Ex1: Spring MVC with Thymeleaf

Consider the table `SB_EMPLOYEES_1` available in `VBSSUITE` under `SSDX_ENG`. The sequence that generates the `EMPLOYEE_ID` value is `EMPLOYEE_ID_SEQ`. This exercise will introduce you to Spring MVC and Thymeleaf. The latter will be used to render Views (HTML pages).

In this exercise, `@Controller` annotation will be used instead of `@RestController`, since we are expecting views to be rendered.

1. Create a new spring boot project and add the dependencies you think you might use. Don't worry, if you missed some of them, you can add them later as needed.
2. Create a `@Entity` class to map our object (Employee) to the database table.
3. Create a `@Repository` interface extending **JpaRepository<*Entity, primaryKeyType*>**.
4. Create a `@Controller` class with the needed methods:
    1. Each method in `@Controller` class is expected to return a template name (HTML page) so that it can be rendered and displayed.
    2. The common endpoint will be *"/valoores/exercise1"*, and the methods should be annotated with `@GetMapping` or `@PostMapping` depending on the functionality.
    3. You will have to display the following Views:
        i. The first one is *"welcome.html"* containing a welcome message and a button "display employees".
        ii. When *"display employees"* button is clicked, you will be redirected to *"employees.html"*, which will display all the available employees in a grid. You will also have to create a button *"add employee"* in the bottom of the grid to add a specific employee.
        iii. When *"add employee"* button is clicked, you will be redirected to "*add-employee.html"* form where you will be able to fill in the information of a new employee to be added and save it when the *"save"* button is clicked. If the save is successful, you will be redirected back to the employees grid.
5. All HTML files should be created under *"src/main/resources/templates"*. In case you wanted to use CSS/JavaScript, the files should be created under *"src/main/resources/static"*.

Thymeleaf notes:

Thymeleaf gives you the ability to display the content of a Java object in HTML once you add the following to the <html> tag of your document:

```
<html xmlns:th="http://www.thymeleaf.org">
```

If you want to use the value of a Java object in Thymeleaf, you need to add it to the Model object in your `@Controller` class methods, as follows:

```java
@GetMapping("/employees")
public String showAllEmployees(Model model) {
    model.addAttribute("employees", empRepository.findAll());
    return "employee";
}
```

In the above image:

(1) “employees”: the name of the object I want to use in the HTML file.

(2) “empRepository.findAll()”: a list of employees fetched from the database using the JPA Repository

(3) return “employee”: employee is in fact the “employee.html” file that is getting returned.

From the HTML file perspective, we can use the object (which is a List) and loop over it to display it in a grid as follows:

```html
<table class="table table-striped table-responsive-md">
    <thead>
        <tr>
            <th>ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Email</th>
            <th>Manager ID</th>
            <th>Job Title</th>
        </tr>
    </thead>
    <tbody>
        <tr th:each="employee : ${employees}">
            <td th:text="${employee.id}"></td>
            <td th:text="${employee.firstName}"></td>
            <td th:text="${employee.lastName}"></td>
            <td th:text="${employee.email}"></td>
            <td th:text="${employee.managerId}"></td>
            <td th:text="${employee.jobTitle}"></td>
        </tr>
    </tbody>
</table>
```

Notice how `${employees}` matches the name “employees” that we added previously in the controller using `model.addAttribute()` function.
“employee.id”, “employee.firstName”, etc match the fields of the Employee `@Entity` class defined in our project.

In case we want to redirect to another HTML file (another view), we can do this from the controller using the *"redirect:/"* keyword in the return statement of the controller methods, as follows:

```
return "redirect:/valoores/exercise1/employees";
```

where *"/valoores/exercise1/employees"* is a known endpoint in our project.

From the HTML file perspective and when using forms, the "action" attribute can be manipulated in Thymeleaf as follows:

```
<form th:action="@{/valoores/exercise1/employee/create}" th:object="${employee}" method="post">
    <div class="row">
        <div class="form-group col-md-5">
            <label for="firstName" class="col-form-label">First Name</label>
            <input type="text" th:field="*{firstName}" class="form-control" id="firstName" placeholder="First Name">
        </div>
        <div class="form-group col-md-5">
            <label for="lastName" class="col-form-label">Last Name</label>
            <input type="text" th:field="*{lastName}" class="form-control" id="lastName" placeholder="Last Name">
        </div>
        <div class="form-group col-md-5">
            <label for="email" class="col-form-label">Email</label>
            <input type="text" th:field="*{email}" class="form-control" id="email" placeholder="Email">
        </div>
        <div class="form-group col-md-5">
            <label for="managerId" class="col-form-label">Manager ID</label>
            <input type="text" th:field="*{managerId}" class="form-control" id="managerId" placeholder="Manager ID">
        </div>
        <div class="form-group col-md-5">
            <label for="jobTitle" class="col-form-label">Job Title</label>
            <input type="text" th:field="*{jobTitle}" class="form-control" id="jobTitle" placeholder="Job Title">
        </div>
    </div>
    <div class="row">
        <div class="col-md-6 mt-5">
            <input type="submit" class="btn btn-primary" value="Add Employee">
        </div>
    </div>
</form>
```

(1) **th:action** points to the POST endpoint where data should be submitted. Thymeleaf uses the "@{/your/post/method}" notation to point to the POST method defined in the `@Controller`.

(2) **th:object** specifies the object we want to send as body data to the POST method.

(3) **th:field** specifies that the content of the body for a specific attribute is the value entered in the corresponding HTML field. For instance, the value entered in the input of the first name in the above picture fills the "firstName" field of the "Employee" object with the value entered by the user. That's how upon submitting the form, the Employee created is persisted to the database using the `save()` method of the @Repository class defined in the project.

(4) You can find the documentation for Thymeleaf at the below location: \\10.1.10.236_Deployment\DEV\V21-5\Documentation\Front End\Templating Engines