TUTORIAL 3 – DEVELOP JAVA WEB WITH SPRING BOOT (1)

❖ Content:

- Create Java Spring Boot project in IntelliJ with autoconfiguration
- Create table with Hibernate
- Make CRUD feature with JPA
- Create view for web with Thymeleaf

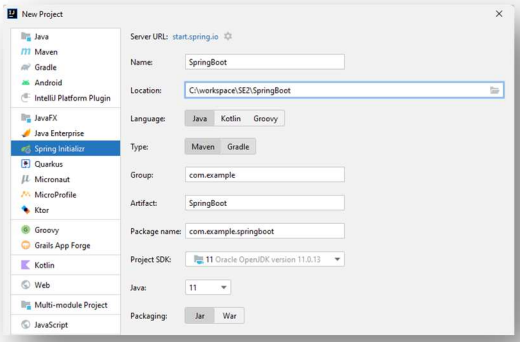
❖ Introduction:

- Spring framework: a Java platform that provides comprehensive infrastructure support for developing Java application
- Spring Boot: a tool that makes developing web application and microservices with Spring framework faster and easier with autoconfiguration
- Hibernate: an object-relational mapping (ORM) tool for Java programming language that simplifies the interaction with the database
- JPA (Java Persistence API): a collection of classes and methods to persistently store that vast amounts of data into a database
- Thymeleaf: a modern server-side Java template engine for both web and standalone environments

❖ Instructions:

1. Create new Java Spring Boot project in IntelliJ using Spring Initializer

- **New Project**
- Select **Spring Initializr**
- Input project parameters:
  - Project name
  - Project location
  - Language: **Java**
  - Type: **Maven**
  - Java version: **11**
  - Packaging: **Jar**



- Click **Next**

Figure 1 - Create new Spring Boot project (1)

- Spring Boot version: **2.6.1**
- Select dependencies:
  - Spring Web
  - Thymeleaf
  - Spring Data JPA
  - MySQL Driver
- Click **Finish**

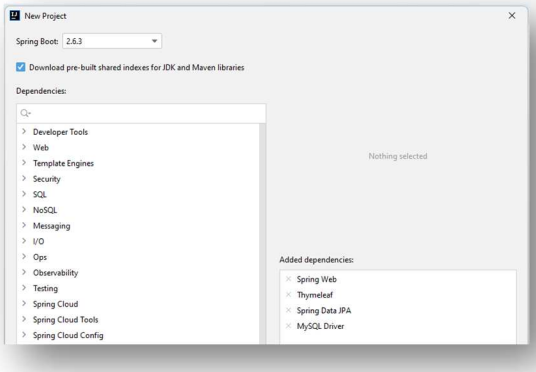


Figure 2 - Create new Spring Boot project (2)

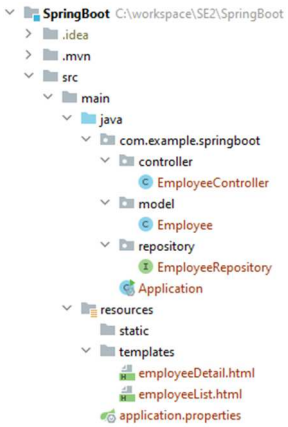


Figure 3 - Sample project structure

2. Config parameters for MySQL connection, JPA & Hibernate

```
# MYSQL
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root

# JPA + HIBERNATE
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.generate-ddl=true
spring.jpa.hibernate.ddl-auto=update

# THYMELEAF
spring.thymeleaf.cache=false
```

*Figure 4 - **application.properties***

3. Create Java class for model (entity) which acts as table in database

```
@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    private String name;
    private int age;
    private String image;
    private String address;

    //auto-generated getters & setters
```

*Figure 5 - **Employee.java***

4. Create Java interface which extends *JpaRepository* for CRUD features

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
```

*Figure 6 - **EmployeeRepository.java** (interface)*

5. Create Java class for controller which gets data from database and renders view

```java
@Controller
public class EmployeeController {
    @Autowired
    EmployeeRepository employeeRepository;

    @RequestMapping(value = "/")
    public String getAllEmployee(Model model) {
        List<Employee> employees = employeeRepository.findAll();
        model.addAttribute( attributeName: "employees", employees);
        return "employeeList";
    }

    @RequestMapping(value = "/{id}")
    public String getEmployeeById(
            @PathVariable(value = "id") Long id, Model model) {
        Employee employee = employeeRepository.getById(id);
        model.addAttribute( attributeName: "employee", employee);
        return "employeeDetail";
    }
}
```

*Figure 7 - **EmployeeController.java***

6. Create HTML page as view

```html
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Employee List</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
          crossorigin="anonymous">
</head>
<body>
  <div class="container col-md-4 text-center mt-4">
      <h2 class="text text-primary">EMPLOYEE LIST</h2>
      <table class="table table-info mt-3">
          <thead>
            <tr>
                <th>ID</th>
                <th>Name</th>
                <th>Image</th>
            </tr>
          </thead>
          <tbody>
              <tr th:each="employee : ${employees}">
                  <td th:text="${employee.id}" />
                  <td> <a th:text="${employee.name}"/> </td>
                  <td>
                      <a th:href="'/' + ${employee.id}" > <img th:src="${employee.image}" width="100" height="100"> </a>
                  </td>
              </tr>
          </tbody>
      </table>
  </div>
</body>
</html>
```

Figure 8 - employeeList.html

```html
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Employee Detail</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
          crossorigin="anonymous">
</head>
<body>
<div class="container col-md-5 text-center mt-4">
    <h2 class="text text-primary mb-4">EMPLOYEE DETAIL</h2>
    <div class="row bg-light">
        <div class="col">
            <img th:src="${employee.image}" width="200" height="200">
        </div>
        <div class="col">
            <h1 class="text-success" th:text="${employee.name}" />
            <h3 th:text="'Age: ' + ${employee.age}" />
            <h3 th:text="'Address: ' + ${employee.address}" />
        </div>
    </div>
</div>
</body>
</html>
```

Figure 9 - employeeDetail.html

7. Run the web application (**CTRL + SHIFT + F10**)

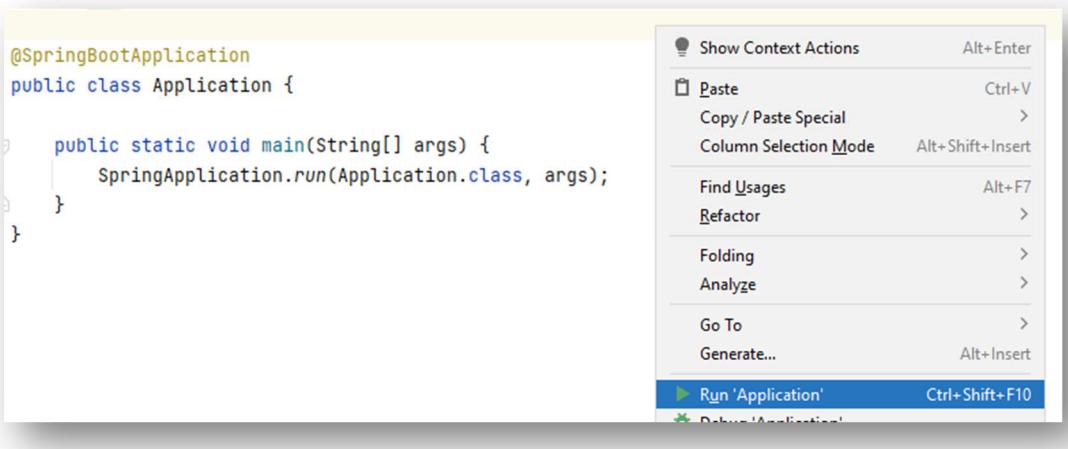


Figure 10 - Application.java

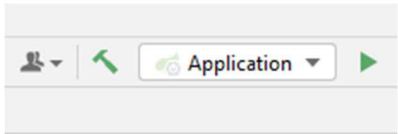


Figure 11 - Run web application

8. Open web browser (such as Chrome) and type address: **http://localhost:8080/**

   Note: Remember to add records to table in database first

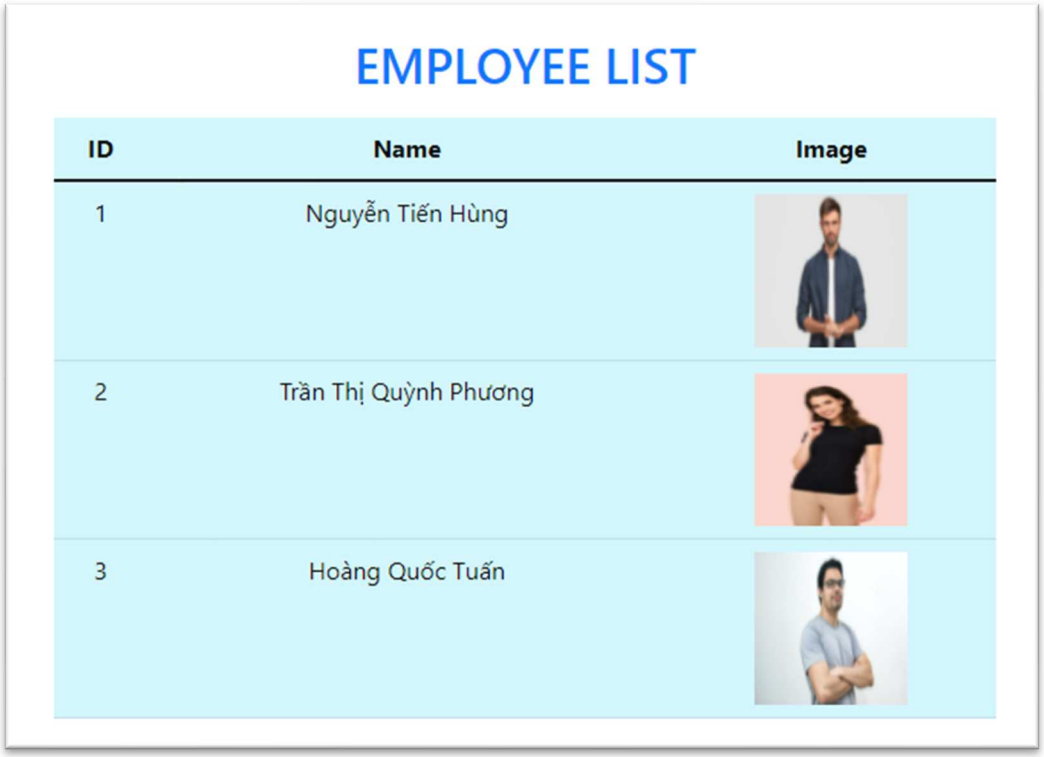


Figure 12 - Employee List page

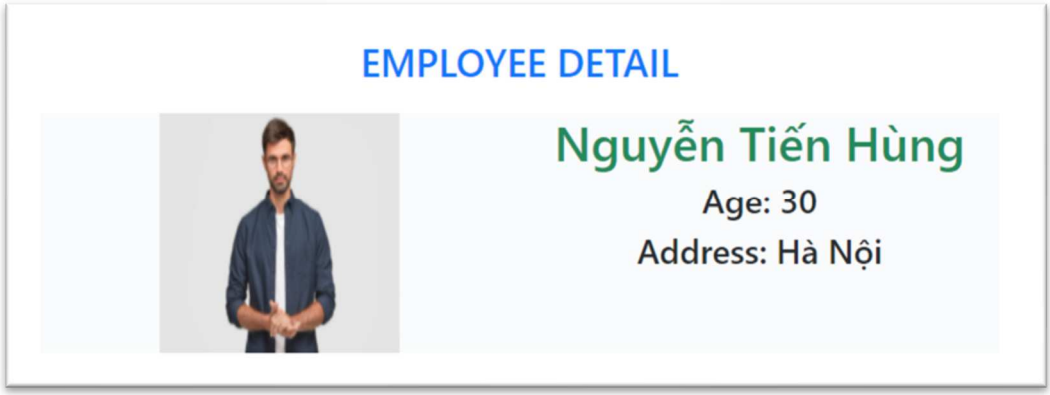


Figure 13 - Employee Detail page

❖ Tasks:

- Complete the remained operations for table CRUD including CREATE, UPDATE and DELETE. You must add new methods in Controller then create new corresponding HTML files (ex: *employeeAdd.html*)
- Compress the whole project and submit to FIT Portal with file name syntax: *FullName_StudentID_SE2_Tut3.rar*

<u>Note:</u> The sample codes will be published after homework deadline