



Projet

*Conception et réalisation d'un
système distribué pour la gestion des
commandes*

Fait par

KACIMI IMAD OUALID

21910835



Année universitaire
2019/2020

Introduction

I. Partie Conception	3
Introduction	3
Diagrammes utilisés	3
II. Partie réalisation	8
Introduction	8
Architectures	8
Schéma de communication entre micro service :	9
Interface « Front-end »	14
Déploiement docker	19
Déploiement kubernetes:	21
Figure 1: diagramme de cas d'utilisation global	3
Figure 2: Diagramme de classe	4
Figure 3:base de données des microservices	7
Figure 4: Affichage des commandes	9
Figure 5 : Affichage des produits	10
Figure 6: affichage des customers	10
Figure 7: modification ou suppression d'un produit	11
Figure 8: modification ou suppression d'un customer	11
Figure 9:Authentification	12
Figure 10: Get tous les produits	12
Figure 11: Ajouter un produit.....	13
Figure 12: Modifier un produit.....	13
Figure 13: Supprimer un produit.....	14
Figure 14: AUTHENTIFICATION.....	15
Figure 15 : tableau des commandes, produits, customers et administrateurs	17
Figure 16: ajout d'une commande pour un customer existant.....	18
Figure 17: ajout d'une commande pour un nouveau customer	19
Figure 18 Docker hub.....	19
Figure 19: Dockerfile du microservice client	20
Figure 20: Dockerfile du front-end.....	20
Figure 21: déploiement du service client dans docker.	21
Figure 22 :déploiements des différents conteneurs.	22
Figure 23: déploiement du client dans kubernetes.	22

Introduction

Le projet que nous allons réaliser est la conception et la réalisation d'une application web dédié à « l'administration de gestion de commandes ». Ayant comme objectif de permettre à un magasin d'avoir :

- L'insertion des de nouveaux produits et de nouveaux clients
- La possibilité de gérer son stock de produit
- La possibilité de réaliser des commandes.

Le présent rapport sera composé de deux parties à savoir :

- Conception
- Réalisation

I. Partie Conception

Introduction

La réalisation des systèmes informatisés se concrétise en différentes étapes comme "**la conception**" qui est définie par des diagrammes permettant de simplifier la lecture et la traduction du projet avec une méthodologie de conception Merise ou UML.

Dans notre cas nous allons utiliser l'approche UML.

Diagrammes utilisés

Diagramme de cas d'utilisation « use case »

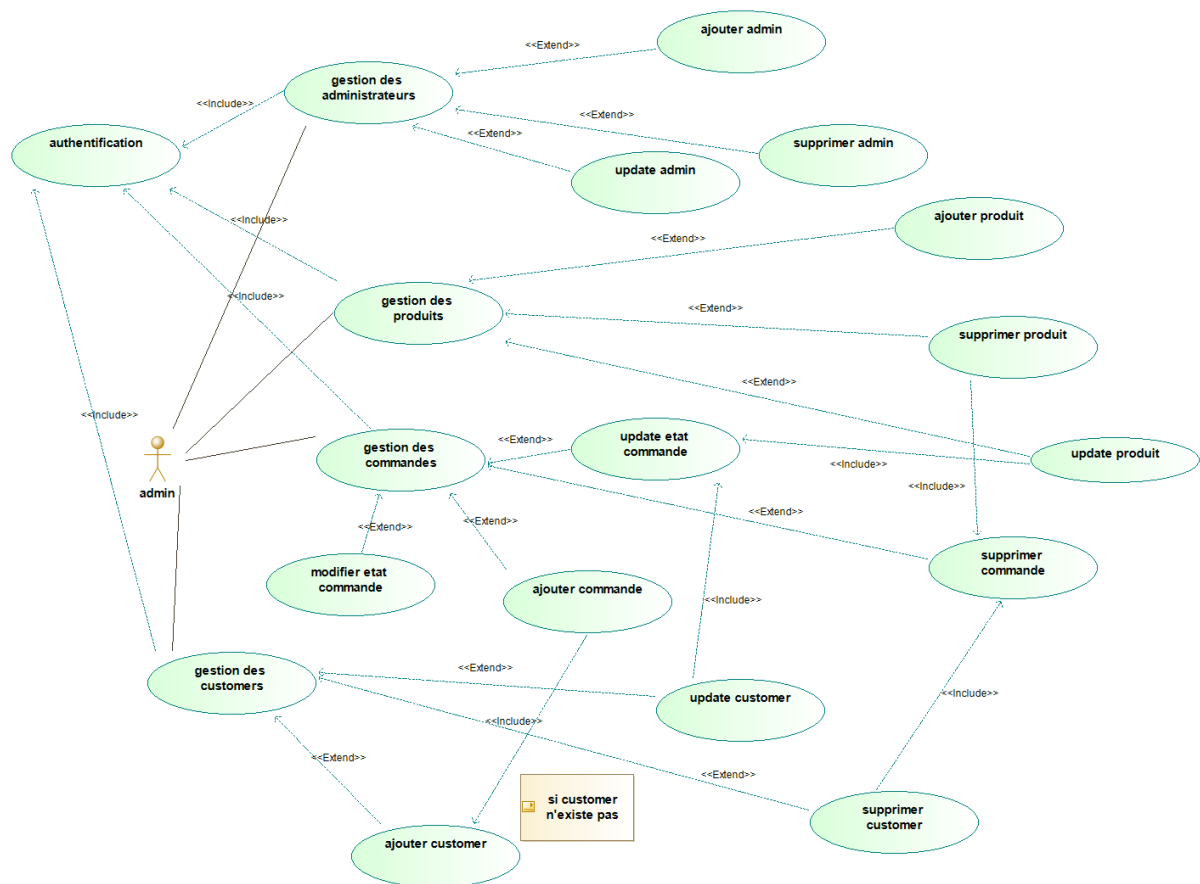


Figure 1: diagramme de cas d'utilisation global

D'après le diagramme de la figure précédente, nous constatant que les fonctionnalités réalisées par l'administrateur sont divisées en quatre familles à savoir :

Gestion des administrateurs

L'utilisateur de l'application pourra ajouter, supprimer ou modifier d'autres administrateurs.

Gestion des produits

L'utilisateur pourra gérer son stock de produit, il aura la possibilité d'ajouter, supprimer ou modifier des produits. La suppression d'un produit produira la suppression de toutes les commandes relatives à ce produit.

Gestion des customers

L'utilisateur pourra gérer ses clients, il aura la possibilité d'ajouter, supprimer ou modifier ses clients, La suppression d'un client produira la suppression de tous les commandes faites par ce dernier.

Gestion des commandes

L'utilisateur pourra gérer les commandes de ses clients, en ajoutant, supprimant ou modifiant des commandes, l'ajout d'une commande peut être attribué directement à un client déjà existant, ou pourra déclencher la création d'un nouveau client si le numéro de téléphone mentionné dans le formulaire d'ajout d'une commande n'existe pas dans notre base de données.

Diagramme de classes

La réalisation du diagramme de classe s'est faite en suivant les critères suivants :

Un produit appartient à une et une seule Catégorie. Autrement dit, une Catégorie peut être vue comme un identifiant ensembliste de plusieurs produits traitant le même thème.

Un Client peut effectuer une commande de plusieurs Livres et comme un produit à plusieurs exemplaires, il peut être commandé par plusieurs Clients.

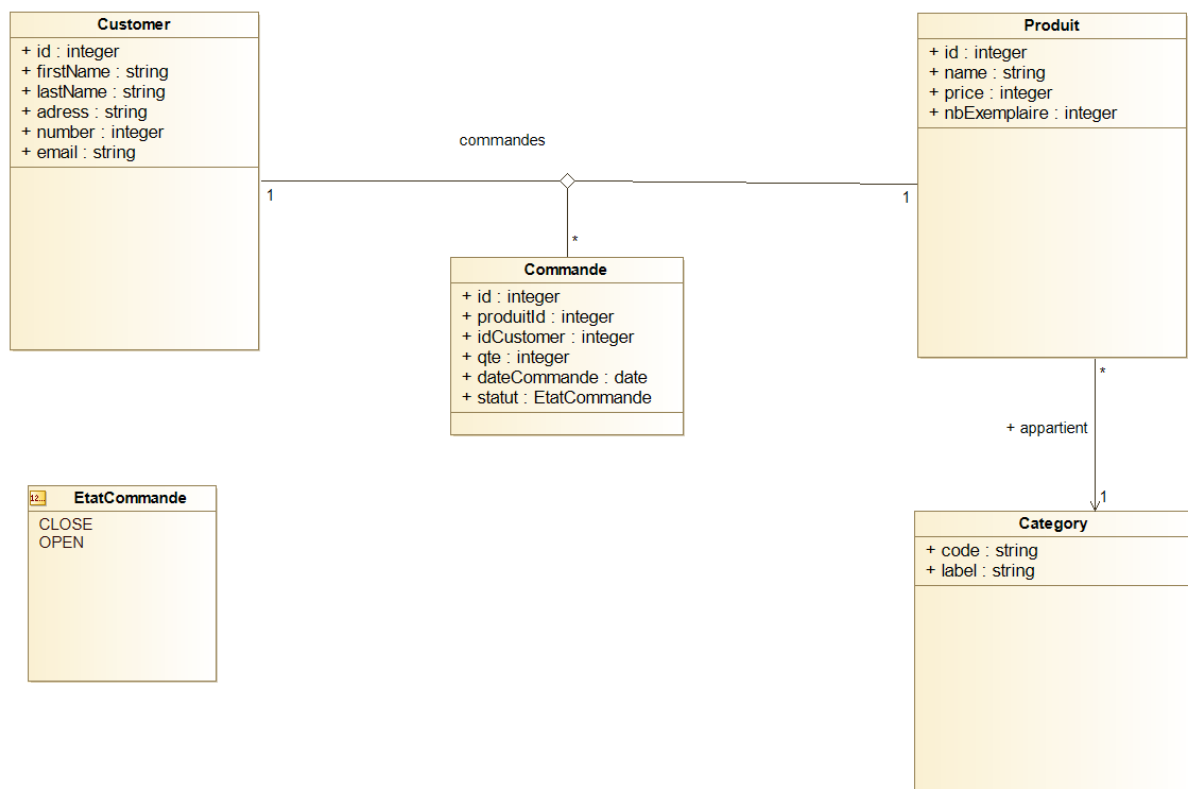


Figure 2: Diagramme de classe

Les règles de passages du diagramme de classe au modèle relationnel

Dans ce qui suit, nous allons présenter les différentes règles de passage, qui nous ont servis lors de l'élaboration du modèle relationnel des données :

- Affecter une table à chaque classe.
- La migration de la clé primaire de la table mère à la table fille, due à une association « un à plusieurs ».
- La représentation d'une association « plusieurs à plusieurs » est effectuée par une table ayant pour clé primaire la concaténation des clés primaires des deux tables associées.
- La représentation d'un héritage complet se fait en intégrant tous les attributs de la classe mère dans ses classes filles si la classe mère est une classe abstraite.

Le schéma relationnel

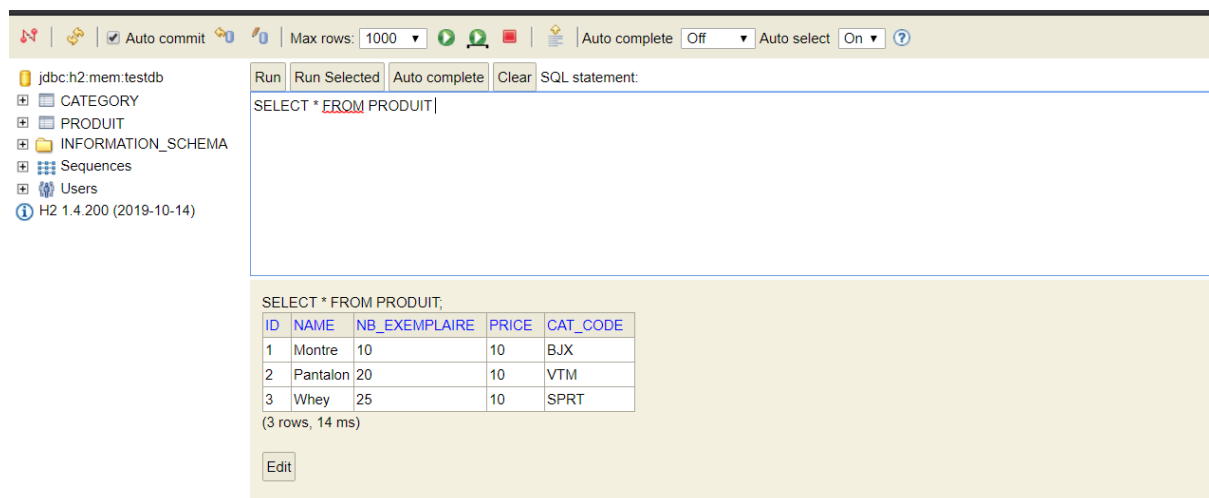
customer(id,first_name,last_name,address,number,mail);

produit(id,name,price,nbExemplaire,code_cat*);

category(code,label);

commande(id,qte,date_commande,produit_id*,customer_id*);

Ci-dessus les base de données in-memory de chaque microservice:



The screenshot shows a database client interface with a sidebar on the left containing a tree view of the database structure: jdbc:h2:mem:testdb, CATEGORY, PRODUIT, INFORMATION_SCHEMA, Sequences, Users, and H2 1.4.200 (2019-10-14). The main area has a toolbar with buttons for Run, Run Selected, Auto complete, and Clear, and a text input for the SQL statement. The SQL statement entered is 'SELECT * FROM PRODUIT;'. Below the input, the results of the query are displayed in a table with 5 columns: ID, NAME, NB_EXEMPLAIRE, PRICE, and CAT_CODE. The table contains 3 rows of data. Below the table, it indicates '(3 rows, 14 ms)' and there is an 'Edit' button.

ID	NAME	NB_EXEMPLAIRE	PRICE	CAT_CODE
1	Montre	10	10	BJX
2	Pantalon	20	10	VTM
3	Whey	25	10	SPRT

jdbc:h2:mem:testdb

CUSTOMER

INFORMATION_SCHEMA

Sequences

Users

H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM CUSTOMER

SELECT * FROM CUSTOMER;

ID	ADDRESS	EMAIL	FIRST_NAME	LAST_NAME	NUMBER
1	Alger	imad@gmail.com	imad	kacimi	12345678
2	alger	zitout@gmail.com	chikou	zitout	55555

(2 rows, 9 ms)

Edit

Figure 3:base de données des microservices

II. Partie réalisation

Introduction

Dans cette partie nous allons montrer les différentes étapes de chaque partie à réaliser avec des illustrations en abordant les points suivants :

- Architecture
- Interface « front-end »
- Docker
- Kubernetes

Architectures

Nous avons découpé notre application en plusieurs micro services en utilisant l'architecture netflix.

- ***Micro service produit***

Ce service gère les produits et leurs catégories, se compose de deux classe produit et category. Ce service gère les actions suivantes : récupération des produits depuis la bdd, récupération d'un produit spécifique suivant l'id, ajout d'un nouveau produit, modification d'un produit et la suppression d'un produit.

- ***Micro service customer***

Ce service permet de gérer les clients se compose de la classe customer, il offre les actions suivantes: récupération de la liste des customers, récupération d'un customer spécifique selon l'id, l'ajout d'un nouveau client, la modification ainsi que la suppression d'un client.

- ***Micro service commande***

Ce service gère les commandes, se compose de la classe commande, il offre les actions suivantes : récupérations des commandes ou d'une commande spécifique à un id, l'ajout d'une nouvelle commande, la modification ainsi que la suppression d'une commande.

- **Micro service client-commande :**

Ce service permet de faire le lien entre nos 3 microservices, il se compose de la classe fullcommande, en effet ce microservice va nous permettre de récupérer toutes les informations relatives à une commande précise, et nous permettra ainsi d'afficher en plus de l'id du produit et customer les autres informations dédiées à ces derniers.

- **Micro service eureka:**

Ce microservice est notre registre eureka qui permettra à nos autres microservices de s'enregistrer.

- **microservice gateway (zuul):**

Ce microservice représente le point d'entrée vers nos autres microservices, de plus il nous permettra les sécuriser en utilisant spring sécurité jwt, il nous offrira aussi les opérations relatives à la gestion des administrateurs décrites dans le diagramme des cas d'utilisation Figure 1.

Nous tenons à préciser que chaque microservice utilise rebbon pour l'équilibrage et le choix d'un service parmi ceux existant.

Schéma de communication entre micro service :

Nous allons illustrer maintenant quelques schémas de communication entre nos micro service.

1. Affichage des commandes

La figure 3 illustre les différents appels https permettant d'afficher la liste des commandes affichants tous détails, comme les flèches l'indiquent notre client frontend envoie une requête http vers notre gateway qui contactera le service client-commande qui va se charger de la récolte d'information. Tout d'abord il récupère la liste des commandes (3,4) pour ensuite solliciter les informations des customers et clients relatifs aux commandes (5,6,7,8) et enfin retourner la liste des commandes full info.

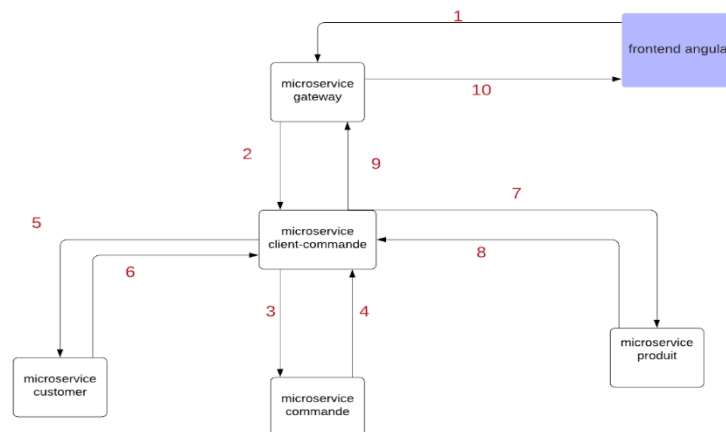


Figure 4: Affichage des commandes

2. Affichage des produits

La figure 4 illustre les appels https permettant d'afficher la liste des produits.

Le client frontend envoie une requête get qui sera redirigé par la gateway vers le microservice produit. ce dernier retournera la liste des produits.

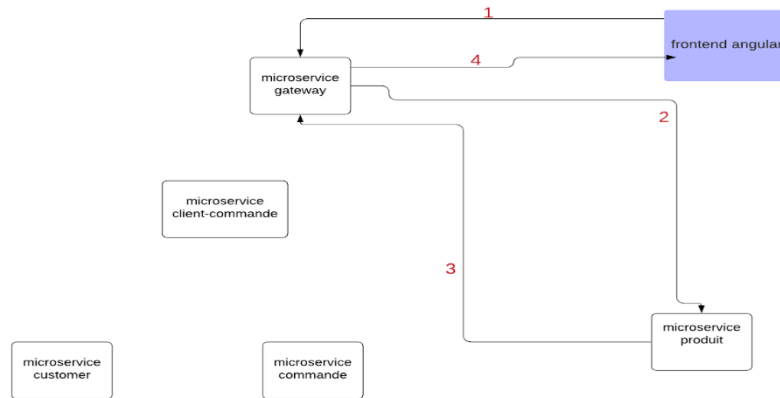


Figure 5 : Affichage des produits

3. Affichage des customers

La figure 5 illustre les appels https permettant d'afficher la liste des customers. Le client frontend envoie une requête http get qui sera redirigé par la gateway vers le microservice customer. Ce dernier retournera la liste des customers.

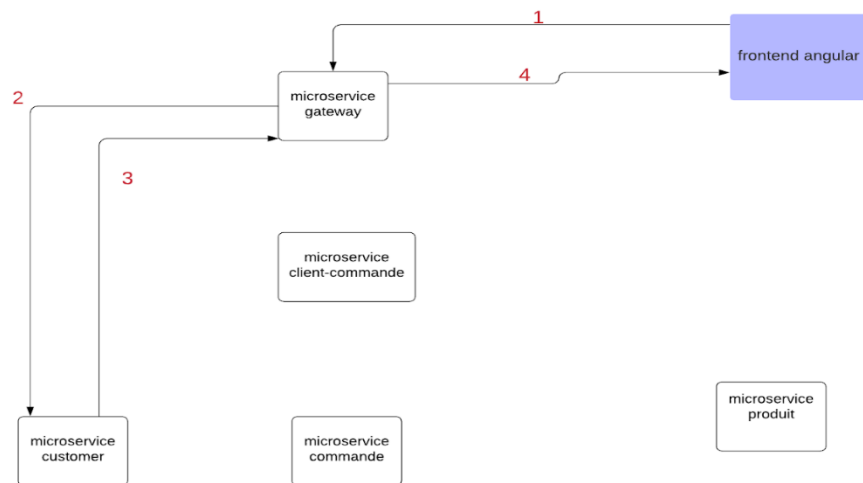


Figure 6: affichage des customers

4. *Suppression ou modification d'un produit*

La figure 6 illustre les appels https permettant de supprimer un produit, le client frontend envoie une requête delete qui sera redirigé par la gateway vers le microservice produit. Ce dernier envoie un appel http get vers le service commande pour récupérer la liste des commandes relatives au produit qui seront supprimés avec le produit.

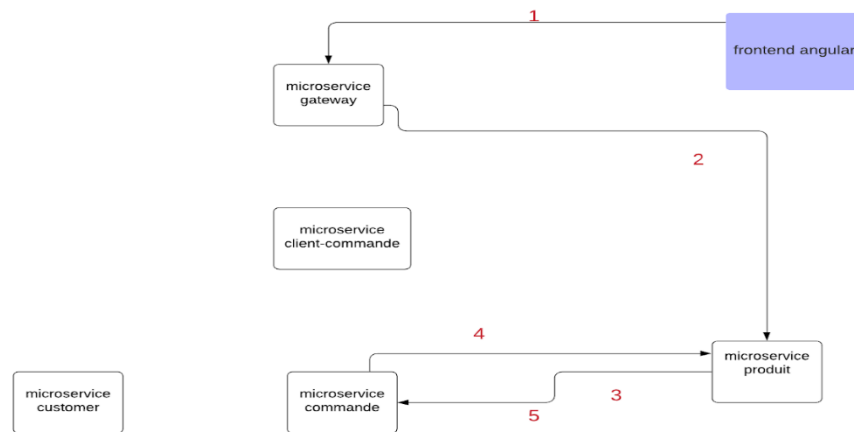


Figure 7: modification ou suppression d'un produit

5. *Suppression ou modification d'un customer*

La figure 7 illustre les appels https permettant de supprimer (modifier) un customer, le client frontend envoie une requête delete (update) qui sera redirigé par la gateway vers le microservice customer. Ce dernier envoie un appel http get vers le service commande pour récupérer la liste des commandes relatives au customer qui seront supprimés (modifiés) avec le customer.

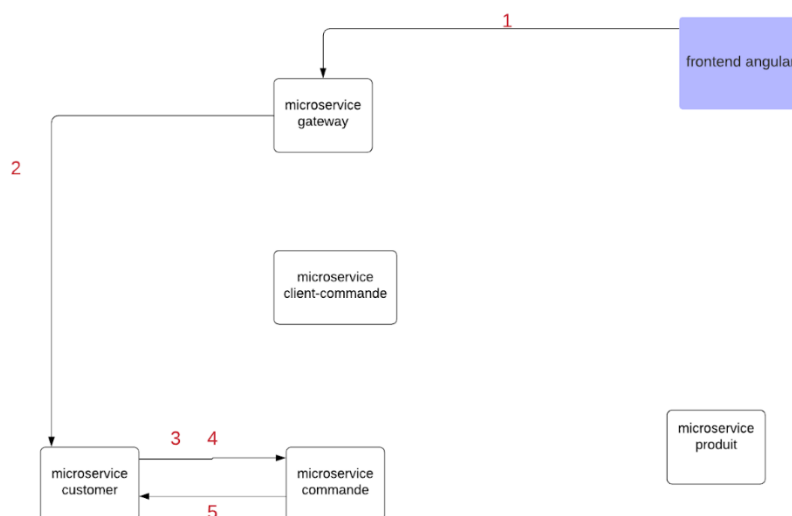


Figure 8: modification ou suppression d'un customer

Vous trouverez quelques captures d'appel postman ici.

- *Authentication* : l'authentification retourne un token qu'on doit rajouter à chaque requête afin de pouvoir accéder aux différents services

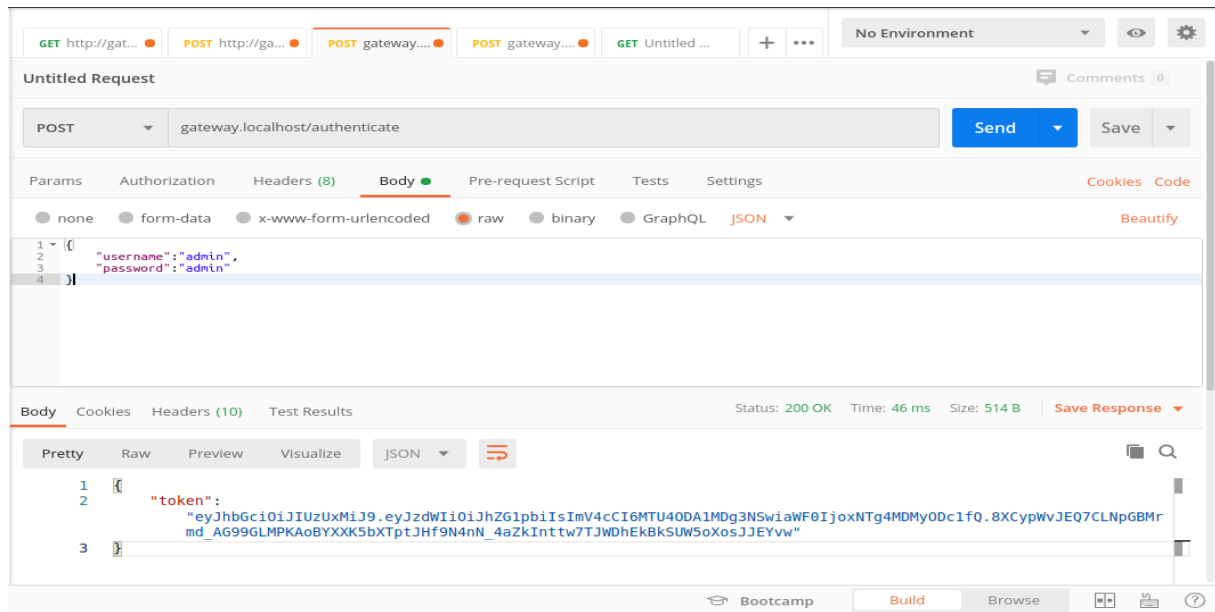


Figure 9: Authentication

- *Retrouver la liste des produits* :

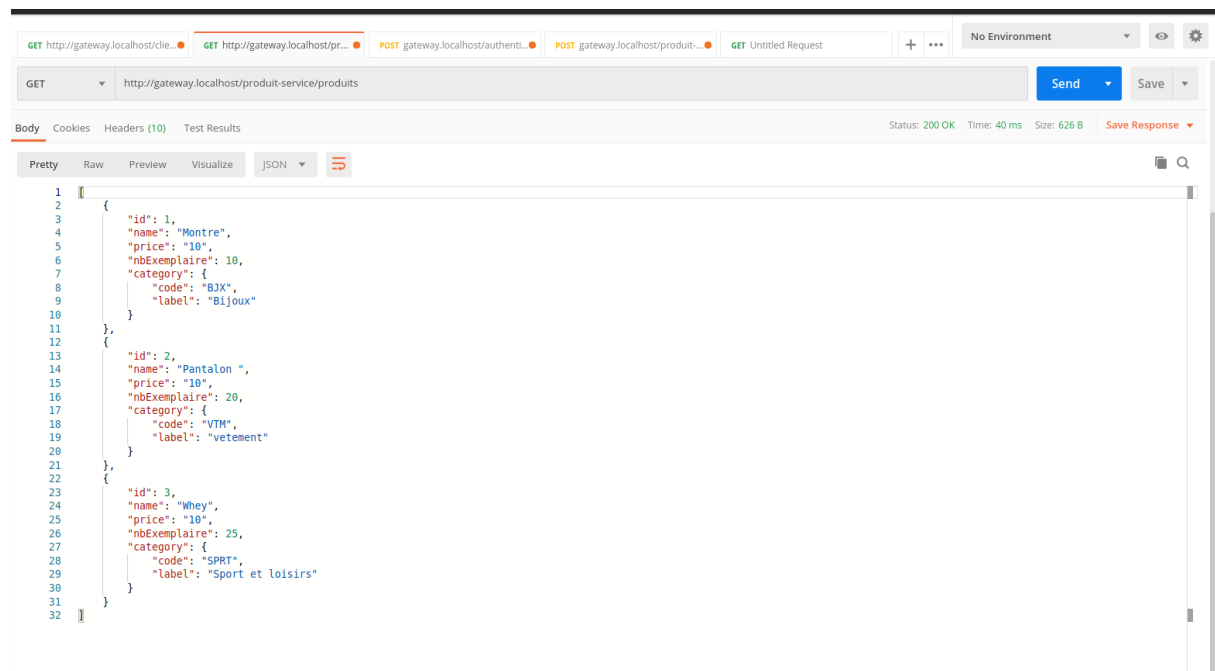


Figure 10: Get tous les produits

- *Ajouter un nouveau produit*

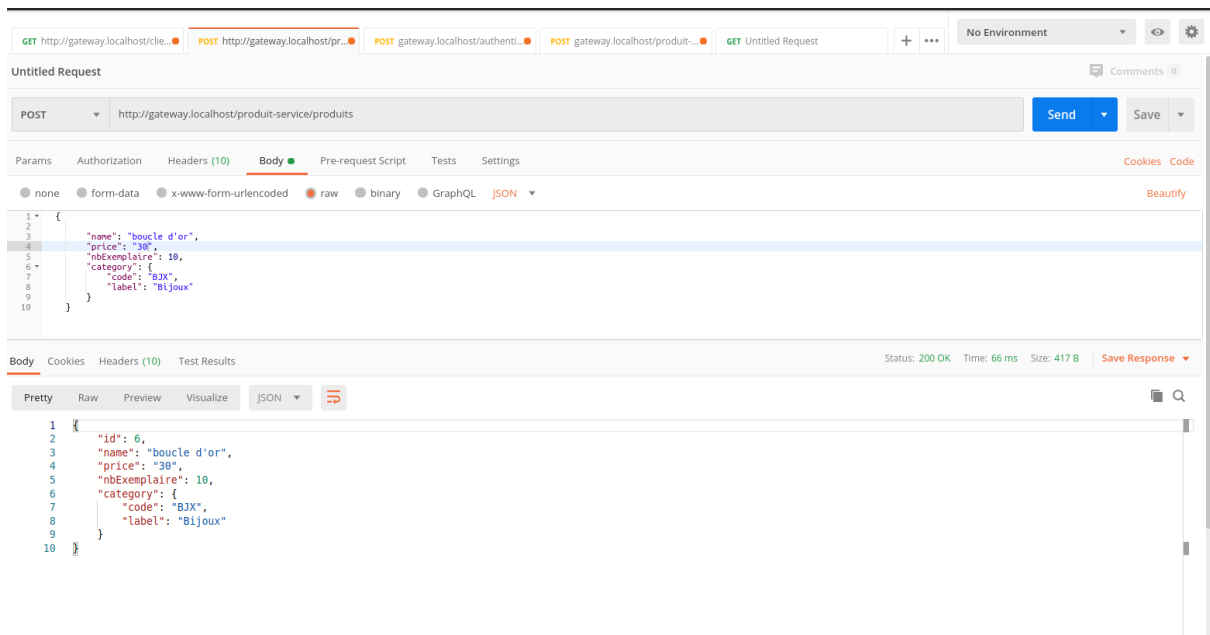


Figure 11: Ajouter un produit

- *Modifier un produit*

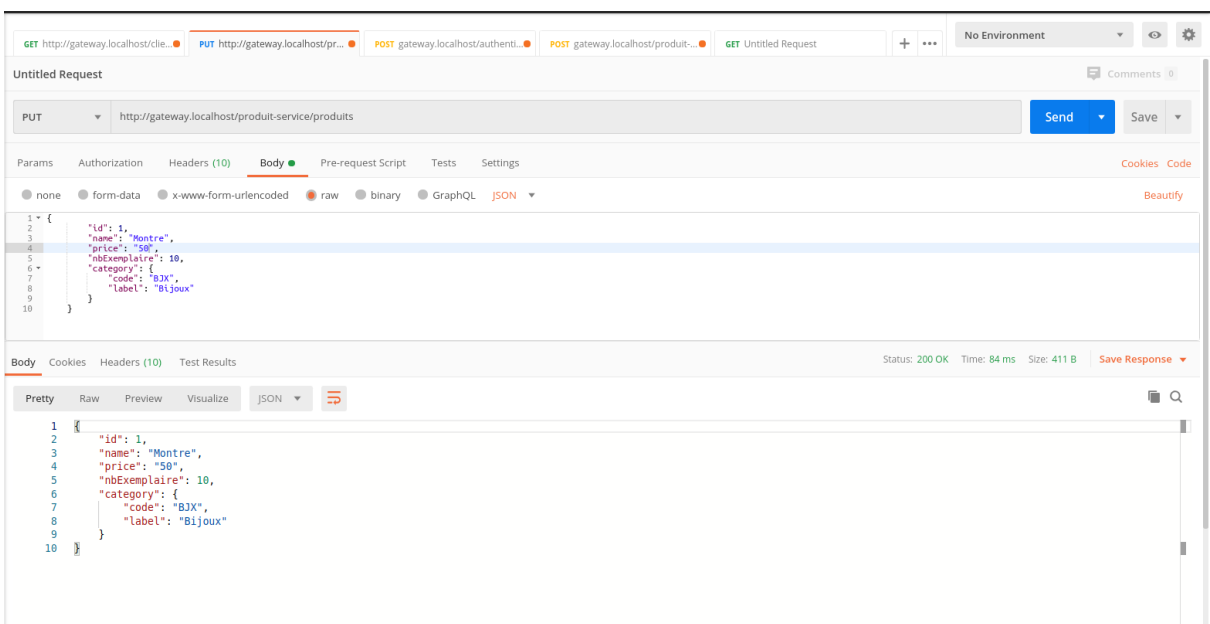


Figure 12: Modifier un produit

- *Supprimer un produit*

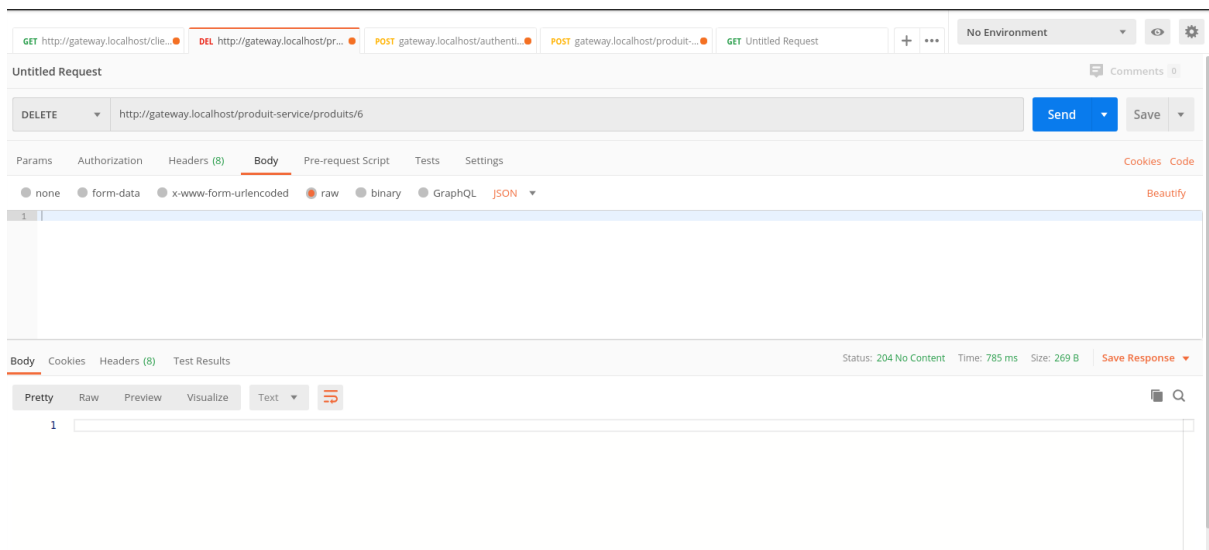


Figure 13: Supprimer un produit

Interface « Front-end »

Nous avons utilisé angular 9 pour la réalisation de notre service front end, en effet, nous avons décomposé notre service en plusieurs composants comme suit :

- Un composant menu (side-bar) : c'est le composant qui permettra de router à travers notre application (accéder aux différentes rubriques).
- Un composant login : ce composant se charge de l'authentification, Nous avons utilisé un garde nommé LoginGuard pour bloquer l'accès aux composants via url si l'on n'est pas authentifié.

Pour la gestion des commandes, customers, produits et administrateurs nous avons suivis la même décomposition :

- Un composant customer : ce composant se charge de l'affichage de notre liste de customers sous forme d'un tableau, il gère aussi la suppression.
- Un composant add-customer : comme son nom l'indique, il se charge d'afficher un formulaire pour l'ajout d'un nouveau customer.
- Un composant update-customer : permet d'afficher un formulaire pour la modification d'un customer.

L'interaction de ses composant avec notre backend se fait grâce aux services implémentés. Ces derniers vont communiquer à travers des requêtes https.

Nous avons implémenté un service pour chaque service backend (service commande, service produit, service user, service customer) ainsi qu'un service pour gérer l'authentification.

Vous trouverez ici le lien de présentation de notre application :

<https://vimeo.com/411263454>

Ainsi que quelques captures d'écran sur l'application

- *Authentication*

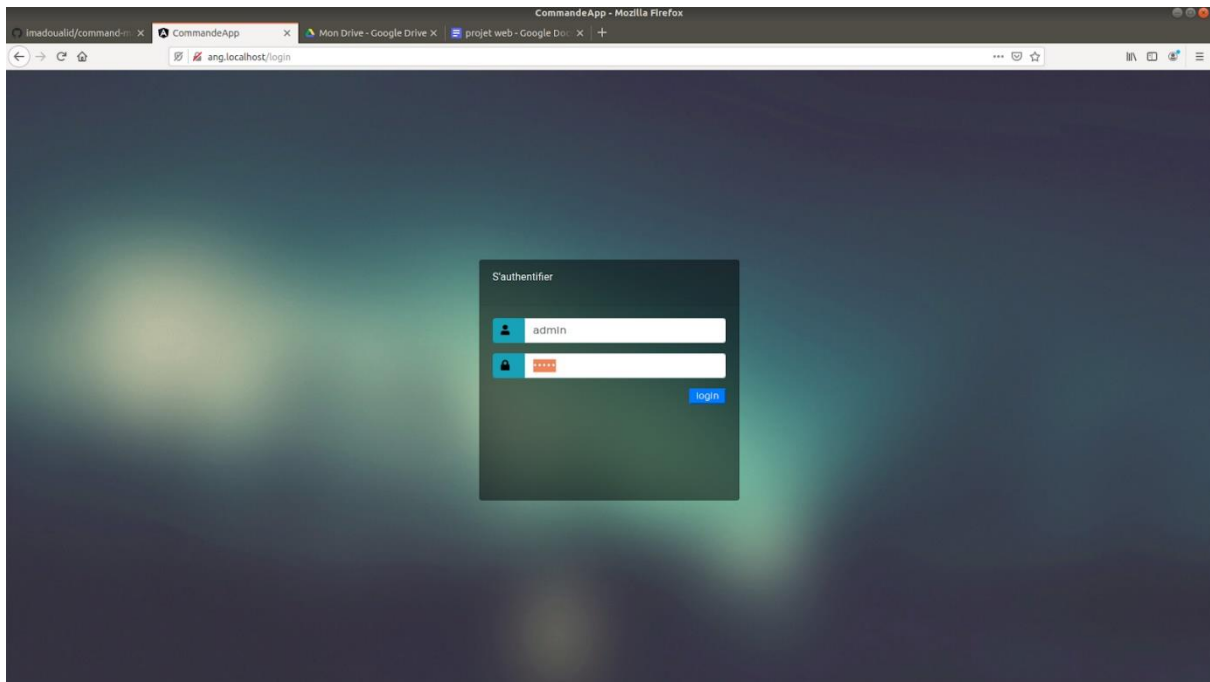
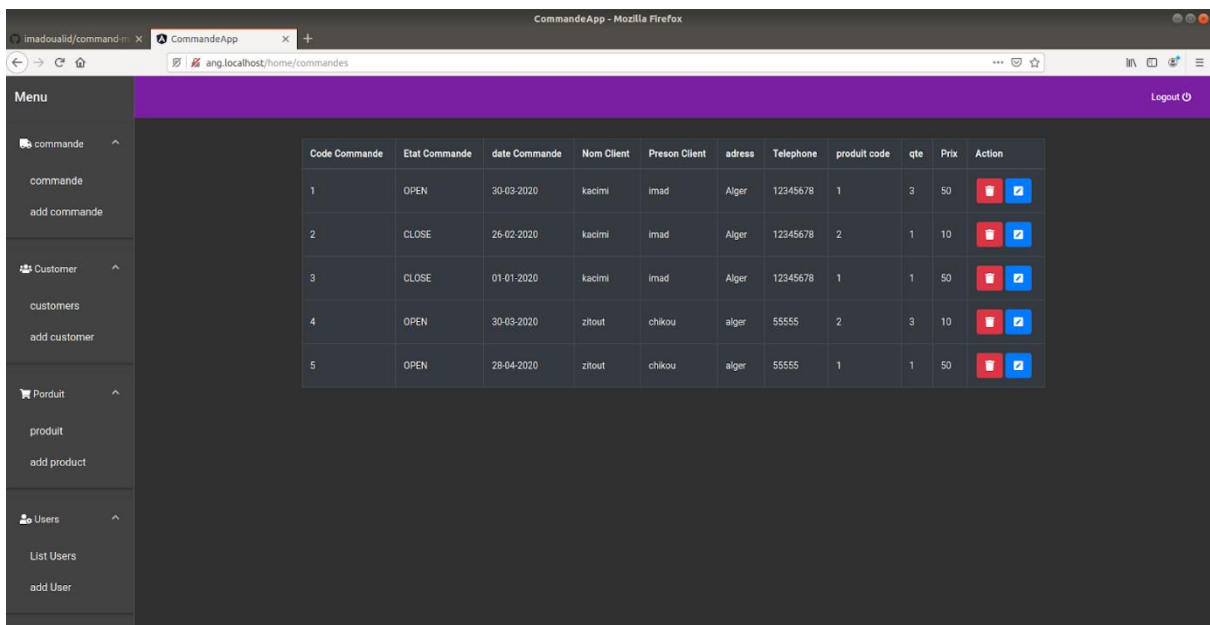
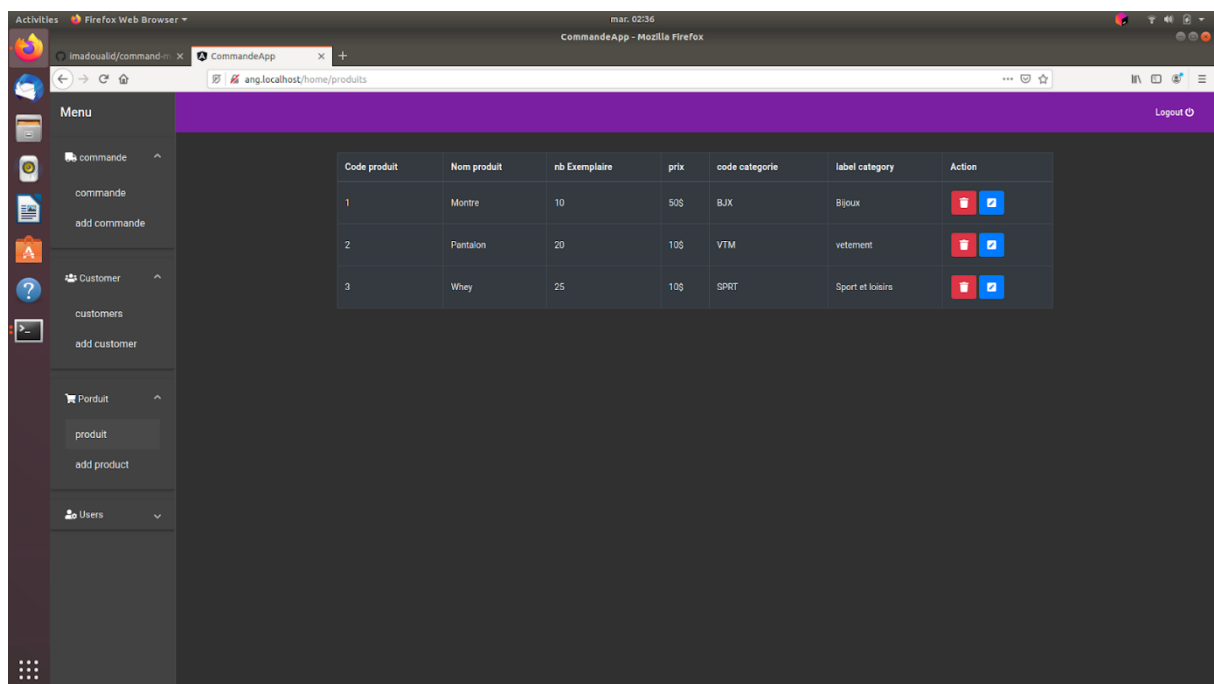
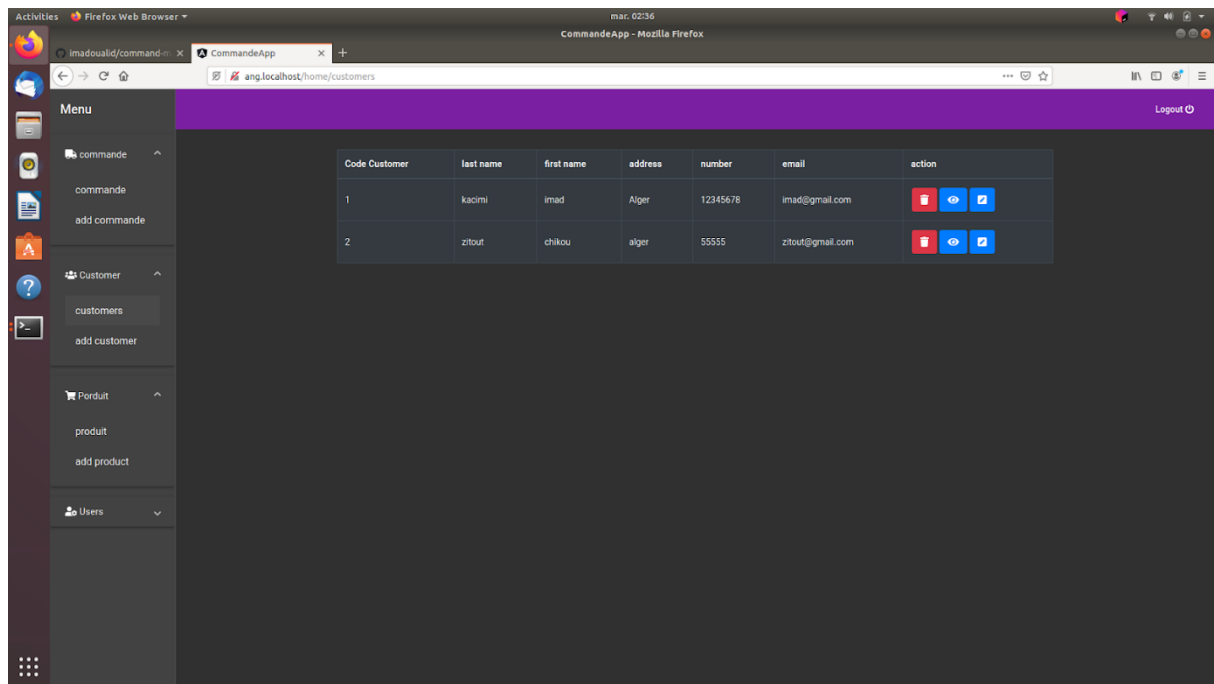


Figure 14: AUTHENTICATION

- *Affichage des commandes, customers, produits et administrateurs*





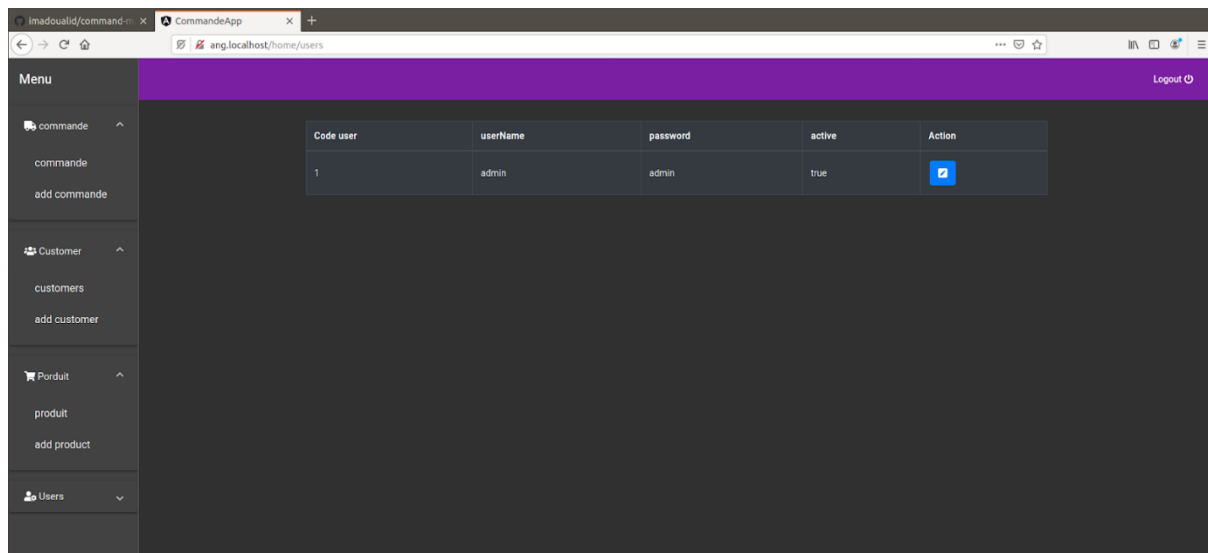


Figure 15 : tableau des commandes, produits, customers et administrateurs

- Ajout d'une commande pour un customer existant « ici nous avons besoin de renseigner que le numéro de téléphone de ce dernier »

The screenshot shows the 'add commande' form in the CommandeApp web application. The form contains the following fields:

- First name
- Last name
- adress
- phone number
- mail
- code product
- statut commande
- quantité
- date de commande

The 'phone number' field is highlighted with a red border, indicating it is a required field. The 'statut commande' field is set to 'OPEN'. The 'date de commande' field is set to '28-04-2020'. A blue 'Envoyer' button is located at the bottom of the form.

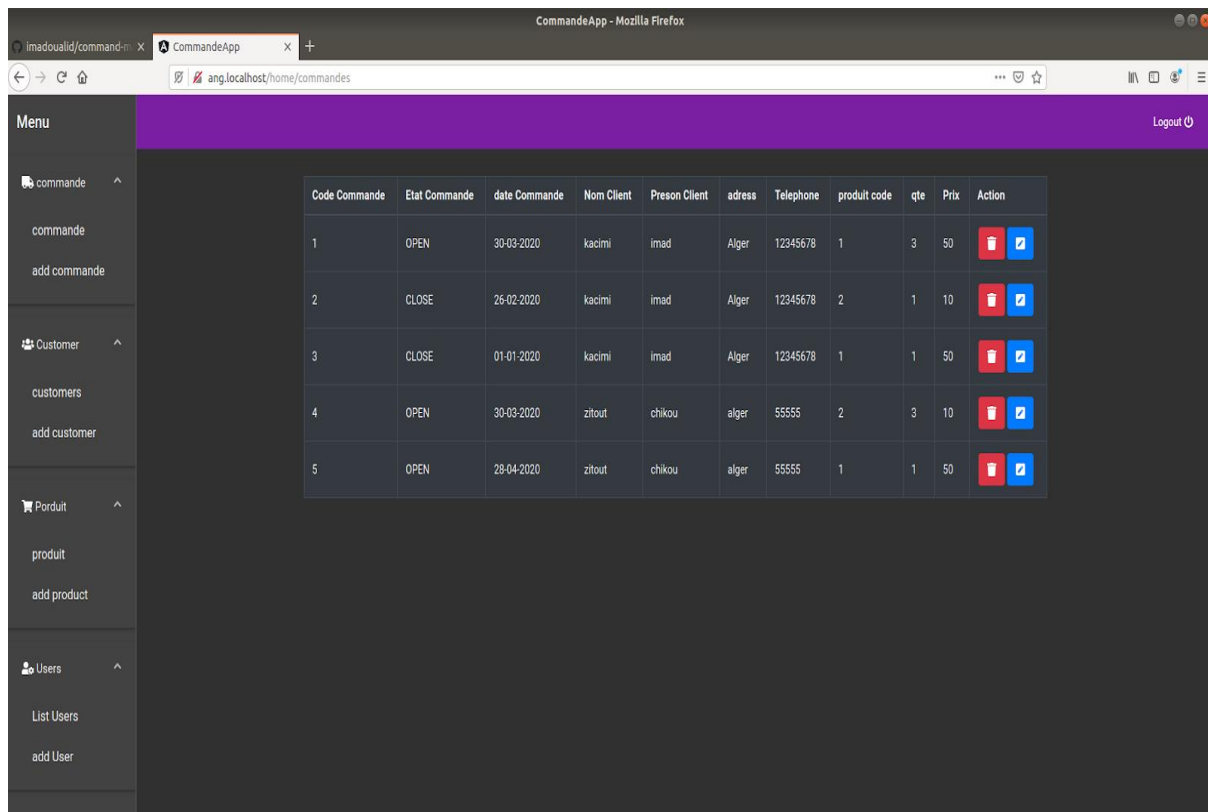


Figure 16: ajout d'une commande pour un customer existant

- Ajout d'une commande pour un customer qui n'existe pas

The screenshot shows the CommandeApp web application in a Mozilla Firefox browser. The address bar displays 'ang.localhost/home/addCommande'. The left sidebar contains a menu with categories: commande, Customer, Porduit, and Users. The main content area displays a form to add a new order.

Form fields:

- First name: kacimi
- Last name: amine
- adress: alger
- phone number: 5454
- mail: amine@gmail.com
- code product: 2
- statut commande: OPEN
- quantité: 2
- date de commande: 28-04-2020

Buttons: Envoyer

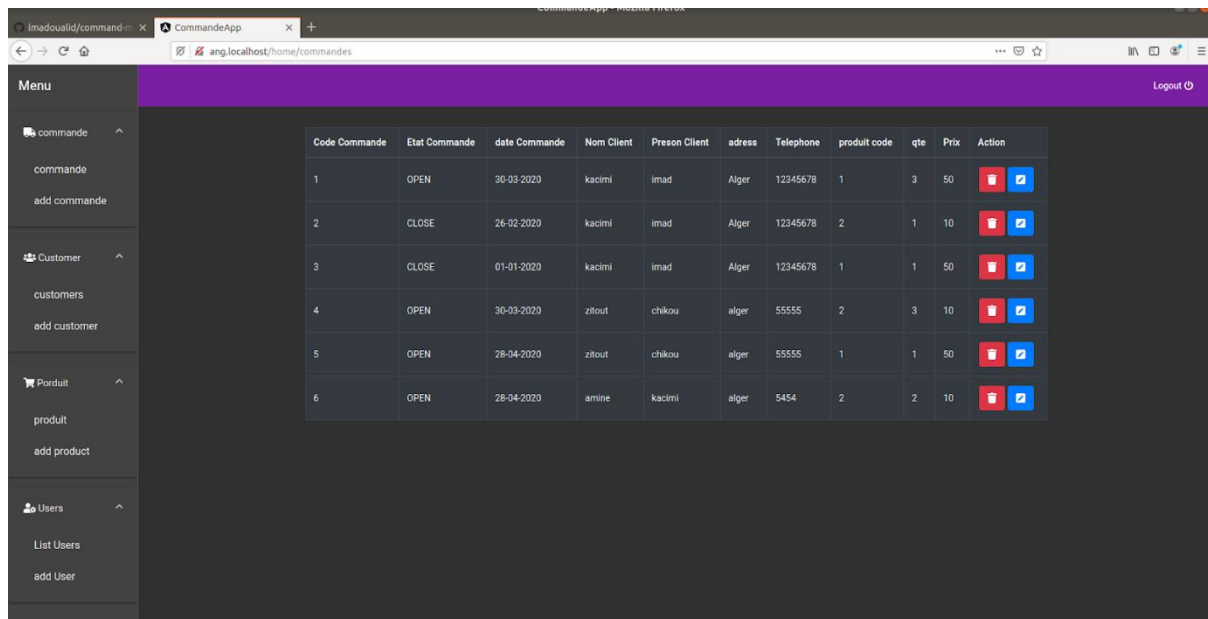


Figure 17: ajout d'une commande pour un nouveau customer

Déploiement docker

Nous avons déployé nos différents micro services dans docker-hub comme le montre la figure 8.

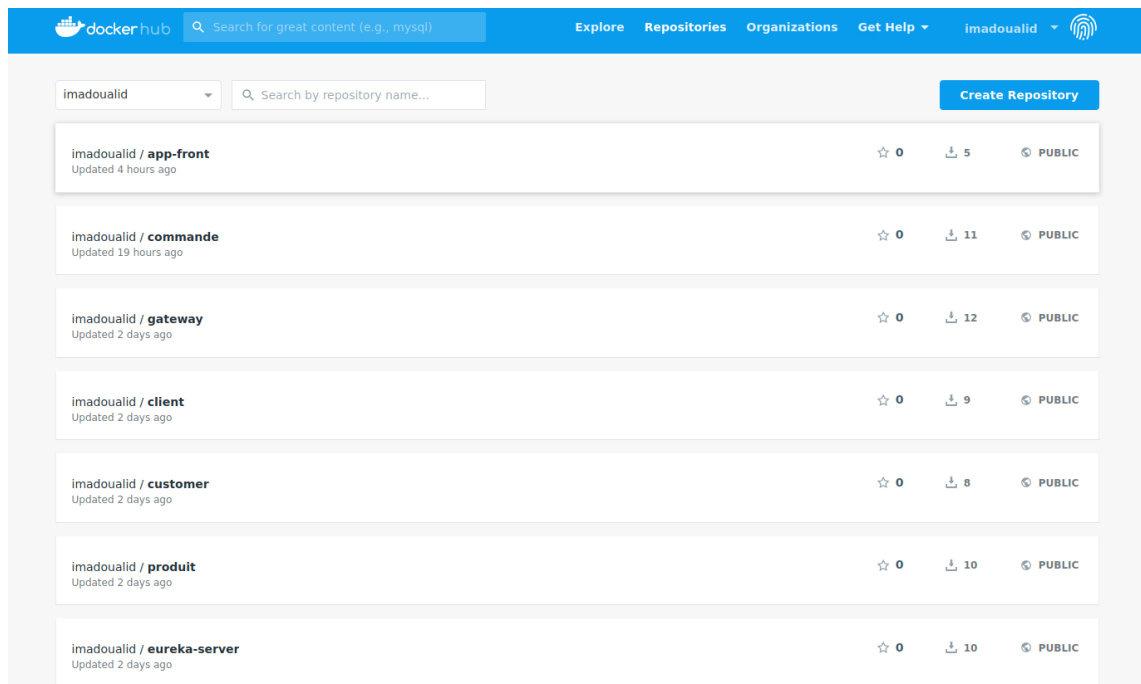
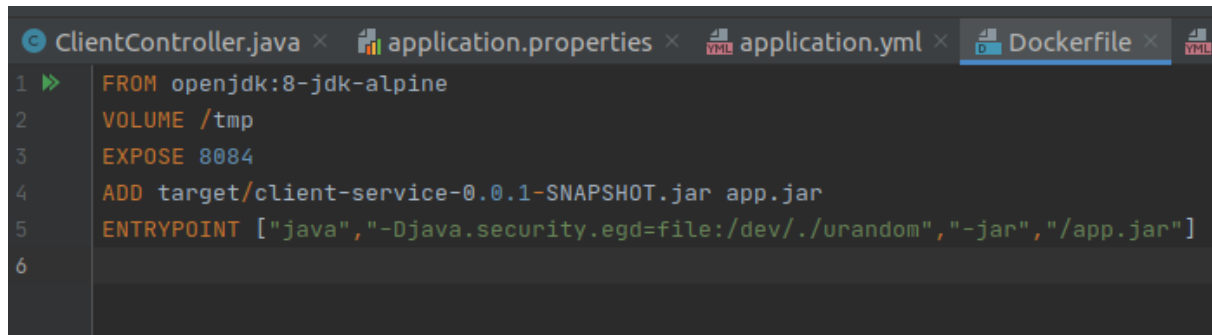


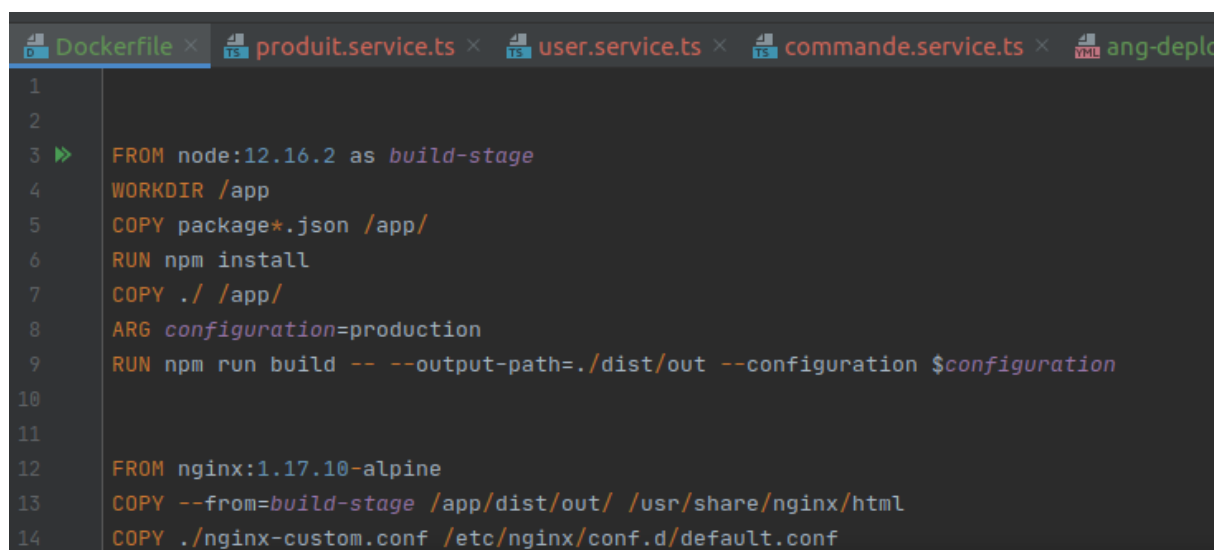
Figure 18 Docker hub

Vous trouverez ici un exemple de fichier Dockerfile pour l'un des micro services spring boot, ainsi que celui du service front.



```
ClientController.java × application.properties × application.yml × Dockerfile ×
1 FROM openjdk:8-jdk-alpine
2 VOLUME /tmp
3 EXPOSE 8084
4 ADD target/client-service-0.0.1-SNAPSHOT.jar app.jar
5 ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
6
```

Figure 19: Dockerfile du microservice client



```
Dockerfile × produit.service.ts × user.service.ts × commande.service.ts × ang-deplo
1
2
3 FROM node:12.16.2 as build-stage
4 WORKDIR /app
5 COPY package*.json /app/
6 RUN npm install
7 COPY ./ /app/
8 ARG configuration=production
9 RUN npm run build -- --output-path=./dist/out --configuration $configuration
10
11
12 FROM nginx:1.17.10-alpine
13 COPY --from=build-stage /app/dist/out/ /usr/share/nginx/html
14 COPY ./nginx-custom.conf /etc/nginx/conf.d/default.conf
```

Figure 20: Dockerfile du front-end.

Ici un exemple de déploiement du service client et front dans docker :

```

root@analyst:~/Desktop/distribue/projet/backend/client-service# docker build -t imadoualid/client:1 .
Sending build context to Docker daemon 64.98MB
Step 1/5 : FROM openjdk:8-jdk-alpine
--> a3562aa0b991
Step 2/5 : VOLUME /tmp
--> Using cache
--> d639df1854d9
Step 3/5 : EXPOSE 8084
--> Using cache
--> b88cb590c850
Step 4/5 : ADD target/client-service-0.0.1-SNAPSHOT.jar app.jar
--> Using cache
--> 176a4f112674
Step 5/5 : ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
--> Using cache
--> f9270afeaeb0
Successfully built f9270afeaeb0
Successfully tagged imadoualid/client:1
root@analyst:~/Desktop/distribue/projet/backend/client-service# docker push imadoualid/client:1
The push refers to repository [docker.io/imadoualid/client]
5cb2674d16e4: Layer already exists
ceaf9e1ebef5: Layer already exists
9b9b7f3d56a0: Layer already exists
f1b5933fe4b5: Layer already exists
1: digest: sha256:5c3c0e3baebbf1f0354bb04a8cf1bcd2f585fc08ad78a869fbde9287607dd94 size: 1159

```

Figure 21: déploiement du service client dans docker.

Déploiement kubernetes:

Nous avons déployé nos différents conteneurs docker dans kubernetes, nous avons utilisé k3s, qui permettra de créer un cluster de pods, chaque pod contiendra un conteneur docker.

La configuration des pods se fera dans les fichiers déploiements, chaque pod aura un service associé afin de lui attribuer une adresse ip fixe qui permettra aux autres pods de communiquer avec lui. Nous aurons aussi un fichier de type ingress pour l'attribution d'une adresse qui permettra l'accès depuis l'extérieur du cluster.

```

root@analyst:~/Desktop/distribue/projet/backend/customers-service# kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/gateway-deployment-d969df99d-jmccd    1/1      Running    0           59m
pod/produit-deployment-c466754c9-k84hd    1/1      Running    0           55m
pod/commande-deployment-6c4475866b-wtvrh    1/1      Running    0           48m
pod/client-deployment-6c747f77bf-6t7j6    1/1      Running    0           46m
pod/eureka-deployment-5657fd79c7-rttnc    1/1      Running    0           60m
pod/customer-deployment-6bb4b9fc4d-fk5tj    1/1      Running    0           52m
pod/ang-deployment-865849499c-bd7hd        1/1      Running    0           26s

NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
service/kubernetes                  ClusterIP           10.43.0.1        <none>            443/TCP          22h
service/eureka-service              ClusterIP           10.43.192.84     <none>            80/TCP           60m
service/gateway-service             ClusterIP           10.43.214.150    <none>            80/TCP           59m
service/produit-service             ClusterIP           10.43.197.233    <none>            80/TCP           55m
service/customer-service            ClusterIP           10.43.171.116    <none>            80/TCP           51m
service/commande-service            ClusterIP           10.43.19.118     <none>            80/TCP           48m
service/client-service              ClusterIP           10.43.255.226    <none>            80/TCP           46m
service/ang-service                 ClusterIP           10.43.39.236     <none>            80/TCP           21s

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/gateway-deployment    1/1      1              1            59m
deployment.apps/produit-deployment    1/1      1              1            55m
deployment.apps/client-deployment      1/1      1              1            46m
deployment.apps/commande-deployment    1/1      1              1            48m
deployment.apps/eureka-deployment      1/1      1              1            60m
deployment.apps/customer-deployment    1/1      1              1            52m
deployment.apps/ang-deployment         1/1      1              1            26s

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/gateway-deployment-d969df99d    1          1          1        59m
replicaset.apps/produit-deployment-c466754c9    1          1          1        55m
replicaset.apps/commande-deployment-6c4475866b    1          1          1        48m
replicaset.apps/client-deployment-6c747f77bf    1          1          1        46m
replicaset.apps/eureka-deployment-5657fd79c7    1          1          1        60m
replicaset.apps/customer-deployment-6bb4b9fc4d    1          1          1        52m
replicaset.apps/ang-deployment-865849499c        1          1          1        26s
root@analyst:~/Desktop/distribue/projet/backend/customers-service# kubectl get ing
NAME                                HOSTS                ADDRESS          PORTS          AGE
eureka-ingress                     eureka.localhost    192.168.0.29    80            60m
gateway-ingress                     gateway.localhost    192.168.0.29    80            59m
produit-ingress                     produit.localhost    192.168.0.29    80            55m
customer-ingress                    customer.localhost   192.168.0.29    80            50m
commande-ingress                    commande.localhost   192.168.0.29    80            48m
client-ingress                      client.localhost     192.168.0.29    80            46m
ang-ingress                         ang.localhost        192.168.0.29    80            22s

```

Figure 22 :déploiements des différents conteneurs.

Ici un exemple d'exécution des commandes de déploiement du service client dans kubernetes :

```

root@analyst:~/Desktop/distribue/projet/backend/client-service# kubectl apply -f client-deployment.yml
deployment.apps/client-deployment created
root@analyst:~/Desktop/distribue/projet/backend/client-service# kubectl apply -f client-service.yml
service/client-service created
root@analyst:~/Desktop/distribue/projet/backend/client-service# kubectl apply -f client-ingress.yml
ingress.extensions/client-ingress created
root@analyst:~/Desktop/distribue/projet/backend/client-service#

```

Figure 23: déploiement du client dans kubernetes.