



# BUỔI 2 – SPRING MVC

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH  
CYBERSOFT.EDU.VN



## ❑ Spring Framework

- Framework là gì?
- Library là gì?
- Module là gì?
- Giới thiệu Spring Framework.
- Các module của Spring Framework.
- Spring MVC Xml Config.
- Spring MVC Annotation Config.
- Bài tập Spring MVC.

CYBERSOFT

ĐÀO TẠO CHUYÊN GIA LẬP TRÌNH

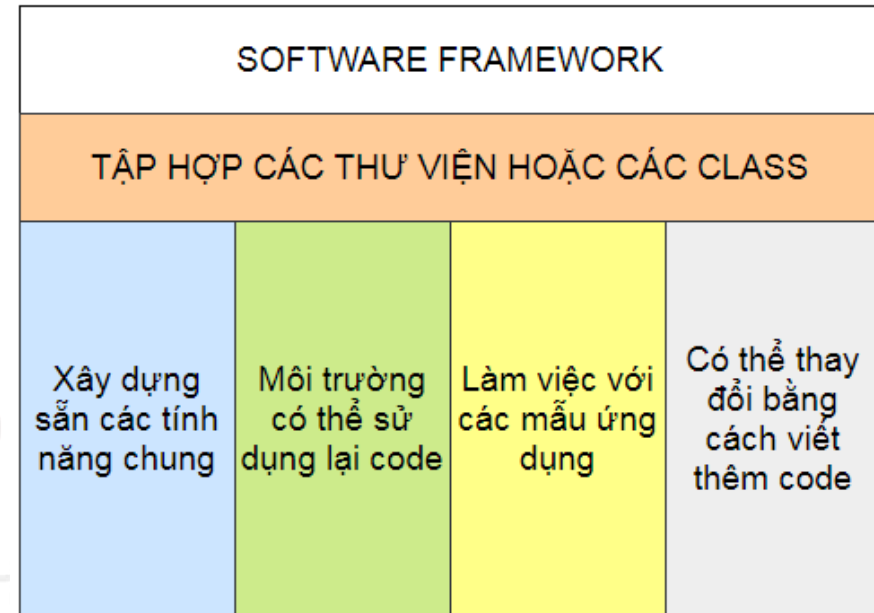
## ❑ Library là gì?

- Library (thư viện) là tập hợp các **chức năng** (function), các **lớp** (class) được **viết sẵn** để có thể **tái sử dụng** và thực hiện một **công việc cụ thể** nào đó.
  - Thư viện thường được tạo khi một **vấn đề** hay một **tác vụ** nào đó thường phát sinh với **tần suất cao**, khi đó các lập trình viên sẽ **đóng gói các giải pháp của mình thành một package** và chia sẻ nó thông qua một giao thức chia sẻ nào đó như: **npm**, **package**, **composer**, **github** ...
- ❖ Ví dụ: ReactJs, JQuery.

# Framework

## ❑ Framework là gì?

- **Framework** được định nghĩa là một **bộ khung** gồm một **tập hợp các thư viện phần mềm, các trình biên dịch hoặc các API** nhằm cung cấp một môi trường giúp tiết kiệm thời gian trong quá trình phát triển ứng dụng.
- **Framework** xây dựng sẵn các **tính năng chung**. Ví dụ các chức năng **đăng nhập, đăng ký tài khoản, kết nối CSDL,...**
- **Framework** giúp giảm thiểu tối đa **thời gian** và **công sức** phát triển ứng dụng.
- Cho phép ứng dụng kế thừa một **cấu trúc được chuẩn hóa**, đảm bảo **dễ dàng** trong vận hành và **bảo trì** sau này.



### ❖ Ví dụ:

- ✓ **Css** : Bootstrap, Pure, Materialize,...
- ✓ **Javascript**: Angular JS, Vue Js, NodeJs.
- ✓ **C#** : .Net Framework
- ✓ **Java**: Spring, Hibernate, Struts, ...
- ✓ **PHP**: CakePHP, Laravel, CodeIgniter, Zend

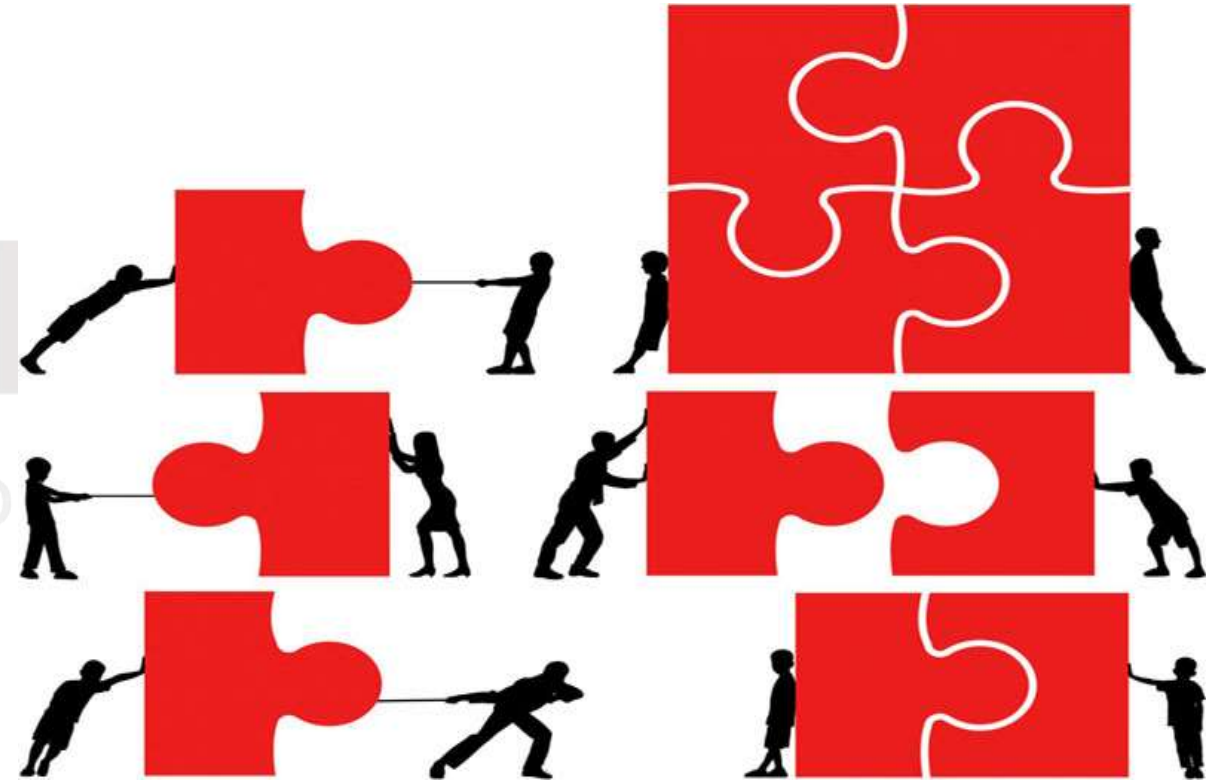
# Framework và Library khác gì?

- ❑ Điều khác biệt lớn nhất giữa **thư viện** và **framework**, khi sử dụng **thư viện** mã code của bạn sử dụng **thư viện**, khi dùng **framework** thì mã code của bạn được **framework** sử dụng.
- ❑ Các mã code trong **framework** sẽ gọi đến **mã code của bạn**, còn với **thư viện** lập trình thì **mã lệnh của bạn** gọi đến các mã lệnh của **thư viện lập trình**.

# Module

## ❑ Module là gì?

- **Module** là một thành phần giữ một **chức năng cụ thể** trong một hệ thống hay một **phần mềm** của **máy tính** cụ thể.
- Chia nhỏ từng module giúp ta **dễ dàng quản lý công việc** hơn.
- Chia thành **nhiều module khác nhau** và có **chức năng khác nhau**. Khi cần **thay đổi** ở module nào chỉ cần **chỉnh sửa** ở module đó mà **không ảnh hưởng** tới các module khác.



# Spring Framework

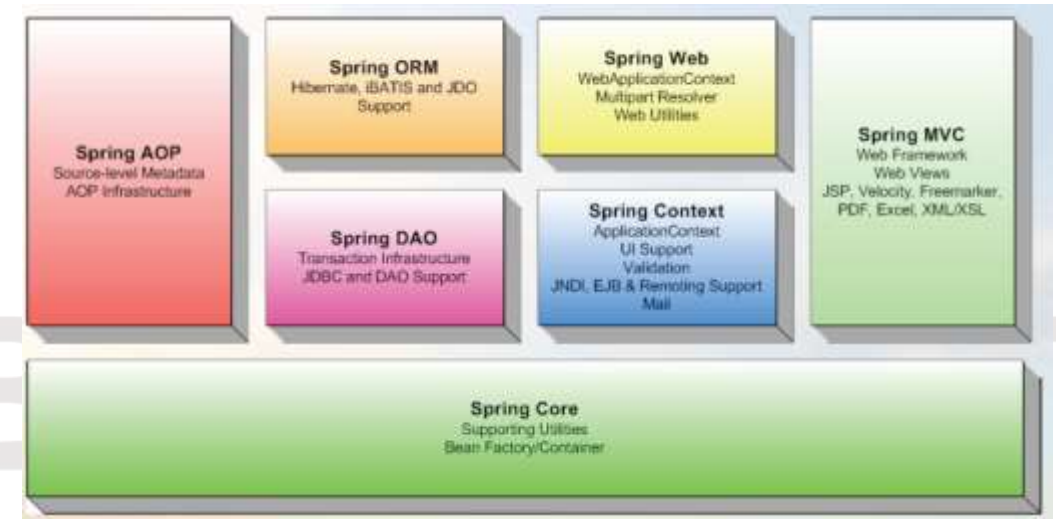
## ❑ Spring Framework

- ✓ **Spring Framework** là một framework **mã nguồn mở** được phát hành phiên bản đầu tiên bởi Rod Johnson năm 2002.
- ✓ **Spring Framework** giúp tạo các ứng dụng có hiệu năng cao, dễ kiểm thử, sử dụng lại code...
- ❖ **Spring Framework** thường dựa trên các **quan điểm** như sau:
  - ✓ Đơn giản hóa công việc phát triển thông qua việc **sử dụng các POJO** (Plain Old Java Object).
  - ✓ Nới lỏng ràng buộc giữa các thành phần thông qua việc sử dụng **Dependency Injection**.
  - ✓ Tiếp cận lập trình khai báo bằng cách sử dụng các **quy tắc** (convention) và các **khía cạnh** (aspect) chung.

# Spring Module

## ❖ Spring chia làm 7 module chính:

- **Core Container** : Cung cấp tính năng IoC, Dependency Injection,...
- **Spring Context**: Hỗ trợ đa ngôn ngữ, cung cấp các service như e-mail, validation, hỗ trợ setting/getting giá trị.
- **Spring AOP (Aspect-Oriented)**: Hỗ trợ cài đặt lập trình hướng khía cạnh.
- **Spring DAO**: Cung cấp khả năng kết nối giao tiếp với database.
- **Spring ORM**: Spring có thể tích hợp với một vài ORM framework để cung cấp Object Relational Mapping bao gồm: JDO, Hibernate, OJB và iBatis SQL Maps.

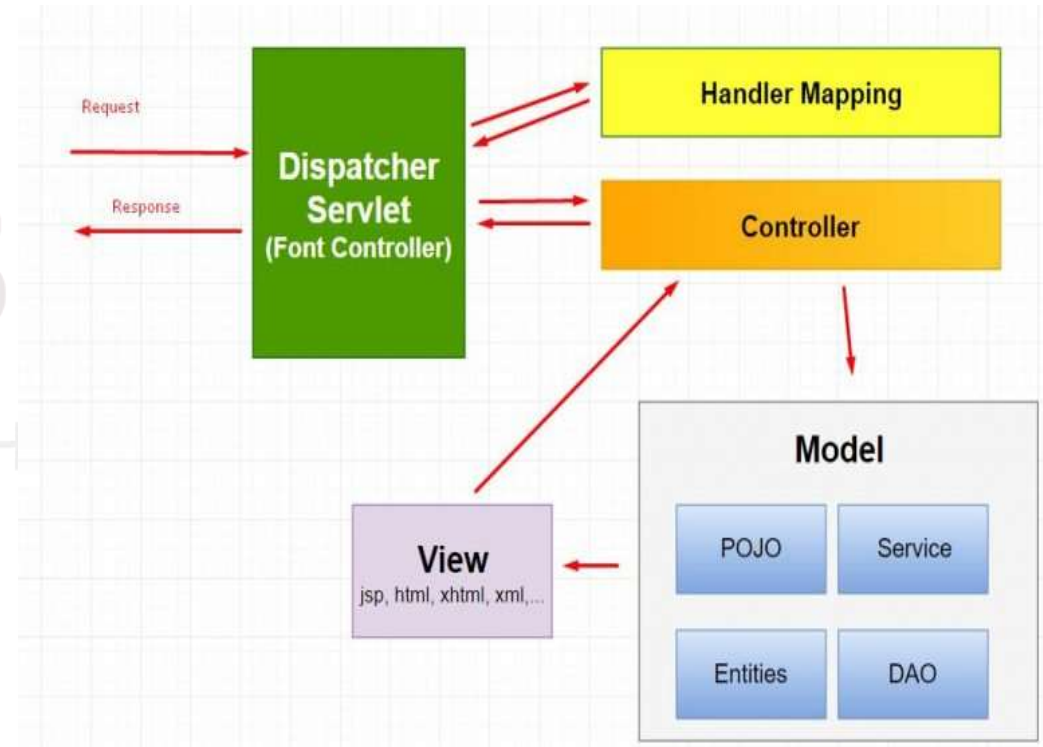


- **Spring Web**: Cung cấp context cho các ứng dụng web, làm giảm bớt các công việc điều khiển nhiều request và gán các tham số của request vào các đối tượng domain.
- **Spring MVC**: Cài đặt đầy đủ đặc tính của MVC pattern để xây dựng các ứng dụng Web.

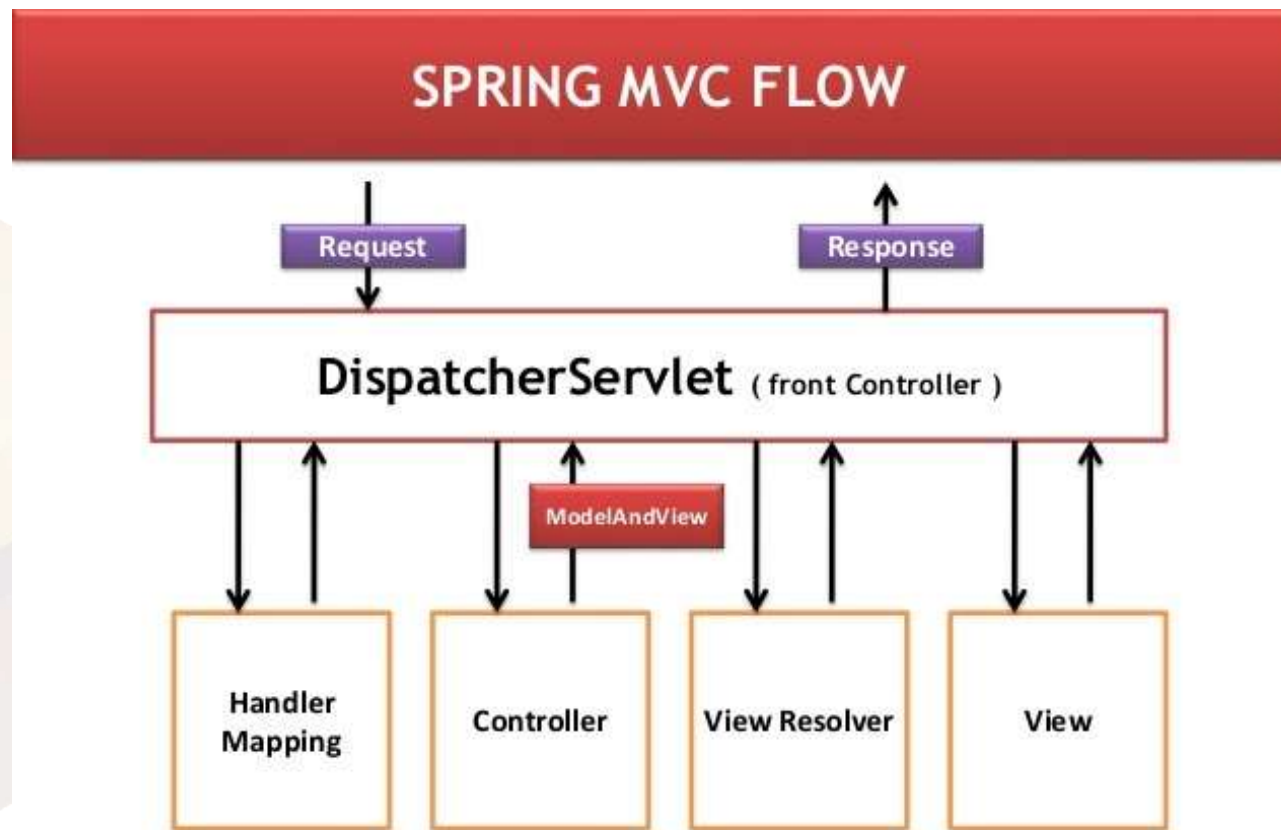


# Spring MVC

- ❑ **Spring MVC** là một module được sử dụng để xây dựng ứng dụng web theo mô hình MVC .
- ❑ **Spring MVC** giúp tạo ứng dụng web **nhANH chóng, đơn giản, dễ hiểu, mạnh mẽ, an toàn, ổn định** và chạy trên **mọi môi trường** Window, Linux, Unix, MacOS,...
- ❖ **Trong mô hình MVC:**
  - ✓ **Model:** là các file POJO, Entities, Service xử lý logic , DAO thực hiện kết nối và truy vấn dữ liệu database.
  - ✓ **View:** là các file JSP, html...
  - ✓ **Controller:** là Dispatcher Controller, Handler Mapping, Controller – thực hiện điều hướng các request.



# Mô hình hoạt động



1. Bất kỳ **request** nào tới ứng dụng web đều sẽ được gửi tới **Front Controller** (Dispatcher Servlet).
2. **Front Controller** sẽ sử dụng **Handler Mapping** để biết được **Controller** nào sẽ xử lý **request** đó.

# Xử lý request trong Spring MVC



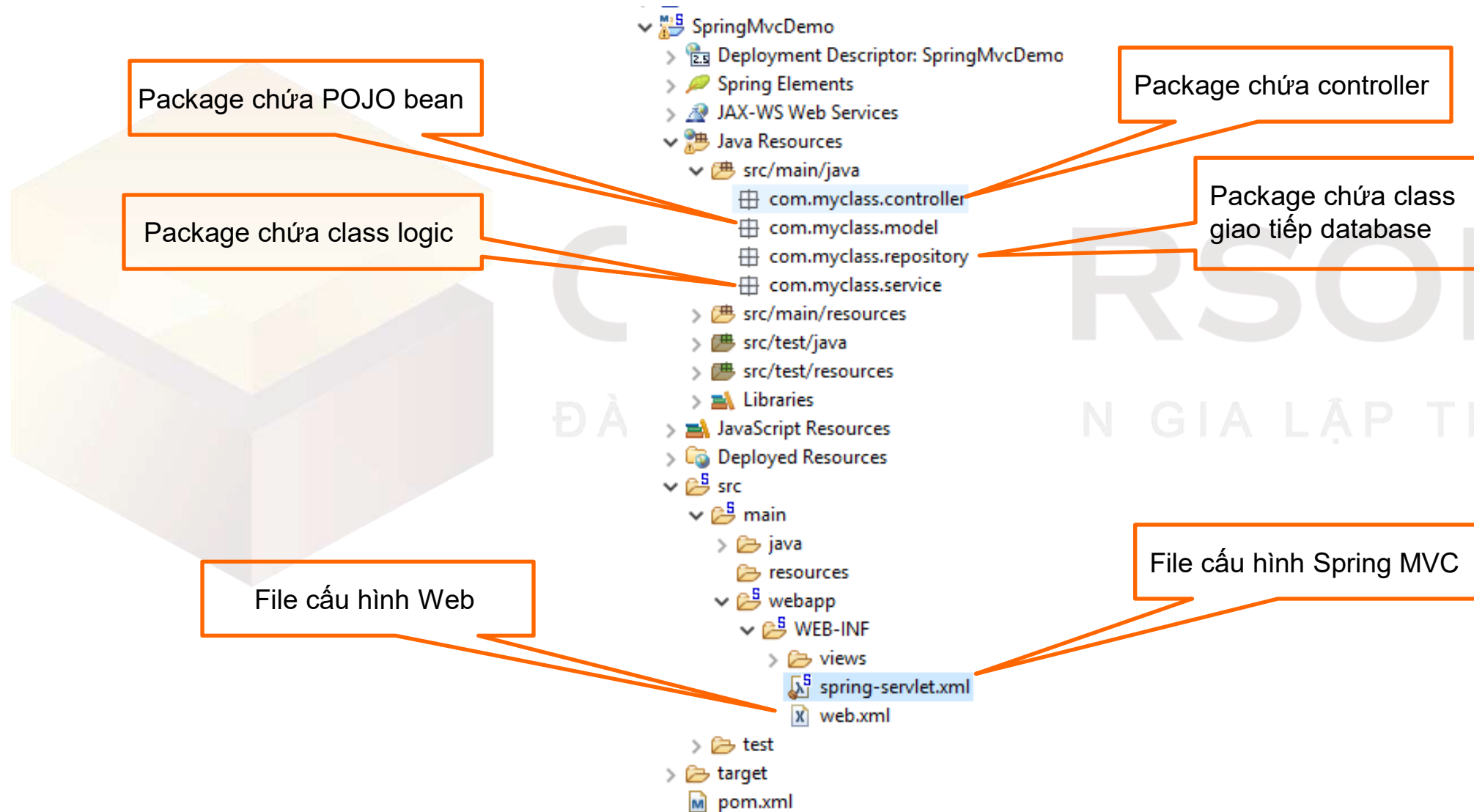
3. **Controller** nhận request, gọi tới các class **Service** thích hợp để xử lý yêu cầu.
4. Sau khi xử lý xong, **Controller** sẽ nhận model từ tầng **Service** hoặc tầng **DAO**.
5. **Controller** gửi model vừa nhận được tới **Dispatcher Servlet**.
6. **Dispatcher Servlet** sẽ tìm các mẫu view, sử dụng **View Resolver** và truyền model vào nó.
7. View template, model, view page được build và gửi trả lại **Dispatcher Servlet**.
8. **Dispatcher Servlet** gửi một page view tới trình duyệt để hiển thị nó cho người dùng.

# Cấu hình dự án (Xml Config)



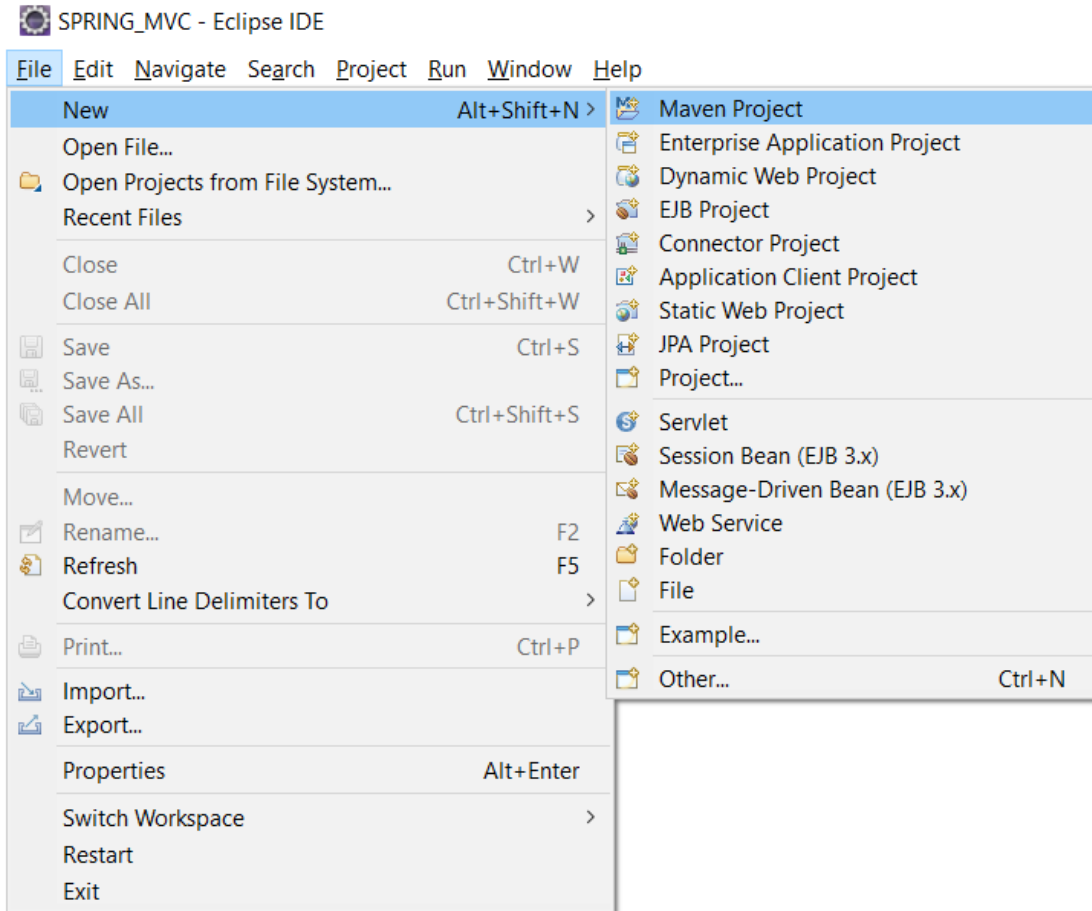
- ❑ **Bước 1:** Tạo dự án, cấu trúc thư mục theo mô hình MVC.
- ❑ **Bước 2:** Tải thư viện Spring MVC cho dự án.
- ❑ **Bước 3:** Tạo controller.
- ❑ **Bước 4:** Viết các phương thức trả về view.
- ❑ **Bước 5:** Tạo views cho dự án (các file jsp đặt trong thư mục views).
- ❑ **Bước 6:** Cấu hình Dispatcher Servlet (web.xml).
  - ✓ Cấu hình Url Mapping và khai báo đường dẫn tới file cấu hình Spring MVC.
- ❑ **Bước 7:** Cấu hình Spring MVC (spring-servlet.xml).
  - ✓ Cấu hình ViewResolver (prefix và suffix).
  - ✓ Kích hoạt Annotation.
  - ✓ Cấu hình Component Scan.

# Cấu trúc thư mục

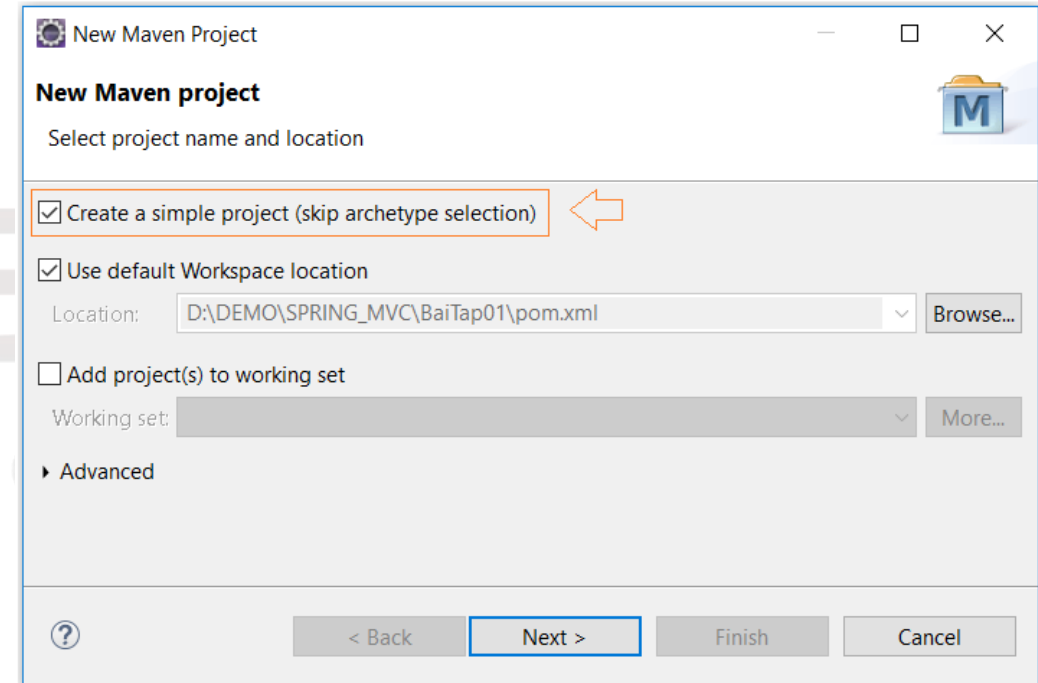


# Tạo dự án

## ❑ File → New → Maven Project



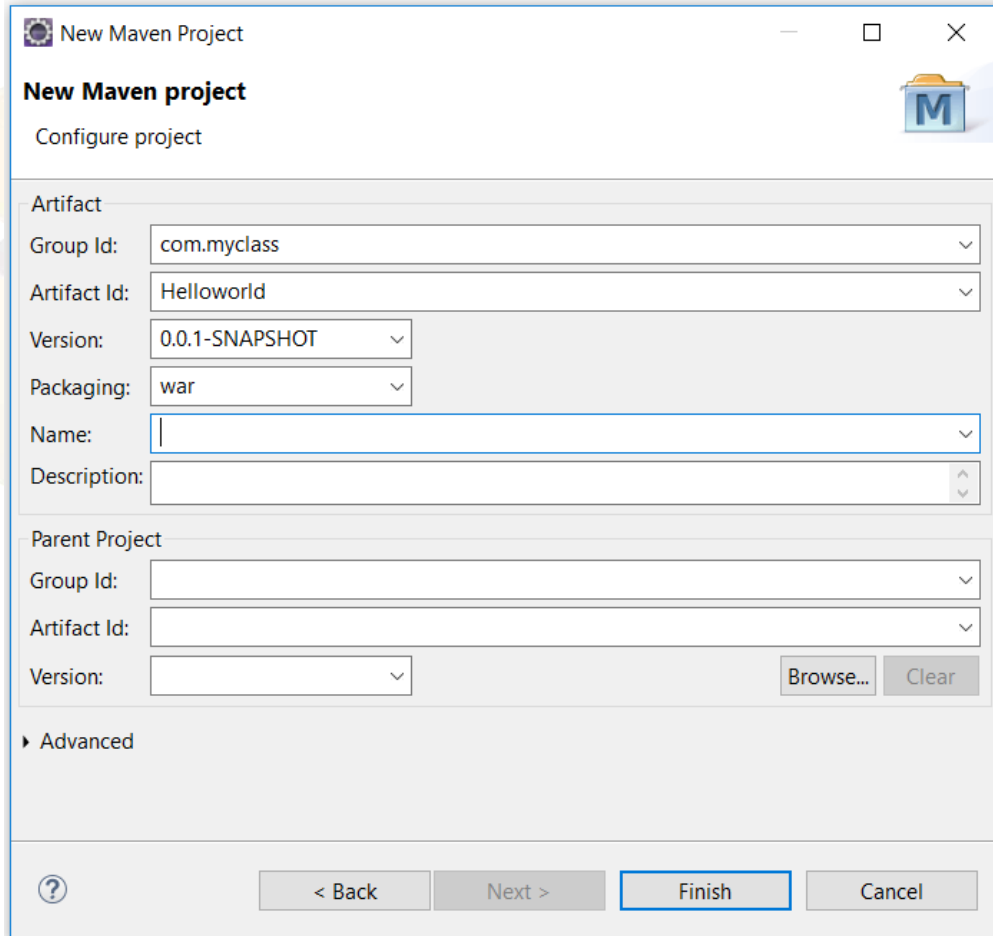
## ❑ Tích chọn Create a simple



# Tạo dự án

❑ Package: chọn **war** (Dành cho ứng dụng web)

❑ Thư viện sử dụng



New Maven Project

New Maven project

Configure project

Artifact

Group Id: com.myclass

Artifact Id: Helloworld

Version: 0.0.1-SNAPSHOT

Packaging: war

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5.RELEASE</version>
  </dependency>
</dependencies>
```

# Cấu hình DispatcherServlet

## ❑ Cấu hình web (web.xml)

```
web.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/200
3   <display-name>Java5</display-name>
4 <!-- DispatcherServlet -->
5 <servlet>
14 <servlet-mapping>
18
19 <!-- CharacterEncodingFilter -->
20 <filter>
32 <filter-mapping>
36
37 <welcome-file-list>
38   <welcome-file>index.jsp</welcome-file>
39 </welcome-file-list>
40 </web-app>
```

Khai báo  
DispatcherServlet

Khai báo  
CharacterEncodingFilter



# Cấu hình DispatcherServlet

## ❑ Khai báo dispatcherServlet (web.xml)

```
<servlet>
  <servlet-name>springDispatcherServlet</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-servlet.xml</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
```

Chỉ định file cấu hình Spring MVC (Chương trình sẽ hiểu bắt đầu tìm kiếm từ thư mục webapp).

```
<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
  <servlet-name>springDispatcherServlet</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

Tất cả các URL bắt đầu bằng / đều được DispatcherServlet tiếp nhận và xử lý

# Cấu hình DispatcherServlet

## ❑ Khai báo CharacterEncodingFilter

```
<filter>
  <filter-name>encodingFilter</filter-name>
  <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
  <init-param>
    <param-name>forceEncoding</param-name>
    <param-value>true</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>encodingFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

CharacterEncodingFilter cho phép ứng dụng web làm việc với utf-8 (tiếng Việt)

# Cấu hình Spring MVC

## ❑ Cấu hình Spring MVC

```
<!-- Cấu hình Spring MVC Annotation -->
```

```
<context:annotation-config />  
<mvc:annotation-driven />
```

Cho phép sử dụng  
Annotation trong ứng  
dụng Spring

```
<!-- Cấu hình package chứa các controller -->
```

```
<context:component-scan base-package="com.myclass.controller"/>
```

Chỉ rõ gói chứa các  
Controller. Sử dụng dấu  
phẩy để phân cách các gói

```
<!-- Cấu hình ViewResolver -->
```

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
```

```
  <property name="prefix" value="/WEB-INF/views/"></property>
```

```
  <property name="suffix" value=".jsp"></property>
```

```
</bean>
```

View = prefix + viewname + suffix

# Luồng điều khiển request

## Request

← → ↻ 🏠 🌐 localhost:8080/SpringMvcDemo/home

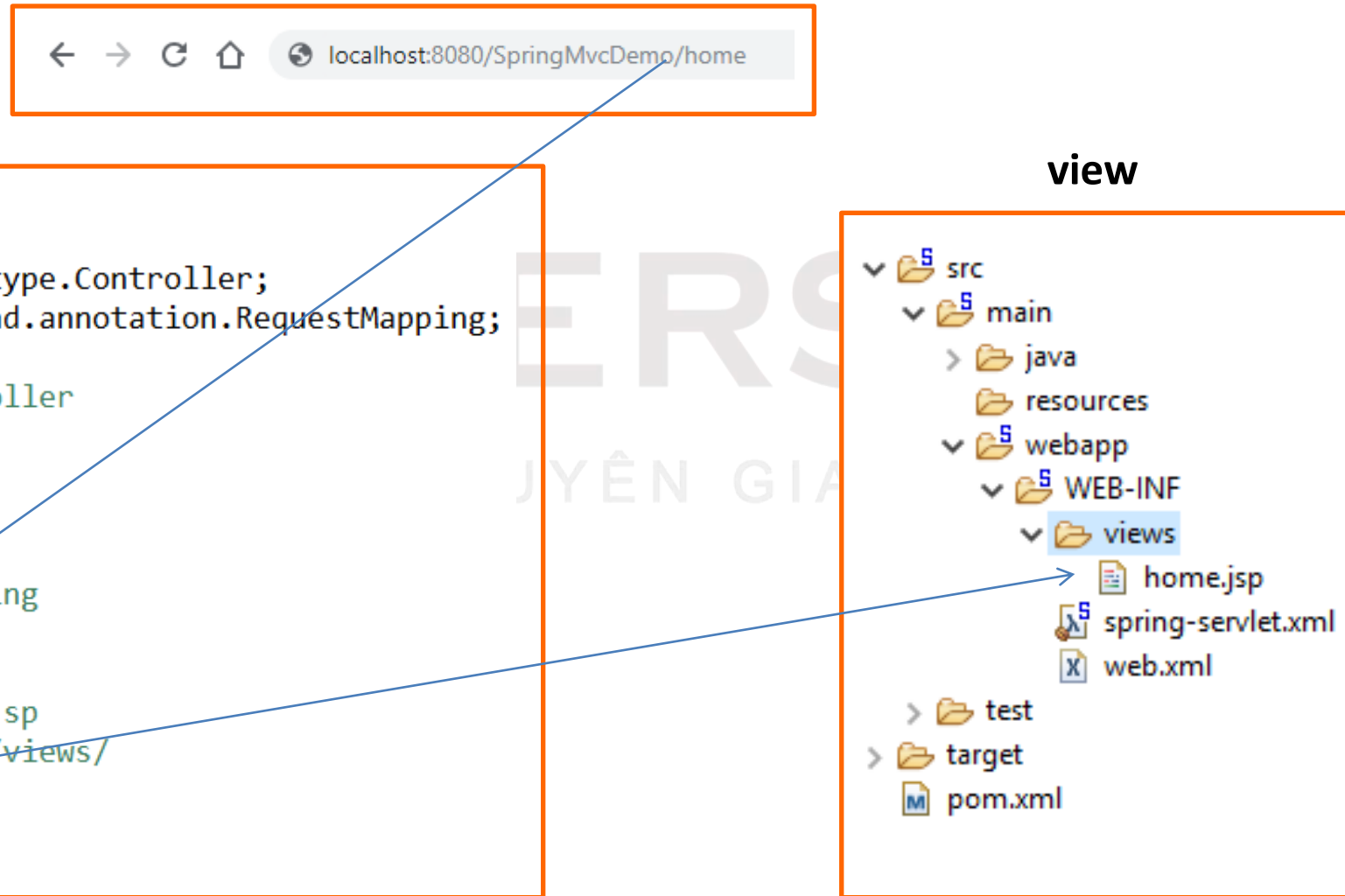
### Controller

```
package com.myclass.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

// Annotation chú thích lớp Controller
// để DispatcherServlet tìm kiếm
@Controller
public class HomeController {
    // Annotation chú thích url
    // DispatcherServlet sẽ mapping
    @RequestMapping("/home")
    public String index() {
        // Forward về trang home.jsp
        // Trong thư mục /WEB-INF/views/
        return "home";
    }
}
```

### view



```
src
├── main
│   ├── java
│   ├── resources
│   └── webapp
│       ├── WEB-INF
│       │   └── views
│       │       ├── home.jsp
│       │       ├── spring-servlet.xml
│       │       └── web.xml
│       ├── test
│       └── target
└── pom.xml
```

# Cấu hình dự án (Java Config)



- ❑ **Bước 1:** Tạo dự án, cấu trúc thư mục theo mô hình MVC.
- ❑ **Bước 2:** Tải thư viện Spring MVC cho dự án.
- ❑ **Bước 3:** Tạo controller.
- ❑ **Bước 4:** Viết các phương thức trả về view.
- ❑ **Bước 5:** Tạo views cho dự án (các file jsp đặt trong thư mục views).
- ❑ **Bước 6:** Cấu hình Spring MVC (class WebMvcConfig).
  - ✓ Cấu hình ViewResolver (prefix và suffix).
  - ✓ Kích hoạt Annotation.
  - ✓ Cấu hình Component Scan.
- ❑ **Bước 7:** Cấu hình Dispatcher Servlet (class AppInitializer).
  - ✓ Cấu hình Url Mapping và khai báo file cấu hình Spring MVC.

# Spring MVC Java config

```
@Configuration // Chú thích đây là class config
@EnableWebMvc // Cho phép sử dụng Annotation
@ComponentScan(basePackages = "com.myclass.controller")
public class WebMvcConfig implements WebMvcConfigurer{

    // Config ViewResolver
    public void configureViewResolvers(ViewResolverRegistry registry) {
        registry.jsp("/WEB-INF/views/", ".jsp");
    }
}
```



Java Resources

- src/main/java
  - com.myclass.configuration
    - WebInitializer.java
    - WebMvcConfig.java
  - com.myclass.controller
  - com.myclass.model
  - com.myclass.repository
  - com.myclass.service
- src/main/resources

```
<!-- Cấu hình Spring MVC Annotation -->
<context:annotation-config />
<mvc:annotation-driven />

<!-- Cấu hình package chứa các controller -->
<context:component-scan base-package="com.myclass.controller"/>

<!-- Cấu hình ViewResolver -->
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>
```

# DispatcherServlet Java Config

```
public class AppInitializer extends
    AbstractAnnotationConfigDispatcherServletInitializer{

    @Override
    protected Class<?>[] getRootConfigClasses() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        // TODO Auto-generated method stub
        return new Class[] { WebMvcConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        // TODO Auto-generated method stub
        return new String[] { "/" };
    }

    @Override
    protected Filter[] getServletFilters() {
        CharacterEncodingFilter filter = new CharacterEncodingFilter();
        filter.setEncoding("UTF-8");
        return new Filter[] { filter };
    }
}
```



```
<servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

# RequestMapping

- ❑ **@RequestMapping** là một trong những **Annotation** được sử dụng nhiều nhất trong **Spring MVC**.
- ❑ **@RequestMapping** được sử dụng để map **request** với **class** hoặc **method** xử lý request đó.
- ❖ **@RequestMapping** với **Class**

```
@Controller
@RequestMapping("/home")
public class HomeController {
    public String index() {
        return "home";
    }
}
```



# RequestMapping

## ❖ @RequestMapping với method

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/login")
    public String index() {
        return "home";
    }
}
```

## ❖ @RequestMapping với nhiều Url

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value = {"/login", "dang-nhap"})
    public String index() {
        return "home";
    }
}
```

# RequestMapping

## ❖ @RequestMapping với Http Method

- Thuộc tính **method** dùng để chỉ định **phương thức Http** (GET, POST, PUT, DELETE, ..)
- Nếu ko chỉ định **Http Method**, mặc định Controller sẽ **nhận tất cả** các loại request.

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value= "/login",
        method = RequestMethod.GET)
    public String index() {
        return "home";
    }
}
```

Chuyển tiếp trang

# Redirect

## ❑ Chuyển trang

- Khi muốn **điều hướng** từ một trang sang trang khác trong **Spring MVC** chúng ta sử dụng “**redirect: url\_trang\_đích**”.

❖ Ví dụ:

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/index")
    public String index() {
        return "index";
    }

    @RequestMapping("/login")
    public String login() {
        ➡ return "redirect:/account/index";
    }
}
```

## ❑ RedirectAttributes

- **RedirectAttributes** được dùng để **truyền các giá trị, tham số** giữa các **controller** khi thực hiện **redirect**.

❖ Ví dụ:

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/index")
    public String index(@RequestParam String name) {
        System.out.println(name);
        return "index";
    }

    @RequestMapping("/login")
    public RedirectView login(RedirectAttributes redirectAttributes) {
        ➡ redirectAttributes.addAttribute("name", "Cybersoft");
        return new RedirectView("account/index");
    }
}
```

# @PathVariable

- ❑ Annotation **@PathVariable** được sử dụng để xử lý những URI động có một hoặc nhiều parameter bên trong URI.

🌐 localhost:8080/BaiTap01/account/test2/acc001/Nguyen-Van-Teo

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping("/test1/{id}")
    public String test1(@PathVariable("id") int id, ModelMap model) {
        model.addAttribute("id", id);
        return "test1";
    }

    @RequestMapping("/test2/{id}/{name}")
    public String test2(@PathVariable("id") int id,
        @PathVariable("name") String name, ModelMap model) {
        model.addAttribute("id", id);
        model.addAttribute("name", name);
        return "test2";
    }
}
```

OFT  
ÁP TRÌNH

# @RequestParam

- ❑ Annotation **@RequestParam** giúp lấy giá trị khi submit dữ liệu từ form hoặc theo cả hai kiểu **GET** và **POST**.
- ❑ **@RequestParam** cũng có thể được sử dụng để lấy giá trị từ url.

localhost:8080/BaiTap01/account?username=admin&password=123456

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value = "login")
    public String test1(
        @RequestParam("username") String username,
        @RequestParam("password") String password) {
        //...
        return "test1";
    }
}
```

# @RequestParam

- **@RequestParam**(**value**, **defaultValue**, **required**) là dạng đầy đủ với ý nghĩa của các tham số:
  - ✓ **value**: chỉ ra tên tham số muốn nhận
  - ✓ **defaultValue**: là giá trị mặc định của tham số khi tham số không tồn tại
  - ✓ **required**: tham số có bắt buộc hay không.
- ❖ Ví dụ:  
**@RequestParam**(**value**="tuoi", **defaultValue**="20", **required**=false) **Integer age**
  - Tên tham số là tuoi sẽ được nhận vào đối số là age
  - Nếu không có tham số thì giá trị của age là 20
  - Tham số tuoi là không bắt buộc

# Produces & Consumes

- ❑ Chỉ định kiểu dữ liệu request và response.
- **consumes**: chỉ chấp nhận các **request** gửi lên có kiểu dữ liệu giống với giá trị khai báo bên trong **consumes**.
- **produces**: Chỉ định kiểu dữ liệu server trả về cho client.

```
@Controller
@RequestMapping("/account")
public class AccountController {

    @RequestMapping(value= "/login",
        method = RequestMethod.GET,
        consumes = "application/json",
        produces = "application/json")
    public String index() {
        return "home";
    }
}
```



# ResponseBody

- ❑ **@ResponseBody** được thêm vào trước các **method** của các controller để chỉ dẫn rằng **method** này sẽ trả về **text** thay vì trả về **view**.



```
@Controller
public class StudentController {

    @GetMapping("/")
    @ResponseBody
    public String index() {

        return "Xin chào!";
    }
}
```

SOFT  
IA LẬP TRÌNH



# ResponseBody

- ❑ Nếu muốn trả về kiểu Json thì cần thêm thư viện Jackson vào.

Thư viện

```
<!-- jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.8</version>
</dependency>
<!-- jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.8</version>
</dependency>
```

Thư viện Jackson tự động chuyển dữ liệu về dạng JSON.

Controller


```
@Controller
public class StudentController {

    @GetMapping("/")
    @ResponseBody
    public Object index() {
        List<Student> students = new ArrayList<Student>();
        students.add(new Student("sv001", "Trần Văn Tâm", 22));
        students.add(new Student("sv002", "Lê Mạnh Cường", 23));
        students.add(new Student("sv003", "Nguyễn Hà My", 20));

        return students;
    }
}
```

# @RestController

- ❑ **@RestController** tương đương với **@Controller** + **@ResponseBody**
- ❑ **@RestController** được dùng trước các **class**, các method trong class này sẽ trả về **text** thay vì trả về **view**.



```
@RestController
public class StudentController {

    @GetMapping("/")
    public String index() {

        return "Xin chào!";
    }
}
```

SOFT  
LẬP TRÌNH

# Bài tập 1



- Tạo dự án BaiTap01 cấu hình bằng Java Config.
  - Tạo HomeController gồm 3 phương thức:
    - ✓ index: Trả về trang index.jsp.
    - ✓ about: Trả về trang about.jsp
    - ✓ contact: Trả về trang contact.jsp.
  - Tạo LoginController gồm 2 phương thức:
    - ✓ index: Nhận request với phương thức GET, trả về trang login.jsp
    - ✓ login: Nhận request với phương thức POST.
  - Tạo WelcomeController trả về trang welcom.jsp.
- ❖ Khi người dùng submit form → gửi thông tin đến phương thức login để kiểm tra:
    - ✓ Nếu username = “cybersoft” và password = “123456” thì điều hướng qua trang welcom với nội dung “Xin chào cybersoft”.
    - ✓ Nếu không đúng hiển thị thông báo “Sai tài khoản hoặc mật khẩu”.

# Bài tập - Các bước thực hiện



## ☐ Các bước thực hiện dự án

- ✓ Bước 1: Tạo dự án.
- ✓ Bước 2: Tạo HomeController.
- ✓ Bước 3: Tạo view cho HomeController.
- ✓ Bước 4: Tạo LoginController và WelcomeController.
- ✓ Bước 5: Tạo login view và welcome view.
- ✓ Bước 6: Viết chức năng xử lý login.

# Bài tập - Cấu trúc thư mục

## Thư viện

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.1.5.RELEASE</version>
  </dependency>
</dependencies>
```

## java

- Java Resources
  - src/main/java
    - com.myclass.configuration
      - AppInitializer.java
      - WebMvcConfig.java
    - com.myclass.controller
      - HomeController.java
      - LoginController.java
      - WelcomeController.java
    - com.myclass.entity
    - com.myclass.repository
    - com.myclass.service
  - src/main/resources
  - src/test/java
  - src/test/resources
  - Libraries
  - JavaScript Resources
  - Deployed Resources

## views

- src
  - main
    - java
    - resources
    - webapp
      - WEB-INF
        - views
          - home
            - about.jsp
            - contact.jsp
            - home.jsp
          - login
            - login.jsp
          - welcome
            - index.jsp
            - welcome.jsp
        - web.xml
    - test
    - target
    - pom.xml

# Bài tập - Controller



## HomeController

```
@Controller
public class HomeController {

    @RequestMapping(value = "/home",
        method = RequestMethod.GET)
    public String index() {
        return "home/home";
    }

    @RequestMapping(value = "/about",
        method = RequestMethod.GET)
    public String about() {
        return "home/about";
    }

    @RequestMapping(value = "/contact",
        method = RequestMethod.GET)
    public String contact() {
        return "home/contact";
    }
}
```

## WelcomeController

```
@Controller
public class WelcomeController {

    @RequestMapping(value = "/welcome",
        method = RequestMethod.GET)
    public String index(ModelMap model, HttpSession session) {
        Object obj = session.getAttribute("name");
        if(obj != null) {
            model.addAttribute("name", obj.toString());
            session.removeAttribute("name");
        }
        return "welcome/welcome";
    }
}
```

# Bài tập - Controller

## LoginController

```
@Controller
public class LoginController {

    @RequestMapping(value = "/login",
                    method = RequestMethod.GET)
    public String index() {
        return "login/login";
    }

    @RequestMapping(value = "/login",
                    method = RequestMethod.POST)
    public String login(@RequestParam("username") String username,
                       @RequestParam("password") String password, ModelMap model,
                       HttpSession session) {

        if(username.equals("admin") && password.equals("123456")) {
            session.setAttribute("name", username);
            return "redirect:/welcome";
        }
        model.addAttribute("message", "Sai tài khoản hoặc mật khẩu!");
        return "login/login";
    }
}
```

# Bài tập - View

login.jsp

```
<div class="container">
  <div class="row my-4">
    <div class="col-md-6 m-auto">
      <h2>LOGIN FORM</h2>
      <form action='<c:url value="/login" />' method="post">
        <p class="text-danger">${ message }</p>
        <div class="form-group">
          <label>Username</label>
          <input type="text" class="form-control" name="username">
        </div>
        <div class="form-group">
          <label>Password:</label>
          <input type="password" class="form-control" name="password">
        </div>
        <button class="btn btn-primary">Login</button>
      </form>
    </div>
  </div>
</div>
```