



­

Assignment Document:

Spring 3 MVC

Version: Spring 3 MVC/ASSIGNMENT/xxxx/x.x

Date: 17-01-2012

Cognizant

500 Glen Pointe Center West

Teaneck, NJ 07666

Ph: 201-801-0233  
[www.cognizant.com](http://www.cognizant.com)

Contents

[Topic: Spring Controller…………………………………………………………………3](#_Toc309826937)

[Guided Assignments…………………………………………………………………………………… 3](#_Toc309826938)

[Guided Exercise 1: Hello World! …………………………………………………………………… 3](#_Toc309826939)

[Guided Exercise 2: Spring MVC and Hibernate …………………………………………..…. 5](#_Toc309826940)

[Topic: Handler Mapping…………………………………………………………………..15](#_Toc309826941)

[Guided Exercise 3: User Login………………………………………………………………………15](#_Toc309826942)

[Guided Exercise 4: User Details…………………………………………………………………….22](#_Toc309826943)

[Topic: Handler Interceptor……………………………………………………………24](#_Toc309826944)

[Guided Exercise 5: Logging Interceptor…………………………………………………………...24](#_Toc309826945)

[Topic: Validator…………………………………………………………………………….26](#_Toc309826946)

[Guided Exercise 6: User Data Validation…………………………………………………………26](#_Toc309826947)

[Case Study Assignments – Spring 3 MVC……………………………………………28](#_Toc309826948)

[Case Study 1: Time Sheet Management……………………………………………………………28](#_Toc309826949)

[Maintenance Project (From Spring 2 to Spring 3)-(Optional)……………28](#_Toc309826950)

[Case Study 1: Product List Maintenance………………………………………………………….28](#_Toc309826951)

[Case Study 2: Online Stock Trading System………………………………………………………31](#_Toc309826952)

Topic: Spring Controller

### Guided Assignments

Guided Exercise 1: Hello World!

**Estimated Completion Time: 15 Minutes**

(15 Marks)

**Objective**: To write a simple hello world program using Spring MVC 3.

**Concept**: To understand how to configure simple Spring MVC 3.

**Step 1**: Create a dynamic web project using SDE/Eclipse/Netbeans.

**Step 2**: Add below configuration to web.xml file:

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

id="WebApp\_ID" version="2.5">

<display-name>Spring3MVC</display-name>

<welcome-file-list>

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

</welcome-file-list>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

</web-app>

**Step 3**: Create spring-servlet.xml and put in WEB-INF directory of your web application with the following contents:

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

<http://www.springframework.org/schema/context/spring-context> 3.0.xsd">

<context:component-scan

base-package="spring3.controller" />

<bean id="viewResolver”

class="org.springframework.web.servlet.view.UrlBasedViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView" />

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

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

</bean>

</beans>

Step 4: Write a simple controller class as follows:

package spring3.controller;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.servlet.ModelAndView;

@Controller

public class HelloWorldController {

@RequestMapping("/hello")

public ModelAndView helloWorld() {

String message = "Hello World, Spring 3.0!";

System.out.println(message);

return new ModelAndView("hello", "message", message);

}

}

Step 5: Create hello.jsp under WEB-INF/jsp folder.

<html>

<head>

<title>Hello World</title>

</head>

<body>

${message}

</body>

</html>

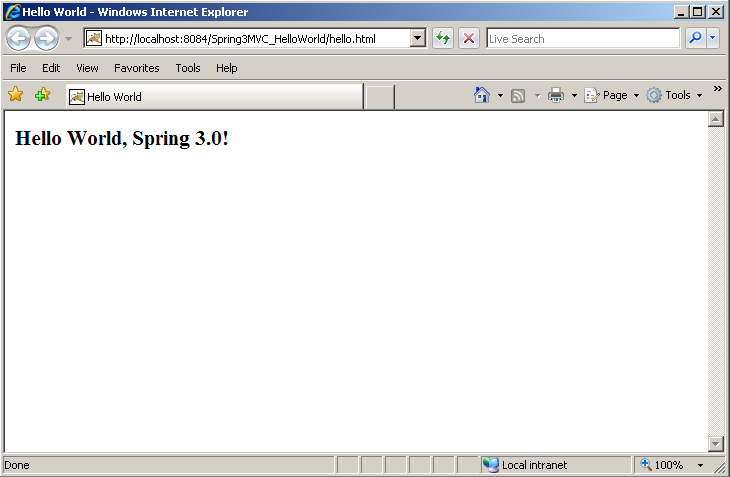
Step 6: Deploy the war into Tomcat and select run on server.

Step 7: Create index.jsp(welcome file) under the web context root.

<%@page contentType="text/html" pageEncoding="UTF-8"%>

<% response.sendRedirect("hello.html"); %>

Step 8: Deploy the webapplication and run.



**Summary of this exercise:**

You have just learnt:

* How to configure spring MVC 3 in your web application.
* How to write simple controller.
* Source Code: 

Guided Exercise 2: Spring MVC and Hibernate

**Estimated Completion Time: 40 Minutes**

(15 Marks)

**Objective**: To develop a web application to add and list articles in to the repository(using Hibernate).

**Concept**: To understand how to develop a presentation tier components and use business objects(Business delegate, DAOs) to perform business logic.

**Step 1**: Create a dynamic web project using SDE/Eclipse/Netbeans.

**Step 2**: Add below configuration to web.xml file.

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd">

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class> org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>\*.html</url-pattern>

</servlet-mapping>

<welcome-file-list>

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

</welcome-file-list>

</web-app>

**Step 3**: Create dispatcher-servlet.xml and put in WEB-INF directory of your web application.

<?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-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd">

<context:property-placeholder location="classpath:jdbc.properties" />

<context:component-scan base-package="spring3" />

<tx:annotation-driven

transaction- manager ="hibernateTransactionManager"/>

<bean id="jspViewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView" />

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

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

</bean>

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="${database.driver}" />

<property name="url" value="${database.url}" />

<property name="username" value="${database.user}" />

<property name="password" value="${database.password}" />

</bean>

<bean id="sessionFactory"

class= "org.springframework.orm.hibernate3.annotation.

AnnotationSessionFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="annotatedClasses">

<list>

<value>spring3.model.Article</value>

</list>

</property>

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">${hibernate.dialect}</prop>

<prop

key="hibernate.show\_sql">${hibernate.show\_sql}</prop>

<prop key="hibernate.hbm2ddl.auto">create</prop>

</props>

</property>

</bean>

<bean id="hibernateTransactionManager"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

</beans>

Step 4: Create a controller class ArticleController.

package spring3.controller;

import java.util.HashMap;

import java.util.Map;

import spring3.model.Article;

import spring3.service.ArticleService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Controller;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.ModelAttribute;

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

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.servlet.ModelAndView;

@Controller

@RequestMapping("/articles")

public class ArticleController {

@Autowired

private ArticleService articleService;

@RequestMapping(value = "/save", method = RequestMethod.POST)

public ModelAndView saveArticle(@ModelAttribute(" article") Article article,

BindingResult result) {

articleService.addArticle( article);

return new ModelAndView("redirect:/articles.html");

}

@RequestMapping(method = RequestMethod.GET)

public ModelAndView listArticles() {

Map<String, Object> model = new HashMap<String, Object>();

model.put("articles", articleService.listArticles());

return new ModelAndView("articlesList", model);

}

@RequestMapping(value = "/add", method = RequestMethod.GET)

public ModelAndView addArticle(@ModelAttribute("article") Article article,

BindingResult result) {

return new ModelAndView("addArticle");

}

}

Step 5:Create DAO classes- ArticleDao and ArticleDaoImpl.

import spring3.model.Article;

public interface ArticleDao {

// To Save the article detail

public void saveArticle ( Article Article );

// To get list of all articles

public List<Article> listArticles();

}

package spring3.dao;

import java.util.Date;

import java.util.List;

import spring3.model.Article;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

@Repository("articleDao")

public class ArticleDaoImpl implements ArticleDao {

@Autowired

private SessionFactory sessionFactory;

// To Save the article detail

public void saveArticle(Article article) {

article.setAddedDate(new Date());

sessionFactory.getCurrentSession().saveOrUpdate(article);

}

// To get list of all articles

@SuppressWarnings("unchecked")

public List<Article> listArticles() {

return (List<Article>) sessionFactory.getCurrentSession().createCriteria(Article.class).list();

}

}

Step 6: Write a Model class – Article.

package spring3.model;

import java.util.Date;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "article")

public class Article {

@Id

@GeneratedValue

@Column(name = "article\_id")

private Long articleId;

@Column(name = "article\_name", nullable = false, length=20)

private String articleName;

@Column(name = "article\_desc", nullable = false)

private String articleDesc;

@Column(name = "date\_added")

private Date addedDate;

public Article() {

}

public Long getArticleId() {

return articleId;

}

public void setArticleId(Long articleId) {

this.articleId = articleId;

}

public String getArticleName() {

return articleName;

}

public void setArticleName(String articleName) {

this.articleName = articleName;

}

public String getArticleDesc() {

return articleDesc;

}

public void setArticleDesc(String articleDesc) {

this.articleDesc = articleDesc;

}

public Date getAddedDate() {

return addedDate;

}

public void setAddedDate(Date addedDate) {

this.addedDate = addedDate;

}

}

Step 7: Write service classes – ArticleService, ArticleServiceImpl.

package spring3.service;

import java.util.List;

import spring3.model.Article;

public interface ArticleService {

public void addArticle(Article article);

public List<Article> listArticles();

}

package spring3.service;

import java.util.List;

import spring3.dao.ArticleDao;

import spring3.model.Article;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Propagation;

import org.springframework.transaction.annotation.Transactional;

@Service("articleService")

@Transactional(propagation = Propagation.SUPPORTS, readOnly = true)

public class ArticleServiceImpl implements ArticleService {

@Autowired

private ArticleDao articleDao;

public ArticleServiceImpl() {

}

@Transactional(propagation = Propagation.REQUIRED, readOnly = false)

public void addArticle(Article article) {

articleDao.saveArticle(article);

}

public List<Article> listArticles() {

return articleDao.listArticles();

}

}

Step 8: Write addArticle.jsp, articleList.jsp under WEB-INF/jsp.

addArticle.jsp

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>

<html>

<head><title>Add Article</title></head>

<body>

<h1>Add Article</h1>

<c:url var="viewArticlesUrl" value="/articles.html" />

<a href="${viewArticlesUrl}">Show All Articles</a>

<br />

<br />

<c:url var="saveArticleUrl" value="/articles/save.html" />

<form:form modelAttribute="article" method="POST" action="${saveArticleUrl}">

<form:label path="articleName">Article Name:</form:label>

<form:input path="articleName" />

<br />

<form:label path="articleDesc">Article Desc:</form:label>

<form:textarea path="articleDesc" />

<br />

<input type="submit" value="Save Article" />

</form:form>

</body>

</html>

articleList.jsp

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<html>

<head>

<title>All Articles</title>

</head>

<body>

<h1>List Articles</h1>

<a href="articles/add.html">Add Article</a>

<c:if test="${!empty articles}">

<table>

<tr>

<th>Article ID</th>

<th>Article Name</th>

<th>Article Desc</th>

<th>Added Date</th>

</tr>

<c:forEach items="${articles}" var="article">

<tr>

<td><c:out value="${article.articleId}"/></td>

<td><c:out value="${article.articleName}"/></td>

<td><c:out value="${article.articleDesc}"/></td>

<td><c:out value="${article.addedDate}"/></td>

</tr>

</c:forEach>

</table>

</c:if>

</body>

</html>

Step 9: Create jdbc.properties file under the classpath with the following entries.

database.driver=org.apache.derby.jdbc.ClientDriver

database.url=jdbc:derby://localhost:1527/article\_db

database.user=app

database.password=app

hibernate.dialect=org.hibernate.dialect.DerbyDialect

hibernate.show\_sql=true

Step 10: Create index.jsp(welcome file) under the web context root.

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Spring 3 MVC and Hibernate 3</title>

</head>

<body>

<h1>Spring 3 MVC and Hibernate 3</h1>

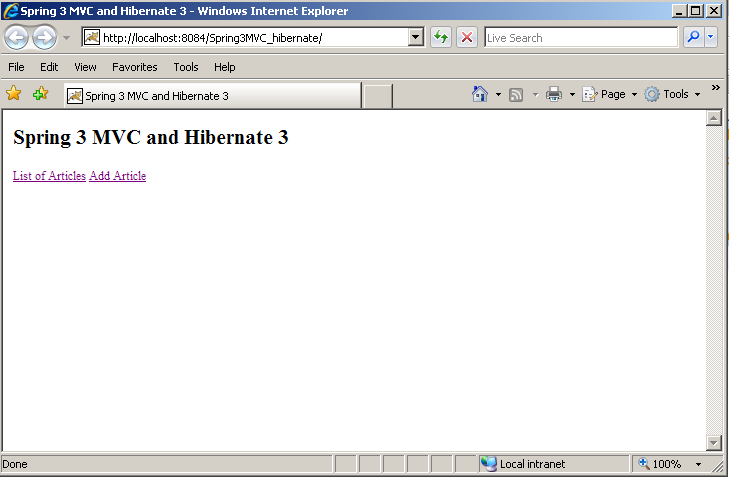
<a href="articles.html">List of Articles</a>

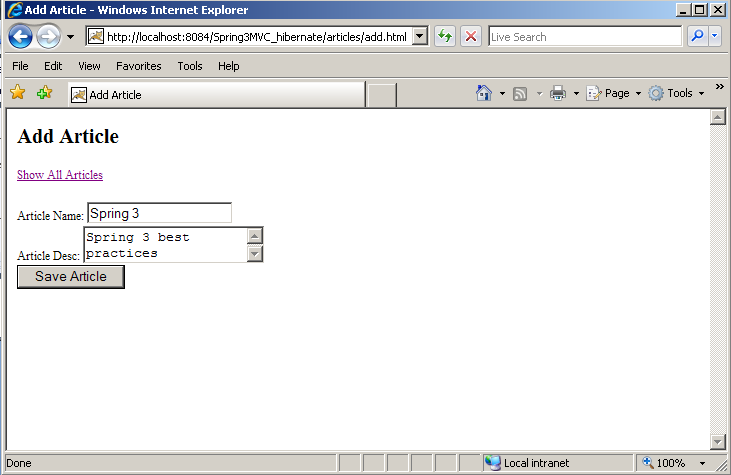
<a href="articles/add.html">Add Article</a>

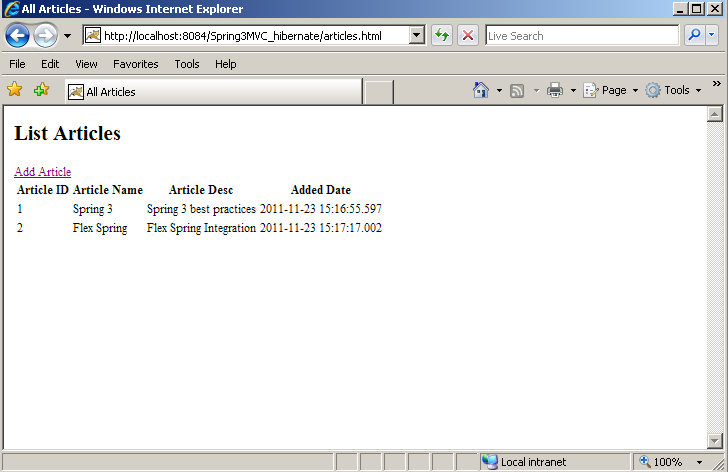
</body>

</html>

Step 11: Deploy the webapplication and run.







Source:



**Summary of this exercise:**

You have just learnt:

* How to develop a presentation tier components and use business objects (Business delegate, DAOs) to perform business logic.

Topic: Handler Mapping

Guided Exercise 3: User Login

**Estimated Completion Time: 40 Minutes**

(15 Marks)

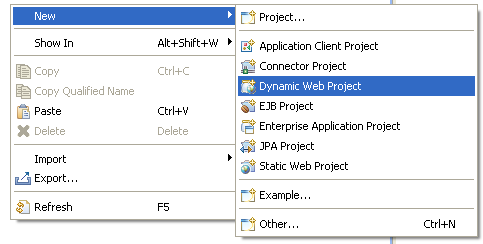
**Objective**: To write web application that accepts user name and redirects to success page or error page.

**Concept**: To understand the working of Handler Mapping

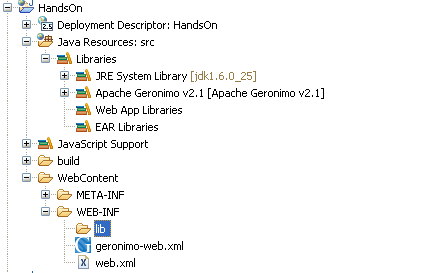
Step 1: Open eclipse.

Step 2: Setup a server in eclipse. Here the Apache Geronimo is added as web server.

In the Project Explorer pane, create Dynamic Web Project.



It will create a Web project and if you’ll expand the project, it will look like below.



Step 3: Add the require jars under the WEB-INF/lib folder.



You can download the jars from Spring’s official site.

Step 4: Create a new folder under WEB-INF folder and name it as *jsp*.

Step 5: Create a package under the SRC folder as *com.cts.controller.*

Step 6: Create a class and name it as *MyController.*

Step 7: Have the content of the *MyController* as below.

package com.cts.controller;

import org.springframework.stereotype.Controller;

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

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.servlet.ModelAndView;

@Controller

public class MyController {

@RequestMapping("loginApplication")

@RequestMapping("loginApplication")

public ModelAndView loginApplication(@RequestParam("userName") String userName) {

ModelAndView result;

if("user".equals(userName)){

result= new ModelAndView("welcomePage");

}

else{

result= new ModelAndView("index");

}

return result;

}}

Step 8: Edit the web.xml as below.

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"

version="2.5">

<display-name>Spring3MVC</display-name>

<welcome-file-list>

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

</welcome-file-list>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/action/\*</url-pattern>

</servlet-mapping>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/spring-servlet.xml

</param-value>

</context-param>

</web-app>

Step 9: Create a file under WEB-INF and name it as *spring-servlet.xml*

Edit the contents of *spring-servlet.xml*  as below.

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

xmlns:secure="http://www.springframework.org/schema/security"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-3.0.xsd

http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.0.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.1.xsd">

<context:component-scan base-package="com.cts" />

<bean id="jspViewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView" />

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

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

</bean>

</beans>

Step 10: Create a index.jsp under the webcontent folder and edit the file with the contents as:

<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form"%>

<html>

<head>

<title>Login to Time Sheet Application </title>

</head>

<body style="background-color:#ffffE9">

<div style="background-color:#1010ff; align:center; margin-top:200px; width:400px; margin-left:450px">

<div style="margin-top:20px" >

<span style="align:left; padding-left:15px; font-size:22px; color:white"> Please Login to enter into the application </span>

</div>

<form action="action/loginApplication" method="post">

<table style="margin-top:30px; align:center; margin-left:70px; color:white">

<tr><td>Username</td><td><input name="userName" /></td></tr>

<tr><td colspan="2" align="right"><input type="submit" value="Login" /></td></tr>

</table>

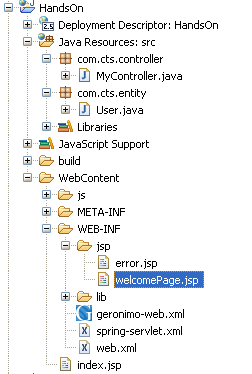
</form>

</div>

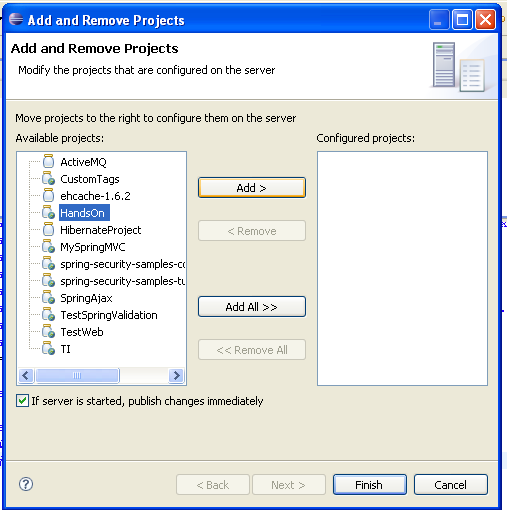
</body>

</html>

Step 11: Create the JSP files under the JSP folder and the final folder structure will be as below.

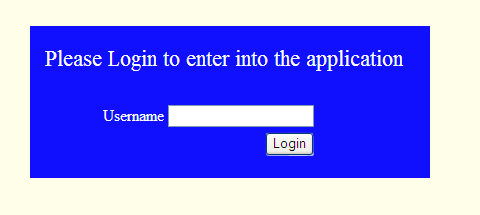


Step 12: Right click on the server and select the option Add and remove projects. Add the project *HandsOn* to the server.



Step 13: Start the server.

Step 14: Open the browser and type the URL <http://localhost:8081/HandsOn/index.jsp>



If the user name is “user”, then it will redirect to Welcome page or it will redirect to Error page.

**Summary of this exercise:**

You have just learnt:

* How to configure a handler mapping.

Guided Exercise 4: User Details

**Estimated Completion Time: 20 Minutes**

(15 Marks)

**Objective**: To create a web application which will accept the user details and print the entered details in the result page. (Continuation of Guided Exercise 3)

**Concept:** To understand the working of Handler Mapping

Step 1: Add the method in the MyController Class.

@RequestMapping("getAscDetails")

public ModelAndView getAscDetails(@ModelAttribute User user) {

ModelAndView modelAndView = new odelAndView("userDetails");

modelAndView.addObject("user", user);

return modelAndView;

}

Step 2: Create a package com.cts.entity and create a java class under this package and name it as User.java.

package com.cts.entity;

public class User {

private String firstName="";

private String secondName="";

private Long contactNumber;

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getSecondName() {

return secondName;

}

public void setSecondName(String secondName) {

this.secondName = secondName;

}

public Long getContactNumber() {

return contactNumber;

}

public void setContactNumber(Long contactNumber) {

this.contactNumber = contactNumber;

}

}

Step 3: Edit the contents of the welcomePage.jsp as below.

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib prefix="form"

uri="http://www.springframework.org/tags/form"%>

<html>

<head>

<meta http-equiv="Content-Type"

content="text/html; charset=ISO-8859-1">

<title>Enter User Details</title>

</head>

<body>

<form:form method="post" action="getAscDetails"

modelAttribute="user">

<form:errors path="\*" cssClass="error" />

<table>

<tr>

<td>

First Name:

</td>

<td><form:input path="firstName" /></td>

</tr>

<tr>

<td>

Second Name:

</td>

<td><form:input path="secondName" /></td>

</tr>

<tr>

<td>

Contact Number:

</td>

<td><form:input path="contactNumber" /></td>

</tr>

<tr>

<td><input type="submit" value="Submit" /></td>

</tr>

</table>

</form:form>

</body>

</html>

Step 4: Create a userDetails.jsp and have the contents as below.

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<%@ taglib uri='http://java.sun.com/jsp/jstl/core' prefix='c'%>

<html>

<head>

<meta http-equiv="Content-Type"

content="text/html; charset=ISO-8859-1">

<title>User Details</title>

</head>

<body>

First Name: <c:out value="${user.firstName}" /><br>

Last Name: <c:out value="${user.secondName}" /> <br>

Contact Number: <c:out value="${user.contactNumber}"></c:out>

</body>

</html>

Step 5: Verify the application in your browser.

**Summary of this exercise:**

You have just learnt:

* How to configure a handler mapping and display user entered data.

Topic: Handler Interceptor

Guided Exercise 5: Logging Interceptor

**Estimated Completion Time: 40 Minutes**

(15 Marks)

**Objective**: To write an interceptor that will intercept the requests and print the logger message.

**Concept**: To understand how to intercept the request using Handler Interceptor

Step 1: Create a java class and name it as LoggerInterceptor.

Step 2: Edit the file and have the content as below.

package com.cts.logger;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.log4j.BasicConfigurator;

import org.apache.log4j.Logger;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

public class LoggerInterceptor extends HandlerInterceptorAdapter {

static Logger logger = Logger.getLogger(LoggerInterceptor.class);

public LoggerInterceptor()

{

BasicConfigurator.configure();

}

@Override

public boolean preHandle(

HttpServletRequest request,

HttpServletResponse response, Object handler) throws Exception {

logger.info("Before handling the request");

return super.preHandle(request, response, handler);

}

@Override

public void postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) throws Exception {

logger.info("After handling the request");

super.postHandle(request, response, handler, modelAndView);

}

@Override

public void afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler,

Exception ex)

throws Exception {

logger.info("After rendering the view");

super.afterCompletion(request, response, handler, ex);

}

}

Step 3: Edit the spring-servlet.xml as:

<bean id="loggerInterceptor"

class="com.cts.logger.LoggerInterceptor" />

<bean

class="

org.springframework.web.servlet.mvc.annotation.

DefaultAnnotationHandlerMapping"

p:interceptors-ref="loggerInterceptor" />

Step 4: Verify the application. After triggering any action, check the console. You will get the log messages on the console.

**Summary of this exercise:**

You have just learnt:

* How to create a Handler Interceptor and intercept requests.

Topic: Validator

Guided Exercise 6: User Data Validation

**Estimated Completion Time: 40 Minutes**

(15 Marks)

**Objective**: To validate the user input values. (Continuation of Guided Exercise 4)

* + First Name should not be empty.
  + First Name should not have special characters.
  + Contact Number should not be empty.

**Concept**: To understand the Spring Validator framework.

Step 1: Create a package com.cts.validator in src folder.

Step 2: Create a class UserValidator and edit the file as below.

package com.cts.validator;

import org.apache.commons.lang.StringUtils;

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validator;

import com.cts.entity.User;

@Component

public class UserValidator implements Validator{

public boolean supports(Class clazz) {

return User.class.equals(clazz);

}

public void validate(Object target, Errors errors) {

ValidationUtils.rejectIfEmptyOrWhitespace(

errors, "firstName",

"required.user.first.Name", "\* User First name Required");

ValidationUtils.rejectIfEmptyOrWhitespace(

errors, "contactNumber",

"required.user.contact.Number",

"\* User Contact Number Required");

User user = (User)target;

String ascFstName = user.getFirstName();

if(!errors.hasFieldErrors("firstName"))

{

if(!StringUtils.isAlphaSpace(ascFstName))

{

errors.rejectValue("firstName",

"required.firstName.special.char.notAllowed","\* User name should not have special characters");

}

}

}

}

Step 3: Create a folder css in WebContent.

Step 4: Create a file under css folder and name it as styles.css and the have the file contents as below.

.error {

color: #ff0000;

font-weight: bold;

}

Step 5: Edit the MyController by adding new property userValidator as below.

@Autowired

UserValidator userValidator;

Step 6: Edit the getAscDetails method of MyController as below.

@RequestMapping("getAscDetails")

public ModelAndView getAscDetails(

@ModelAttribute User user,

BindingResult result,Model model) {

userValidator.validate(user, result);

if(result.hasFieldErrors())

{

ModelAndView modelAndView = new odelAndView("welcomePage");

modelAndView.addObject("user",user);

return modelAndView;

}

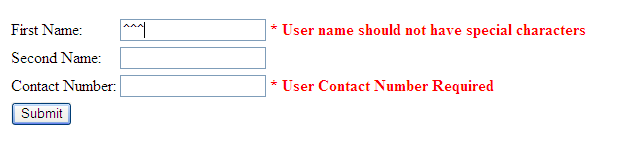
ModelAndView modelAndView = new odelAndView("userDetails");

modelAndView.addObject("user", user);

return modelAndView;

}

Step 7: Verify the application in your browser. You will get the warning messages as below.



**Summary of this exercise:**

You have just learnt:

* How to use Spring Validator to validate user data.

Case Study Assignments – Spring 3 MVC

Case Study 1: Time Sheet Management

Please refer to the following document for the case study with solution. This case study should be discussed with the participants during the session.



Maintenance Project (From Spring 2 to Spring 3)-(Optional)

The below 2 case studies have solution in Spring 2.0 MVC. Particpants are requested to re-factor the code and convert it into Spring 3.0 MVC specific. Use appropriate Spring 3.0 MVC annotations.

Case Study 1: Product List Maintenance

**Estimated Completion Time: 120 Minutes**

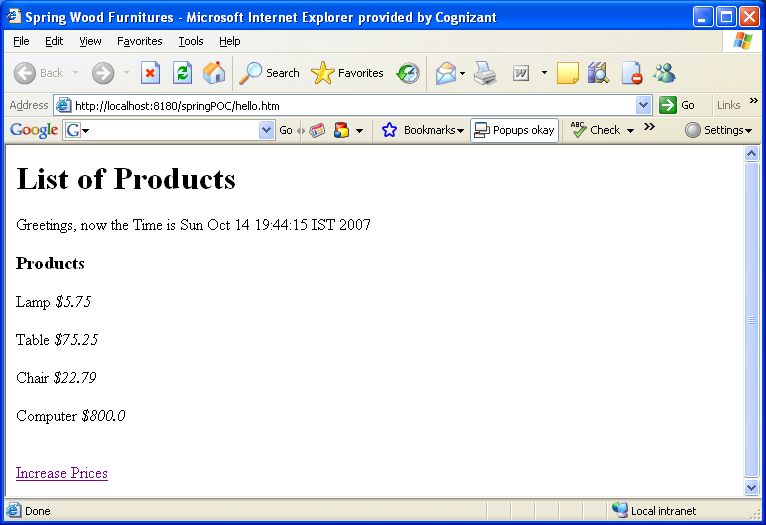
(100 Marks)

**Objective**: To create a web application to increase the price of the products.

**Concept**: An XYZ product sales company provides online sales of its products. A web application need to be created to allow the administrator to increase the price (in %) of all the products.

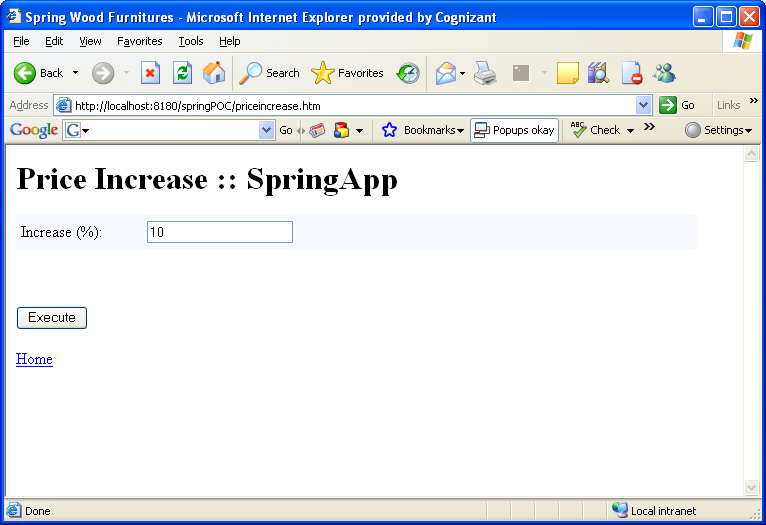
**Case:**

The home page should display a list of products and their prices (configurable via DI) and provide an Increase Prices link.



Clicking on Increase Prices link should take the user to the price Increase page. The default value of Increase % field should be 10, but should be configurable during deployment.

There should also be a link for Home page provided in the price increase page.

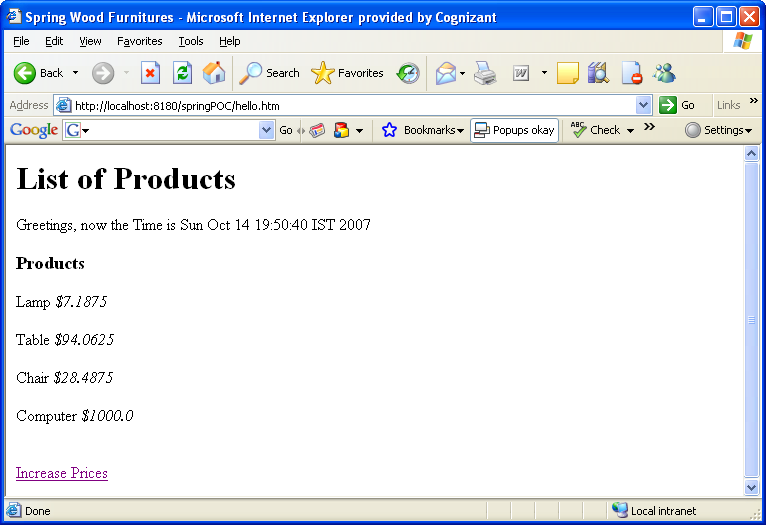


After the user has provided the percentage value and clicks Execute button, the following validations should be performed:

* Increase % cannot be blank or non-numeric.
* Increase % Cannot be less than a minimum value (configurable via DI).
* Increase % Cannot exceed maximum value (configurable via DI).



On successful price increase, the user should be taken back to the Home Page. The Home page should reflect the price increase.



**Solution:**

<Please provide the solution to the case study>

Evaluation Rubrics

|  |  |
| --- | --- |
| Parameters | Weightage |
| Completeness | 60 |
| Accuracy | 15 |
| Clarity of understanding | 15 |
| Presentation | 10 |
| Total | 100 |

**Summary of this Case Study:**

You have just learnt:

* How to use Controller and SimpleFormControlelr
* How to use Spring Validator.
* How to use view resolvers.
* How to use Resource Bundle
* 

Case Study 2: Online Stock Trading System

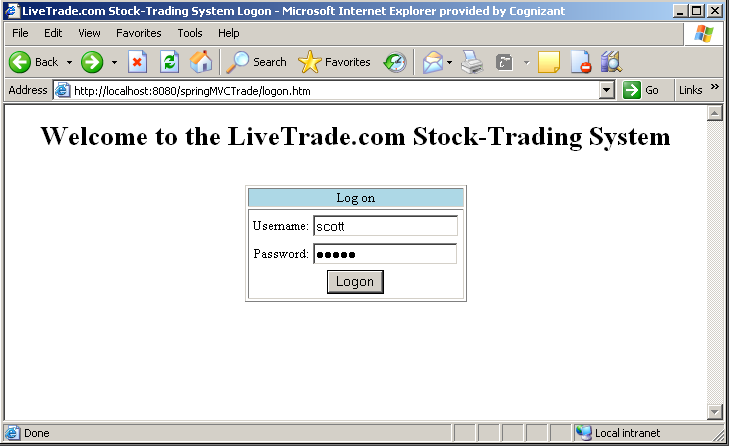
**Estimated Completion Time: 240 Minutes**

(100 Marks)

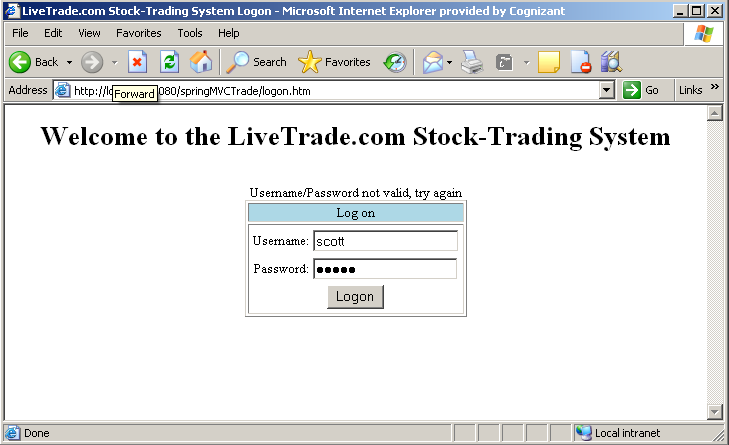
**Objective**: To create a web application to do online share trading.

**Concept**: LiveTrade.com wants to make its share trading process online. A web application needs to be created which allows the registered user to buy and sell shares.

The home page(logon page) should request the user to provide the user name and password.



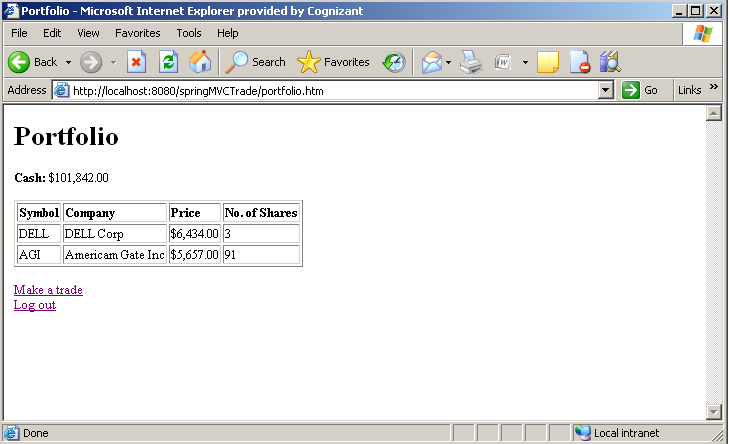
Appropriate error message should be displayed if username/password is invalid.



On successful logon a Portfolio page for the user should be displayed. The Portfolio page should contain the following information:

* Available cash for the user account(Portfolio account) .
* List of shares owned by the user.
* Share information should contain the following information
* Share symbol
* Company Name
* Price per share
* Total number of shares

The Portfolio page should also provide link to make a trade and logout.



When the user selects Make a trade he should be directed to trade page. The trade page should allow the user to sell his/her share or to buy a new share.

New share can be purchased only if the user has enough amount (cash) in his portfolio account.

User can sell only the share that is available in his portfolio account.

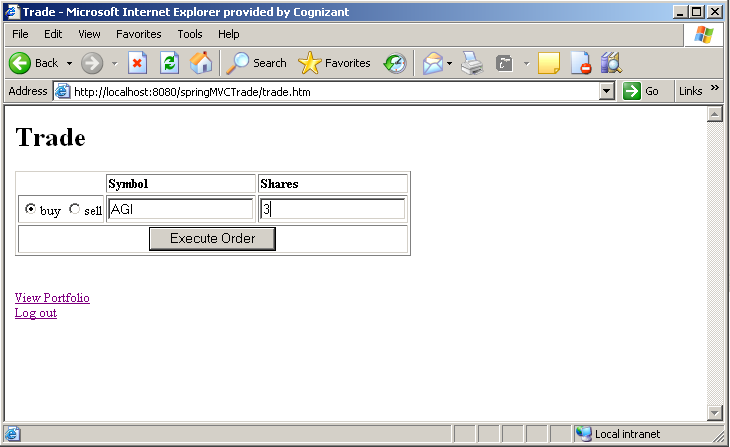
Whenever the user sell the share the user’s cash value should be updated in the portfolio account.

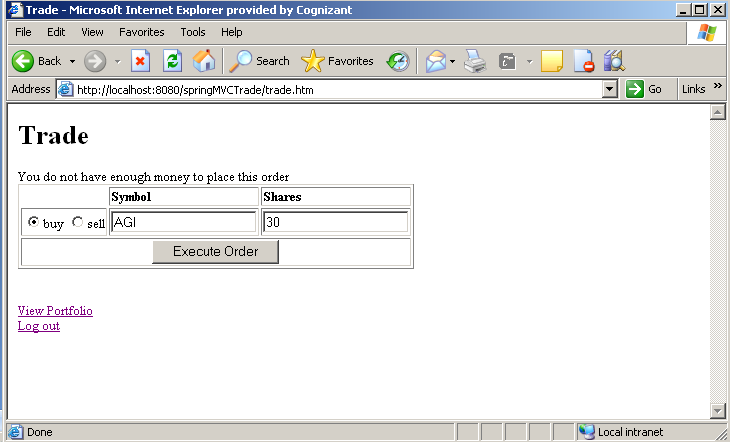
Only acceptable share symbols can be used to sell/buy shares.

The trade page should provide a button to Execute order.

The trade page should also contain links to see the Portfolio account details and logout.

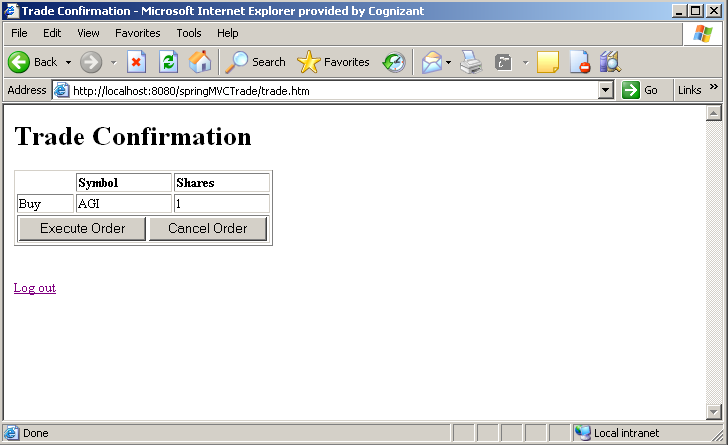
Appropriate error messages should be displayed whenever the user provides an invalid share symbol or if the cash is insufficient to buy shares.





When the user selects Execute Order he should be directed to Trade Confirmation page where he/she can continue with his/her order or cancel the order.

This page should also provide an option to logout.



When the user selects Cancel Order the user should be directed to Portfolio page to make a new order.

If the user selects Execute Order, an Acknowledgement page should be shown to the user listing his/her order.

This page should provide link to go to the Portfolio page and logout page.

At any point of time, when the user selects logout he/she should be directed to the logon page.

Evaluation Rubrics

|  |  |
| --- | --- |
| Parameters | Weightage |
| Completeness | 60 |
| Accuracy | 15 |
| Clarity of understanding | 15 |
| Presentation | 10 |
| Total | 100 |

**Summary of this Case Study:**

You have just learnt:

* How to write spring controller(Controller, SimpleFormController, AbstractWizardFormController)
* How to write JSP page using Spring tags.
* How to write form validators
* How to write DAO using JDBC/Hibernate
* How to use view resolvers
* How to use Resource Bundle.

