA de ce Topic avec Spring Boot.
8. **Architecture et conception en fonction des cas**

Le diagramme de classe utilisé sur cette micro service de Dot Net :



Voici l’architecture de produit et des catégorie liée avec l’architecture de Dot Net que j’ai adopté pour réaliser cette application :

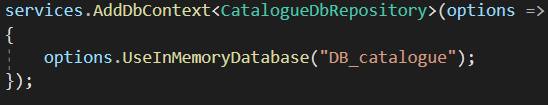


1. **Code source des classes, interfaces et fichiers de configuration**

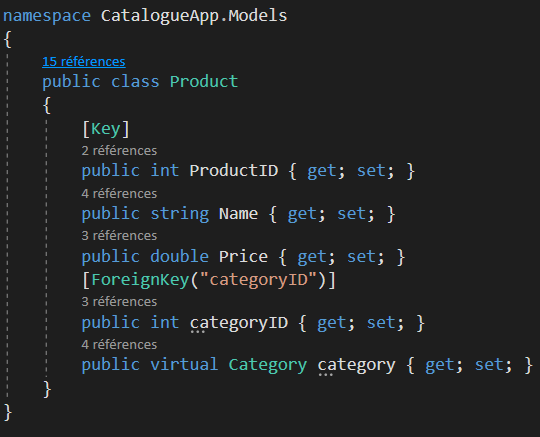
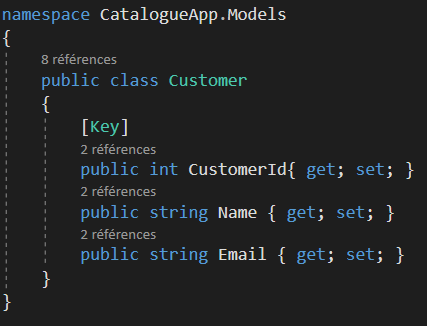
Les dépendances qu’elles faut intégrer pour réaliser cette application rendu serveur :

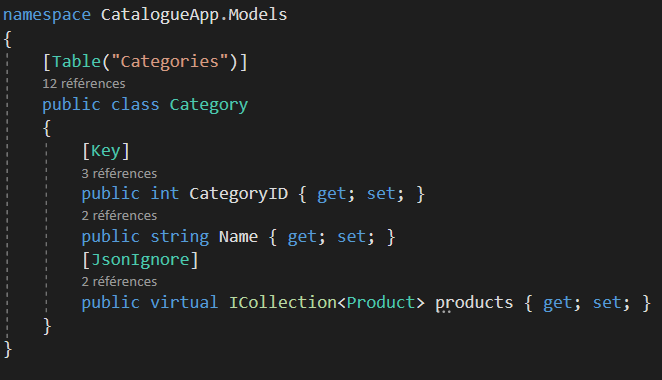
* Microsoft Entity Framework Core 3.1.0
* Microsoft Entity Framework Core in Memory 3.1.0 (memory database)
* Confluent Kafka 1.4.0

Pour créer une base de données pendant l’exécution du programme :

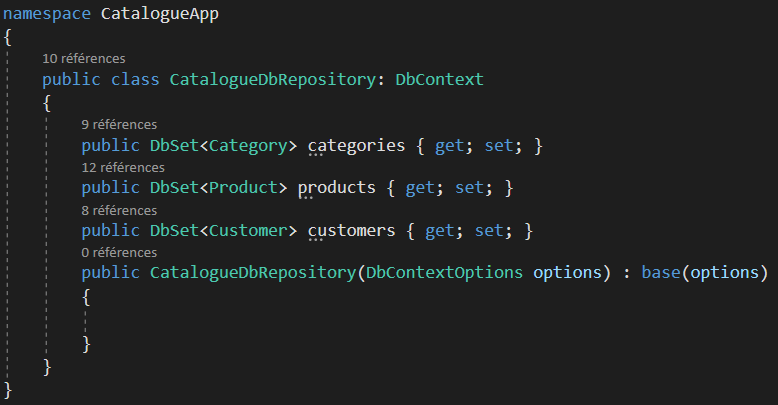


Après, j’ai créé les 3 modèles :



J’ai créé également le repository des 3 modèles :



Pour tester l’application, j’ai ajouté la classe **Dbinit** pour initialiser quelques données sur la base de données pendant l’exécution :

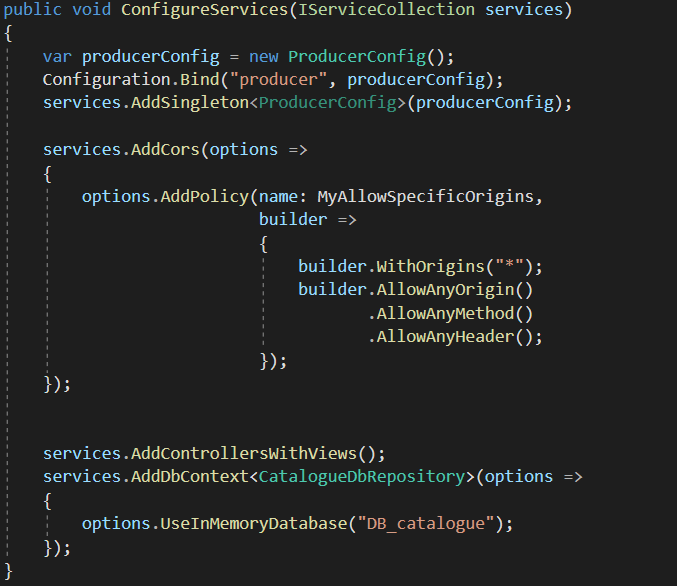


Pour prendre en considération cette classe pendant l’exécution du programme, je l’ai ajouté à la classe « **startup** » :



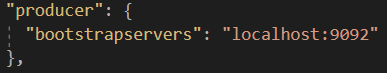
J’ai intégré également des configurations de CORS et de Kafka Producer à la même classe :





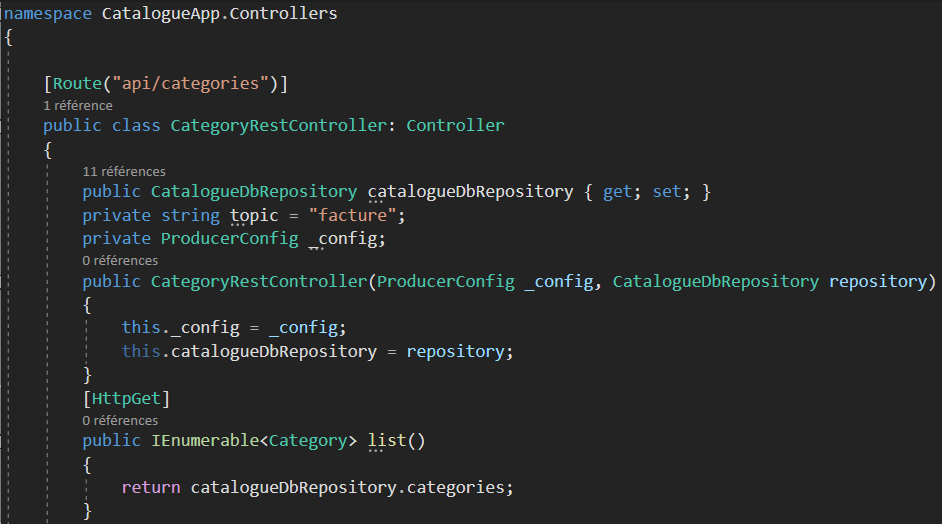


Les lignes suivantes sont nécessaires pour activer Kafka Producer :

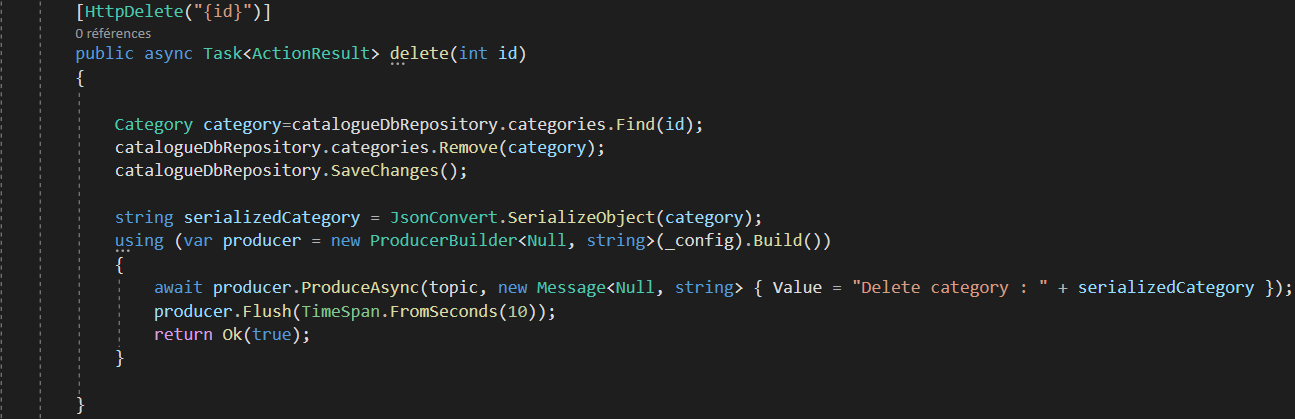


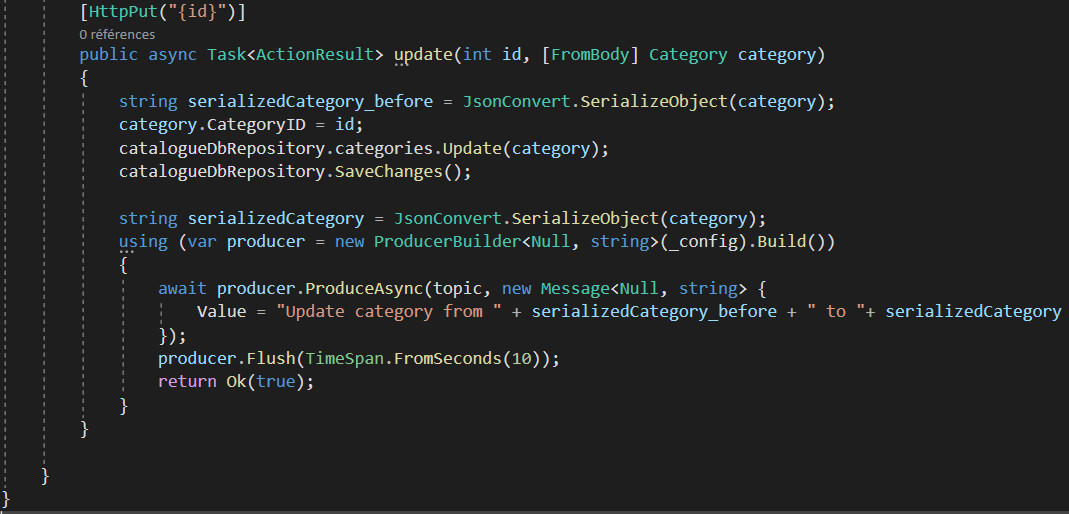
Après, j’ai créé les 3 contrôleurs pour les 3 modèles :

* Le contrôleur des catégories : j’ai créé des méthodes habituelles comme afficher, ajouter, modifier et supprimer une catégorie. J’ai également intégré au même sens une méthode pour afficher la liste des catégories et une autre méthode pour afficher les produits d’une catégorie.



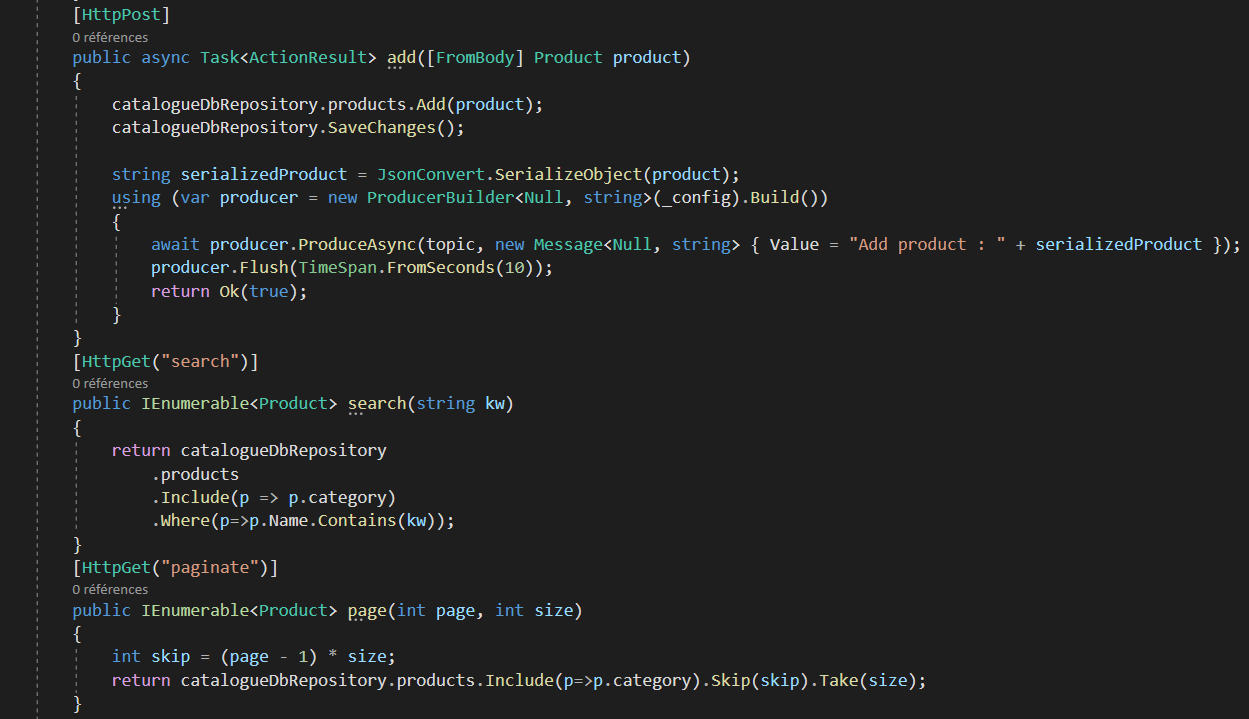




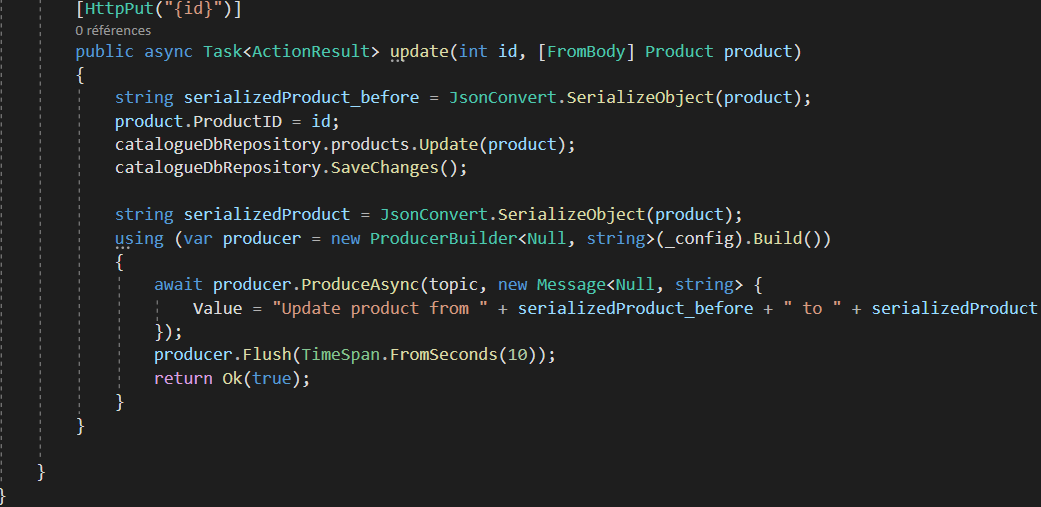


* Le contrôleur des produits : j’ai créé des méthodes habituelles comme afficher, ajouter, modifier et supprimer un produit. Dans le même sens, j’ai intégré une méthode pour faire une recherche en fonction de désignation de produit, une autre méthode pour la pagination de la liste des produits.



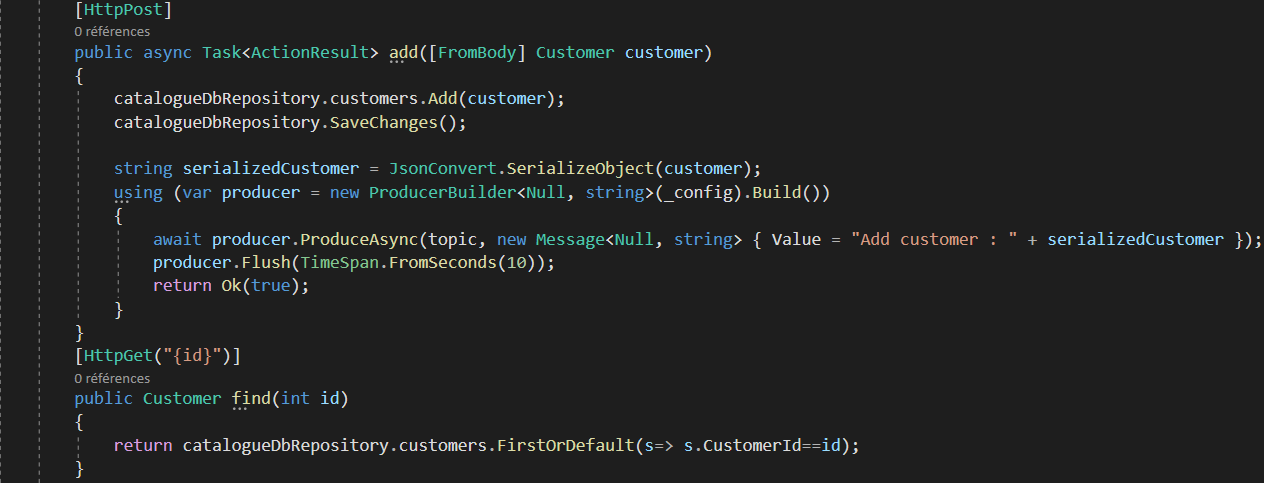


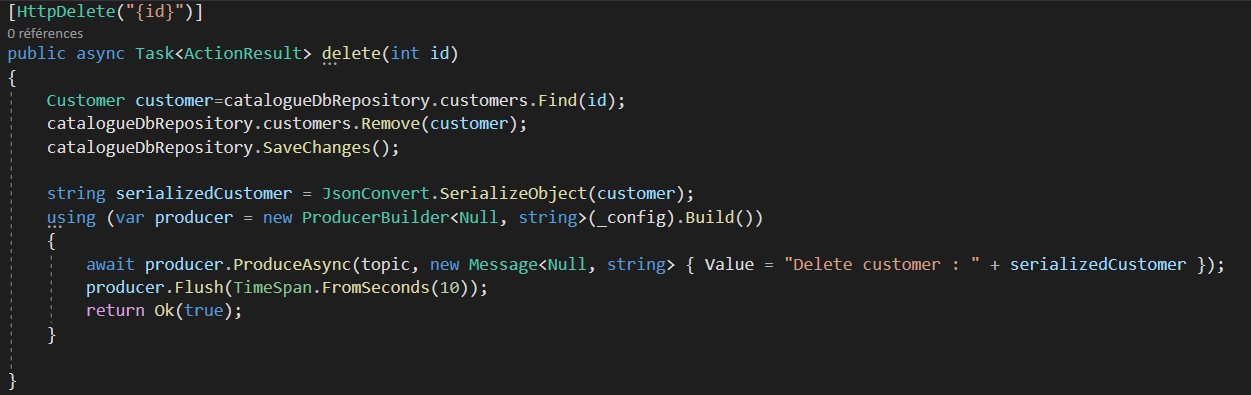


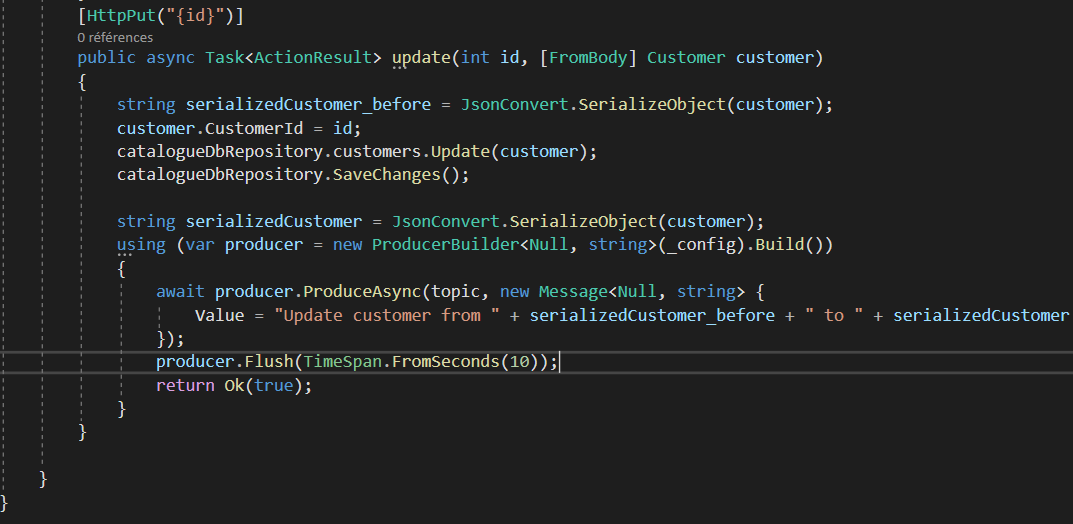


* Le contrôleur des clients : j’ai créé des méthodes habituelles comme afficher, ajouter, modifier et supprimer un client.

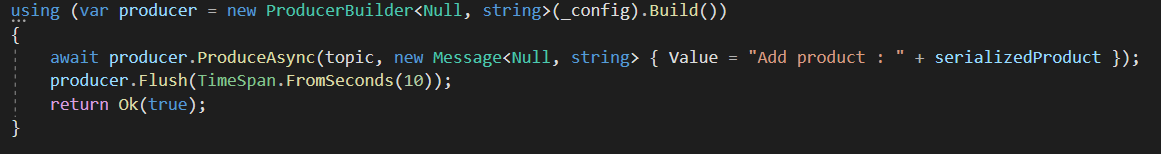






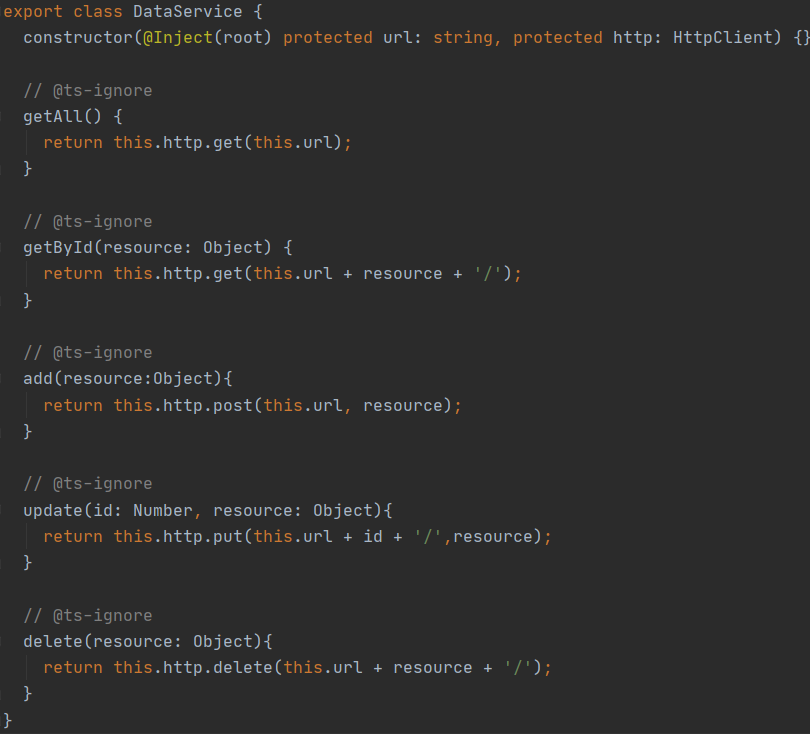


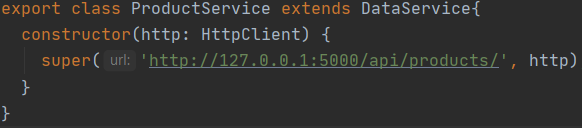
Normalement, la configuration suivante sur les méthodes d’ajouter, modifier et supprimer permet de sérialiser la classe et d’envoyer vers le broker Kafka :

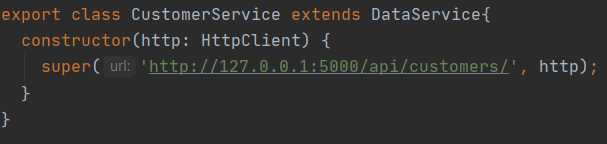


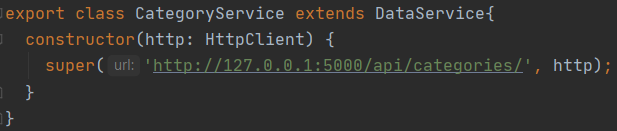
Après la création de l’application rendu serveur qui permet de générer l’API, j’ai commencé la réalisation de l’application rendu client avec la technologie **Angular**.

La première étape concerne la création des services. Le service Data sera crée comme une classe généraliste (méthodes communes) pour les 3 autres services :









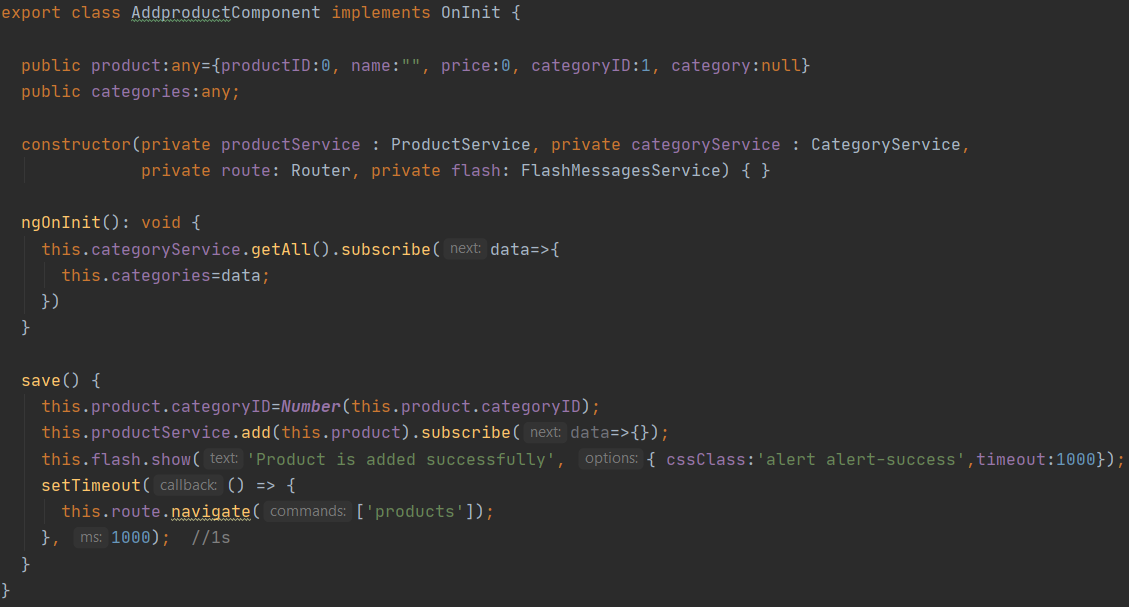
Deuxième étape, j’ai ajouté les components :

* + Component des catégories :

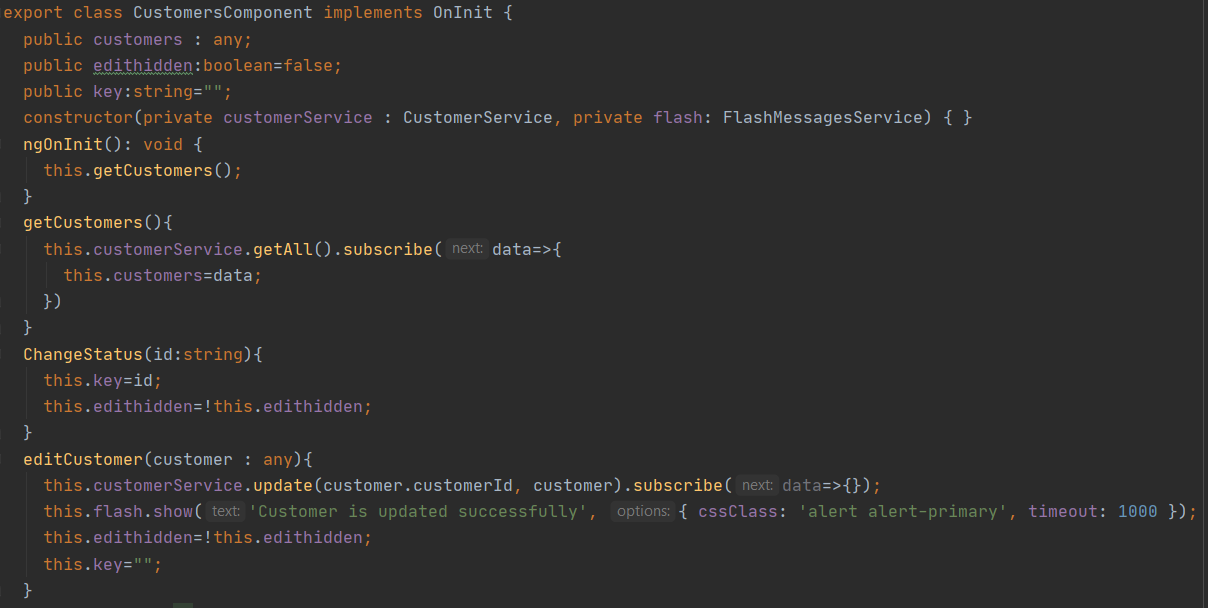


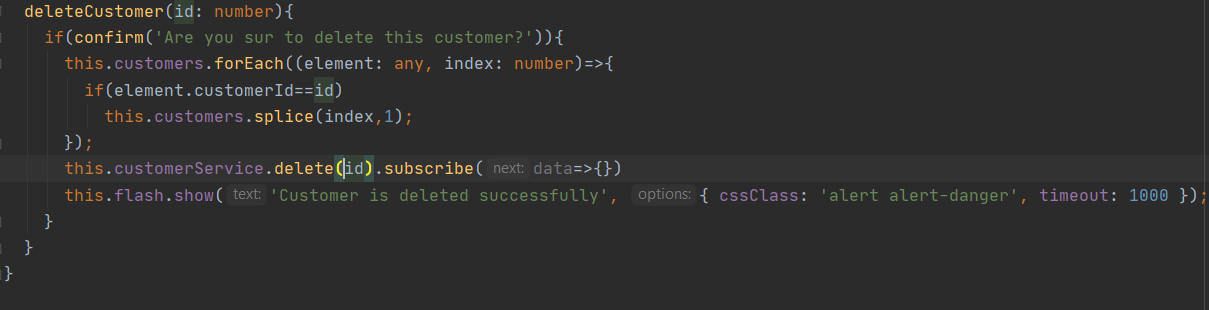


* + Component d’ajouter une catégorie :



* + Component des clients :

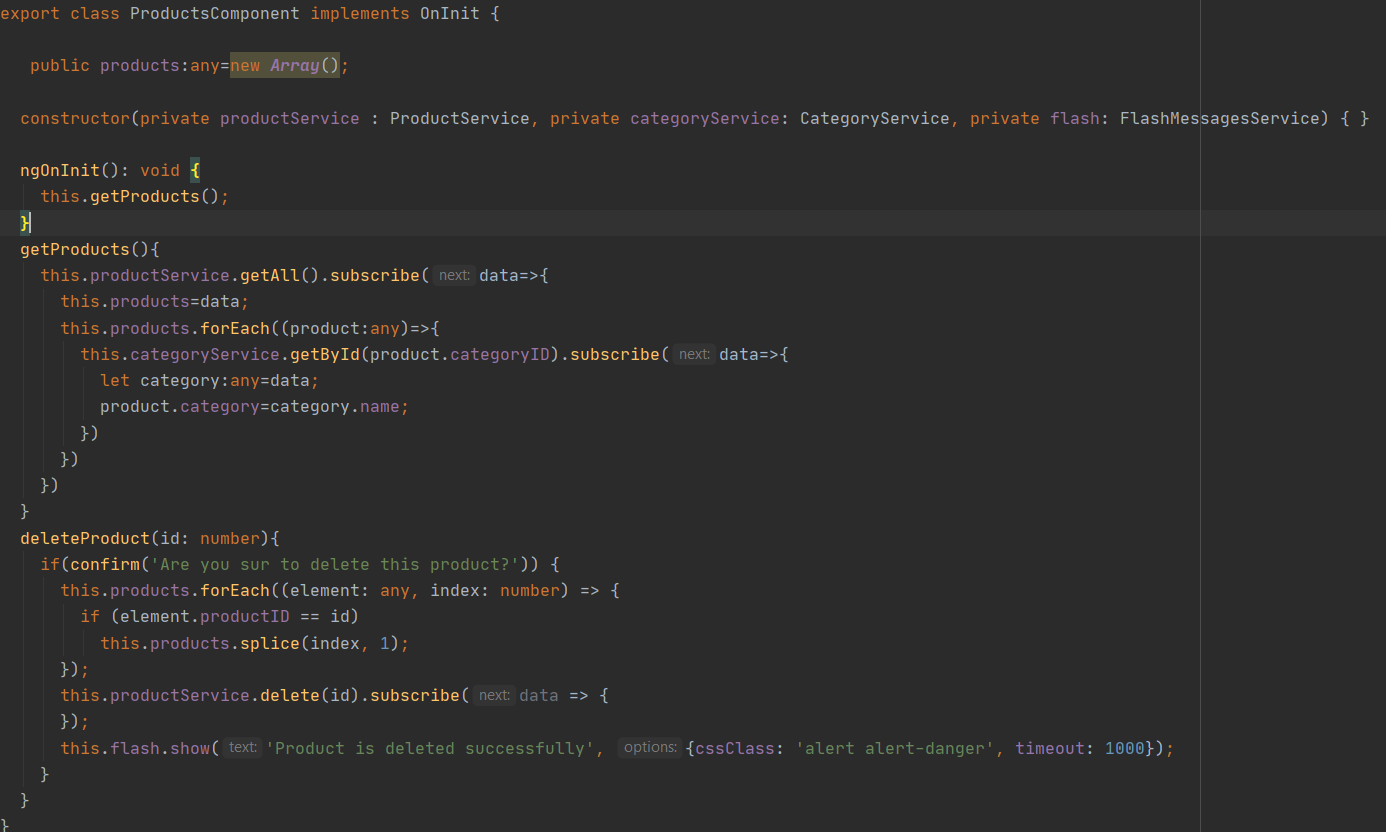




* + Component d’ajouter un client :



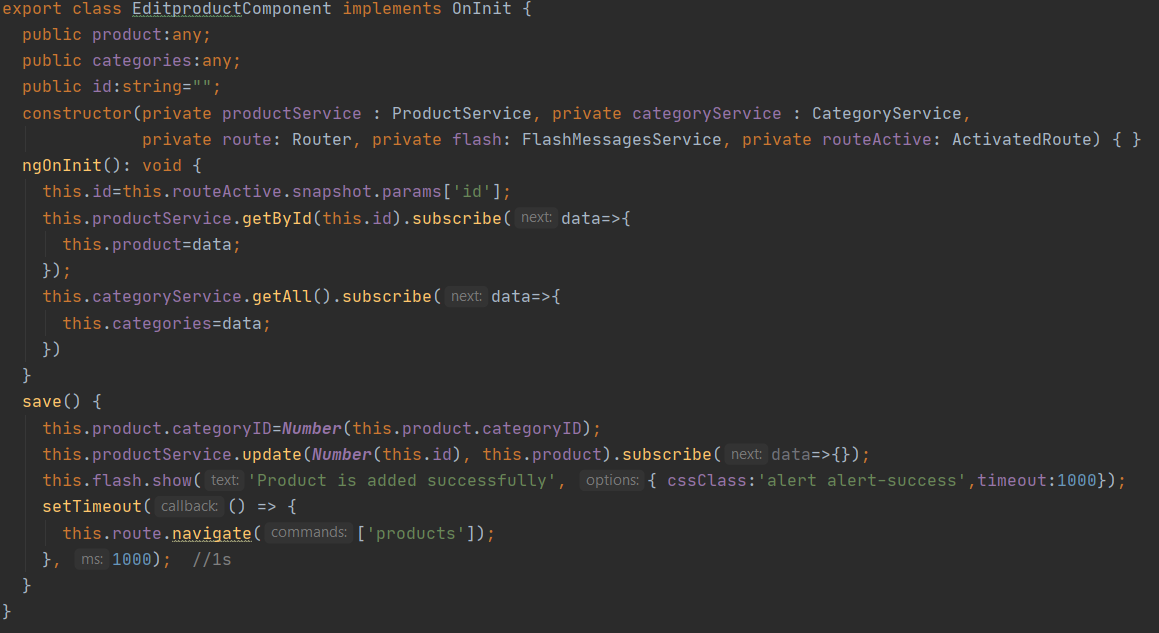
* + Component des produits :



* + Component d’ajouter un produit :

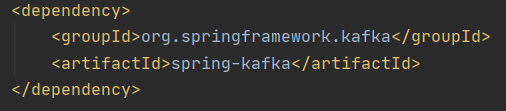


* + Component de modifier un produit :

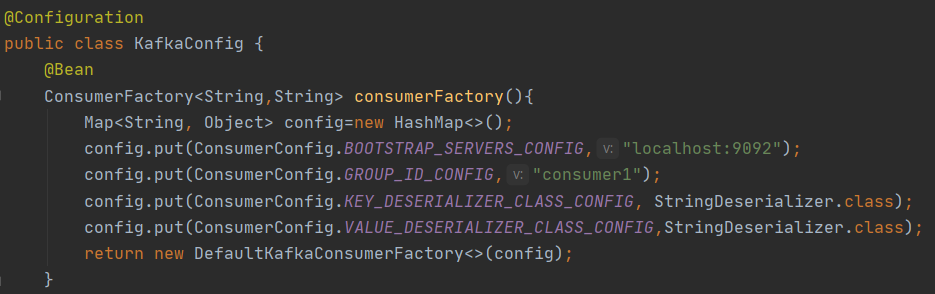


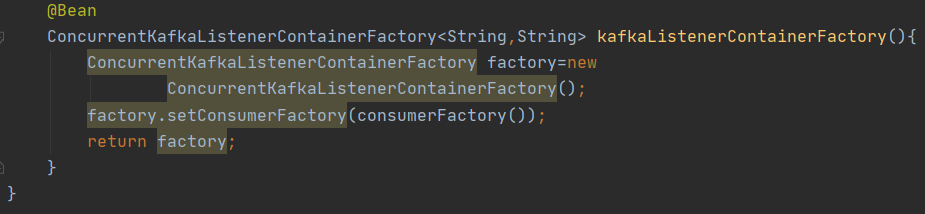
Enfin, j’ai réalisé une microservice avec Spring qui permet de consommer les topics des évènements (ajouter, modifier et supprimer) et les sauvegarder dans un fichier log.

Pour utiliser Kafka dans Spring, il faut importer sa bibliothèque :

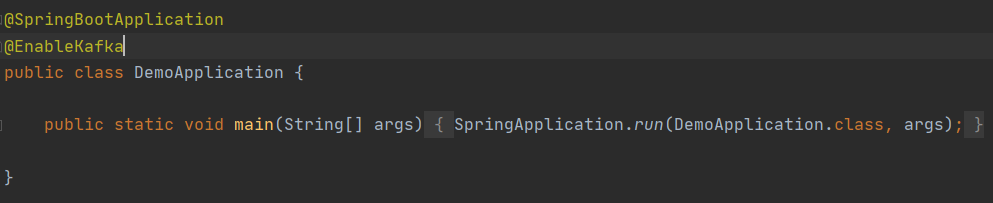


Après, il faut créer une configuration de Kafka :





Et ajouter l’annotation @EnableKafka à la classe exécutable de Spring :



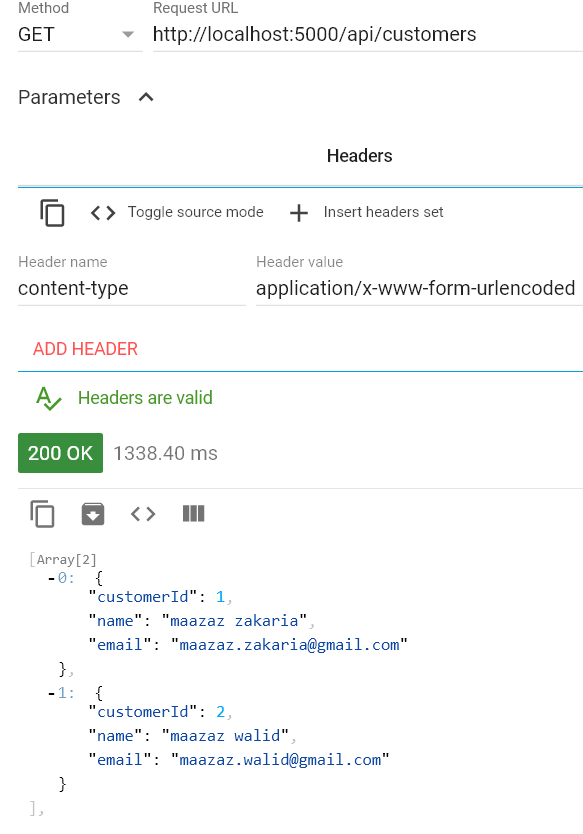
Et finalement, il faut créer un consommateur de Kafka :



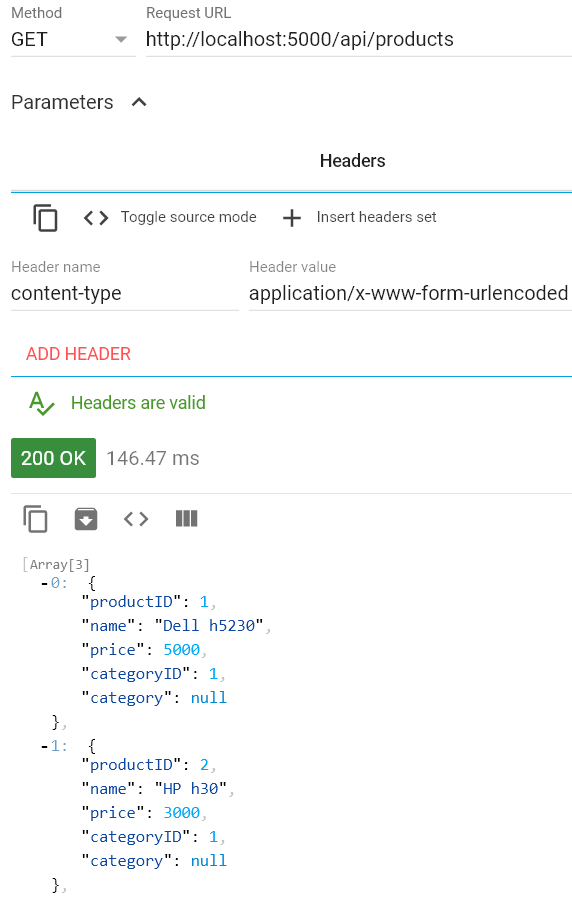
1. **Captures d'écrans relatives à l'exécution**

**4-1- Tester les API Rest en utilisant un Client REST**

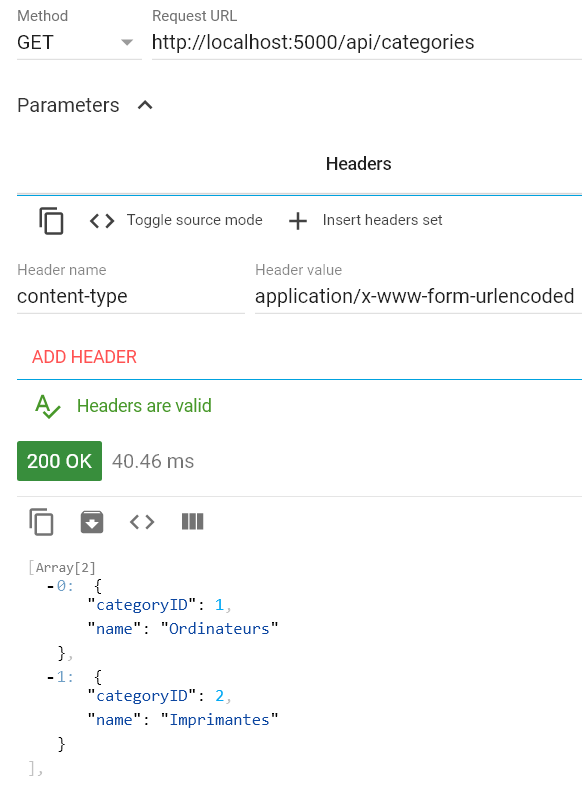
La liste des clients :



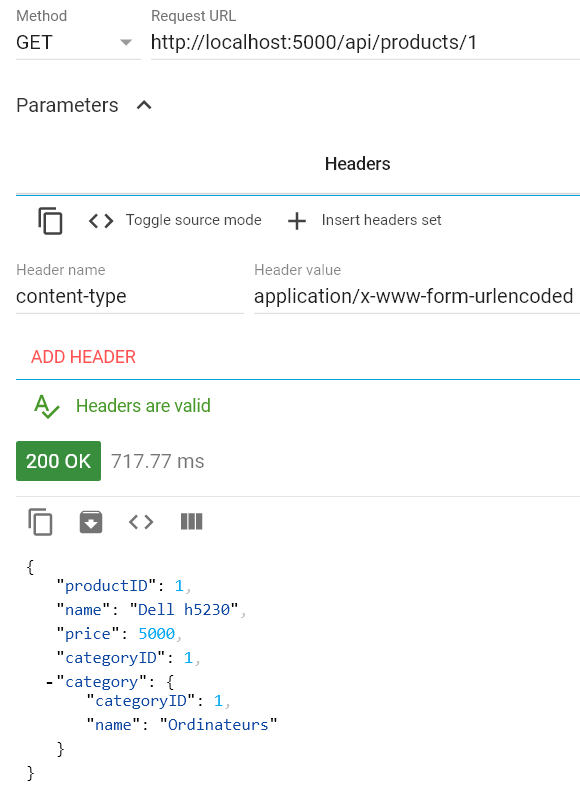
La listes des produits :



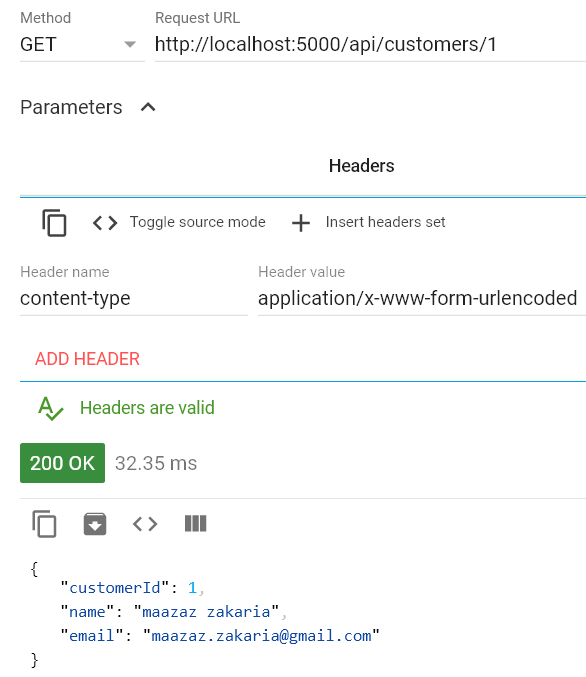
La listes des catégories :



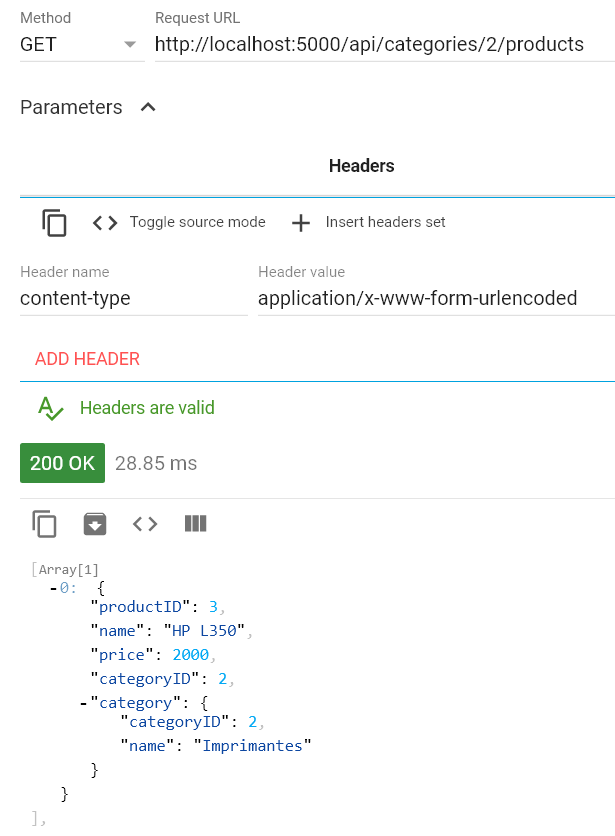
Pour afficher un produit :



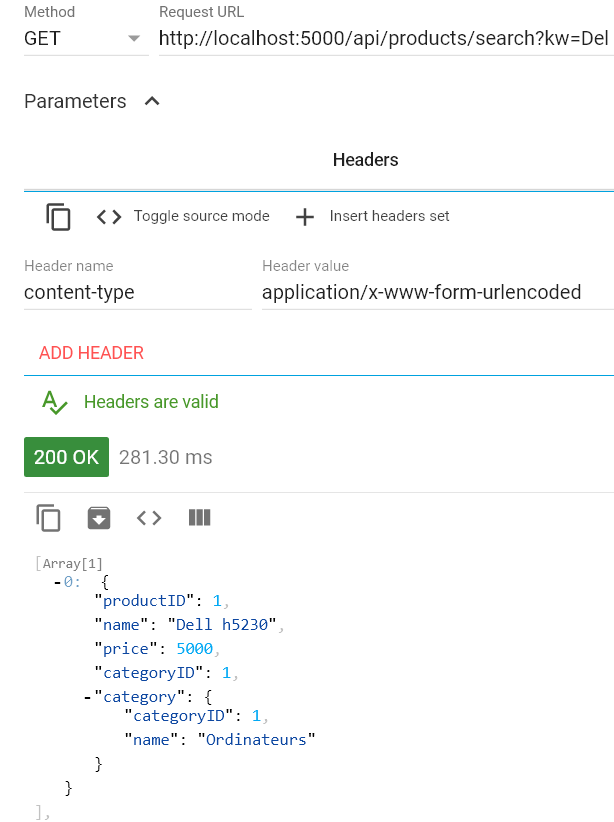
Pour afficher un client :



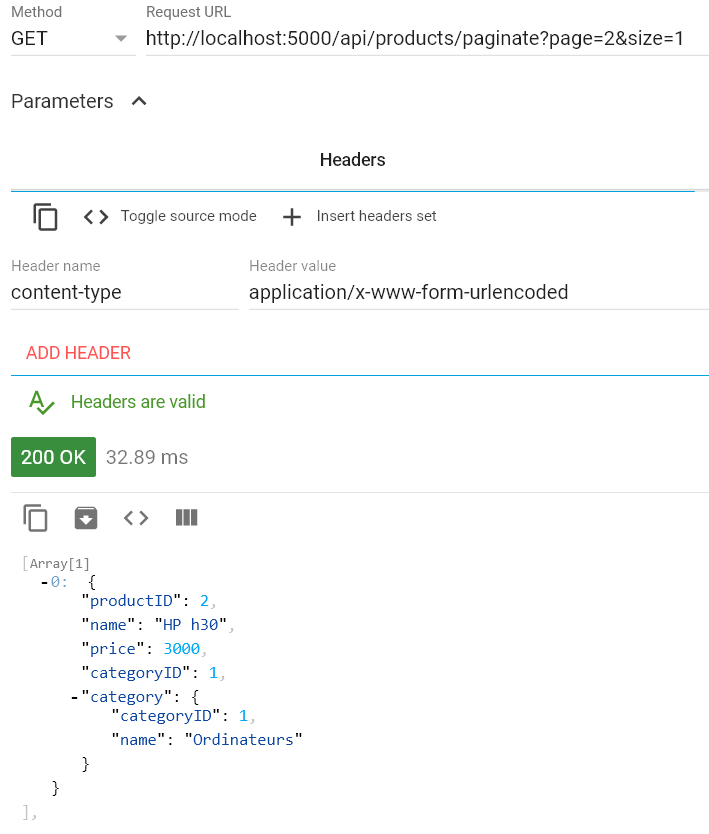
Pour afficher les produits d’une catégorie :



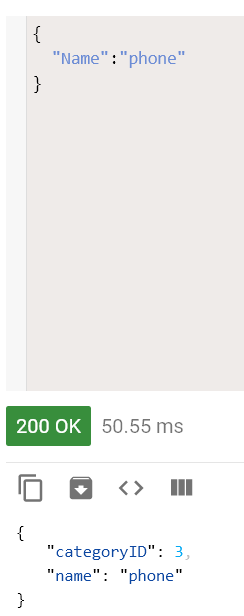
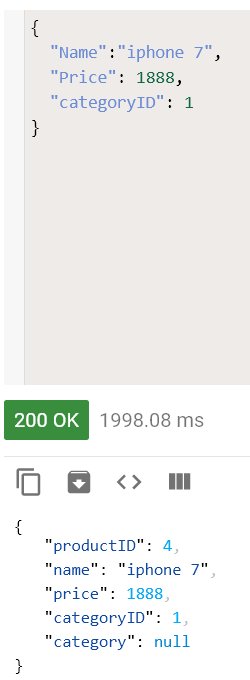
Pour faire une recherche d’un produit :



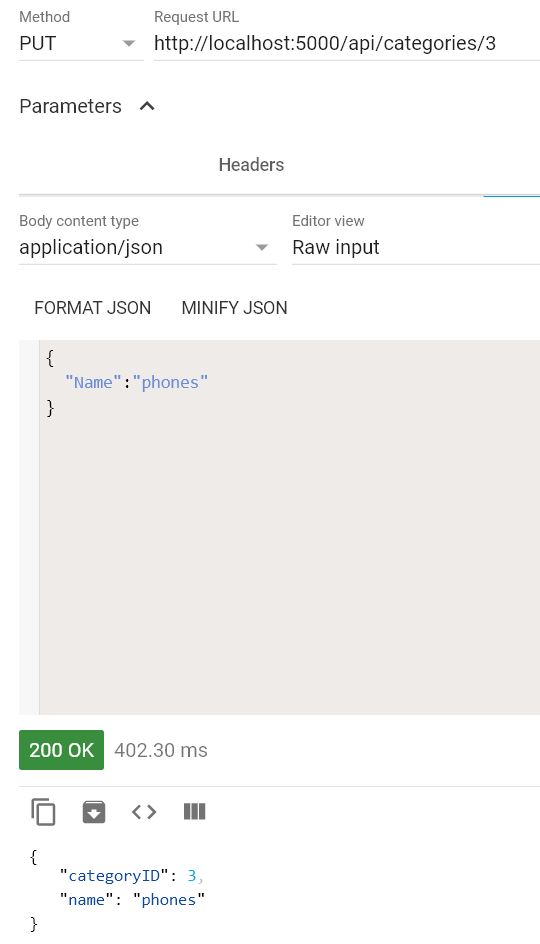
Utiliser la pagination dans l’affichage de la liste de produits :



Ajouter un produit et/ou une catégorie :



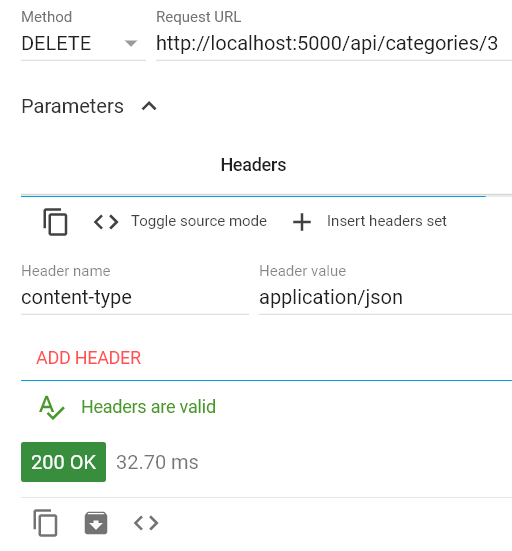
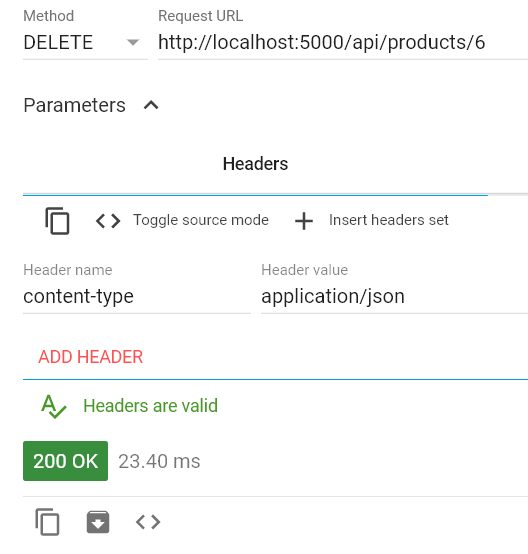
Modifier une catégorie :



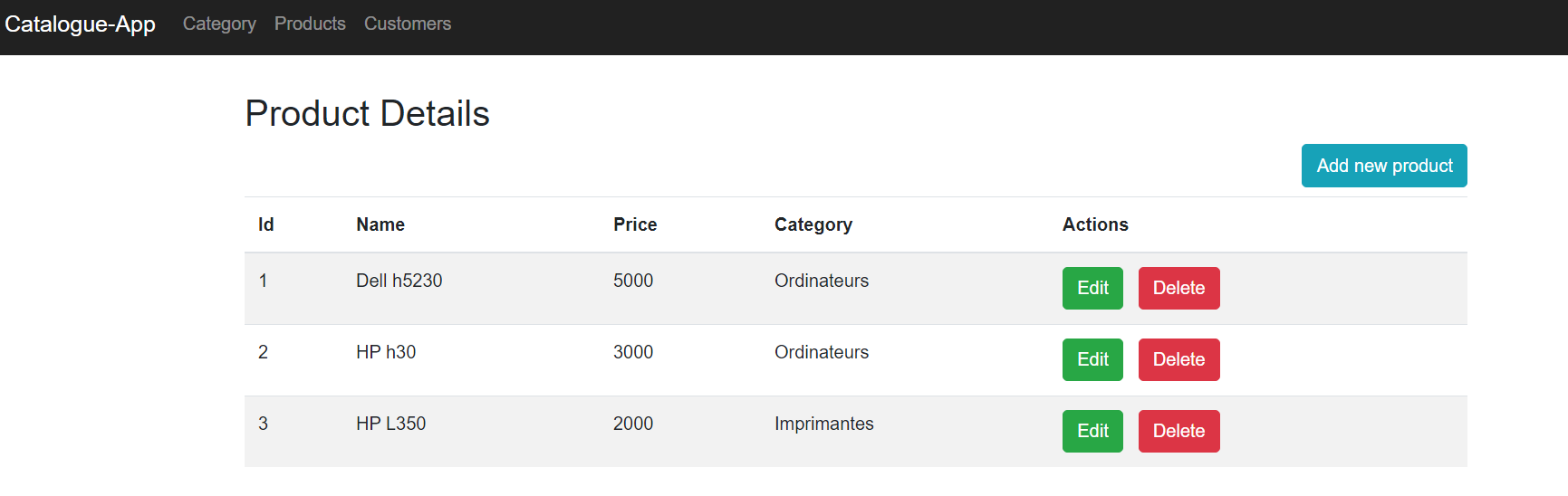
Modifier un produit :

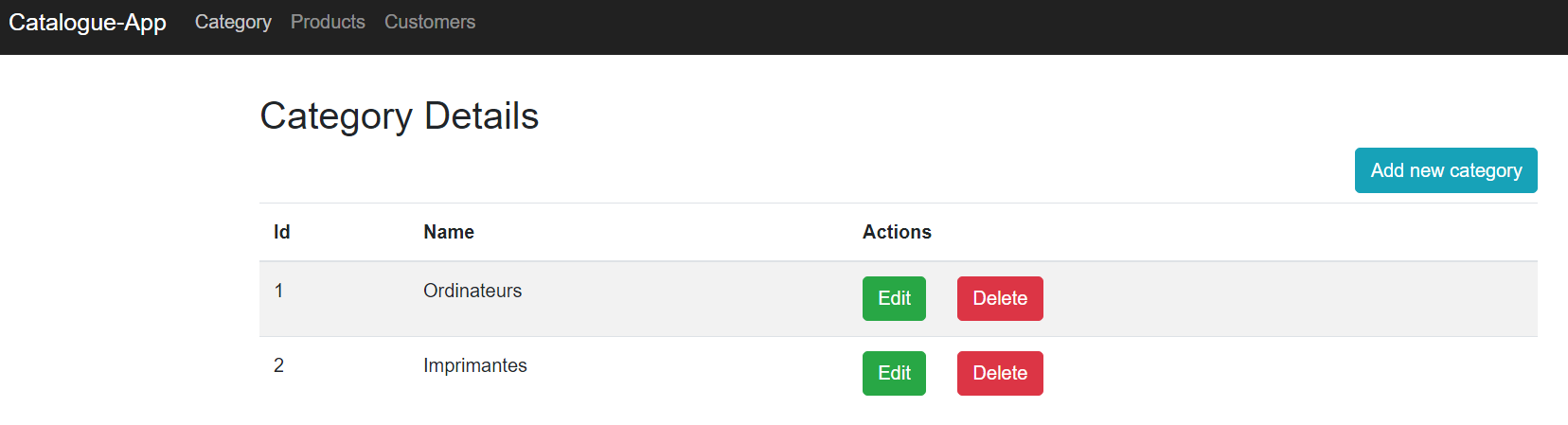


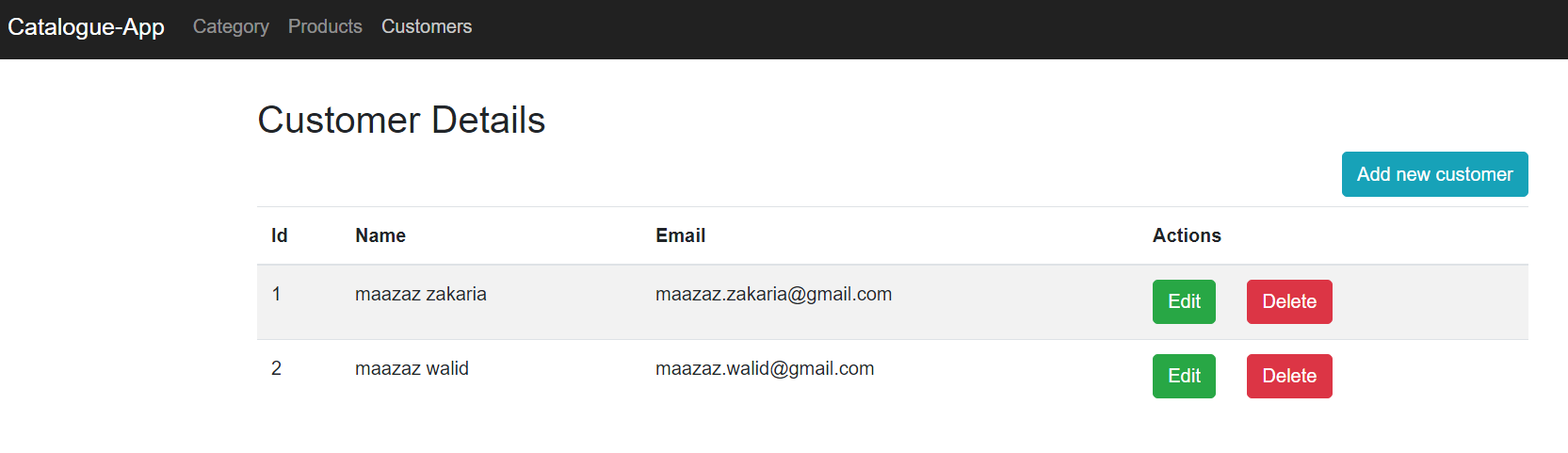
Supprimer un produit ou/et une catégorie :



**4-2- Utilisant Angular avec microservice de KAFKA**

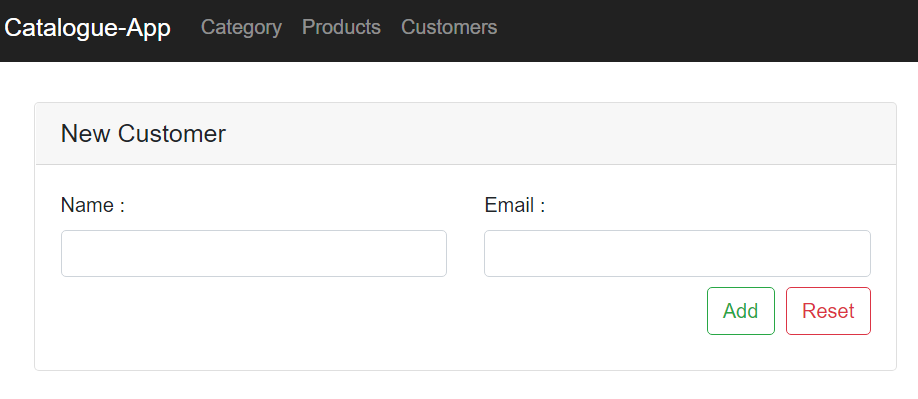


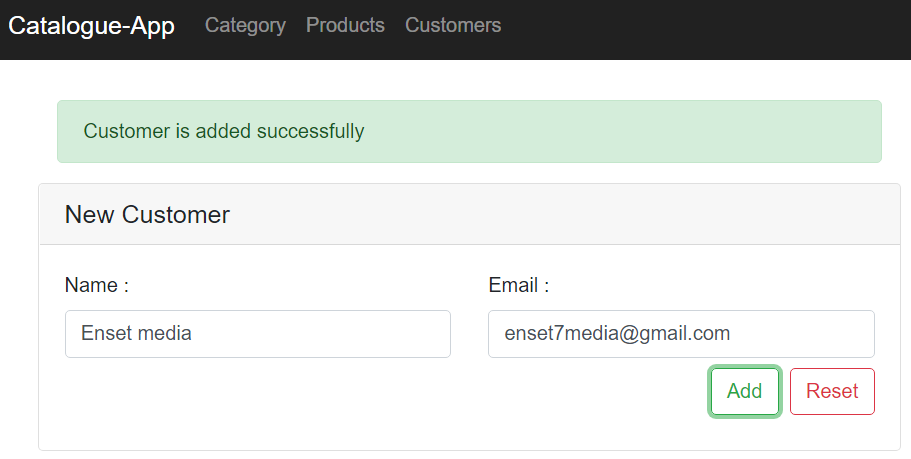


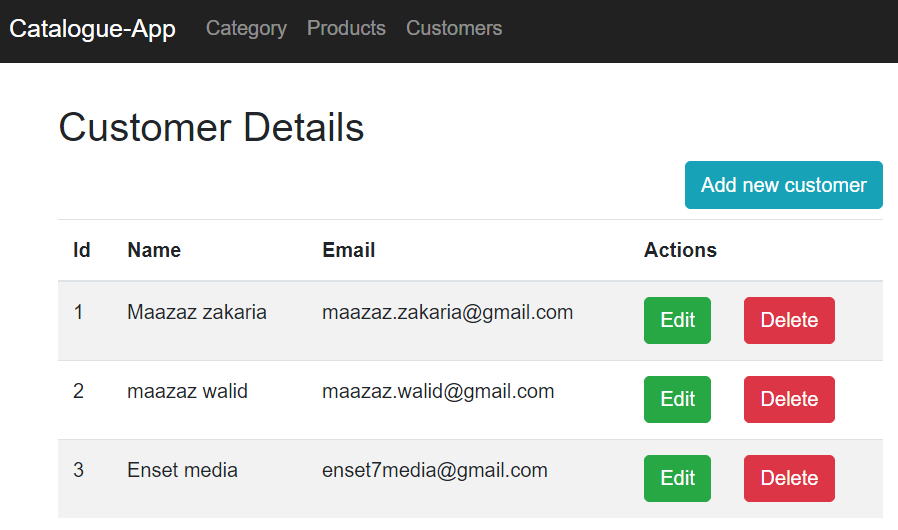


En utilisant en même le consommateur de Kafka, on peut suivre tous les événements traitements effectués avec les méthodes ajouter, modifier et supprimer et leur stocker dans un fichier de log. Prenons par exemple :

* Ajouter un client :



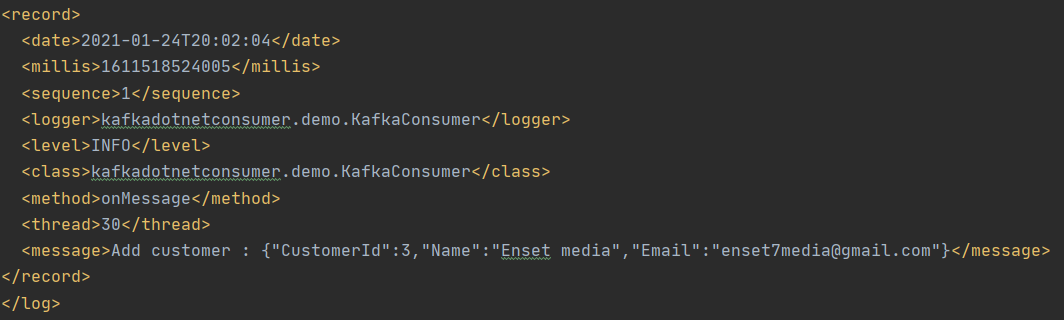




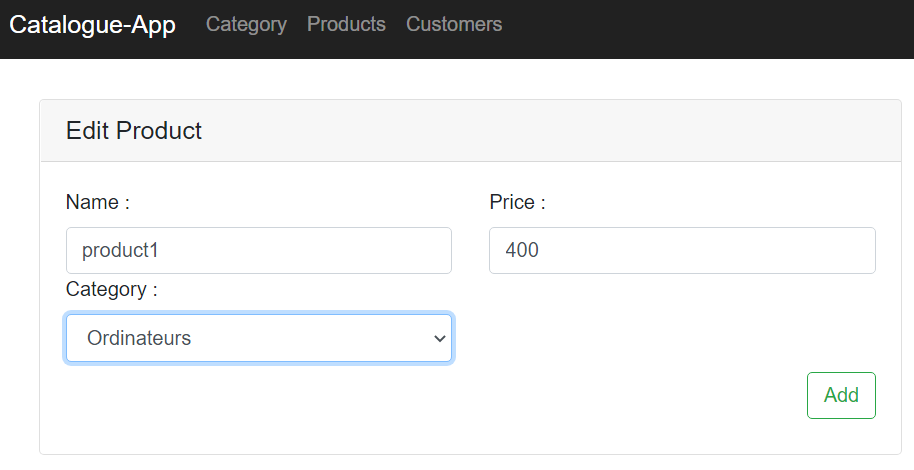
Sur la console du consommateur de Kafka :

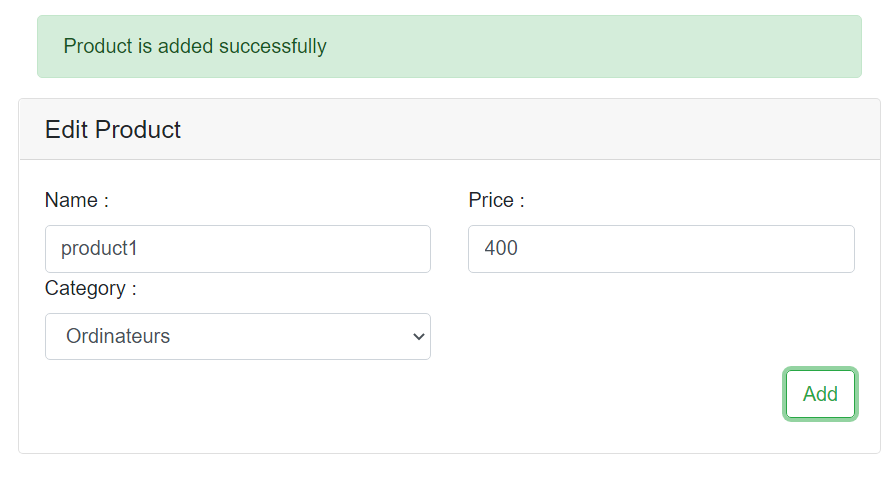


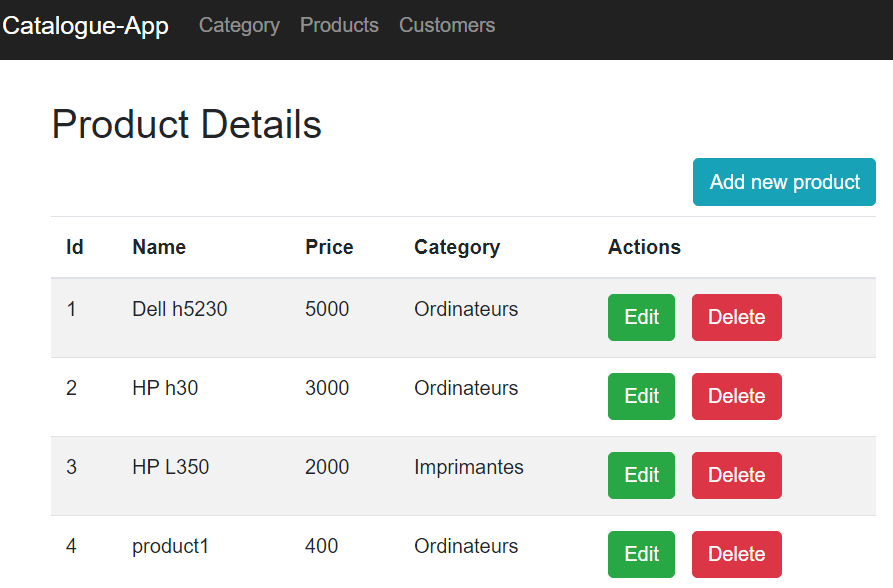
Sur le fichier de logger :



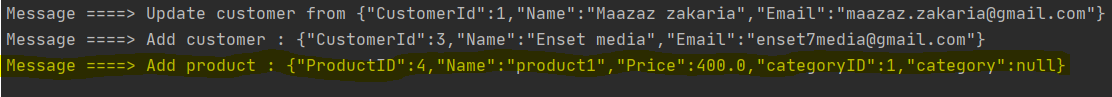
* Ajouter un nouveau produit :



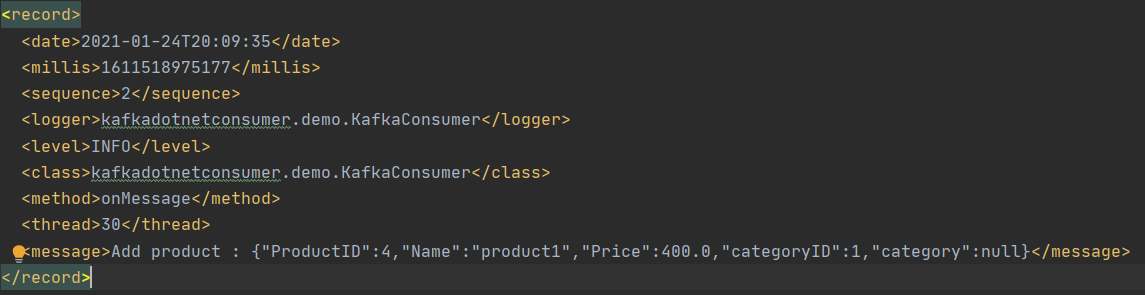




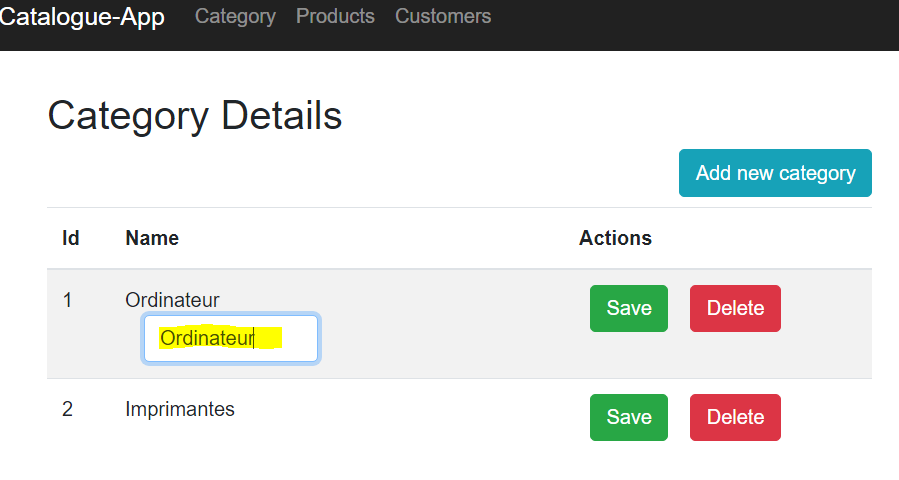
Sur la console du consommateur de Kafka :

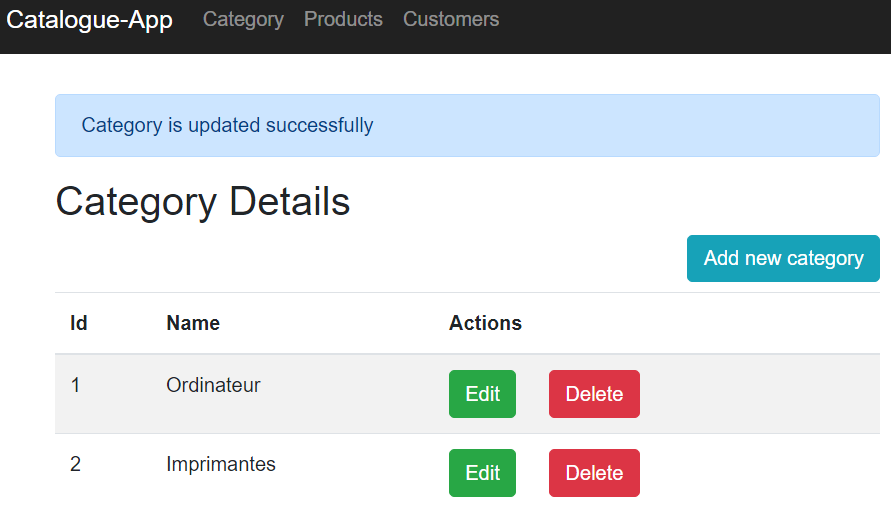


Sur le fichier de logger :

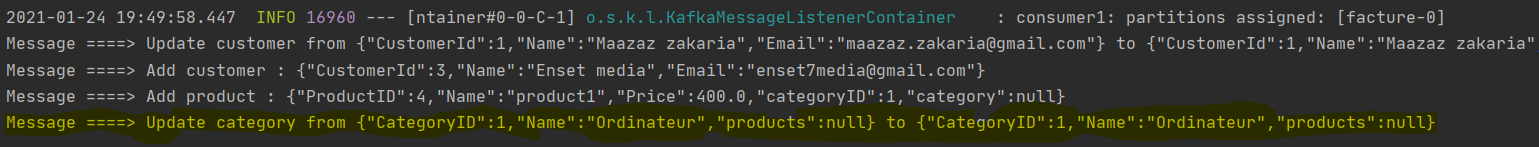


* Modifier une catégorie :

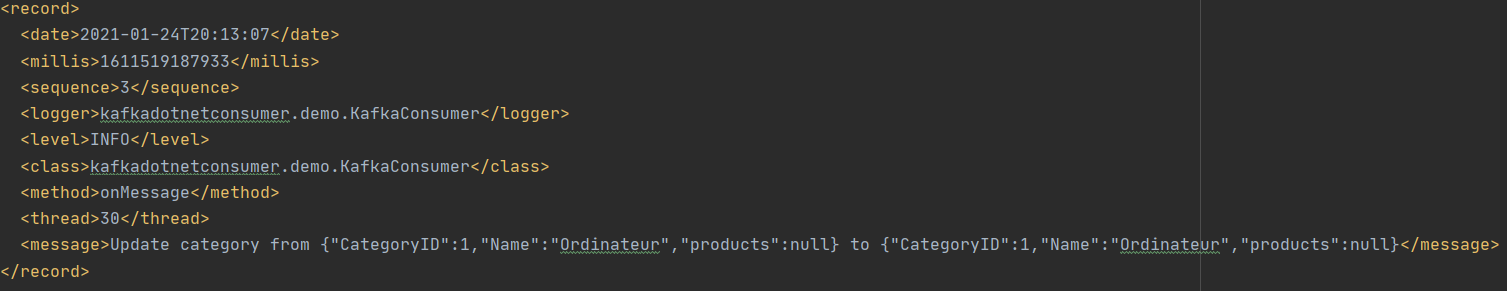




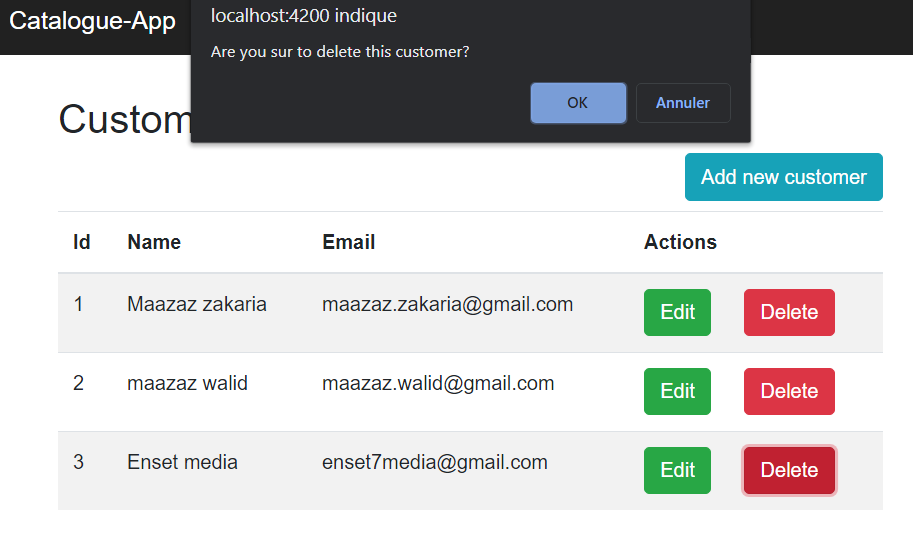
Sur la console du consommateur de Kafka :

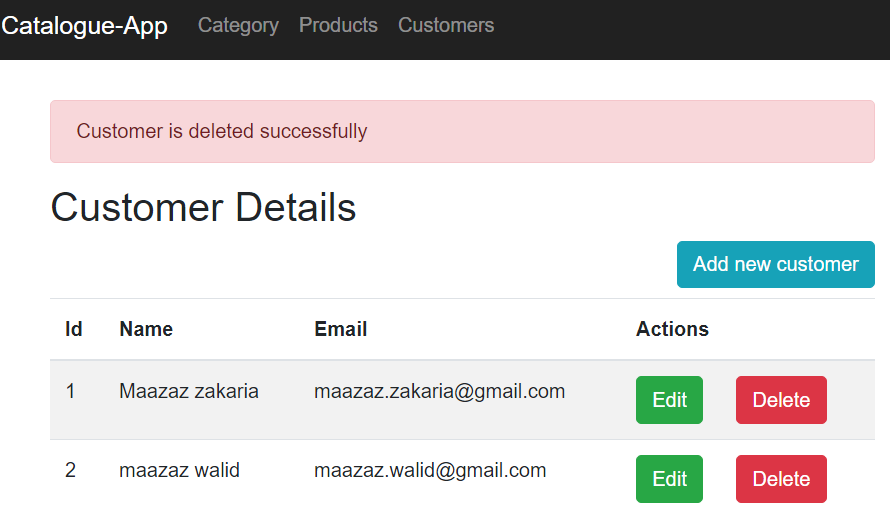


Sur le fichier de logger :

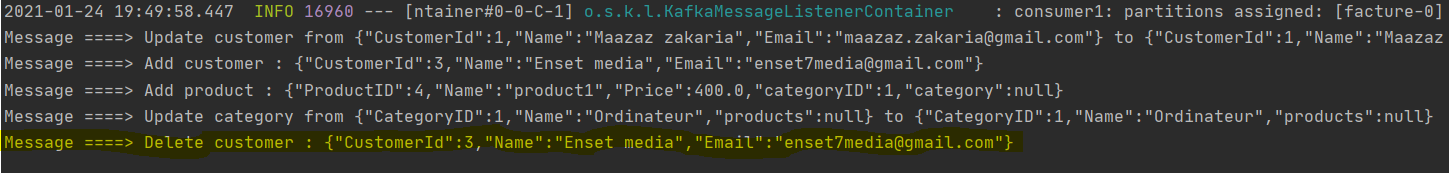


* Supprimer un client :

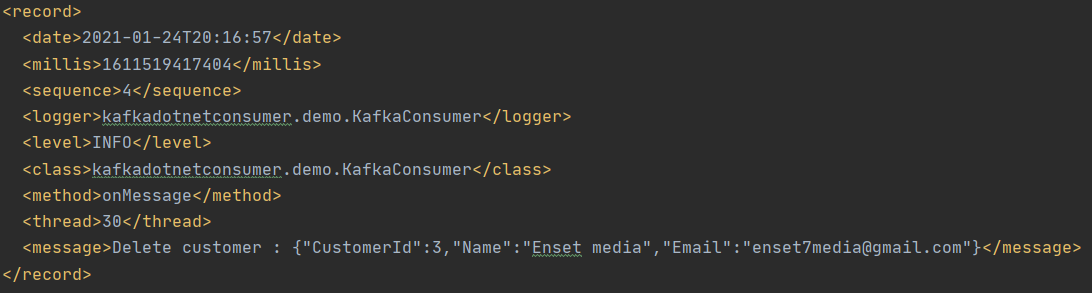




Sur la console du consommateur de Kafka :



Sur le fichier de logger :



**Conclusion**

L'architecture .NET a un impact important sur la réutilisabilité d'une application et sur sa capacité à évoluer. Cet exemple nous permet d’approprier les outils conceptuels et logiciels de l'environnement du Framework .NET pour concevoir efficacement une architecture applicative à la fois robuste et flexible.