**Partie 3 : Ajout du microservice-department**

**1. Set Up the Department Microservice**

* **Allez à**[**Spring Initializr**](https://start.spring.io/):
  + Group: com.myHR
  + Artifact: department-api
  + Dependencies: Spring Web, Spring Data JPA, H2 Database, Lombok
* **Mettez à jour le fichier application.properties**:

|  |
| --- |
| properties  Copier le code  spring.application.name=department-api  server.port=9002  spring.h2.console.enabled=true  logging.level.com.myHR=INFO |

**2. Définir l’entité : Department Entity**

Créer un modèle pour l'entité Department similaire au modèle Employee :

|  |
| --- |
| package com.myHR.department\_api.model;  import javax.persistence.Entity;  import javax.persistence.GeneratedValue;  import javax.persistence.GenerationType;  import javax.persistence.Id;  import javax.persistence.Table;  import lombok.Data;  @Data  @Entity  @Table(name = "departments")  public class Department {  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  private Long id;  private String name;  private String location;  } |

**3. Couche Repository**

Définir une interface DepartmentRepository pour gérer les opérations de base de données :

|  |
| --- |
| package com.myHR.department\_api.repository;  import org.springframework.data.repository.CrudRepository;  import org.springframework.stereotype.Repository;  import com.myHR.department\_api.model.Department;  @Repository  public interface DepartmentRepository extends CrudRepository<Department, Long> {  } |

**4. Couche Service**

Implémenter le service pour gérer la logique métier liée aux départements :

|  |
| --- |
| package com.myHR.department\_api.service;  import com.myHR.department\_api.model.Department;  import com.myHR.department\_api.repository.DepartmentRepository;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import java.util.Optional;  @Service  public class DepartmentService {  @Autowired  private DepartmentRepository departmentRepository;  public Optional<Department> getDepartment(Long id) {  return departmentRepository.findById(id);  }  public Iterable<Department> getDepartments() {  return departmentRepository.findAll();  }  public Department saveDepartment(Department department) {  return departmentRepository.save(department);  }  public void deleteDepartment(Long id) {  departmentRepository.deleteById(id);  }  } |

**5. Couche Controller**

Définir les endpoints pour la gestion des départements :

|  |
| --- |
| package com.myHR.department\_api.controller;  import com.myHR.department\_api.model.Department;  import com.myHR.department\_api.service.DepartmentService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.web.bind.annotation.\*;  import java.util.Optional;  @RestController  @RequestMapping("/departments")  public class DepartmentController {  @Autowired  private DepartmentService departmentService;  @GetMapping  public Iterable<Department> getDepartments() {  return departmentService.getDepartments();  }  @GetMapping("/{id}")  public Optional<Department> getDepartment(@PathVariable Long id) {  return departmentService.getDepartment(id);  }  @PostMapping  public Department addDepartment(@RequestBody Department department) {  return departmentService.saveDepartment(department);  }  @DeleteMapping("/{id}")  public void deleteDepartment(@PathVariable Long id) {  departmentService.deleteDepartment(id);  }  } |

**6. Testing and Integrating**

* Lancer les microservices Employee et Department et s'assurer qu'ils fonctionnent sur des ports différents (9000 et 9002).
* Tester les endpoints du microservice Department à l'aide d'outils comme Postman pour effectuer des opérations CRUD.

1. **Microservice Étendre l'application Web pour intégrer le microservice Department**

Si vous souhaitez que l'application Web interagisse également avec le microservice Department, ajoutez une classe DepartmentProxy similaire à la classe EmployeeProxy dans l'application Web pour gérer les requêtes vers le service des départements, et mettez à jour le front-end pour gérer les départements.

Dans le fichier application.properties de l'application Web, ajoutez une nouvelle propriété pour l'URL du microservice Department :

|  |
| --- |
| com.employees.webapp.departmentApiUrl=http://localhost:9002 |

Cette URL doit pointer vers le microservice Department, que nous avons précédemment configuré pour fonctionner sur le port 9002.

### 7.1  ****Créer la classe : DepartmentProxy****

The DepartmentProxy class will act as a client for the Department microservice, similar to the EmployeeProxy class. This class will use RestTemplate to send HTTP requests to the Department API.

|  |
| --- |
| package com.employees.webapp.proxy;  import com.employees.webapp.model.Department;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.core.ParameterizedTypeReference;  import org.springframework.http.HttpMethod;  import org.springframework.http.ResponseEntity;  import org.springframework.stereotype.Component;  import org.springframework.web.client.RestTemplate;  @Component  public class DepartmentProxy {  @Autowired  private CustomProperties props;  private final RestTemplate restTemplate = new RestTemplate();  public Iterable<Department> getDepartments() {  String apiUrl = props.getDepartmentApiUrl() + "/departments";  ResponseEntity<Iterable<Department>> response = restTemplate.exchange(  apiUrl,  HttpMethod.GET,  null,  new ParameterizedTypeReference<Iterable<Department>>() {}  );  return response.getBody();  }  public Department getDepartment(Long id) {  String apiUrl = props.getDepartmentApiUrl() + "/departments/" + id;  return restTemplate.getForObject(apiUrl, Department.class);  }  public Department createDepartment(Department department) {  String apiUrl = props.getDepartmentApiUrl() + "/departments";  return restTemplate.postForObject(apiUrl, department, Department.class);  }  public void deleteDepartment(Long id) {  String apiUrl = props.getDepartmentApiUrl() + "/departments/" + id;  restTemplate.delete(apiUrl);  }  } |

### 7.2 ****Ajouter la classe modele Department****

Créer un modèle Department dans l'application Web pour représenter l'entité Department :

|  |
| --- |
| package com.employees.webapp.model;  import lombok.Data;  @Data  public class Department {  private Long id;  private String name;  private String location;  } |

### 7.3. ****Mettre à jour la classe : CustomProperties****

Modifier CustomProperties pour inclure un getter pour departmentApiUrl :

|  |
| --- |
| package com.employees.webapp.config;  import org.springframework.boot.context.properties.ConfigurationProperties;  import org.springframework.context.annotation.Configuration;  import lombok.Data;  @Configuration  @ConfigurationProperties(prefix = "com.employees.webapp")  @Data  public class CustomProperties {  private String apiUrl;  private String departmentApiUrl; // Add this line  } |

### 7.4. ****Service Layer: DepartmentService****

Créer une classe DepartmentService pour gérer la logique métier liée aux départements et utiliser la classe DepartmentProxy pour interagir avec l'API Department.

|  |
| --- |
| package com.employees.webapp.service;  import com.employees.webapp.model.Department;  import com.employees.webapp.proxy.DepartmentProxy;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  @Service  public class DepartmentService {  @Autowired  private DepartmentProxy departmentProxy;  public Iterable<Department> getDepartments() {  return departmentProxy.getDepartments();  }  public Department getDepartment(Long id) {  return departmentProxy.getDepartment(id);  }  public Department saveDepartment(Department department) {  return departmentProxy.createDepartment(department);  }  public void deleteDepartment(Long id) {  departmentProxy.deleteDepartment(id);  }  } |

### 7.5. ****Controller Layer: DepartmentController****

Créer un DepartmentController pour gérer les requêtes HTTP et servir les pages liées aux départements :

|  |
| --- |
| package com.employees.webapp.controller;  import com.employees.webapp.model.Department;  import com.employees.webapp.service.DepartmentService;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.\*;  @Controller  public class DepartmentController {  @Autowired  private DepartmentService departmentService;  @GetMapping("/departments")  public String departments(Model model) {  Iterable<Department> departments = departmentService.getDepartments();  model.addAttribute("departments", departments);  return "departments";  }  @GetMapping("/department/{id}")  public String departmentDetail(@PathVariable Long id, Model model) {  Department department = departmentService.getDepartment(id);  model.addAttribute("department", department);  return "departmentDetail";  }  @GetMapping("/department/create")  public String createDepartmentForm(Model model) {  model.addAttribute("department", new Department());  return "createDepartment";  }  @PostMapping("/department/create")  public String createDepartment(@ModelAttribute Department department) {  departmentService.saveDepartment(department);  return "redirect:/departments";  }  @GetMapping("/department/delete/{id}")  public String deleteDepartment(@PathVariable Long id) {  departmentService.deleteDepartment(id);  return "redirect:/departments";  }  } |

### 7. ****Création des templates: Thymeleaf****

Créer des modèles HTML pour afficher et gérer les départements dans le répertoire src/main/resources/templates.

* **departments.html** : Affiche une liste des départements.
* **departmentDetail.html** : Affiche les détails d'un département spécifique.
* **createDepartment.html** : Fournit un formulaire pour créer un nouveau département.

Voici un aperçu de chaque modèle Thymeleaf que vous pouvez créer pour gérer les départements.

**1. departments.html**

Ce modèle affiche une liste de tous les départements avec des options pour afficher les détails, modifier et supprimer chaque département.

|  |
| --- |
| <!DOCTYPE html>  <html xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="UTF-8">  <title>Departments</title>  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  </head>  <body>  <div class="container">  <h2>Department List</h2>  <a th:href="@{/department/create}" class="btn btn-primary">Add New Department</a>  <table class="table table-striped">  <thead>  <tr>  <th>ID</th>  <th>Name</th>  <th>Location</th>  <th>Actions</th>  </tr>  </thead>  <tbody>  <tr th:if="${departments.empty}">  <td colspan="4">No departments available</td>  </tr>  <tr th:each="department : ${departments}">  <td th:text="${department.id}"></td>  <td th:text="${department.name}"></td>  <td th:text="${department.location}"></td>  <td>  <a th:href="@{/department/{id}(id=${department.id})}" class="btn btn-info">View</a>  <a th:href="@{/department/create/{id}(id=${department.id})}" class="btn btn-warning">Edit</a>  <a th:href="@{/department/delete/{id}(id=${department.id})}" class="btn btn-danger">Delete</a>  </td>  </tr>  </tbody>  </table>  </div>  </body>  </html> |

**2. departmentDetail.html**

Ce modèle affiche les détails d'un département spécifique.

|  |
| --- |
| <!DOCTYPE html>  <html xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="UTF-8">  <title>Department Details</title>  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  </head>  <body>  <div class="container">  <h2>Department Details</h2>  <table class="table table-bordered">  <tr>  <th>ID</th>  <td th:text="${department.id}"></td>  </tr>  <tr>  <th>Name</th>  <td th:text="${department.name}"></td>  </tr>  <tr>  <th>Location</th>  <td th:text="${department.location}"></td>  </tr>  </table>  <a th:href="@{/departments}" class="btn btn-secondary">Back to List</a>  </div>  </body>  </html> |

**3. createDepartment.html**

Ce modèle fournit un formulaire pour créer ou modifier un département. Les champs du formulaire sont préremplis si vous mettez à jour un département existant.

|  |
| --- |
| <!DOCTYPE html>  <html xmlns:th="http://www.thymeleaf.org">  <head>  <meta charset="UTF-8">  <title>Create or Update Department</title>  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css">  </head>  <body>  <div class="container">  <h2 th:text="${department.id == null ? 'Create New Department' : 'Edit Department'}"></h2>  <form th:action="@{/department/create}" th:object="${department}" method="post">  <div class="form-group">  <label for="name">Name</label>  <input type="text" id="name" th:field="\*{name}" class="form-control" placeholder="Department Name" required>  </div>  <div class="form-group">  <label for="location">Location</label>  <input type="text" id="location" th:field="\*{location}" class="form-control" placeholder="Department Location" required>  </div>  <button type="submit" class="btn btn-success">Save</button>  <a th:href="@{/departments}" class="btn btn-secondary">Cancel</a>  </form>  </div>  </body>  </html> |