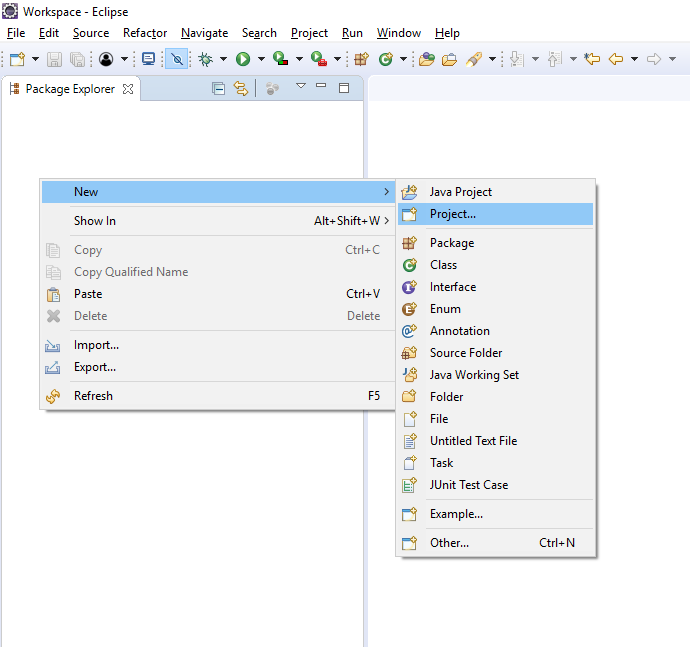
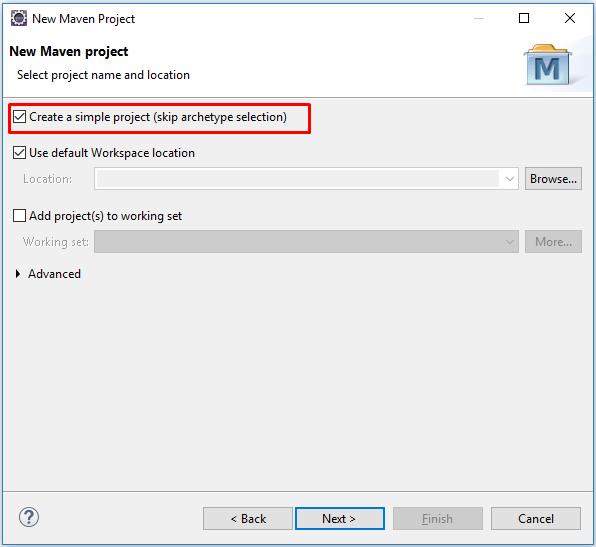
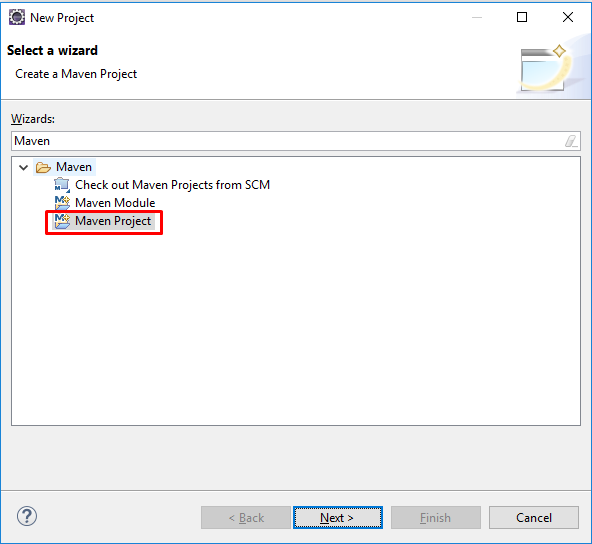
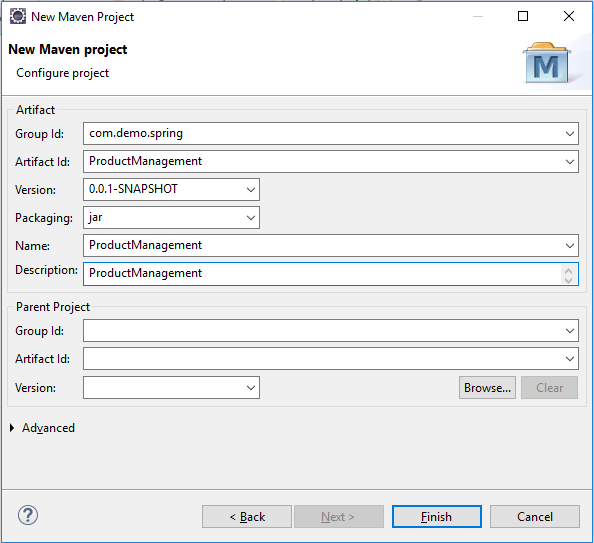
# WebSite với Spring: Spring Boot - Security, MVC, JPA

## Tạo Project maven

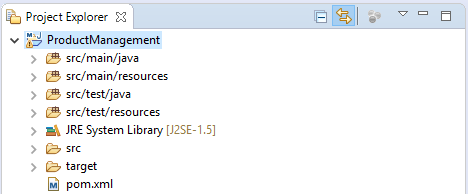
### Step1: Tạo Maven Project







Điền các thong số và chờ đợi Eclipse tạo project



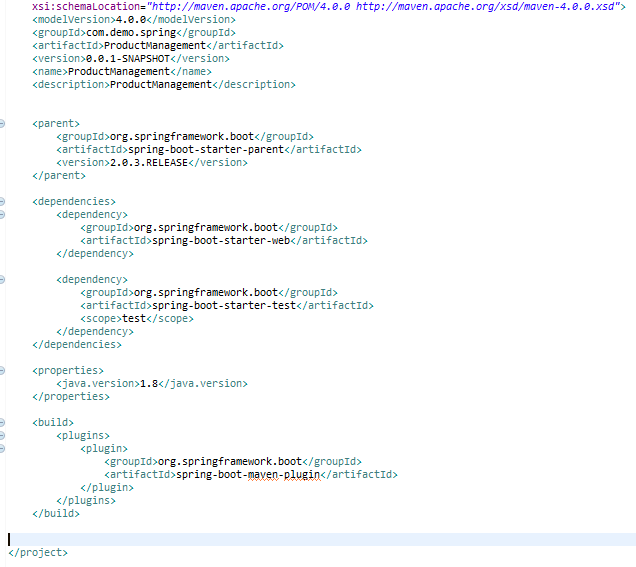
Kết quả Step 1 là hình trên. Nếu không thành công các bạn có thể tạo bằng tay thêm thư mục hoặc buil lại

### Fix lỗi Step1

* Cài maven riêng
* Chạy lệnh mvn eclipse:eclipse với thư mục Project
* Tạo Folder và add Resources

### Step2: Add thư viện Spring cho project

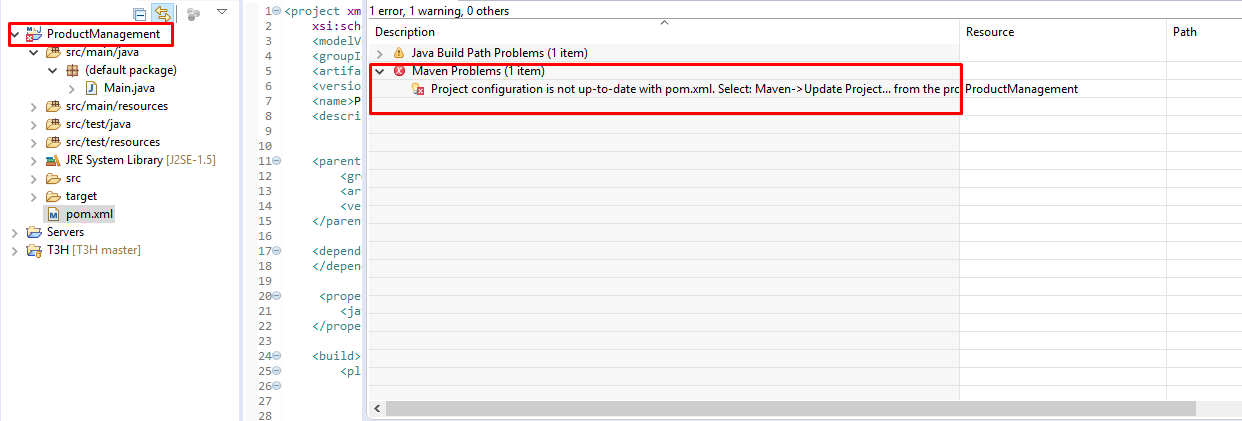
Tới file pom.xml và add thư viện và các thông số



Nội dung add mới

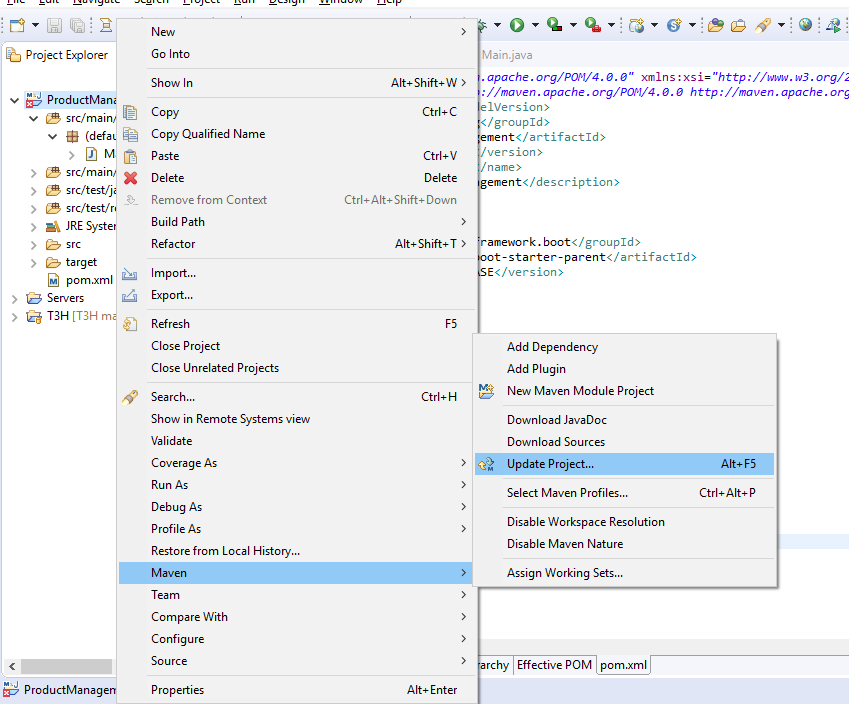
|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.demo.spring</groupId>  <artifactId>ProductManagement</artifactId>  <version>0.0.1-SNAPSHOT</version>  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <properties>  <java.version>1.8</java.version>  </properties>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |

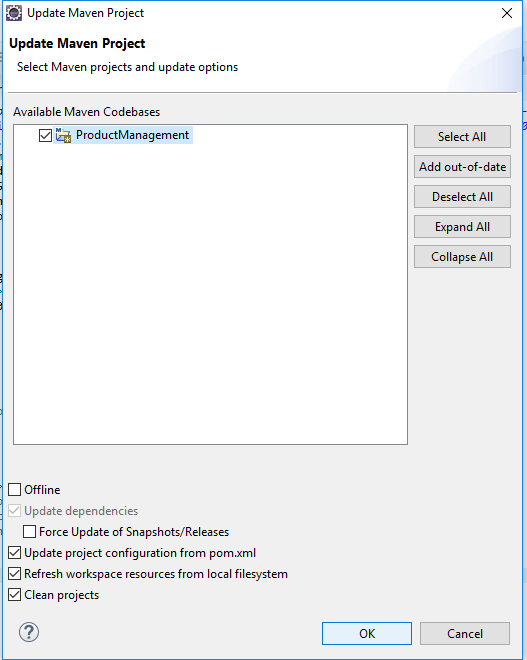
Hãy lưu lại và Build project lại



Khi bị lỗi project bạn có thể thấy gợi ý của Eclipse như trên

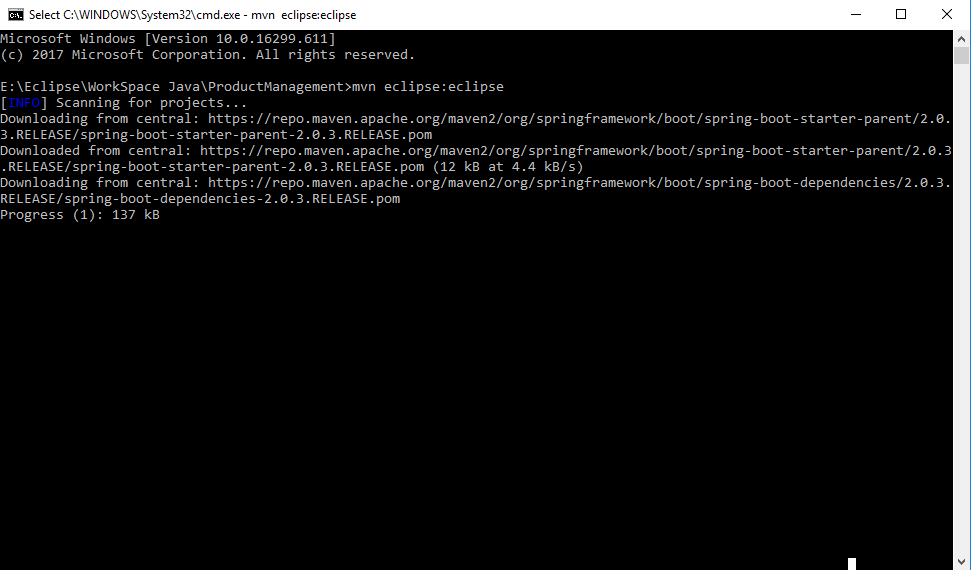
Chuột phải Project và Chọn như hình

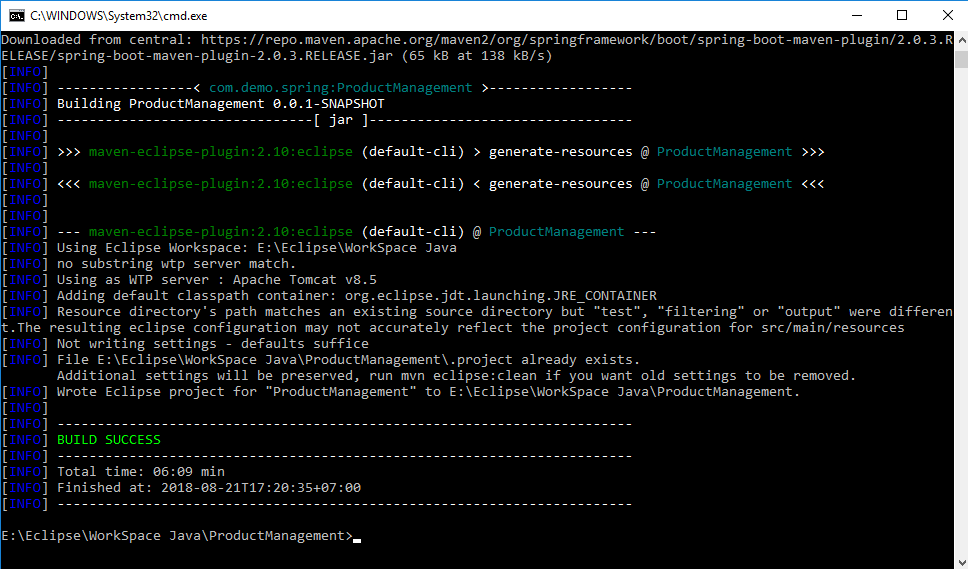




### Fix lỗi Step 2

* Có thể trỏ tới thư mục chứa project và chạy lệnh maven với maven cài riêng





Nếu không cài maven riêng hãy cố gắng buil lại project khi Build project thành công

Và update lại thư viện của project như trên Maven > Update project

## HelloWorld Project Spring Boot

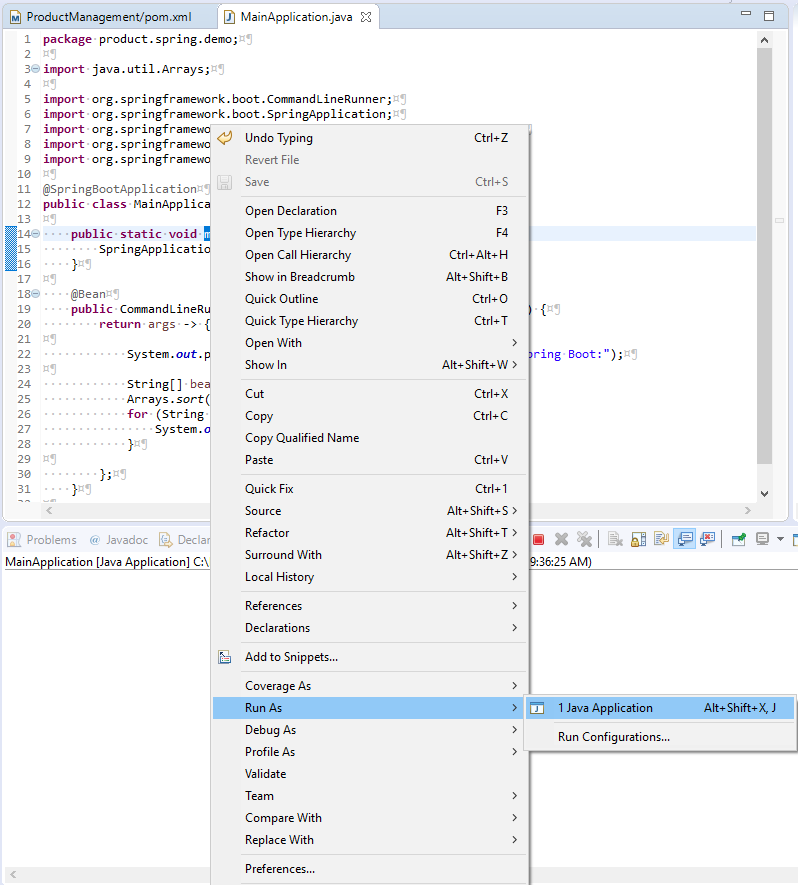
### Link tham khảo

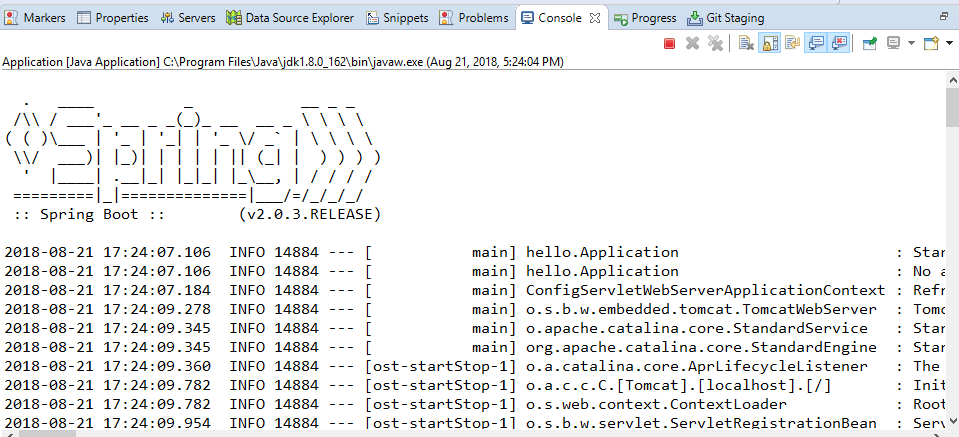
##### https://spring.io/guides/gs/spring-boot/

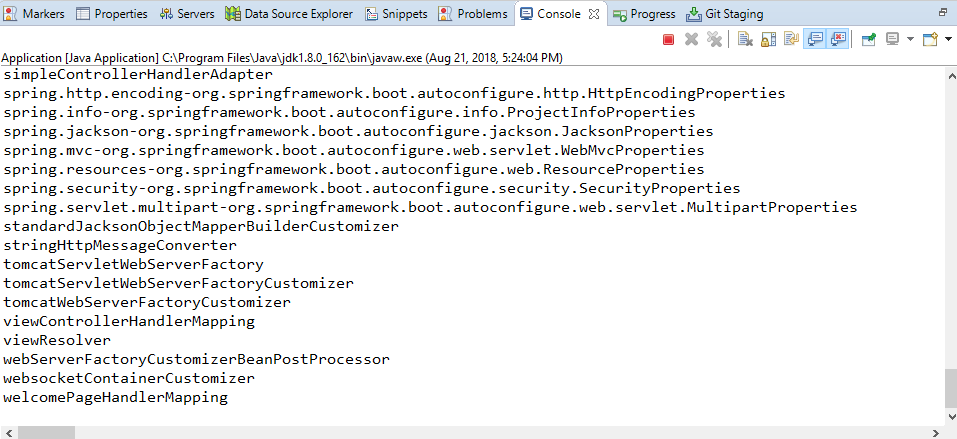
### Step 3 Tạo Main Application

|  |
| --- |
| **package** product.spring.demo;  **import** java.util.Arrays;  **import** org.springframework.boot.CommandLineRunner;  **import** org.springframework.boot.SpringApplication;  **import** org.springframework.boot.autoconfigure.SpringBootApplication;  **import** org.springframework.context.ApplicationContext;  **import** org.springframework.context.annotation.Bean;  @SpringBootApplication  **public** **class** MainApplication {  **public** **static** **void** main(String[] args) {  SpringApplication.run(MainApplication.**class**, args);  }  @Bean  **public** CommandLineRunner commandLineRunner(ApplicationContext ctx) {  **return** args -> {  System.out.println("Let's inspect the beans provided by Spring Boot:");  String[] beanNames = ctx.getBeanDefinitionNames();  Arrays.sort(beanNames);  **for** (String beanName : beanNames) {  System.out.println(beanName);  }  };  }  } |

Hãy tạo controller và chạy thử kết quả







Hình kết quả là các bean tạo sẵn của Spring Boot nói riêng và Spring nói chung

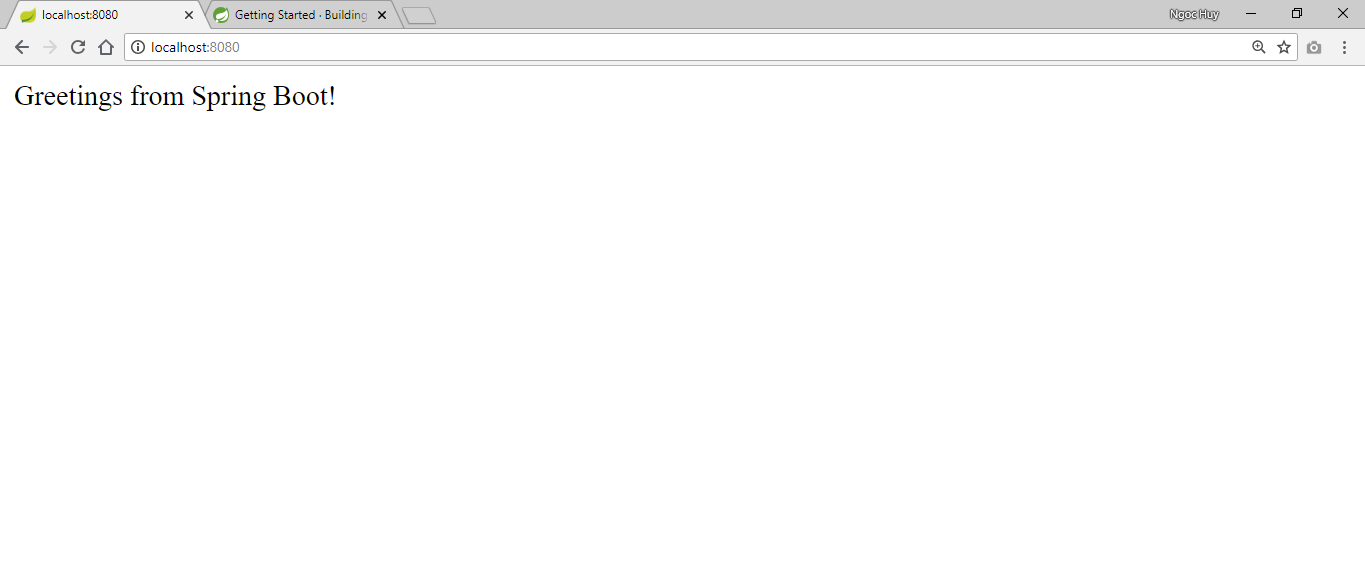
### Fix lỗi trong Step3

Step này không có lỗi nếu như bước 2 đã thành công, hãy build lại project.

### Step 4: Tạo Controller và chạy web

|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RestController;  @RestController  **public** **class** HelloController {  @RequestMapping("/")  **public** String index() {  **return** "Greetings from Spring Boot!";  }  } |

Khi tạo controller này hãy chạy lại Main Application và kiểm tra kết quả với đường dẫn trình duyệt web: <http://localhost:8080>



Kết quả chạy: <http://localhost:8080>

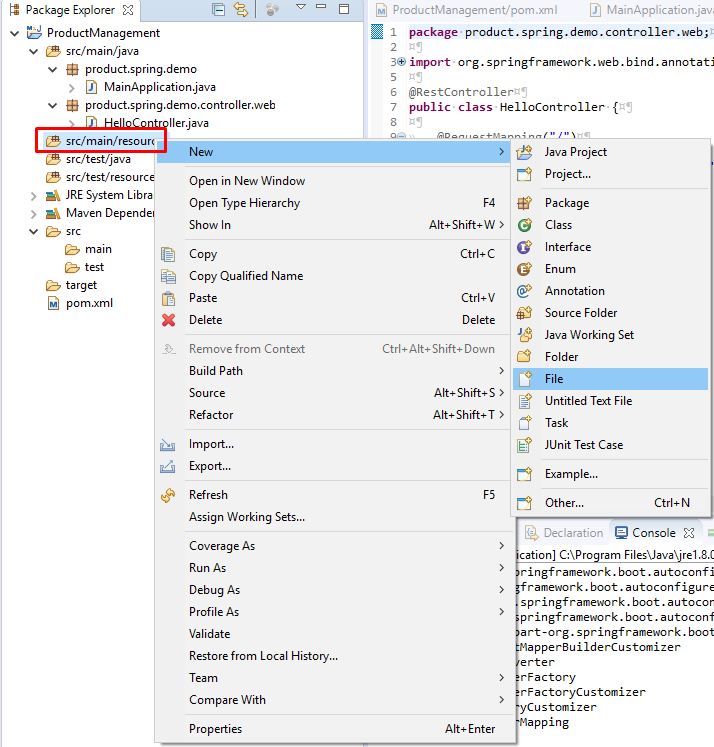
### Lưu ý Step4

* Mặc định Spring Boot chạy với Tomcat là 8080
* Tomcat được nhúng trong Project, Khi Spring Boot chạy nó sẽ khởi động Tomcat này

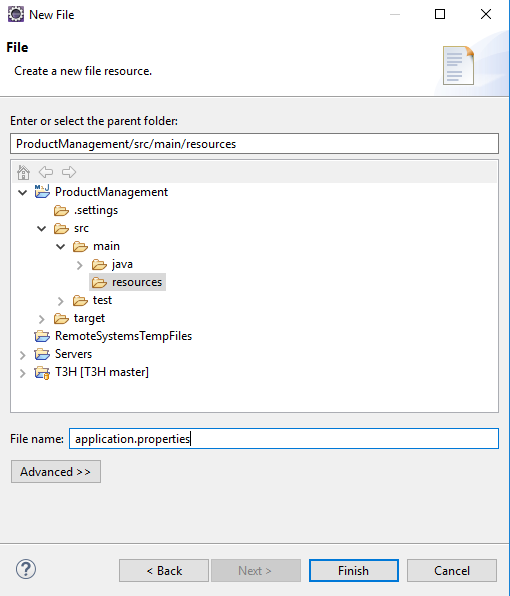
### Step 5: Tạo file cấu hình ứng dụng application.properties

Cách hay nhất trong Spring Boot là sử dụng application.properties để config ứng dụng

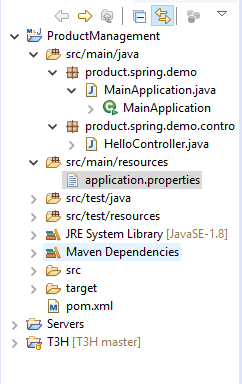
Hãy làm theo hình sau.



Chuột phải folder để tạo file



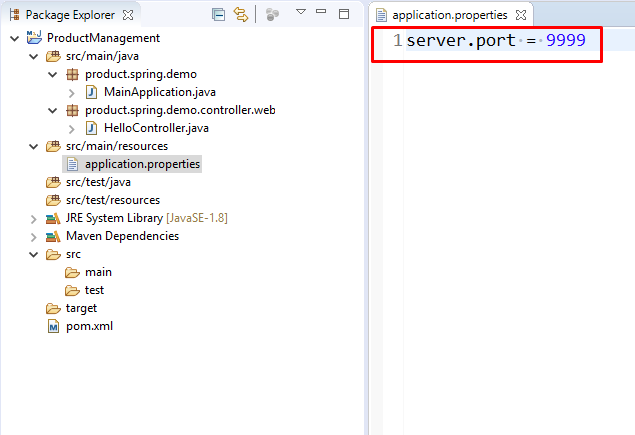
Tạo đúng tên file



Kết quả

### Step 6: Cấu hình cổng cho ứng dụng trong application.properties

Hãy cấu hình port bạn muốn:

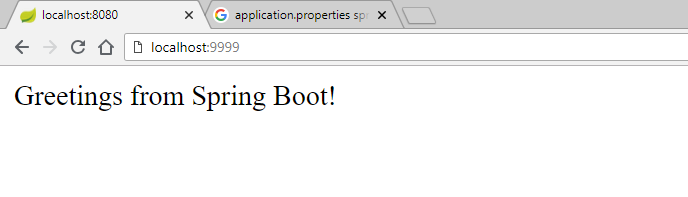


Cấu hình port 9999

Lưu lại và khởi động và kiểm tra

Sử dụng server.port = 9999 trong file application.properties để cấu hình ứng dụng chạy cổng 9999

Kết quả khi chạy lại ứng dụng:



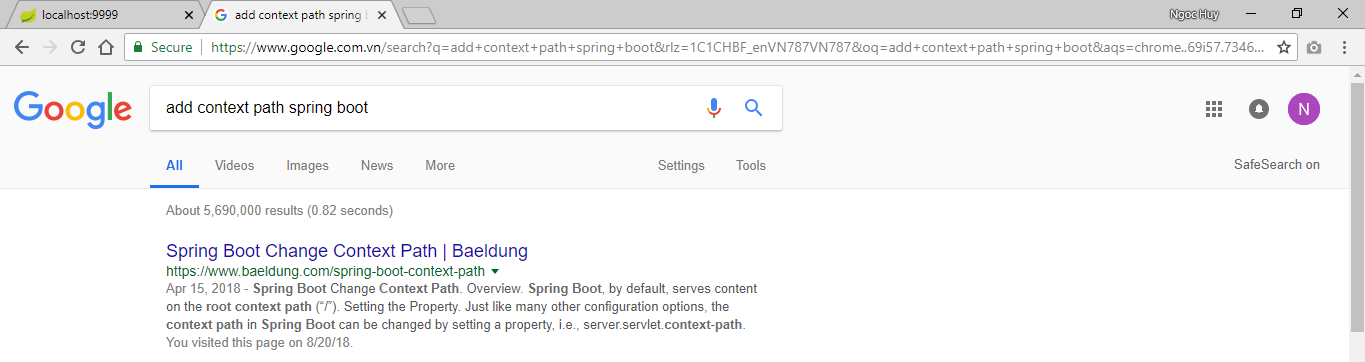
Kết quả

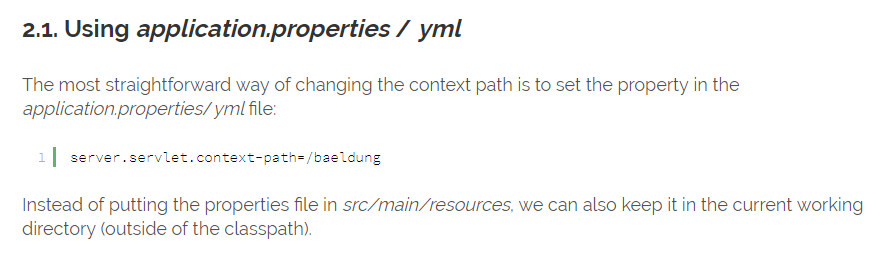
### Step7: Cấu hình context-path ứng dụng

Context Path có thể tạm hiểu là đường tới trang chủ của ứng dụng

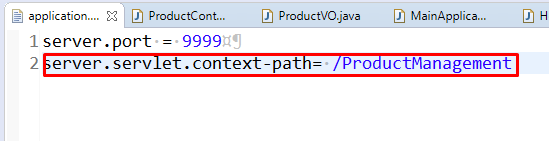
Nếu không cấu hình có thể hiểu là localhost:port là context path mặc định của ứng dụng Spring Boot

Nếu chúng ta muốn localhost:port /ProductManagement thì làm thế nào?



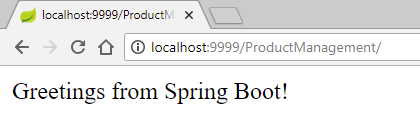
Chủ động tìm kiếm

Hướng dẫn sử dụng context-path

Hãy add cấu hình sau:



### Kết quả bước 6 và 7



### Step8: Add dev-tool Spring Boot

Dùng devtool để hỗ trợ phát triển ứng dụng. Kết hợp auto build của Eclipse và dev tool chúng ta sẽ không phải khởi động lại ứng dụng mỗi lần chạy

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <optional>true</optional>  </dependency> |

Hãy Maven => Update Project khi add thư viện vào ứng dụng

Với phần này chúng ta có thể coi project hiện tại là Hello với Spring MVC và Spring Boot

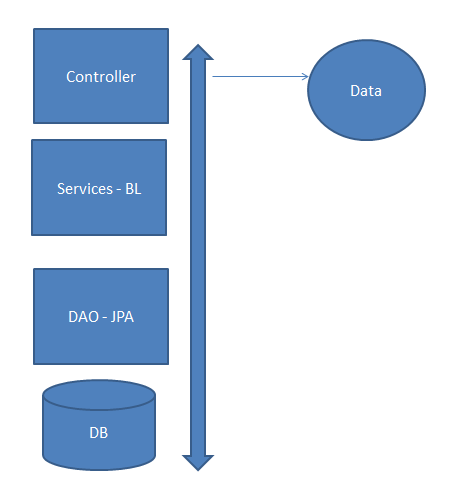
File pom.xml phần này

## Nâng cấp ứng dụng từ phần 2

### Link tham khảo mô hình

##### <https://techtalk.vn/mo-hinh-3-lop-co-gi-hay.html>

### Mô hình ứng dụng



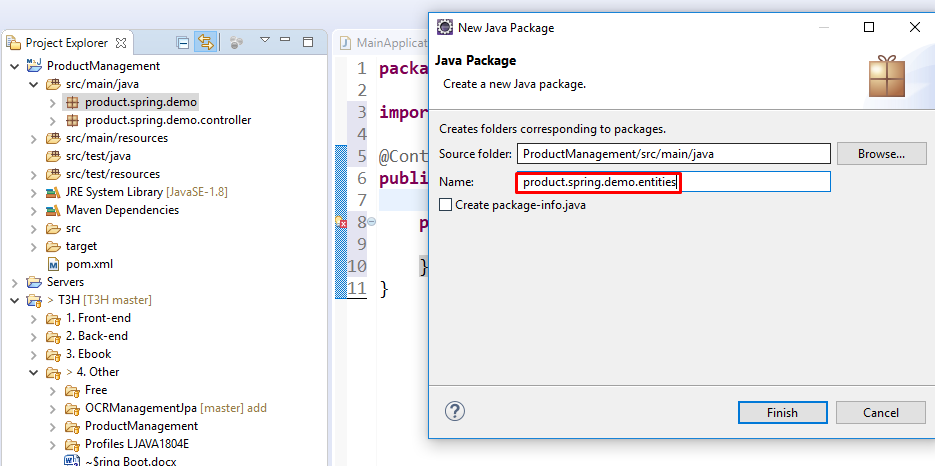
Mô hình giữa các lớp ứng dụng

### Step9: Tạo package

Hãy tạo các package riêng biệt ứng với mỗi phần

|  |  |
| --- | --- |
| product.spring.demo | Chứa main ứng dụng |
| product.spring.demo.controller | Chứa các controller cho web services |
| product.spring.demo.controller.web | Tạo các controller cho Spring MVC |
| product.spring.demo.entities | Chứa các entities class mapping với bảng |
| product.spring.demo.services | Chứa tầng Service |
| product.spring.demo.dao | Chứa tầng DAO kết nối với Cơ sở dữ liệu |
| product.spring.demo.config | Chứa các cấu hình cho ứng dụng |
| product.spring.demo.vo | Chứa các value object |
| //Toto | //Todo |

Hãy tự tạo các package theo ứng dụng của mình và hãy chủ động tìm hiểu về các mô hình tốt nhất



## Tạo web service với Spring Boot

### Link tham khảo

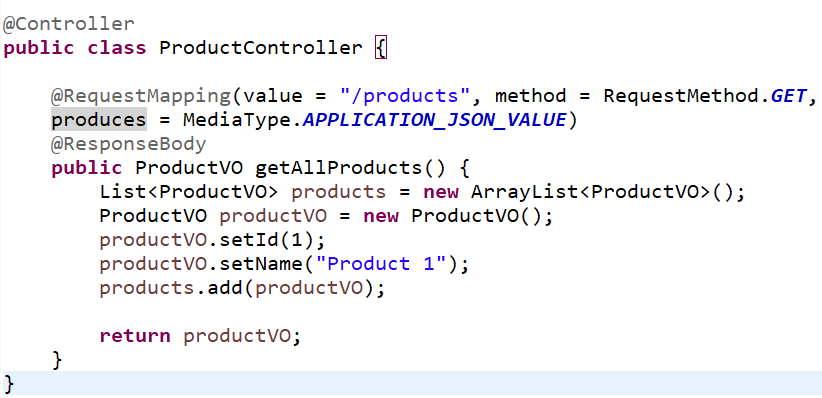
Hãy tham khảo link sau với RESTful service

##### <https://spring.io/guides/gs/rest-service/>

##### <https://springframework.guru/spring-framework-annotations/>

### Step10: Tạo Controller cho web service

Có rất nhiều controllers đây là một controllers cho việc quản lý Product. Và mỗi controller có rất nhiều phương thức quản lý các hành vi với mỗi đối tượng





### Giải thích Step 10

|  |  |
| --- | --- |
| @Controller | Để spring có thể scan quét được |
| @RequestMapping => value | Mapping URL: context path + value |
| @RequestMapping => method | Metho với URL trên |
| @RequestMapping => produces | Loại dữ dữ liệu được sử dụng |
| @ResponseBody | Dữ liệu trả về được nhúng trong Body của response |

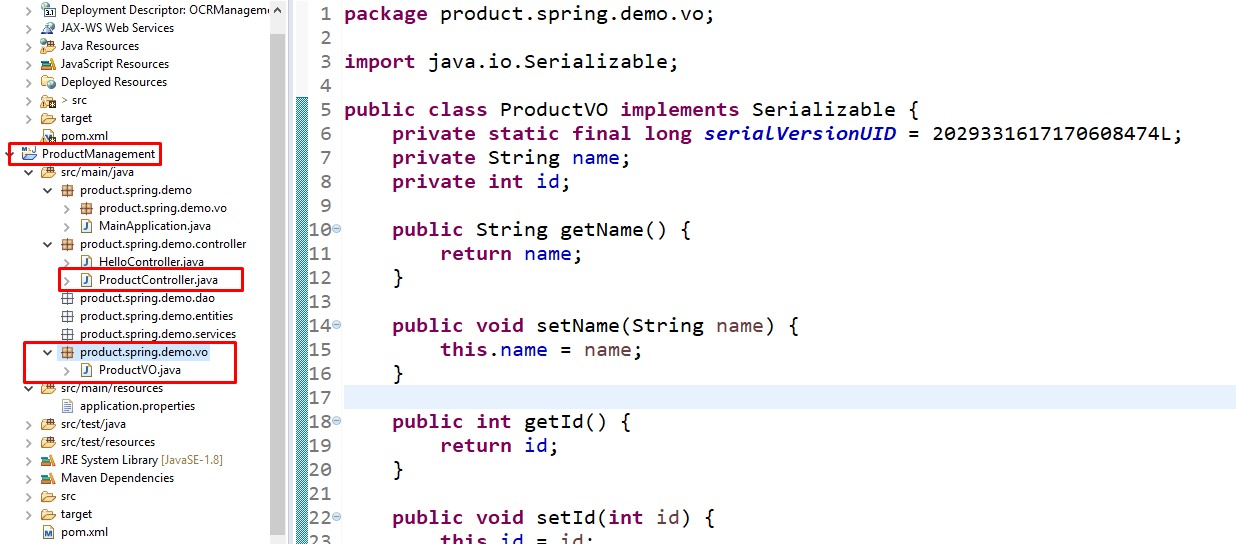
### Lưu ý Step 10

* **Dùng thì mới tạo** .Hãy tạo các controllers tạo các web services với khi chúng ta sử dụng. Nếu chúng ta không sử dụng hiện tại và tương lai thì cũng không nên tạo làm gì. Và code như vậy sẽ tạo ra rác trong ứng dụng
* Hãy tham khảo cách dùng với: **produces** và **consumes** với các method GET, POST trong Spring
* Package-scan: Trong Spring có package-scan là package đẻ Spring ‘quét’ để tạo đọc các bean, controller, cấu hình,…. Với ứng dụng ‘product.spring.demo’ là package-scan. Nếu package chứa controller mà không là con của package-scan thì Spring không hiểu được

### Step 11: Tạo POJO – Value Object cho Service

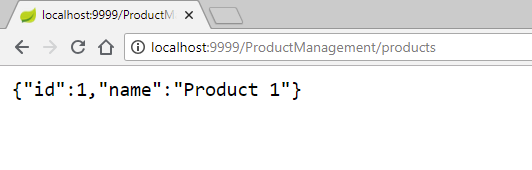
Bước này có thể tạo trước bước 10, đó là chúng ta tạo một class đơn giản cho Service mình muốn





### Kết quả Kiểm tra service Step 11

Chạy đường dẫn:



### Step 12: Sử dụng Swagger để test WebService

Swagger là một “plugin” thú vị trong quá trình phát triển Service.

Các bạn có thể chạy trình duyệt để kiểm tra kết quả Step 11. Nhưng để quản lý và hiển thị các controller để test thì chỉ có thể Swagger là mạnh mẽ.

Swagger tự động quét các phương trong các Controller

#### Add thêm thư viện

|  |
| --- |
| <!--For create test case -->  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger2</artifactId>  <version>2.7.0</version>  </dependency>  <dependency>  <groupId>io.springfox</groupId>  <artifactId>springfox-swagger-ui</artifactId>  <version>2.7.0</version>  </dependency> |

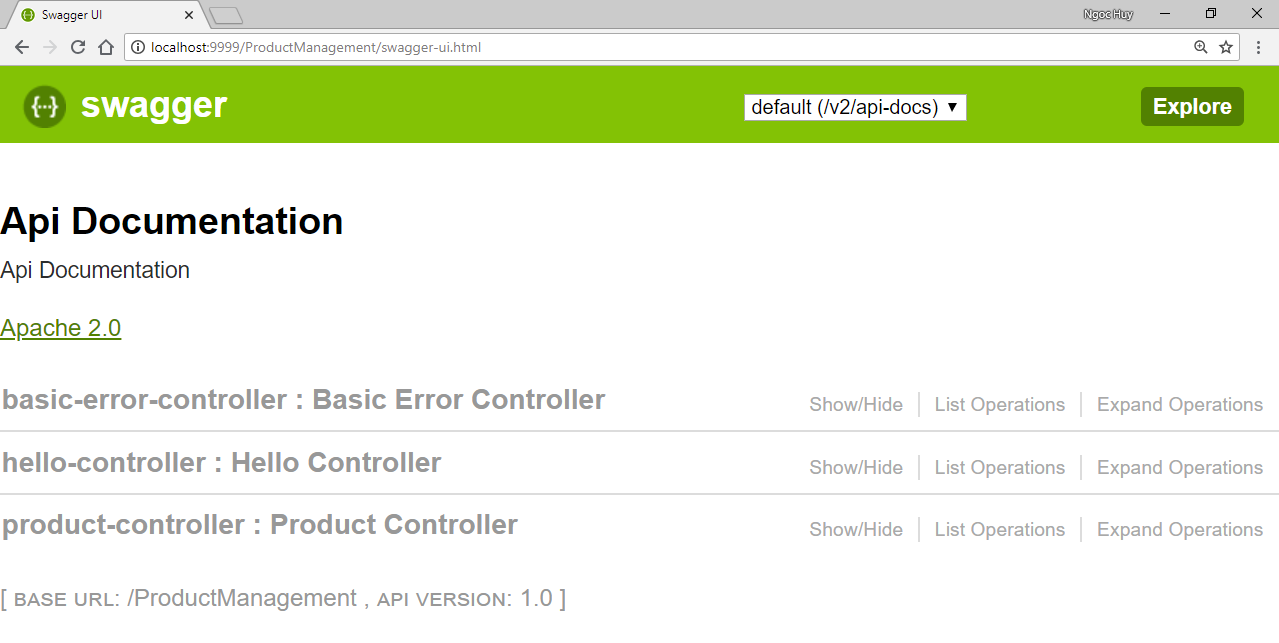
#### Build lại và Update Project

#### Add Bean Swagger



|  |
| --- |
| @EnableSwagger2  **public** **class** SwaggerConfig {  @Bean  **public** Docket api() {  **return** **new** Docket(DocumentationType.***SWAGGER\_2***).select().apis(RequestHandlerSelectors.*any*())  .paths(PathSelectors.*any*()).build();  }  } |

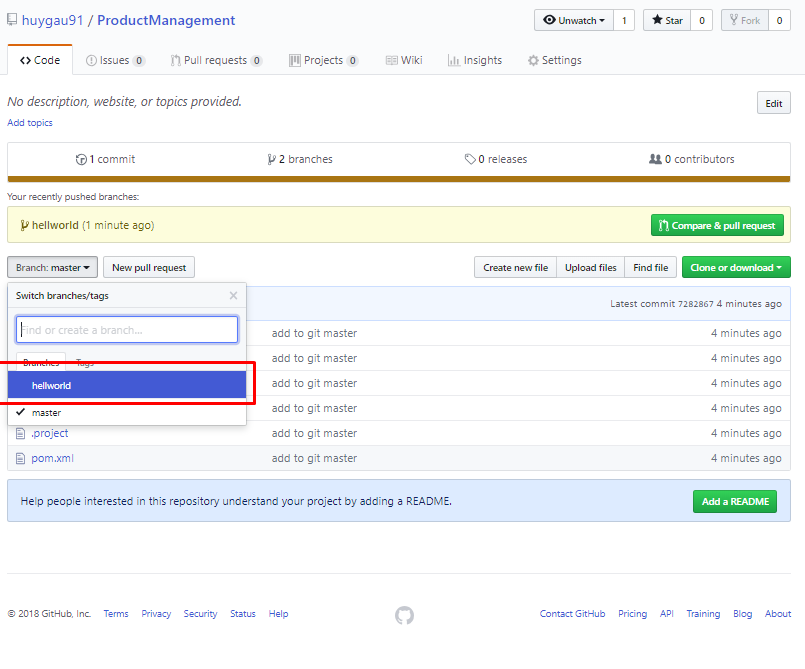
#### Test với đường dẫn: Context-Path + /swagger-ui.html



<http://localhost:9999/ProductManagement/swagger-ui.html>

## Project thời điểm phần 4



Hoặc <https://github.com/huygau91/ProductManagement> với Branch 

## Cấu hình template Appache Tiles và Tomcat

Với JSP để tạo template thì Appche Tiles được sử dụng khá phổ biến và dùng dễ dàng

### Step 13: Thư viện, JSP, Appche Tiles

#### Thư viện cho JSP

Muốn sử dụng JSP thì phải sử dụng thư viện hỗ trợ JSP trong Spring Boot

|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  <version>1.2</version>  </dependency> |

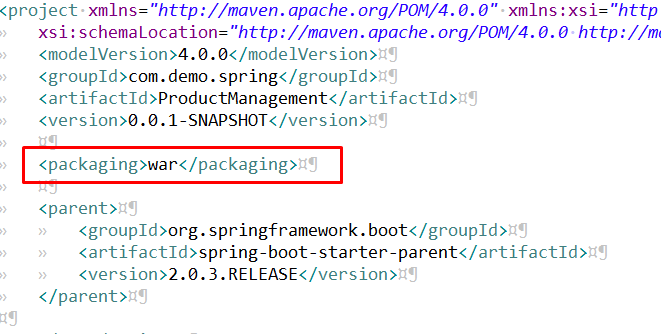
#### Add Appche Tiles

|  |
| --- |
| <!-- Tiles API -->  <!-- http://mvnrepository.com/artifact/org.apache.tiles/tiles-api%20 -->  <dependency>  <groupId>org.apache.tiles</groupId>  <artifactId>tiles-api</artifactId>  <version>3.0.8</version>  </dependency>  <!-- Tiles Core -->  <!-- http://mvnrepository.com/artifact/org.apache.tiles/tiles-core%20 -->  <dependency>  <groupId>org.apache.tiles</groupId>  <artifactId>tiles-core</artifactId>  <version>3.0.8</version>  </dependency>  <!-- Tiles Servlet -->  <!-- http://mvnrepository.com/artifact/org.apache.tiles/tiles-servlet%20 -->  <dependency>  <groupId>org.apache.tiles</groupId>  <artifactId>tiles-servlet</artifactId>  <version>3.0.8</version>  </dependency>  <!-- Tiles JSP -->  <!-- http://mvnrepository.com/artifact/org.apache.tiles/tiles-jsp%20 -->  <dependency>  <groupId>org.apache.tiles</groupId>  <artifactId>tiles-jsp</artifactId>  <version>3.0.8</version>  </dependency>  <!-- https://mvnrepository.com/artifact/org.apache.tiles/tiles-request-api -->  <dependency>  <groupId>org.apache.tiles</groupId>  <artifactId>tiles-request-api</artifactId>  <version>1.0.6</version>  </dependency> |

### Step 14: Cấu hình Project với Tomcat

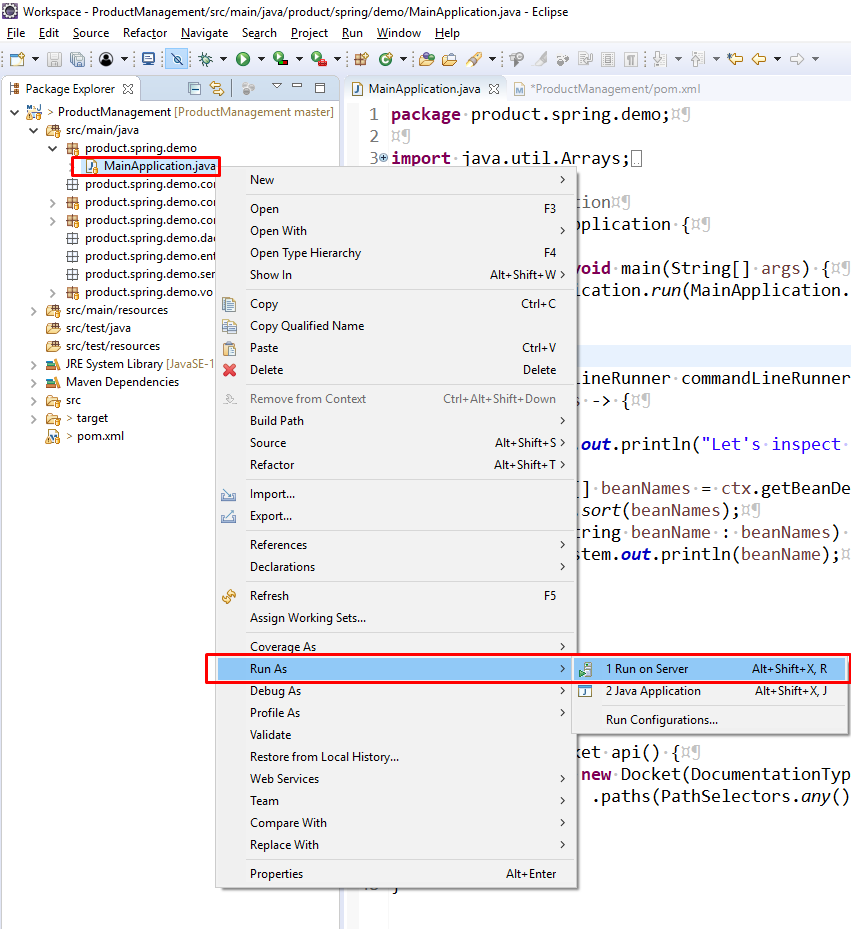
Project hiện tại có hàm main chạy Run as với Java Application. Nhưng làm sao để chạy trên Server có sẵn ?

#### Add Buil Project dạng War



File pom.xml trong ứng ở Step 14 này

Khi add tới đây thì ứng dụng có thể build ra dạng war file để deploy trên bất kì server Java nào hỗ trợ war file



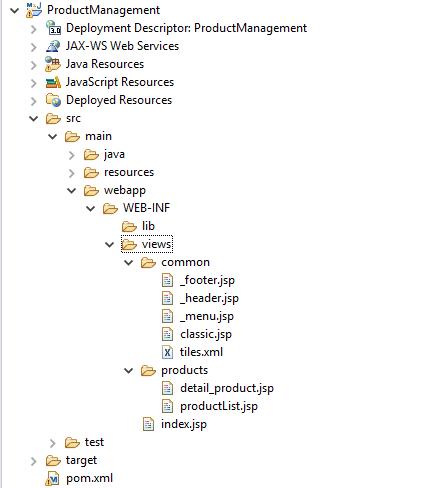
Dạng .jar là mặc định build của Spring Boot. Add War hỗ trợ build server Tomcat ngoài. Mỗi dạng file mà một cách chạy.

#### Hãy Build và Update Project với Maven

#### Add thư mục Web

Sau khi add <packaging>war</packaging> trong pom.xml và buil lại ứng dụng chúng ta có thư mục mới “webapp”. Nếu không có ta tạo bằng tay

Hãy tạo thêm các thư mục khác và các file như trong hình ảnh



Hãy Tạo thư mục các File như trên

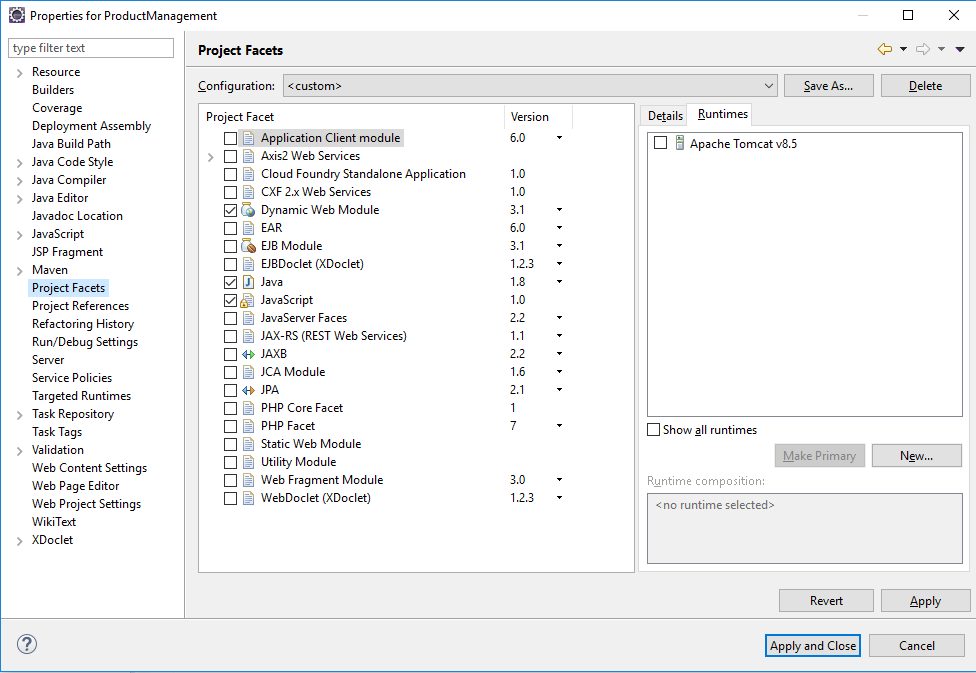
Các thành phần cần tạo:

* **webapp/WEB\_INF** thư mục chứa các thành phần web….
* **webapp/WEB\_INF/views** chứa các file web .jsp
* **webapp/WEB\_INF/views/common** chứa các file jsp dùng chung
* **webapp/WEB\_INF/views/product** chứa các file cho việc hiển thị sản phẩm
* **webapp/WEB\_INF/views/index.jsp** là trang chủ ứng dụng
* **webapp/WEB\_INF/views/ common/tiles.xml** là file định nghĩa các thành phần template và tên view
* **webapp/WEB\_INF/views/ common/classic.jsp** là file định nghĩa template

Trên là các thành phần sẽ phục vụ demo, các bạn có thể tạo khác và tạo thêm theo ứng dụng và ý đồ của các bạn

#### Kiểm tra lại Cấu hình Project - Project Pacets

Chuột phải Project > **Properties** và Chọn **Project Pacets**



Xem lại cấu hình đã phù hợp môi trường máy tính của bạn không?

### Step 15: Cấu hình ViewResole, CSS, JS,..

#### Tham khảo

##### https://www.baeldung.com/spring-mvc-view-resolver-tutorial

Website thì cần đọc CSS và JS, Bootstrap, Font, Các Trang JSP

|  |
| --- |
| server.port = 9999  server.servlet.context-path=/ProductManagemenet  #web mvc  spring.mvc.view.prefix:/WEB-INF/views/  spring.mvc.view.suffix:.jsp  security.ignored=/css/\*\*,/js/\*\*,/images/\*\*,/font/\*\* |

Đây là cấu hình ViewResole và đọc CSS và JS

Trong cấu hình trên khi Spring MVC trả về tên view thì nó sẽ tìm trong thư mục

/WEB-INF/views/ và chỉ chấp nhận file .jsp

File application.properties

### Step 16: Config tiles.xml

Như đã nói bước trước tiles.xml tại Step 14 là file định nghĩa các thần phần template

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"utf-8"* ?>  <!DOCTYPE tiles-definitions PUBLIC  "-//Apache Software Foundation//DTD Tiles Configuration 3.0//EN"  "http://tiles.apache.org/dtds/tiles-config\_3\_0.dtd">  <tiles-definitions>  <!-- Base Define -->  <definition name=*"base.definition"* template=*"/WEB-INF/views/common/classic.jsp"*>  <put-attribute name=*"title"* value=*""* />  <put-attribute name=*"header"* value=*"/WEB-INF/views/common/\_header.jsp"* />  <put-attribute name=*"menu"* value=*"/WEB-INF/views/common/\_menu.jsp"* />  <put-attribute name=*"body"* value=*""* />  <put-attribute name=*"footer"* value=*"/WEB-INF/views/common/\_footer.jsp"* />  </definition>  <definition name=*"homePage"* extends=*"base.definition"*>  <put-attribute name=*"title"* value=*"Home Page"* />  <put-attribute name=*"body"*  value=*"/WEB-INF/views/index.jsp"* />  </definition>    <definition name=*"productListPage"* extends=*"base.definition"*>  <put-attribute name=*"title"* value=*"Product List"* />  <put-attribute name=*"body"*  value=*"/WEB-INF/views/products/productList.jsp"* />  </definition>    <definition name=*"detailProductPage"* extends=*"base.definition"*>  <put-attribute name=*"title"* value=*"Detail Product"* />  <put-attribute name=*"body"*  value=*"/WEB-INF/views/products/detail\_product.jsp"* />  </definition>  </tiles-definitions> |

### Giải thích Step 16

#### Định nghĩa template

|  |
| --- |
| <!-- Base Define -->  <definition name=*"base.definition"* template=*"/WEB-INF/views/common/classic.jsp"*>  <put-attribute name=*"title"* value=*""* />  <put-attribute name=*"header"* value=*"/WEB-INF/views/common/\_header.jsp"* />  <put-attribute name=*"menu"* value=*"/WEB-INF/views/common/\_menu.jsp"* />  <put-attribute name=*"body"* value=*""* />  <put-attribute name=*"footer"* value=*"/WEB-INF/views/common/\_footer.jsp"* />  </definition> |

* <definition name=*"base.definition"* template=*"/WEB-INF/views/common/classic.jsp"*> định nghĩa một template với tên name=*"base.definition"* với nội dung file =*"/WEB-INF/views/common/classic.jsp*
* <put-attribute name=*"title"* value=*""* /> định nghĩa thuộc tính template tên *title.* Thuộc tính thay đổi thì sẽ có giá trị là: value=*""*
* <put-attribute name=*"menu"* value=*"/WEB-INF/views/common/\_menu.jsp"* />. Thuộc tính name=*"menu"* để hiển thị menu cho trang web

#### Cấu hình View

|  |
| --- |
| <definition name=*"homePage"* extends=*"base.definition"*>  <put-attribute name=*"title"* value=*"Home Page"* />  <put-attribute name=*"body"*  value=*"/WEB-INF/views/index.jsp"* />  </definition> |

* Hãy coi *base.definition* là một “supperclass” trong Java
* *homepage* là một “subclass” kế thừa *base.definition*
* Những thuộc tính chưa có giá trị trong *base.definition* ta sẽ đi định nghĩa trong “subclass” này để tương đương nó với một đường dẫn mới
* Những thuộc tính title, body chưa có giá trị trong supperclass thì subclass sẽ ghi đè để nội dung body, tiêu đề trang sẽ hiển thị phù hợp với mỗi trang tương ứng

### Step 17: Bean Appache Tile gọi tiles.xml

Hãy tạo class Java để config ứng dụng gọi tới tiles.xml

|  |
| --- |
| **package** product.spring.demo.config;  **import** org.springframework.context.annotation.Bean;  **import** org.springframework.context.annotation.Configuration;  **import** org.springframework.web.servlet.view.UrlBasedViewResolver;  **import** org.springframework.web.servlet.view.tiles3.TilesConfigurer;  **import** org.springframework.web.servlet.view.tiles3.TilesView;  @Configuration  **public** **class** TilesConfiguration {  @Bean  **public** TilesConfigurer tilesConfigurer() {  TilesConfigurer tilesConfigurer = **new** TilesConfigurer();  String[] defs = { "WEB-INF/views/common/tiles.xml" };  tilesConfigurer.setDefinitions(defs);  **return** tilesConfigurer;  }  @Bean  **public** UrlBasedViewResolver tilesViewResolver() {  UrlBasedViewResolver tilesViewResolver = **new** UrlBasedViewResolver();  tilesViewResolver.setViewClass(TilesView.**class**);  **return** tilesViewResolver;  }  } |

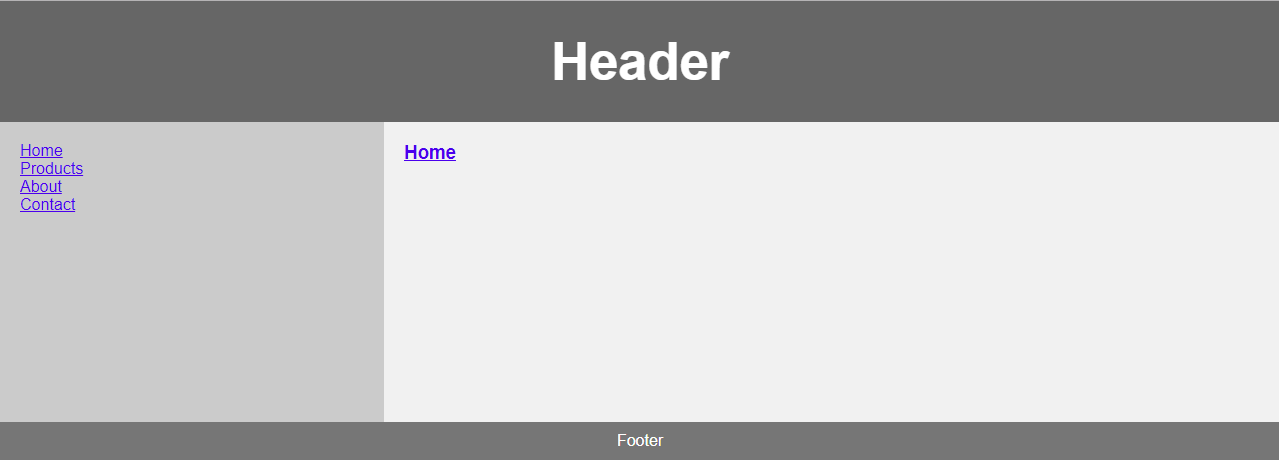
String[] defs = { "WEB-INF/views/common/tiles.xml" }; giúp chúng ta gọi tới cấu hình ở tiles.xml

### Step 18: Chuẩn bị layout cho teamplate

Classic.jsp là file cấu hình template chính. Ta cần gọi các thành phần header, footer,… để cấu hình ứng dụng.

Hãy sử dụng những giao diện mà các bạn có. Trong bài viết này sẽ sử dụng layout đã có sẵn sau:

[Click > Template mẫu demo](https://www.w3schools.com/code/tryit.asp?filename=FUSRUO3MGEZ1)



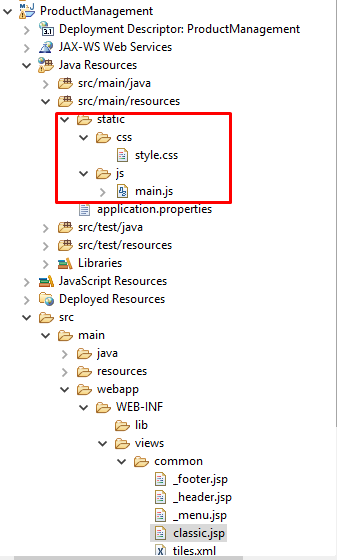
Tạo giao diện Template

### Step 19: Cắt html layout header, footer, menu,..

Hãy cắt các phần trong giao diện cho các file **\_footer.jsp, \_header.jsp, \_menu.jsp.**

Dựa vào layout mẫu chúng ta đưa nội dung mỗi phần vào mỗi file tương ứng

#### Cắt CSS, JS



“static“ là folder mặc định trong Spring Boot

Hãy tạo thư mục **static** trong “**src/main/resources**”. Và tạo thư mục **css, js.** Và hãy cắt style và javascript trong layout cho vào các file .css, .js .

Thư mục static là thư mục mặc định chứa css và js của Spring MVC trong Spring Boot project.



#### Cắt header, footer,…

Hãy cắt nội dung **header** và **footer, menu** vào trong các file **\_footer.jsp**,….



#### Gọi nội dung header, …js, css trong classic.jsp

|  |
| --- |
| <%@ taglib uri=*"http://tiles.apache.org/tags-tiles"* prefix=*"tiles"*%>  <%@ page language=*"java"* contentType=*"text/html; charset=utf-8"*  pageEncoding=*"utf-8"*%>  <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"* %>  <!DOCTYPE html>  <html>  <head>  <meta charset=*"utf-8"*>  <meta name=*"viewport"* content=*"width=device-width, initial-scale=1"*>    <title><tiles:getAsString name=*"title"* /></title>  <link rel=*"stylesheet"* href=*"*${pageContext.request.contextPath}*/css/style.css"*></link>  <script src=*"*${pageContext.request.contextPath}*/js/main.js"*></script>    <!-- pageContext.request.contextPath = http://localhost:9999/ProductManagermenet/ -->    </head>  <body>  <!--Header-->  <header>  <tiles:insertAttribute name=*"header"* />  </header>      <div class=*"main"*>  <tiles:insertAttribute name=*"menu"* />  <div class=*"content"*>  <tiles:insertAttribute name=*"body"* />  </div>  </div>  <tiles:insertAttribute name=*"footer"* />    </body>  </html> |

Nội dung của classic.jsp

Lưu ý:

* Gọi CSS, JS: Khi các bạn đưa JS và CSS và gọi qua context path

pageContext.request.contextPath = <http://localhost:9999/ProductManagement/>

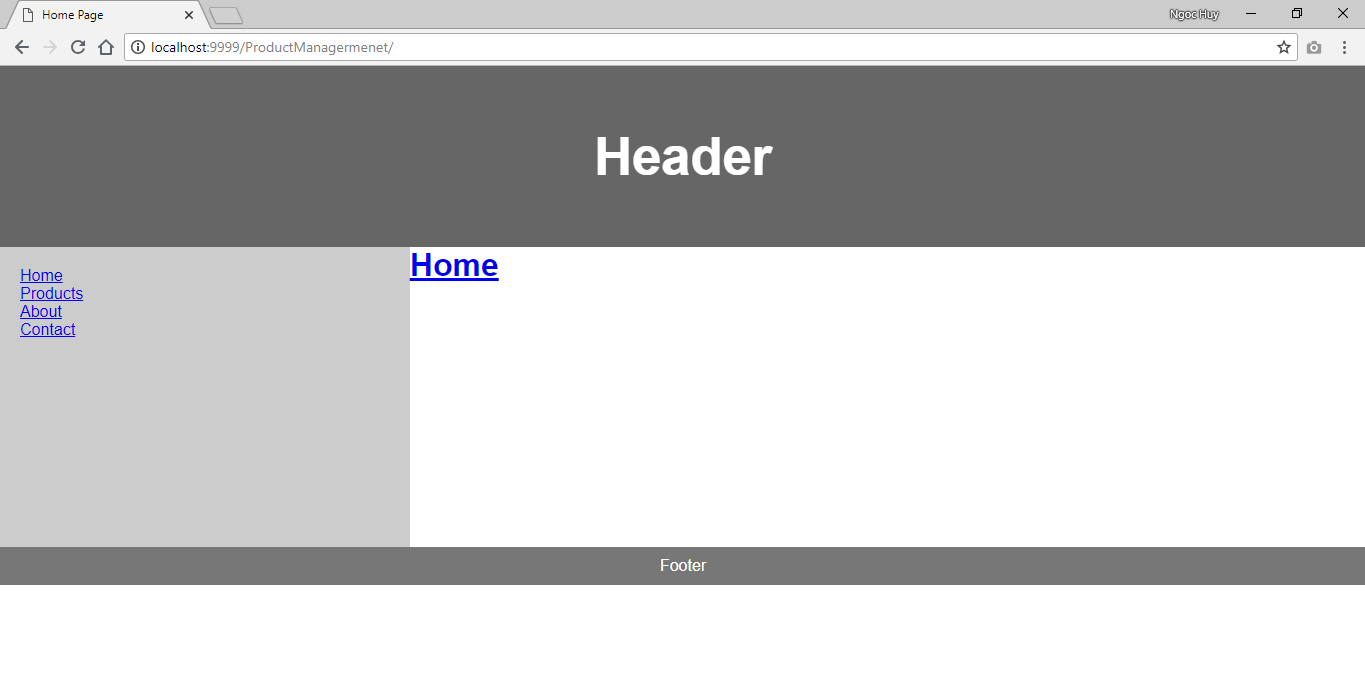
* Gọi nội dung file .jsp <tiles:insertAttribute name=*"header"* />
* Gọi text <tiles:getAsString name=*"title"* />

### Step 19.1 Update HelloController.java

|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller  **public** **class** HelloController {  @RequestMapping("/")  **public** String index() {  **return** "homePage";  }  } |

Bước này sẽ ảnh hưởng tới kết quả dưới. homepage ở đây phải trùng khớp với tên homepage tiles.xml. Đây là tên view. Mọi điều sẽ giải thích ở mục 6

### Kết quả sử dụng Appche Tiles



Kiểm tra kết quả <http://localhost:9999/ProductManagement/>

## Giới thiệu Spring MVC

### Giới thiệu Design Pattern

Design Pattern (mẫu thiết kế) có lẽ là từ khóa quá quen thuộc. Đơn giản như này khi các phần mềm thiết kế cần có sự tính toán về thiết kế: giao diện? Luồng đi dữ liệu từ front tới back và ngược lại.

Nhiều phần mềm thì lại có thiết kế riêng, và khó đối với lập trình viên khi đi từ dự án này sang dự án kia, và sự tái sử dụng thiết kế không tốt nếu không có chuẩn chung

Design Pattern đơn giản chỉ là sự thống kết về cách giải quyết một vấn đề theo nhiều cách khác nhau và phù hợp. Việc của lập trình viên là chọn ra thiết kế mà mình có thể áp dụng được và đảm bảo phần mềm chạy ổn định,…tùy theo nhu cầu hệ thống.

### Giới thiệu về MVC

MVC là một design pattern để thiết kế một phần mềm. Và nó rất phổ biến:

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>



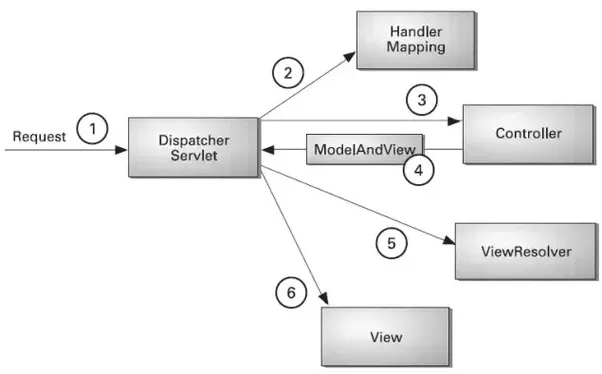
Mô hình MVC

Tóm gọn MVC:

* Model đại diện cho dữ liệu
* View đại diện cho hiển thị
* Controller đại diện cho xử lý

Một yêu cầu người dùng thì việc cần làm là 1.xử lý, và hiển thị kết quả xử lý 2.view. và gọi ngầm tới 3.Model để tương tác dữ liệu nếu cần.

### Giới thiệu Spring MVC



Spring MVC đơn giản kết hợp giữa Spring và MVC trên nền tảng web của Spring

1. Dispatcher Servlet – cơ bản là Servlet và là Controller trong MVC.

Giả sử người dùng request tới: /products để lấy các sản phẩm về thì ở đây Dispatcher Servlet sẽ lắng nghe nhân và xử lý request này với các thành phần khác

1. Dispatcher Servlet nghe rằng “/products” từ người dùng. Và nó phải đi tìm soi xem function nào thực hiện yêu cầu “/products” và nó nằm trong Controller (Controller của Spring)nào. Và Handler Mapping sẽ biết điều đó và nó sẽ hỏi Handler Mapping
2. Khi hỏi được con Contrller nào thực hiện nó sẽ đi gọi hàm trong Controller tương ứng
3. Contrller sẽ trả về một đối tượng ModelAndView. Bản thân đối tượng này chỉ là mang tên view và dữ liệu hiển thị trên view. Ví dụ *productListPage*
4. Khi biết tên view *productListPage* nó sẽ soi trong cấu hình ViewResolver xem nó sẽ đọc tới file nào? đuôi ra sao? trong folder nào?

|  |
| --- |
| #web mvc  spring.mvc.view.prefix:/WEB-INF/views/  spring.mvc.view.suffix:.jsp |

Ví dụ nó sẽ chấp nhận đọc file trong folder **prefix** và chấp nhận đuôi file **suffix.** Tuy nhiên khi ta cấu hình Appache Tiles như phần 5 thì nó sẽ có sự kế thừa và nó đọc tiles.xml thì đó là sự đặc biệt

1. Sau khi tìm view ví dụ .jsp, .html thì dữ liệu đổ qua jsp và trả về giao diện trên màn hình người dùng.

## Spring MVC với ứng dụng

### Kiểm tra cấu hình ViewResolver trong application.properties

|  |
| --- |
| server.port = 9999  server.servlet.context-path=/ProductManagermenet  #web mvc  spring.mvc.view.prefix:/WEB-INF/views/  spring.mvc.view.suffix:.jsp  security.ignored=/css/\*\*,/js/\*\*,/images/\*\*,/font/\*\* |

### Step 20: Tạo Controller

Tạo các Contrller quản lý các view của ứng dụng. Ví dụ quản lý Product thì tạo ProductPageContrller, Customer thì tạo CustomerPageContrller,….

Trong đây sẽ tạo 2 contrller: HomePageContrller và ProductPageController

#### 20.1 HomePageContrller

Hãy xóa bỏ hoặc đổi tên HelloController thành HomePageController và sửa với dung dung sau

|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  @Controller  **public** **class** HomePageController {  @RequestMapping(value = {  "",  "/",  "/index",  "/home"  })  **public** String index() {  **return** "homePage";  }  } |

Trong Contrller trên sẽ:

* value cấu hình đường dẫn
* homePage tên view trả về
* @Controller là annotation để biết class đó là Controller để Handler Mapping biết
* “”, “/”, “/index”, “/home” để biết người dùng nhập

http://localhost:9999/ProductManagermenet

<http://localhost:9999/ProductManagermenet/>

<http://localhost:9999/ProductManagermenet/index>

<http://localhost:9999/ProductManagermenet/home>

thì nó sẽ để quy về hàm index trong controller để trả về homepage

#### 20.2 ProductPageController

#### 20.2.1 Tạo ProductVO

VO –value object. Đây là object hiển thị

|  |
| --- |
| **package** product.spring.demo.vo;  **import** java.io.Serializable;  **public** **class** ProductVO **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 2029331617170608474L;  **private** String name;  **private** **int** id;  **private** String price;  **private** String image;  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **int** getId() {  **return** id;  }  **public** **void** setId(**int** id) {  **this**.id = id;  }  **public** String getPrice() {  **return** price;  }  **public** **void** setPrice(String price) {  **this**.price = price;  }  **public** String getImage() {  **return** image;  }  **public** **void** setImage(String image) {  **this**.image = image;  }  } |

#### 20.2.2 Tạo ProductPageController

|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.servlet.ModelAndView;  **import** product.spring.demo.vo.ProductVO;  @Controller  **public** **class** ProductPageController {  @RequestMapping(value = "/products", method = RequestMethod.GET)  **public** ModelAndView productPage() {  ModelAndView m = **new** ModelAndView("productListPage");  List<ProductVO> products = **new** ArrayList<ProductVO>();  ProductVO productVO = **new** ProductVO();  productVO.setId(1);  productVO.setName("Product 1");  productVO.setImage("/images/item.png");  products.add(productVO);  m.addObject("productList", products);  **return** m;  }  } |

Trong Controller trả về một trên view productListPage. Và tên view sẽ trả về trong Appche Tiles.

Giải thích Controller:

* @RequestMapping(value = "/products", method = RequestMethod.GET) lắng nghe method GET lấy dữ liệu
* ModelAndView m = **new** ModelAndView("productListPage"); chuẩn bị muột tên view
* Chuẩn bị dữ liệu hiển thị trên view

|  |
| --- |
| List<ProductVO> products = **new** ArrayList<ProductVO>();  ProductVO productVO = **new** ProductVO();  productVO.setId(1);  productVO.setName("Product 1");  productVO.setImage("/images/item.png");  products.add(productVO); |

* m.addObject("productList", products); Nhét object chứa dữ liệu vào view. View có thể dùng tên productList để lấy dữ liệu
* **return** m; trả về ModelAndView có chứa tên view và dữ liệu productList

#### 20.3 Kiểm tra các tên view

Chúng ta đã định nghĩa các tên view trong tiles.xml. Nên phải kiểm tra tên view trong Controller và Appche Tile có trùng khớp không?

|  |
| --- |
| <definition name="homePage" extends="base.definition"><put-attribute name="title" value="Home Page" /><put-attribute name="body"value="/WEB-INF/views/index.jsp" /></definition><definition name="productListPage" extends="base.definition"><put-attribute name="title" value="Product List" /><put-attribute name="body"value="/WEB-INF/views/products/productList.jsp" /></definition><definition name="detailProductPage" extends="base.definition"><put-attribute name="title" value="Detail Product" /><put-attribute name="body"value="/WEB-INF/views/products/detail\_product.jsp" /></definition> |

#### Lưu ý

1. Tùy biến trả về method trong Controller

Các bạn sẽ thấy chúng ta trả về HomePageController > index trả về String

Và ProductPageController > productPage trả về ModelAndView.

Nếu các bạn cần dữ liệu hiển thị thì hãy sử dụng object bằng cách “m.addObject("productList", products);”.

Nếu các bạn không cần dữ liệu hiển thị. Ví dụ trang 404 lỗi thì các bạn trả về String. Hãy làm sao mà sử dụng phù hợp

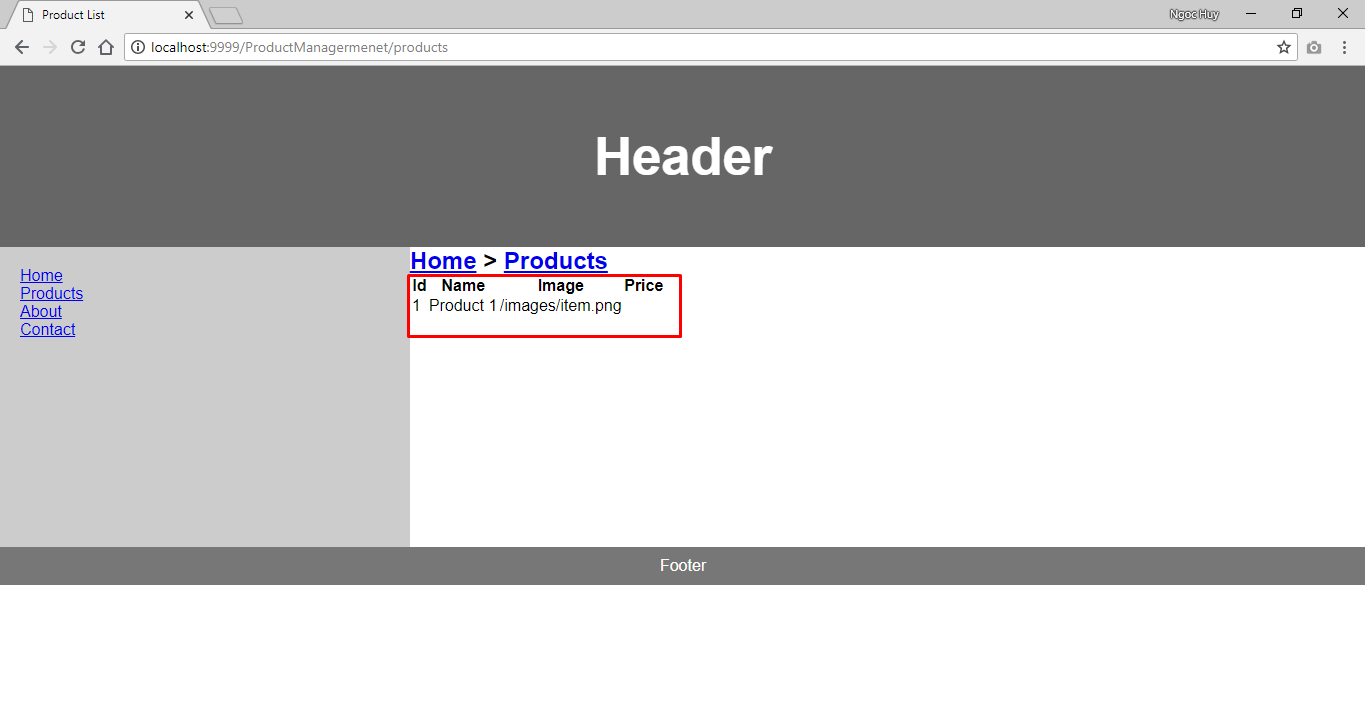
1. Kiểm trả tên view trả về trùng khớp với Appache Tiles config: tiles.xml như bước 20.3

### Step 21: Hiển thị Product trên view: *productList.jsp*

Như bước 20.2.2 , nếu người dùng nhập <http://localhost:9999/ProductManagermenet/products>” sẽ gọi tới file *"/WEB-INF/views/products/productList.jsp”.* Chúng ta sẽ đi sử dụng tên “productList” trong *productList.jsp* Như sau:

|  |
| --- |
| <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>  <%@ taglib prefix=*"fmt"* uri=*"http://java.sun.com/jsp/jstl/fmt"*%>  <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"*%>  <%@ taglib prefix=*"fn"* uri=*"http://java.sun.com/jsp/jstl/functions"*%>  <%@page import=*"java.util.List"*%>  <%@page import=*"product.spring.demo.vo.ProductVO"*%>  <h2>  <a href=*"/home"*>Home</a> > <a href=*"/products"*>Products</a>  </h2>  <p></p>  <table>  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Image</th>  <th>Price</th>  </tr>  </thead>  <tbody>  <c:forEach var=*"productItem"* items=*"*${productList}*"*>  <tr>  <td>${productItem.id}</td>  <td>${productItem.name}</td>  <td>${productItem.image}</td>  <td>${productItem.price}</td>  </tr>  </c:forEach>  </tbody>  </table> |

Hãy chạy đường dẫn : <http://localhost:9999/ProductManagermenet/products> để kiểm tra kết quả



Kết quả: Dữ liệu fake đã được hiển thị.

### Project thời điểm kết thúc phần 7

Link: <https://github.com/huygau91/ProductManagement.git>

Branch: apache\_tile\_v2

Viêc cần làm của các bạn là make-up giao diện cho đẹp và lung ling với kiến thức CSS, JS, Servlet JSP, JSTL của các bạn.

Hiện tại dữ liệu hiển thị là dữ liệu fake mà các bạn đã đưa vào trong ProductPageContrller. Bước tiêp theo là ta phải lấy dữ liệu cần có trong CSDL chúng ta.

Ở bước tới ta sẽ sử dụng Spring JPA bên trong đã tích hợp sẵn Hibernate. Ta sẽ cấu trúc Spring JPA theo DAO pattern <https://www.baeldung.com/java-dao-pattern>

## **8. Tích hợp Spring JPA**

Yêu cầu bước này, các bạn cần có kiến thức về Hibernate và cơ sở dữ liệu, thành thạo việc mapping giữa class và table trong hibernate

### Tham khảo một vài Annotation

* @Component, @Service, @Repository <https://www.baeldung.com/spring-component-repository-service>
* Các annotation của Hibernate <http://www.techferry.com/articles/hibernate-jpa-annotations.html> hỗ trợ việc viết Entity

### Step 22: Add thư viện Spring JPA và connector

Add vào dependencies và Run as Maven và Maven Update project

|  |
| --- |
| <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  </dependency>  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>6.0.5</version>  </dependency> |

### Step 24: Cấu hình Hibernate

|  |
| --- |
| # data source  spring.datasource.url=jdbc:mysql://localhost:3306/ProductManagement  spring.datasource.username=root  spring.datasource.password=  spring.jpa.properties.hibernate.show\_sql=true |

* url: Connection tring với CSDL của bạn
* username: Username có thể truy cập vào Database ProductManagement
* password của username đó
* show sql bằng config với hibernate

### Step 23: Tạo DB

Tạo bảng product

|  |
| --- |
| CREATE DATABASE IF NOT EXISTS ProductManagement;  USE ProductManagement;  CREATE TABLE IF NOT EXISTS product(  idProduct INT(11) PRIMARY KEY AUTO\_INCREMENT,  name VARCHAR(200),  image VARCHAR(200),  price VARCHAR(200),  idCategory int(11),  created\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  updated\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP); |

Tạo bảng category

|  |
| --- |
| CREATE DATABASE IF NOT EXISTS ProductManagement;  USE ProductManagement;  CREATE TABLE IF NOT EXISTS category(  idCategory INT(11) PRIMARY KEY AUTO\_INCREMENT,  name VARCHAR(200),  image VARCHAR(200),  description VARCHAR(200),  created\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,  updated\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP); |

Tạo kết nối các bảng

|  |
| --- |
| ALTER TABLE product ADD FOREIGN KEY(idCategory) REFERENCES category(idCategory) |

### Step 25 Tạo các Entities

Đồng hành với tạo bảng, chắc chắn là tạo các class entities là điều hiển nhiên. Với CSDL tạo 2 bảng ở bước trên ta sẽ tạo ra hai Entities.

Tạo package “product.spring.demo.entities” và tạo hai class bên trong.

Qúa nhiều cách mẹo tạo các Entities. Dùng tool trong Eclipse để kết nối tới CSDL để generate code hoặc viết tay

Tạo Entity Product

|  |
| --- |
| **package** product.spring.demo.entities;  **import** java.io.Serializable;  **import** java.util.Date;  **import** javax.persistence.Basic;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.JoinColumn;  **import** javax.persistence.ManyToOne;  **import** javax.persistence.Table;  **import** javax.persistence.Temporal;  **import** javax.persistence.TemporalType;  @Entity  @Table(name = "product")  **public** **class** Product **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = "idproduct")  **private** Integer id;  **private** String name;  **private** String image;  **private** String price;  @ManyToOne  @JoinColumn(name = "idcategory")  **private** Category category = **new** Category();  @Basic  @Column(name = "created\_date")  @Temporal(TemporalType.***TIMESTAMP***)  **private** Date createdDate;  @Basic  @Column(name = "updated\_date")  @Temporal(TemporalType.***TIMESTAMP***)  **private** Date updatedDate;  **//getter and setter** |

Tạo Category

|  |
| --- |
| **package** product.spring.demo.entities;  **import** java.io.Serializable;  **import** java.util.ArrayList;  **import** java.util.Date;  **import** java.util.List;  **import** javax.persistence.Basic;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.OneToMany;  **import** javax.persistence.Table;  **import** javax.persistence.Temporal;  **import** javax.persistence.TemporalType;  @Entity  @Table(name = "category")  **public** **class** Category **implements** Serializable {  **private** **static** **final** **long** ***serialVersionUID*** = 1L;  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  @Column(name = "idcategory")  **private** Integer id;  **private** String name;  **private** String image;  **private** String price;  @OneToMany(mappedBy = "category")  **private** List<Product> products = **new** ArrayList<>();  @Basic  @Column(name = "created\_date")  @Temporal(TemporalType.***TIMESTAMP***)  **private** Date createdDate;  @Basic  @Column(name = "updated\_date")  @Temporal(TemporalType.***TIMESTAMP***)  **private** Date updatedDate;  **//getter and setter** |

### Lưu ý bước 25

Nguyên tắc khi mapping

1. Mặc định của Spring JPA đó là các biến trong class mapping với các cột trong bảng là chữ thường. Nếu muốn mapping với đúng tên chữ viết vật lý trong bảng thì sử dụng thuộc tinh trong application.properties

|  |
| --- |
| spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl |

1. Biến trong class và tên cột trong bảng giống nhau thì sẽ không cần @Column
2. Kiểu dữ liệu cần mapping. Ví dụ trong Database dùng Date với kiểu TimeStamp. Thì cần mapping @Temporal(TemporalType.***TIMESTAMP***).
3. Có thể viết các annotation trên các biến hoặc getter của biến ấy
4. Một mapping với biến bảng thì cần có getter và setter để có thể sử dụng
5. Nếu không có khóa chính trong bảng đi chăng nữa thì @Id trong class không thể thiếu

Nên chạy lại ứng dụng trong bước này để xem có mapping chưa, Vì khi Spring Boot khởi động sẽ gọi mapping. Nếu như cấu hình sau thì sẽ thông báo lỗi

### Step 26. Tạo Repositories – Spring JPA

Tạo package repository và tạo 2 interface sau:

ProductRepository

|  |
| --- |
| **package** product.spring.demo.repositories;  **import** java.util.List;  **import** org.springframework.data.jpa.repository.JpaRepository;  **import** org.springframework.data.jpa.repository.Query;  **import** org.springframework.data.repository.query.Param;  **import** org.springframework.stereotype.Repository;  **import** product.spring.demo.entities.Product;  @Repository  **public** **interface** ProductRepository **extends** JpaRepository<Product, Integer> {  **final** String ***SELECT\_ALL*** = "SELECT p FROM Product p";  **final** String ***SELECT\_BY\_ID*** = "SELECT p FROM Product p WHERE p.id =:productId";  @Query(***SELECT\_ALL***)  List<Product> findAllProduct();    @Query(***SELECT\_BY\_ID***)  Product findCategoryById(@Param("productId") Integer id);  } |

CategoryRepository

|  |
| --- |
| **package** product.spring.demo.repositories;  **import** java.util.List;  **import** org.springframework.data.jpa.repository.JpaRepository;  **import** org.springframework.data.jpa.repository.Query;  **import** org.springframework.data.repository.query.Param;  **import** org.springframework.stereotype.Repository;  **import** product.spring.demo.entities.Category;  @Repository  **public** **interface** CategoryRepository **extends** JpaRepository<Category, Integer> {  **final** String ***SELECT\_ALL*** = "SELECT c FROM Category c";  **final** String ***SELECT\_BY\_ID*** = "SELECT c FROM Category c WHERE c.id =:categoryId";  @Query(***SELECT\_ALL***)  List<Category> findAllCategory();    @Query(***SELECT\_BY\_ID***)  Category findCategoryById(@Param("categoryId") Integer id);    } |

### Giải thích bước 26

Mỗi entity class bây giờ coi nó là một “bảng” trong Java, repository bây giờ hãy coi nó là một thành phần hứng những yêu cầu query để tương tác với các “bảng” ấy.

* @Repository bắt buộc phải có để Spring hiểu đây là thành phần Repository
* **final** String ***SELECT\_ALL*** = "SELECT p FROM Product p"; là câu lệnh HQL
* @Query("SELECT\_ALL") sẽ gọi tới câu lệnh trên
* List<Product> findAllProduct(); dữ liệu sau khi thực hiên câu lệnh sẽ trả về bảng
* (@Param("idCategory") truyền giá trị qua tới câu lệnh thông qua tham số idCategory trong câu lệnh

Trong bước 26 thực sự là chúng ta custom Repository, thực sự thì Spring JPA xây dựng rất nhiều Repository: CRUDRepository, JPARepository, chúng xây dựng cho các bản Java 8 trở lên với nhiều thức có sẵn.

Tuy vậy để đơn giản dễ dùng với những người mới học Java, hoặc những người quá quen với Java 7 thì ta sẽ dùng custom repository ở đây.

### Step 27. Tạo Services Layer

ở bước này có 3 bước:

* Tạo Interface
* Tạo Implementation
* Sử dụng Repository trong Implementation

#### Step 27.1 Tạo Interface

Tạo service cho Category

|  |
| --- |
| **package** product.spring.demo.services;  **import** java.util.List;  **import** product.spring.demo.entities.Category;  **public** **interface** CategoryService {    List<Category> getAllCategories();  Category getById(String id);  } |

Tương tự với Product

|  |
| --- |
| **package** product.spring.demo.services;  **import** java.util.List;  **import** product.spring.demo.entities.Product;  **public** **interface** ProductService {  List<Product> getAllProducts();  Product getById(String id);  } |

#### Step 27.2-3 Tạo Implementation và sử dụng Repository

Tạo Imp

Tạo Implementation cho Product Service

|  |
| --- |
| **package** product.spring.demo.services;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** product.spring.demo.entities.Product;  **import** product.spring.demo.repositories.ProductRepository;  @Service  **public** **class** ProductServiceImpl **implements** ProductService {    @Autowired  **private** ProductRepository productRepository;  @Override  **public** List<Product> getAllProducts() {  **return** productRepository.findAll();  }  @Override  **public** Product getById(String id) {  **return** productRepository.findCategoryById(Integer.*parseInt*(id));  }  } |

Tương tự với Category Services

|  |
| --- |
| **package** product.spring.demo.services;  **import** java.util.List;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Service;  **import** product.spring.demo.entities.Category;  **import** product.spring.demo.repositories.CategoryRepository;  @Service  **public** **class** CategoryServiceImpl **implements** CategoryService {  @Autowired  **private** CategoryRepository categoryRepository;  @Override  **public** List<Category> getAllCategories() {  **return** categoryRepository.findAllCategory();  }  @Override  **public** Category getById(String id) {  **return** categoryRepository.findCategoryById(Integer.*parseInt*(id));  }  } |

#### Giải thích 27.2-3

* @Service có thể thay bằng @Component đây là annotation để thông báo Spring: đây là thành phần cần config tạo bean. Hãy xem link tham khảo ở đầu bước 8
* @Autowired sử dụng móc nối tự động của Spring.
* **private** CategoryRepository categoryRepository; Sử dụng repository
* **return** categoryRepository.findAll(); gọi tới các câu lệnh để truy vấn

### Step 28: Nhúng Services vào Controller – Update Contrller

Cũng như Repository ta sẽ sử dụng @Autowired để nhúng Service trong Controller bằng Interface của nó

ở bước 7 ta đã làm show dữ liệu với fake dữ liệu

|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** java.util.ArrayList;  **import** java.util.List;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.servlet.ModelAndView;  **import** product.spring.demo.vo.ProductVO;  @Controller  **public** **class** ProductPageController {  @RequestMapping(value = "/products", method = RequestMethod.***GET***)  **public** ModelAndView productPage() {  ModelAndView m = **new** ModelAndView("productListPage");  List<ProductVO> products = **new** ArrayList<ProductVO>();  ProductVO productVO = **new** ProductVO();  productVO.setId(1);  productVO.setName("Product 1");  productVO.setImage("/images/item.png");  products.add(productVO);  m.addObject("productList", products);  **return** m;  }  } |

Sau khi sửa Controller

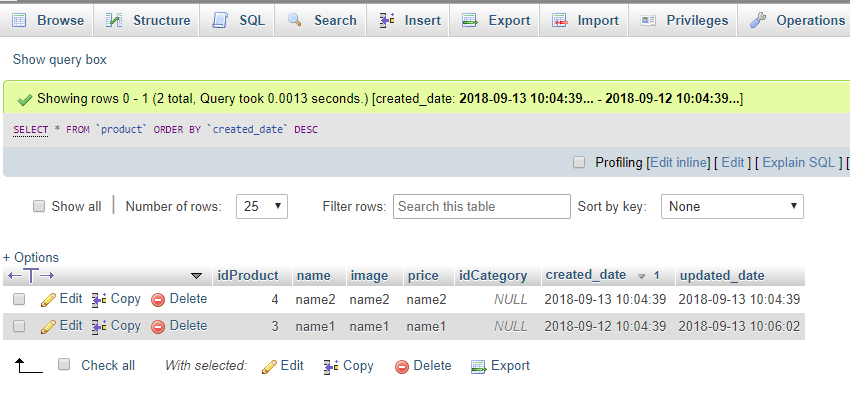
|  |
| --- |
| **package** product.spring.demo.controller.web;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.servlet.ModelAndView;  **import** product.spring.demo.services.ProductService;  @Controller  **public** **class** ProductPageController {  @Autowired  **private** ProductService productService;    @RequestMapping(value = "/products", method = RequestMethod.***GET***)  **public** ModelAndView productPage() {  ModelAndView m = **new** ModelAndView("productListPage");  m.addObject("productList", productService.getAllProducts());  **return** m;  }  } |

Sửa lại view – JSP

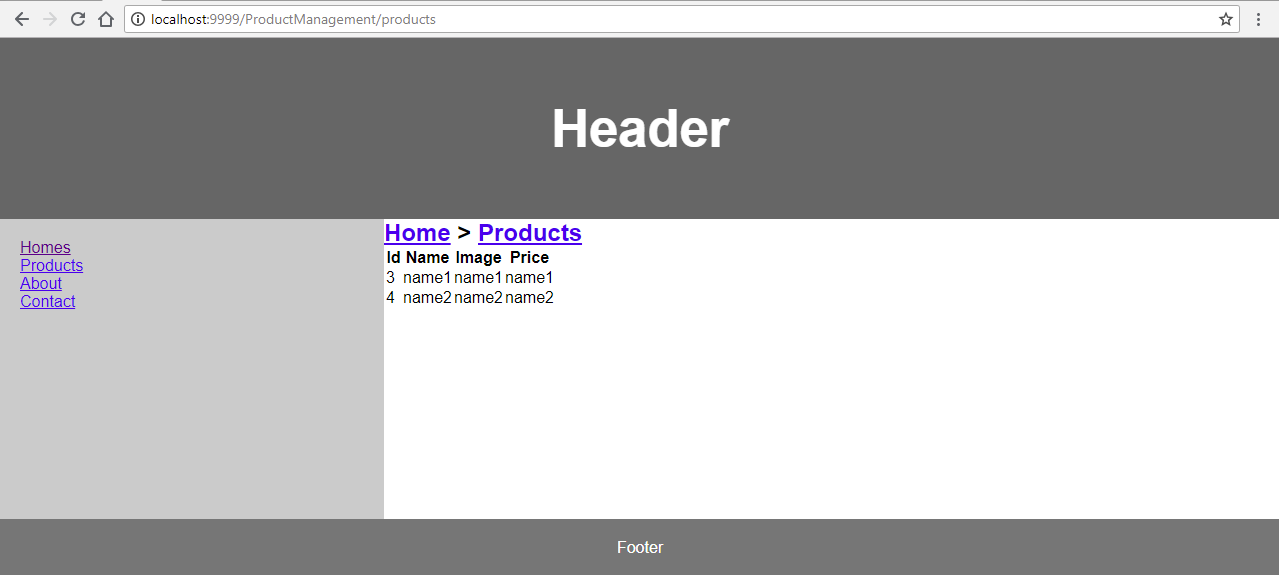
|  |
| --- |
| <%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"*%>  <%@ taglib prefix=*"fmt"* uri=*"http://java.sun.com/jsp/jstl/fmt"*%>  <%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"*%>  <%@ taglib prefix=*"fn"* uri=*"http://java.sun.com/jsp/jstl/functions"*%>  <%@page import=*"java.util.List"*%>  <%@page import=*"product.spring.demo.entities.Product"*%>  <h2>  <a href=*"/home"*>Home</a> > <a href=*"/products"*>Products</a>  </h2>  <p></p>  <table>  <thead>  <tr>  <th>Id</th>  <th>Name</th>  <th>Image</th>  <th>Price</th>  </tr>  </thead>  <tbody>  <c:forEach var=*"productItem"* items=*"*${productList}*"*>  <tr>  <td>${productItem.id}</td>  <td>${productItem.name}</td>  <td>${productItem.image}</td>  <td>${productItem.price}</td>  </tr>  </c:forEach>  </tbody>  </table> |

### Test

Insert dữ liệu vào bảng product trong CSDL của bạn



Chạy ứng dụng và xem kết quả



Tới bước này, bạn cần có ý tưởng tạo các View và link giữa các view với nhau.

### Có cần try-catch với Spring JPA?

Không, không cần thiết. Khi chạy ứng dụng thì Repository luôn báo lỗi, sẽ không bao giờ nó “nuốt” mất lỗi của bạn.

Nếu sử dụng try-catch trừ khi bạn Custom thông báo về người dùng cuối. Nên bạn có thể Custom bằng cách sử dụng try-catch trong Controller để thông báo với người dùng cuối.

### Project thời điểm kết thúc bước 8

Đường dẫn : <https://github.com/huygau91/ProductManagement.git>

Branch: spring-jpa

Các bạn cần làm là gì?

* Chúng ta cần tạo ra các view thêm sửa xóa (show dữ liệu đã show rồi)
* Link các view với nhau

Đây là kiến thức về tư duy, cách tạo view cũng đã hướng dẫn trước đó nên trong tài liệu sẽ không hướng dẫn ở đây

## Thêm sửa xóa show dữ liệu

## Spring Security

### Giới thiệu Spring Security

### Step 29: Add thư viện Spring Security

### Step 30: Tạo DB

### Step 32: Nhúng SQL trong application.properties