Manoj Shrestha's Blog (http://shrestha-manoj.blogspot.in/)

Sunday, May 25, 2014

Spring MVC + Maven + Hibernate CRUD Example Complete

This is a step-by-step tutorial for creating a simple web application starting from scratch using the following technology:

- **Spring MVC 4.0** as a web framework
- **Maven** as a build tool and for dependency management
- **Tomcat 7** as a web server
- **Hibernate** as an ORM tool

Let's take a look into what we are going to build.

Objective:

I will be explaining Spring MVC basic features by depicting a CRUD operations. Let's create an application called 'Book Store' where you can have the all the **CRUD** operations such as adding a book, editing a book, deleting a book and listing out all the books. This application can be implemented in projects like Bookstore Management, Library Management etc.

Snapshots:

The following snapshots will give you an idea how it looks like at the end.

Listing out all the book records:

Translate

Select Language ▾

Powered by Google Translate (https://translate.google.com)

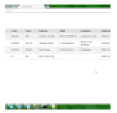
Total Pageviews

123689

Search This Blog

Search

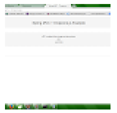
Most Popular Posts



(http://shrestha-manoj.blogspot.in/2014/05/spring-mvc-maven-hibernate-crud-example.html)

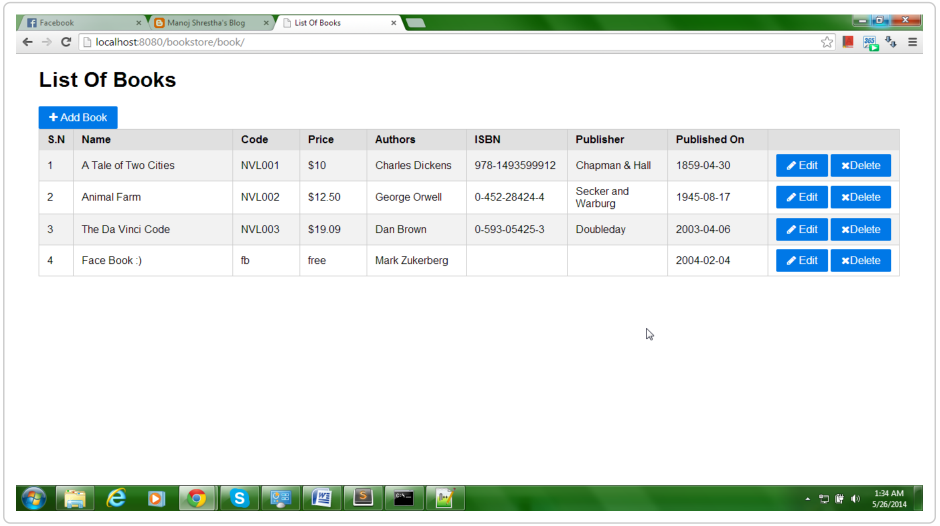
Spring MVC + Maven + Hibernate CRUD Example Complete (http://shrestha-manoj.blogspot.in/2014/05/spring-mvc-maven-hibernate-crud-example.html)

This is a step-by-step tutorial for creating a simple web application starting from scratch using the following technology: Spring MVC...



(http://shrestha-manoj.blogspot.in/2014/07/dropzonejs-spring-mvc-hibernate.html)

Dropzone JS + Spring MVC + Hibernate Integration (Complete Example) (http://shrestha-manoj.blogspot.in/2014/07/dropzonejs-spring-mvc-hibernate.html)



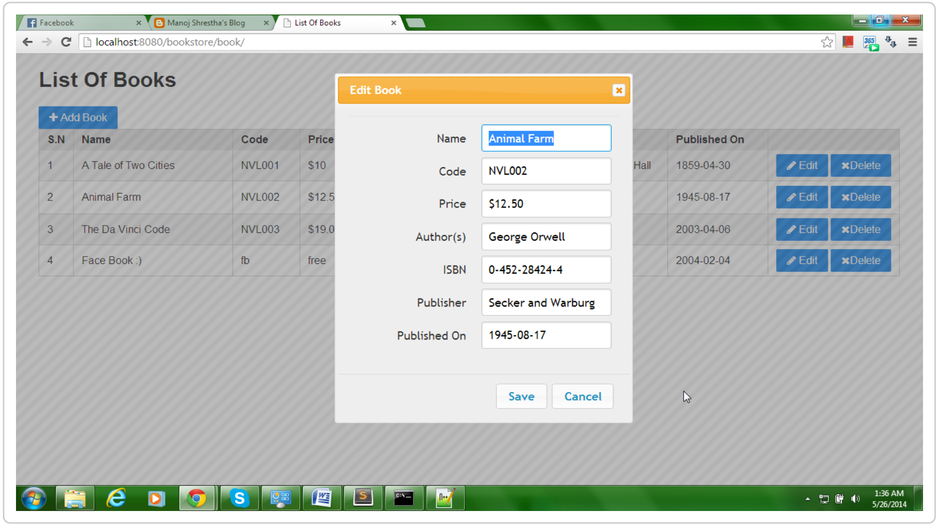
List Of Books

+ Add Book

S.N	Name	Code	Price	Authors	ISBN	Publisher	Published On	
1	A Tale of Two Cities	NVL001	$10	Charles Dickens	978-1493599912	Chapman & Hall	1859-04-30	Edit Delete
2	Animal Farm	NVL002	$12.50	George Orwell	0-452-28424-4	Secker and Warburg	1945-08-17	Edit Delete
3	The Da Vinci Code	NVL003	$19.09	Dan Brown	0-593-05425-3	Doubleday	2003-04-06	Edit Delete
4	Face Book :)	fb	free	Mark Zukerberg			2004-02-04	Edit Delete

(http://1.bp.blogspot.com/-7zLKy0LCYRs/U4JisPcy4iI/AAAAAAAADlQ/HJa91eWFY88/s1600/listBooks.png)

Adding/Editing a book:



Edit Book

Name	Animal Farm
Code	NVL002
Price	$12.50
Author(s)	George Orwell
ISBN	0-452-28424-4
Publisher	Secker and Warburg
Published On	1945-08-17

Save Cancel

(http://4.bp.blogspot.com/-7L5fEHxPREg/U4Ji4T9OATI/AAAAAAAADlY/ZxT0Fk9PZiw/s1600/Edit.png)

Project Setup

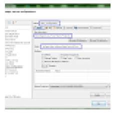
Let's begin setting up the project for eclipse. Maven provides several Archetype artifacts to define your project structure. I guess maven-archetype-webapp (http://maven.apache.org/archetype/maven-archetype-bundles/maven-archetype-webapp/) is the most common one for our purpose. When I was new, I used to google a lot to search

This tutorial will demonstrate how to integrate DropzoneJS with Spring MVC. I'll cover in detail starting from creating a simple Spr...

Can AngularJS have multiple ng-app directives in a single page? (http://shrestha-manoj.blogspot.in/2014/06/can-angularjs-have-multiple-ng-app.html)
The answer is NO . The ng-app directive is used to auto-bootstrap an AngularJS application. And according to AngularJS Documentation , on...

8 puzzle Solver using A* Algorithm (Java Code) (http://shrestha-manoj.blogspot.in/2011/06/8-puzzle-solver-java.html)
This project was done as a part of academic study in subject "Artificial Intelligence" . I have attached the java source code...

Binary Heap Data Structure - Java Code (http://shrestha-manoj.blogspot.in/2015/07/binary-heap-data-structure-java-code.html)
A nearly complete binary tree, where parent node has a priority over child nodes. Two types: Max heap The key of parent nodes is al...

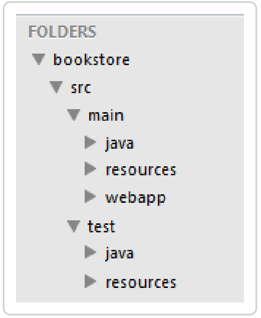

(http://shrestha-manoj.blogspot.in/2014/08/running-maven-project-inside-eclipse.html)
Run Maven Project inside Eclipse (http://shrestha-manoj.blogspot.in/2014/08/running-maven-project-inside-eclipse.html)
For running the project in eclipse, please follow the steps below (I'm assuming you've already imported the project in eclipse): ...

Convert English Date To Nepali Date in JAVA (complete working code) (http://shrestha-manoj.blogspot.in/2012/06/convert-english-date-to-nepali-date-in.html)
So far, if you need to work with English dates, you'll find a lot of date APIs, calender APIs. But unfortunately, we still don't ...

Bootstrap Login Template Example (Modal Dialog) Code (http://shrestha-manoj.blogspot.in/2014/06/bootstrap-login-template-example-code.html)
The following login template is one of my favourite and simple bootstrap login forms using modal dialog. Bootstrap Login Template ...

[SOLVED] Easiest Way to Fix the Battery Drain Issue after Rooting Samsung Galaxy S6 (http://shrestha-which Archetype to use for my every project I create. But these days, due to some personal reasons, I prefer starting from scratch. This way, you can get rid of unwanted files and folders.

First create a root project folder 'bookstore'. And then create inner folders in the following hierarchical structure:

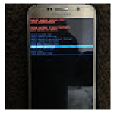


(http://4.bp.blogspot.com/-NKNwagxTgwo/U29cAqEVdlI/AAAAAAAADcM/12Nsn-yq2Mg/s1600/Folder+Structure.png)

You can omit the 'test' directory if you'd like cause we won't be using that folder in this example. I just added it to show a professional project structure. Now, we're almost done. All you need more is to add pom.xml. Add the following pom.xml under the root directory 'bookstore'.

pom.xml (https://github.com/frenmanoj/bookstore/blob/master/pom.xml) (path: /bookstore/pom.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>np.com.mshrestha</groupId>
    <artifactId>bookstore</artifactId>
    <name>bookstore</name>
    <packaging>war</packaging>
    <version>1.0.0-BUILD-SNAPSHOT</version>
    <properties>
        <java-version>1.7</java-version>
        <spring-version>4.0.3.RELEASE</spring-version>
        <hibernate-version>4.2.7.Final</hibernate-version>
        <mysql-version>5.1.27</mysql-version>
        <apache-connection-pooling-version>1.4</apache-connection-pooling-version>
    </properties>
    <dependencies>
        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring-version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring-version}</version>
        </dependency>
        <dependency>
```


(http://shrestha-manoj.blogspot.in/2016/01/solved-easiest-way-to-fix-battery-drain.html)
manoj.blogspot.in/2016/01/solved-easiest-way-to-fix-battery-drain.html)
I recently rooted my Samsung Galaxy S6 DUOS (SM-920F), a factory unlocked UK version. But after rooting, I faced the battery draining ...

Nepali Unicode | Nepali Unicode Converter (http://shrestha-manoj.blogspot.in/2014/05/nepali-unicode-nepali-unicode-converter.html)
converts Nepali Romanized texts to Nepali Unicode As of now, there are so many Nepali fonts such as Preeti, Kantipur, H...

Blog Archive

- ► 2016 (http://shrestha-manoj.blogspot.in/search?updated-min=2016-01-01T00:00:00-06:00&updated-max=2017-01-01T00:00:00-06:00&max-results=1) (1)
- ► 2015 (http://shrestha-manoj.blogspot.in/search?updated-min=2015-01-01T00:00:00-06:00&updated-max=2016-01-01T00:00:00-06:00&max-results=11) (11)
- ▼ 2014 (http://shrestha-manoj.blogspot.in/search?updated-min=2014-01-01T00:00:00-06:00&updated-max=2015-01-01T00:00:00-06:00&max-results=14) (14)
  - ► August (http://shrestha-manoj.blogspot.in/2014_08_01_archive.html) (1)
  - ► July (http://shrestha-manoj.blogspot.in/2014_07_01_archive.html) (1)
  - ► June (http://shrestha-manoj.blogspot.in/2014_06_01_archive.html) (5)
  - ▼ May (http://shrestha-manoj.blogspot.in/2014_05_01_archive.html) (3)
    - Spring MVC + Maven + Hibernate CRUD Example Comple... (http://shrestha-manoj.blogspot.in/2014/05/spring

```
        <groupId>org.springframework</groupId>
        <artifactId>spring-orm</artifactId>
        <version>${spring-version}</version>
    </dependency>

    <!-- Hibernate -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>${hibernate-version}</version>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-entitymanager</artifactId>
        <version>${hibernate-version}</version>
    </dependency>

    <!--
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-commons-annotations</artifactId>
        <version>3.3.0.ga</version>
    </dependency>
    -->

    <!-- Apache's Database Connection Pooling -->
    <dependency>
        <groupId>commons-dbcp</groupId>
        <artifactId>commons-dbcp</artifactId>
        <version>${apache-connection-pooling-version}</version>
    </dependency>
    <dependency>
        <groupId>commons-pool</groupId>
        <artifactId>commons-pool</artifactId>
        <version>${apache-connection-pooling-version}</version>
    </dependency>

    <!-- MySQL -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>${mysql-version}</version>
    </dependency>

    <!-- Servlet -->
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>servlet-api</artifactId>
        <version>2.5</version>
        <scope>provided</scope>
    </dependency>
```

-mvc-maven-hibernate-crud-example.html)

Nepali Unicode | Nepali Unicode Converter (http://shrestha-manoj.blogspot.in/2014/05/nepali-unicode-nepali-unicode-converter.html)

javax.persistence vs. org.hibernate.annoatations (http://shrestha-manoj.blogspot.in/2014/05/javaxpersistence-vs-orghibernateannoata.html)

► April (http://shrestha-manoj.blogspot.in/2014_04_01_archive.html) (4)

► 2012 (http://shrestha-manoj.blogspot.in/search?updated-min=2012-01-01T00:00:00-06:00&updated-max=2013-01-01T00:00:00-06:00&max-results=3) (3)

► 2011 (http://shrestha-manoj.blogspot.in/search?updated-min=2011-01-01T00:00:00-06:00&updated-max=2012-01-01T00:00:00-06:00&max-results=6) (6)

G+1 4

```
            <dependency>
                  <groupId>javax.servlet.jsp</groupId>
                  <artifactId>jsp-api</artifactId>
                  <version>2.1</version>
                  <scope>provided</scope>
            </dependency>
            <dependency>
                  <groupId>javax.servlet</groupId>
                  <artifactId>jstl</artifactId>
                  <version>1.2</version>
            </dependency>

      </dependencies>
      <build>
        <plugins>

            <!-- Tomcat 7 plugin -->
            <plugin>
                 <groupId>org.apache.tomcat.maven</groupId>
                 <artifactId>tomcat7-maven-plugin</artifactId>
                 <version>2.0</version>
            </plugin>

            <plugin>
                <artifactId>maven-eclipse-plugin</artifactId>
                <version>2.9</version>
                <configuration>
                     <additionalProjectnatures>
                         <projectnature>
                             org.springframework.ide.eclipse.core.springnature
                         </projectnature>
                     </additionalProjectnatures>
                     <additionalBuildcommands>
                         <buildcommand>
                             org.springframework.ide.eclipse.core.springbuilder
                         </buildcommand>
                     </additionalBuildcommands>
                     <downloadSources>true</downloadSources>
                     <downloadJavadocs>true</downloadJavadocs>
                </configuration>
             </plugin>
          </plugins>
      </build>
      <repositories>
            <repository>
                  <id>java.net</id>
                  <url>http://download.java.net/maven/2/</url>
            </repository>
      </repositories>
</project>
```

Now, open the command prompt, go to the root directory and then type mvn eclipse:eclipse. This will generate the Eclipse configuration files. For instance, you'll see the files .classpath, .project created under the root directory.

Note that, the following segment adds the Tomcat 7 plugin to the project.

```
<build>
   <plugins>
       <plugin>
           <groupId>org.apache.tomcat.maven</groupId>
           <artifactId>tomcat7-maven-plugin</artifactId>
           <version>2.0</version>
           <!-- version 2.2 if you're using JDK 1.8 -->
        </plugin>
   </plugins>
</build>
```

Now, let's import the project from eclipse. You can import the project either as 'General -> Existing Projects into Workspace' or as 'Maven -> Existing Maven Projects'.

Time to create some java source packages. Let's create the following packages under the 'src/java/main' directory. I believe the package names are self-descriptive.

- np.com.mshrestha.bookstore.dao
- np.com.mshrestha.bookstore.dao.impl
- np.com.mshrestha.bookstore.service
- np.com.mshrestha.bookstore.service.impl
- np.com.mshrestha.bookstore.controller

Then, let's create another folder 'WEB-INF' under webapp directory and add the deployment descriptor web.xml under WEB-INF. The web.xml defines root spring container and the heart of the spring MVC - 'DispatcherServlet'.

web.xml (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/WEB-INF/web.xml) (path: /webapp/WEB-INF/web.xml)

```
<?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.xsd">

    <!-- The definition of the Root Spring Container shared by all Servlets
        and Filters -->

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>

    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
        <listener-class>
                                org.springframework.web.context.ContextLoaderListener
(http://docs.spring.io/spring/docs/3.0.x/api/org/springframework/web/context/ContextLoaderListener.html)
        </listener-class>
    </listener>

    <!-- Processes application requests -->
    <servlet>
        <servlet-name>dispatcherServlet</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>
                /WEB-INF/spring/dispatcherServlet/servlet-context.xml
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>dispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

In the above web.xml, we see the two configuration xmls. I intentionally created those two config xmls to let you know that we can initialize parameters in two ways. This will be needed if you have multiple servlets and filters that shares common parameters. You can define only one servlet config xml and put all of contents from both xmls into the one. The tag <context-param> defines parameters that are shared by all servlets and filters whereas <init-param> defines parameters that are accessible by only that <servlet> inside which it is placed. We also defined the DispatcherServlet which is the default request handler for all the requests as defined by the pattern "/" in <url-pattern>.

<web-app>:

- Defines the root element of the document called Deployment Descriptor

<context-param>:

- This is an optional element
- Contains the declaration of a Web application's servlet context initialization parameters available to the entire scope of the web application
- The methods for accessing context init parameter is as follows:
    - String paramName = getServletContext().getInitParamter("paramName")

<init-param>:

- An optional element within the servlet
- Defines servlet configuration initialization parameters only available to the servlet inside which it's defined
- The methods for accessing servlet init parameter is as follows:
    - String paramName = getServletConfig().getInitParamter("paramName")

<listener>:

- Defines a listener class to start up and shut down Spring's root application context
- You might not need this if you don't define the element <context-param>

<servlet>:

- Declares a servlet including a refering name, defining class and initialization parameters
- You can define multiple servlets in the document
- Interesting thing is you can even declare multiple servlets using the same class with different initialization parameters
- The name of each servlet must be unique

<servlet-mapping>:

- Maps a URL pattern to a servlet
- In the above case, all requests will be handled by the DispatcherServlet named dispatcherServlet

Now, let's create them in the respective locations too. You can put both xmls alongside web.xml under WEB-INF. But I put them one under WEB-INF/spring directory and another under WEB-INF/spring/dispatcherServlet. Let's look at those xmls.

servlet-context.xml (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/WEB-INF/spring/dispatcherServlet/servlet-context.xml) (path: /webapp/WEB-INF/spring/dispatcherServlet/servlet-context.xml)

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

    <!-- DispatcherServlet Context: defines this servlet's request-processing
        infrastructure -->

    <!-- Enables the Spring MVC @Controller programming model -->
    <annotation-driven />
    <context:component-scan base-package="np.com.mshrestha.bookstore.controller" />

    <!-- Handles HTTP GET requests for /web-resources/** by efficiently serving
        up static resources in the ${webappRoot}/resources/ directory -->
    <resources mapping="/web-resources/**" location="/resources/" />

    <!-- Resolves views selected for rendering by @Controllers to .jsp resources
        in the /WEB-INF/views directory -->
    <beans:bean
        class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <beans:property name="prefix" value="/WEB-INF/views/" />
        <beans:property name="suffix" value=".jsp" />
    </beans:bean>

</beans:beans>
```

The various components from the above xmls are described as follows:

<annotation-driven />:

- Initialiazes components required for dispatching the requests to your Controllers
- Registers a RequestMappingHandlerMapping, a RequestMappingHandlerAdapter, and an ExceptionHandlerExceptionResolver (among others) in support of processing requests with annotated controller methods using annotations such as @RequestMapping, @ExceptionHandler, and others
- Configures support for new Spring MVC features such as declarative validation with @Valid, HTTP message conversion with @RequestBody/@ResponseBody, formatting Number fields with @NumberFormat etc.
    - For more details:
        - Spring Docs (http://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-config-enable)
        - Spring Blog (http://spring.io/blog/2009/12/21/mvc-simplifications-in-spring-3-0/)
- This element should be placed in your DispatcherServlet context (or in your root context if you have no DispatcherServlet context defined)

<context:component-scan>:

- Scans for the classes annotated with @Component, @Service, @Repository and @Controller defined under the base-package and registers their corresponding beans
- @Component serves as a generic stereotype for any Spring-managed component; whereas, @Repository, @Service, and @Controller serve as specializations of @Component for more specific use cases (e.g., in the persistence, service, and presentation layers, respectively)

<resources>: (http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-static-resources)

- This tag allows static resource requests following a particular URL pattern to be served by a ResourceHttpRequestHandler from any of a list of Resource location(s)
- This provides a convenient way to serve static resources from locations other than the web application root, including locations on the classpath
- e.g. <mvc:resources mapping="/resources/**" location="/, classpath:/web-resources/"/>

InternalResourceViewResolver:

- Spring provides different view resolvers to render models in a browser without tying you with a specific view technology.
- Out of the box, Spring enables you to use JSPs, Velocity templates and XSLT views.
- InternalResourceViewResolver is one of them.
- In the above example, when returning *test* as a viewname, this view resolver will hand the request over to the RequestDispatcher that will send the request to */WEB-INF/views/test.jsp*.
- For more details:
    - Spring Doc: View Resolvers (http://docs.spring.io/spring/docs/3.0.x/spring-framework-reference/html/mvc.html#mvc-viewresolver)

root-context.xml (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/WEB-INF/spring/root-context.xml) (path: /webapp/WEB-INF/spring/root-context.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:p="http://www.springframework.org/schema/p"
       xmlns:tx="http://www.springframework.org/schema/tx"
      xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx.xsd">

       <!-- Root Context: defines shared resources visible to all other web components -->

       <bean id="propertyConfigurer"
            class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

           <property name="locations">
                <list>
                <value>classpath:database.properties</value>
            </list>
        </property>
    </bean>

    <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close"
        p:driverClassName="${jdbc.driverClassName}"
        p:url="${jdbc.url}"
        p:username="${jdbc.username}"
        p:password="${jdbc.password}" />

    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />

        <property name="packagesToScan">
            <list>
                <value>np.com.mshrestha.bookstore.model</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.hbm2ddl.auto">update</prop>
                <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                <prop key="hibernate.show_sql">false</prop>
            </props>
        </property>
    </bean>

    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <tx:annotation-driven transaction-manager="transactionManager" />
    <context:component-scan base-package="np.com.mshrestha.bookstore" />
</beans>
```

The database property file looks like as follows:

database.properties (https://github.com/frenmanoj/bookstore/blob/master/src/main/resources/database.properties) (path: /src/main/resources/database.properties)

```
jdbc.driverClassName=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost/bookstore?createDatabaseIfNotExist=true&amp;amp;useUnicode=true&amp;amp;characterEncoding=utf-8&amp;amp;autoReconnect=true

jdbc.username=root
jdbc.password=root

hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

Finally, we're done setting up the project. Let's deal with rest of the parts.

Persistence

Let's create a persistent annotated POJO class 'Book' under the respective model package 'np.com.mshrestha.bookstore.model'.

```
package np.com.mshrestha.bookstore.model;

import java.sql.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "book")
public class Book {

    private Long id;
    private String name;
    private String code;
    private String price;
    private String authors;
    private String isbn;
    private String publisher;
    private Date publishedOn;

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    public Long getId() {
        return id;
    }

    @Column(nullable = false)
    public String getName() {
        return name;
    }

    @Column(length = 15, nullable = false)
    public String getCode() {
        return code;
    }

    @Column(length = 10)
    public String getPrice() {
        return price;
    }

    @Column(nullable = false)
    public String getAuthors() {
        return authors;
    }

    @Column
    public String getIsbn() {
        return isbn;
    }

    @Column
    public String getPublisher() {
        return publisher;
    }

    @Column(name = "published_date")
    public Date getPublishedOn() {
        return publishedOn;
    }

    // setter methods...
}
```

In the above example, I've shown different attributes we can use for @Column annotation such as *name*, *length*, *nullable*.

Couple points to notice in the above POJO. If you look into the import statements, all the annotations are imported from the package javax.persistence rather than from the package org.hibernate.annotations. The package javax.persistence is provided by **J**ava **P**ersistence **A**PI (JPA) whereas org.hibernate.annotations is provided by Hibernate. For the reason behind this, please go to:

- javax.persistence vs. org.hibernate.annoatations (http://shrestha-manoj.blogspot.com/2014/05/javaxpersistence-vs-orghibernateannoata.html)

Another thing is, the annotations are applied on the getter methods rather than on the fields. You can put annotations either all on the fields or all on the getter methods. It's just a matter of preference which approach to follow. Some people prefer annotations on the fields while others prefer annotations on the getter methods. But don't mix up both approaches unless you know well to use the @Access annotation.You can go to the following links to see the discussion regarding which one is better:

- javax.persistence Annotations on field, getter or setter? (http://stackoverflow.com/questions/8965116/javax-persistence-annotations-on-field-getter-or-setter)
- Performance difference between annotating fields or getter methods in Hibernate / JPA (http://stackoverflow.com/questions/332591/performance-difference-between-annotating-fields-or-getter-methods-in-hibernate)
- Hibernate Annotations - Which is better, field or property access? (http://stackoverflow.com/questions/594597/hibernate-annotations-which-is-better-field-or-property-access)

DAO Layer

Now, time to work on the **D**ata **A**ccess **O**bject (DAO) layer to communicate with database. Let's create an interface BookDao under package np.com.mshrestha.bookstore.dao and it's implementation class BookDaoImpl under package np.com.mshrestha.bookstore.dao.impl.

BookDao.java (https://github.com/frenmanoj/bookstore/blob/master/src/main/java/np/com/mshrestha/bookstore/dao/BookDao.java)

```
package np.com.mshrestha.bookstore.dao;

import java.util.List;
import np.com.mshrestha.bookstore.model.Book;

public interface BookDao {

    /*
     * CREATE and UPDATE
     */
    public void saveBook(Book book); // create and update

    /*
     * READ
     */
    public List<Book> listBooks();
    public Book getBook(Long id);

    /*
     * DELETE
     */
    public void deleteBook(Long id);
}
```

BookDaoImpl.java (https://github.com/frenmanoj/bookstore/blob/master/src/main/java/np/com/mshrestha/bookstore/dao/impl/BookDaoImpl.java)

```
package np.com.mshrestha.bookstore.dao.impl;

import java.util.List;

import np.com.mshrestha.bookstore.dao.BookDao;
import np.com.mshrestha.bookstore.model.Book;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

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

@Repository
public class BookDaoImpl implements BookDao {

    @Autowired
    private SessionFactory sessionFactory;
    public void saveBook(Book book) {

        getSession().merge(book);
    }

    public List<Book> listBooks() {

        return getSession().createCriteria(Book.class).list();
    }

    public Book getBook(Long id) {
        return (Book) getSession().get(Book.class, id);
    }

    public void deleteBook(Long id) {

        Book book = getBook(id);

        if (null != book) {
            getSession().delete(book);
        }
    }

    private Session getSession() {
        Session sess = getSessionFactory().getCurrentSession();
        if (sess == null) {
            sess = getSessionFactory().openSession();
        }
        return sess;
    }

    private SessionFactory getSessionFactory() {
        return sessionFactory;
    }
}
```

The method saveBook() calls a hibernate method Sesson.merge (http://docs.jboss.org/hibernate/orm/3.5/javadocs/org/hibernate/Session.html#merge(java.lang.Object))(). We're going to call the same method for both saving a new book as well as updating an existing book. Sesson.merge() takes a good care of that. It copies the state of the passed object onto the persistent object with the same identifier. If there is no persistence instance currently associated with the session, it will be loaded. And returns the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The passed object will not be attached to the session.

Other methods implementation seems to be straight-forward.

The class BookDaoImpl contains two Spring annotations @Repository and @Autowired.

- @Repository (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/stereotype/Repository.html):
    - Indicates that the annotated class (BookDaoImpl in this case) is a Repository or DAO
    - A stereotype (a specialization of @Component) for persistence layer

- @Autowired (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html)
    - Marks the field (or it could be a constructor or a setter method) as to be autowired by Spring's dependency injection facilities
    - The field is injected right after construction of the bean, before any config methods are invoked
    - Autowires the bean by matching the data type in the configuration metadata

Now, we're done with DAO layer. Let's move ahead to Service layer.

Service Layer

The Service Layer is a bridge between the DAO (Persistence) layer and the Presentation (Web) layer. Just like we did in DAO layer, let's create an interface BookService under the package np.com.mshrestha.bookstore.service and it's implementation class BookServiceImpl under the package np.com.mshrestha.bookstore.service.impl.

BookService.java (https://github.com/frenmanoj/bookstore/blob/master/src/main/java/np/com/mshrestha/bookstore/service/BookService.java)

```
package np.com.mshrestha.bookstore.service;

import java.util.List;

import np.com.mshrestha.bookstore.model.Book;

public interface BookService {

    /*
     * CREATE and UPDATE
     */
    public void saveBook(Book book);

    /*
     * READ
     */
    public List<Book> listBooks();
    public Book getBook(Long id);

    /*
     * DELETE
     */
    public void deleteBook(Long id);

}
```

BookServiceImpl.java (https://github.com/frenmanoj/bookstore/blob/master/src/main/java/np/com/mshrestha/bookstore/service/impl/BookServiceImpl.java)

```
package np.com.mshrestha.bookstore.service.impl;

import java.util.List;

import np.com.mshrestha.bookstore.dao.BookDao;
import np.com.mshrestha.bookstore.model.Book;
import np.com.mshrestha.bookstore.service.BookService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class BookServiceImpl implements BookService {

    @Autowired
    private BookDao bookDao;

    @Transactional
    public void saveBook(Book book) {
        bookDao.saveBook(book);
    }

    @Transactional( readOnly = true)
    public List<Book> listBooks() {
        return bookDao.listBooks();
    }

    @Transactional( readOnly = true)
    public Book getBook(Long id) {
        return bookDao.getBook(id);
    }

    @Transactional
    public void deleteBook(Long id) {
        bookDao.deleteBook(id);

    }
}
```

In the above class, we can see three Spring annotations: @Service @Autowired @Transactional

- @Service (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/stereotype/Service.html):
  - Indicates that the annotated class (BookServiceImpl in this case) is a "Service"
  - A stereotype (a specialiation of @Component) for service layer
- @Autowired (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html)
  - Marks the field (or it could be a constructor or a setter method) as to be autowired by Spring's dependency injection facilities

- The field is injected right after construction of the bean, before any config methods are invoked
- Autowires the bean by matching the data type in the configuration metadata

- @Transactional (http://docs.spring.io/spring/docs/3.1.x/javadoc-api/org/springframework/transaction/annotation/Transactional.html)
    - Enables Spring's transactional behaviour
    - Can be applied to an interface, a class or a method
    - This annotation is enabled by putting <tx:annotation-driven/> in the context configuration file.
    - The attribute readOnly = **true** sets the transaction to read only mode so that by any chance it cannot modify data.

Presentation (Web) Layer

Controller

Let's create a controller that handles the web requests dispatched via. DispatcherServlet. Let the name of the controller be BookController and put it under the package np.com.mshrestha.bookstore.controller.

(https://www.blogger.com/goog_1403327270) **BookController.java (https://github.com/frenmanoj/bookstore/blob/master/src/main/java/np/com/mshrestha/bookstore/controller/BookController.java)**

```
package np.com.mshrestha.bookstore.controller;

import java.util.Map;

import np.com.mshrestha.bookstore.model.Book;
import np.com.mshrestha.bookstore.service.BookService;

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

@Controller
@RequestMapping("/book")
public class BookController {

    @Autowired
    private BookService bookService;

    @RequestMapping(value = { "/", "/listBooks" })
    public String listBooks(Map<String, Object> map) {

        map.put("book", new Book());
        map.put("bookList", bookService.listBooks());

        return "/book/listBooks";
    }

    @RequestMapping("/get/{bookId}")
    public String getBook(@PathVariable Long bookId, Map<String, Object> map) {

        Book book = bookService.getBook(bookId);

        map.put("book", book);

        return "/book/bookForm";
    }

    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String saveBook(@ModelAttribute("book") Book book,
            BindingResult result) {

        bookService.saveBook(book);

        /*
         * Note that there is no slash "/" right after "redirect:"
         * So, it redirects to the path relative to the current path
         */
        return "redirect:listBooks";
    }

    @RequestMapping("/delete/{bookId}")
    public String deleteBook(@PathVariable("bookId") Long id) {

        bookService.deleteBook(id);

        /*
         * redirects to the path relative to the current path
         */
        // return "redirect:../listBooks";

        /*
         * Note that there is the slash "/" right after "redirect:"
         * So, it redirects to the path relative to the project root path
         */
        return "redirect:/book/listBooks";
    }
}
```

Let's go through the Spring annotations in the above controller:

- @Controller (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/stereotype/Controller.html)
    - Indicates that the annotated class (BookController) serves the role of a "Controller"

- A stereotype (a specialiation of @Component) for web layer
- To enable autodetection of such annotated controllers, you add component scanning to your configuration (see: servlet-context.xml)
- <context:component-scan base-package=*"np.com.mshrestha.bookstore.controller"* />

- @Autowired (http://docs.spring.io/spring/docs/3.2.6.RELEASE/javadoc-api/org/springframework/beans/factory/annotation/Autowired.html)
    - Marks the field (or it could be a constructor or a setter method) as to be autowired by Spring's dependency injection facilities
    - The field is injected right after construction of the bean, before any config methods are invoked
    - Autowires the bean by matching the data type in the configuration metadata
- @RequestMapping (http://docs.spring.io/spring/docs/3.2.8.RELEASE/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html)
    - DispatcherServlet uses this annotation to dispatch the requests to the correct controller and the handler method.
    - Maps URLS or web requests onto specific handler classes (e.g. in the above controller class @RequestMapping("/book") and/or handler methods (e.g. in the above example @RequestMapping("/get/{bookId}")
    - RequestMapping element value is a String array (String[]) so it can accept multiple URL paths
    - e.g. @RequestMapping(value = { "/", "/listBooks" })
    - The handler methods which are annotated with this annotation are allowed to have very flexible return types and method signatures. Please go through the spring documentation (http://docs.spring.io/spring/docs/3.2.8.RELEASE/javadoc-api/org/springframework/web/bind/annotation/RequestMapping.html)for that.

I want to go little further on the following annotations.

@PathVariable (http://docs.spring.io/spring/docs/3.2.8.RELEASE/javadoc-api/org/springframework/web/bind/annotation/PathVariable.html):

It binds the method parameter to a URI template variable. URI template is a parameterized URI i.e. URI having one or more variables. For instance, the URI template *http://localhost:8080/bookstore/book/delete/{bookId}* contains the variable bookId. Assigning the value 10 to the variable yields *http://localhost:8080/bookstore/book/delete/10*.

```
@RequestMapping("/delete/{bookId}")
public String deleteBook(@PathVariable("bookId") Long id) {

    bookService.deleteBook(id);

    return "redirect:/book/listBooks";
}
```

The URI Template "/delete/{bookId}" specifies the variable name bookId. When the controller handles this request, the value of bookId is set to the value found in the appropriate part of the URI. For example, when a request comes in for /delete/10, the value of bookId is 10.

If the URI template variable name matches the method argument name, we can omit the value element of PathVariable. As long as your code is not compiled without debugging information, Spring MVC will match the method argument name to the URI template variable name. This scenario has been depicted in the method BookController.getBook(). Or you could write the above example as follows:

```
@RequestMapping("/delete/{bookId}")
public String deleteBook(@PathVariable Long bookId) {

    bookService.deleteBook(bookId);

    return "redirect:/book/listBooks";
}
```

@ModelAttribute (http://docs.spring.io/spring/docs/3.2.8.RELEASE/javadoc-api/org/springframework/web/bind/annotation/ModelAttribute.html)

In a nutshell, this annotation is used for preparing the model data and also to define the command object that would be bound with the HTTP request data. This annotation can be applied on the methods as well as on the method arguments. In the above controller example, we used it on the argument of the saveBook() method. So, I'll be explaing its usage only on the method arguments.

An @ModelAttribute on a method argument indicates the argument should be retrieved from the model. If not present in the model, the argument should be instantiated first and then added to the model. Once present in the model, the argument's fields should be populated from all request parameters that have matching names. This is known as data binding in Spring MVC, a very useful mechanism that saves you from having to parse each form field individually.

```
@RequestMapping(value = "/save", method = RequestMethod.POST)
public String saveBook(@ModelAttribute("book") Book book,
        BindingResult result) {

    bookService.saveBook(book);

    /*
     * redirects to the path w.r.t the current path
     */
    return "redirect:listBooks";
}
```

In the above example, the Book instance can come from several options as follows:

- It may already be in the model due to use of @SessionAttributes storing model attributes in the HTTP session between requests
- It may already be in the model due to an @ModelAttribute method in the same controller
- It may be retrieved based on a URI template variable in conjunction with Type Converter
- It may be instantiated using its default constructor

I won't go through each of the above points which is beyong the scope of this tutorial. If you would like to go through all of the above points, here's the Spring documentation for that.

- Using @ModelAttribute on a method argument (http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/mvc.html#mvc-ann-modelattrib-method-args)

Finally, we're done with the Controller class too. Now, let's move ahead to Views i.e. JSPs.

JSPs

Time to create some JSPs. We can have separate JSP pages for creating, editing and listing out the book records. However, in this tutorial, I'm gonna show you a little different way where adding and editing is done in a JQuery Dialog (http://jqueryui.com/dialog/). We'll be creating two JSP pages:

- bookForm.jsp
  - contains a form common for both Add and Edit actions.
- listBooks.jsp
  - lists out all the book records
  - deletes specific record from the list
  - includes bookForm.jsp for adding and editing

bookForm (https://www.blogger.com/goog_1403327275).jsp (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/WEB-INF/views/book/bookForm.jsp) (path: /webapp/WEB-INF/views/book/bookForm.jsp)

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<c:url var="actionUrl" value="save" />

<form:form id="bookForm" commandName="book" method="post"
       action="${actionUrl }" class="pure-form pure-form-aligned">

    <fieldset>
        <legend></legend>

        <div class="pure-control-group">
            <label for="name">Name</label>
            <form:input path="name" placeholder="Book Name" />
        </div>
        <div class="pure-control-group">
             <label for="code">Code</label>
             <form:input path="code" placeholder="Code" maxlength="15" />
        </div>
        <div class="pure-control-group">
            <label for="price">Price</label>
            <form:input path="price" placeholder="Price" maxlength="10" />
        </div>
        <div class="pure-control-group">
            <label for="authors">Author(s)</label>
            <form:input path="authors" placeholder="Authors" />
        </div>
        <div class="pure-control-group">
            <label for="isbn">ISBN</label>
            <form:input path="isbn" placeholder="ISBN" />
        </div>
        <div class="pure-control-group">
            <label for="publisher">Publisher</label>
            <form:input path="publisher" placeholder="Publisher" />
        </div>
        <div class="pure-control-group">
            <label for="publishedOn">Published On</label>
            <form:input path="publishedOn" placeholder="YYYY-MM-DD" class="datepicker" />
        </div>

        <form:input path="id" type="hidden" />
     </fieldset>
</form:form>
```

The various components of the above jsp are explained below:

- Added `taglib` directive to use spring's Form (http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/view.html#view-jsp-formtaglib-formtag) tag and JSTL (http://docs.oracle.com/cd/E17802_01/products/products/jsp/jstl/1.1/docs/tlddocs/c/tld-summary.html) core tag.
- The attribute `commandName="book"` puts the book object in the PageContext so that is it can be accessed by inner tags like `<form:input>`.

- You might have seen another form attribute modelAttribute. There is no difference between them, two different attributes exist for some historical reasons.
- Remember that it is the book object that we put as a model in the controller. *Please refer the methods BookController.listBooks() and BookController.getBook().*
- When a page is requested, the controller puts the command object, book in this case, in the PageContext. Then Spring binds the object properties with the form elements using path attribute.
- By default Spring sets the id and name of the input field from the path attribute value. Having said that you can explicitly set the id. Name attribute can not be overridden though.
- A command class can basically be any Java class; the only requirement is a no-arg constructor. The command class should preferably be a JavaBean in order to be able to populate bean properties with request parameters.
- <form:input path="id" type="hidden" />
    - The hidden field holds the id of a book object. Required for editing a book record.
- The classes *pure-form*, *pure-form-aligned* and *pure-control-group* have nothing to do with Spring. They used merely used for styling purpose. To be more specifix, they came from purecss.io (http://purecss.io/).

listBooks.jsp (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/WEB-INF/views/book/listBooks.jsp) (path: /webapp/WEB-INF/views/book/listBooks.jsp)

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

<html>
<head>
<title>List Of Books</title>

<link rel="stylesheet" href='<c:url value="/web-resources/css/pure-0.4.2.css"/>'>

<link rel="stylesheet"
      href='<c:url value="/web-resources/css/font-awesome-4.0.3/css/font-awesome.css"/>'>
<link rel="stylesheet"
      href='<c:url value="/web-resources/css/jquery-ui-1.10.4.custom.css"/>'>
<style type="text/css">
  th {
      text-align: left
  }

</style>
</head>

<body>
   <div style="width: 95%; margin: 0 auto;">

        <div id="bookDialog" style="display: none;">
            <%@ include file="bookForm.jsp"%>
        </div>

        <h1>List Of Books</h1>

        <button class="pure-button pure-button-primary" onclick="addBook()">
```

```
            <i class="fa fa-plus"></i> Add Book
        </button>
        <br>
        <table class="pure-table pure-table-bordered pure-table-striped">
            <thead>
                <tr>
                    <th width="4%">S.N</th>
                    <th width="12%">Name</th>
                    <th width="12%">Code</th>
                    <th width="12%">Price</th>
                    <th width="12%">Authors</th>
                    <th width="12%">ISBN</th>
                    <th width="12%">Publisher</th>
                    <th width="12%">Published On</th>
                    <th width="12%"></th>
                </tr>
            </thead>
            <tbody>
                <c:forEach items="${bookList}" var="book" varStatus="loopCounter">
                <tr>
                    <td><c:out value="${loopCounter.count}" /></td>
                    <td><c:out value="${book.name}" /></td>
                    <td><c:out value="${book.code}" /></td>
                    <td><c:out value="${book.price}" /></td>
                    <td><c:out value="${book.authors}" /></td>
                    <td><c:out value="${book.isbn}" /></td>
                    <td><c:out value="${book.publisher}" /></td>
                    <td><c:out value="${book.publishedOn}" /></td>

                    <td>
                      <nobr>
                        <button onclick="editBook(${book.id});"
                                class="pure-button pure-button-primary">
                            <i class="fa fa-pencil"></i> Edit
                        </button>

                        <a href="delete/${book.id}" class="pure-button pure-button-primary"
                    onclick="return confirm('Are you sure you want to delete this book?');">
                            <i class="fa fa-times"></i>Delete
                        </a>
                      </nobr>
                    </td>
                </tr>
                </c:forEach>
            </tbody>
        </table>
    </div>

    <!--  It is advised to put the <script> tags at the end of the document body so that they don't block rendering of the page -->

    <script type="text/javascript"
```

```
        src='<c:url value="/web-resources/js/lib/jquery-1.10.2.js"/>'></script>
    <script type="text/javascript"
        src='<c:url value="/web-resources/js/lib/jquery-ui-1.10.4.custom.js"/>'></script>
    <script type="text/javascript"
        src='<c:url value="/web-resources/js/lib/jquery.ui.datepicker.js"/>'></script>
    <script type="text/javascript"
        src='<c:url value="/web-resources/js/js-for-listBooks.js"/>'></script>
</body>
</html>
```

Also, let's define the following JS file that contains the methods we defined in the above JSP page.

js-for-listBooks.js (https://github.com/frenmanoj/bookstore/blob/master/src/main/webapp/resources/js/js-for-listBooks.js) (path: /webapp/resources/js/js-for-listBooks.js)

```
function addBook() {
    $('#bookDialog').dialog("option", "title", 'Add Book');
    $('#bookDialog').dialog('open');
}

function editBook(id) {

    $.get("get/" + id, function(result) {

        $("#bookDialog").html(result);
        $('#bookDialog').dialog("option", "title", 'Edit Book');
        $("#bookDialog").dialog('open');

        initializeDatePicker();
    });
}

function initializeDatePicker() {
    $(".datepicker").datepicker({
        dateFormat : "yy-mm-dd",
        changeMonth : true,
        changeYear : true,
        showButtonPanel : true
    });
}

function resetDialog(form) {

    form.find("input").val("");
}

$(document).ready(function() {

    $('#bookDialog').dialog({

        autoOpen : false,
        position : 'center',
        modal : true,
        resizable : false,
        width : 440,
        buttons : {
            "Save" : function() {
                $('#bookForm').submit();
            },
            "Cancel" : function() {
                $(this).dialog('close');
            }
        },
        close : function() {

            resetDialog($('#bookForm'));
            $(this).dialog('close');
        }
    });

    initializeDatePicker();
});
```

The above listBooks.jsp basically lists out all the book records. When this page is requested, the request is handled by BookController.listBooks(). If you look into the method, it fetches all the book records and puts them in a map with the model name bookList which will be exposed to the web view.

```
map.put("bookList", bookService.listBooks());
```

We used JSTL core tag to iterate over the map model and listed them in a table.

```
<c:forEach items="${bookList}" var="book" varStatus="loopCounter">
```

Also, you must have noticed that we also created a hidden div that includes *bookForm.jsp*.

```
<div id="bookDialog" style="display: none;">
    <%@ include file="bookForm.jsp"%>
</div>
```

The above hidden div has been initialized as a JQuery Dialog in document ready method in **js-for-listBooks.js**. The dialog has been provided with Save button and Cancel button. The Save button submits the form (id=*"bookForm"*) and the Cancel button simply closes the form and resets all the input field values.

Add Book: When the page **listBooks.jsp** is requested, the method BookController.listBooks() also instantiate a new Book object and puts it in the map with the model name book.

```
map.put("book", new Book());
```

When the Add Book is clicked, the bookDialog pops up. Since it's a new object, the dialog form will be of all empty input fields, including the hidden "id" field. As defined by form action action="save", when the form is submitted, the controller method BookController.saveBook() is called to save the new book record. Since, the id field is empty, Hibernate's merge() method saves it as a new record.

Edit: Each rows has been provided with an Edit button. When the the button is clicked, it calles the method editBook() method and also passing the corresponding book id as the method argument. If you look into the method, you'll see that it makes an AJAX call to fetch the corresponding book record by calling the controller method BookController.getBook(). With the book id in the request parameter acquired via. @PathVariable, the controller fetches the corresponding book record, puts it in the map with the model name book and returns the view **bookForm.jsp**. Then the response is set as the content of *bookDialog* div and which is then displayed as a dialog. Note that this time the hidden id field will hold the id of the persistent book object. So, when you save the form, Hibernate's merge() method updates the record.

Delete: When you hit the delete button, it sends the corresponding book id to the controller which retrieves the id via @PathVariable. Note that the delete request is GET method, rather than POST.

Running the Application

Now, running the app is quite simple enough.

1. Open the Command Prompt
2. Go to the root project directory (bookstore)
3. Run the **mvn** command to download all dependent JARs.
    - Type **mvn** and hit <Enter>.
4. Run Tomcat server
    - **mvn tomcat7:run**
5. Go to the browser and enter the following URL:
    - **http://localhost:8080/bookstore/book/**
    - The port number might be different in your case. Please have a look at the tomcat log in console for that.

Download

The source code is available in GitHub (https://github.com/). You can either download the zip or directly import the project in eclipse from the following location:

- Download bookstore example (https://github.com/frenmanoj/bookstore)

Questions?

I must have missed many things to explain. If you have any questions or feedback, feel free to put them in the comment below.

at 3:42:00 PM (2014-05-25T15:42:00-05:00) (http://shrestha-manoj.blogspot.in/2014/05/spring-mvc-maven-hibernate-crud-example.html) (https://www.blogger.com/email-post.g?blogID=6571417643057380&postID=3226123106019342441)

G+1 +9 Recommend this on Google

Labels: spring mvc maven hibernate (http://shrestha-manoj.blogspot.in/search/label/spring%20%20mvc%20maven%20hibernate) , spring maven hibernate (http://shrestha-manoj.blogspot.in/search/label/spring%20maven%20hibernate) , Spring MVC (http://shrestha-manoj.blogspot.in/search/label/Spring%20MVC) , spring mvc + maven + hibernate (http://shrestha-manoj.blogspot.in/search/label/spring%20mvc%20%2B%20maven%20%2B%20hibernate) , spring mvc crud (http://shrestha-manoj.blogspot.in/search/label/spring%20mvc%20crud) , spring mvc crud example (http://shrestha-manoj.blogspot.in/search/label/spring%20mvc%20crud%20example) , spring mvc form (http://shrestha-manoj.blogspot.in/search/label/spring%20mvc%20form)

40 comments Google+

Add a comment as Bubble mishra

Top comments

Due N 1 week ago - Shared publicly
Thanks for the great tutorial Manoj, I found also one great tutorial for you to review, this is Docker + Spring boot + JPA tutorial on how to run a hashtag messaging API
http://weall.com/book/chapter?name=Running%20Hashtag%20Messaging%20application%20inside%20a%20Docker%20container

+1 · Reply

Ferhad 10 months ago
Nice Tutorial

khyati 1 year ago
Hi Manoj Shrestha, Thank you very much for such a nice tutorial..........its working perfect without any error................nice explination...

Unknown 1 year ago
Request processing failed; nested exception is org.springframework.transaction.CannotCreateTransactionException: Could not open Hibernate Session for transaction; nested exception is org.hibernate.exception.GenericJDBCException: Could not open connection

Anuja Mantode 1 year ago
please share test part of this project

Anuja Mantode 1 year ago
I had imported this project in eclipse bt it give java build errors i.e. src java test missing where is the code of java src test and src main

Unknown 1 year ago
colleagues and I run, I had a problem with the driver and ignore nodifique

1 year ago

please provide proper explanation for the above project.

Unknown 1 year ago
Manoj Shrestha the edit does not work for me, not shown me the form, we would be grateful for your help, if you could give me some information because not to be, otherwise no problem, I'm new to the philosophy of Spring MVC and finally find something like what I was looking for some time thank you very much

1 year ago
Can you share the test part of the project aswell???

Santosh pokhrel 1 year ago
can you please give any method to write those jsp file without using javascript and jquery ..i am trying to do but i am getting very difficulty..

Rajesh 1 year ago
Awesome example.. Thanks Manoj..

1 year ago
What is the Difference between BookServiceImpl and BookDaoImpl?

1 year ago
Nice example and it really works for me and gives me perfect what I was looking for....

Muhammed Riyas T 1 year ago
Hi Manoj Shrestha, Your spring mvc crud example is excellent. We already follow this methods. Now I look for a good spring boot curd example like this. I need clarification in configurations. I expect a good spring boot curd example app from you side(Spring Boot + Spring MVC + Maven + Hibernate CRUD Example Complete). Try to add spring security (role based login using databse values (user, role, user_role))

qurbonov 1 year ago
Hi Manoj please help with : WARNING: No mapping found for HTTP request with URI [/CallCenterDemo/book/$%7BactionUrl%20%7D] in DispatcherServlet with name 'dispatcherServlet' Thanks

RAGHURAM RAJU BACHU 2 years ago
You are the best man! I have no words to speak, from last two weeks i was fighting and was continuously searching for building project on Spring hibernate integration. But could not start off, I searched for many best of best sites but each of them had issues. However yours code too had issues

but ultimately after 2days of complete hard work and your example I could finally do it. Actually I cant explain my emotions but I salute you. And one more thing you are Shreshta(Best) in spring

2 years ago

solved my problem above explained after update dynamic web module 2.3 to 3.0 because jstl1.2 support for dynamic web module 2.5 or above

2 years ago

Dear thanks a lot for the tutorial , list is not displaying on listBooks.jsp. output like that {$book.name} but not displaying the value while data is fetch from DB. what may be the reasons?please help.

2 years ago

Dear thanks a lot for the tutorial , list is not displaying on listBooks.jsp. what may be the reasons?

Show more

Newer Post (http://shrestha-manoj.blogspot.in/2014/06/getting-started-with-angularjs.html) Home (http://shrestha-manoj.blogspot.in/)
Older Post (http://shrestha-manoj.blogspot.in/2014/05/nepali-unicode-nepali-unicode-converter.html)

Subscribe to: Post Comments (Atom) (http://shrestha-manoj.blogspot.com/feeds/3226123106019342441/comments/default)

Powered by Blogger (https://www.blogger.com).