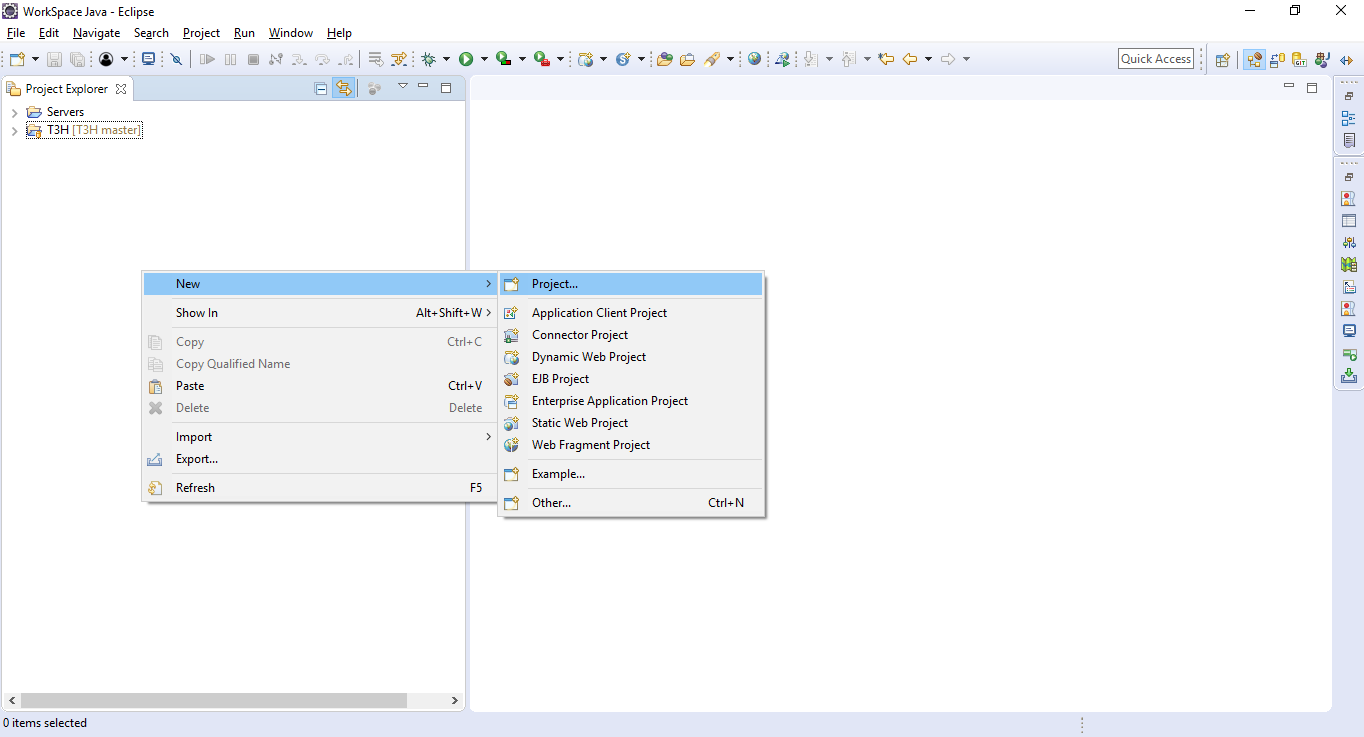
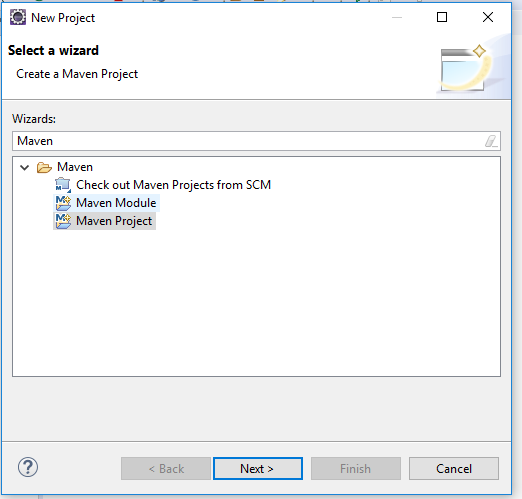
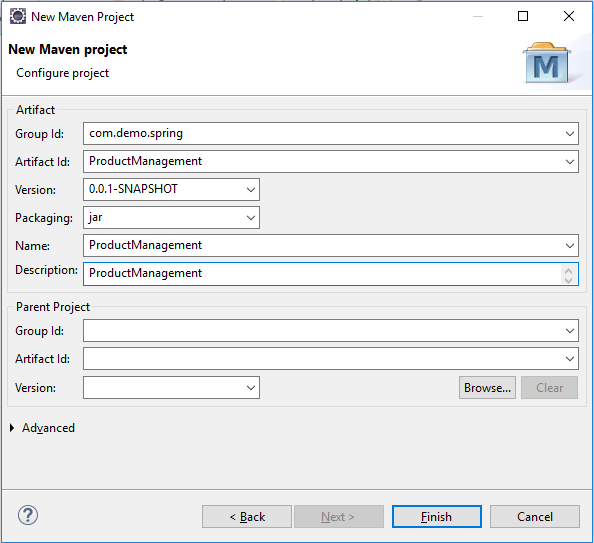
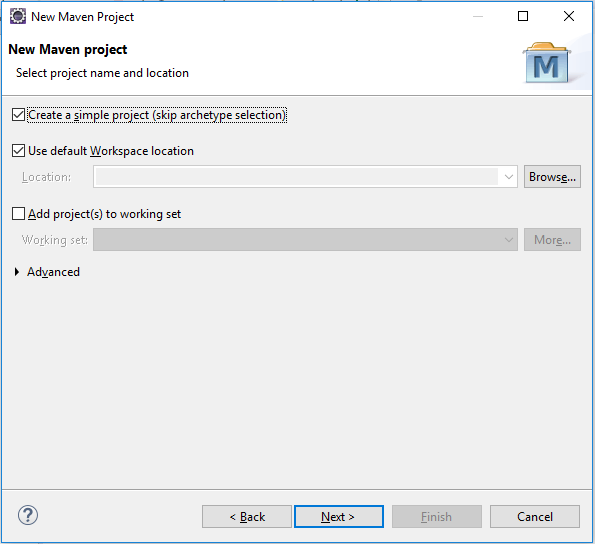
# Hello Worl với Spring Boot

## Tạo Project maven

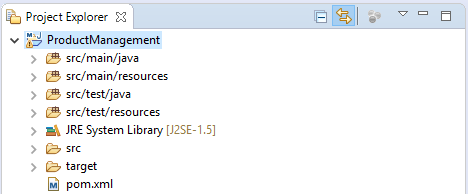
### Step1: Tạo Maven Project







Điền các thong số và chờ đợi Eclipse tạo project



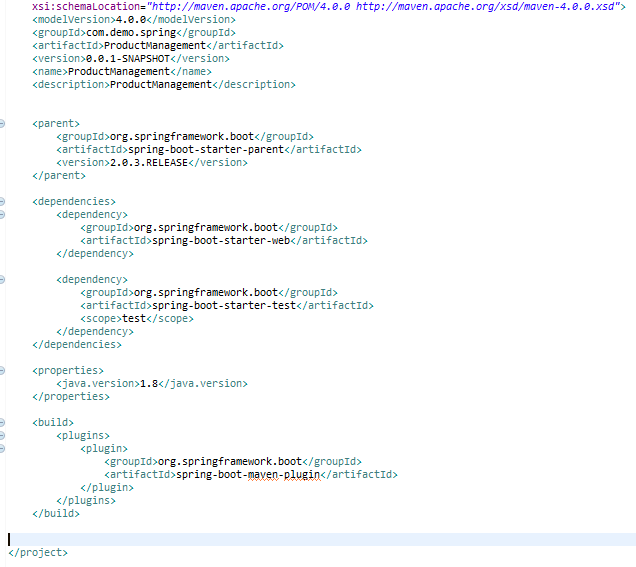
Kết quả Step 1 là hình trên. Nếu không thành công các bạn có thể tạo bằng tay thêm thư mục hoặc buil lại

### 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

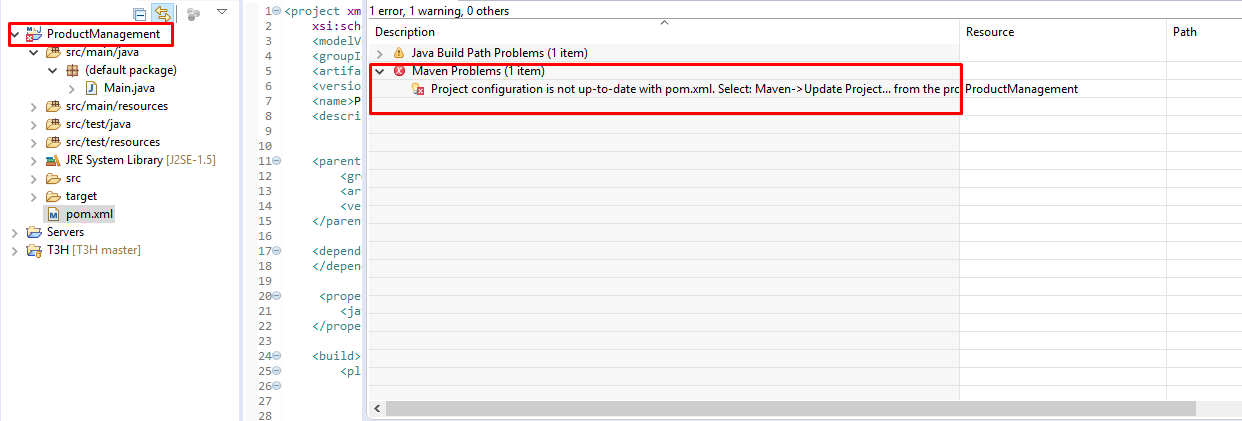
Tới file pom.xml và add thư viện và các thông số



Nội dung add mới

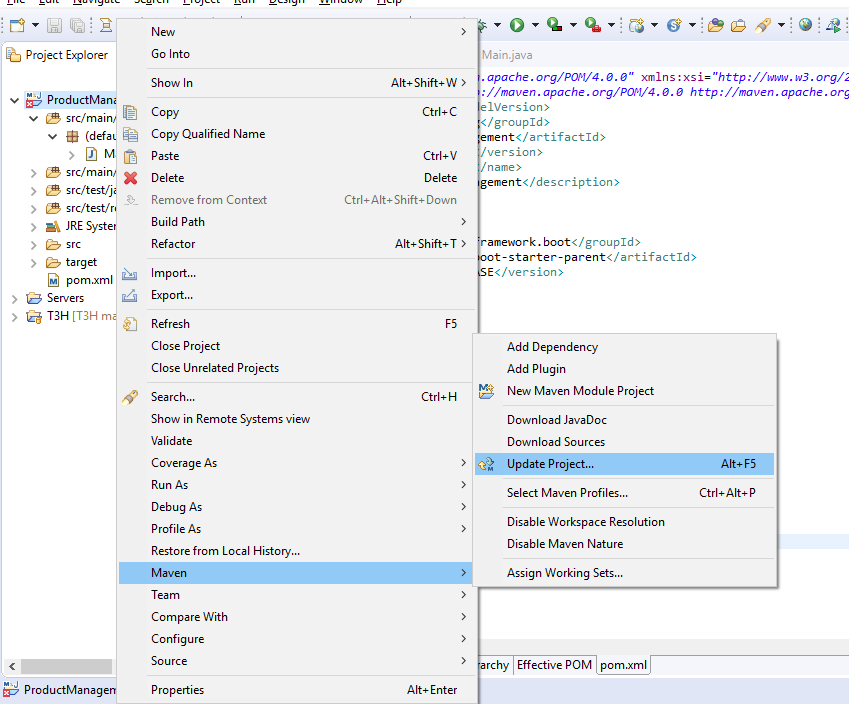
|  |
| --- |
| <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>2.0.3.RELEASE</version>  </parent>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <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> |

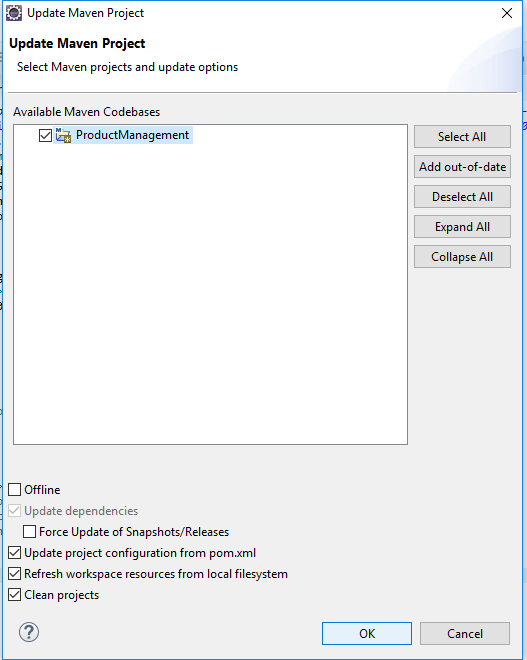
Hãy lưu lại và Build project lại



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

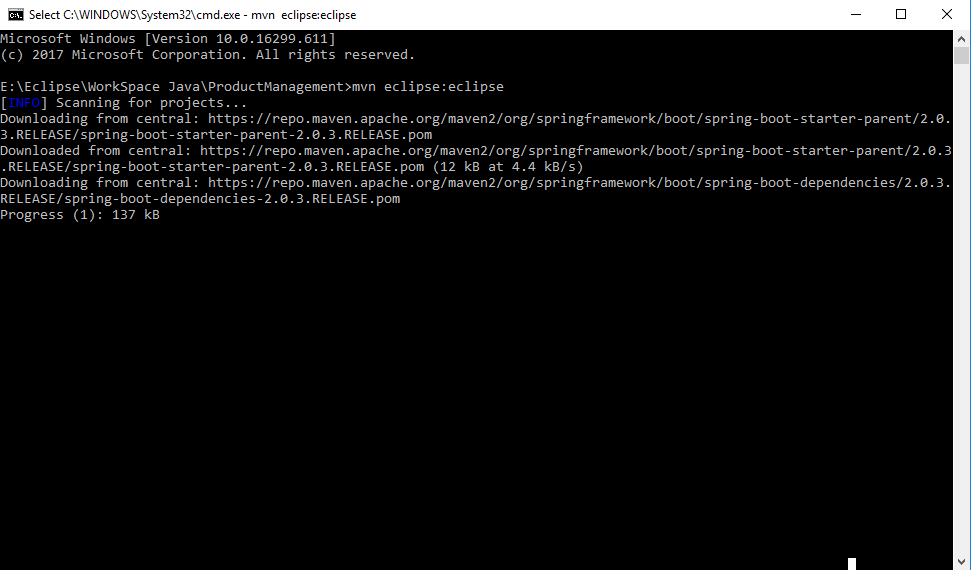
Chuột phải Project và Chọn như hình

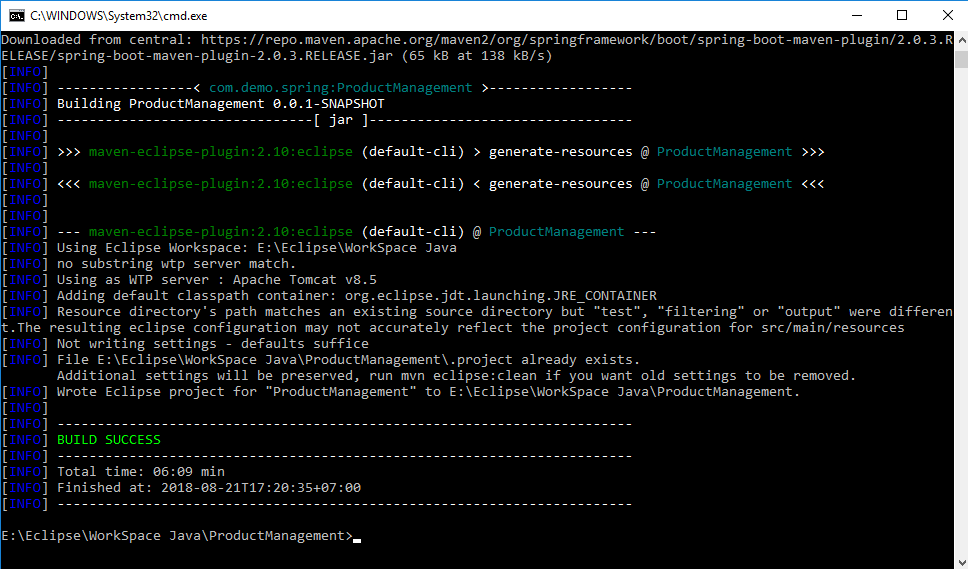




### Fix lỗi Step 2

* 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

## Tạo Demo với Project Spring Boot

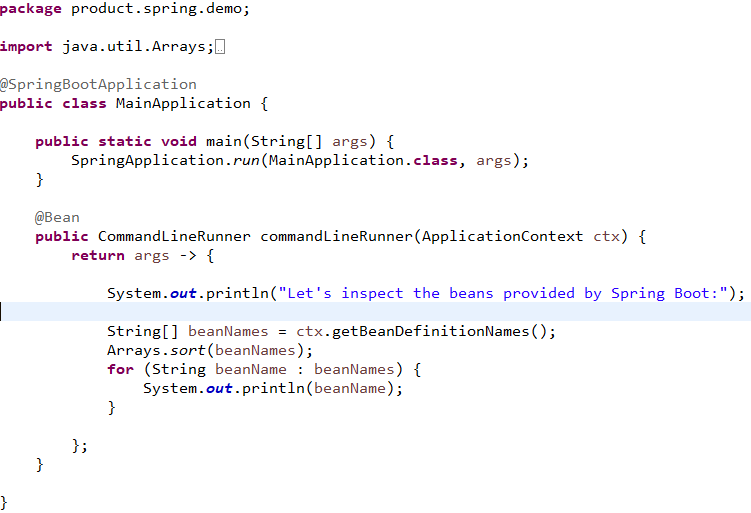
### Link tham khảo

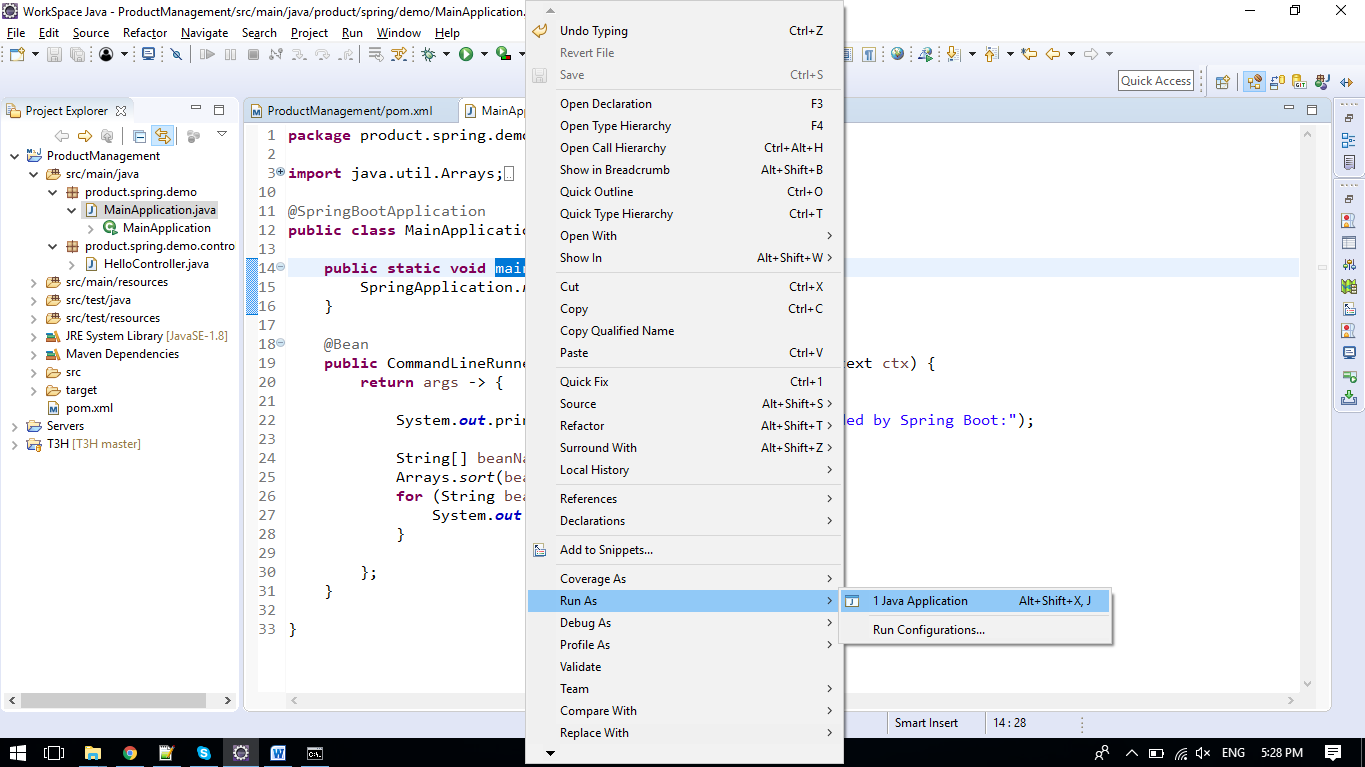
##### https://spring.io/guides/gs/spring-boot/

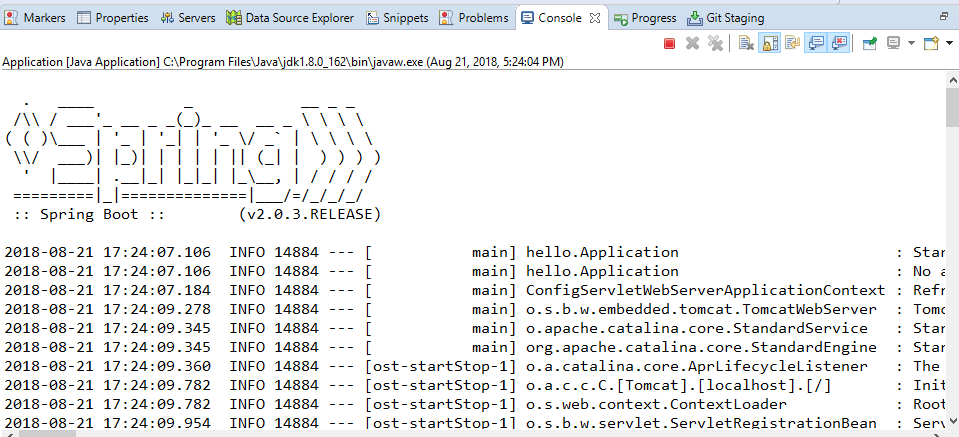
### Step 3 Tạo Main Application

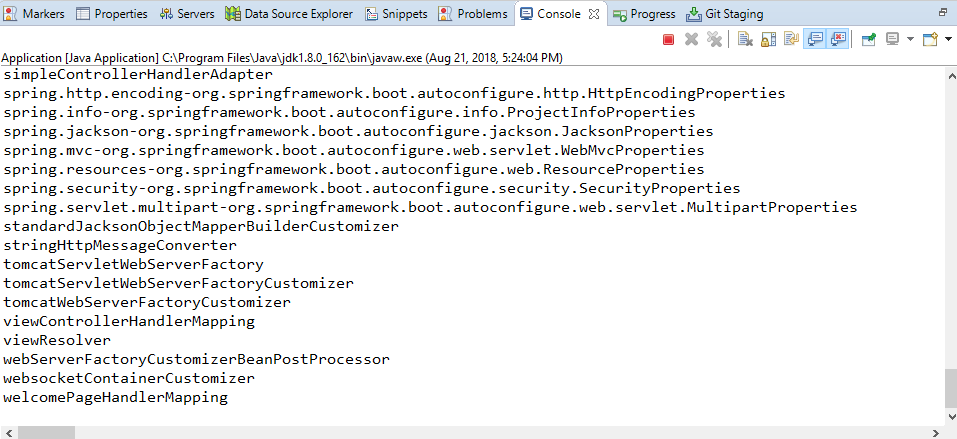
|  |
| --- |
| package product.spring.demo;  import java.util.Arrays;  import org.springframework.boot.CommandLineRunner;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  import org.springframework.context.ApplicationContext;  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

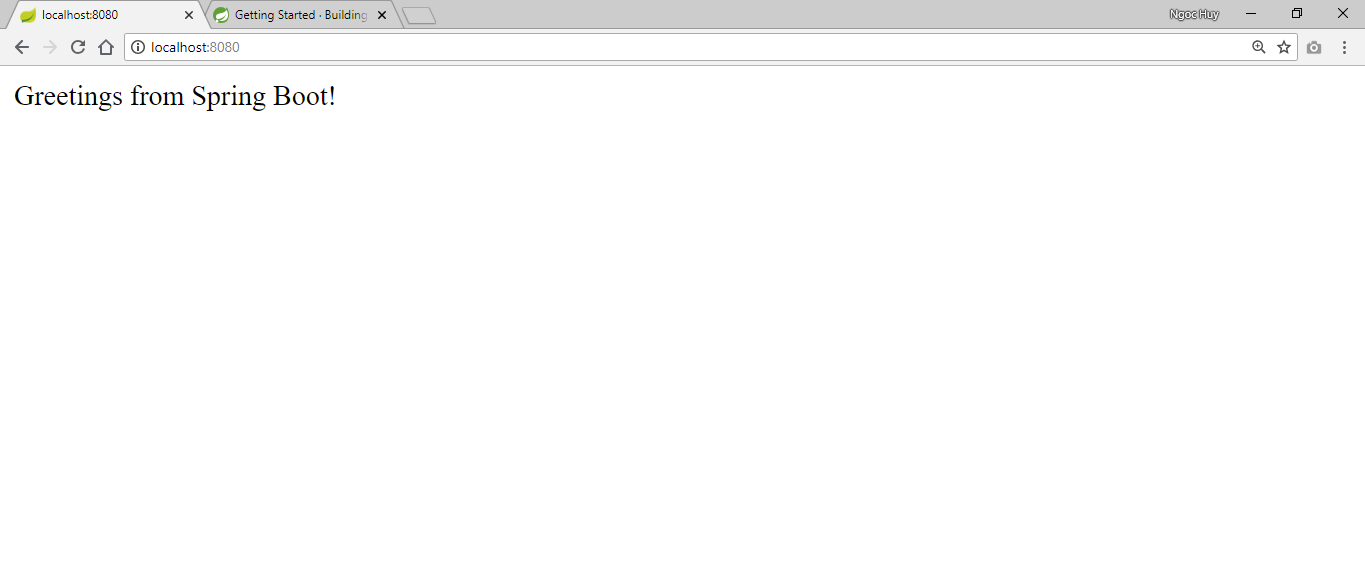
### Fix lỗi trong Step3

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;  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



Kết quả chạy: localhost:8080

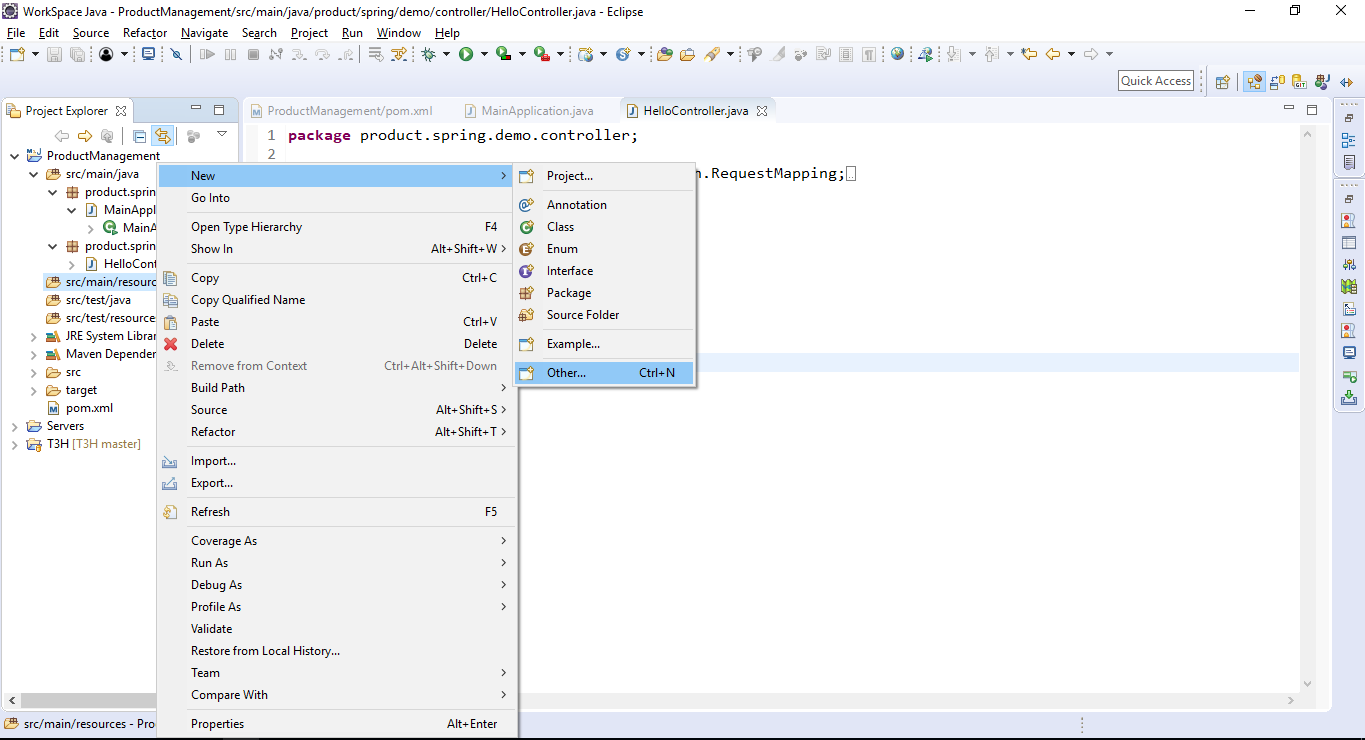
### Lưu ý Step4

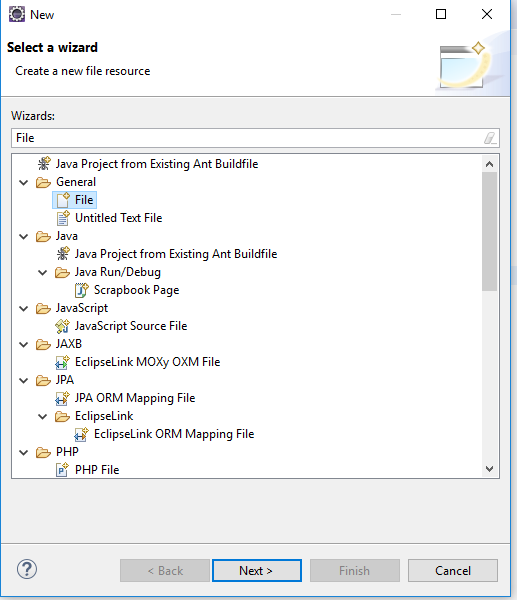
* Mặc định Spring Boot chạy với Tomcat là 8080
* Tomcat được nhúng trong Project, Khi Spring Boot chạy nó sẽ khởi động Tomcat này

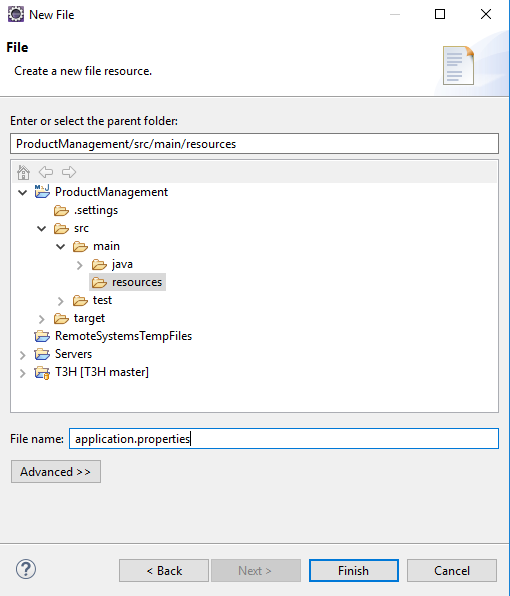
### Step 5: Tạo file cấu hình ứng dụng application.properties

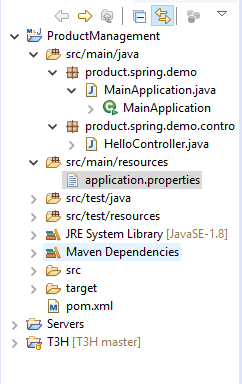
Cách hay nhất trong Spring Boot là sử dụng application.properties để config ứng dụng

Hãy làm theo hình sau. Chuột phải vào 



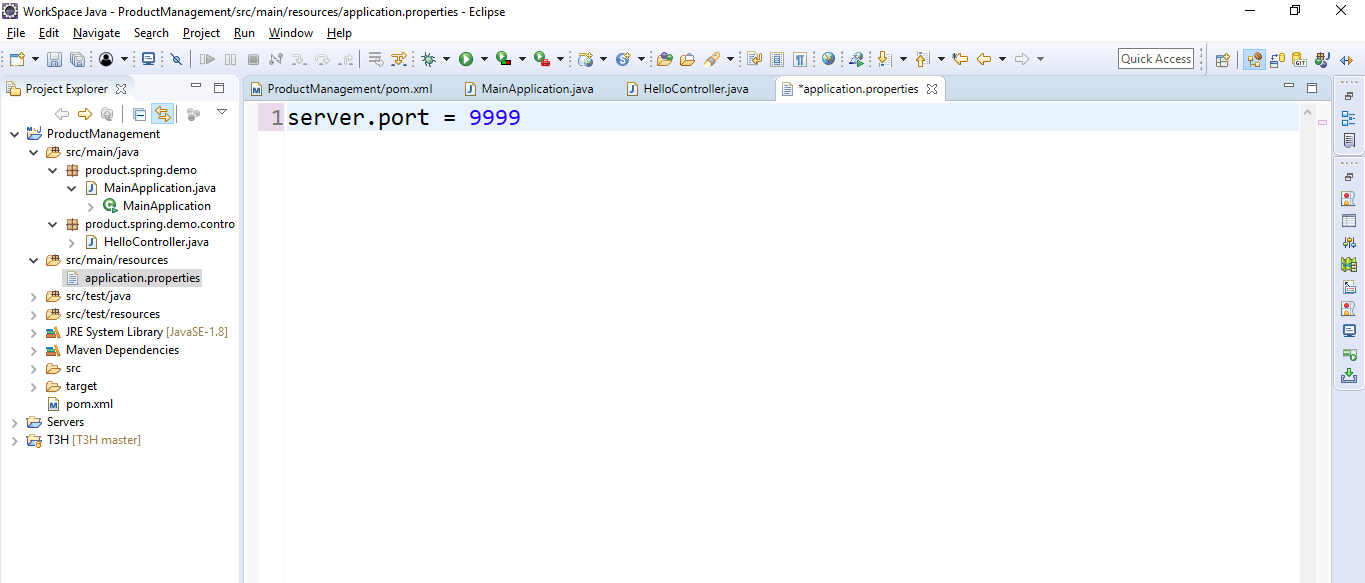






### Step 6: Cấu hình cổng cho ứng dụng trong application.properties

Hãy cấu hình port bạn muốn:

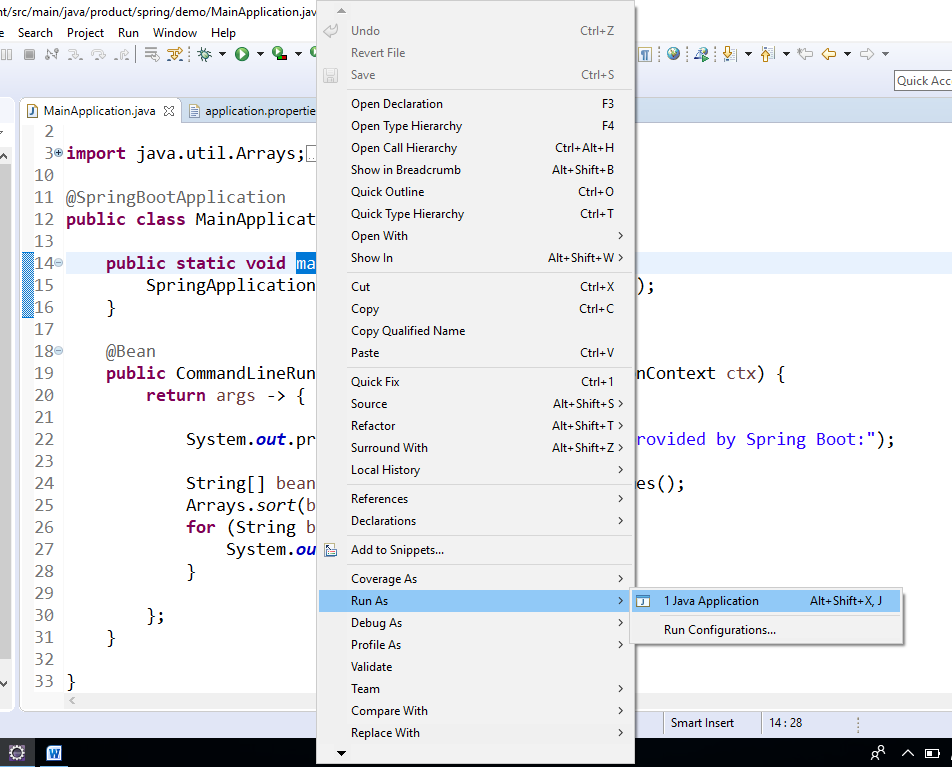


Cấu hình port 9999

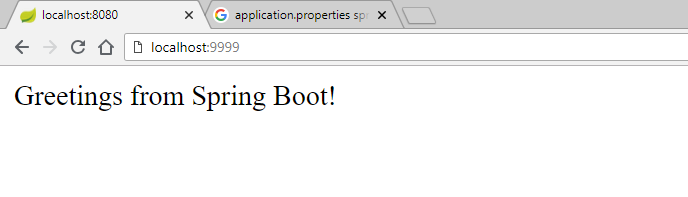
Lưu lại và khởi động và kiểm tra

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

Kết quả khi chạy



Kết quả



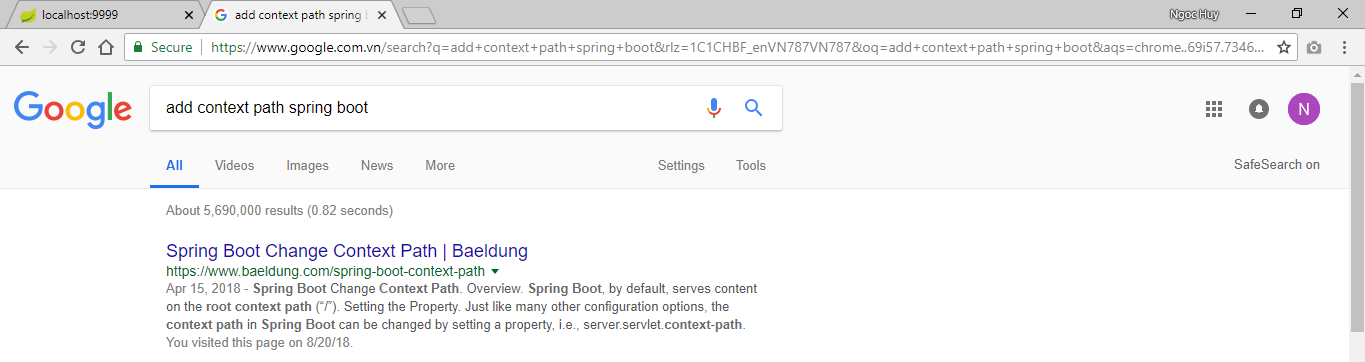
Kết quả

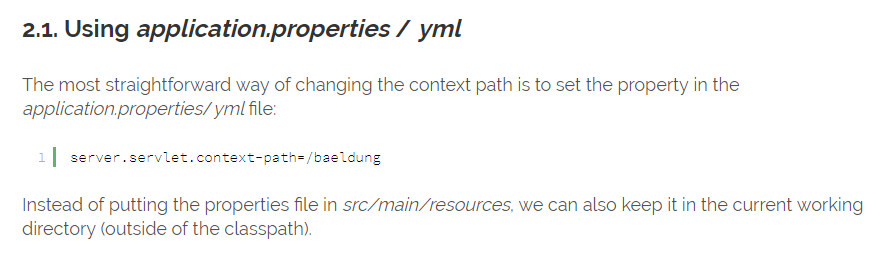
### Step7: Cấu hình context-path ứng dụng

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

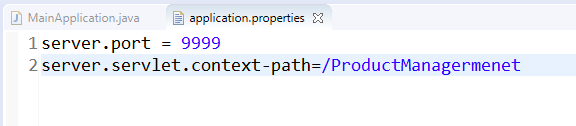
Nếu không cấu hình có thể hiểu là localhost:port là context path mặc định của ứng dụng Spring Boot

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



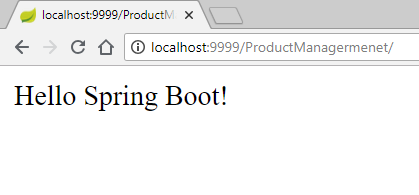
Chủ động tìm kiếm

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 không phải khởi động ứng dụng mỗi lần chạy. 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

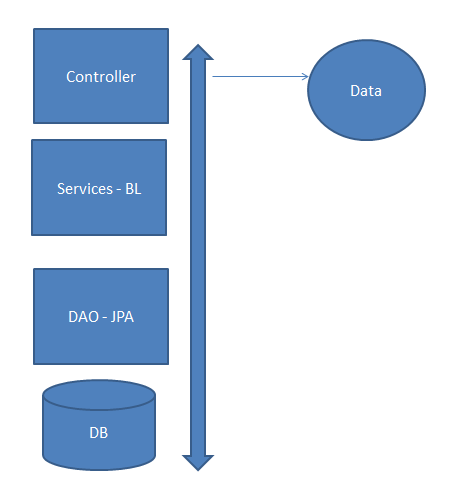
Với phần này chúng ta có thể coi project hiện tại là Hello với Spring MVC và Spring Boot

## 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



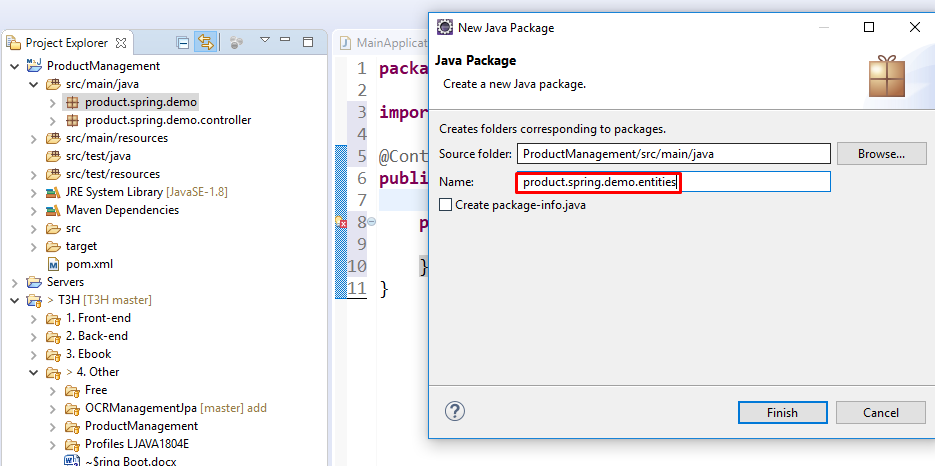
Mô hình giữa các lớp ứng dụng

### 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

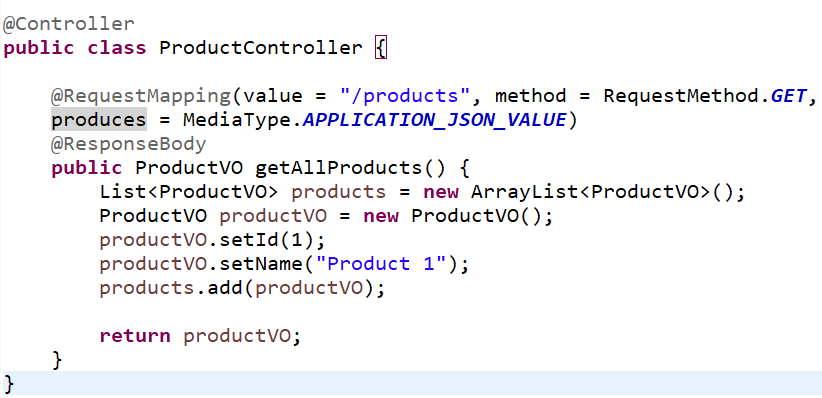
Hãy tham khảo link sau với RESTful service

##### <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





### Step 10: Giải thích

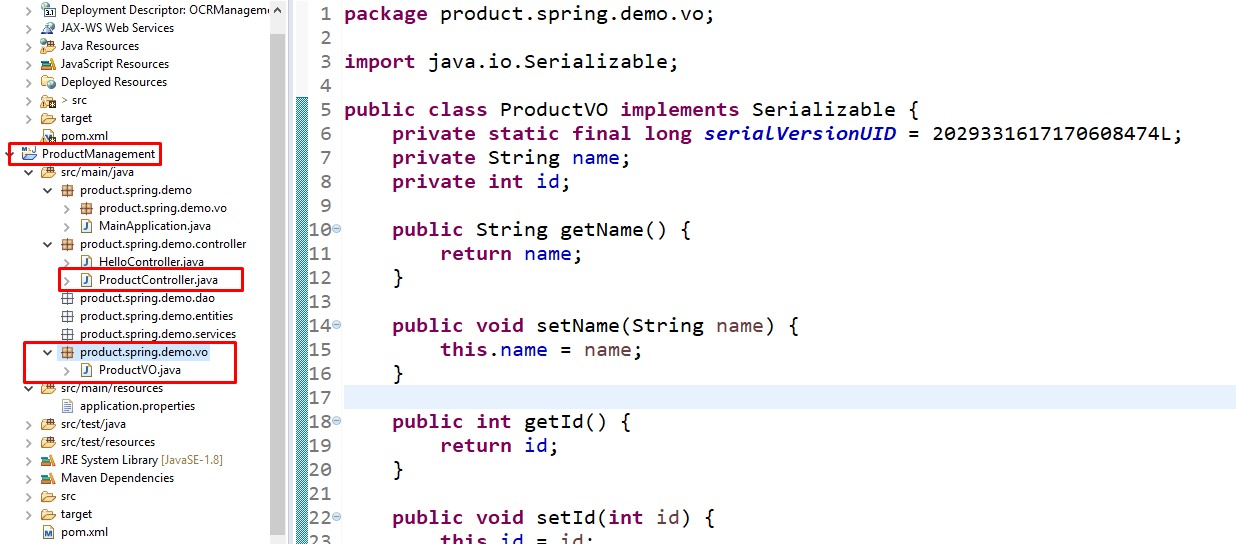
|  |  |
| --- | --- |
| @Controller | Để spring có thể scan quét được |
| @RequestMapping => value | Mapping URL: context path + value |
| @RequestMapping => method | Metho với URL trên |
| @RequestMapping => produces | Loại dữ dữ liệu được sử dụng |
| @ResponseBody | Dữ liệu trả về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

### Step 10: Lưu ý

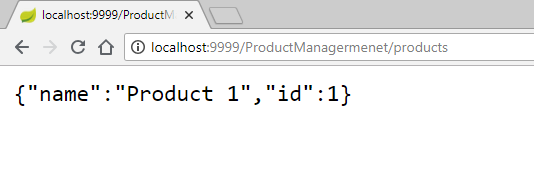
* **Dùng thì mới tạo** .Hãy tạo các controllers tạo các web services với khi chúng ta sử dụng. Nếu chúng ta không sử dụng hiện tại và tương lai thì cũng không nên tạo làm gì. Và code như vậy sẽ tạo ra rác trong ứng dụng
* Hãy tham khảo cách dùng với: **produces** và **consumes** với các method GET, POST trong Spring

### 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



### Step 11: Kết quả Kiểm tra service



### 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> |

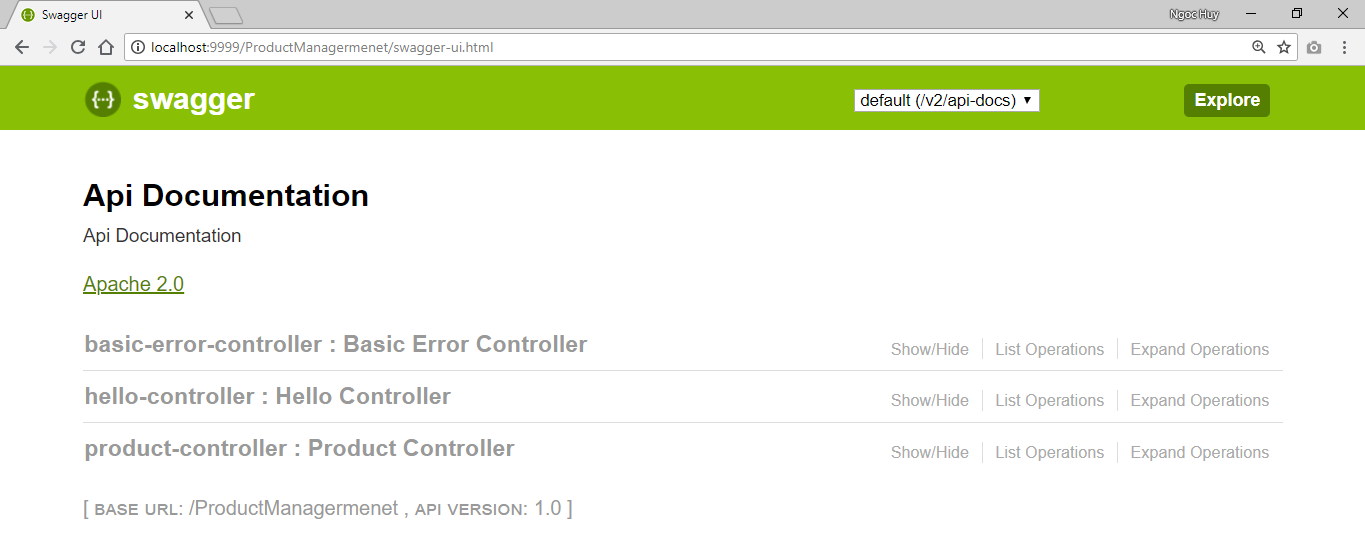
#### Build lại và Update Project

#### 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/ProductManagermenet/swagger-ui.html>

### Project thời điểm phần 4



## 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

|  |
| --- |
| <packaging>war</packaging> |

File pom.xml trong ứng ở Step 14 này

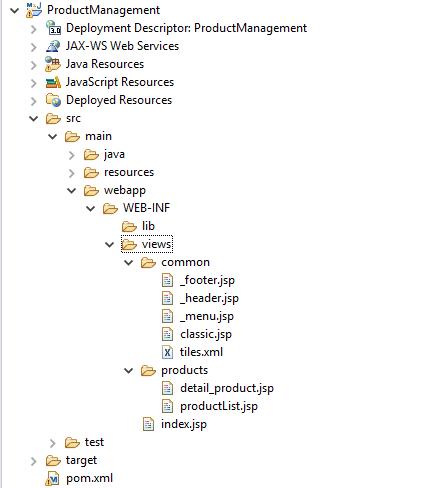
Khi add tới đây thì ứng dụng có thể build ra dạng war file để deploy trên bất kì server Java nào

#### 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”

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



Hãy Tạo thư mục các File như trên

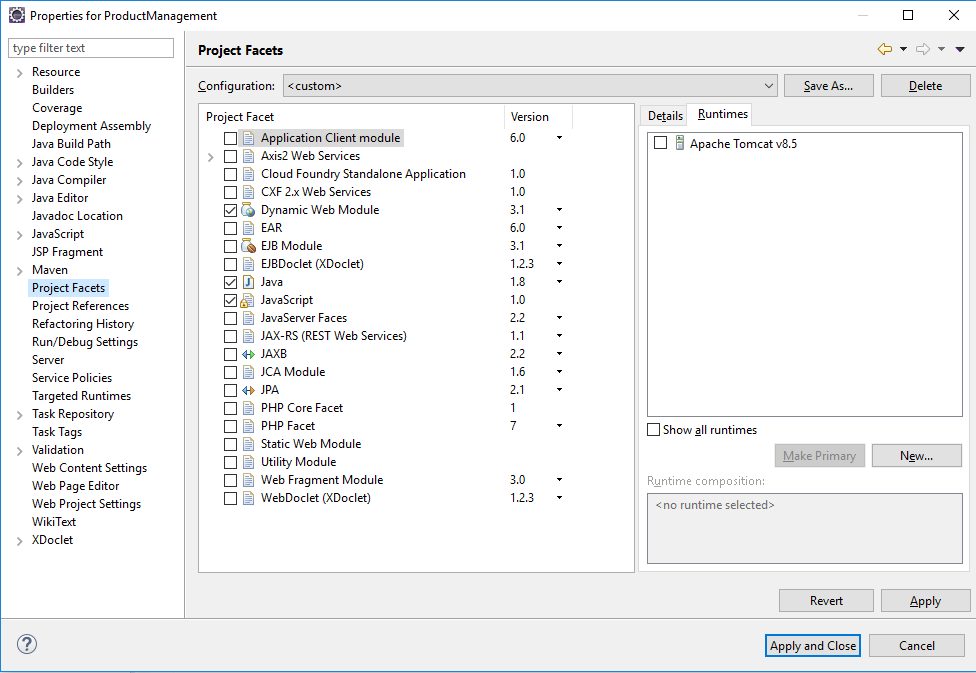
Các thành phần cần tạo:

* **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/classi.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**



### 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=/ProductManagermenet  #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 ở step14

### 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.definition

|  |
| --- |
| <!-- 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

### 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;  }  } |

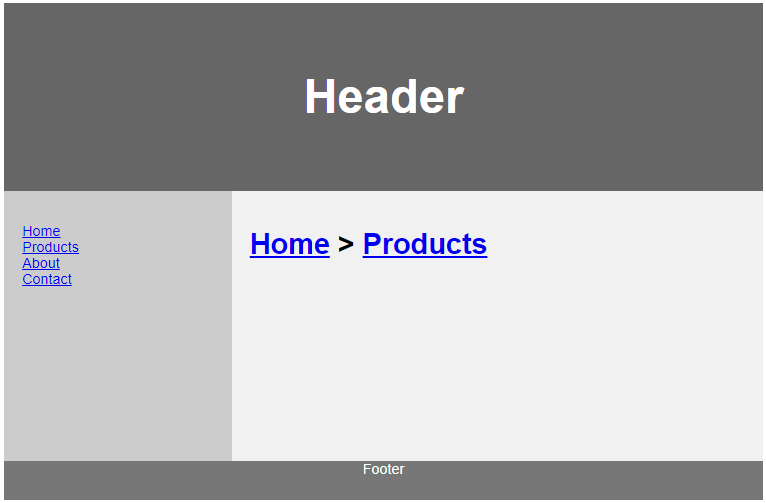
String[] defs = { "WEB-INF/views/common/tiles.xml" }; giúp chúng ta gọi tới tiles.xml

### Step 18: Chuẩn bị layput 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:

<https://www.w3schools.com/code/tryit.asp?filename=FURHCQELM8OR>

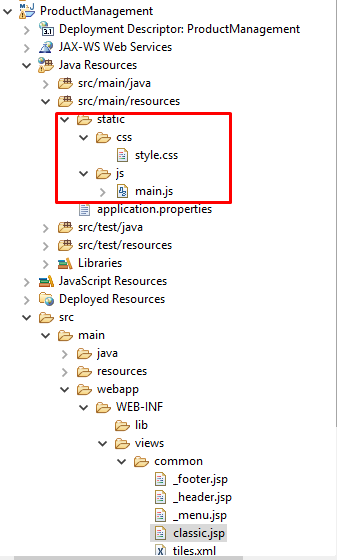


Tạo giao diện Template

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

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

#### 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 .

#### 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> |

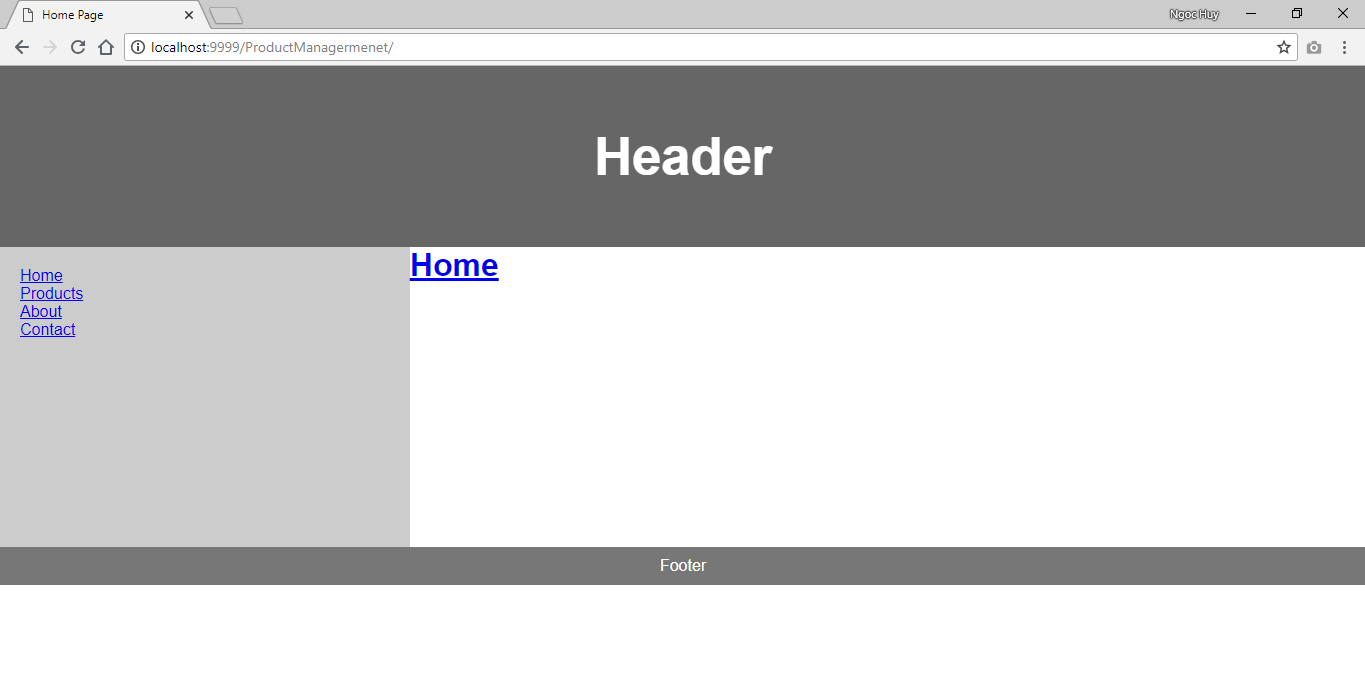
Lưu ý:

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

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

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

### Kết quả sử dụng Appche Tiles



## 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>



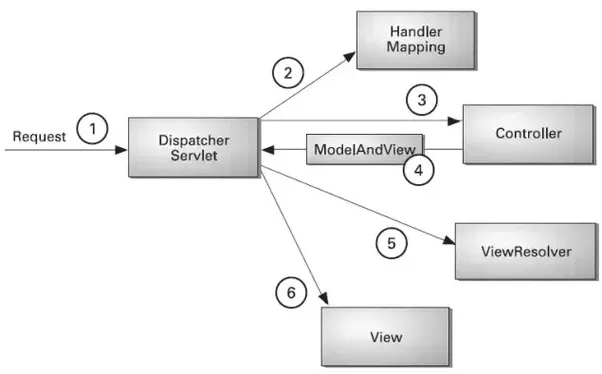
Mô hình MVC

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

|  |
| --- |
| **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 Controller

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

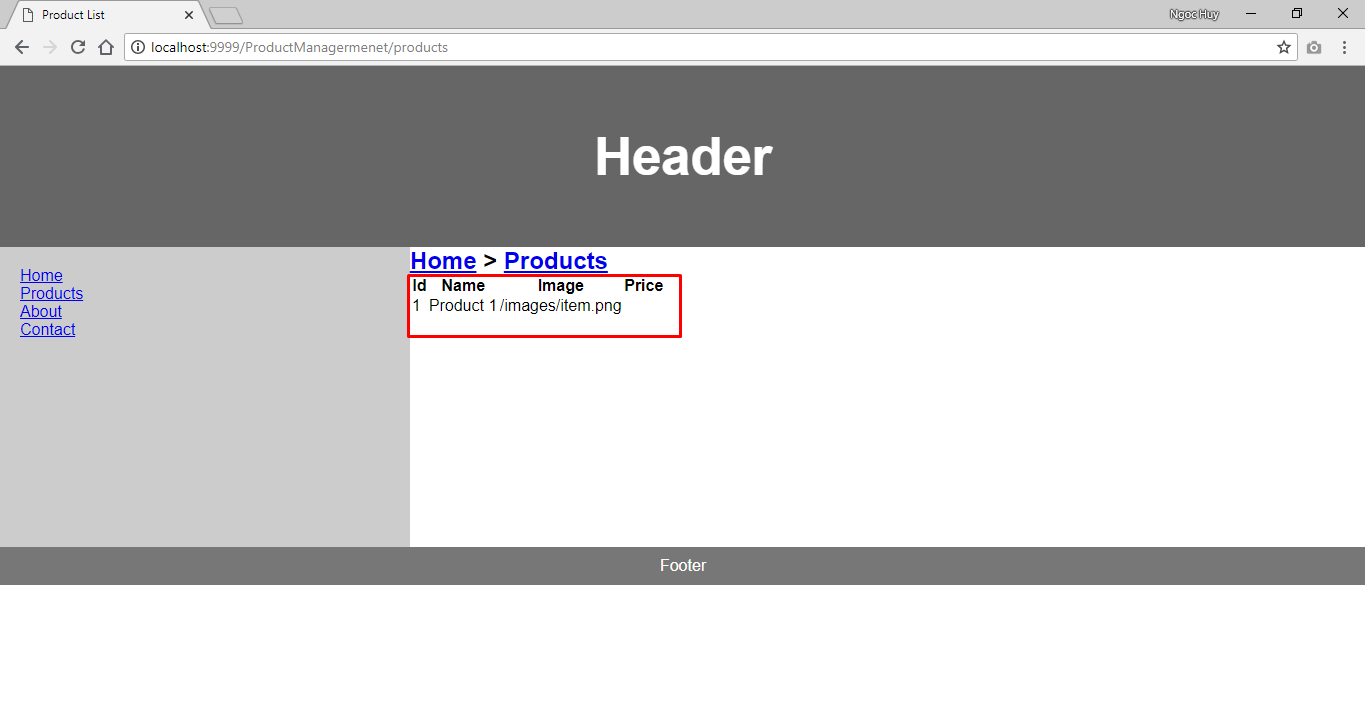
1. Kiểm trả tên view trả về trùng khớp với Appache Tiles config: tiles.xml như bước 20.3

### 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.

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

Đây 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**

### Step 22: Add thư viện Spring JPA