Spring MVC

KurumsalJava.com
KurumsalJavaAkademisi.com

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com

Giriş

Günümüzde kullanılan birçok program web arayüzleriyle internet ve intranet ortamlarında kullanılmaktadır. Web tabanlı programların popüler olmasının sebepleri bir taraftan sadece bir web tarayıcısının (browser) yeterli olması, diğer taraftan da bu tür aplikasyonların bilgisayar üzerinde herhangi bir kurulum (install/setup) gerektirmemesidir.

Web tabanlı programlar kullanıcı (client) için herhangi bir kurulum gerektirmezken, server tarafında programın içinde çalışabileceği bir ortamın oluşturulması gerekmektedir. Bu ortamlar web protokolü olan HTTP[1] üzerinden kullanıcı ile interaksiyona girebilecek yapıdadırlar. Gerekli bu ortamları oluşturmak için Apache[2] ve Tomcat[3] gibi server programları kullanılır. Java dünyasında web aplikasyonları oluşturmak için Servlet/JSP teknolojileri kullanılır. Bu teknolojilerle implemente edilmiş bir sistem Tomcat gibi bir Servlet Container içinde çalışabilir, çünkü Tomcat program için gerekli altyapı hizmetlerini sunmaktadır.

Bir önceki bölümde incelediğimiz Spring framework, web tabanlı programların implementasyonu için kullanılabilecek Spring MVC ismini taşıyan bir modüle sahiptir. Spring MVC'nin sağladığı avantajları şu şekilde sıralayabiliriz:

- Çevik süreç XP'de test güdümlü implementasyon büyük önem taşımaktadır. Kullanılan araçlar bunu desteklemediği sürece web tabanlı aplikasyonları XP tarzı implemente etmek zor bir süreç olabilir. Spring MVC HTTP protokolünü simule eden mock sınıflarla (örneğin MockHttpServletRequest) test güdümlü implementasyonu kolaylaştırır.
- Spring MVC, dependency injection ve diğer Spring özelliklerini web aplikasyonlarında kullanma imkanı tanıyarak, daha esnek ve test edilebilir yapıların oluşmasını sağlar.
- JSP sayfalarında implementasyonu kolaylaştırmak ve içeriği Java kodundan bağımsız kılmak için JSP tagler (tag library) kullanılır. Spring MVC, JSP EL (expression language) dilini kullanan kendi tag kütüphanesi sahiptir.
- Spring Struts[4], Tapestry[5], WebWork[6] gibi diğer popüler web frameworkleri ile entegre edilebilir. Bunun yanısıra Spring MVC, JSP harici Velocity[7] ve Freemarker[8] gibi diğer gösterim teknolojilerinin kullanımını mümkün kılmaktadır.
- HTML formları için özel model sınıflarının oluşturulması gerekmemektedir. İşletme (business) katmanında bulunan domain modele ait sınıflar HTML formlardaki verileri tutmak için kullanılabilir. Spring MVC kullanıcının girdiği verileri otomatik olarak kullanılan model sınıfından bir nesneye dönüştürerek, verilerin kullanımını kolaylaştırır.
- Spring Web Flow modülü, birbiriyle ilişkili (wizard) sayfaların oluşturulmasını kolaylaştırır.

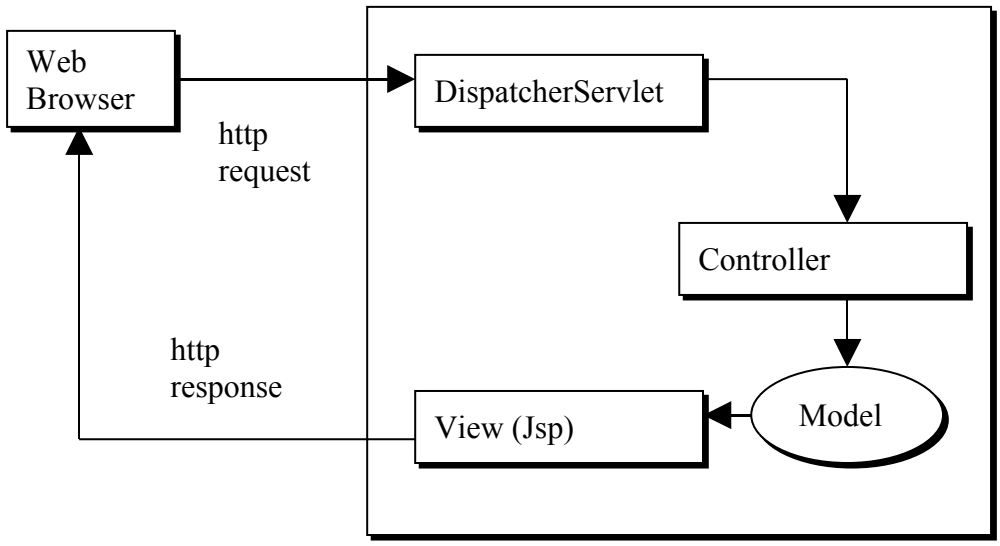
[1] Bakınız: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[2] Bakınız: http://www.apache.org
[3] Bakınız: http://tomcat.apache.org
[4] Bakınız: http://struts.apache.org
[5] Bakınız: http://jakarta.apache.org/tapestry
[6] Bakınız: http://www.opensymphony.com/webwork
[7] Bakınız: http://jakarta.apache.org/velocity
[8] Bakınız: http://www.freemarker.org

- Web aplikasyonu için gerekli tüm konfigürasyon Spring'den tanıdığımız XML dosyalarında yapılır. Bu, programın tekrar derlemek zorunda kalmadan değiştirilebilir ve genişletilebilirliğini artırır.

MVC Web Modeli

Spring MVC, isminden de anlaşılacağı gibi MVC tasarım şablonu kullanılarak implemente edilmiştir. MVC tasarım şablonunu implemente etmek için Front Controller tasarım şablonu kullanılır. Front Controller tasarım şablonu ile sisteme yöneltilen tüm istekler (request) merkezi bir yerde toplanarak işlem görürler. Spring MVC bünyesinde DispatcherServlet Front Controller görevini üstlenmektedir.

MVC tasarım şablonunun kullanıldığı bir projede model sınıfları işlenen verileri ihtiva eder. Örneğin bir üyenin hesap bilgileri bilgibankasından JDBC[9] ile okunduktan sonra bir model sınıfında tutulur. Model basit bir POJO (Plain Old Java Object) sınıfı olabilir. View gösterim katmanına aittir ve genelde JSP gösterim teknolojisi kullanılarak programlanır. View elementinin görevi model sınıflarında yer alan verilerin kullanıcıya gösterilmesini sağlamaktır. Model değiştiği zaman, view buna göre kendisini adapte ederek, modelin ihtiva ettiği verileri gösterir. Controller sınıfları, model ve view arasındaki interaksiyonu koordine etmek için kullanılır.



Resim 13.1 Spring MVC yapısı

Spring MVC'nin mimarik yapısı resim 13.1 de yer almakadır. Sistemin merkezinde DispatcherServlet bulunmaktadır. Kullanıcı isteklerinin (http request) hepsi ilk önce DispatcherServlet'e yönlendirilir. Kullanılan URL (örneğin http://localhost) ve URI (örneğin /login) adreslerine göre DispatcherServlet bir Controller sınıfını seçerek, işlemi bu Controller sınıfına devreder. Controller bünyesinde gerekli işlemler yapılarak Model sınıfı oluşturulur ya da mevcut bir Model sınıfı yeniden yapılandırılır. Controller bu işlemin

[9] Bakınız: http://java.sun.com/javase/technologies/database/

ardından kontrolü JSP (View) sayfasına devreder. View bölümünde JSP yerine Velocity[10] gibi başka gösterim teknolojileri kullanmak mümkündür. JSP sayfasının görevi, Model sınıfında yer alan verilerin kullanıcıya gösterilmesini sağlamaktır. View, Model sınıflarında yer alan veriler üzerinde değişiklik yapmaz. Görevi sadece verilerin kullanıcıya gösterilmesiyle sınırlıdır.

MVC ile modüler bir yapının oluşturulması kolaylaşır. Kullanıcı isteğinin (request) taranması, yönlendirme (navigasyon) ve işlem mantığı Controller sınıflarında implemente edilir. Controller sınıfları verilerin kullanıcıya gösterilmesi için gerekli mantığı ihtiva etmezler. Bu View sınıflarının görevidir. İşlem gören veriler Model sınıflarında tutulur. Katmanların bu şekilde birbirlerinden ayrılmaları, programın geliştirilmesini ve bakımını kolaylaştırır.

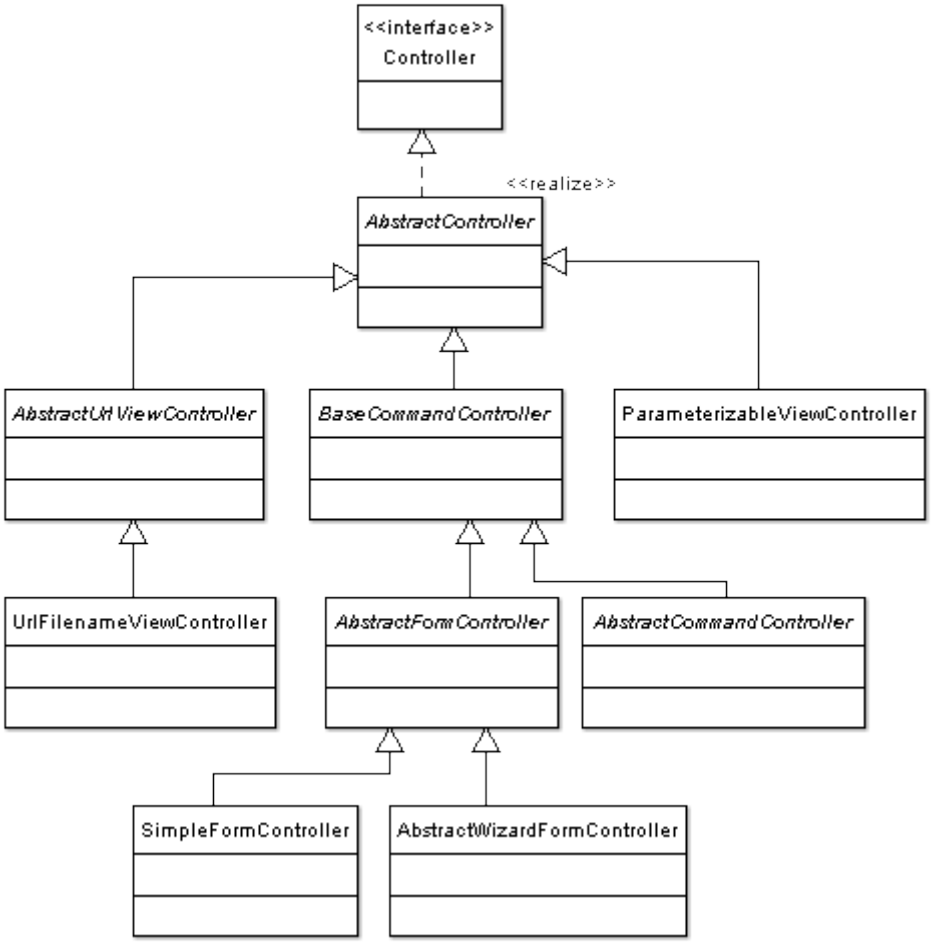
Spring MVC Konseptleri

Resim 13.1 de yer aldığı gibi Spring MVC ile implemente edilen bir aplikasyon değişik komponentlerden oluşur. Bu komponentleri yakından inceleyerek, bir Spring MVC aplikasyonun anatomisini daha iyi anlayabiliriz.

Controller

Spring MVC aplikasyonlarında Controller sınıfları değişik komponentler arasında veri akışını ve kontrolü sağlayan modüllerdir. Spring değişik tipte Controller sınıfları sunmaktadır.

[10] Bakınız: http://jakarta.apache.org/velocity



Resim 13.2 Spring MVC Controller hiyerarşisi

Tablo 13.1: Spring Controller Sınıfları

Controller	Hiyerarşinin en üstünde bulunan interface sınıftır. handleRequest() metoduna sahiptir. Bu metot HttpServletRequest ve HttpServletResponse sınıflarından olan nesneleri parametre olarak kabul eder.
AbstractController	Template Method tasarım şablonunu kullanmak isteyen controller sınıfları için kullanılabilecek üst sınıftır.
AbstractUrlViewController	View ismini kullanılan URL'e göre tespit etmek için kullanılan soyut controller sınıfıdır.
BaseCommandController	Command nesneleri oluşturarak, kullanıcı tarafından gönderilen request parametrelerini command nesnesinin değişkenlerine eşitleyen soyut üst sınıf implementasyonudur.

ParameterizableViewController	Tanımlanmış bir view elementini geri gönderen basit bir controller sınıfı implementasyonudur.
UrlFilenameViewController	Kendi controller sınıfı olmayan view elementleri için kullanılan controller sınıfı.
AbstractFormController	Bir form bean (command) sınıfını gelen request parametreleri ile otomatik olarak yapılandırabilen soyut controller sınıfı.
AbstractCommandController	Spesifik command controller sınıfları için kullanılabilecek soyut üst sınıf.
SimpleFormController	HTML formları için kullanılan controller implementasyonu.
AbstractWizardFormController	Wizard tipi HTML formları oluşturmak için kullanılan soyut üst sınıf.

Spring bünyesinde birçok Controller implementasyonu bulunması seçimi zorlaştırmaktadır. Yapılacak implementasyonun gerekleri doğrultusunda Controller seçimi yapılmalıdır. Örneğin bir HTML formu için SimpleFormController seçilmelidir. Ya da herhangi bir Controller sınıfına ihtiyaç duymayan basit bir JSP sayfası için UrlFilenameViewController seçilebilir. Kullanılacak Controller tipi kullanım senaryosuna (use case) göre değişik olacaktır.

Model ve View

Veriler model sınıflarında tutulur. Controller sınıfları model nesnelerini view elementlerine devretmek için ModelAndView sınıfını kullanırlar. Bunun bir örneği kod 13.1 de yer almaktadır.

Kod 13.1 Controller örneği

```
package controller;

import java.util.HashMap;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.springframework.web.servlet.ModelAndView;
import org.springframework.web.servlet.mvc.AbstractController;

public class TestController extends AbstractController
{

      private CustomerDao customerDao;
```

```
	@Override
	protected ModelAndView handleRequestInternal(
				HttpServletRequest request,
				HttpServletResponse response) throws Exception
	{
		Map model = new HashMap();
		String view ="/customer/list";

		model.put("customerList",
				customerDao.getCustomerList());
		return new ModelAndView(view,model);
	}
}
```

Kullanıcı tarafından gönderilen istek (request) TestController sınıfının handleRequestInternal() metoduna ulaşır. Bu metot içinde basit bir model nesnesi (Map) oluşturulmaktadır. Bu model nesnesi bünyesinde customerList etiketi altında customerDao sınıfından gelen müşteri listesini barındırır. Metot sonunda oluşturulan ModelAndView nesnesi, hem modeli hem de kontrolün hangi view (sayfa) elementine devredileceğini ihtiva eder. Örnekte kontrol /customer/list view elementine bırakılmaktadır. View bünyesinde model içinde bulunan müşteri listesi edinilerek, kullanıcıya gösterilir.

Spring Tag Library

JSP sayfalarında Java kodu kullanmadan, veriler üzerinde işlem yapabilmek için tagler kullanılır. Spring'in JSP sayfaları için kullanılabilecek tag kütüphanesi vardır. Bu tagler aracılığıyla JSP sayfalarında örneğin model nesnelerinde tutulan verilere ulaşılır ya da <spring:bind> tagında olduğu gibi bir HTML formunda yer alan veriler form için kullanılan model nesnesine eklenir. Kısaca Spring tag kütüphanesi, JSP sayfalarında Java kodu kullanmak zorunda kalmadan Controller sınıflarında oluşturulan model nesneleri üzerinde işlem yapmayı kolaylaştırır.

Kod 13.2 Tag örneği

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core_rt"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt_rt"%>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ page buffer="16kb"%>

<spring:bind path="Customer">
<font color="red">
	<c:out value="${status.errorCode}" escapeXml="false" />
</font>
</spring:bind>

<form:form name="/app/login/login" commandName="Customer">
```

```
<table width="100%" cellspacing="1">
      <tr>
            <td>Email:</td>
            <td><form:input size="20" maxlength="50" path="email" /> 
            </td>
            <td> </td>
      </tr>
      <tr>
            <td>Password:</td>
      <td><form:input size="20" maxlength="50" path="password" /> 
            </td>
            <td> </td>
      </tr>
      <tr>
            <td></td>
            <td><input type="submit" value="Login" name="Login"/></td>
            <td> </td>
      </tr>
</table>

</form:form>
```

Command Nesne

HTML formlarında yer alan verileri tutmak için Spring terminolojisinde command nesne olarak bilinen Java sınıfları (POJO[11]) kullanılır. Form butonuna tıklanmasıyla birlikte, formda yer alan veriler otomatik olarak command nesnede bulunan değişkenlere eşitlenir. HTML formunda örneğin email isminde bir alan varsa, command nesnenin de String tipinde ve email isminde bir sınıf değişkeni olacaktır. Spring setEmail() metodunu kullanarak, form üzerinden girilen email adresini command nesnenin email değişkenine eşitler. Command nesneleri Java bean konform olmak zorundadırlar, yani her değişkenin get() ve set() metotları olmak zorundadır, aksi taktirde Spring otomatik olarak verileri command nesnesi değişkenlerine eşitleyemez.

Command nesneleri için Validator interface sınıfını implemente eden validasyon sınıfları oluşturulduğu taktirde, validator sınıf command nesnesinin sahip olduğu değişken değerlerini valide (kontrol) edebilir.

Validasyon (Validation)

Web tabanlı sistemlerde kullanıcı tarafından HTML formlara girilen verilerin işlem öncesi güvenlik gereği kontrol edilmesi gerekmektedir. Spring, veri validasyonunu kolaylaştırmak için Validator isminde bir interface sınıfı sunmaktadır. Bir command nesne için validasyonu gerçekleştirmek üzere Validator interface sınıfını implemente bir validasyon sınıfı oluşturulabilir. Spring, otomatik olarak, command nesneyi Controller sınıfına devretmeden önce validasyon sınıfını kullanarak, verileri valide edecektir. Bunun bir örneği kod 13.3 de yer almaktadır.

Kod 13.3 Validasyon örneği

[11] Plain Old Java Object

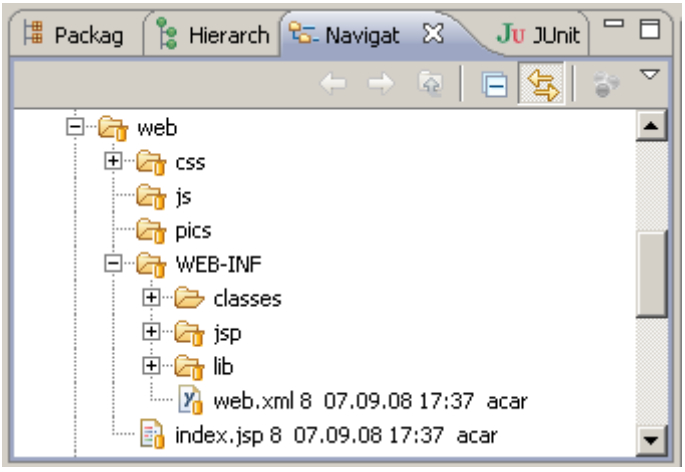
```java
package validation;

import org.springframework.validation.Errors;
import org.springframework.validation.Validator;

public class CustomerValidator implements Validator
{

	public boolean supports(Class clazz)
	{
		return clazz.equals(Customer.class);
	}

	public void validate(Object target, Errors errors)
	{
		Customer customer = (Customer)target;

		if(customer.getName().equals(""))
		{
			errors.rejectValue("name", "müşteri ismi eksik!");
		}
	}
}
```

Kod 13.3 de kullanılan command sınıfı Customer sınıfıdır. Bu sınıftan bir nesne validasyon için parametre olarak validate() metoduna aktarılır. validate() metodunda veriler kontrol edilir. Hatalı bir durum tespit edildiğinde errors değişkeni üzerinden hata mesajları oluşturulabilir.

Spring MVC Konfigürasyon Konseptleri

web.xml

Java web aplikasyonları belirli bir dizin yapısına sahiptir. Bunun bir örneğini resim 13.3 de görmekteyiz.



Resim 13.3 Spring MVC Controller hiyerarşisi

Web aplikasyonun konfigürasyonu için web.xml isminde bir dosya kullanılır. Bu dosya WEB-INF dizininde bulunur.

Spring MVC'yi kullanabilmemiz için öncelikli olarak web.xml bünyesinde DispatcherServlet'i tanımlamamız gerekmektedir.

Kod 13.4 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Shop</display-name>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>

  <servlet>
      <servlet-name>spring</servlet-name>
      <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                  classpath:spring-servlet.xml
            </param-value>
      </init-param>

  </servlet>


  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-servlet.xml</param-value>
  </context-param>



  <servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/app/*</url-pattern>
  </servlet-mapping>

</web-app>
```

Spring MVC için spring-servlet.xml (herhangi bir isim olabilir) isminde bir konfigürasyon dosyası oluşturulması gerekmektedir. DispatcherServlet ve diğer komponentler tarafından bu

konfigürasyon dosyanın kullanılabilmesi için ContextLoaderListener listener sınıfı kullanılır (<listener/>). Bu dosyanın bulunabilmesi için classpath içinde olması gerekmektedir, örneğin WEB-INF/classes dizininde. Bu dosyanın classpath içinde aranması gerektiğini contextConfigLocation değişkeni ile belirliyoruz. Son bölümde tanımladığımız servlet-mapping ile /app/* adresine gelen tüm requestler işlem görmek üzere DispatcherServlet'e gönderilir.

Handler Mappings

Kullanıcı isteklerinin hangi Controller tarafından işlem göreceği handler mapping ile tanımlanır.

Kod 13.5 spring-servlet.xml – handler mapping

```
<bean id="urlMap"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
      <property name="urlMap">
            <map>
                  <entry key="/login/login">
                        <ref bean="loginController" />
                  </entry>
                  <entry key="/home">
                        <ref bean="urlFilenamController" />
                  </entry>
            </map>
      </property>
</bean>
```

Kod 13.5 de oluşturduğumuz handler mapping listesi ile http://localhost/app/login/login adresinden loginController ismini taşıyan Spring Bean sorumludur.

View Resolver

Fiziksel JSP sayfalarıyla View elementleri arasındaki bağı oluşturmak için view resolver sınıfları kullanılır. Bunlardan birisi InternalResourceViewResolver sınıfıdır.

Kod 13.6 spring-servlet.xml – view resolver

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

Kod 13.6 da yer alan viewResolver ile tüm JSP sayfalarının /WEB-INF dizini içinde bulunan jsp dizininde olduğu ifade edilmektedir. Örneğin http://localhost/app/login/login şeklinde bir sayfa istendiğinde, Spring viewResolver yardımıyla istenen JSP sayfasının /WEB-INF/jsp/login dizininde bulunan login.jsp sayfası olduğunu tespit eder.

InternalResourceViewResolver kullanılarak JSP sayfaların WEB-INF altında bir dizine konmasıyla bu sayfaların web üzerinden direk erişilebilir olması engellenir. WEB-INF dizininde olan dosya ve diğer dizinler web üzerinden gösterime kapalıdır. Böylece JSP sayfaları direk gösterimden korunmuş olur. Bu sayfalara sadece spring-servlet.xml'de tanımlanan handler mapping üzerinden ulaşılabilir.

Spring MVC Kurulumu

Spring MVC kurulumu için spring.jar ve spring-webmvc.jar dosyalarının /WEB-INF/lib dizinine eklemesi gerekmektedir. Bu dosyalar http://www.springframework.org adresinden temin edilebilir.

Bunun yanı sıra JSP sayfalarında JSP tagleri kullanabilmek için JSTL paketinde bulunan jstl.jar ve standard.jar dosyalarının /WEB-INF/lib dizinine eklenmesi gerekmektedir. JSTL paketi http://jakarta.apache.org/taglibs adresinden temin edilebilir.

Kod 13.7 de yeralan spring-servlet.xml Spring MVC'nin en basit konfigüre edilmiş halidir.

Kod 13.7 spring-servlet.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:context="http://www.springframework.org/schema/context"
      xmlns:tx="http://www.springframework.org/schema/tx"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
      http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
      http://www.springframework.org/schema/tx
      http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">
```

```
      <bean id="urlMap"
            class="org.springframework.web.servlet.
                   handler.SimpleUrlHandlerMapping">
            <property name="urlMap">
                  <map>
                     <entry key="/home">
                        <ref bean="urlFilenameViewController" />
                     </entry>
                  </map>
            </property>
      </bean>


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

      <bean id="urlFilenameViewController"
               class="org.springframework.web.
                      servlet.mvc.UrlFilenameViewController" />
</beans>
```

Bir Controller sınıfına ihtiyaç duymayan view elementleri için UrlFilenameViewController Controller sınıfı kullanılır. Kod 13.7 de yer alan konfigürasyonda http://localhost/app/home adresinde yer alan home.jsp sayfası dinamik içeriği olmayan basit bir HTML sayfasıdır. urlMap ismiyle oluşturduğumuz handler mapping ile /home adresinden urlFilenameViewController isimli Spring Bean'i sorumlu kılıyoruz. Bu şu anlama gelmektedir: Kullanıcı tarafından http://localhost/app/home adresi istendiğinde, UrlFilenameViewController /WEB-INF/jsp dizinindeki home.jsp sayfasını view olarak kullanıcıya gönderir.

Kod 13.8 de web aplikasyonu konfigüre etmek için kullanılan web.xml dosyası bulunmaktadır. Bu dosya direk /WEB-INF dizini altında yeralır.

Kod 13.8 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
  http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

  <display-name>Shop</display-name>

  <listener>
    <listener-class>
      org.springframework.web.context.ContextLoaderListener
    </listener-class>
  </listener>
```

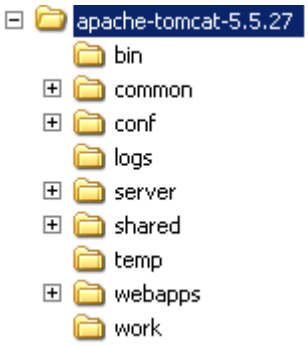
```
  <servlet>
      <servlet-name>spring</servlet-name>
      <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                  classpath:spring-servlet.xml
            </param-value>
      </init-param>

  </servlet>


  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-servlet.xml</param-value>
  </context-param>



  <servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/app/*</url-pattern>
  </servlet-mapping>



  <welcome-file-list>
    <welcome-file>index.HTML</welcome-file>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>

</web-app>
```

Kod 13.7 de yer alan spring-servlet.xml dosyasının Spring sınıfları tarafından lokalize edilebilmesi için /WEB-INF/classes dizinine eklenmesi gerekmektedir.

Tomcat ve Spring

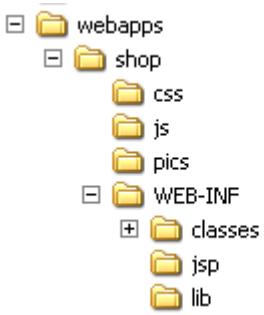
Servlet Container olarak Tomcat 5.5.x yada 6 sürümü kullanılabilir. Tomcat 5.5.x, Servlet 2.4 ve JSP 2.0 spesifikasyonunu, Tomcat 6 ise Servlet 2.5 ve JSP 2.1 spesifikasyonunu implemente etmektedir. Spring 2.5 için Tomcat 5.5.x ve üzeri gerekmektedir. Tomcat http://tomcat.apache.org adresinden temin edilebilir.

Tomcat'in 55.27 versiyonu http://tomcat.apache.org adresinden temin ettikten sonra, paketin içeriğini sistemdeki herhangi bir dizine açıyoruz. Dizin yapısı resim 13.4 de yer almaktadır.



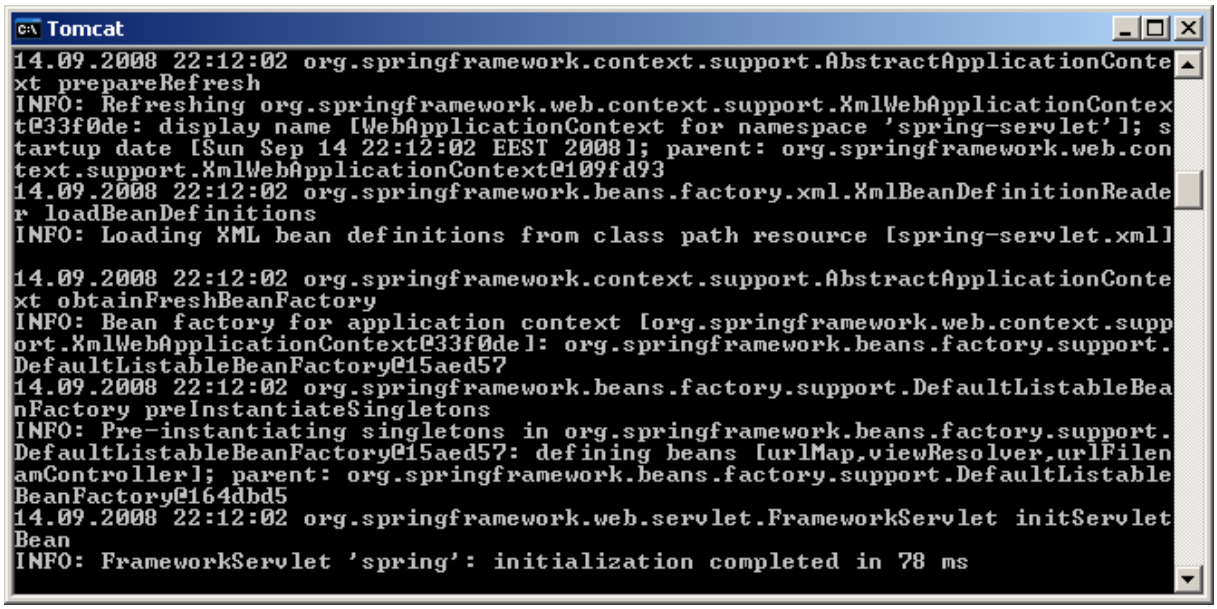
Resim 13.4 Tomcat 5.5.27 dizin yapısı

Tomcat, webapps dizininde bulunan web applikasyonlarını otomatik olarak çalışır hale getirir. Her web aplikasyonun kendi dizini içinde olması gerekmektedir. Resim 13.5 de yer aldığı gibi webapps dizini altında shop isminde yeni bir dizin oluşturuyoruz. Bu dizin bünyesinde, shop sistemi için gerekli tüm JSP sayfalarını, Controller sınıflarını ve xml konfigürasyon dosyalarını barındırır.

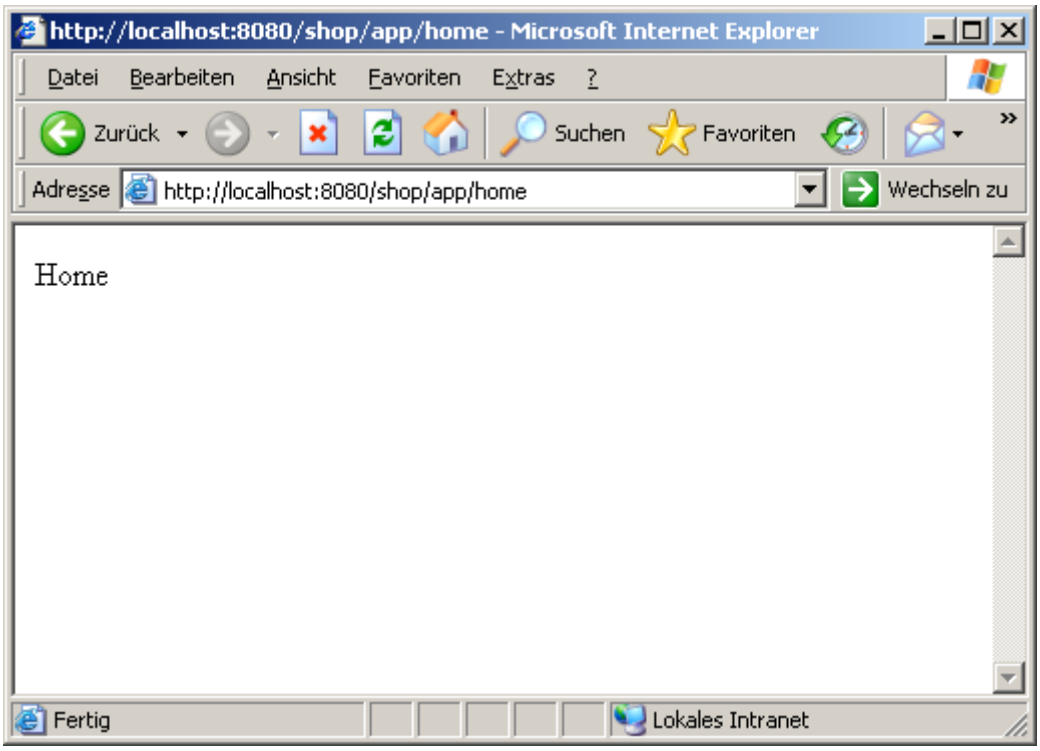


Resim 13.5 webapps dizin yapısı

Kod 13.8 de yer alan web.xml dosyasını webapps/shop/WEB-INF dizinine yerleştiriyoruz. Kod 13.7 de yer alan spring-servlet.xml dosyasının webapps/shop/WEB-INF/classes dizine yerleştirilmesi gerekmektedir. Bu hali ile web aplikasyonda bulunan home.jsp sayfasına Tomcat çalıştırıldıktan sonra http://localhost/shop/app/home adresinden ulaşabiliriz.



Resim 13.6 Tomcat console çıktısı



Resim 13.7 home.jsp

SimpleFormController Örneği

Bir örnek üzerinde, implementasyon detaylarına girmeden SimpleFormController sınıfının kullanımını inceleyelim. HTML formları için SimpleFormController sınıfı kullanılır. Sunduğu onSubmit() metodu ve kullanıcı tarafından forma girilen verilerin otomatik olarak command

nesnesine dönüştürülmesi, HTML formları için SimpleFormController sınıfının kullanımını ideal yapmaktadır.

Resim 13.8 Müşteri kayıt arayüzü

Resim 13.8 de müşteri kayıt arayüzü bulunmaktadır. Şimdi bu form için gerekli Controller sınıfı, command sınıfı ve spring-servlet.xml üzerinde yapmamız gereken konfigürasyonu yakından inceleyelim.

Öncelikle resim 13.8 de yer alan form için gerekli dosyaları tespit etmemiz gerekiyor. Bunlar:

- **register.jsp**: Gerekli formu ihtiva eder. /WEB-INF/jsp/register dizininde yer alır.
- **RegisterController.java**: Form için kullanılan Controller sınıfı. SimpleFormController sınıfını genişleterek bir SimpleFormController haline gelir. Bu sınıf shop.presentation.register.controller paketinde yer alır.
- **Customer.java**. Formda bulunan verileri tutmak için kullanılan sınıf. Bu sınıf Spring açısından bir command sınıfı, müşteri bilgilerini ihtiva ettiği için sistem açısından bir domain sınıfıdır (domain model). Bu sınıf shop.domain paketinde yer alır.

Fiziksel olarak bu dosyaları oluşturduğumuzu düşünelim. Müşterinin http://localhost/shop/app/register/register adresine girerek, forma ulaşabilmesi için, spring-servlet.xml dosyasında yapmamız gereken eklemeler şöyledir:

Kod 13.9 spring-servlet.xml

```
<bean id="urlMap"
      class="org.springframework.web.servlet.
             handler.SimpleUrlHandlerMapping">
      <property name="urlMap">
          <map>
              <entry key="/register/register">
                  <ref bean="registerController" />
              </entry>
              <entry key="/home">
                  <ref bean="urlFilenamController" />
              </entry>
          </map>
      </property>
</bean>
```

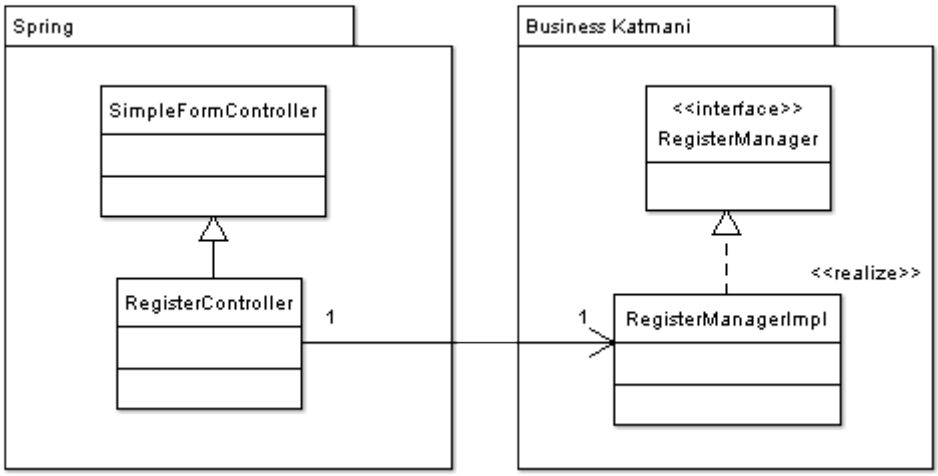
register.jsp sayfasını web üzerinden erişilebilir hale getirmek için handler mapping kaydı oluşturmamız gerekiyor. urlMap ismini taşıyan bean tanımlaması ile sistemdeki hangi JSP sayfalarından hangi Controller sınıflarının sorumlu olduğu tanımlanır. Buna göre /shop/app/register/register adresi istendiği zaman, sorumlu Controller sınıfı olarak registerController seçilir. registerController tanımlaması kod 13.10 da yer almaktadır.

Kod 13.10 spring-servlet.xml

```
<bean id="registerController"
        class="shop.presentation.register.controller.RegisterController">
    <property name="formView" value="/register/register"/>
    <property name="successView" value="redirect:/app/home"/>
    <property name="commandName" value="Customer"/>
    <property name="commandClass" value="shop.domain.Customer"/>
</bean>
```

Formlar için oluşturulması gereken genel yapı kod 13.10 da yer almaktadır. Kullanılan Controller sınıfını ihtiva eden bir <bean/> tagı oluşturulması gerekmektedir. Bu tag içinde kullanılan atributlar şöyledir:

- **id**: Bu tanımlanan Bean'in ismidir. Başka Bean tanımlamaları ref kelimesiyle bu Bean'i id atributuyla referense edebilir.
- **property**: Sınıf bünyesinde bulunan değişkenlere değer atamak için kullanılır.
- **formView**: HTML formun bulunduğu arayüz (register.jsp). Hata oluşması durumunda formView gösterime girer, yani Controller sınıfı kontrolü bu view elementine devreder. Böylece hata oluşması durumunda, kullanıcıya HTML formun tekrar gösterilmesi sağlanmış olur.
- **successView:** Eğer hata oluşmadan form tamamlanırsa, Controller sınıfı successView değişkenindeki view elementine kontrolü devreder.
- **commandName**: Kullanılan command nesnesinin ismi. Bu isim JSP sayfasında <spring:bind/> tagı kullanılarak, HTML form ile Controller ve model sınıfı arasında bağı oluşturmak için kullanılır.
- **commandClass**: Kullanılan command sınıfı. Her form için bu sınıftan bir nesne oluşturulur ve kullanıcı tarafından girilen veriler bu nesnenin değişkenlerine eşitlenir.



Resim 13.9 RegisterController ve işletme (business) katmanı arasındaki ilişki

RegisterController gösterim katmanında olan bir sınıftır. Tipik web aplikasyonlarında üç katmanlı mimari kullanılır. Gösterim (presentation) katmanı yanı sıra, asıl işlemlerin yapıldığı işletme (business) ve persistens (persistence) katmanları mevcuttur.

Kod 13.11 spring-servlet.xml

```
<bean id="registerController"
          class="shop.presentation.login.
                 controller.RegisterController">
      <property name="formView" value="/login/login"/>
      <property name="successView" value="redirect:/app/home"/>
      <property name="commandName" value="Customer"/>
      <property name="commandClass" value="shop.domain.Customer"/>
      <property name="manager">
            <ref bean="registerManager" />
      </property>
</bean>

<bean id="registerManager"
                class="shop.business.register.RegisterManagerImpl"/>
```

Resim 13.9 da görüldüğü gibi RegisterController sınıfı, kayıt işlemini gerçekleştirmek için işletme katmanında bulunan RegisterManagerImpl sınıfını kullanmaktadır. Bu iki sınıf arasındaki ilişki kod 13.11 de yer aldığı şekilde tanımlanır. registerController bünyesinde <ref/> tagı yardımıyla registerManager'i referense edebiliriz. Sınıf bazında bu iki sınıf arasındaki bağı oluşturabilmek için RegisterController sınıfında RegisterManager tipinde manager ismini taşıyan bir değişken tanımlamamız gerekiyor. Bu durumda Spring tarafından RegisterController sınıfından bir nesne oluşturulduğunda, otomatik olarak RegisterManager sınıfından oluşturulan bir nesne RegisterController nesnesine enjekte edilir.

Spring MVC ve Test Edilebilirlik

Spring MVC'yi seçmemin sebebi, web tabanlı bir projede test güdümlü implementasyonu mümkün kılmasıdır. Birçok web frameworkü için bunu söylemek mümkün değil. Çoğu zaman kendi imkanlarınızla sistemi test etmeye mahkum bırakılıyorsunuz ya da test edilebilecek ortamın kurulumu ve çalıştırılması zaman alıcı oluyor. TDD (Test Driven Development = test güdümlü yazılım) yazılım konseptleriyle tanışmadan önce, web aplikasyonlarını test etmenin benim için çok emek gerektiren bir süreç olduğunu söylemem gerekir. Çalıştığım projelerde çoğu zaman Cactus[12] gibi bir framework kullanarak sistem testleri yapılıyordu. Cactus çalışır bir web aplikasyonunu gerekli kılmaktadır. Bu durumda testleri koşturabilmek için tüm web aplikasyonunu Servlet container (Tomcat) içinde çalışır hale getirmemiz gerekir. Bu şekilde yapılan testler hem hatalara çok açıktır hemde çok zaman alıcıdır. Böyle test olacaksa, hiç olmasın demeye kadar geliyorsunuz zaman zaman ☺.

Tabi Spring ve diğer test edilebilir frameworklerin geliştirilmesiyle biz programcıların hayatı çok daha kolaylaştı. TDD'yi benimsemiş programcılar için Spring MVC ve Spring tarzı yapılar gerçekten bir hediyedir, çünkü test güdümlü çalışmak kolay bir hale gelmiştir.

Spring MVC ile herhangi bir Servlet container olmadan web aplikasyonu test edebiliriz. Eğer Servlet container yoksa, o zaman HttpServletRequest ve HttpServletResponse nesnelerini kendimiz oluşturmamız gerekiyor, çünkü Servlet container işlem esnasında bu sınıflardan nesneler oluşturur. Spring bu noktada yardımımıza koşuyor ve MockHttpServletRequest ve MockHttpServletResponse isminde iki mock sınıf ile tüm HTTP trafiğini simule etmemizi kolaylaştırıyor. Bu iki mock sınıfı kullanarak, Servlet container varmışcasına web aplikasyonlarımızı, yani Controller ve diğer sınıfları test edebiliriz. Bunun bir örneği kod 13.12 de yer almaktadır.

Kod 13.12 MockHttpServletRequest kullanımı

```
public void testLoginUsernameAndPasswordEmpty()
{
      try
      {
            MockHttpServletRequest request = null;
            MockHttpServletResponse response = null;

            request = new MockHttpServletRequest("POST", LOGIN_PAGE);
            response = new MockHttpServletResponse();

            request.setParameter("email","");
            request.setParameter("password","" );

            ModelAndView modelAndView =
                  controller.handleRequest( request, response);

            BeanPropertyBindingResult errors =
              (BeanPropertyBindingResult)modelAndView.getModelMap()
              .get("org.springframework.validation.BindingResult." +
                        "Customer");

            assertTrue(errors.getAllErrors().size() == 1);
```

[12] Bakınız: jakarta.apache.org/cactus/index.html

```
            assertEquals(((ObjectError)errors
                  .getAllErrors().get(0)).getCode(),INVALID_LOGIN);

      }
      catch (Exception e)
      {
            e.printStackTrace();
            fail();
       }
}
```

Kod 13.12 de yer alan JUnit testi LoginController sınıfını test etmek için oluşturulmuştur. Eğer kullanıcı isim ve şifresini girmeden login butonuna tıklarsa, sistem nasıl reaksiyon gösterecektir, bu test edilmektedir. Görüldüğü gibi setParameter() metoduyla request (MockHttpServletRequest) nesnesini istediğimiz şekilde yapılandırabiliyoruz. Bu bizim kullanıcı ile Servlet container arasındaki trafiği simule etmemizi kolaylaştırmaktadır. LoginController sınıfında bulunan handleRequest() metodu, mock nesne ve gerçek HttpServletRequest arasında ayrım yapamaz ve ona gönderdiğimiz veriler doğrultusunda işlemi gerçekleştirir. Bu işlem sonucunda oluşan verileri kontrol ederek (assert) işlemin LoginController sınıfı tarafından doğru yapılıp, yapılmadığını tespit edebiliriz.

Görüldüğü gibi Spring'in sunduğu mock sınıfları kullanarak, tüm HTTP trafiğini simule edebiliriz. Bu bize, tüm aplikasyonu bir Servlet container içinde çalışır duruma getirmeden test etmek imkanı sağlamaktadır. Ayrıca bu mock sınıflar test güdümlü implementasyonu kolaylaştırdıkları için Spring MVC, TDD ve XP uyumlu bir web frameworkdür diyebiliriz.

EOF (End Of Fun)

Özcan Acar
Bilgisayar Mühendisi
KurumsalJava.com
KurumsalJavaAkademisi.com