TP2 : Développement de microservices autonomes et interconnectés avec Spring Boot et H2

Objectifs :

- Créer plusieurs microservices Spring Boot indépendants.
- Connecter chaque microservice à une base de données H2.
- Échanger des données **JSON** entre microservices via des **API REST**.
- Comprendre le concept de **Service Discovery** et l'importance de la communication entre services.

Outils :

- Java JDK (version 17 ou supérieure).
- IntelliJ IDEA.
- Maven.
- Spring Boot.

Travail demandé

L'application simulera un mini-système de **gestion de commandes** composé de deux microservices :

1. ***microservice-customer :***
   - Gère les clients (id, nom, email).
   - Expose une API REST /customers pour récupérer la liste des clients.
2. ***microservice-order :***
   - Gère les commandes (id, produit, quantité, clientId).
   - Fait appel au microservice-customer pour obtenir les informations du client lié à chaque commande.

Partie 1 : Création du microservice "Customer"

1. **Création du projet :**

Aller au site « start.spring.io » et créer un projet avec les caractéristiques suivantes : Projet Maven avec Java et Spring Boot

- **Group :** tn.tp.dev
- **Artifact :** microservice-customer
- **Dépendances: Spring Web, Spring Boot DevTools, Spring Data JPA, H2 Database**

Générer le projet en utilisant le packaging Jar.

Ouvrir l'archive téléchargé et l'extraire.

Ouvrir ItelliJ IDEA et importer le projet téléchargé à travers le fichier pom.xml

2. Les propriétés de l'application :

Dans le fichier application.properties :

- Définir le port à utiliser
- Configurer la base de données H2 en ajoutant les lignes suivantes :

```
spring.h2.console.enabled=true
spring.datasource.url=jdbc:h2:mem:customerdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
```

3. Implémentation du microservice :

- Créer une entité (utilisation de l'annotation @Entity) appelée Customer pour définir les clients : Chaque client est défini avec un id (de type Long), un nom (de type String), et un e-mail (de type String). Utiliser les annotations suivantes :

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

NB: Cette entité doit contenir deux constructeurs : un constructeur vide obligatoire pour Hibernate, et un constructeur utile pour initialiser les données.

- Créer un repository appelé CustomerRepository :

```
public interface CustomerRepository extends
JpaRepository<Customer, Long> { }
```

- Créer maintenant un contrôleur appelé CustomerController.

```
@RestController
@RequestMapping("/customers")
public class CustomerController {
```

```
    private final CustomerRepository repository;
    public CustomerController(CustomerRepository
repository) {
        this.repository = repository;
    }
    @GetMapping
    public List<Customer> getAllCustomers() {
        return repository.findAll();
    }
    @GetMapping("/{id}")
    public Customer getCustomerById(@PathVariable
Long id) {
        return repository.findById(id).orElse(null);
    }
}
```

- Insérer des données initiales dans CommandLineRunner dans l'application principale:

```
@Bean
    CommandLineRunner initDatabase(CustomerRepository
repo) {
        return args -> {
            repo.saveAll(List.of(
                new Customer(null, "Alice",
"alice@mail.com"),
                new Customer(null, "Bob", "bob@mail.com")
            ));
        };
    }
```

4. Test du microservice :

- Lancer l'application avec IntelliJ. (RQ : Pour le premier lancement, il faut ajouter une configuration maven en utilisant la commande suivante : `spring-boot:run`)
- Ouvrir le navigateur et accéder à l'application à travers le port définit : http://localhost: « ***port*** »/ « ***mappinValue*** »

Exemple : http://localhost:8082/customers pour afficher tous les clients, http://localhost:8082/customers/1 pour afficher un client spécifique à travers son id.

- Accéder à la base de données H2 à travers l'URL : `http://localhost: « port »/h2-console`

Partie 2 : Création du microservice "Order"

1. **Création du second projet :**

En utilisant Spring Initializr, créer un projet avec les caractéristiques suivantes : Projet Maven avec Java et Spring Boot

- **Group :** tn.tp.dev
- **Artifact :** microservice-order
- **Dépendances: Spring Web, Spring Boot DevTools, Spring Data JPA, H2 Database**

Ouvrir ItelliJ IDEA et importer le projet téléchargé.

2. **Les propriétés de l'application :**

Dans le fichier application.properties :

- Définir le port à utiliser
- De même que pour le microservice-customer, configurer la base de données H2 à travers l'URL suivant : `jdbc:h2:mem:order-db`

3. **Implémentation du microservice:**

- Créer une entité (utilisation de l'annotation @Entity) appelée Order pour définir les commandes : Chaque commande est définie avec un id (de type Long), un produit (de type String), une quantité (de type int) et un clientId (de type Long). Utiliser les annotations suivantes :

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
```

NB: Cette entité doit contenir deux constructeurs : un constructeur vide obligatoire pour Hibernate, et un constructeur utile pour initialiser les données.

NB : Changer le nom de la base de donnée à travers l'annotation suivante @Table(name = "order-db") //

- Créer un repository appelé OrderRepository contenant la liste suivante :

```
List<Order> findByCustomerId(Long customerId);
```

- Créer maintenant un contrôleur appelé OrderController (Utiliser RestTemplate dans OrderController pour consommer microservice-customer).

```
@RestController
@RequestMapping("/orders")
public class OrderController {
    @Autowired
    private OrderRepository orderRepository;

    @Autowired
    private RestTemplate restTemplate;

    @GetMapping
    public List<Order> getAllOrders() {
        return orderRepository.findAll();
    }

    @GetMapping("/with-customer/{id}")
    public Map<String, Object>
getOrderWithCustomer(@PathVariable Long id) {
     Order order =
orderRepository.findById(id).orElseThrow();
     Customer customer = restTemplate.getForObject(
"http://localhost:8082/customers/" +
order.getCustomerId(),Customer.class );
        return Map.of("order", order, "customer",
customer);
    }
}
```

- Insérer des données initiales dans CommandLineRunner dans l'application principale:

```
@Bean
CommandLineRunner initDatabase(OrderRepository repo) {
```

```
            return args -> {
                repo.saveAll(List.of(
                        new Order(null, "Laptop", 1, 1L),
                        new Order(null, "Smartphone", 2, 2L),
                        new Order(null, "Headphones", 1, 3L)
                ));
            };
        }
```

- Ajoutez une méthode annotée @Bean dans votre classe de configuration (Application) pour créer un objet RestTemplate.
- Ajouter une entité simple DTO Customer (pas d'annotation @Entity) identique à celle du microservice-customer.

4. Test et validation :

- Lancer l'application avec IntelliJ. (RQ : Pour le premier lancement, il faut ajouter une configuration maven en utilisant la commande suivante : `spring-boot:run`)
- Ouvrir le navigateur et accéder à l'application à travers le port définit :
  http://localhost: « ***port*** »/ « ***mappinValue*** »
  Exemple : http://localhost:8083/orders/with-customer/1 pour afficher la commande avec le client associé (via appel REST).
- Accéder à la base de données H2 à travers l'URL :
  http://localhost: « port »/h2-console

NB : Lancer d'abord le microservice-customer.