

GIỚI THIỆU VỀ SPRING BOOT

GV: Nguyễn Huy Cường
Email: nh.cuong@hutech.edu.vn

04/2024

Nội dung

1. Spring boot là gì?
2. Tạo project Spring Boot
3. Spring initializr
4. Cấu trúc project spring boot
5. Viết ứng dụng đơn giản
6. Annotations trong Spring Boot

1. Spring boot là gì?

- Spring Boot là một module trong hệ sinh thái Spring framework, cung cấp tính năng để phát triển ứng dụng nhanh (RAD)



- Các chức năng chính Spring boot:
- Spring boot starter: quản lý các dependencies, tự động config
 - Spring boot auto configuration
 - External configuration: cấu hình từ bên ngoài
 - Spring boot actuator: cho phép theo dõi, giám sát ứng dụng, thu thập số liệu, lưu lượng truy cập hay trạng thái cơ sở dữ liệu
 - ...

Vấn đề của Spring là quá nhiều cấu hình

Vấn đề của Spring MVC

- Vấn đề của Spring là quá nhiều cấu hình
- ❑ Khi sử dụng Spring MVC, chúng ta cần cấu hình:
 - Component scan
 - Dispatcher servlet
 - View resolver
 - Web jars
- ❑ Khi sử dụng Hibernate/ JPA chúng ta cần phải cấu hình
 - data source
 - entity manager factory / session factory
 - transaction
- ❑ Khi sử dụng cache
 - cấu hình Cache

...

2. Cách tạo project spring boot

Có 3 cách tạo project spring boot

1. Tạo từ Spring initializer từ <https://start.spring.io/>

2. Sử dụng Spring Starter Project

- ❑ Eclipse: Đã Cài thêm STS (Spring tool suite)

- ❑ Spring initializer plugin ở IntelliJ

- ❑ Maven / initializer sau khi Nb SpringBoot

3. Spring Boot CLI



3. Spring initializr

- Truy cập vào Spring Initializr: <https://start.spring.io/>



Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ **Maven**

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.3.0 (SNAPSHOT) ☐ 3.3.0 (RC1) ☐ 3.2.6 (SNAPSHOT) ☒ 3.2.5

☐ 3.1.12 (SNAPSHOT) ☐ 3.1.11

Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☒ 22 ☐ 21 ☐ 17

Dependencies ADD DEPENDENCIES... CTRL + B

No dependency selected

GENERATE CTRL + G EXPLORE CTRL + SPACE SHARE...

- Cần khai báo thông tin project và chọn dependencies

Spring initializr

1- Thông tin project

☐ Loại project: **Maven/ Gradle.**

☐ Language: Java

☐ Phiên bản Spring Boot:

SNAPSHOT là bản chưa ổn định

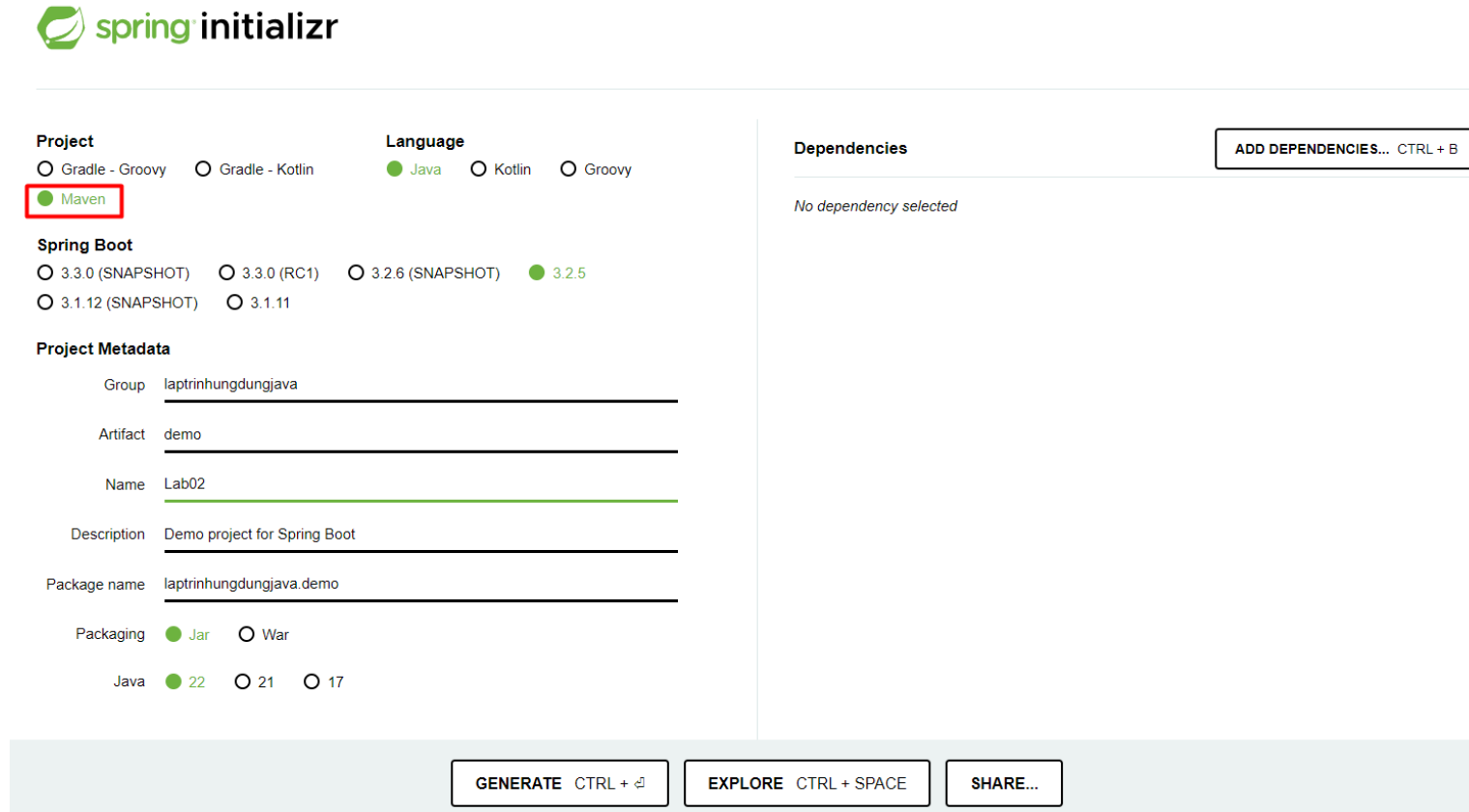
☐ Loại file build ra:

JAR: dành cho ứng dụng java

War: ứng dụng web

☐ Phiên bản Java: Từ java 17

Ngoài ra cần khai báo thêm các metadata như
tên project, tên package, artifact,...



The screenshot shows the Spring Initializr web interface. The 'Project' section has radio buttons for 'Gradle - Groovy', 'Gradle - Kotlin', and 'Maven' (which is selected and highlighted with a red box). The 'Language' section has radio buttons for 'Java' (selected), 'Kotlin', and 'Groovy'. The 'Spring Boot' section has radio buttons for various versions: '3.3.0 (SNAPSHOT)', '3.3.0 (RC1)', '3.2.6 (SNAPSHOT)', '3.2.5' (selected), '3.1.12 (SNAPSHOT)', and '3.1.11'. The 'Project Metadata' section includes input fields for 'Group' (laptopinhungdungjava), 'Artifact' (demo), 'Name' (Lab02), 'Description' (Demo project for Spring Boot), and 'Package name' (laptopinhungdungjava.demo). The 'Packaging' section has radio buttons for 'Jar' (selected) and 'War'. The 'Java' section has radio buttons for '22' (selected), '21', and '17'. The 'Dependencies' section on the right has a button 'ADD DEPENDENCIES... CTRL + B' and the text 'No dependency selected'. At the bottom, there are three buttons: 'GENERATE CTRL + G', 'EXPLORE CTRL + SPACE', and 'SHARE...'.

Spring initializr

2- Chọn dependency

- Dependencies: là các thư viện phụ trợ
- ☐ Spring web.
- ☐ Spring Boot DevTools
- ☐ Thymeleaf
- ☐ Lombok
- ☐ ...

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 Boot DevTools **DEVELOPER TOOLS**

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

—


Thymeleaf **TEMPLATE ENGINES**

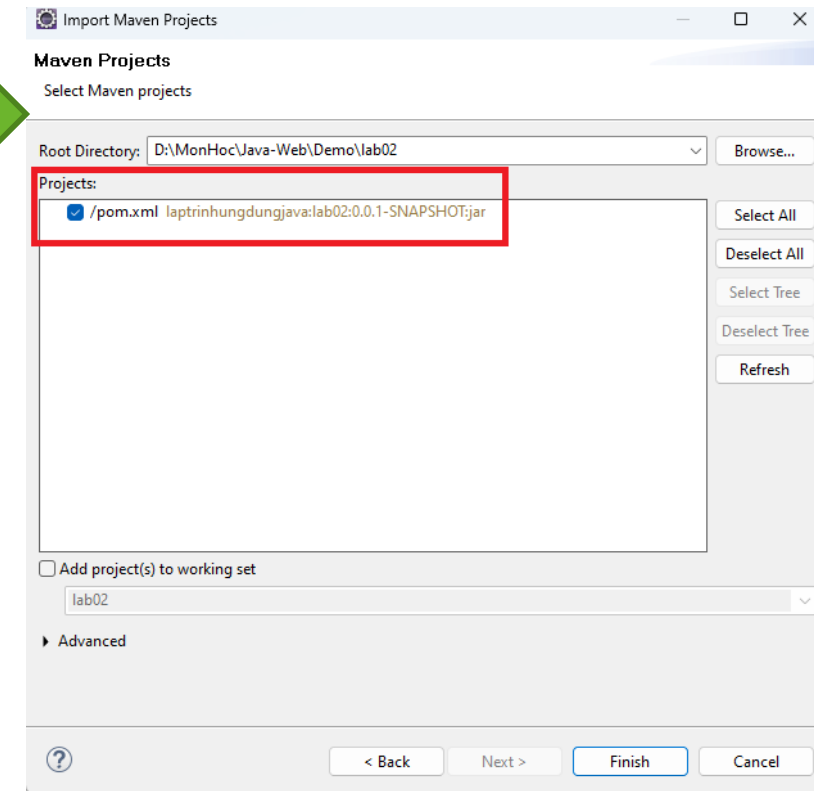
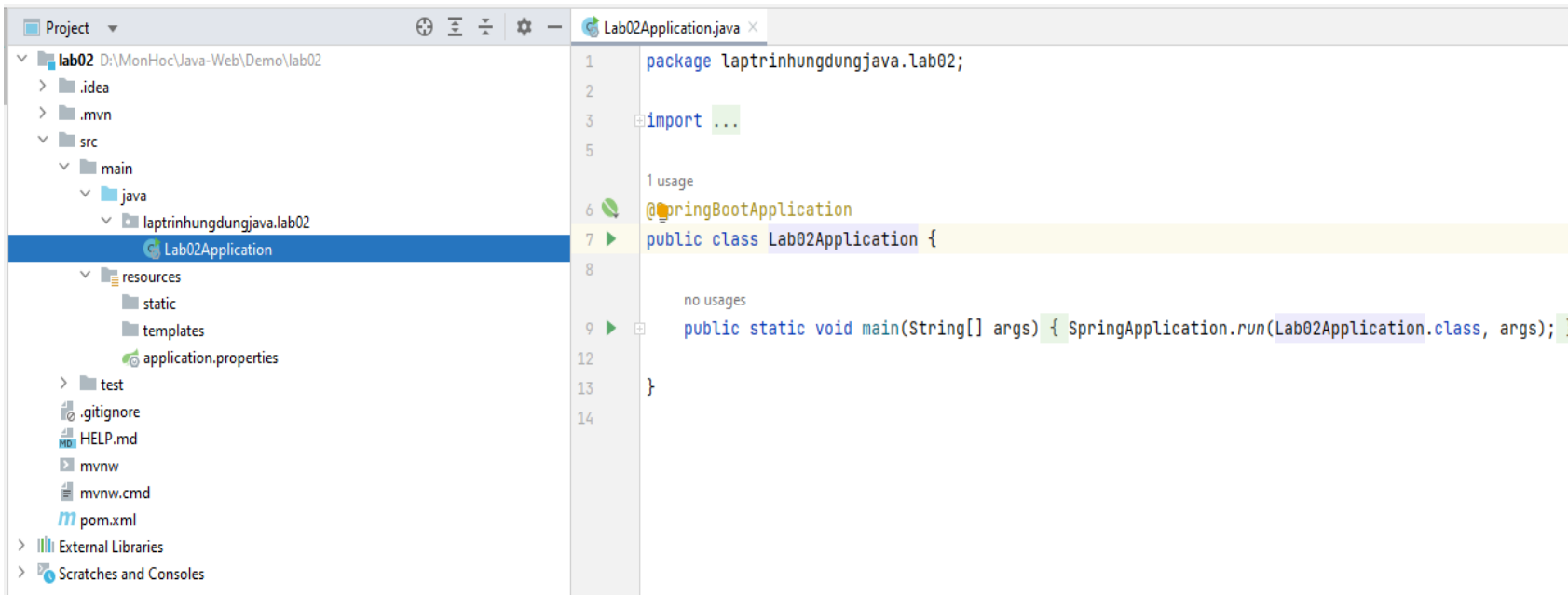
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

—

Spring initializr

3- Giải nén và mở project

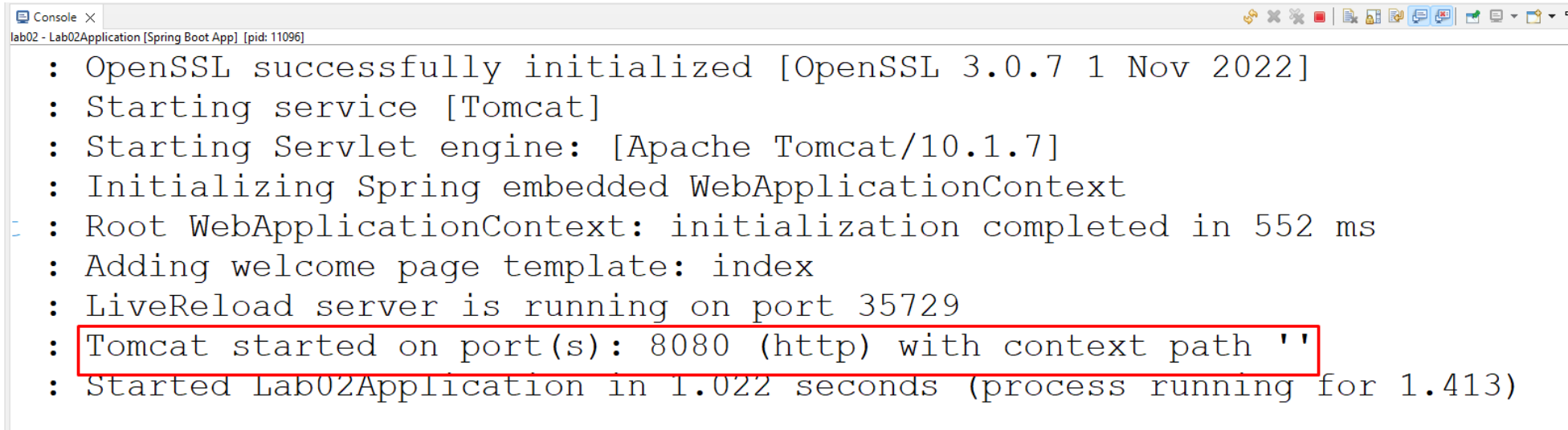
- Sau khi Generate sẽ tải xuống file nén của project, unzip và sử dụng IDE để mở ứng dụng
- ❑ **Eclipse:** Click **File>>Import>>Existing Maven Projects.** 
- ❑ **IntelliJ:** Open tới folder đã giải nén
- Sau khi mở project



Spring initializr

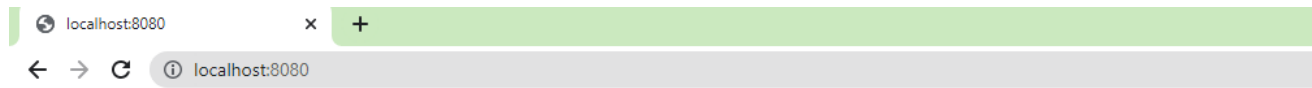
4- Chạy project spring boot đầu tiên

- Run Spring boot: thành công sử dụng cổng 8080



```
lab02 - Lab02Application [Spring Boot App] [pid: 11096]
: OpenSSL successfully initialized [OpenSSL 3.0.7 1 Nov 2022]
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/10.1.7]
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 552 ms
: Adding welcome page template: index
: LiveReload server is running on port 35729
: Tomcat started on port(s): 8080 (http) with context path ''
: Started Lab02Application in 1.022 seconds (process running for 1.413)
```

- Truy cập vào trình duyệt để xem kết quả



Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

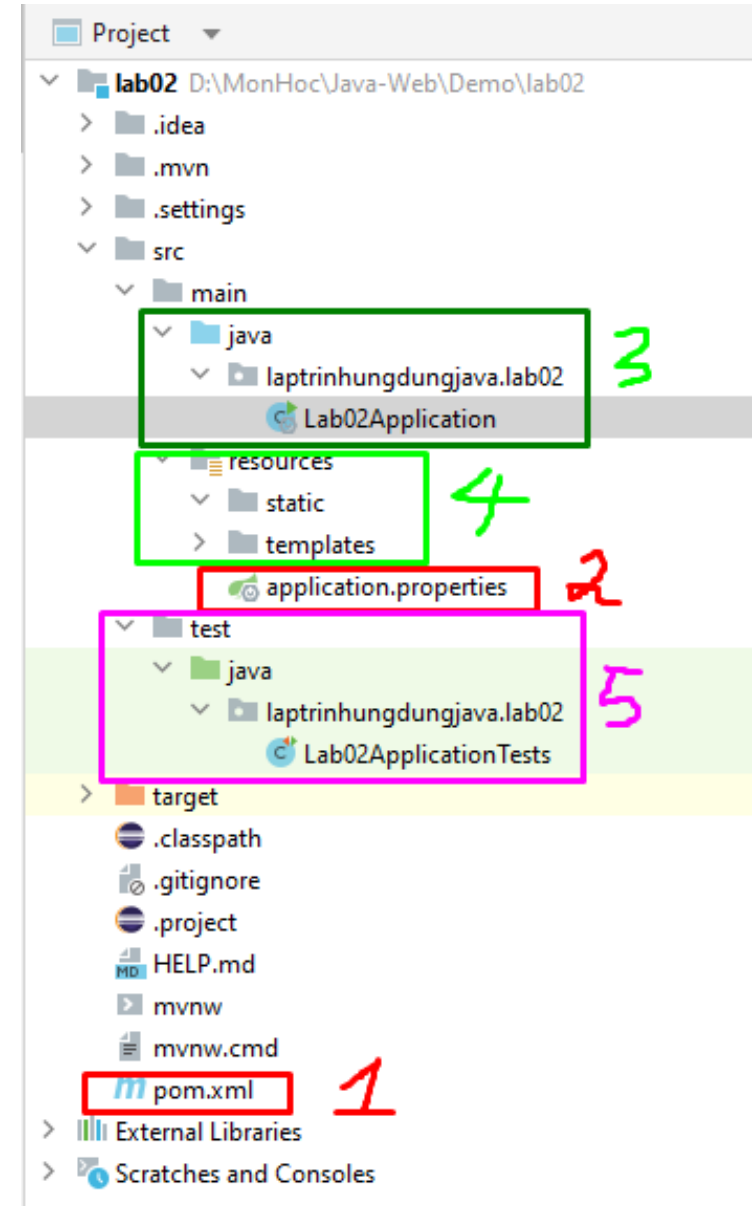
Mon Apr 24 13:37:43 ICT 2023

There was an unexpected error (type=Not Found, status=404).

No message available

4. Cấu trúc project

- ***pom.xml***
- ***Application.properties***
- ***src/main/java***: Source code của ứng dụng
- ***src/resources***
- ***test***
- ***target***



Cấu trúc project Spring Boot

1- pom.xml

- **pom.xml** là Developer Files - Maven Build File: khai báo tất cả những gì liên quan đến project:

- ❑ Version của project, tên dự án, repository...
- ❑ Các dependency: Các thư viện sử dụng được khai báo trong `<dependencies>` / `</dependencies>`

```

*lab02/pom.xml X
https://maven.apache.org/xsd/maven-4.0.0.xsd (xsi:schemaLocation)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>3.0.5</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>laptrinhungdungjava</groupId>
12  <artifactId>lab02</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>lab02</name>
15  <description>Demo project for Spring Boot</description>
16  <properties>
17    <java.version>20</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-starter-thymeleaf</artifactId>
23    </dependency>
24    <dependency>
25      <groupId>org.springframework.boot</groupId>
26      <artifactId>spring-boot-starter-web</artifactId>
27    </dependency>
28
29    <dependency>
30      <groupId>org.springframework.boot</groupId>
31      <artifactId>spring-boot-devtools</artifactId>
32      <scope>runtime</scope>
33      <optional>true</optional>
34    </dependency>
35    <dependency>
36      <groupId>org.springframework.boot</groupId>
37      <artifactId>spring-boot-starter-test</artifactId>
38      <scope>test</scope>
39    </dependency>
40  </dependencies>
41
42  <build>
43    <plugins>
44      <plugin>
45        <groupId>org.springframework.boot</groupId>
46        <artifactId>spring-boot-maven-plugin</artifactId>
47      </plugin>
48    </plugins>
49  </build>
50 </project>

```

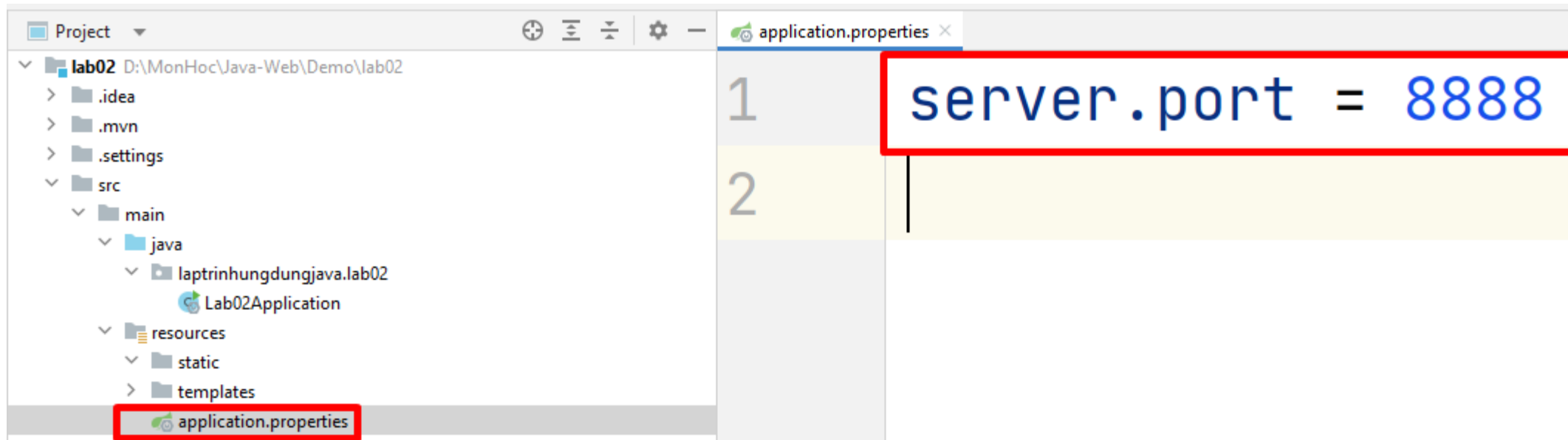
Cấu trúc project Spring Boot

2- Application.properties

- **Application.properties:** Các thông tin cấu hình mặc định

Ví dụ: Để Spring Boot chạy trên port 8888
(thay vì 8080)

`server.port = 8888`



Cấu trúc project Spring Boot

3-src/main/java

- Chứa source code của ứng dụng
- Tổ chức source code của spring boot

❑ Controller:

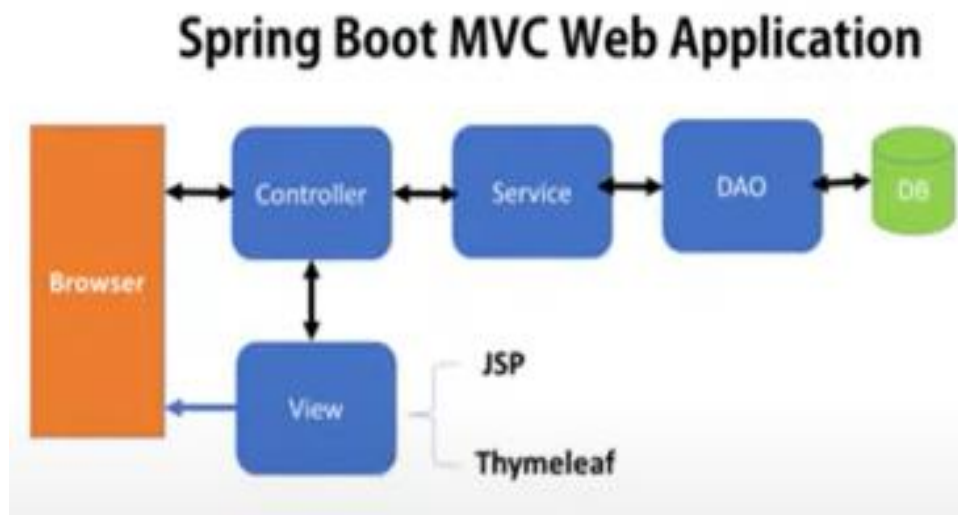
- Các class là controller sẽ có hậu tố Controller (ví dụ UserController, AuthController,...)

❑ Service

- Các class có hậu tố là Service và thường tương ứng với controller. Ví dụ: UserService

❑ DAO (Data Access Layer)

❑ View



Cấu trúc project Spring Boot

4- src/main/resource

- Chứa các tài nguyên của ứng dụng: static/ dynamic

❑ **static:** chứa các file tĩnh như CSS, js và hình ảnh.

❑ **templates:**

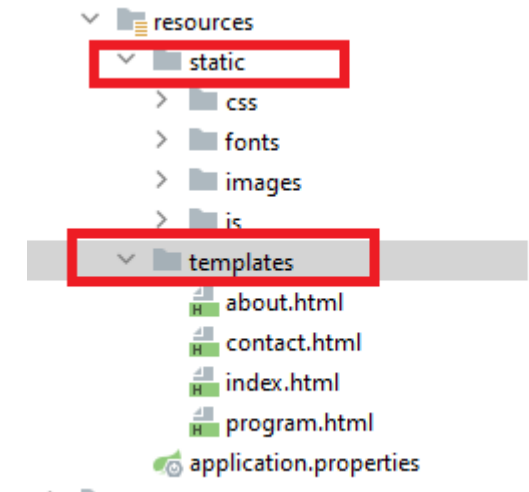
chứa các server-side templates được render từ Spring

Thymeleaf

JSP: Java Server Pages

FreeMarker

Jade4j



Cấu trúc project Spring Boot

5-src/test

- Chứa lớp kiểm thử của ứng dụng với JUnit và Spring

The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure is as follows:

- lab02
 - .idea
 - .mvn
 - .settings
 - src
 - main
 - java
 - laptrinhungdungjava.lab02
 - controller
 - HomeController
 - Lab02Application
 - resources
 - test

The code editor shows the following code in `Lab02ApplicationTests.java`:

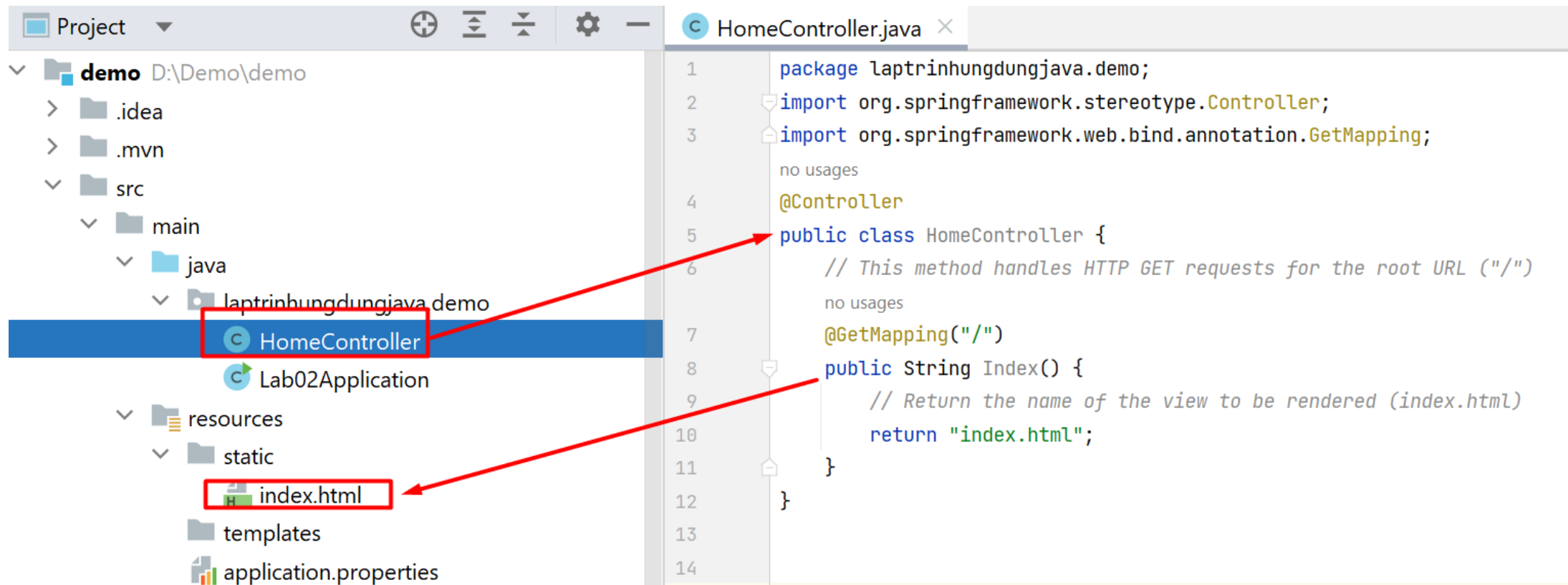
```
1 package laptrinhungdungjava.lab02;
2
3 import org.junit.jupiter.api.Test;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.boot.test.context.SpringBootTest;
6 import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
7 import org.springframework.boot.test.web.client.TestRestTemplate;
8 import org.springframework.beans.factory.annotation.Value;
9
10 import static org.assertj.core.api.Assertions.assertThat;
11
12 @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
13 class Lab02ApplicationTests {
14
15     @Value(value="${local.server.port}")
16     private int port;
17
18     @Autowired
19     private TestRestTemplate restTemplate;
20
21     @Test
22     public void greetingShouldReturnDefaultMessage() throws Exception {
23         assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/", String.class)).contains("Hello World");
24     }
25 }
```


5- Viết ứng dụng đơn giản

B1: Tạo **controller** đơn giản: Sử dụng các *Annotation*: **@Controller**, **@GetMapping**

- **Controller** là thành phần đầu tiên để bắt URL truy cập.

Ví dụ: bạn vào trang chủ của web, thì controller method có mapping tới URL / sẽ được gọi.

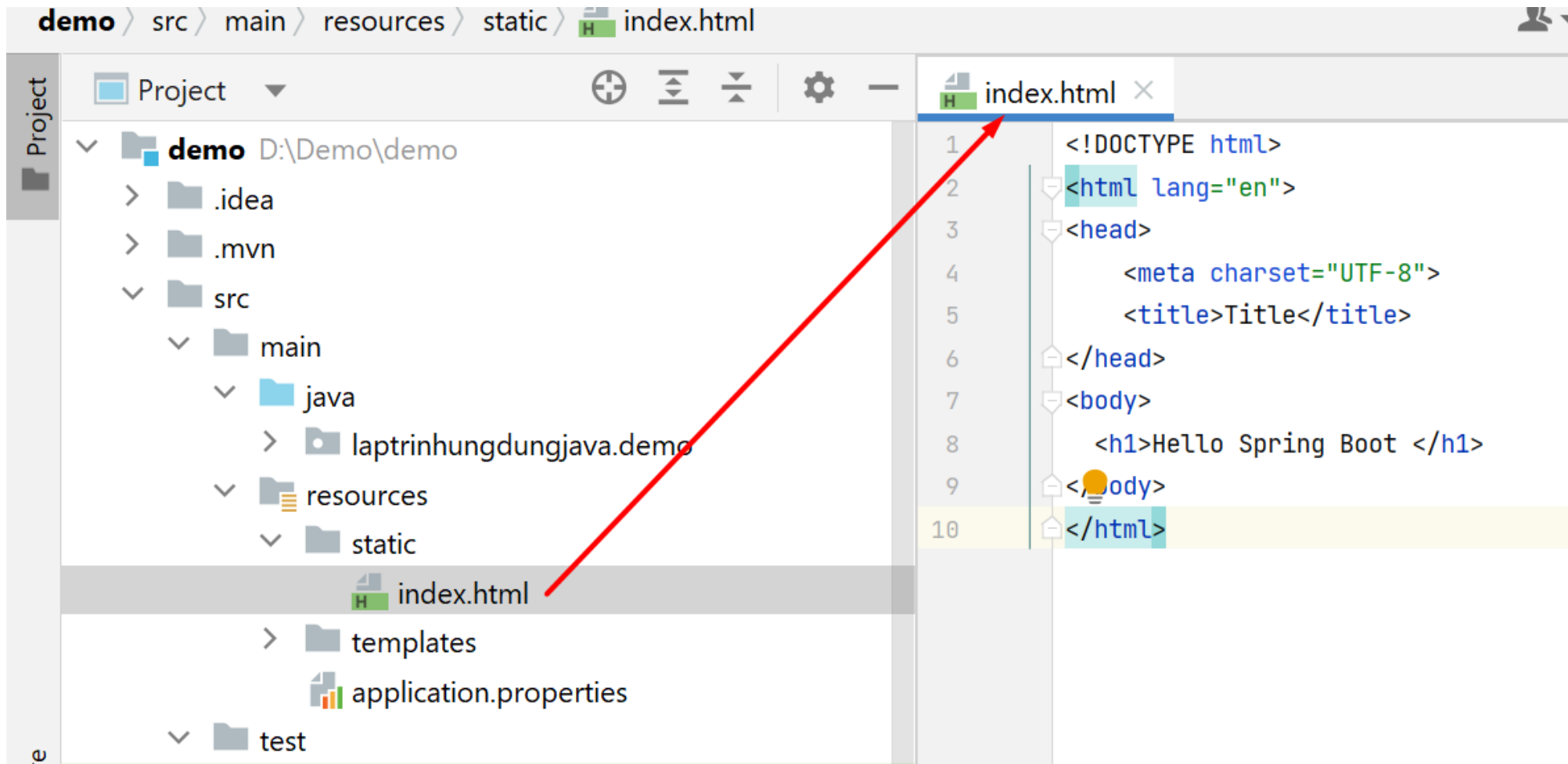


```
1 package laptrinhungdungjava.demo;
2 import org.springframework.stereotype.Controller;
3 import org.springframework.web.bind.annotation.GetMapping;
4 @Controller
5 public class HomeController {
6     // This method handles HTTP GET requests for the root URL ("/")
7     @GetMapping("/")
8     public String Index() {
9         // Return the name of the view to be rendered (index.html)
10        return "index.html";
11    }
12 }
```

Viết ứng dụng đơn giản

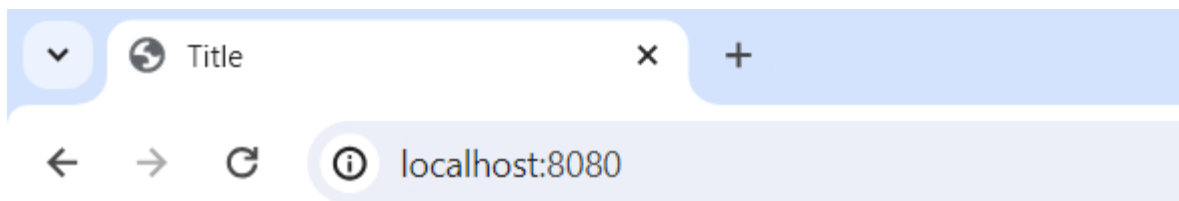
B2: Tạo trang HTML để trả về

- Right click resources/**static**, chọn New > HTML file và gõ tên **index.html**



Kết quả chương trình

- Khi truy cập vào địa chỉ localhost:8080



Hello Spring Boot

Sử dụng template engine

- Sử dụng Thymeleaf bằng cách bổ sung vào file cấu hình

- ☐ Maven

```
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-thymeleaf</artifactId>  
</dependency>
```

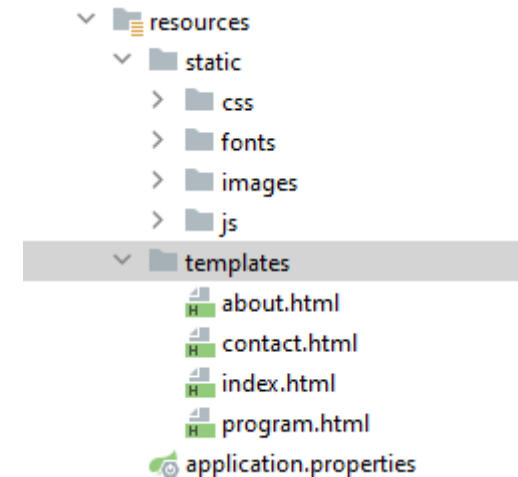
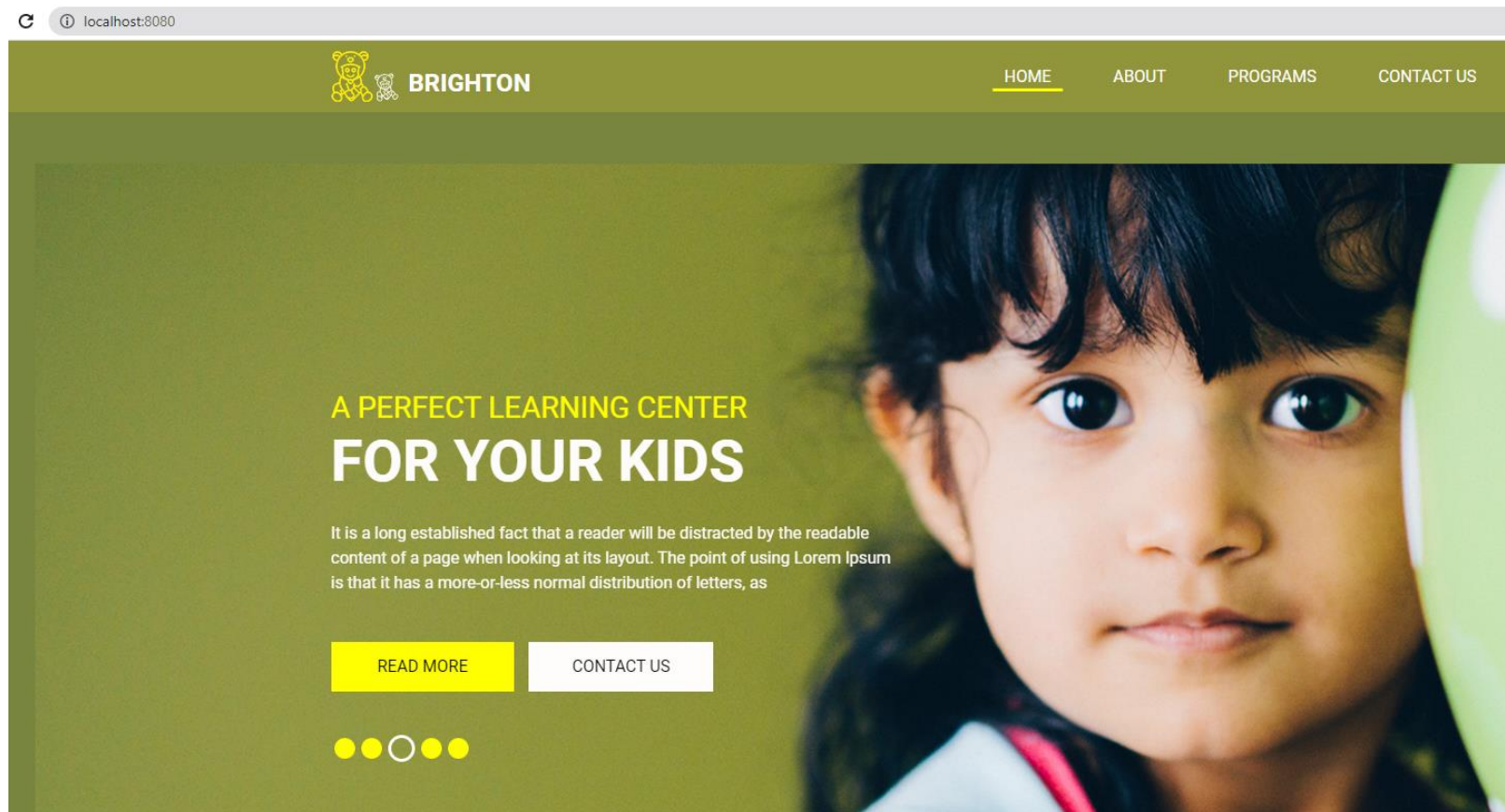
- Các file html được lấy từ folder “templates”



Sử dụng template kết hợp project spring boot

- Sử dụng template và đưa vào project spring boot

<https://www.free-css.com/free-css-templates/page290/brighton>



ANNOTATIONS TRONG SPRING BOOT



6. annotations là gì ?

- **Annotation:** chú thích, được sử dụng để cung cấp *thông tin dữ liệu* cho code Java. Thường đi kèm với **class**, **interface**, thuộc tính hoặc phương thức dữ liệu.
 - ❑ Annotation được bắt đầu với @
 - ❑ Annotation không phải là chú thích thuần túy vì chúng có thể thay đổi cách trình biên dịch xử lý chương trình
 - ❑ Trước khi có chú thích, hành vi của Spring Framework được kiểm soát phần lớn thông qua cấu hình XML.
- Một số annotations có sẵn trong java:
 - @Deprecated:** chỉ ra class/ method đã bị lỗi thời và không nên sử dụng nữa
 - @Override:** sử dụng cho các phương thức ghi đè của phương thức trong class cha.
 - @SuppressWarnings:** để hướng dẫn trình biên dịch bỏ qua những cảnh báo cụ thể

annotations trong spring boot

Spring Boot and Web Annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses **@Configuration**, **@EnableAutoConfiguration** and **@ComponentScan**.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests.

T **@RestController** - a convenience annotation of a **@Controller** and **@ResponseBody**.

M **T** **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.

Spring Framework Annotations

Spring uses dependency injection to configure and bind your application together.

T **@Configuration** - used to mark a class as a source of the bean definitions.

T **@ComponentScan** - makes Spring scan the packages configured with it for the **@Configuration** classes.

T **@Import** - loads additional configuration. This one works even when you specify the beans in an XML file.

T **@Component** - turns the class into a Spring bean at the auto-scan time.

T **@Service** - tells Spring that it's safe to manage **@Components** with more freedom than regular components.

C **F** **M** **@Autowired** - wires the application parts together, on the fields, constructors, or methods in a component.

M **@Bean** - specifies a returned bean to be managed by Spring context. The returned bean has the same name as the factory method.

M **@Lookup** - tells Spring to return an instance of the method's return type when we invoke it.

T **M** **@Primary** - gives higher preference to a bean when there are multiple beans of the same type.

C **F** **M** **@Required** - shows that the setter method must be configured to be dependency-injected with a value at configuration time.

C **F** **M** **@Value** - used to assign values into fields in Spring-managed beans. It's compatible with the constructor, setter, and field injection.

T **M** **@DependsOn** - makes Spring initialize other beans before the annotated one.

T **M** **@Lazy** - makes beans to initialize lazily. **@Lazy** annotation may be used on any class directly or indirectly annotated with **@Component** or on methods annotated with **@Bean**.

T **M** **@Scope** - used to define the scope of a **@Component** class or a **@Bean** definition and can be either singleton, prototype, request, session, globalSession, or custom scope.

T **@Profile** - adds beans to the application only when that profile is active.

Spring Boot and Web Annotations

- **@SpringBootApplication**: Nó là sự kết hợp của 3 chú thích: @EnableAutoConfiguration, @ComponentScan và @Configuration.
- **@RestController**: là một composed annotation được kết hợp từ **@Controller** và **@ResponseBody**
khi đặt **@RestController** trên một class controller thì mọi method controller trong class đó sẽ được thừa hưởng annotation **@ResponseBody** và response data trong controller này sẽ được trả về dưới dạng message.

Spring Boot and Web Annotations

Use annotations to configure your web application.

T **@SpringBootApplication** - uses @Configuration, @EnableAutoConfiguration and @ComponentScan.

T **@EnableAutoConfiguration** - make Spring guess the configuration based on the classpath.

T **@Controller** - marks the class as web controller, capable of handling the requests.

T **@RestController** - a convenience annotation of a @Controller and @ResponseBody.

M T **@ResponseBody** - makes Spring bind method's return value to the web response body.

M **@RequestMapping** - specify on the method in the controller, to map a HTTP request to the URL to this method.

P **@RequestParam** - bind HTTP parameters into method arguments.

P **@PathVariable** - binds placeholder from the URI to the method parameter.



Tạo Rest BookController

1. Tạo project book từ spring initializr
 2. Thêm file java class: BookController
 - Đánh dấu class: @RestController
 - Viết phương thức Welcome
- Trả về chuỗi, phương thức GET

The image shows the Spring Initializr web interface. The 'Project' section has 'Maven' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '3.0.6' selected. The 'Project Metadata' section shows the following details:

- Group: laptrinhungdungjava.bai2
- Artifact: book
- Name: book
- Description: Demo project for Spring Boot
- Package name: laptrinhungdungjava.bai2.book
- Packaging: Jar
- Java: 20

The 'Dependencies' section is empty. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and 'SHARE'.

The image shows an IDE window with the following tabs: BookApplication.java, BookController.java, and generated-requests.http. The 'Project' view on the left shows the project structure:

- book
 - .idea
 - .mvn
 - src
 - main
 - java
 - laptrinhungdungjava.bai2.book
 - BookApplication
 - BookController
 - resources
 - test
 - target

The 'BookController.java' file is open, showing the following code:

```

2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 no usages
7 @RestController
8
9 no usages
10 public class BookController {
11     @GetMapping(value="/")
12     public String Welcome() {
13         return "Welcome to Book store";
14     }
15 }

```



Tạo BookController là Rest API

3. Thêm lớp Book (id, title, author, price)

Viết các Rest API cho các phương thức: get, post, put, delete

```
@RestController
public class BookController {

    6 usages
    private List<Book> listBook = new ArrayList<>() Arrays.asList(
        new Book( id: 1, title: "Lập trình Windows", author: "Nguyễn Huy Cường", price: 99999),
        new Book( id: 2, title: "Lập trình Web", author: "Nguyễn Huy Cường", price: 12345),
        new Book( id: 3, title: "Lập trình ứng dụng Java", author: "Nguyễn Huy Cường", price: 454534),
        new Book( id: 4, title: "Thương mại điện tử", author: "Nguyễn Đình Ảnh", price: 848593));

    3 usages
    @GetMapping("/books")
    public List<Book> GetAll() {
        return listBook;
    }

    3 usages
    @GetMapping("/books/{id}")
    public Book get(@PathVariable int id)
    {
        var findBook = listBook.stream().filter(p->p.getId()== id).findFirst().orElse( other: null);
        if(findBook == null)
            throw new RuntimeException(HttpStatus.NOT_FOUND);
        return findBook;
    }

    0 usages
    @PostMapping("/books")
    public Book create(@RequestBody Book book)
    {
        //get max bookid
        var maxId = listBook
            .stream() Stream<Book>
            .mapToInt(v -> v.getId()) IntStream
            .max().orElse( other: 0);

        book.setId(maxId+1);
        listBook.add(book);
        return book;
    }
}
```

```
@DeleteMapping("/books/{id}")
public void delete(@PathVariable int id)
{
    var findBook = listBook.stream().filter(p->p.getId()== id).findFirst().orElse( other: null);
    if(findBook == null)
        throw new RuntimeException(HttpStatus.NOT_FOUND);
    listBook.remove(findBook);
}
```

```
// usages
public class Book
{
    3 usages
    private int id;
    3 usages
    private String title;
    3 usages
    private String author;
    3 usages
    private long price;
}
```

Vấn đề khi viết thêm xóa sửa trong cùng controller ?

- Với cách viết trong cùng Controller thì có thể thêm, xóa, sửa ...trong 1 controller

Có cách nào có thể tiếp tục truy xuất books từ controller khác không ?

VD: Từ SearchController, gọi “Search” để lấy tên tất cả cuốn sách hoặc tựa sách theo từ khóa tìm kiếm ?

(1) sử dụng public static list : không được quản lý bởi Spring và không có quyền truy cập vào Spring context hoặc bean do Spring quản lý

(2) sử dụng bean:

=> và đưa về kiến trúc: controller - Service – model

sử dụng các annotation: @Service , @Autowired



Chuyển sang mô hình controller – service – model

The screenshot displays an IDE with a project structure on the left and a code editor on the right. The project structure shows a package hierarchy: `book` (root) → `src` → `main` → `java` → `com.trinhbongdung.java.bai2` → `book`. Inside the `book` package, there are three sub-packages: `controller`, `model`, and `service`. The `controller` package contains `BookController`, `model` contains `Book`, and `service` contains `BookService` and `BookApplication`. The `BookController` class is highlighted in the project structure with a green box. The code editor shows the `BookController.java` file. It starts with imports for `java.util.ArrayList`, `java.util.Arrays`, and `java.util.List`. The class is annotated with `@RestController`. The `BookController` class contains a `private BookService bookService;` field, which is annotated with `@Autowired`. The constructor `public BookController()` initializes `bookService` with `new BookService()`. The class also contains three methods: `getAll()` (returns `bookService.getAll()`), `get(int id)` (returns `bookService.get(id)`), and `create(Book book)` (calls `bookService.add(book)` and returns `book`). The `delete(int id)` method is also shown, calling `bookService.remove(id)`.

```
11 import java.util.ArrayList;
12 import java.util.Arrays;
13 import java.util.List;
14
15 no usages
16 @RestController
17 public class BookController {
18
19     no usages
20     private BookService bookService;
21
22     public BookController()
23     {
24         bookService = new BookService();
25     }
26
27     no usages
28     @GetMapping("/books")
29     public List<Book> GetAll() { return bookService.getAll(); }
30
31     no usages
32     @GetMapping("/books/{id}")
33     public Book get(@PathVariable int id) { return bookService.get(id); }
34
35     no usages
36     @PostMapping("/books")
37     @ResponseBody
38     public Book create(@RequestBody Book book)
39     {
40         bookService.add(book);
41         return book;
42     }
43
44     no usages
45     @DeleteMapping("/books/{id}")
46     @ResponseBody
47     public void delete(@PathVariable int id) { bookService.remove(id); }
48 }
```

controller – service – model (POJO)

@Service

```
public class BookService {
    private List<Book> listBook = new ArrayList<>( Arrays.asList(
        new Book(1, "Lập trình Windows", "Nguyễn Huy Cường", 99999),
        new Book(2, "Lập trình Web", "Nguyễn Huy Cường", 12345),
        new Book(3, "Lập trình ứng dụng Java", "Nguyễn Huy Cường", 454534),
        new Book(4, "Thương mại điện tử", "Nguyễn Đình Ảnh", 848593)));
```

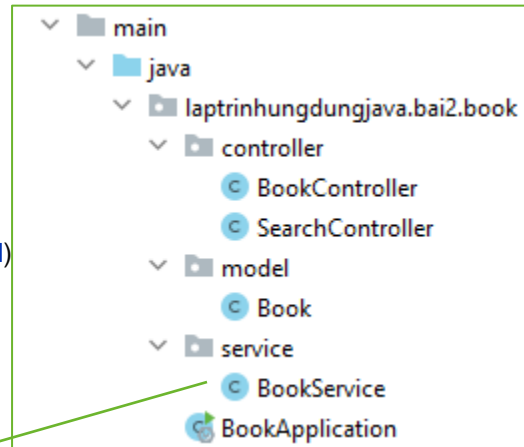
```
public List<Book> getAll()
{
    return listBook;
}
```

```
public Book get(int id)
{
    var findBook = listBook.stream().filter(p->p.getId()== id).findFirst().orElse(null)
    if(findBook == null)
        throw new RuntimeException(HttpStatus.NOT_FOUND);
    return findBook;
}
```

```
public void add(Book newbook)
{
    var maxId = listBook
        .stream()
        .mapToInt(v -> v.getId())
        .max().orElse( 0);
    newbook.setId(maxId + 1);
    listBook.add(newbook);
}
```

```
public void remove(int id)
{
    var findBook = listBook.stream().filter(p->p.getId() == id).findFirst().orElseThrow();
    listBook.remove(findBook);
}
```

```
public List<Book> search(String key)
{
    return listBook.stream().filter(p->p.getAuthor().toLowerCase().contains(key) || p.getTitle().toLowerCase().contains(key)).toList(
}
```



```
public class Book
{
    private int id;
    private String title;
    private String author;
    private long price;
    ....
}
```

@RestController

```
public class BookController {
```

@Autowired

```
private BookService bookService;
```

@GetMapping("/books")

```
public List<Book> GetAll() {
    return bookService.getAll();
}
```

@GetMapping("/books/{id}")

```
public Book get(@PathVariable int id)
{
    return bookService.get(id);
}
```

@PostMapping("/books")

```
public Book create(@RequestBody
Book book)
{
    bookService.add(book);
    return book;
}
```

@DeleteMapping("/books/{id}")

```
public void delete(@PathVariable int
id)
{
    bookService.remove(id);
}
```

@RestController

```
public class SearchController {
```

@Autowired

```
private BookService bookService;
```

@GetMapping("/search")

```
public List<Book>
search(@RequestParam String key)
{
    return bookService.search(key);
}
```




model : Lombok vs record

Vấn đề: Khi bạn làm việc nhiều với data model, việc tạo ra các getters, setters hay các constructors với các params khác nhau cho *Model class* => công việc nhàm chán

Giải pháp:

❑ **Sử dụng Lombok** là 1 thư viện Java giúp tự sinh ra các hàm setter/getter, hàm khởi tạo, toString... và tinh gọn chúng.

- Để các IDE có thể hiểu được cần cài đặt Project Lombok plugin.
- Tạo project sử dụng thư viện Project Lombok ở dependency
- 1 số Annotation : `@NoArgsConstructor`, `@AllArgsConstructor`, `@Data`, `@Getter`/ `@Setter`

❑ **Sử dụng record** trong java (JDK 14) để thay thế DTO object

```
public record Book(int id, String title, String author, long price) {  
  
}
```

ASM2: Viết Rest API cho quản lý xe ô tô

- Viết Rest API trên spring boot cho phép quản lý xe ô tô (lấy tất cả, thêm, xóa, sửa).
Viết thêm 1 Search API cho phép tìm kiếm: (1) Tìm biển số đẹp (2) Tìm xe theo số chỗ ngồi (3) tìm xe theo năm sản xuất
Biết mỗi xe có
 - biển số xe: (String) có 9 kí tự xxxx-yyyyy
 - ngày sản xuất: (LocalDate)
 - số chỗ ngồi (integer)
 - có đăng ký kinh doanh hay không (Boolean)
- Các yêu cầu:
- kiến trúc controller – service – model
 - Biết rằng: số xe đẹp là có biển số yyyyy mà có ít nhất 4 số giống nhau, hoặc tăng liên tiếp