TUTORIAL 7 – DEVELOP JAVA WEB WITH SPRING BOOT (1)

❖ Part 1:

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

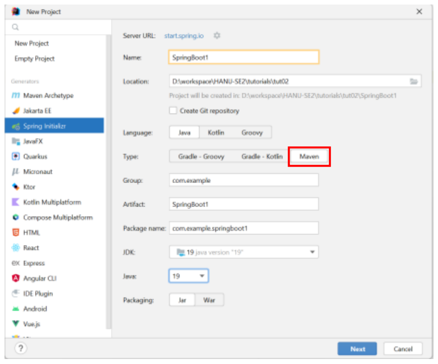
❖ Introduction:

- 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 Initializr

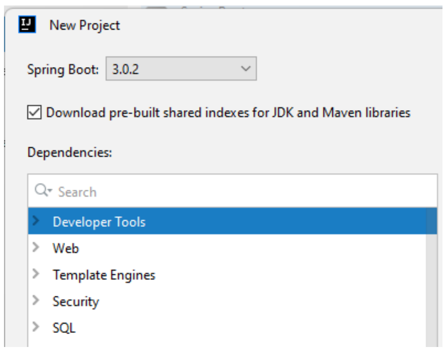
- **New Project**
- Select **Spring Initializr**
- Input project parameters:
  - Project name
  - Project location
  - Language: **Java**
  - Type: **Maven**
  - JDK: **19**
  - Java: **19**
  - Packaging: **Jar**



Server URL: start.spring.io
Name: SpringBoot1
Location: D:\workspace\HANU-SE2\tutorials\tut02
Project will be created in: D:\workspace\HANU-SE2\tutorials\tut02\SpringBoot1
Create Git repository
Language: Java Kotlin Groovy
Type: Gradle - Groovy Gradle - Kotlin Maven
Group: com.example
Artifact: SpringBoot1
Package name: com.example.springboot1
JDK: 19 java version "19"
Java: 19
Packaging: Jar War

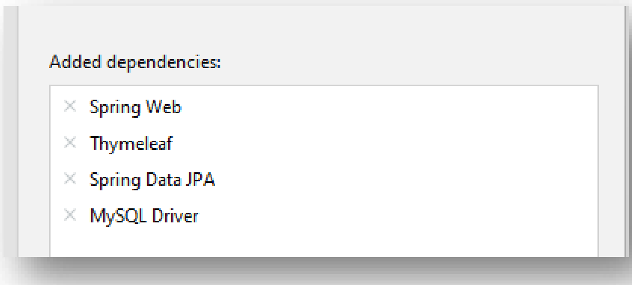
Figure 1 - Create new Spring Boot project (1)

- Click **Next**
- Spring Boot version: **3.0.2**
- Select dependencies:
  - Spring Web
  - Thymeleaf
  - Spring Data JPA
  - MySQL Driver
- Click **Finish**



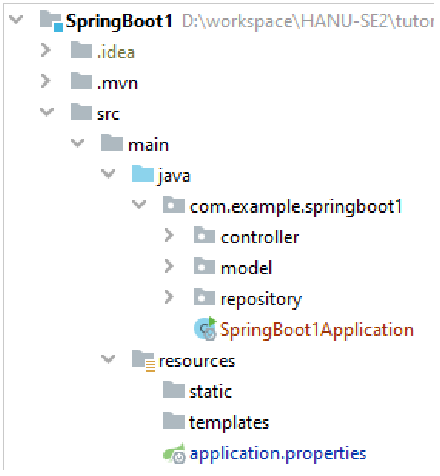
Spring Boot: 3.0.2
☑ Download pre-built shared indexes for JDK and Maven libraries
Dependencies:

Figure 2 - Create new Spring Boot project (2)



Added dependencies:
Spring Web
Thymeleaf
Spring Data JPA
MySQL Driver

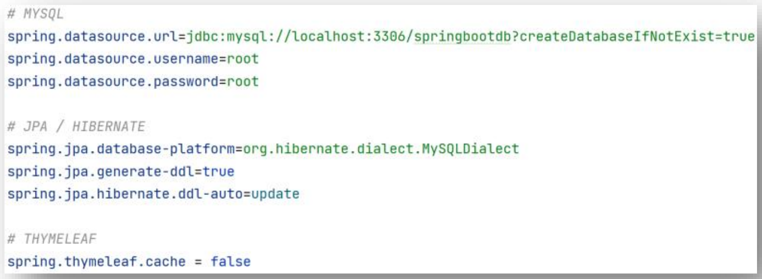
Figure 3 - Create new Spring Boot project (3)



```
SpringBoot1 D:\workspace\HANU-SE2\tutor
  .idea
  .mvn
  src
    main
      java
        com.example.springboot1
          controller
          model
          repository
          SpringBoot1Application
      resources
        static
        templates
        application.properties
```

Figure 4 - Sample project structure

2. Config parameters for MySQL connection, JPA & Hibernate (*located at* ***src/main/resources*** *folder*)



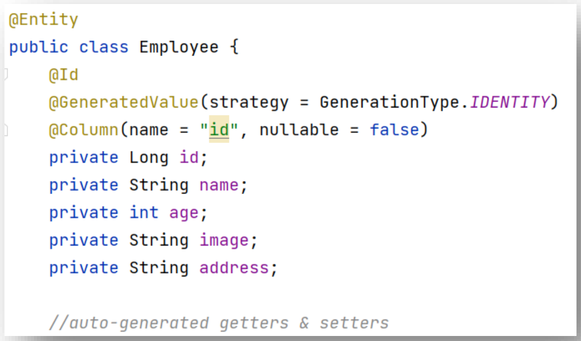
```
# MYSQL
spring.datasource.url=jdbc:mysql://localhost:3306/springbootdb?createDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=root

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

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

Figure 5 - ***application.properties***

3. Create Java class for model (entity) which acts as table in database (*located at sub-package* ***model*** *in* ***src/main/java*** *folder*). Don't forget to create ***get*** and ***set*** function

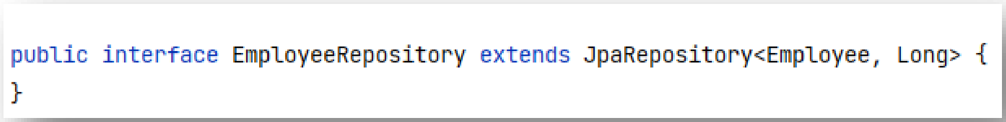


```
@Entity
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id", nullable = false)
    private Long id;
    private String name;
    private int age;
    private String image;
    private String address;

    //auto-generated getters & setters
```

Figure 6 - ***Employee.java***

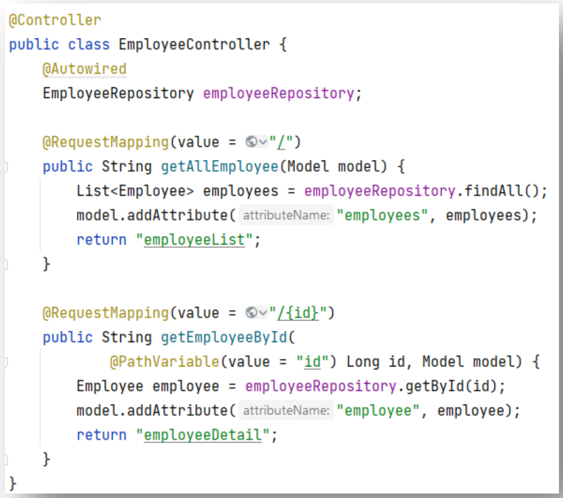
4. Create Java interface which extends *JpaRepository* for CRUD features (*located at sub-package* ***repository*** *in* ***src/main/java*** *folder*)



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

Figure 7 - ***EmployeeRepository.java***

5. Create Java class for controller which gets data from database and renders view (*located at sub-package* ***controller*** *in* ***src/main/java*** *folder*)



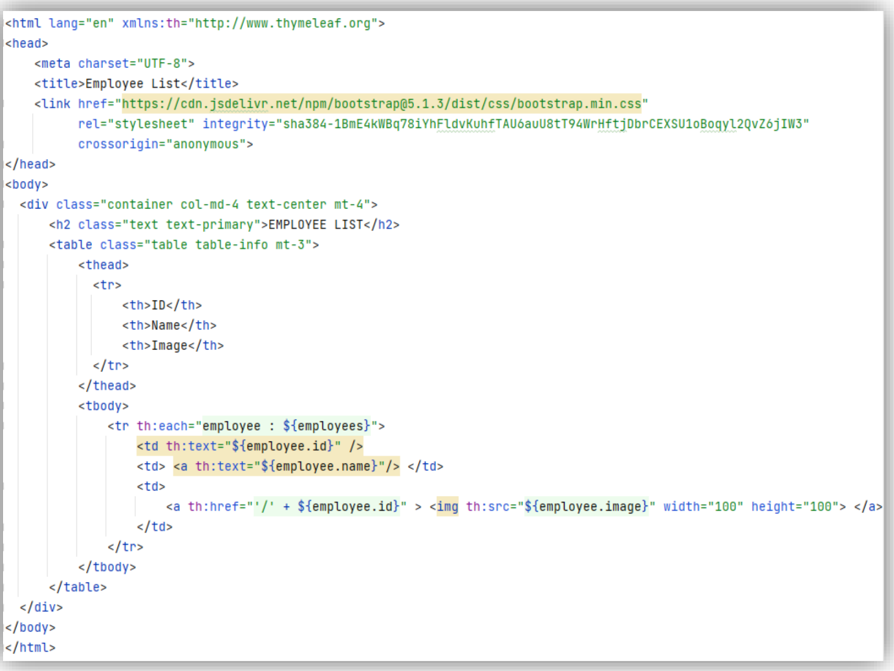
```java
@Controller
public class EmployeeController {
    @Autowired
    EmployeeRepository employeeRepository;

    @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";
    }
}
```

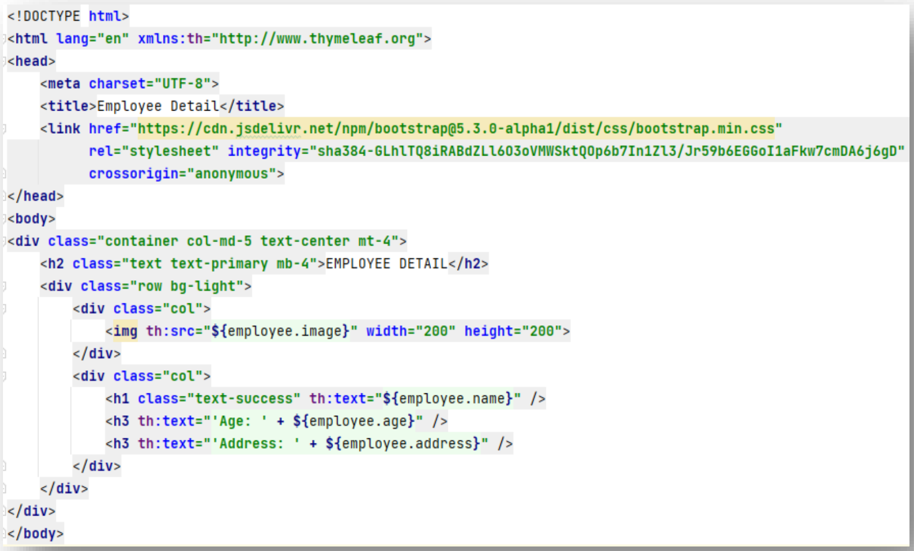
Figure 8 - ***EmployeeController.java***

6. Create HTML pages as view (*located at* ***src/main/resources/templates*** *folder*)



```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 9 - **employeeList.html***



```html
<!DOCTYPE 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@5.3.0-alpha1/dist/css/bootstrap.min.css"
          rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/Jr59b6EGGoI1aFkw7cmDA6j6gD"
          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>
```

*Figure 10 - **employeeDetail.html***

7. Add sample data (records) to that table in database using MySQL Workbench (*refer to Tutorial 6*) or integrated MySQL database in IntelliJ

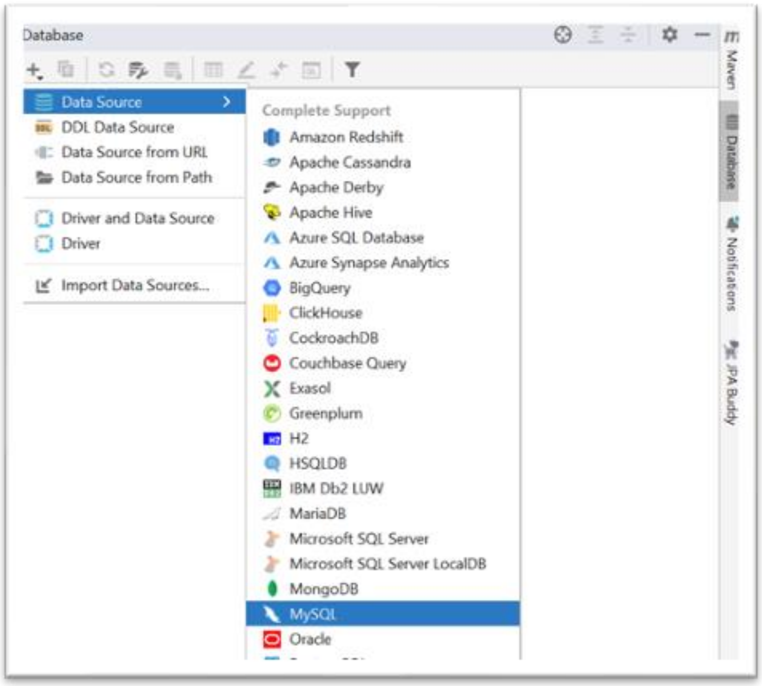


Figure 11 – Setup connection to integrated MySQL database in IntelliJ (1)

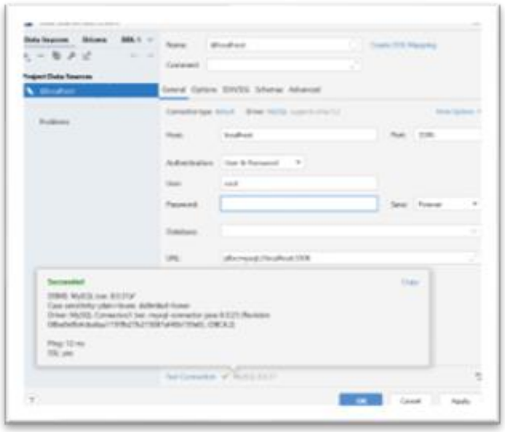
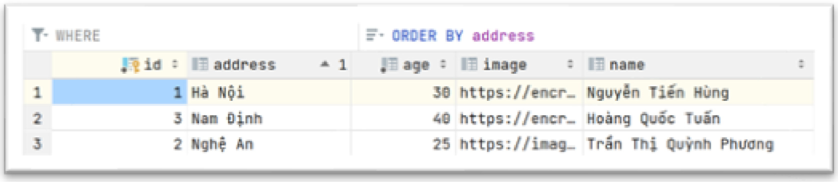


Figure 12 – Setup connection to integrated MySQL database in IntelliJ (2)

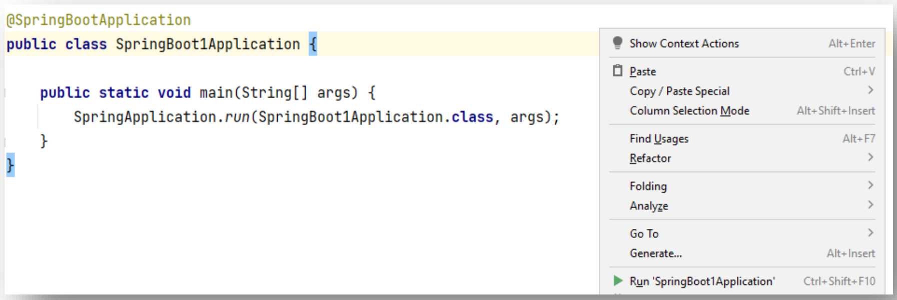


WHERE   ORDER BY address

	id	address	age	image	name
1	1	Hà Nội	30	https://encr…	Nguyễn Tiến Hùng
2	3	Nam Định	40	https://encr…	Hoàng Quốc Tuấn
3	2	Nghệ An	25	https://imag…	Trần Thị Quỳnh Phương

Figure 13 - Add sample data to table

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



```java
@SpringBootApplication
public class SpringBoot1Application {

    public static void main(String[] args) {
        SpringApplication.run(SpringBoot1Application.class, args);
    }
}
```

Show Context Actions Alt+Enter
Paste Ctrl+V
Copy / Paste Special
Column Selection Mode Alt+Shift+Insert
Find Usages Alt+F7
Refactor
Folding
Analyze
Go To
Generate... Alt+Insert
Run 'SpringBoot1Application' Ctrl+Shift+F10

*Figure 14 – **SpringBoot1Application.java***

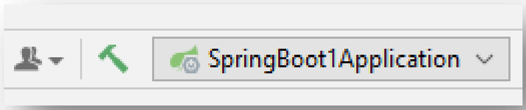
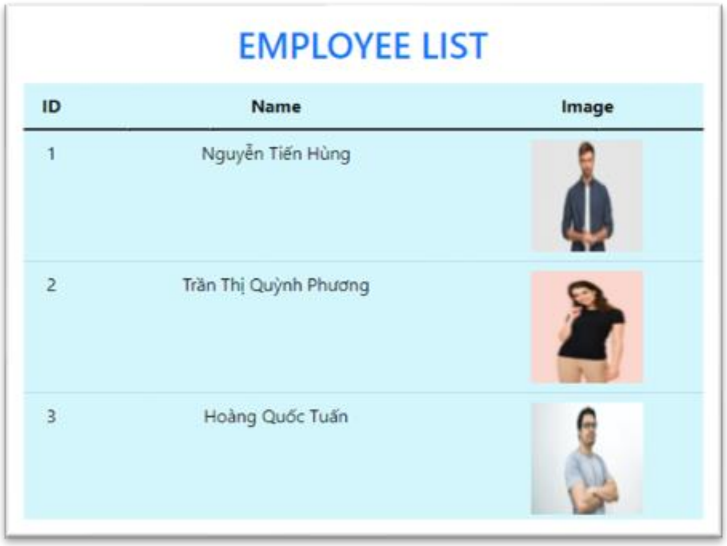


Figure 15 - Run web application

9. Open a web browser (ex: Chrome) and type address: http://localhost:8080/ to access the web application



EMPLOYEE LIST

ID	Name	Image
1	Nguyễn Tiến Hùng	
2	Trần Thị Quỳnh Phương	
3	Hoàng Quốc Tuấn	

Figure 16 - Employee List page

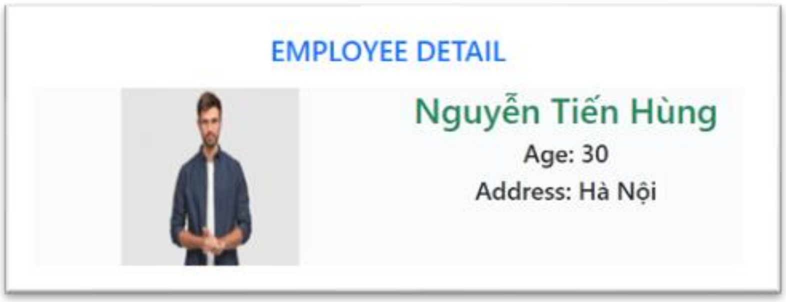


Figure 17 - Employee Detail page

❖ 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 LMS with file name syntax: *FullName_StudentID_SE1_Tut7.zip*

❖ Part 2:

- **Practice use case diagram and use case template**

❖ Exercise:

- A trader buys many stocks for different prices and in different quantities. The Stock Trader program keeps track of a trader's investments. This application require user to login before using. A trader can use the program to buy stocks or sell his stocks. The program is required to store information about all stocks owned by a user, including the amount owned and other information that the user may want to record, such as the date and price when purchased. In addition, the program needs to be able to find out the current price of any stock in the portfolio and to compute the current value of the user’s investments.

❖ Task:

- Brainstorm about what features would a user need from the program and how those features are provided to the user. Write the solution in a text document. Also, draw a use case diagram to describe the features of the program.
- Using use case template to discribe these features of the program.
- Submit a Word document then submit to FIT LMS with file name syntax: *FullName_StudentID_SE1_Tut7.docx*