# ĐẠI HỌC KHOA HỌC TỰ NHIÊN

# KHOA CÔNG NGHỆ THÔNG TIN

**LẬP TRÌNH ỨNG DỤNG JAVA**

# BÁO CÁO ĐỒ ÁN TÌM HIỂU

**GVHD : Nguyễn Văn Khiết**

|  |  |
| --- | --- |
| Nhóm SV thực hiện : |  |
| Nguyễn Hoàng Nam | 0712292 |
| Lâm Xuân Vinh | 0712039 |
| Võ Mai Vĩnh Thành | 0712031 |

|  |
| --- |
| **[Spring mvc]** |
| [Spring MVC giúp cho việc xây dựngứng dụng web một cách chặt chẽ và linh động. Mẫu thiết kế Model-View-Controller giúp phân chia rạch ròi 3 công việc business logic, presentation logic, và navigation logic. Models chịu trách nhiệm đóng gói dữ liệu của ứng dụng. Views có nhiệm vụ hiển thị thông tin cung cấp bởi đối tượng Model trả về cho người dùng. Controllers chịu trách nhiệm nhận request từ người dùng và gọi các dịch vụ bên dưới để xử lý.] |

Mục lục

[PHẦN I : SPRING FRAMEWORK 4](#_Toc294253135)

[I. Giới thiệu chung về Spring Framework : 4](#_Toc294253136)

[II. Đặc điểm của Spring : 5](#_Toc294253137)

[III. Các module chính của Spring Framework : 5](#_Toc294253138)

[1. Spring Core 6](#_Toc294253139)

[2. Spring Context 7](#_Toc294253140)

[3. Spring DAO 7](#_Toc294253141)

[4. Spring ORM 8](#_Toc294253142)

[5. Spring AOP 8](#_Toc294253143)

[6. Spring Web 8](#_Toc294253144)

[7. Spring MVC 8](#_Toc294253145)

[PHẦN II : MODULE SPRING MVC 8](#_Toc294253146)

[I. Giới thiệu về Spring MVC : 8](#_Toc294253147)

[II. Các thành phần chính của Spring MVC : 10](#_Toc294253149)

[1. DispatcherServlet : 11](#_Toc294253150)

[2. Controllers: 11](#_Toc294253151)

[2.1. AbstractController: 12](#_Toc294253152)

[2.2. SimpleFormController: 13](#_Toc294253153)

[3. Handler mappings : 14](#_Toc294253154)

[3.1. BeanNameUrlHandlerMapping : 15](#_Toc294253155)

[3.2. SimpleUrlHandlerMapping : 16](#_Toc294253156)

[3.3. Bộ chặn requests – HandlerInterceptor interface: 17](#_Toc294253157)

[4. ViewResolver : 18](#_Toc294253158)

[4.1. Internal Resource View Resolver : 19](#_Toc294253159)

[4.2. BeanNameViewResolver : 19](#_Toc294253160)

[III. Các ưu điểm và khuyết điểm của Spring MVC Web Framework : 20](#_Toc294253161)

[1. Ưu điểm : 20](#_Toc294253162)

[2. Khuyết điểm : 20](#_Toc294253163)

[IV. Demo : 20](#_Toc294253164)

[V. Tài liệu tham khảo : 26](#_Toc294253165)

# PHẦN I : SPRING FRAMEWORK

# Giới thiệu chung về Spring Framework :

Spring được phát minh bởi Rod Johnson. Nó được giới thiệu lần đầu tiên trong cuốn Expert One-on-One J2EE Design and Development, vào năm 2002. Trong đó, Rod giải thích các kinh nghiệm làm việc với J2EE và làm thế nào EJB mang đến thành công cho các project. Rod tin rằng một framework lightweight như JavaBean thì phù hợp cho nhu cầu của developer. Framework được mô tả đó được gọi Spring khi nó được đưa lên SourceForge vào tháng 2-2003. Vào lúc đó, Juergen Hoeller đã tham gia với Rod để cùng phát triển Spring, họ trở thành cánh tay phải của Spring. Sau đó họ cộng tác thêm một vài developer nữa. Gần đây, Rod và Juergen đã viết một cuốn sách có tên là:“Expert One-on-One J2EE Development without EJB” để mô tả Spring giải quyết nhiều vấn đề của J2EE như thế nào.

Cơ sở kiến trúc của Spring được phát triển bởi Rod vào năm 2000 (trước Struts). Cơ sở này được xây dựng từ kinh nghiệm xây dựng cơ sở hạ tầng trong một số project của Rod.

# Đặc điểm của Spring :

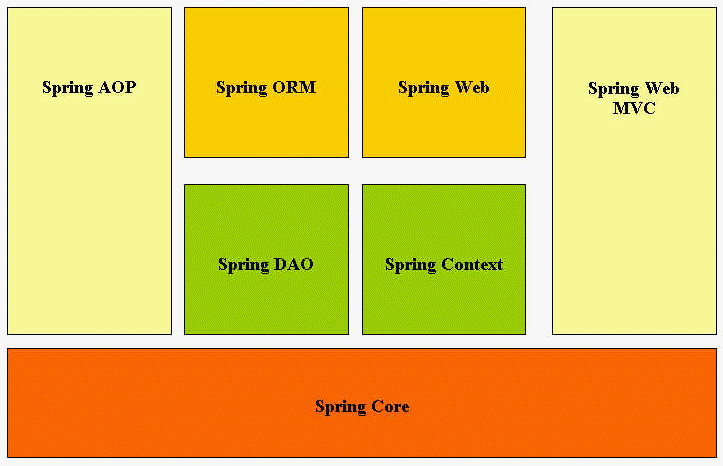
Spring là một J2EE application framework được cung cấp để quản lý các đối tượng nghiệp vụ và sự phụ thuộc giữa chúng. Ví dụ, với IoC (Inversion of Control), nó cho phép bạn xác định DAO (Data Access Object) phụ thuộc vào DataSource. Nó cũng cho phép developer viết các giao tiếp và đơn giản định nghĩa cài đặt trong XML file. Spring chứa nhiều lớp hỗ trợ các framework khác (như Hibernate và Struts) để tạo sự tích hợp dễ dàng hơn.

Áp dụng J2EE Design Pattern thì cồng kềnh và không cần thiết ở một số trường hợp. Spring thì giống design pattern nhưng mọi thứ thì đơn giản hơn. Ví dụ: thay vì viết ServerLocator để tìm Hibernate Session, bạn có thể cấu hình SessionFactory trong Spring. Điều này giúp bạn tập trung vào kỹ thuật (best practices) nhiều hơn là hình dung vào mẫu gần nhất.

# Các module chính của Spring Framework :

Spring là cách cấu hình ứng dụng với JavaBean. Khi nói tới JavaBean, có nghĩa là các lớp với phương thức getter và setter cho các biến của nó. Đặc biệt, nếu một lớp đưa ra setters, Spring sẽ cấu hình lớp đó. Sử dụng Spring, bạn có thể đưa ra bất kỳ sự phụ thuộc nào với setter và sau đó cấu hình Spring để thiết lập sự phụ thuộc này. Thậm chí tốt hơn, thay vì bạn phải viết một lớp để thiết lập kết nối đến database, bạn có thể cấu hình điều đó trong Spring. Cách giải quyết sự phụ thuộc này có tên là: Inversion of Control hoặc Dependency Injection.

Spring có 7 module riêng, mỗi module sẽ có các file JAR riêng:



## Spring Core :

Là phần cơ bản nhất của framework, cung cấp những đặc tính IoC và Dependency Injection. Khái niệm cơ bản là BeanFactory - cài đặt factory pattern cho phép bạn móc nối sự phụ thuộc giữa các đối tượng trong file cấu hình.

IoC là một nguyên tắc trừu tượng mô tả một khía cạnh của một số mẫu thiết kế kiến trúc phần mềm đối lập với lập trình thủ tục. IoC là một phương pháp xây dựng phần mềm mà ở đó mã code có thể được dùng lại chung để quản lý việc thực thi các vấn đề cụ thể. Nó có ý nghĩa là nhiều module mã code có thể được phát triển độc lập với nhau sau đó được tái sủ dụng tích hợp trong một ứng dụng duy nhất

IoC thường được sử dụng cho các mục đích sau:

* Mỗi hệ thống có thể tập trung xung quanh những gì mà nó được thiết kế
* Các hệ thống được giả định về các hệ thống khác hoặc phải làm những gì nên làm
* Thay đổi một mắc xích mà không làm ảnh hưởng đến các mắc xích khác

Dependency Injection là các đối tượng định nghĩa sự phụ thuộc của chúng thông qua tham số của phương thức khởi tạo (constructor) hoặc các thuộc tính được thiết lập trên thể hiện (instance) sau khi nó được khởi tạo hoặc trả về từ phương thức factory.

## Spring Context :

Phía trên của Core package là Context package - cung cấp cách để truy cập đối tượng. Context package kết thừa các đặc tính từ bean package và thêm vào chức năng đa ngôn ngữ, truyền sự kiện, resource-loading,... Thành phần chính của Spring Context là AppicationContext.

Một bean factory có thể sử dụng tốt trong những ứng dụng đơn giản, nhưng để tận dụng lợi thế của tất cả sức mạnh của Spring, bạn sẽ có thể load các beans trong ứng dụng của bạn sử dụng nhiều container nâng cao của Spring: Application context.

Nhìn bên ngoài, AppicationContext thì giống với BeanFactory. Cả hai đều load các beans, kết nối và phân phối chúng theo các yêu cầu. Nhưng ApplicationContext còn có:

* Application context cung cấp một cách thức để giải quyết các thông điệp text (text messages), bao gồm hỗ trợ việc quốc tế hóa cho các thông điệp đó.
* Application context cung cấp một cách tổng quát để load các file tài nguyên, chẳng hạn file ảnh.
* Application context có thể xuất bản các sự kiện sang các beans.

Vì các tính năng nổi bật trên mà ApplicationContext được ưa chuộng hơn BeanFactory trong các ứng dụng gần đây.

Một số ApplicationContext phổ biến:

* ClassPathXmlApplicationContext: Load một định nghĩa context từ một file xml trong class path, xử lý các file định nghĩa context như là các class path resources.
* FileSystemXmlApplicationContext: Load một định nghĩa context từ một file xml trong hệ thống.
* XmlWebApplicaitonContext: Load một định nghĩa context từ một file xml trong ứng dụng web.

## Spring DAO :

DAO package cung cấp cho tầng JDBC, bỏ bớt những coding dài dòng của JDBC và chuyển đổi mã lỗi được xác định bởi database vendor. JDBC package cung cấp cách lập trình tốt như declarative transaction management, không chỉ cho các lớp cài đặt các giao tiếp đặc biệt mà còn cho tất cả POJO (plain old Java objects).

Tầng JDBC và DAO đưa ra một cây phân cấp exception để quản lý kết nối đến database, điều khiển exception và thông báo lỗi được ném bởi vendor của database. Tầng exception đơn giản điều khiển lỗi và giảm khối lượng code mà chúng ta cần viết như mở và đóng kết nối. Module này cũng cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong ứng dụng Spring.

## Spring ORM :

Spring Framework cung cấp tích hợp với Hibernate, JDO, Oracle TopLink, iBATIS SQL Maps và JPA: về quản lý tài nguyên, hỗ trợ thực hiện DAO. Ví dụ đối với Hibernate, có lớp đầu tiên hỗ trợ với nhiều tính năng tiện lợi IoC, giải quyết nhiều vấn đề điển hình của Hibernate mà việc tích hợp đưa ra. Tất cả các gói hỗ trợ cho các O / R (Object Relational) mappers tuân theo giao thức chung của Spring và sự phân cấp DAO exception. Thường có hai phong cách kết hợp: hoặc bằng cách sử dụng DAO của Spring 'mẫu' hoặc mã DAO dựa theo Hibernate / JDO / TopLink / API vv. Trong cả hai trường hợp, DAO có thể được cấu hình thông qua Dependency Injection và tham gia vào các nguồn tài nguyên của Spring và quản lý giao thức.

## Spring AOP :

Module tích hợp chức năng lập trình hướng khía cạnh vào Spring framework thông qua cấu hình của nó. Spring AOP module cung cấp các dịch vụ quản lý giao dịch cho các đối tượng trong bất kỳ ứng dụng nào sử dụng Spring. Với Spring AOP chúng ta có thể tích hợp declarative transaction management vào trong ứng dụng mà không cần dựa vào EJB component.

Spring AOP module cũng đưa lập trình metadata vào trong Spring. Sử dụng cái này chúng ta có thể thêm annotation vào source code để hướng dẫn Spring nơi và làm thế nào để liên hệ với aspect.

## Spring Web :

Cung cấp đặc tính của web như: chức năng file-upload, khởi tạo IoC container sử dụng trình lắng nghe serlvet và web-oriented application context. Package này để tích hợp với WebWork và Struts.

## Spring MVC :

Cung cấp mô hình MVC cho ứng dụng web. Spring MVC framework cung cấp sự phân biệt rõ ràng giữa domain model và web form - cho phép bạn sử dụng tất cả các đặc tính khác của Spring framework. Đây là module trọng tâm mà ta sẽ cùng tìm hiểu trong bài báo cáo này.

# PHẦN II : MODULE SPRING MVC

# Giới thiệu về Spring MVC :

Spring Framework, hay ngắn hơn là *Spring*, là một cấu trúc dùng để xây dựng chương trình ứng dụng mã nguồn mở dành cho ngôn ngữ lập trình Java. Phiên bản đầu tiên của nó do Rod Johnson viết, và đã được ông xuất bản cùng với quyển sách đầu tay *Expert One-on-One Java EE Design and Development* (Nhà xuất bản Wrox Press, Tháng 10 năm 2002) - tạm dịch là "Thiết kế và kiến tạo Java EE - tương giao cá nhân một đối một chuyên ngành". Phiên bản phần mềm này cũng còn được xuất sang nền tảng .NET (*.NET platform*), được gọi là Spring.net.

Spring Framework có 7 module chính. Trong báo cáo này, ta sẽ tập trung tìm hiểu về Spring MVC. Đây là một module giúp cho việc xây dựng ứng dụng web trở nên chặt chẽ và linh động. Mẫu thiết kế Model-View-Controller giúp phân chia rạch ròi 3 công việc business logic, presentation logic, và navigation logic.

**MODEL** gồm các lớp java có nhiệm vụ:

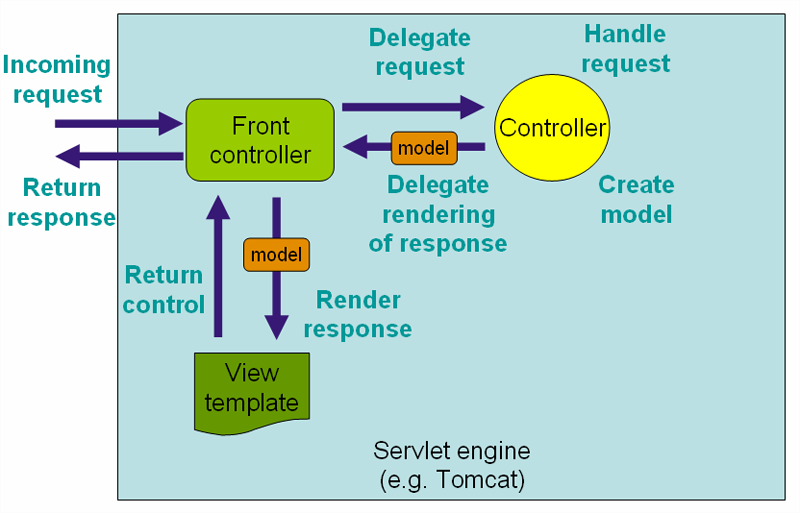
* Biểu diễn data và cho phép truy cập tới để get và set data trong này (JAVABEAN), Thường thì phần layer này mô phỏng 1 cách đầy đủ đối tượng từ thế giới thực.
* Nhận các yêu cầu từ khung nhìn
* Thi hành các yêu cầu đó (tính toán, kết nối CSDL …)
* Trả về các giá trị tính toán cho View.

**VIEW** bao gồm các mã tương tự như JSP để hiển thị form nhập liệu, các kết quả trả về từ Mô hình…

**CONTROLLER**

* Đồng bộ hoá giữa View và Model. Tức là với một trang JSP này thì sẽ tương ứng với lớp java nào để xử lý nó và ngược lại, kết quả sẽ trả về trang jsp nào.Nó đóng vai trò điều tiết giữa View và Model.
* Như vậy, chúng ta có thể tách biệt được các mã java ra khỏi mã html. Người thiết kế giao diện và người lập trình java có thể mang tính chất độc lập tương đối. Việc debug hay bảo trì sẽ dễ dàng hơn, việc thay đổi các theme của trang web cũng dễ dàng hơn …

# Các thành phần chính của Spring MVC :



Việc xử lý request và response trong Spring MVC Framework được mô tả như sau :

- **Bước 1 :**

o DispatcherServlet nhận Request.

o DispatcherServlet tra trong HandlerMapping và gọi Controller kết hợp với Request.

- **Bước 2** **:**

o Controller xử lý Request bằng cách gọi những phương thức dịch vụ thích hợp và sau đó trả về một đối tượng ModelAndView cho DispatcherServlet. Đối tượng ModelAndView này chứa dữ liệu trong đối tượng Model và tên của View.

- **Bước 3 :**

o DispatcherServlet gửi tên của View đến cho một ViewResolver. ViewResolver sẽ tìm View thực sự cần dùng.

- **Bước 4 :**

o DispatcherServlet truyền đối tượng Model đến cho View đã xác định để hiển thị kết quả.

o View lấy dữ liệu trong đối tượng Model và hiển thị kết quả cho người dùng.

## DispatcherServlet :

Tiếp nhận tất cả các request từ Browser.

Điều khiển luồng xử lý và trung chuyển giữa các thành phần (components) trong MVC. Hơn thế nữa, nó còn tích hợp các Spring IoC container, cho phép ta sử dụng đến tất cả các tính năng của Spring.

Trong hình minh họa trên, DispatcherServlet chính là Front controller.

DispatcherServlet thực sự là một Servlet (nó thừa kế từ lớp cơ sở HttpServlet), và được khai báo trong tập tin **web.xml** của ứng dụng web. Các **requests** bạn muốn DispatcherServlet xử lý sẽ phải được ánh xạ bằng cách sử dụng một bộ ánh xạ URL trong cùng một tập tin **web.xml**. Dưới đây là ví dụ về khai báo và định nghĩa DispatcherServlet:

|  |
| --- |
| <web-app>  <servlet>  <servlet-name>example</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>example</servlet-name>  <url-pattern>\*.form</url-pattern>  </servlet-mapping>  </web-app> |

Như ta thấy, tất cả các **requests** có đuôi .form sẽ được DispatcherServlet ‘example’ xử lý.

Khi DispatcherServlet được khởi tạo, framework sẽ gọi file có tên **[tên-servlet]-servlet.xml** trong thư mục **WEB-INF**. Với ví dụ trên, framework sẽ gọi file **example-servlet.xml.**

## Controllers:

Controllers là các thành phần được Dispatcher Servlet gọi để xử lý các nghiệp vụ Business.Tất cả các Controller trong Spring đều implement từ org.springframework.web.servlet.mvc.Controller interface.

|  |
| --- |
| public interface Controller {  /\*\*  \* Process the request and return a ModelAndView object which the DispatcherServlet  \* will render.  \*/  ModelAndView handleRequest(  HttpServletRequest request,  HttpServletResponse response) throws Exception;  } |

Controller interface định nghĩa ra 1 phương thức chịu trách nhiệm xử lí 1 request và trả về model và view thích hợp. Ba khái niệm này chính là nền tảng của Spring MVC framework : Model – View – Controller.

Sau đây là các Controller sử dụng được trong Spring.

* SimpleFormController
* AbstractController
* AbstractCommandController
* CancellableFormController
* AbstractCommandController
* MultiActionController
* ParameterizableViewController
* ServletForwardingController
* ServletWrappingController
* UrlFilenameViewController
  1. **AbstractController:**

Khi sử dụng AbstractController làm baseclass cho các controller của bạn, chỉ cần override hàm **handleRequestInternal (HttpServletRequest, HttpServletResponse)**, viết code xử lý, và trả về một đối tượng ModelAndView. Sau đây là ví dụ:

|  |
| --- |
| package samples;  public class SampleController extends AbstractController {  public ModelAndView handleRequestInternal(  HttpServletRequest request,  HttpServletResponse response) throws Exception {  ModelAndView mav = new ModelAndView("hello");  mav.addObject("message", "Hello World!");  return mav;  }  } |

Ta định nghĩa controller này trong ApplicationContext.xml

|  |
| --- |
| <bean id="sampleController" class="samples.SampleController">  <property name="cacheSeconds" value="120"/>  </bean> |

Controller này sẽ buộc client cache dữ liệu cứ mỗi 2 phút trước khi kiểm tra lại. Đồng thời nó cũng trả về một view được code cứng.

* 1. **SimpleFormController:**

Việc người dùng nhập các thông tin trong 1 form rồi submit là rất thông thường trong Web Applications. Simple Form Controller được dùng chủ yếu cho việc đó. Xét ví dụ sau :

Đây là thông tin cơ bản của 1 form.

|  |
| --- |
| public class EmpInfo{  private String empName;  private int empAge;  private double empSalary;  // các hàm getter, setter  } |

Lưu ý: khi extend **Simple Form Controller** , hàm *doSubmitAction()* thường phải được override vì đây chính là hàm sẽ được gọi khi người dùng submit.

|  |
| --- |
| public class EmpFormController extends SimpleFormController{  public EmpFormController(){  setCommandClass(EmpInfo.class);  }  public void doSubmitAction(Object command){  EmpInfo info = (EmpInfo)command;  process(info);  }  private void process(EmpInfo info){  // do something  }  } |

Giả sử như ta có 2 trang, 1 trang **empInfo.jsp** để người dùng nhập thông tin, 1 trang để thông báo **empSuccess.jsp** sau khi người dùng submit xong, thì ta cấu hình như sau.

|  |
| --- |
| <bean id = "empForm" class="EmpFormController">  <property name="formView">  <value>empInfo</value>  </property>  <property name="successView">  <value>empSuccess</value>  </property>  </bean> |

## Handler mappings :

Sử dụng handler mapping bạn có thể **ánh xạ từng requests đến các handler thích hợp.**

Các chức năng cơ bản của một HandlerMapping là đưa ra một HandlerExecutionChain (chuỗi hành động), và có thể chứa một số bộ chặn. Khi requests đến, các DispatcherServlet sẽ trao cho các handler mapping để nó kiểm tra và xuất ra HandlerExecutionChain thích hợp. Sau đó, các DispatcherServlet sẽ gọi handler và bộ chặn trong hệ thống (nếu có).

Phần này ta đề cập đến 2 handler mapping thông dụng nhất của Spring. Cả hai đều kế thừa AbstractHandlerMapping và chia sẻ các thuộc tính sau đây:

* interceptors: danh sách các bộ chặn được sử dụng.
* defaultHandler: bộ xử lý mặc định, được gọi khi handler mapping không trả ra kết quả phù hợp.
* order: Spring sẽ sắp xếp thứ tự tất cả các ánh xạ handler trong ngữ cảnh và gọi handler đầu tiên phù hợp.
* alwaysUseFullPath: true = đường dẫn đến handler được truyền đầy đủ.
* urlPathHelper: nên để default.
* urlDecode: giá trị mặc định cho thuộc tính này là false. HttpServletRequest trả về các request URL và URI không được giải mã. Nếu bạn không muốn chúng được giải mã trước khi HandlerMapping sử dụng chúng để tìm handler thích hợp, hãy set giá trị nó bằng true.
* lazyInitHandlers: cho phép các handler singleton khởi tạo với chế độ “lazy”. Mặc định giá trị này là false.

(Lưu ý: trong bốn đặc tính cuối cùng chỉ có sẵn cho các lớp con của org.springframework.web.servlet.handler.AbstractUrlHandlerMapping)

* 1. **BeanNameUrlHandlerMapping :**

Một handler mapping đơn giản nhưng rất mạnh mẽ là BeanNameUrlHandlerMapping, nó **ánh xạ các HTTP requests đến tên của các beans**, được định nghĩa trong bối cảnh ứng dụng web. Ví dụ chúng ta muốn cho phép một người dùng tạo một tài khoản, ở đây ta đã có một form controller thích hợp và một trang JSP để hiển thị. Khi sử dụng BeanNameUrlHandlerMapping, ta có thể ánh xạ HTTP requests với địa chỉ http://samples.com/editaccount.form vào form Controller như sau:

|  |
| --- |
| <beans>  <bean id="handlerMapping" class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>  <bean name="/editaccount.form" class="org.springframework.web.servlet.mvc.SimpleFormController">  <property name="formView" value="account"/>  <property name="successView" value="account-created"/>  <property name="commandName" value="account"/>  <property name="commandClass" value="samples.Account"/>  </bean>  <beans> |

Tất cả requests gửi đến yêu cầu địa chỉ /editaccount.form bây giờ sẽ được xử lý bởi form Controller ở trên. Tất nhiên chúng ta phải định nghĩa một servlet-mapping trong **web.xml**, để tất cả các requests đều kết thúc bằng .form.

|  |
| --- |
| <web-app>  ...  <servlet>  <servlet-name>sample</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  *<!-- maps the sample dispatcher to \*.form -->*  <servlet-mapping>  <servlet-name>sample</servlet-name>  <url-pattern>\*.form</url-pattern>  </servlet-mapping>  ...  </web-app> |

* 1. **SimpleUrlHandlerMapping :**

|  |
| --- |
| <web-app>  ...  <servlet>  <servlet-name>sample</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <load-on-startup>1</load-on-startup>  </servlet>  *<!-- maps the sample dispatcher to \*.form -->*  <servlet-mapping>  <servlet-name>sample</servlet-name>  <url-pattern>\*.form</url-pattern>  </servlet-mapping>  *<!-- maps the sample dispatcher to \*.html -->*  <servlet-mapping>  <servlet-name>sample</servlet-name>  <url-pattern>\*.html</url-pattern>  </servlet-mapping>  ...  </web-app> |

Cấu hình web.xml trong đoạn code trên cho phép tất cả requests có kết thúc .form và .html đều được xử lý bởi ‘sample’ dispatcher servlet.

|  |
| --- |
| <beans>    *<!-- no 'id' required, HandlerMapping beans are automatically detected by the DispatcherServlet -->*  <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  <property name="mappings">  <value>  /ex/view\*.html=helpController  /\*\*/help.html=helpController  </value>  </property>  </bean>  <bean id="helpController"  class="org.springframework.web.servlet.mvc.UrlFilenameViewController"/>  <bean id="editAccountFormController"  class="org.springframework.web.servlet.mvc.SimpleFormController">  <property name="formView" value="account"/>  <property name="successView" value="account-created"/>  <property name="commandName" value="Account"/>  <property name="commandClass" value="samples.Account"/>  </bean>  <beans> |

Ánh xạ handler này sẽ chuyển requests yêu cầu ‘help.html’ trong bất cứ thư mục nào đến ‘helpController’. Requests yêu cầu resource bắt đầu với ‘view’ và kết thúc với ‘html’ trong thư mục ‘ex’ sẽ được chuyển đến ‘helpController’.

* 1. **Bộ chặn requests – HandlerInterceptor interface:**

Cơ chế handler mapping của Spring có khái niệm xử lý đánh chặn, nó có thể cực kỳ hữu ích khi bạn muốn áp dụng các chức năng cụ thể cho một request xác định, ví dụ, kiểm tra điều kiện chẳng hạn.  
Ví dụ dưới đây tạo ra một bộ chặn, nó chặn tất cả các requests và redirect người dùng đến một trang xác định nếu thời gian truy cập không nằm từ 9h sáng đến 6h tối.

|  |
| --- |
| <beans>  <bean id="handlerMapping"  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">  <property name="interceptors">  <list>  <ref bean="officeHoursInterceptor"/>  </list>  </property>  <property name="mappings">  <value>  /\*.form=editAccountFormController  /\*.view=editAccountFormController  </value>  </property>  </bean>  <bean id="officeHoursInterceptor"  class="samples.TimeBasedAccessInterceptor">  <property name="openingTime" value="9"/>  <property name="closingTime" value="18"/>  </bean>  <beans> |

|  |
| --- |
| package samples;  public class TimeBasedAccessInterceptor extends HandlerInterceptorAdapter {  private int openingTime;  private int closingTime;  public void setOpeningTime(int openingTime) {  this.openingTime = openingTime;  }  public void setClosingTime(int closingTime) {  this.closingTime = closingTime;  }  public boolean preHandle(  HttpServletRequest request,  HttpServletResponse response,  Object handler) throws Exception {  Calendar cal = Calendar.getInstance();  int hour = cal.get(HOUR\_OF\_DAY);  if (openingTime <= hour < closingTime) {  return true;  } else {  response.sendRedirect("http://host.com/outsideOfficeHours.html");  return false;  }  }  } |

## ViewResolver :

Như đã nói ở phần trên, tất cả controller trong Spring Web MVC framework trả về 1 đối tượng gọi là ModelAndView. Views trong Spring được gán cho 1 “view name “ và được phân giải bởi 1 “view resolver”.Sau đây là các “view resolver” trong Spring

* *BeanNameViewResolver*
* *FreeMarkerViewResolver*
* *InternalResourceViewResolver*
* *JasperReportsViewResolver*
* *ResourceBundleViewResolver*
* *UrlBasedViewResolver*
* *VelocityLayoutViewResolver*
* *VelocityViewResolver*
* *XmlViewResolver*
* *XsltViewResolver*

Ở đây ta sẽ đề cập đến 2 ViewResolver chính là “*InternalResourceViewResolver*” và “*BeanNameViewResolver*”.

* 1. **Internal Resource View Resolver :**

View Resolver này sẽ ánh xạ Logical name của Resource (ModelAndView object được trả về bởi Controller) đến một Physical View(myView.jsp,…). Ví dụ :

|  |
| --- |
| public class MyController {  public void handle(){  if(condition1()){  return new ModelAndView("myView1");  }else if (condition2()){  return new ModelAndView("myView2");  }  return new ModelAndView("myView3");  } |

Giả sử như Request của người dùng thỏa ***condition1()***, thì view “**myView.jsp**”(trong thư mục ” /WEB-INF”) sẽ được hiển thị,nếu không thì hoặc “**myView2.jsp**” hoặc “**myView3.jsp**” sẽ được hiển thị.

Ngoài ra, ta còn phải cấu hình file web.xml :

|  |
| --- |
| <bean id="viewResolver" class="org.springframework.web.servlet.view.  InternalResourceViewResolver">  <property name="prefix"><value>/WEB-INF/</value></property>  <property name="suffix"><value>.jsp</value></property>  </bean> |

Khi logical View name là myView1, thì view thực sự sẽ là : **prefix + logical View Name + the suffix** , tức là: **/WEB-INF/myView.jsp**

* 1. **BeanNameViewResolver :**

Resolver này có nhiệm vụ ánh xạ một “view name” vào một bean trong ApplicationContext.

# Các ưu điểm và khuyết điểm của Spring MVC Web Framework :

## Ưu điểm :

* Spring là một framework Java mạnh mẽ được sử dụng trong những ứng dụng Java có phạm vi lớn. Nó cung cấp những dịch vụ Enterprise đến những Plain Old Java Objects (POJOs). Cơ chế IoC giúp ứng dụng đạt được sự đơn giản hoá và tăng khả năng kiểm tra lỗi.
* Spring MVC cung cấp một sự phân chia rất rõ ràng, rành mạch giữa những Controller, Java Bean models và Views.
* Spring MVC rất linh hoạt, toàn bộ Spring MVC được xây dựng dựa trên những interfaces. Mọi phần của Spring MVC framework được cấu hình thông qua việc lắp ghép những interface, class tiện ích sẵn có, và thậm chí được tạo bởi người dùng.
* Spring không chỉ sử dụng công nghệ JSP mà còn có thể dễ dàng tích hợp các công nghệ view khác như
* Velocity, XSLT, FreeMarker, XL, …
* Cung cấp cơ chế che dấu nền công nghệ bên dưới, trang web khi hiển thị chỉ có extension là .htm, không thể biết được bên dưới ta sử dụng công nghệ, kỹ thuật gì, JSP hay Velocity, XLST, … thậm chí là những công nghệ view được custom bởi người dùng.
* Spring Controller được cấu hình thông qua IoC như mọi đối tượng khác. Điều này làm chúng dễ dàng
* được test, và được tích hợp dễ dàng với những đối tượng khác được quản lý bởi Spring.
* Kết buộc trực tiếp các input từ view với những đối tượng domains.

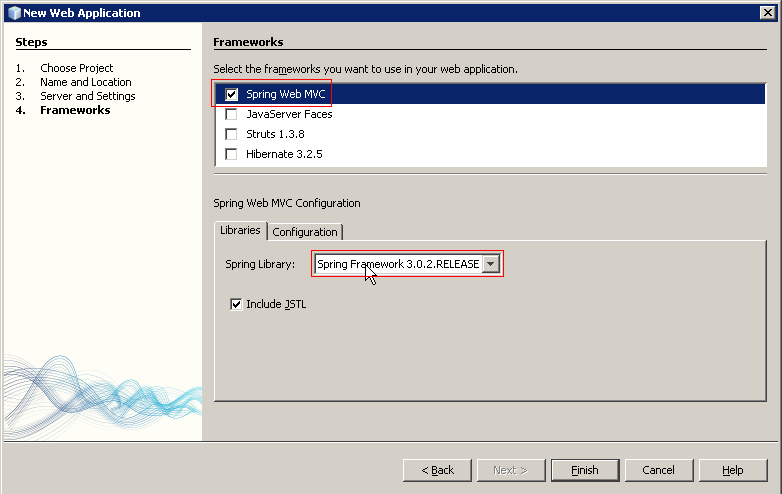
## Khuyết điểm :

* Cấu hình phức tạp và cồng kềnh => không phát huy được sức mạnh khi áp dụng cho các ứng dụng quy mô nhỏ mà có thể ngược lại còn làm cho ứng dụng phức tạp.

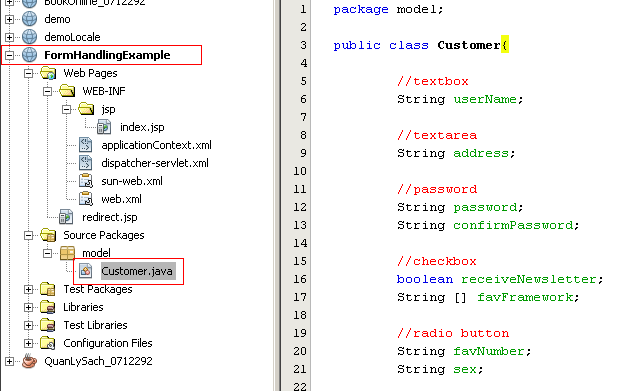
# Demo :

Trong bài demo này, ta sẽ tạo một form để người dùng nhập dữ liệu, đánh dấu textbox, chọn combo box item v.v… là trang CustomerForm.htm. Nếu nhập liệu sai, validator của chương trình sẽ xuất thông báo lỗi. Nếu đúng, người dùng được redirect đến trang CustomerSuccess.htm. Chi tiết xin xem thêm trên video demo.

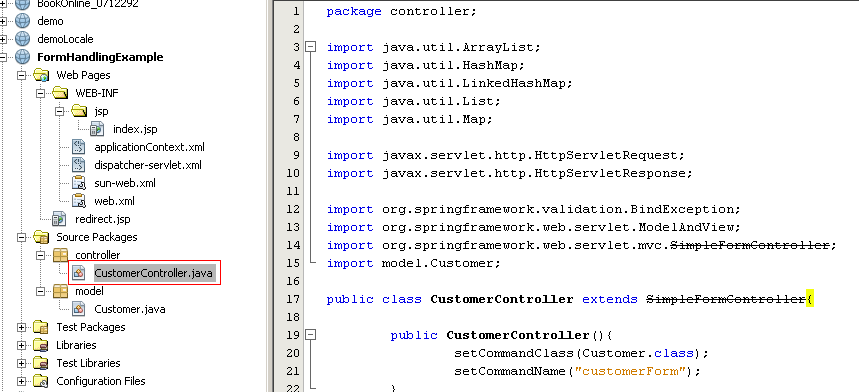
**Bước 1: Tạo project web application**

****

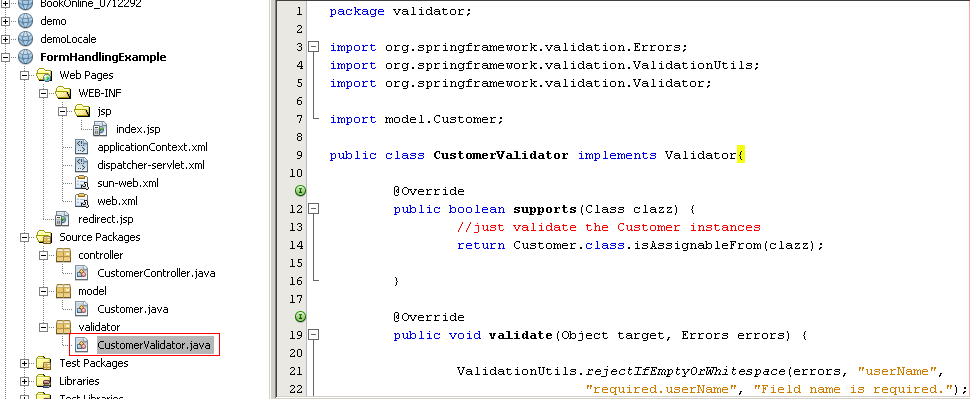
**Bước 2: Tạo model**

****

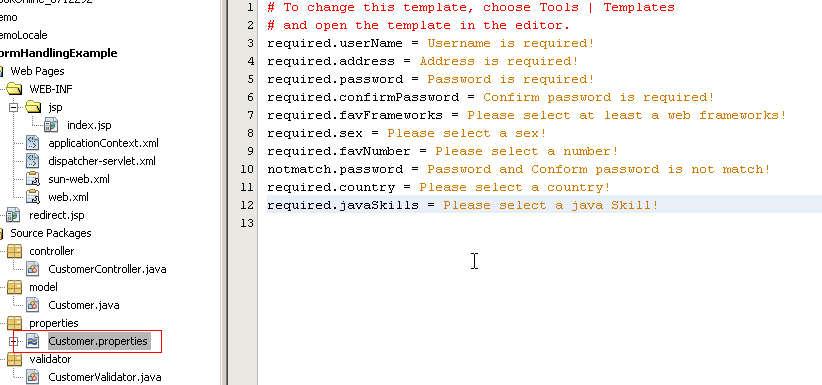
**Bước 3: Tạo controller**

****

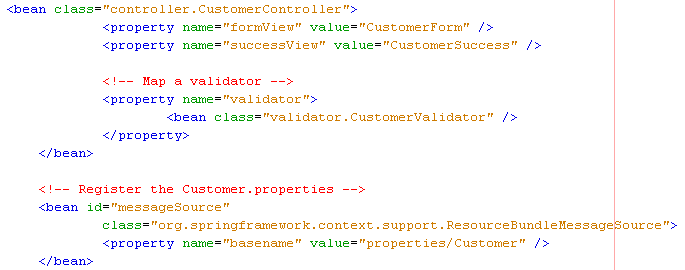
**Bước 4: Tạo validator để kiểm tra dữ liệu nhập**

****

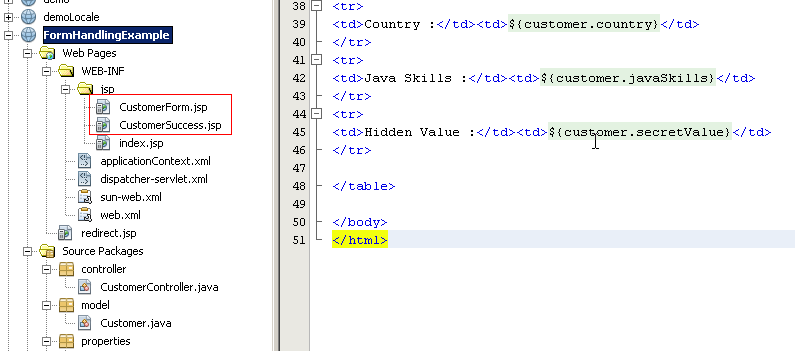
**Bước 5: Tạo file properties để lưu các thông báo lỗi**

****

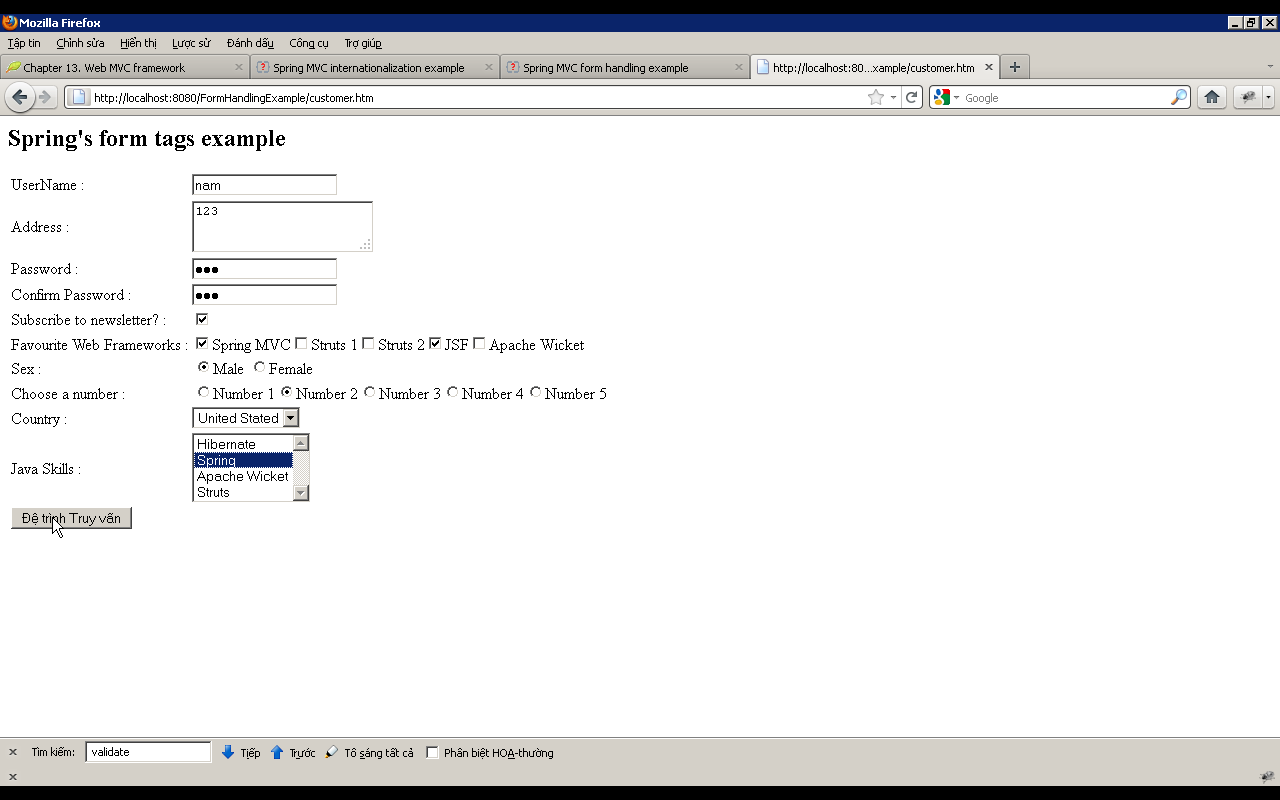
**Bước 6: Cấu hình dispatcher-servlet.xml**

****

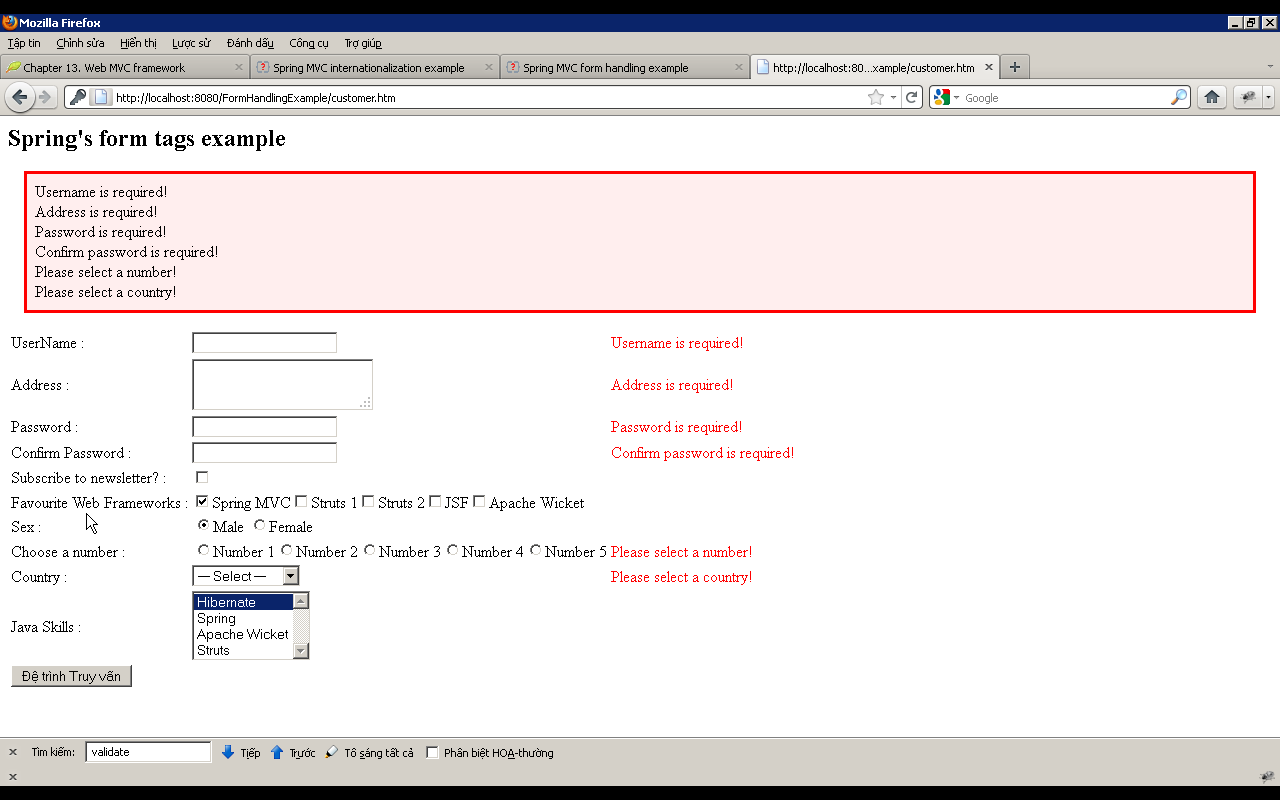
**Bước 7: Viết trang jsp, một trang hiện form CustomerForm.jsp, một trang xuất thông tin người dùng vừa nhập sau khi đã kiểm tra tính hợp lệ CustomerSuccess.jsp**

****

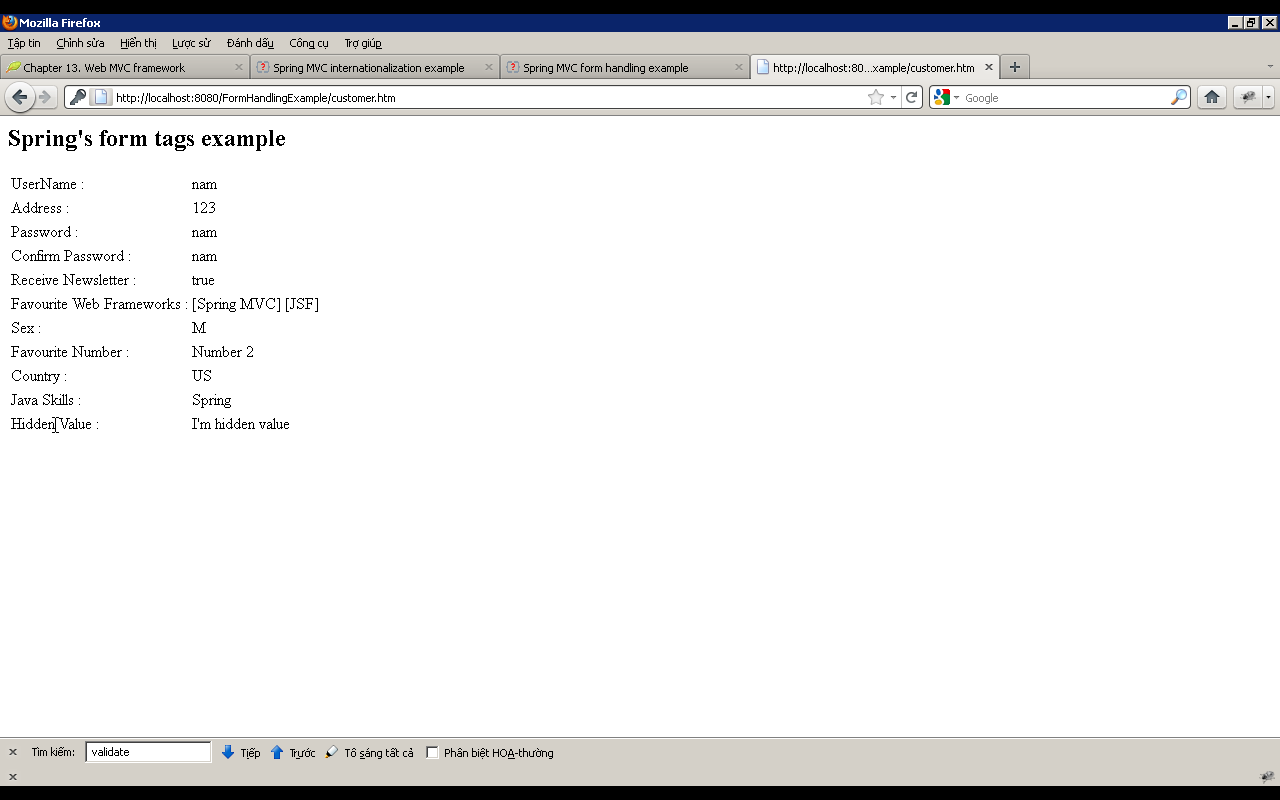
**Kết quả sau khi build và run project :**

****

**Nếu thông tin nhập không hợp lệ :**

****

**Nếu thông tin nhập hợp lệ :**

****

# Tài liệu tham khảo :

<http://www,javavietnam.org>

<http://static.springsource.org/spring/docs/2.0.7/reference/mvc.html>

[http://www.vaannila.com/spring/spring-tutorial/spring-tutorial.html](http://www.vaannila.com/spring/spring-tutorial/spring-tutorial.html%20)

<http://netbeans.org/kb/docs/web/quickstart-webapps-spring.html>

<http://sites.google.com/site/springmvcnetbeans/step-by-step>

<http://forum.springsource.org/showthread.php?t=16553>

<http://www.mkyong.com/spring-mvc/spring-mvc-internationalization-example>