
TP N°7 :

**Communication entre les microservices
avec
Spring Cloud : Feign/OpenFeign**

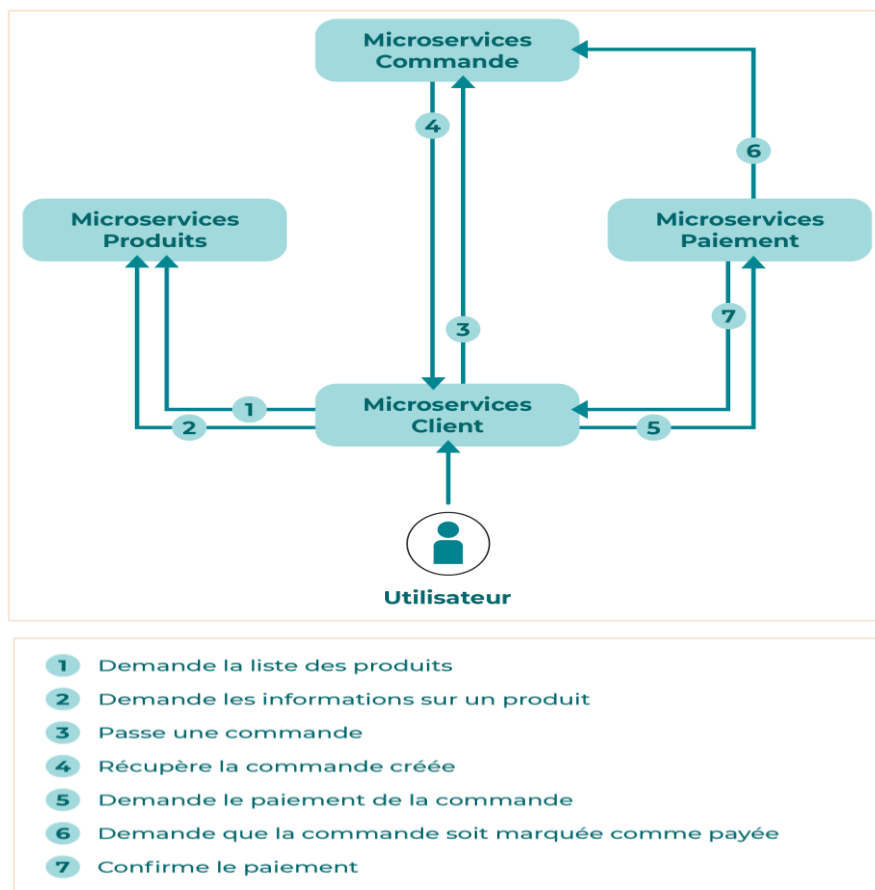
1. Prérequis

- JDK.17
- Connexion internet

2. Objectifs

1. Mise en place d'une application distribuée basée sur 4 microservices différents
2. Communiquer les microservices grâce à OpenFeign : <https://spring.io/projects/spring-cloud-openfeign>
3. Utiliser l'annotation `@EnableFeignClients`
4. Utiliser l'annotation `@FeignClient`
5. Orchestration des microservices

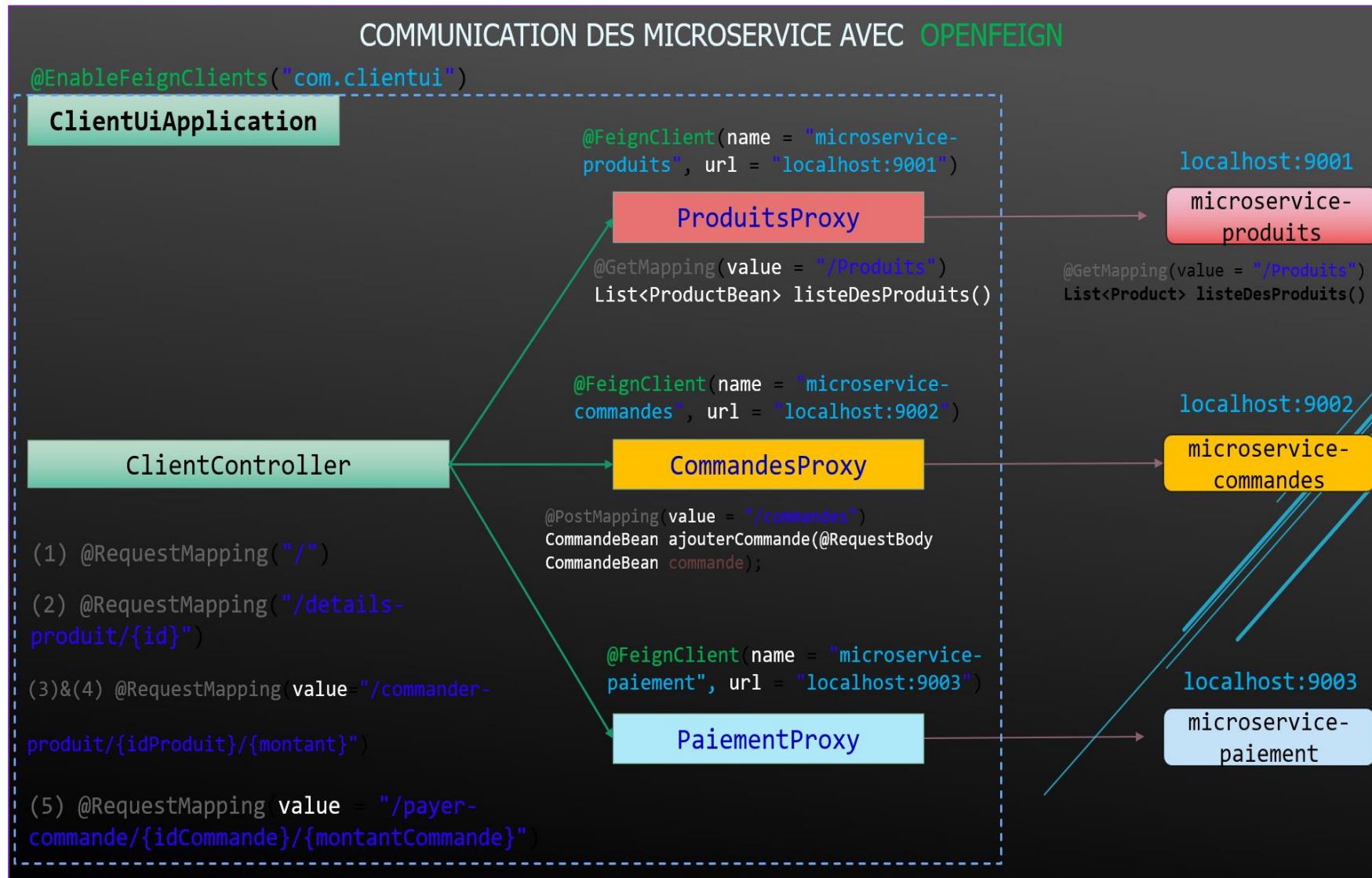
3. Schéma des échanges entre les MS



L'application MCommerce est basée sur 4 microservices : client, microservice-produits, Microservice-commandes, microservice-paiement.

Le microservice « Client » va jouer le rôle de point d'entrée de notre application, et va Orchestrer les appels aux différents microservice sous-jacents.

4. Architecture de mise en œuvre



5. Etapes de développment de l'application MCommerce

Etape 0 : pom.xml du MS-Client :

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.6.1</version>
    <relativePath/> <!-- lookup parent from repository -->
  </parent>
  <groupId>com.clientui</groupId>
  <artifactId>clientui</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>clientui</name>
  <description>Demo project for Spring Boot</description>
  <properties>
    <java.version>17</java.version>
    <spring-cloud.version>2021.0.8</spring-cloud.version>
    <!--      <spring-cloud.version>2022.0.4</spring-cloud.version> -->
    <!--      <spring-cloud.version>Finchley.M8</spring-cloud.version> -->
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
      <groupId>org.webjars</groupId>
      <artifactId>bootstrap</artifactId>
      <version>4.0.0-2</version>
    </dependency>

    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-starter-openfeign</artifactId>
    </dependency>

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>

<repositories>
  <repository>
    <id>spring-milestones</id>
    <name>Spring Milestones</name>
    <url>https://repo.spring.io/milestone</url>
    <snapshots>
      <enabled>false</enabled>
    </snapshots>
  </repository>
</repositories>
</project>

```

Étape 1 : Class principale annotée par `@EnableFeignClients`

```

package com.clientui;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients("com.clientui")
public class ClientUiApplication {

    public static void main(String[] args) {
        SpringApplication.run(ClientUiApplication.class, args);
    }
}

```

Étape 2 : Récupérez la liste des produits

1. Quand **Feign** fait appel à **Microservice-produits** afin de récupérer la liste des produits, il lui faudra stocker chaque produit dans un objet de type « **Product** » afin de les manipuler facilement plus tard.
Nous allons créer un bean « **ProductBean** » qui reprend les mêmes champs que **Product.java**.
2. Créer une interface qui va regrouper les requêtes qu'on souhaite passer au **Microservice-produits**.

Cette interface est souvent appelée un proxy, car elle se positionne comme une classe intermédiaire qui fait le lien avec les microservices extérieurs à appeler.

Créez une classe **MicroserviceProduitsProxy** sous un package "**proxies**"

```
package com.clientui.proxies;

import com.clientui.beans.ProductBean;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import java.util.List;

@FeignClient(name = "microservice-produits", url = "localhost:9001")
public interface MicroserviceProduitsProxy {

    // ProductController.java : Récupérer les memes signatures des méthodes
    // @GetMapping(value = "/Produits")
    // public List<Product> listeDesProduits()

    @GetMapping(value = "/Produits")
    List<ProductBean> listeDesProduits();

    // Notez ici la notation @PathVariable("id") qui est différente de celle
    // qu'on utilise dans le contrôleur
    // ProductController.java : Récupérer les memes signatures des méthodes
    // @GetMapping( value = "/Produits/{id}")
    // public Optional<Product> recupererUnProduit(@PathVariable int id)

    @GetMapping( value = "/Produits/{id}")
    ProductBean recupererUnProduit(@PathVariable("id") int id);
}
```

Explications :

@FeignClient déclare cette interface comme client Feign. Feign utilisera les informations fournies pour construire les requêtes HTTP appropriées afin d'appeler le **Microservice-Produits**.

On donne à cette annotation 2 paramètres : le premier est "[name](#)", il s'agit du nom du microservice à appeler. Il ne s'agit pas ici de n'importe quel nom, mais du nom "officiel" qui sera utilisé plus tard par des Edge Microservices comme Eureka, Spring Cloud Config, Zuul GateWay ... Celui-ci est à renseigner dans « application.properties » du microservice à Appeler :

```
spring.application.name=microservice-  
produits  
server.port 9001  
#Configurations H2  
spring.jpa.show-sql=true  
spring.h2.console.enabled=true  
#defini l'encodage pour data.sql  
spring.datasource.sql-script-encoding=UTF-8
```

Le deuxième paramètre est l'[URL](#) du microservice ([localhost:9001](#)).

Dans cette interface, il faut déclarer les signatures des opérations à appeler dans le microservice "produits". Dans notre cas, comme nous avons accès au code source de microservice-produits, il suffit de copier ses signatures.

Néanmoins, même sans avoir accès aux sources, il suffit de préciser les types de retour, par exemple List , un nom quelconque pour votre méthode et un URI. Toutes ces informations sont normalement disponibles dans la documentation qui accompagne chaque microservice. Il faut veiller à remplacer « [Product](#) » par son équivalent dans le client « [ProductBean](#) ».

[Optional](#) a été changée par [ProductBean](#) dans la deuxième méthode, afin de simplifier son utilisation.

[Feign](#) a désormais tout ce qu'il faut pour déduire qu'il faut une requête HTTP de type GET (grâce à GetMapping), à quelle URL l'envoyer, et une fois la réponse reçue, dans quel objet la stocker ([ProductBean](#)).

Il ne reste plus qu'à utiliser ce proxy. Revenez dans le contrôleur :

```
package com.clientui.controller;  
  
import com.clientui.beans.CommandeBean;  
import com.clientui.beans.PaiementBean;  
import com.clientui.beans.ProductBean;  
import com.clientui.proxies.MicroserviceCommandeProxy;  
import com.clientui.proxies.MicroservicePaiementProxy;
```

```

import com.clientui.proxies.MicroserviceProduitsProxy;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import java.util.*;
import java.util.concurrent.ThreadLocalRandom;

@Controller
public class ClientController {

    @Autowired
    private MicroserviceProduitsProxy ProduitsProxy;

    @Autowired
    private MicroserviceCommandeProxy CommandesProxy;

    @Autowired
    private MicroservicePaielementProxy paielementProxy;

    /**
     * Étape (1)
     * Opération qui récupère la liste des produits et on les affichent dans la page
d'accueil.
     * Les produits sont récupérés grâce à ProduitsProxy
     * On fini par retourner la page Accueil.html à laquelle on passe la liste d'objets
"produits" récupérés.
     */
    @RequestMapping("/")
    public String accueil(Model model){

        List<ProductBean> produits = ProduitsProxy.listeDesProduits();

        model.addAttribute("produits", produits);

        return "Accueil";
    }

    /**
     * Étape (2)
     * Opération qui récupère les détails d'un produit
     * On passe l'objet "produit" récupéré et qui contient les détails en question à
FicheProduit.html
     */
    @RequestMapping("/details-produit/{id}")
    public String ficheProduit(@PathVariable int id, Model model){

        ProductBean produit = ProduitsProxy.recupererUnProduit(id);

        model.addAttribute("produit", produit);

        return "FicheProduit";
    }

    /**
     * Étape (3) et (4)
     * Opération qui fait appel au microservice de commande pour placer une commande et
récupérer les détails de la commande créée
     */
    @RequestMapping(value = "/commander-produit/{idProduit}/{montant}")

```



```

    public String passerCommande(@PathVariable int idProduit, @PathVariable Double
montant, Model model){

        CommandeBean commande = new CommandeBean();

        //On renseigne les propriétés de l'objet de type CommandeBean que nous avons créé
        commande.setProductId(idProduit);
        commande.setQuantite(1);
        commande.setDateCommande(new Date());

        //appel du microservice commandes grâce à Feign et on récupère en retour les
détails de la commande créée, notamment son ID (étape 4).
        CommandeBean commandeAjoutée = CommandesProxy.ajouterCommande(commande);

        //on passe à la vue l'objet commande et le montant de celle-ci afin d'avoir les
informations nécessaire pour le paiement
        model.addAttribute("commande", commandeAjoutée);
        model.addAttribute("montant", montant);

        return "Paiement";
    }

    /*
    * Étape (5)
    * Opération qui fait appel au microservice de paiement pour traiter un paiement
    */
    @RequestMapping(value = "/payer-commande/{idCommande}/{montantCommande}")
    public String payerCommande(@PathVariable int idCommande, @PathVariable Double
montantCommande, Model model){

        PaiementBean paiementAExecuter = new PaiementBean();

        //on renseigne les détails du produit
        paiementAExecuter.setIdCommande(idCommande);
        paiementAExecuter.setMontant(montantCommande);
        paiementAExecuter.setNumeroCarte(numcarte()); // on génère un numéro au hasard
pour simuler une CB

        // On appelle le microservice et (étape 7) on récupère le résultat qui est sous
forme ResponseEntity<PaiementBean> ce qui va nous permettre de vérifier le code retour.
        ResponseEntity<PaiementBean> paiement =
paiementProxy.payerUneCommande(paiementAExecuter);

        Boolean paiementAccepte = false;
        //si le code est autre que 201 CREATED, c'est que le paiement n'a pas pu aboutir.
        if(paiement.getStatusCode() == HttpStatus.CREATED)
            paiementAccepte = true;

        model.addAttribute("paiementOk", paiementAccepte); // on envoie un Boolean
paiementOk à la vue

        return "confirmation";
    }

    //Génère une série de 16 chiffres au hasard pour simuler vaguement une CB
    private Long numcarte() {

        return ThreadLocalRandom.current().nextLong(1000000000000000L, 9000000000000000L );
    }
}

```

Explications :

Nous créons une variable de type `MicroserviceProduitsProxy` qui sera instanciée automatiquement par Spring. Nous avons accès à toutes les méthodes que nous avons définies dans `MicroserviceProduitsProxy` ; il suffit de faire appel à `listeDesProduits`.

`Feign` ira exécuter la requête HTTP, et nous renverra une liste de « `ProductBean` » .

Nous utilisons la méthode `addAttribute` de `model` afin de passer en revue la liste des Produits. La Vue du modèle MVC « `Accueil.html` » :

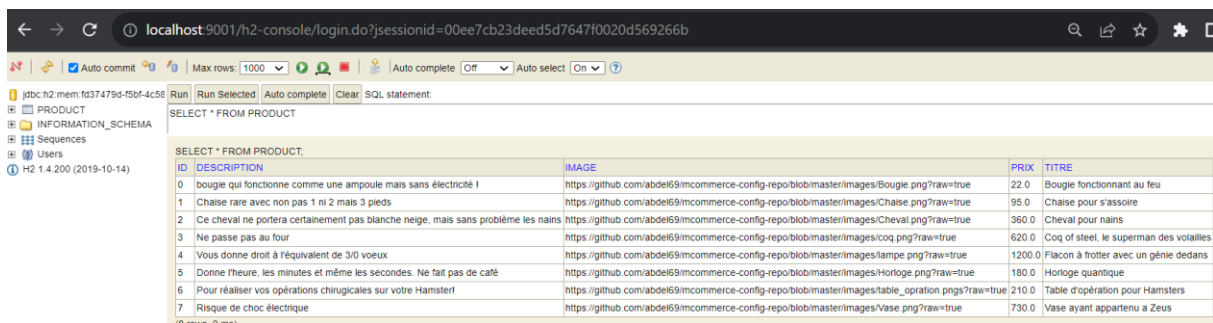
```
.....
<div class="row">
<!--Boucle sur tous les objets de type "produit" de la liste et affichage des
informations produit. -->

    <div th:each="produit : ${produits}" class="col-md-4 my-1">
        <a th:href="@{/details-produit/${produit.id}}/" >
            
            <p th:text= "${produit.titre}"></p>
        </a>
    </div>
</div>
```

Étape 3 : Adapter les développements aux autres microservices

Démarrer l'ensemble des 4 microservices : **clientui, microservice-produits, microservice commandes, microservice-paiement** puis vérifier le bon fonctionnement :

microservice-produits : <http://localhost:9001/h2-console/>



ID	DESCRIPTION	IMAGE	PRIX	TITRE
0	bougie qui fonctionne comme une ampoule mais sans électricité !	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/Bougie.png?raw=true	22.0	Bougie fonctionnant au feu
1	Chaise rare avec non pas 1 ni 2 mais 3 pieds	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/Chaise.png?raw=true	95.0	Chaise pour s'assoire
2	Ce cheval ne portera certainement pas blanche neige, mais sans problème les nains	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/Cheval.png?raw=true	360.0	Cheval pour nains
3	Ne passe pas au four	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/coq.png?raw=true	620.0	Coq of steel, le superman des volailles
4	Vous donne droit à l'équivalent de 3/0 voeux	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/lampe.png?raw=true	1200.0	Flacon à froter avec un génie dedans
5	Donne l'heure, les minutes et même les secondes. Ne fait pas de café	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/Horloge.png?raw=true	180.0	Horloge quantique
6	Pour réaliser vos opérations chirurgicales sur votre Hamster!	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/table_opration.png?raw=true	210.0	Table d'opération pour Hamsters
7	Risque de choc électrique	https://github.com/abdel69/mcommerce-config-repo/blob/master/images/Vase.png?raw=true	730.0	Vase ayant appartenu à Zeus

microservice-commandes : <http://localhost:9002/h2-console/>

localhost:9002/h2-console/login.do?sessionId=47dd0e06a253770e1edfa1662b2b3f49

Auto commit ☒ Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:a1c33637-1e66-46 COMMANDE

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM COMMANDE

SELECT * FROM COMMANDE;

ID	COMMANDE_PAYEE	DATE_COMMANDE	PRODUCT_ID	QUANTITE
(no rows, 2 ms)				

microservice-paiement : <http://localhost:9003/h2-console/>

localhost:9003/h2-console/login.do?sessionId=4dfc4a7c843f711eacdbfbdca4130228

Auto commit ☒ Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:3544ae5a-46d6-43 PAIEMENT

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PAIEMENT


SELECT * FROM PAIEMENT;

ID	ID_COMMANDE	MONTANT	NUMERO_CARTE
(no rows, 3 ms)			


Accéder au microservice « clientui » <http://localhost:8080/>

Application Mcommerce


Appuyez sur F11 pour quitter le mode plein écran.




Bougie fonctionnant au feu




Chaise pour s'asseoir



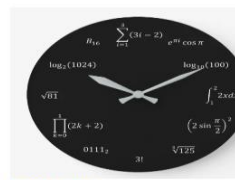
Cheval pour mains



Coq of steel, le superman des volailles



Flacon à frotter avec un génie dedans



Horloge quantique





Table d'opération pour Hamsters



Vase ayant appartenu à Zeus

localhost:8080/details-produit/7

Application Mcommerce



Vase ayant appartenu a Zeus

Risque de choc électrique

[COMMANDER](#)

localhost:8080/commander-produit/7/730.0

Application Mcommerce

Ici l'utilisateur sélectionne en temps normal un moyen de paiement et entre les informations de sa carte bancaire.

Nous allons éviter d'ajouter les formulaires nécessaireS afin de garder l'application la plus basique et simple possible pour la suite.

Si vous vous sentez à l'aise, vous pouvez créer un formulaire pour accepter le numéro de la CB, que vous traiterez dans le contrôleur grâce à un

PostMapping.

[Payer Ma Commande](#)

localhost:8080/payer-commande/1/730.0

Application Mcommerce

Paieiment Accepté

← → ↻ ⓘ localhost:9002/h2-console/login.do?jsessionId=fc15ead7196eefc258bd06e13662f5bc

🔧 | 🔒 | ☒ Auto commit | 🔄 | 📄 | Max rows: 1000 | 🟢 | 📄 | 📄 | Auto complete Off | Auto select On | ?

📄 jdbc:h2:mem:a1c33637-1e66-46- COMMANDE
📄 INFORMATION_SCHEMA
📄 Sequences
📄 Users
📄 H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM COMMANDE

SELECT * FROM COMMANDE;

ID	COMMANDE_PAYEE	DATE_COMMANDE	PRODUCT_ID	QUANTITE
1	TRUE	2023-12-04 14:37:50.211	7	1

(1 row, 2 ms)

← → ↻ ⓘ localhost:9003/h2-console/login.do?jsessionId=2e757e897397a0ca60f553f25a0991c2

🔧 | 🔒 | ☒ Auto commit | 🔄 | 📄 | Max rows: 1000 | 🟢 | 📄 | 📄 | Auto complete Off | Auto select On | ?

📄 jdbc:h2:mem:3544ae5a-46d6-43- PAIEMENT
📄 INFORMATION_SCHEMA
📄 Sequences
📄 Users
📄 H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PAIEMENT

SELECT * FROM PAIEMENT;

ID	ID_COMMANDE	MONTANT	NUMERO_CARTE
1	1	730.0	8286427521531801

(1 row, 2 ms)