**Dans Controller**

Déclarer comme attributs le repository qu’on va utiliser (avec @autowired genre les depences seul)

Puis déclarer les méthodes de notre service web repository.méthodequ’on veut

**-quand dans une méthode on a un paramèter par exemple** updateEmploye(long id) :

Il faut préciser que c’est un parametre de chemin @PathVariable long int et avant la fonction uri + {id}

On aura dans le projet un GET ( car on veut récup réservation ) et un POST ( pour créer la reservation) et préciser qd c’est un POST @ResponseStatus(HttpStatus.CREATE)

**Étape 1 : Configuration du Projet**

1. **Créer un projet Spring Boot :**
   * Utilisez [Spring Initializer](https://start.spring.io/) pour créer un nouveau projet Spring Boot.
   * Ajoutez les dépendances nécessaires (Spring Web, Spring Data JPA, H2 Database, etc.).

**Étape 2 : Modèle de Données**

1. **Créer les classes de modèle :**
   * Créez une classe Hotel pour représenter les hôtels.
   * Ajoutez les annotations JPA (@Entity, @Id, @GeneratedValue) à la classe pour la mapper à une table.

java

1. import javax.persistence.Entity;
2. import javax.persistence.GeneratedValue;
3. import javax.persistence.GenerationType;
4. import javax.persistence.Id;
5. @Entity
6. public class Hotel {
7. @Id
8. @GeneratedValue(strategy = GenerationType.IDENTITY)
9. private Long id;
10. private String name;
11. private String city;
12. private int stars;
13. // Constructeurs, getters, setters
14. }

**Étape 3 : Repositories**

1. **Créer le repository :**
   * Créez une interface HotelRepository qui étend JpaRepository<Hotel, Long>.

java

1. import org.springframework.data.jpa.repository.JpaRepository;
2. public interface HotelRepository extends JpaRepository<Hotel, Long> {
3. // Ajoutez des méthodes spécifiques si nécessaire
4. }

**Étape 4 : Services Web**

1. **Créer le service web :**
   * Créez une classe HotelService qui contiendra la logique métier.

java

1. import org.springframework.beans.factory.annotation.Autowired;
2. import org.springframework.stereotype.Service;
3. @Service
4. public class HotelService {
5. @Autowired
6. private HotelRepository hotelRepository;
7. // Ajoutez des méthodes pour la logique métier
8. }

**Étape 5 : Contrôleur REST**

1. **Créer le contrôleur REST :**
   * Créez une classe HotelController pour exposer les fonctionnalités via des API REST.

java

1. import org.springframework.beans.factory.annotation.Autowired;
2. import org.springframework.web.bind.annotation.\*;
3. @RestController
4. @RequestMapping("/hotels")
5. public class HotelController {
6. @Autowired
7. private HotelService hotelService;
8. // Ajoutez des méthodes pour gérer les requêtes REST
9. }

**Étape 6 : Opérations CRUD**

1. **Ajouter les opérations CRUD :**
   * Dans HotelService, utilisez les méthodes de HotelRepository pour ajouter des opérations CRUD personnalisées.

**Étape 7 : Configuration d'Application**

1. **Configurer l'application.properties :**
   * Configurez les propriétés de la base de données dans application.properties.

**Étape 8 : Tests**

1. **Tester avec des clients RESTful :**
   * Utilisez des outils comme cURL, Postman pour tester vos services web REST.

### 2. ****Répositories (Repositories) :****

Les interfaces de répositories définissent les méthodes permettant d'interagir avec la base de données pour les entités correspondantes. Elles héritent souvent de frameworks comme Spring Data JPA, qui fournit des fonctionnalités de base pour l'accès aux données

**3. Services :**

* Les services encapsulent la logique métier de votre application. Ils agissent comme une couche intermédiaire entre les contrôleurs et les répositories. Les services peuvent effectuer des opérations complexes, coordonner plusieurs répositories et gérer la logique métier.

**4. Contrôleurs (Controllers) :**

* Les contrôleurs gèrent les requêtes HTTP et les réponses. Ils sont responsables de l'interaction avec les utilisateurs, appelant les méthodes appropriées des services et retournant les résultats sous forme de réponses HTTP