Tutorial 3 – Web Application With Spring Boot (1b)

❖ Contents:

- Develop the project from Tutorial 2
- Add more CRUD features with Spring JPA
- Create more Thymeleaf views for the web application

❖ Creating a page to update an employee:

1. Add a controller method into `EmployeeController`:

```
@GetMapping(value = "/update/{id}")
public String updateEmployee(
        @PathVariable(value = "id") Long id, Model model) {
    Employee employee = employeeRepository.getById(id);
    model.addAttribute(employee);
    return "employeeUpdate";
}
```

2. Create a Thymeleaf template to display a form containing employee information at `src/main/resources/templates/employeeUpdate.html` (pay attention to the form's action and the hidden input):

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Edit Employee</title>
</head>
<body>
<div>
    <h2>UPDATE EMPLOYEE</h2>
    <form method="post" action="/save" th:object="${employee}">
        <input type="hidden" th:field="*{id}"/>
        <p>
            <label>Employee name:</label>
            <input class="form-control" type="text" th:field="*{name}"/>
        </p>
        <p>
            <label>Employee age:</label>
            <input class="form-control" type="number" th:field="*{age}"/>
        </p>
        <p class="form-group">
            <label>Employee image:<br>
                <img th:src="*{image}" width="100" height="100"/>
                <input class="form-control" type="text" th:field="*{image}"/>
            </label>
        </p>
        <p class="form-group">
            <label>Employee address:
                <input class="form-control" type="text" th:field="*{address}"/>
            </label>
        </p>
        <p>
            <button type="submit">UPDATE</button>
        </p>
    </form>
</div>
</body>
</html>
```

3. Create another method in `EmployeeController` to save the submitted form data into database:

```
@PostMapping(value = "/save")
public String saveUpdate(Employee employee) {
    employeeRepository.save(employee);
    return "redirect:/detail/" + employee.getId();
}
```

This controller method automatically receives the request parameters and merge them into an `Employee` object (including the important `id` parameter).

4. Modify `employeeList.html` to make the UPDATE text next to each employee become a link to the employee's update page:

```
<td><a th:href="'/update/' + ${employee.id}">UPDATE</a></td>
<td>DELETE</td>
```

❖ Creating a page to add an employee:

1. Create a method in `EmployeeController` to show a form for entering new employee information. The form will be displayed in a Thymeleaf template called `employeeAdd.html`. This form doesn't need to include an `id` field. Also, to make sure the form contains only valid `Employee` fields, we will attach a new (empty) Employee object to it using the `th:object` attribute and the asterisk `*` operator (Thymeleaf selection expression):

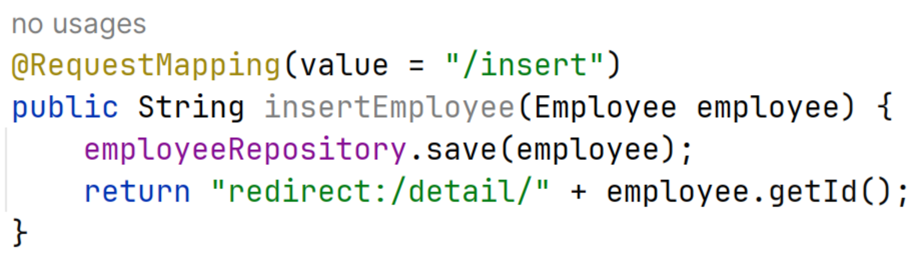
```java
@GetMapping(value = "/add")
public String addEmployee(Model model) {
    Employee employee = new Employee();
    model.addAttribute("employee", employee);
    return "employeeAdd";
}
```

Below is source code of `src/main/resources/templates/employeeAdd.html`

```html
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Add Employee</title>
</head>
<body>
<div>
    <h2>ADD EMPLOYEE</h2>
    <form th:action="'/save'" th:object="${employee}" method="post">
        <p>
            <label>Employee name: </label>
            <input type="text" th:field="*{name}">
        </p>
        <p>
            <label>Employee age: </label>
            <input type="number" th:field="*{age}">
        </p>
        <p class="form-group">
            <label>Employee image: </label>
            <input type="text" th:field="*{image}">
        </p>
        <p class="form-group">
            <label>Employee address </label>
            <input type="text" th:field="*{address}">
        </p>
        <p>
            <button type="submit">ADD</button>
        </p>
    </form>
</div>
</body>
</html>
```

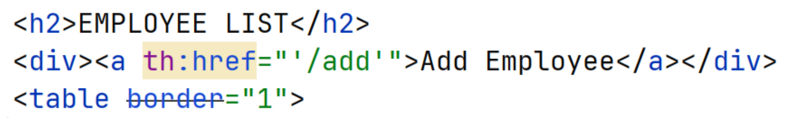
2. Please note that, by using `/save` as the above form's action, we intend to use the existing `saveUpdate()` method in `EmployeeController` to receive the submitted form

and insert the `Employee` object as a new record in database. This method inserts a new record to the database when the received `Employee` object doesn't contain an `id`.



```
no usages
@RequestMapping(value = "/insert")
public String insertEmployee(Employee employee) {
    employeeRepository.save(employee);
    return "redirect:/detail/" + employee.getId();
}
```

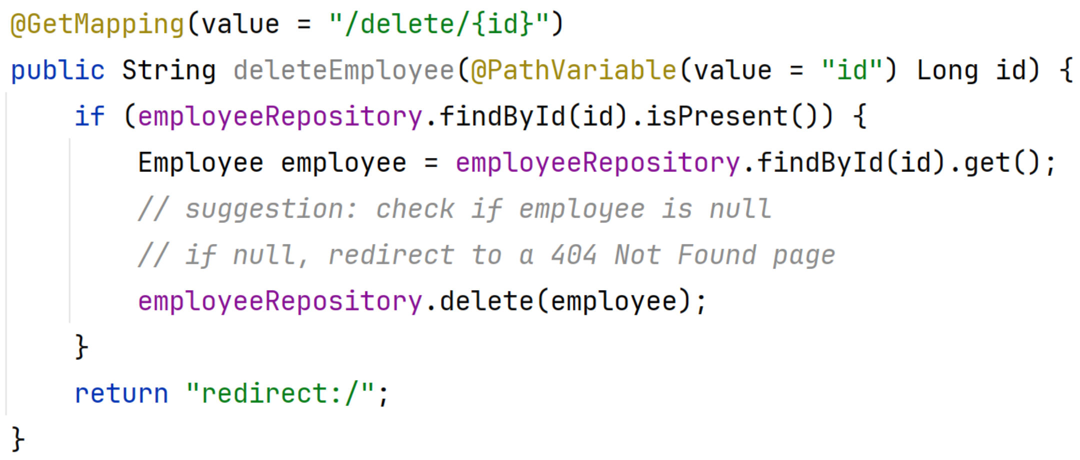
3. Modify `employeeList.html` to include a link to the Add Employee page:



```
<h2>EMPLOYEE LIST</h2>
<div><a th:href="'/add'">Add Employee</a></div>
<table ~~border~~="1">
```

❖ Developing the feature to delete an employee:

1. Create a controller method in `EmployeeController` which receives a path parameter named `id`, retrieves the `Employee` object with that `id`, then deletes the object from database if it exists. After that, redirect to the employee list page. Make sure that you research the difference between `getById` and `findById` and learn how to use `findById` properly.



```
@GetMapping(value = "/delete/{id}")
public String deleteEmployee(@PathVariable(value = "id") Long id) {
    if (employeeRepository.findById(id).isPresent()) {
        Employee employee = employeeRepository.findById(id).get();
        // suggestion: check if employee is null
        // if null, redirect to a 404 Not Found page
        employeeRepository.delete(employee);
    }
    return "redirect:/";
}
```

2. Modify the `employeeList.html` template to make the DELETE texts become links to delete the corresponding employee.

```
<td><a th:href="'/update/' + ${employee.id}">UPDATE</a></td>
<td><a th:href="'/delete/' + ${employee.id}">DELETE</a></a>
```

❖ TASKS:

- Compress your project into a `.zip` file and submit to the tutorial’s submission box.