Tutorial 2 – Web Application With Spring Boot (1)

❖ Contents:

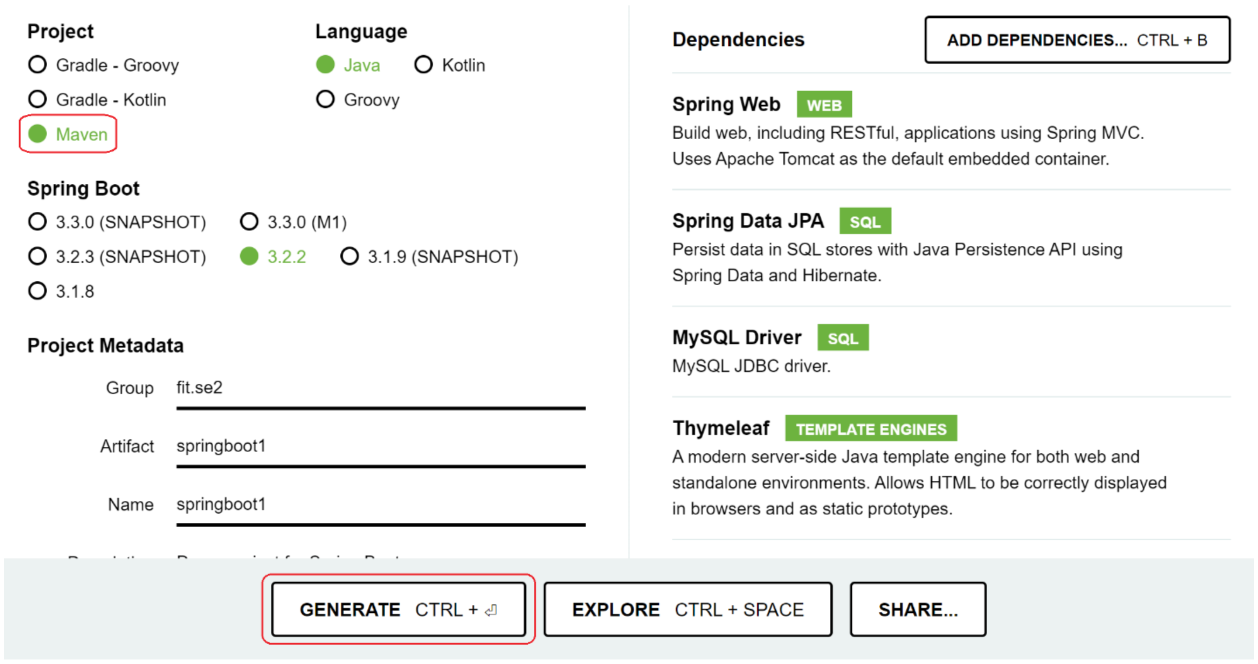
- Create Spring Boot project using start.spring.io and import into IntelliJ IDEA
- Create table with Hibernate
- Make CRUD feature with Spring JPA
- Create views for the web application with Thymeleaf

❖ Concepts:

- **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
- **Spring JPA**: a collection of classes and methods to persistently and conveniently store data in a database from Spring application
- **Thymeleaf**: a modern server-side Java template engine for both web and standalone environments

❖ Create & Configure a Spring Boot project:

1. Create a Spring Boot project by visiting https://start.spring.io



Project
- Gradle - Groovy
- Gradle - Kotlin
- [✓ Maven]

Language
- [✓ Java]
- Kotlin
- Groovy

Spring Boot
- 3.3.0 (SNAPSHOT)
- 3.3.0 (M1)
- 3.2.3 (SNAPSHOT)
- [✓ 3.2.2]
- 3.1.9 (SNAPSHOT)
- 3.1.8

Project Metadata

Group	fit.se2
Artifact	springboot1
Name	springboot1

Dependencies ADD DEPENDENCIES... CTRL + B

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

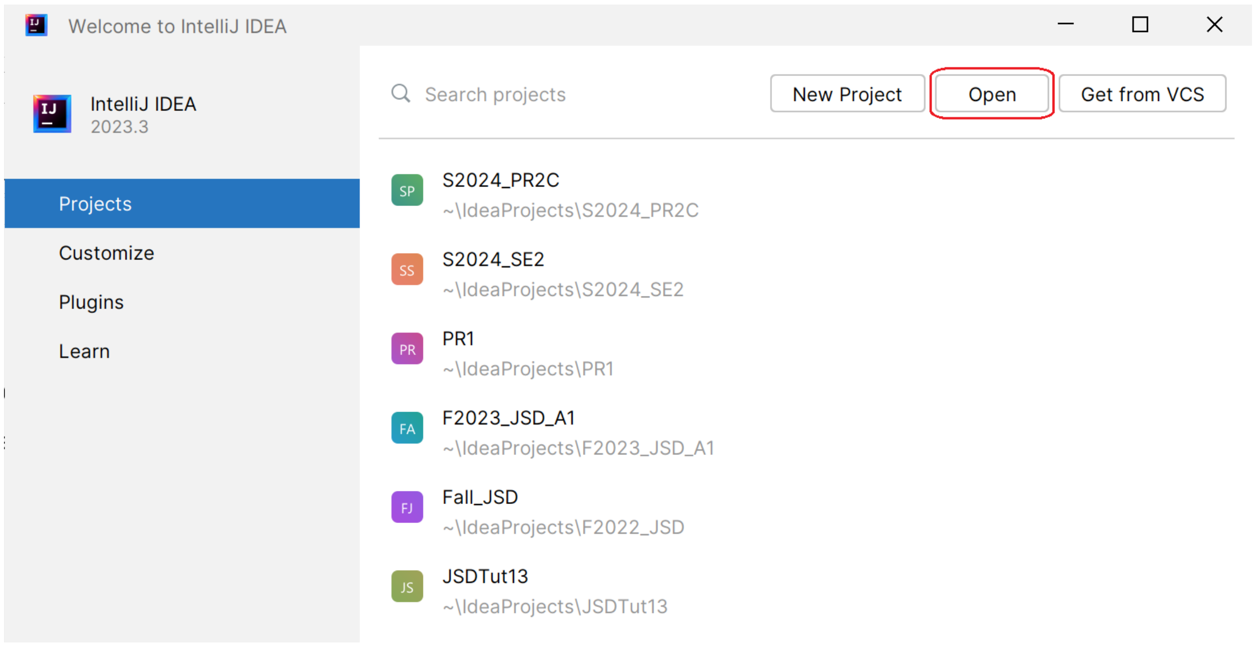
Spring Data JPA SQL
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver SQL
MySQL JDBC driver.

Thymeleaf TEMPLATE ENGINES
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

2. Click GENERATE to download a `zip` file, which contains the newly created Maven project . Extract it to a suitable folder in your computer. For example: `D:\Study\Spring2024\SE2\Week2\springboot1`

3. Open that folder as a project in IntelliJ IDEA:



Welcome to IntelliJ IDEA

IntelliJ IDEA
2023.3

S2024_PR2C
~\IdeaProjects\S2024_PR2C

S2024_SE2
~\IdeaProjects\S2024_SE2

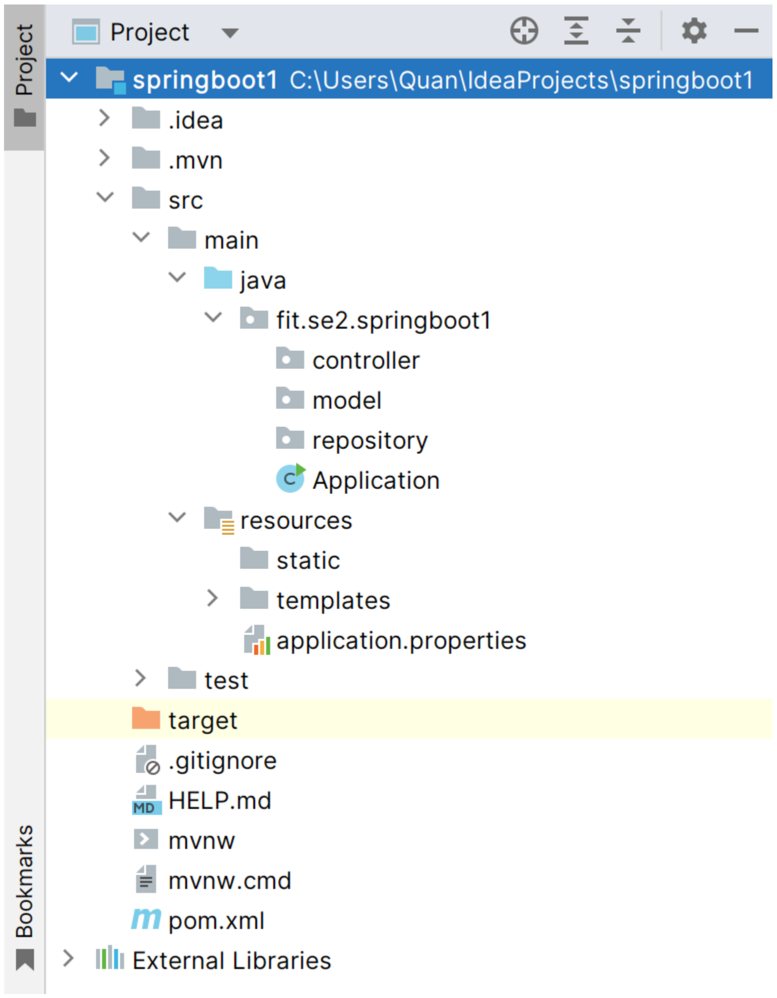
PR1
~\IdeaProjects\PR1

F2023_JSD_A1
~\IdeaProjects\F2023_JSD_A1

Fall_JSD
~\IdeaProjects\F2022_JSD

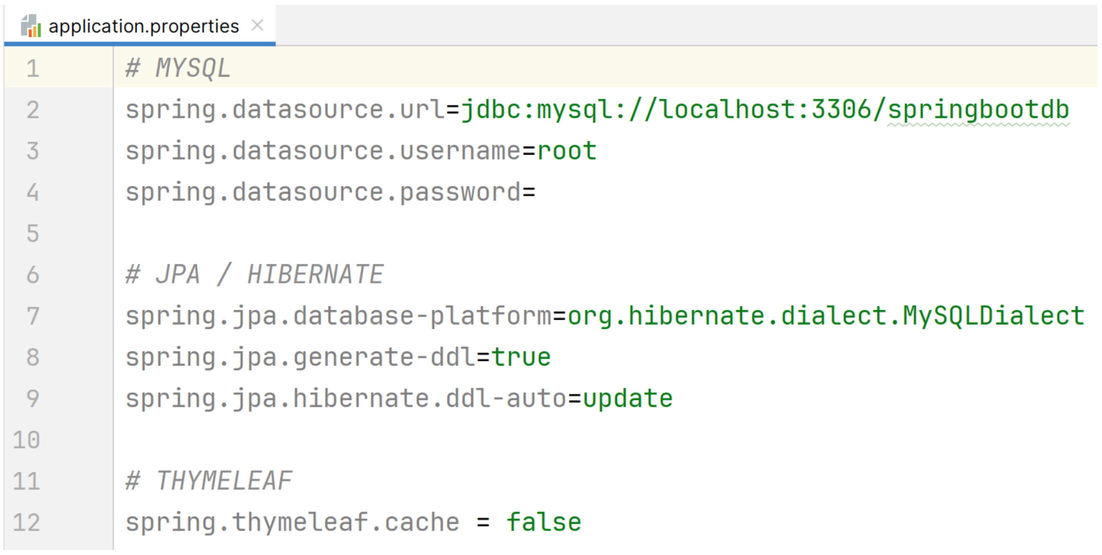
JSDTut13
~\IdeaProjects\JSDTut13

4. Create 3 packages: `controller`, `model` and `repository`. Following is a sample project structure:



```
Project
springboot1  C:\Users\Quan\IdeaProjects\springboot1
  .idea
  .mvn
  src
    main
      java
        fit.se2.springboot1
          controller
          model
          repository
          Application
      resources
        static
        templates
        application.properties
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
External Libraries
```

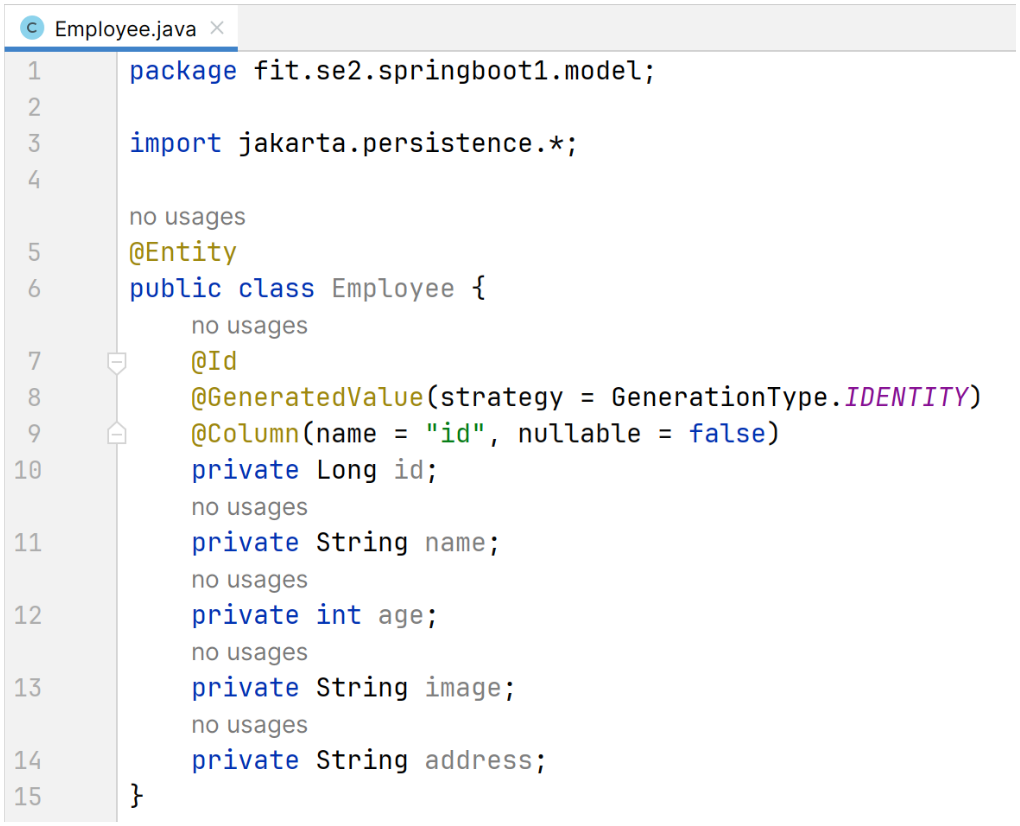
5. Configure parameters for MySQL connection, JPA & Hibernate. The config file is `src/main/resources/application.properties`


application.properties

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

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

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

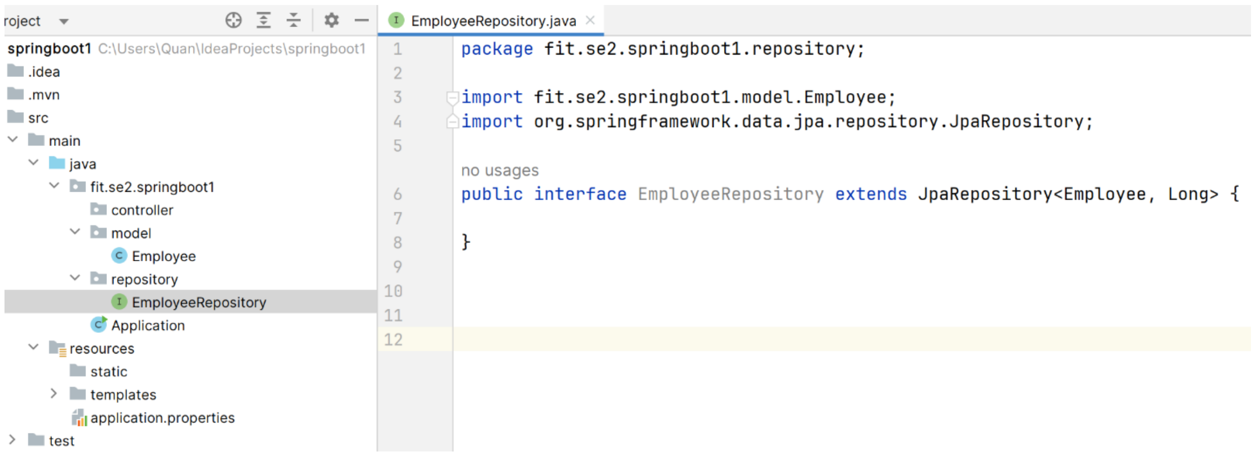
❖ **Implement the Employee List page:**

1. Create Java class for model (entity) called `Employee` which is associated with the `Employee` table in database (*located at sub-package* `model`). Generate getters and setters for all attributes of this class.


Employee.java

```
package fit.se2.springboot1.model;

import jakarta.persistence.*;

no usages
@Entity
public class Employee {
    no usages
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    no usages
    private String name;
    no usages
    private int age;
    no usages
    private String image;
    no usages
    private String address;
}
```

2. Create Java interface which extends `JpaRepository` for CRUD features (*located at sub-package* `repository`)

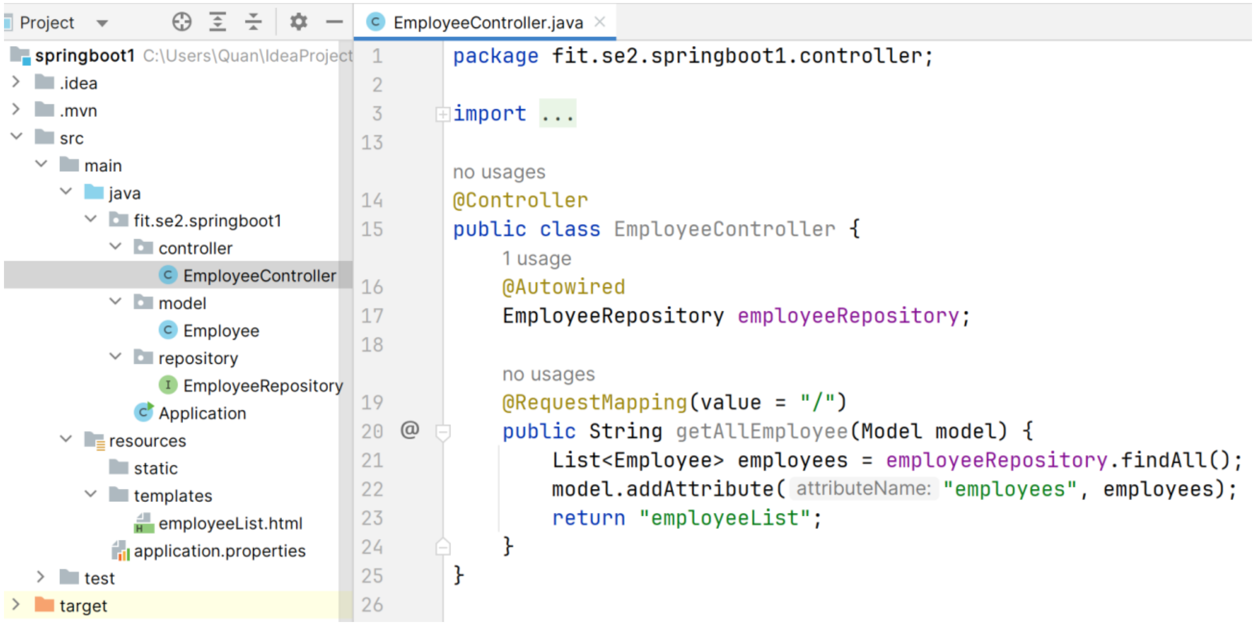

roject

springboot1 C:\Users\Quan\IdeaProjects\springboot1
- .idea
- .mvn
- src
  - main
    - java
      - fit.se2.springboot1
        - controller
        - model
          - Employee
        - repository
          - EmployeeRepository
        - Application
    - resources
      - static
      - templates
      - application.properties
  - test

EmployeeRepository.java

```java
package fit.se2.springboot1.repository;

import fit.se2.springboot1.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;

no usages
public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}
```

3. Create Java class for controller which gets data from database and renders view (*located at sub-package* `controller`)

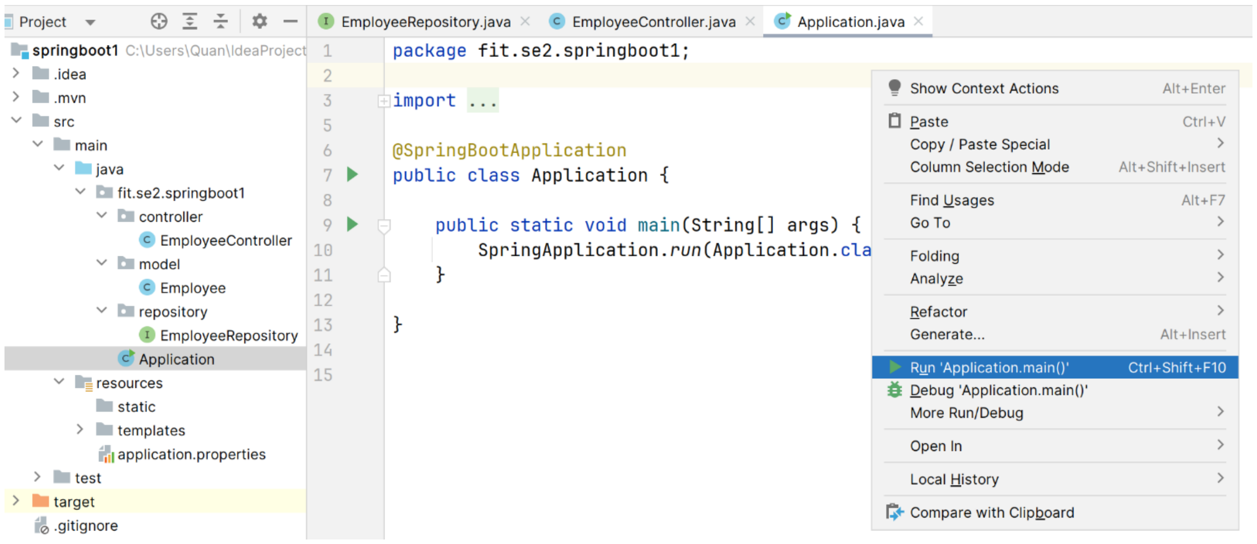

Project

springboot1 C:\Users\Quan\IdeaProject
- .idea
- .mvn
- src
  - main
    - java
      - fit.se2.springboot1
        - controller
          - EmployeeController
        - model
          - Employee
        - repository
          - EmployeeRepository
        - Application
    - resources
      - static
      - templates
        - employeeList.html
      - application.properties
  - test
- target

EmployeeController.java

```java
package fit.se2.springboot1.controller;

import ...

no usages
@Controller
public class EmployeeController {
    1 usage
    @Autowired
    EmployeeRepository employeeRepository;

    no usages
    @RequestMapping(value = "/")
    public String getAllEmployee(Model model) {
        List<Employee> employees = employeeRepository.findAll();
        model.addAttribute( attributeName: "employees", employees);
        return "employeeList";
    }
}
```

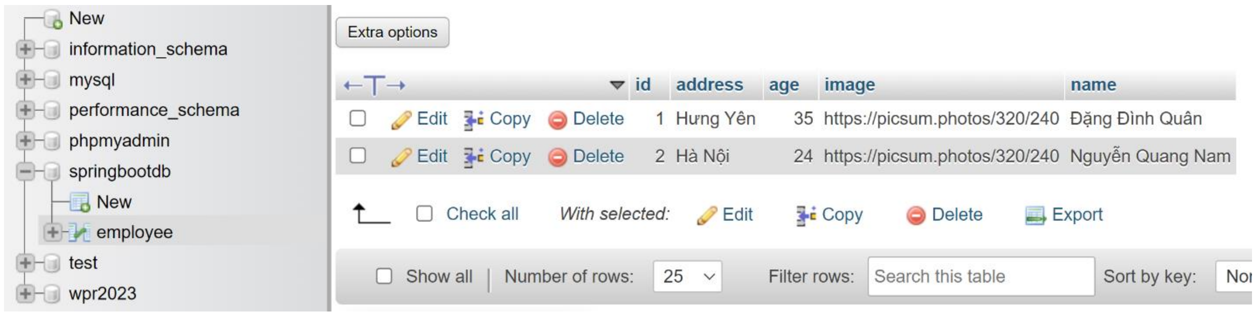
4. Run the application once by running the main application class to let Spring create the table in the database.



```
package fit.se2.springboot1;

import ...

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.cla
    }

}
```

5. Add more sample data into the created table:

   (You can store images in the `main/resources/static` folder)



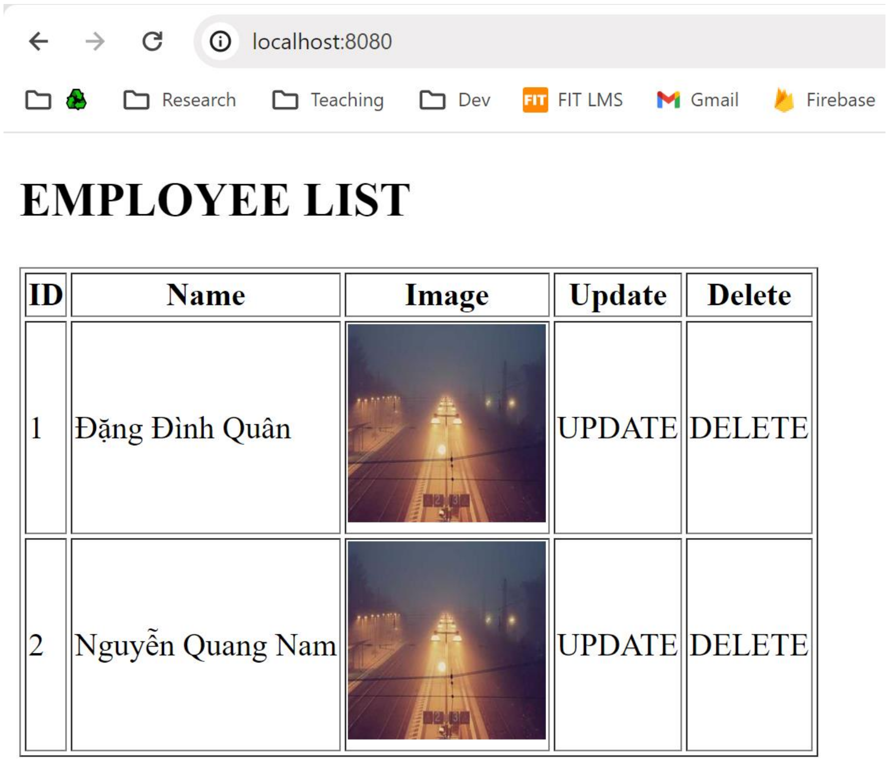
				id	address	age	image	name
☐	Edit	Copy	Delete	1	Hưng Yên	35	https://picsum.photos/320/240	Đặng Đình Quân
☐	Edit	Copy	Delete	2	Hà Nội	24	https://picsum.photos/320/240	Nguyễn Quang Nam

6. Create `employeeList.html` template under `src/main/resources/template` folder

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Employee List</title>
</head>
<body>
<div>
    <h2>EMPLOYEE LIST</h2>
    <table>
        <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="${employee.id}"/>
                <td th:text="${employee.name}">
                </td>
                <td>
                    <img th:src="${employee.image}" width="100" height="100"/>
                </td>
                <td>UPDATE</td>
                <td>DELETE</td>
            </tr>
        </table>
</div>
</body>
</html>
```

7. Run the web application again and visit http://localhost:8080/ on a browser (e.g. Google Chrome, Firefox…)

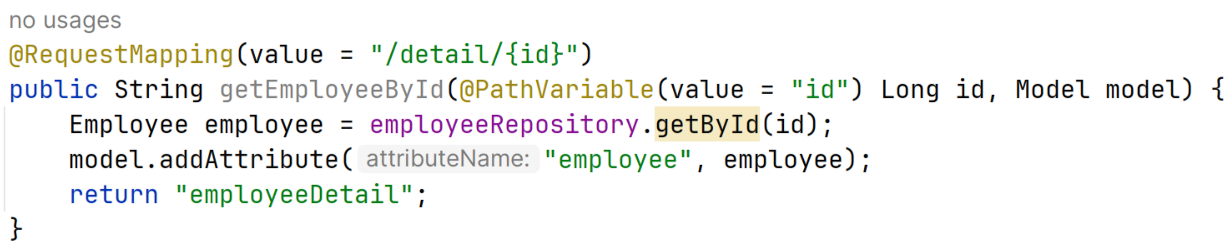


EMPLOYEE LIST

ID	Name	Image	Update	Delete
1	Đặng Đình Quân		UPDATE	DELETE
2	Nguyễn Quang Nam		UPDATE	DELETE

❖ Implement the Employee Detail page:

1. Create a new method in `EmployeeController` which maps to the `/detail/{id}` URL. Note that this URL contains a path parameter named `id`. Which is accessed using the `@PathVariable` annotation.



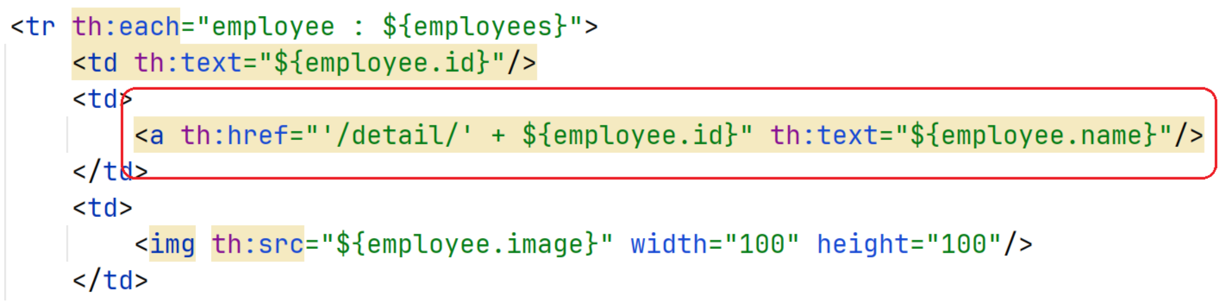
```
no usages
@RequestMapping(value = "/detail/{id}")
public String getEmployeeById(@PathVariable(value = "id") Long id, Model model) {
    Employee employee = employeeRepository.getById(id);
    model.addAttribute( attributeName: "employee", employee);
    return "employeeDetail";
}
```

2. Create a new Thymeleaf template for the Employee Detail page named `employeeDetail.html` under the folder `src/main/resources/templates`:

   (*) Pay attention to the relationship between the `employee` attribute that was added into the template from the controller method above and the variable with the same name that is available inside the template.

```
<div>
    <img th:src="${employee.image}" width="210" height="210">
</div>
<div>
    <h2 th:text="${employee.name}" />
```

3. Modify the `employeeList.html` template so that employee names become links to employee details:



```
<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>
```

❖ Submission:

- Submit a `zip` file which contains your full source code project along with a Word (`docx`) document containing the following screenshots:
    1. *Your finished project structure with all the relevant packages expanded.*
    2. *The console inside IntelliJ IDEA when you run the Spring application.*
    3. *The Employee List page displayed on a browser.*
    4. *The Employee Detail page displayed on a browser.*