Tutorial 4 – Web Application With Spring Boot (1c)

❖ Contents:

- Continue the project from **Tutorial 3**
- Practice the one-to-many relationship between data entities in Spring Boot
- Establish web template using Thymeleaf layout dialect

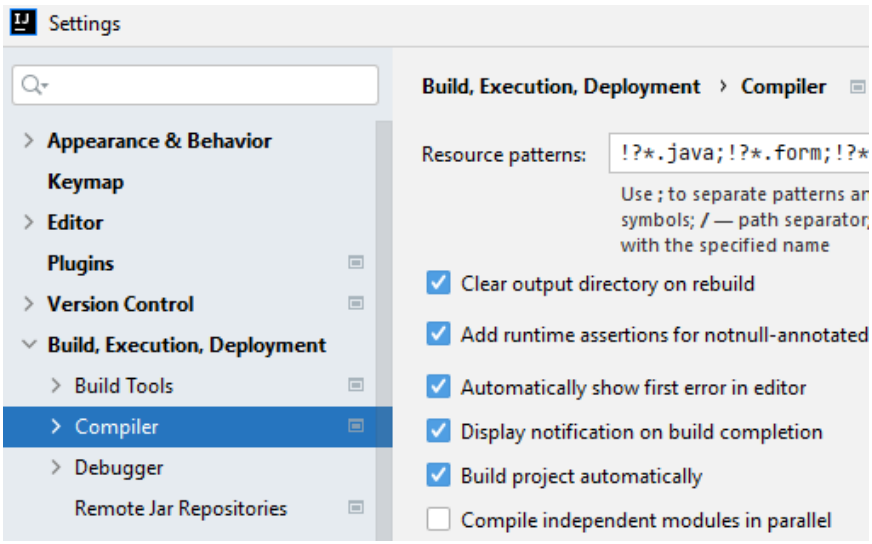
❖ Spring Boot development tips:

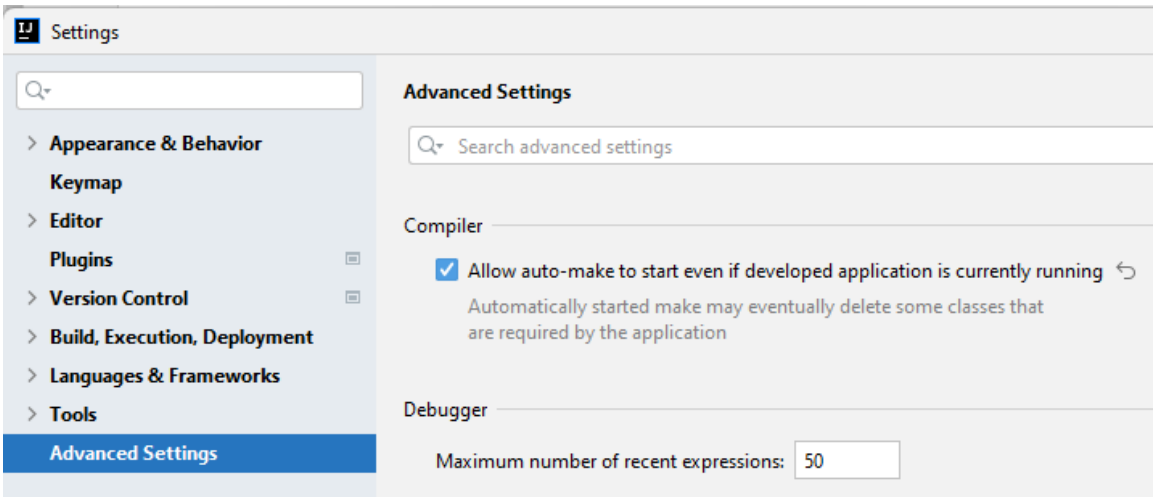
1. When you use start.spring.io to create new Java Spring Boot project, you should be aware of the type of database server that you have. Add "MySQL Driver" dependency when you have MySQL server installed (usually along with MySQL Workbench) or "MariaDB Driver" if you have MariaDB server installed (usually along with the XAMPP package.
2. Serve static web pages and setup automatic reload for static resources:

   Spring Boot comes with a pre-configured implementation of `ResourceHttpRequestHandler` to facilitate serving static resources. By default, this handler serves static content from any of the `/static`, `/public`, `/resources`, and `/META-INF/resources` directories that are on the Java **classpath** of the project.

   Since `src/main/resources` is typically on the classpath by default, we can place any of static files under the `src/main/resources/static` folder which is already created by **start.spring.io**. For example, if you put a `style.css` file inside the `/static` directory in your **classpath**, then you can access that file via `http://localhost:8080/style.css`.

   To automatically reload static files, in IntelliJ IDEA, go to **File → Settings**, then check the following checkboxes:





❖ Implement One-To-Many relationship:

1. Add new entity (`Company`) then add a proper annotations to represent the relationship between `Company` and `Employee`. In `Company`, the relationship is `@OneToMany` and is represented by the `employees` attribute as shown in the figure below. In `Employee`, the relationship is `@ManyToOne` and is represented by the `company` attribute. Don't forget to create getter method for `employees` attribute and getter & setter methods for `company` attribute.

```java
@Entity
public class Company {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String image;
    private String address;
    @OneToMany(mappedBy = "company")
    private List<Employee> employees;

    // constructors, getters & setters
}
```

Figure 1: Company.java

```java
@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private int age;
    private String image;
    private String address;
    @ManyToOne
    private Company company;

    // constructors, getters & setters
}
```

Figure 2: Employee.java

2. Add the `CompanyRepository` for `Company` entity.

3. Add the `CompanyController` to contain controller methods related to `Company` entity. This controller class should have a `@RequestMapping` annotation which sets a prefix of `/company` for the URL of all controller methods within this class. This way, the actual URL for the `getCompanyById()` method is not `/{id}` but actually `/company/{id}` (the prefix URL is added before the controller method's URL).

```java
@Controller
@RequestMapping("/company")
public class CompanyController {
    @Autowired
    CompanyRepository companyRepository;

    @RequestMapping(value = "/{id}")
    public String getCompanyById(
            @PathVariable(value = "id") Long id, Model model) {
        Company company = companyRepository.getById(id);
        model.addAttribute("company", company);
        model.addAttribute("employees", company.getEmployees());
        return "companyDetail";
    }
```

4. Create CRUD features for `Company`. You need to add these controller methods:
    - The `/company/list` URL triggers `getAllCompany()` controller method, which uses the `companyList.html` template.
    - The `/company/add` URL triggers `addCompany()` controller method, which uses the `companyAdd.html` template.
    - The `/company/{id}` URL triggers `getCompanyById()` controller method, which uses the `companyDetail.html` template. This page should show a list of employees in the company.
    - The `/company/update/{id}` URL triggers `updateCompany()` controller method, which uses the `companyUpdate.html` template.
    - The `/company/delete/{id}` URL triggers `deleteCompany()` controller method, which redirects to the `/company/list` page after deleting the company.

    (*) Refer to previous tutorials to learn how to code CRUD features for an entity.

5. Update `EmployeeController` to change URLs of employee CRUD features and to add `CompanyRepository` so that you can choose an employee's company when adding a new employee.

```java
@Controller
@RequestMapping(value = "/employee")
public class EmployeeController {
    @Autowired
    EmployeeRepository employeeRepository;
    @Autowired
    CompanyRepository companyRepository;
```

```
@RequestMapping(value = "/add")
public String addEmployee(Model model) {
    Employee employee = new Employee();
    List<Company> companies = companyRepository.findAll();
    model.addAttribute("companies", companies);
    model.addAttribute("employee", employee);
    return "employeeAdd";
}
```

6. Update the template `employeeAdd.html` to add a `<select>` element (drop-down list) for selecting company.

```
<p class="form-group">
    <label>Company </label>
    <select class="form-select" th:field="*{company}">
        <option th:each="comp : ${companies}"
                th:value="${comp.id}"
                th:text="${comp.name}"/>
    </select>
</p>
```

❖ Re-organizing Thymeleaf templates to use layout:

1. Add Thymeleaf layout dependency in `pom.xml`:

```
<dependency>
    <groupId>nz.net.ultraq.thymeleaf</groupId>
    <artifactId>thymeleaf-layout-dialect</artifactId>
    <version>3.0.0</version>
</dependency>
```

Remember to reload Maven project to resolve this new dependency.

2. Add a layout template named `_layout.html`:

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
<head>
    <meta charset="UTF-8">
    <title>Employee Management System</title>
    <link rel="stylesheet" th:href="@{/css/style.css}">
</head>
<body>
    <nav>
        <a href="/employee/">Employee List</a>
        <a href="/employee/add">Add Employee</a>
```

```
        <a href="/company/list">Company List</a>
        <a href="/company/add">Add Company</a>
    </nav>
    <div layout:fragment="content">
        <!-- content page will override this -->
    </div>
</body>
</html>
```

This layout contains a navigation bar with all the links to other pages. This navigation bar is meant to be displayed on top of every page. This layout also contains a fragment area named `content` which will be filled up by any template that *decorates* this layout.

3. Edit employee list page (`employeeList.html`):

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org"
      xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
layout:decorate="_layout">
<head>
    <meta charset="UTF-8">
    <title>Employee List</title>
</head>
<body>
<div layout:fragment="content">
    <h2>EMPLOYEE LIST</h2>
    <table border="1">
        <tr>
            <th>ID</th>
            <th>Name</th>
            <th>Image</th>
            <th>Update</th>
            <th>Delete</th>
        </tr>
        <tr th:each="employee : ${employees}">
            <td th:text="${employee.id}"/>
            <td>
              <a th:href="'/detail/' + ${employee.id}" th:text="${employee.name}"/>
            </td>
            <td>
              <img th:src="${employee.image}" width="100" height="100"/>
            </td>
            <td><a th:href="'/update/' + ${employee.id}">UPDATE</a></td>
            <td><a th:href="'/delete/' + ${employee.id}">DELETE</a></td>
        </tr>
    </table>
</div>
</body>
</html>
```

4. Edit other templates (`employeeAdd`, `employeeDetail`, `employeeUpdate`) similarly to the way you editted the `employeeList` template.

❖ Lastly, run and debug your application!

Once finished, compress your project into a `.zip` file and submit the file to the tutorial's submission box.