**https://www.jackrutorial.com/search/label/spring%20mvc%20tutorial**

**Setting up Database**

Execute the following MySQL script in order to create a database named ***jack\_rutorial\_demo*** with a table named ***customer***.

CREATE DATABASE `jack\_rutorial\_demo`;

CREATE TABLE `customer` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL DEFAULT '',

`lastname` varchar(45) NOT NULL DEFAULT '',

`gender` varchar(10) NOT NULL DEFAULT '',

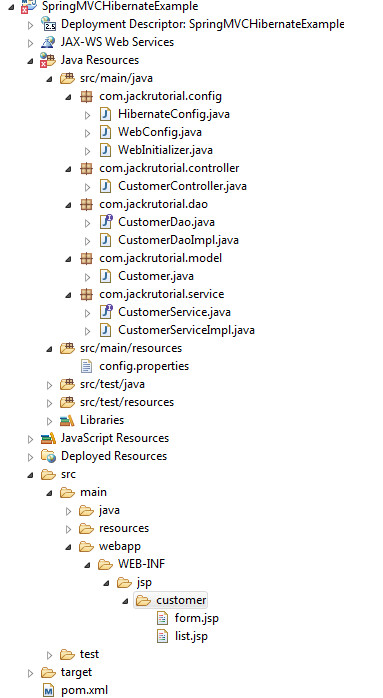
`address` varchar(200) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

**Project Structure**

The following screenshot shows final structure of the project.

[](https://2.bp.blogspot.com/-rHRjknT2Gt8/WnQWL2cyi0I/AAAAAAAAAMM/9eaBWHr5sgccjUpJwecAfR1xnE3qCaHKQCLcBGAs/s1600/Project%2BStructure%2BSpring%2B4%2BMVC%2BHibernate%2BMySQL%2BDatabase%2BMaven%2BCRUD%2BOperations%2BIntegration.jpg)

**Create Maven Project**

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "***com.jackrutorial***" in the "Group ID:" field
* Type "***SpringMVCHibernateExample***" in the "Artifact Id:" field
* Packaging -> War
* Click Finish.

**Maven Dependencies**

We will update ***pom.xml*** file to add the required dependencies for the following dependencies.

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringMVCHibernateExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.6.Final</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.1.0.Final</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

**Configure WebApp and Hibernate**

To configure WebApp, we create class ***WebConfig***, class ***WebInitializer***in ***src***folder with package name***com.jackrutorial.config*** and write the following code in it.  
  
***WebConfig.java***

package com.jackrutorial.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter {

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("resources/\*\*").addResourceLocations("/resources/");

}

@Bean

public InternalResourceViewResolver viewResolver(){

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/jsp/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

}

***WebInitializer.java***

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

To configure hibernate, we create class ***HibernateConfig***in ***src***folder with package name ***com.jackrutorial.config*** and write the following code in it.  
  
***HibernateConfig.java***

package com.jackrutorial.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.PropertySource;

import org.springframework.core.env.Environment;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.orm.hibernate4.HibernateTransactionManager;

import org.springframework.orm.hibernate4.LocalSessionFactoryBean;

import org.springframework.transaction.annotation.EnableTransactionManagement;

@Configuration

@EnableTransactionManagement

@ComponentScan({ "com.jackrutorial.config" })

@PropertySource(value = { "classpath:config.properties" })

public class HibernateConfig {

@Autowired

private Environment environment;

@Bean

public LocalSessionFactoryBean sessionFactory() {

LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

sessionFactory.setDataSource(dataSource());

sessionFactory.setPackagesToScan(new String[] { "com.jackrutorial.model" });

sessionFactory.setHibernateProperties(hibernateProperties());

return sessionFactory;

}

@Bean

public DataSource dataSource() {

DriverManagerDataSource dataSource = new DriverManagerDataSource();

dataSource.setDriverClassName(environment.getRequiredProperty("jdbc.driverClassName"));

dataSource.setUrl(environment.getRequiredProperty("jdbc.url"));

dataSource.setUsername(environment.getRequiredProperty("jdbc.username"));

dataSource.setPassword(environment.getRequiredProperty("jdbc.password"));

return dataSource;

}

private Properties hibernateProperties() {

Properties properties = new Properties();

properties.put("hibernate.dialect", environment.getRequiredProperty("hibernate.dialect"));

properties.put("hibernate.show\_sql", environment.getRequiredProperty("hibernate.show\_sql"));

properties.put("hibernate.format\_sql", environment.getRequiredProperty("hibernate.format\_sql"));

return properties;

}

@Bean

@Autowired

public HibernateTransactionManager transactionManager(SessionFactory s) {

HibernateTransactionManager txManager = new HibernateTransactionManager();

txManager.setSessionFactory(s);

return txManager;

}

}

We still have to configure ***config.properties*** file in classpath. We create a ***config.properties*** file in ***src/main/resources*** folder and write the following code in it.

jdbc.driverClassName = com.mysql.jdbc.Driver

jdbc.url = jdbc:mysql://localhost:3306/jack\_rutorial\_demo

jdbc.username = root

jdbc.password = root

hibernate.dialect = org.hibernate.dialect.MySQLDialect

hibernate.show\_sql = true

hibernate.format\_sql = false

MySQL Database Name: ***jack\_rutorial\_demo***  
Password for MySQL: ***root***  
Username for MySQL: ***root***

**Create Hibernate Entity Bean**

Create a ***Customer***class under ***com.jackrutorial.model*** package and write the following code in it.  
  
***Customer.java***

package com.jackrutorial.model;

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="customer")

public class Customer {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "firstname")

private String firstname;

@Column(name = "lastname")

private String lastname;

@Column(name = "gender")

private String gender;

@Column(name = "address")

private String address;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public String getGender() {

return gender;

}

public void setGender(String gender) {

this.gender = gender;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

In the above code, we are using JPA annotations in our entity bean class. Our entity bean maps to ***customer***table in MySQL database.

**Creating DAO Layer**

Create a ***CustomerDao***Interface under ***com.jackrutorial.dao*** package and write the following code in it.  
  
***CustomerDao.java***

package com.jackrutorial.dao;

import java.util.List;

import com.jackrutorial.model.Customer;

public interface CustomerDao {

public List listAllCustomers();

public void saveOrUpdate(Customer customer);

public Customer findCustomerById(int id);

public void deleteCustomer(int id);

}

Create a ***CustomerDaoImpl***class implements ***CustomerDao***Interface under ***com.jackrutorial.dao*** package and write the following code in it.  
  
***CustomerDaoImpl.java***

package com.jackrutorial.dao;

import java.util.List;

import org.hibernate.Criteria;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

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

import org.springframework.stereotype.Repository;

import com.jackrutorial.model.Customer;

@Repository

public class CustomerDaoImpl implements CustomerDao {

@Autowired

private SessionFactory sessionFactory;

protected Session getSession(){

return sessionFactory.getCurrentSession();

}

@SuppressWarnings("unchecked")

public List listAllCustomers() {

Criteria criteria = getSession().createCriteria(Customer.class);

return (List) criteria.list();

}

public void saveOrUpdate(Customer customer) {

getSession().saveOrUpdate(customer);

}

public Customer findCustomerById(int id) {

Customer customer = (Customer) getSession().get(Customer.class, id);

return customer;

}

public void deleteCustomer(int id) {

Customer customer = (Customer) getSession().get(Customer.class, id);

getSession().delete(customer);

}

}

**Creating Service Layer**

Create a ***CustomerService***Interface under package ***com.jackrutorial.service*** package and write the following code in it.  
  
***CustomerService.java***

package com.jackrutorial.service;

import java.util.List;

import com.jackrutorial.model.Customer;

public interface CustomerService {

public List listAllCustomers();

public void saveOrUpdate(Customer customer);

public Customer findCustomerById(int id);

public void deleteCustomer(int id);

}

Create a ***CustomerServiceImpl***class implements ***CustomerService***Interface under ***com.jackrutorial.service*** package and write the following code in it.  
  
***CustomerServiceImpl.java***

package com.jackrutorial.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.jackrutorial.dao.CustomerDao;

import com.jackrutorial.model.Customer;

@Service

@Transactional

public class CustomerServiceImpl implements CustomerService {

CustomerDao customerDao;

@Autowired

public void setCustomerDao(CustomerDao customerDao) {

this.customerDao = customerDao;

}

public List listAllCustomers() {

return customerDao.listAllCustomers();

}

public void saveOrUpdate(Customer customer) {

customerDao.saveOrUpdate(customer);

}

public Customer findCustomerById(int id) {

return customerDao.findCustomerById(id);

}

public void deleteCustomer(int id) {

customerDao.deleteCustomer(id);

}

}

**Creating Controller Layer**

Create a ***CustomerController***class under package ***com.jackrutorial.controller*** package and write the following code in it.  
  
***CustomerController.java***

package com.jackrutorial.controller;

import java.util.List;

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

import org.springframework.stereotype.Controller;

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;

import org.springframework.web.servlet.ModelAndView;

import com.jackrutorial.model.Customer;

import com.jackrutorial.service.CustomerService;

@Controller

@RequestMapping(value="/customer")

public class CustomerController {

@Autowired

CustomerService customerService;

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

public ModelAndView list(){

ModelAndView model = new ModelAndView("customer/list");

List list = customerService.listAllCustomers();

model.addObject("list", list);

return model;

}

@RequestMapping(value="/update/{id}", method=RequestMethod.GET)

public ModelAndView update(@PathVariable("id") int id){

ModelAndView model = new ModelAndView("customer/form");

Customer customer = customerService.findCustomerById(id);

model.addObject("customerForm", customer);

return model;

}

@RequestMapping(value="/delete/{id}", method=RequestMethod.GET)

public ModelAndView delete(@PathVariable("id") int id){

customerService.deleteCustomer(id);

return new ModelAndView("redirect:/customer/list");

}

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

public ModelAndView add(){

ModelAndView model = new ModelAndView("customer/form");

Customer customer = new Customer();

model.addObject("customerForm", customer);

return model;

}

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

public ModelAndView save(@ModelAttribute("customerForm") Customer customer){

customerService.saveOrUpdate(customer);

return new ModelAndView("redirect:/customer/list");

}

}

**Creating JSP Views**

Create ***customer***folder under ***src\main\webapp\WEB-INF\jsp\customer*** folder.  
Create ***list.jsp*** file under ***src\main\webapp\WEB-INF\jsp\customer*** folder and write the following code in it.  
  
***list.jsp***

<%@ 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" %>

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

<!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>Customer</title>

</head>

<body>

<spring:url value="/customer/add" var="addURL" />

<a href="${addURL }">Add new Customer</a>

<table width="100%" border="1">

<tr>

<td>ID</td>

<td>Firstname</td>

<td>Lastname</td>

<td>Gender</td>

<td>Address</td>

<td colspan="2">Action</td>

</tr>

<c:forEach items="${list }" var="customer" >

<tr>

<td>${customer.id }</td>

<td>${customer.firstname }</td>

<td>${customer.lastname }</td>

<td>${customer.gender }</td>

<td>${customer.address }</td>

<td>

<spring:url value="/customer/update/${customer.id }" var="updateURL" />

<a href="${updateURL }">Update</a>

</td>

<td>

<spring:url value="/customer/delete/${customer.id }" var="deleteURL" />

<a href="${deleteURL }">Delete</a>

</td>

</tr>

</c:forEach>

</table>

</body>

</html>

Create ***form.jsp*** file under ***src\main\webapp\WEB-INF\jsp\customer*** folder and write the following code in it.  
  
***form.jsp***

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

pageEncoding="ISO-8859-1"%>

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

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

<!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>Form</title>

</head>

<body>

<spring:url value="/customer/save" var="saveURL" />

<form:form action="${saveURL }" method="POST" modelAttribute="customerForm">

<form:hidden path="id"/>

<table>

<tr>

<td>Firstname</td>

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

</tr>

<tr>

<td>Lastname</td>

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

</tr>

<tr>

<td>Gender: </td>

<td>

<form:radiobutton path="gender" value="Male" /> Male

<form:radiobutton path="gender" value="Female" /> Female

</td>

</tr>

<tr>

<td>Address</td>

<td><form:textarea path="address" rows="3" /></td>

</tr>

<tr>

<td></td>

<td><button type="submit">Save</button></td>

</tr>

</table>

</form:form>

</body>

</html>

**Building**

* Right click this project
* Select Run As -> Maven clean
* Right click this project
* Select Run As -> Maven install

**Configuring Apache Tomcat**

* Under Servers tab, click link "No servers are available. Click this link to create a new server ...", select Apache tomcat 7
* Click Finish
* Right click "Tomcat v7.0 Server at localhost [Stopped, Republish]", select "Add and Remove ..."
* Add ***SpringMVCHibernateExample***project, then Click Finish
* Open ***server.xml*** file under Servers Folder
* Find line

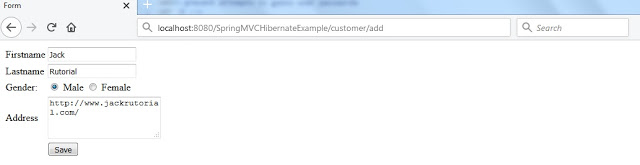
<Context docBase="***SpringMVCHibernateExample***" path="/SpringMVCHibernateExample" reloadable="true" ***source="org.eclipse.jst.jee.server:SpringMVCHibernateExample"*** />  
  
Update its as below:  
  
<Context docBase="***<Project Folder Location>\SpringMVCHibernateExample\target\SpringMVCHibernateExample-0.0.1-SNAPSHOT\***" path="/SpringMVCHibernateExample" reloadable="true" />  
  
***Watch video add Apache Tomcat Server in Eclipse IDE***

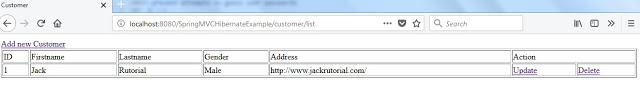
**Run application & Check result**

* Start Apache Tomcat from Eclipse IDE.
* Type the following URLs in browser's address bar to open the customer list.

http://localhost:8080/SpringMVCHibernateExample/customer/list

***Below screenshots shows the view pages for our application.***

[](https://1.bp.blogspot.com/-TumTjs14dR4/WnQYBrgfRpI/AAAAAAAAAMY/XYtoI5VnmkM385ujMRVC5R8qWBgfBJT0QCLcBGAs/s1600/add%2Bnew%2Bcustomer%2Bform.jpg)

[](https://4.bp.blogspot.com/-2JmkfrFs8DU/WnQYLNJyRqI/AAAAAAAAAMc/HOY3p7XggQAYrtutpy-tK1X4-wkdSNtSwCLcBGAs/s1600/customer%2Blist%2Bpage.jpg)

**Required Libraries**

# Creating a Web Application With Spring 4 MVC Example for User Signup, Login and Logout with Password encoder Tutorial

In [Java](https://www.jackrutorial.com/search/label/java?max-results=6) [Java Tutorial](https://www.jackrutorial.com/search/label/java%20tutorial?max-results=6)Published On [January 30, 2018](https://www.jackrutorial.com/2018/01/creating-a-web-application-with-spring-4-mvc-example-for-user-signup-login-and-logout-with-password-encoder-using-eclipse-mysql-database.html)

## Overview

In this tutorial, we show you how to create Web Application with Spring 4 MVC, Spring Security User Registration, Spring MVC Login Form and Logout Example using Eclipse IDE, Mysql Database  
  
***Follow the steps mentioned below to develop this application.***

[](https://1.bp.blogspot.com/-kjDii2S0oLk/WsZEBcBqKnI/AAAAAAAAAo0/LqOTrMBC7zwtJ6olPn_h7wEUuGGjT_2zwCLcBGAs/s1600/Spring-4-MVC-Example-for-User-Signup%252C-Login-and-Logout-with-Password-encoder.jpg)

## Video Tutorials

## Create Database

CREATE DATABASE `jack\_rutorial\_demo`;

## Create Database Table

*Creating `users` Table*

CREATE TABLE `users` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`username` varchar(45) NOT NULL DEFAULT '',

`password` varchar(200) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

*Creating `user\_roles` Table*

CREATE TABLE `user\_roles` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`username` varchar(45) NOT NULL DEFAULT '',

`role` varchar(45) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

*Creating `persistent\_logins` Table*

CREATE TABLE `persistent\_logins` (

`username` varchar(64) NOT NULL,

`series` varchar(64) NOT NULL,

`token` varchar(64) NOT NULL,

`last\_used` timestamp NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`series`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

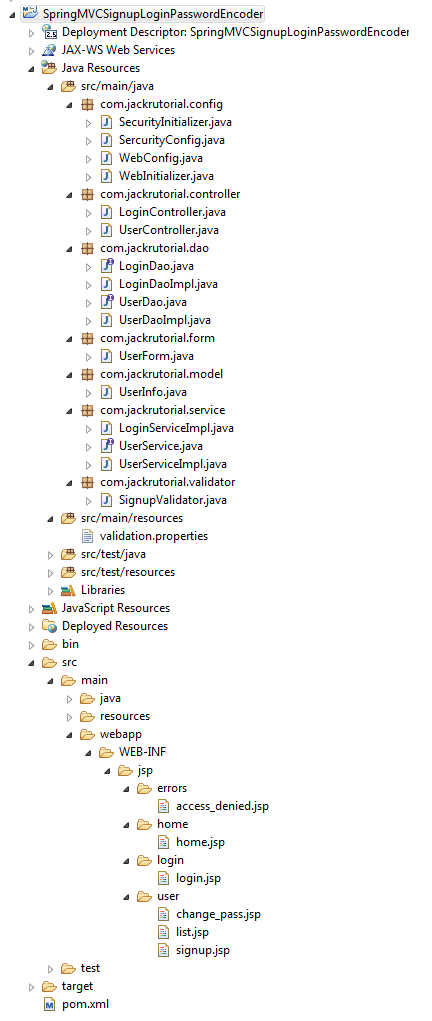
## Creating a default admin user

The following code creates a users named “**admin**” with password is '**123**', defines an “**ROLE\_ADMIN**” role.

INSERT INTO USERS(USERNAME,PASSWORD) VALUES('admin','$2a$10$Ii8O.wtxIeYQuXcbhcQ/WOngxwb1KrnAVOOLi6SaW8KfPWq7O6tsa');

INSERT INTO USER\_ROLES(USERNAME,ROLE) VALUES('admin','ROLE\_ADMIN');

## Project Structure

[](https://4.bp.blogspot.com/-6vEmEsIMono/WnAjby7iMcI/AAAAAAAAAJE/a9VvKIkVW3IdOblRWmI3ymUmMs2YVImAQCLcBGAs/s1600/Project%2BStructure%2BSpring%2B4%2BMVC%2BSignup%2BLogin%2BLogout%2BForm%2Bwith%2Bpassword%2Bencoder.png)

## Create Maven Project

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "**com.jackrutorial**" in the "Group ID:" field
* Type "**SpringMVCSignupLoginPasswordEncoder**" in the "Artifact Id:" field
* Packaging -> War
* Click Finish.

## Update pom.xml to include required dependencies

*Open pom.xml file and add the following dependencies in it.*

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringMVCSignupLoginPasswordEncoder</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>4.0.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>4.0.4.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>jsp-api</artifactId>

<version>2.0</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

## Configure WebApp and Spring Security

* Right click to the src folder, select New -> Package
* Enter "com.jackrutorial.config" in Name: field
* Click Finish

*Create a****WebConfig****class under****com.jackrutorial.config****package and write the following code in it*

package com.jackrutorial.config;

import javax.sql.DataSource;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.support.ResourceBundleMessageSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.datasource.DriverManagerDataSource;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc

@Configuration

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter {

@Autowired

DataSource dataSource;

@Bean

public NamedParameterJdbcTemplate geNamedParameterJdbcTemplate(){

return new NamedParameterJdbcTemplate(dataSource);

}

@Bean

public DataSource getDataSource(){

DriverManagerDataSource ds = new DriverManagerDataSource();

ds.setDriverClassName("com.mysql.jdbc.Driver");

ds.setUrl("jdbc:mysql://localhost/jack\_rutorial\_demo");

ds.setUsername("root");

ds.setPassword("root");

return ds;

}

@Bean

public InternalResourceViewResolver viewResolver(){

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/jsp/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

@Bean

public ResourceBundleMessageSource messageSource(){

ResourceBundleMessageSource rb = new ResourceBundleMessageSource();

rb.setBasenames(new String[]{"validation"});

return rb;

}

}

*Create a****WebInitializer****class under****com.jackrutorial.config****package and write the following code in it*

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

*Create a****SercurityConfig****class under****com.jackrutorial.config****package and write the following code in it*

package com.jackrutorial.config;

import javax.sql.DataSource;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;

import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import com.jackrutorial.service.LoginServiceImpl;

@Configuration

@EnableWebSecurity

public class SercurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

DataSource dataSource;

@Autowired

LoginServiceImpl loginServiceImpl;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(loginServiceImpl);

auth.authenticationProvider(authenticationProvider());

}

@Bean

public DaoAuthenticationProvider authenticationProvider(){

DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();

authenticationProvider.setUserDetailsService(loginServiceImpl);

authenticationProvider.setPasswordEncoder(passwordEncoder());

return authenticationProvider;

}

@Bean

public PasswordEncoder passwordEncoder(){

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity http) throws Exception{

http.csrf().disable();

http.authorizeRequests().antMatchers("/login", "/user/sigup", "/user/register").permitAll();

http.authorizeRequests().antMatchers("/", "/home")

.access("hasRole('ROLE\_ADMIN') or hasRole('ROLE\_USER')");

http.authorizeRequests().antMatchers("/user/list")

.access("hasRole('ROLE\_ADMIN')");

http.authorizeRequests().and().formLogin()

.loginProcessingUrl("/j\_spring\_security\_check")

.loginPage("/login")

.failureUrl("/login?error=true")

.usernameParameter("username")

.passwordParameter("password")

.and().logout().logoutUrl("/j\_spring\_security\_logout").logoutSuccessUrl("/login")

.and().rememberMe()

.tokenRepository(persistentTokenRepository())

.tokenValiditySeconds(60\*60)

.and().exceptionHandling().accessDeniedPage("/accessDenied");

}

@Bean

public PersistentTokenRepository persistentTokenRepository(){

JdbcTokenRepositoryImpl db = new JdbcTokenRepositoryImpl();

db.setDataSource(dataSource);

return db;

}

}

*Create a****SecurityInitializer****class under****com.jackrutorial.config****package and write the following code in it*

package com.jackrutorial.config;

import org.springframework.security.web.context.AbstractSecurityWebApplicationInitializer;

public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer {

}

## Creating Model Layer

*Create a****UserInfo****class under package****com.jackrutorial.model****and write the following code in it*

package com.jackrutorial.model;

public class UserInfo {

private String username;

private String password;

public UserInfo() {

super();

}

public UserInfo(String username, String password) {

super();

this.username = username;

this.password = password;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

## Creating Form Layer

*Create a****UserForm****class under package****com.jackrutorial.form****and write the following code in it*

package com.jackrutorial.form;

public class UserForm {

private String username;

private String password;

private String confirmPassword;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getConfirmPassword() {

return confirmPassword;

}

public void setConfirmPassword(String confirmPassword) {

this.confirmPassword = confirmPassword;

}

}

## Creating DAO Layer

*Create a****LoginDao****Interface under****com.jackrutorial.dao****package and write the following code in it*

package com.jackrutorial.dao;

import java.util.List;

import com.jackrutorial.model.UserInfo;

public interface LoginDao {

UserInfo findUserInfo(String username);

List getUserRoles(String username);

}

*Create a****LoginDaoImpl****class implements****LoginDao****Interface under****com.jackrutorial.dao****package and write the following code in it*

package com.jackrutorial.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

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

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.stereotype.Repository;

import com.jackrutorial.model.UserInfo;

@Repository

public class LoginDaoImpl implements LoginDao {

NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Autowired

public void setNamedParameterJdbcTemplate(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public UserInfo findUserInfo(String username) {

String sql = "select username,password from users where username = :username";

UserInfo userInfo = namedParameterJdbcTemplate

.queryForObject(sql, getSqlParameterSource(username, ""), new UserInfoMapper());

return userInfo;

}

private static final class UserInfoMapper implements RowMapper{

public UserInfo mapRow(ResultSet rs, int rowNum) throws SQLException {

String username = rs.getString("username");

String password = rs.getString("password");

return new UserInfo(username, password);

}

}

private SqlParameterSource getSqlParameterSource(String username, String password){

MapSqlParameterSource parameterSource = new MapSqlParameterSource();

parameterSource.addValue("username", username);

parameterSource.addValue("password", password);

return parameterSource;

}

public List getUserRoles(String username) {

String sql = "select role from user\_roles where username = :username";

List roles = namedParameterJdbcTemplate

.queryForList(sql, getSqlParameterSource(username, ""), String.class);

return roles;

}

}

*Create a****UserDao****interface under****com.jackrutorial.dao****package and write the following code in it*

package com.jackrutorial.dao;

import java.util.List;

import com.jackrutorial.model.UserInfo;

public interface UserDao {

public List list();

public UserInfo findUserByUsername(String username);

public void update(String username, String password);

public void add(String username, String password);

public boolean userExists(String username);

}

*Create a****UserDaoImpl****class implements****UserDao****Interface under****com.jackrutorial.dao****package and write the following code in it*

package com.jackrutorial.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

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

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.stereotype.Repository;

import com.jackrutorial.model.UserInfo;

@Repository

public class UserDaoImpl implements UserDao {

NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Autowired

public void setNamedParameterJdbcTemplate(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public List list() {

String sql = "select username from users";

List list = namedParameterJdbcTemplate

.query(sql, getSqlParameterSource(null, null), new UserMapper());

return list;

}

private SqlParameterSource getSqlParameterSource(String username, String password){

MapSqlParameterSource parameterSource = new MapSqlParameterSource();

if(username != null){

parameterSource.addValue("username", username);

}

if(password != null){

parameterSource.addValue("password", password);

}

return parameterSource;

}

private static final class UserMapper implements RowMapper{

public UserInfo mapRow(ResultSet rs, int rowNum) throws SQLException {

UserInfo user = new UserInfo();

user.setUsername(rs.getString("username"));

return user;

}

}

public UserInfo findUserByUsername(String username) {

String sql = "select username from users where username = :username";

List list = namedParameterJdbcTemplate

.query(sql, getSqlParameterSource(username, null), new UserMapper());

return list.get(0);

}

public void update(String username, String password) {

String sql = "update users set password = :password where username = :username";

namedParameterJdbcTemplate.update(sql, getSqlParameterSource(username, password));

}

public void add(String username, String password) {

String sql = "insert into users(username, password) values(:username, :password)";

namedParameterJdbcTemplate.update(sql, getSqlParameterSource(username, password));

sql = "insert into user\_roles(username, role) values(:username, 'ROLE\_USER')";

namedParameterJdbcTemplate.update(sql, getSqlParameterSource(username, password));

}

public boolean userExists(String username) {

String sql = "select \* from users where username = :username";

List list = namedParameterJdbcTemplate

.query(sql, getSqlParameterSource(username, null), new UserMapper());

if(list.size() > 0){

return true;

}

return false;

}

}

## Creating Service Layer

*Create a****LoginServiceImpl****class implements****UserDetailsService****Interface under****com.jackrutorial.service****package and write the following code in it*

package com.jackrutorial.service;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.security.core.GrantedAuthority;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import com.jackrutorial.dao.LoginDao;

import com.jackrutorial.model.UserInfo;

@Service

public class LoginServiceImpl implements UserDetailsService {

LoginDao loginDao;

@Autowired

public void setLoginDao(LoginDao loginDao) {

this.loginDao = loginDao;

}

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

UserInfo userInfo = loginDao.findUserInfo(username);

if(userInfo == null){

throw new UsernameNotFoundException("username was not found in the database");

}

List roles = loginDao.getUserRoles(username);

List grantList = new ArrayList();

if(roles != null){

for(String role : roles){

GrantedAuthority authority = new SimpleGrantedAuthority(role);

grantList.add(authority);

}

}

UserDetails userDetails = new User(userInfo.getUsername(), userInfo.getPassword(), grantList);

return userDetails;

}

}

*Create a****UserService****Interface under package****com.jackrutorial.service****package and write the following code in it*

package com.jackrutorial.service;

import java.util.List;

import com.jackrutorial.model.UserInfo;

public interface UserService {

public List list();

public UserInfo findUserByUsername(String username);

public void update(String username, String password);

public void add(String username, String password);

public boolean userExists(String username);

}

*Create a****UserServiceImpl****class implements****UserService****Interface under****com.jackrutorial.service****package and write the following code in it*

package com.jackrutorial.service;

import java.util.List;

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

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

import com.jackrutorial.dao.UserDao;

import com.jackrutorial.model.UserInfo;

@Service

public class UserServiceImpl implements UserService {

UserDao userDao;

@Autowired

private PasswordEncoder passwordEncoder;

@Autowired

public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

public List list() {

return userDao.list();

}

public UserInfo findUserByUsername(String username) {

return userDao.findUserByUsername(username);

}

public void update(String username, String password) {

userDao.update(username, passwordEncoder.encode(password));

}

public void add(String username, String password) {

userDao.add(username, passwordEncoder.encode(password));

}

public boolean userExists(String username) {

return userDao.userExists(username);

}

}

## Custom Validator

*Create a****SignupValidator****class implements****Validator****Interface under****com.jackrutorial.validator****package and write the following code in it*

package com.jackrutorial.validator;

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

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.ValidationUtils;

import org.springframework.validation.Validator;

import com.jackrutorial.form.UserForm;

import com.jackrutorial.service.UserService;

@Component

public class SignupValidator implements Validator {

@Autowired

UserService userService;

public boolean supports(Class clazz) {

return UserForm.class.isAssignableFrom(clazz);

}

public void validate(Object target, Errors errors) {

UserForm user = (UserForm) target;

ValidationUtils.rejectIfEmpty(errors, "username", "notEmpty.username");

ValidationUtils.rejectIfEmpty(errors, "password", "notEmpty.password");

ValidationUtils.rejectIfEmpty(errors, "confirmPassword", "notEmpty.confirmPassword");

if(user.getPassword() != null && user.getConfirmPassword() != null &&

!user.getPassword().equals(user.getConfirmPassword())){

errors.rejectValue("password", "notMatch.confirmPassword");

}

if(userService.userExists(user.getUsername())){

errors.rejectValue("username", "exists.username");

}

}

}

*We will define the error codes and their corresponding values in****validation.properties****file under src/main/resources folder as follows.*

notEmpty.username=Username is required

notEmpty.password=Password is required

notEmpty.confirmPassword=Confirm Password is required

notMatch.confirmPassword=Password does not match the confirm password

exists.username=username already exists

## Creating Controller Layer

*Create a****LoginController****class under package****com.jackrutorial.controller****package and write the following code in it*

package com.jackrutorial.controller;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;

import org.springframework.stereotype.Controller;

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

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

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

import org.springframework.web.servlet.ModelAndView;

@Controller

@RequestMapping(value="/")

public class LoginController {

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

public ModelAndView login(@RequestParam(value="error", required = false) String error){

ModelAndView model = new ModelAndView();

if(error != null){

model.addObject("msg", "The username or password is incorrect!");

}

model.setViewName("login/login");

return model;

}

@RequestMapping(value={"/", "/home"}, method=RequestMethod.GET)

public ModelAndView home(){

ModelAndView model = new ModelAndView();

model.setViewName("home/home");

return model;

}

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

public String logoutPage(HttpServletRequest request, HttpServletResponse response){

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

if(auth != null){

new SecurityContextLogoutHandler().logout(request, response, auth);

}

return "redirect:/login";

}

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

public ModelAndView accessDenied(){

ModelAndView model = new ModelAndView();

model.setViewName("errors/access\_denied");

return model;

}

}

*Create a****UserController****class under package****com.jackrutorial.controller****package and add the custom validator in it.*

package com.jackrutorial.controller;

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;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.jackrutorial.form.UserForm;

import com.jackrutorial.model.UserInfo;

import com.jackrutorial.service.UserService;

import com.jackrutorial.validator.SignupValidator;

@Controller

@RequestMapping("/user")

public class UserController {

@Autowired

SignupValidator signupValidator;

@Autowired

UserService userService;

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

public ModelAndView list(){

ModelAndView model = new ModelAndView("user/list");

model.addObject("list", userService.list());

return model;

}

@RequestMapping(value="/changePass/{username}", method=RequestMethod.GET)

public ModelAndView changePass(@PathVariable("username") String username){

ModelAndView model = new ModelAndView("user/change\_pass");

model.addObject("user", userService.findUserByUsername(username));

return model;

}

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

public ModelAndView save(@ModelAttribute("user") UserInfo user){

ModelAndView model = changePass(user.getUsername());

userService.update(user.getUsername(), user.getPassword());

model.addObject("msg", "Your password has been changed successfully!");

return model;

}

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

public ModelAndView signup(){

ModelAndView model = new ModelAndView("user/signup");

model.addObject("userForm", new UserForm());

return model;

}

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

public String register(@ModelAttribute("userForm") UserForm userForm,

BindingResult result, RedirectAttributes redirectAttributes){

signupValidator.validate(userForm, result);

if(result.hasErrors()){

return "/user/signup";

} else {

userService.add(userForm.getUsername(), userForm.getPassword());

redirectAttributes.addFlashAttribute("msg", "Your account has been created successfully!");

return "redirect:/login";

}

}

}

## Creating JSP Views

*Create****login****folder under****src\main\webapp\WEB-INF\jsp****folder.*  
*Create****login.jsp****file under****src\main\webapp\WEB-INF\jsp\login****folder and write the following code in it.*

<%@ 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" %>

<!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>Login</title>

</head>

<body>

<form name="loginForm" action='<c:url value="j\_spring\_security\_check" />' method="post" >

<table>

<tr>

<td colspan="2">Login</td>

</tr>

<tr>

<td colspan="2">${msg }</td>

</tr>

<tr>

<td>Username: </td>

<td>

<input type="text" name="username" />

</td>

</tr>

<tr>

<td>Password: </td>

<td>

<input type="password" name="password" />

</td>

</tr>

<tr>

<td>Remember Me: </td>

<td>

<input type="checkbox" name="remember-me" />

</td>

</tr>

<tr>

<td></td>

<td>

<button type="submit">Login</button>

</td>

</tr>

<tr>

<td></td>

<td>

<a href='<c:url value="/user/signup" />'>Sign up</a>

</td>

</tr>

</table>

</form>

</body>

</html>

*Create****home****folder under****src\main\webapp\WEB-INF\jsp****folder.*  
*Create****home.jsp****file under****src\main\webapp\WEB-INF\jsp\home****folder and write the following code in it.*

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

<!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>Home</title>

</head>

<body>

<a href='<c:url value="/logout" />'>Logout</a>

<h1>Welcome

<c:if test="${pageContext.request.userPrincipal.name != null }">

${pageContext.request.userPrincipal.name }

</c:if> </h1>

<br/>

<a href='<c:url value="/user/list" />'>Users List</a>

</body>

</html>

*Create****user****folder under****src\main\webapp\WEB-INF\jsp****folder.*  
*Create****signup.jsp****file under****src\main\webapp\WEB-INF\jsp\user****folder and write the following code in it.*

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

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

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

<!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>Signup</title>

</head>

<body>

<spring:url value="/user/register" var="registerURL" />

<form:form action="${registerURL}" modelAttribute="userForm" method="post" >

<label>Username: </label>

<form:input path="username" type="text" />

<form:errors path="username" />

<br/>

<label>Password: </label>

<form:password path="password" />

<form:errors path="password" />

<br/>

<label>Confirm Password: </label>

<form:password path="confirmPassword" />

<form:errors path="confirmPassword" />

<br/>

<button type="submit">Sign up</button>

</form:form>

</body>

</html>

*Create****list.jsp****file under****src\main\webapp\WEB-INF\jsp\user****folder and write the following code in it.*

<%@ 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" %>

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

<!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>Users List</title>

</head>

<body>

<a href='<c:url value="/logout" />'>Logout</a>

<br/>

<table border="1">

<tr>

<td>Username</td>

<td>Action</td>

</tr>

<c:forEach items="${list }" var="user">

<tr>

<td>${user.username }</td>

<td>

<spring:url value="/user/changePass" var="changePassURL" />

<a href="${changePassURL }/${user.username }">Change Pass</a>

</td>

</tr>

</c:forEach>

</table>

</body>

</html>

*Create****change\_pass.jsp****file under****src\main\webapp\WEB-INF\jsp\user****folder and write the following code in it.*

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

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

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

<!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>Change Password</title>

</head>

<body>

<spring:url value="/user/save" var="saveURL" />

<form:form method="post" modelAttribute="user" action="${saveURL }">

<form:hidden path="username"/>

<table border="1">

<tr>

<td colspan="2">${msg }</td>

</tr>

<tr>

<td>Username: </td>

<td>${user.username }</td>

</tr>

<tr>

<td>Password: </td>

<td><form:password path="password" /></td>

</tr>

<tr>

<td></td>

<td>

<button type="submit" >Change Password</button>

</td>

</tr>

</table>

</form:form>

</body>

</html>

*Create****errors****folder under****src\main\webapp\WEB-INF\jsp****folder.*  
*Create****access\_denied.jsp****file under****src\main\webapp\WEB-INF\jsp\errors****folder and write the following code in it.*

<%@ 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>Access denied</title>

</head>

<body>

<h1>Access denied!</h1>

</body>

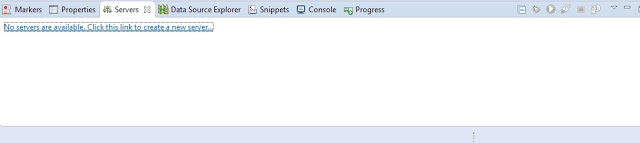
</html>

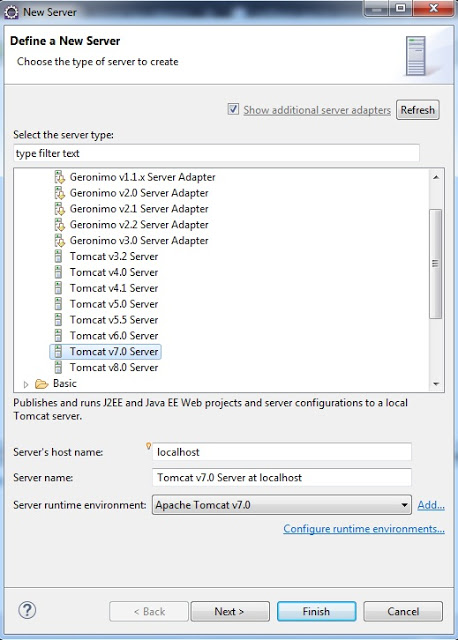
## Building

* Right click this project
* Select Run As -> **Maven clean**
* Right click this project
* Select Run As -> **Maven install**

## Configuring Apache Tomcat

* Under Servers tab, click link "No servers are available. Click this link to create a new server ...", select Apache tomcat 7
* Click Finish

[](https://2.bp.blogspot.com/-htNsfXUYRZc/Wmiq-_D6jrI/AAAAAAAAAGI/GlTmdzCTvm4_nSloYawfDfZtveO5hbaHwCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2Beclipse.jpg)

[[](https://4.bp.blogspot.com/-E9eOOzbOQUY/WmirUbsBtkI/AAAAAAAAAGM/hpdZL9HUWlU9jK4i-SWWHxRg_4vNVI_0wCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2B7%2Beclipse.jpg)](https://4.bp.blogspot.com/-E9eOOzbOQUY/WmirUbsBtkI/AAAAAAAAAGM/hpdZL9HUWlU9jK4i-SWWHxRg_4vNVI_0wCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2B7%2Beclipse.jpg)

* Right click "Tomcat v7.0 Server at localhost [Stopped, Republish]", select "Add and Remove ..."
* Add SpringMVCSignupLoginPasswordEncoder project, then Click Finish
* Open server.xml file under Servers Folder
* Find line

*<Context docBase="****SpringMVCSignupLoginPasswordEncoder****" path="/SpringMVCSignupLoginPasswordEncoder" reloadable="true" source="org.eclipse.jst.jee.server:SpringMVCSignupLoginPasswordEncoder" />*  
  
Update its as below:   
  
*<Context docBase="****<Project Folder Location>\SpringMVCSignupLoginPasswordEncoder\target\SpringMVCSignupLoginPasswordEncoder-0.0.1-SNAPSHOT\****" path="/****SpringMVCSignupLoginPasswordEncoder****" reloadable="true" />*

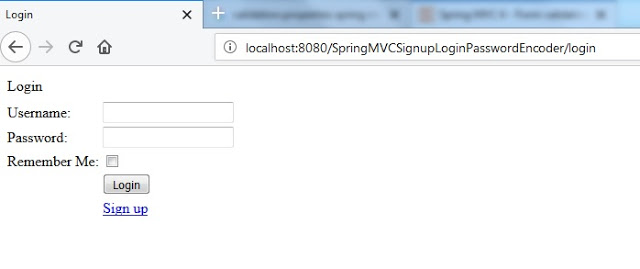
* Copy mysql-connector-java-5.1.45-bin.jar file to**<Apache Tomcat Folder Location>\apache-tomcat-7.0.68\lib\**

**Watch video add Apache Tomcat Server in Eclipse IDE**

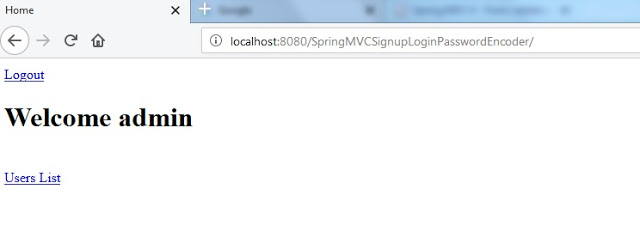
## Run application & Check result

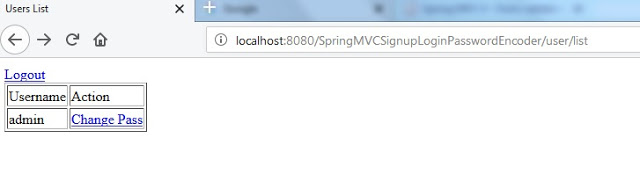
* Start Apache Tomcat from Eclipse IDE.
* Type the following URLs in browser's address bar to open the login from.

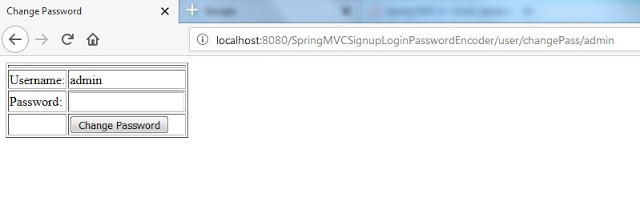
http://localhost:8080/SpringMVCSignupLoginPasswordEncoder/

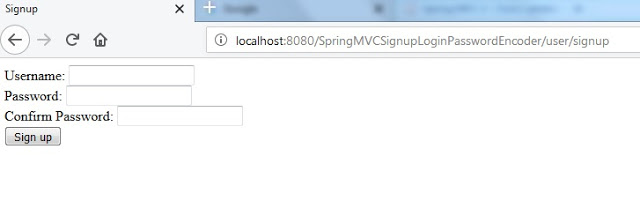
[](https://4.bp.blogspot.com/-ChZgb8P3Jlg/Wm_u5SLsi9I/AAAAAAAAAIM/D9IKPcz0NRIZ7qPTYLAqwtSK4PQ5fd9fQCLcBGAs/s1600/login_form.jpg)

Enter username/password: admin/123 to login

[](https://1.bp.blogspot.com/-cmT8Tq5TopI/Wm_v9SYTwAI/AAAAAAAAAIY/0FoFsCXtWXcdbpTx6SXnbEIQOyik6KJtgCLcBGAs/s1600/welcome_admin_home_page.jpg)

[](https://3.bp.blogspot.com/-ytW-dSTqYig/Wm_wRdKRiiI/AAAAAAAAAIc/AAd7GpbnAzArB1pmpAEa2RThDSL_8uUBwCLcBGAs/s1600/user_list_page.jpg)

[](https://2.bp.blogspot.com/-L_mbi92mkxA/Wm_wgXtuHKI/AAAAAAAAAIg/tSe_WZ6Vp6AF3n9V2yrKg7wWd2nlpMHUACLcBGAs/s1600/change_password_page.jpg)

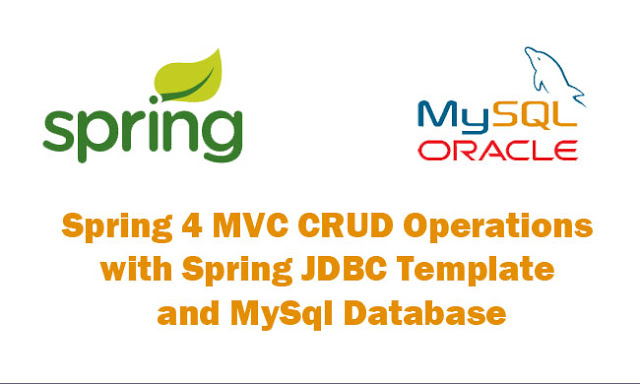
[](https://3.bp.blogspot.com/-aSEwEyYJXpQ/Wm_wz4TxWPI/AAAAAAAAAIo/DaXFIZCjPi8hnPATmVQ6zZQ0AzG5ZT6bwCLcBGAs/s1600/signup_page.jpg)

# Spring 4 MVC CRUD Operations with Spring JDBC Template and Mysql Database Tutorial

In [Java](https://www.jackrutorial.com/search/label/java?max-results=6) [Java Tutorial](https://www.jackrutorial.com/search/label/java%20tutorial?max-results=6)Published On [January 31, 2018](https://www.jackrutorial.com/2018/01/spring-mvc-crud-operations-with-spring-jdbc-template-and-mysql-database-tutorial.html)

## Overview

*In this tutorial, we show you how to create User CRUD (Create, Read, Update, Delete) Web Application using Spring 4 MVC with Spring JDBC Template using Eclipse IDE, Mysql Database.*  
  
**Follow the steps mentioned below to develop this application.**

[](https://3.bp.blogspot.com/-nwqbyldKAGU/WsZCXQZRS3I/AAAAAAAAAog/hXrOA-tb1Lg-8YX0C-RHqCjmcCcItA-bACLcBGAs/s1600/Spring-4-MVC-CRUD-Operations-with-Spring-JDBC-Template.jpg)

## Video Tutorials

## Database & Table Creation

The following MySQL script is used to create a database called **jack\_rutorial\_demo** with table **users**.

CREATE DATABASE `jack\_rutorial\_demo`;

CREATE TABLE `jack\_rutorial\_demo`.`users` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL DEFAULT '',

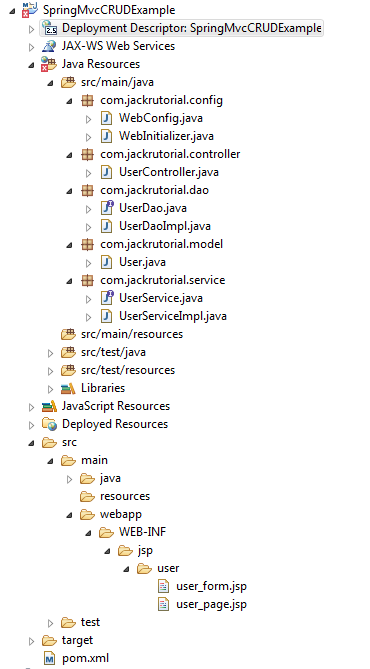
`lastname` varchar(100) NOT NULL DEFAULT '',

`address` varchar(100) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

## Project Structure

[](https://2.bp.blogspot.com/-n29B3fbqEDA/WnGWbdHuX7I/AAAAAAAAAJc/PkbsErfAy4U-T44epBGy61dtCxc1-zTngCLcBGAs/s1600/Project%2BStructure%2BSpring%2B4%2BMVC%2BCRUD%2BOperations%2Bwith%2BSpring%2BJDBC%2BTemplate.png)

## Create Maven Project

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "**com.jackrutorial**" in the "Group ID:" field
* Type "**SpringMvcCRUDExample**" in the "Artifact Id:" field
* Packaging -> War
* Click Finish.

## Maven Dependencies

We specify the dependency for the **Spring WebMVC, Spring Jdbc, jstl** and **Servlet api**. The rest dependencies will be automatically resolved by Maven. The updated **pom.xml**file will have the following code:

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringMvcCRUDExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

## Configure WebApp

* Right-click on src/main/java folder, New -> Package
* Enter  the package name as: "**com.jackrutorial.config**"
* Click Finish

Create a **WebConfig**class under **com.jackrutorial.config** package and write the following code in it.

package com.jackrutorial.config;

import javax.naming.NamingException;

import javax.sql.DataSource;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jndi.JndiTemplate;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter {

@Autowired

DataSource dataSource;

@Bean

public NamedParameterJdbcTemplate geNamedParameterJdbcTemplate(){

return new NamedParameterJdbcTemplate(dataSource);

}

@Bean

public DataSource getDataSource() throws NamingException{

JndiTemplate jndiTemplate = new JndiTemplate();

DataSource dataSource = (DataSource) jndiTemplate.lookup("java:comp/env/jdbc/springmvc");

return dataSource;

}

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

registry.addResourceHandler("/resources/\*\*").addResourceLocations("/resources/");

}

@Bean

public InternalResourceViewResolver viewResolver(){

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/jsp/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

}

Create a **WebInitializer**class under **com.jackrutorial.config** package and write the following code in it.

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

## Creating Model Layer

Create a **UserInfo**class under package **com.jackrutorial.model**, this class simply maps a row in the users table to a Java object and write the following code in it.

package com.jackrutorial.model;

public class User {

private Integer id;

private String firstname;

private String lastname;

private String address;

public User() {

super();

}

public User(Integer id) {

super();

this.id = id;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

## Creating DAO Layer

Create a **UserDao**Interface under **com.jackrutorial.dao** package and write the following code in it.

package com.jackrutorial.dao;

import java.util.List;

import com.jackrutorial.model.User;

public interface UserDao {

public List listAllUsers();

public void addUser(User user);

public void updateUser(User user);

public void deleteUser(int id);

public User findUserById(int id);

}

Create a **UserDaoImpl**class implements **UserDao**Interface under **com.jackrutorial.dao**package and write the following code in it.

package com.jackrutorial.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

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

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.stereotype.Repository;

import com.jackrutorial.model.User;

@Repository

public class UserDaoImpl implements UserDao {

NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Autowired

public void setNamedParameterJdbcTemplate(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public List listAllUsers() {

String sql = "SELECT id, firstname, lastname, address FROM users";

List list = namedParameterJdbcTemplate

.query(sql, getSqlParameterByModel(null), new UserMapper());

return list;

}

private SqlParameterSource getSqlParameterByModel(User user){

MapSqlParameterSource parameterSource = new MapSqlParameterSource();

if(user != null){

parameterSource.addValue("id", user.getId());

parameterSource.addValue("firstname", user.getFirstname());

parameterSource.addValue("lastname", user.getLastname());

parameterSource.addValue("address", user.getAddress());

}

return parameterSource;

}

private static final class UserMapper implements RowMapper{

public User mapRow(ResultSet rs, int rowNum) throws SQLException {

User user = new User();

user.setId(rs.getInt("id"));

user.setFirstname(rs.getString("firstname"));

user.setLastname(rs.getString("lastname"));

user.setAddress(rs.getString("address"));

return user;

}

}

public void addUser(User user) {

String sql = "INSERT INTO users(firstname, lastname, address)

VALUES(:firstname, :lastname, :address)";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(user));

}

public void updateUser(User user) {

String sql = "UPDATE users SET firstname = :firstname, lastname = :lastname

, address = :address WHERE id = :id";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(user));

}

public void deleteUser(int id) {

String sql = "DELETE FROM users WHERE id = :id";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(new User(id)));

}

public User findUserById(int id) {

String sql = "SELECT \* FROM users WHERE id = :id";

return namedParameterJdbcTemplate

.queryForObject(sql, getSqlParameterByModel(new User(id)), new UserMapper());

}

}

## Creating Service Layer

Create a **UserService**Interface under **com.jackrutorial.service** package and write the following code in it.

package com.jackrutorial.service;

import java.util.List;

import com.jackrutorial.model.User;

public interface UserService {

public List listAllUsers();

public void addUser(User user);

public void updateUser(User user);

public void deleteUser(int id);

public User findUserById(int id);

}

Create a **UserServiceImpl**class implements **UserService**Interface under **com.jackrutorial.service** package and write the following code in it.

package com.jackrutorial.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.jackrutorial.dao.UserDao;

import com.jackrutorial.model.User;

@Service

public class UserServiceImpl implements UserService {

UserDao userDao;

@Autowired

public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

public List listAllUsers() {

return userDao.listAllUsers();

}

public void addUser(User user) {

userDao.addUser(user);

}

public void updateUser(User user) {

userDao.updateUser(user);

}

public void deleteUser(int id) {

userDao.deleteUser(id);

}

public User findUserById(int id) {

return userDao.findUserById(id);

}

}

## Creating Controller Layer

Create a **UserController**under **com.jackrutorial.controller** package and write the following code in it.

package com.jackrutorial.controller;

import java.util.List;

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

import org.springframework.stereotype.Controller;

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;

import org.springframework.web.servlet.ModelAndView;

import com.jackrutorial.model.User;

import com.jackrutorial.service.UserService;

@Controller

@RequestMapping(value="/user")

public class UserController {

@Autowired

UserService userService;

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

public ModelAndView list(){

ModelAndView model = new ModelAndView("user/user\_page");

List list = userService.listAllUsers();

model.addObject("listUser", list);

return model;

}

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

public ModelAndView add(){

ModelAndView model = new ModelAndView("user/user\_form");

User user = new User();

model.addObject("userForm", user);

return model;

}

@RequestMapping(value="/update/{id}", method=RequestMethod.GET)

public ModelAndView update(@PathVariable("id") int id){

ModelAndView model = new ModelAndView("user/user\_form");

User user = userService.findUserById(id);

model.addObject("userForm", user);

return model;

}

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

public ModelAndView save(@ModelAttribute("userForm") User user){

if(user != null && user.getId() != null){

userService.updateUser(user);

} else {

userService.addUser(user);

}

return new ModelAndView("redirect:/user/list");

}

@RequestMapping(value="/delete/{id}", method=RequestMethod.GET)

public ModelAndView delete(@PathVariable("id") int id){

userService.deleteUser(id);

return new ModelAndView("redirect:/user/list");

}

}

## Creating JSP Views

Create ***user***folder under ***src\main\webapp\WEB-INF\jsp*** folder.   
Create ***user\_page.jsp*** file under ***src\main\webapp\WEB-INF\jsp\user*** folder and write the following code in it.

<%@ 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" %>

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

<!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>User Page</title>

</head>

<body>

<spring:url value="/user/add" var="addURL" />

<a href="${addURL }">Add User</a>

<h1>Users List</h1>

<table width="100%" border="1">

<tr>

<th>ID</th>

<th>Firstname</th>

<th>Lastname</th>

<th>Address</th>

<th colspan="2">Action</th>

</tr>

<c:forEach items="${listUser }" var="user" >

<tr>

<td>${user.id }</td>

<td>${user.firstname }</td>

<td>${user.lastname }</td>

<td>${user.address }</td>

<td>

<spring:url value="/user/update/${user.id }" var="updateURL" />

<a href="${updateURL }">Update</a>

</td>

<td>

<spring:url value="/user/delete/${user.id }" var="deleteURL" />

<a href="${deleteURL }">Delete</a>

</td>

</tr>

</c:forEach>

</table>

</body>

</html>

Create ***user\_form.jsp*** file under ***src\main\webapp\WEB-INF\jsp\user*** folder and write the following code in it.

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

pageEncoding="ISO-8859-1"%>

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

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

<!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>User Form</title>

</head>

<body>

<spring:url value="/user/save" var="saveURL" />

<form:form modelAttribute="userForm" method="post" action="${saveURL }" >

<form:hidden path="id"/>

<table>

<tr>

<td>First name: </td>

<td>

<form:input path="firstname"/>

</td>

</tr>

<tr>

<td>Last name: </td>

<td>

<form:input path="lastname"/>

</td>

</tr>

<tr>

<td>Address: </td>

<td>

<form:input path="address"/>

</td>

</tr>

<tr>

<td></td>

<td>

<button type="submit">Save</button>

</td>

</tr>

</table>

</form:form>

</body>

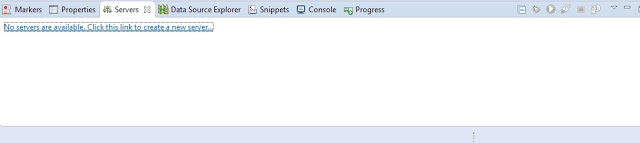
</html>

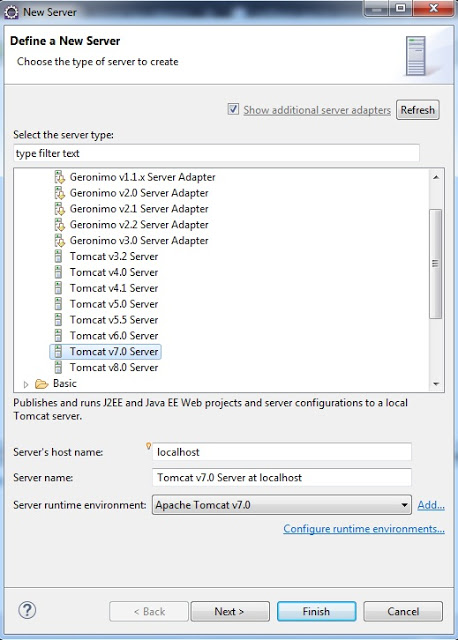
## Building

* Right click this project
* Select Run As -> Maven clean
* Right click this project
* Select Run As -> Maven install

## Configuring Apache Tomcat

* Under Servers tab, click link "No servers are available. Click this link to create a new server ...", select Apache tomcat 7
* Click Finish

[[](https://2.bp.blogspot.com/-htNsfXUYRZc/Wmiq-_D6jrI/AAAAAAAAAGI/GlTmdzCTvm4_nSloYawfDfZtveO5hbaHwCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2Beclipse.jpg)](https://2.bp.blogspot.com/-htNsfXUYRZc/Wmiq-_D6jrI/AAAAAAAAAGI/GlTmdzCTvm4_nSloYawfDfZtveO5hbaHwCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2Beclipse.jpg)

[](https://4.bp.blogspot.com/-E9eOOzbOQUY/WmirUbsBtkI/AAAAAAAAAGM/hpdZL9HUWlU9jK4i-SWWHxRg_4vNVI_0wCPcBGAYYCw/s1600/configuration%2Bapache%2Btomcat%2B7%2Beclipse.jpg)

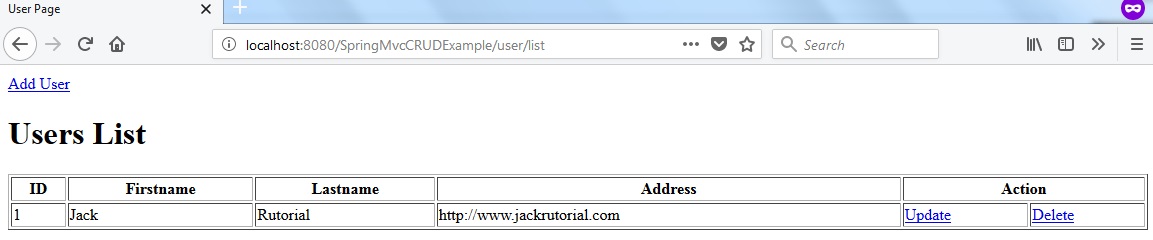
* Right click "Tomcat v7.0 Server at localhost [Stopped, Republish]", select "Add and Remove ..."
* Add ***SpringMvcCRUDExample***project, then Click Finish
* Open server.xml file under Servers Folder
* Find line

***<Context docBase="SpringMvcCRUDExample" path="/SpringMvcCRUDExample" reloadable="true" source="org.eclipse.jst.jee.server:SpringMvcCRUDExample" />***  
  
***Update its as below:***  
  
***<Context docBase="<Project Folder Location>\SpringMvcCRUDExample\target\SpringMvcCRUDExample-0.0.1-SNAPSHOT\" path="/SpringMvcCRUDExample" reloadable="true" >***  
***<Resource***  
***name="jdbc/springmvc"***  
***auth="Container"***  
***type="javax.sql.DataSource"***  
***username="root"***  
***password="root"***  
***driverClassName="com.mysql.jdbc.Driver"***  
***url="jdbc:mysql://localhost:3306/jack\_rutorial\_demo"***  
***maxActive="5"***  
***maxIdle="3"***  
***/>***  
***</Context>***  
  
  
**Watch video add Apache Tomcat Server in Eclipse IDE**

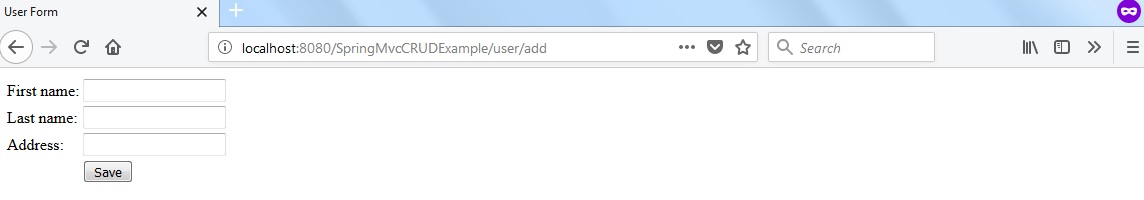
* Run application & Check result
* Start Apache Tomcat from Eclipse IDE.
* Type the following URLs in browser's address bar to open the user list from.

http://localhost:8080/SpringMvcCRUDExample/user/list

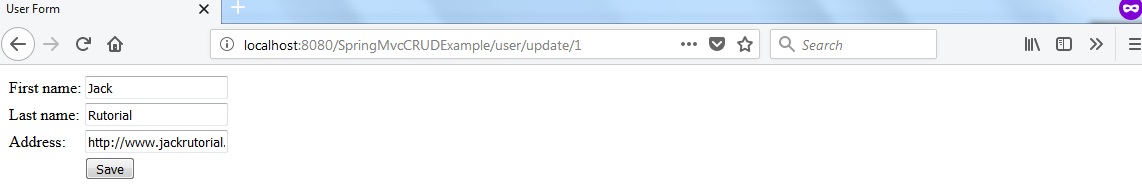
***Users List Screen:***

[](https://3.bp.blogspot.com/-0caw3pqUMmE/WnGaELUAJVI/AAAAAAAAAJo/6HQMVQUvoFAhCs7hJNJjmJ-KNJW8zBdewCLcBGAs/s1600/user_list_page.jpg)

***Users Add New Screen:***

[](https://1.bp.blogspot.com/-vOiy6EjhCMc/WnHc1RDHkqI/AAAAAAAAAJ0/sKUtAhEFleEdkn9IIsQbv1aRKQgM-qkCwCLcBGAs/s1600/add_user_page.jpg)

***Users Update Screen:***

[](https://3.bp.blogspot.com/-JZvzDJcZc_s/WnHdFq3HgwI/AAAAAAAAAJ4/XFPiR4H9J7wtidAeHvTU3Q6BYl6HkKefACLcBGAs/s1600/update_user_page.jpg)

# Spring Restful Web Services example JSON CRUD using Maven within the Eclipse IDE and MySQL Database Tutorial

In [Java](https://www.jackrutorial.com/search/label/java?max-results=6) [Java Tutorial](https://www.jackrutorial.com/search/label/java%20tutorial?max-results=6)Published On [February 05, 2018](https://www.jackrutorial.com/2018/02/spring-4-mvc-restful-web-services-json-crud-using-maven-within-the-eclipse-ide-and-mysql-database-tutorial.html)

## Overview

In this tutorial, we show you how to create a simple [Restful web services CRUD](https://www.jackrutorial.com/2018/01/building-hibernate-crud-restful.html) example in java using Spring MVC return json. We'll building a spring restful web services annotations using eclipse and tomcat step by step.  
  
***Follow the steps mentioned below to develop Restful web services.***

[](https://1.bp.blogspot.com/-EqrXe90UeiE/WsY68IfQ8MI/AAAAAAAAAns/3vbw0vtq71InFSPfP9wEmgO0NbkAHsBXwCLcBGAs/s1600/Spring-4-MVC-RESTFul-Web-Services-JSON-CRUD.jpg)

## Watch Tutorial

## MySQL database settings

Execute the following MySQL script in order to create a database named ***jack\_rutorial\_demo*** with a table named ***users***. 

CREATE DATABASE `jack\_rutorial\_demo`;

CREATE TABLE `users` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`firstname` varchar(45) NOT NULL DEFAULT '',

`lastname` varchar(100) NOT NULL DEFAULT '',

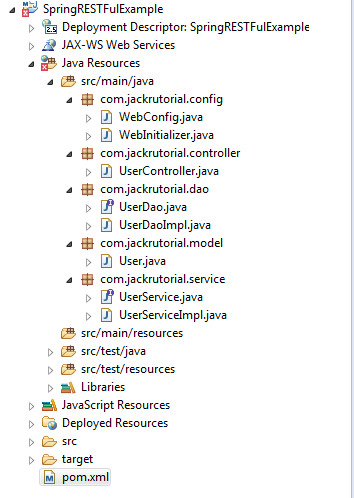
`address` varchar(100) NOT NULL DEFAULT '',

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

## Project Structure

The following screenshot shows final structure of the project.

[](https://4.bp.blogspot.com/-iUB2RLeQtWA/WnlXFf5BJuI/AAAAAAAAANo/UU77AEbG3mEcw1EJgsKG3r8Qzus0j7fEACLcBGAs/s1600/Project%2BStructure%2BSpring%2B4%2BRestful%2Bweb%2Bservices%2Bcrud%2Bjson%2Bexample.jpg)

## Create Spring Project in Eclipse

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "***com.jackrutorial***" in the "Group ID:" field
* Type "***SpringRESTFulExample***" in the "Artifact Id:" field
* Packaging -> War
* Click Finish.

## Maven Dependencies

We will update ***pom.xml*** file to add the required dependencies for the following dependencies.

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringRESTFulExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.7.0</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

## Spring Application Configuration

To configure Spring web MVC framework, we create class ***WebConfig***, class ***WebInitializer***in src folder with package name ***com.jackrutorial.config*** and write the following code in it.  
  
***WebConfig.java***

package com.jackrutorial.config;

import javax.naming.NamingException;

import javax.sql.DataSource;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jndi.JndiTemplate;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

@Configuration

@EnableWebMvc

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter {

@Autowired

DataSource dataSource;

@Bean

public NamedParameterJdbcTemplate geNamedParameterJdbcTemplate(){

return new NamedParameterJdbcTemplate(dataSource);

}

@Bean

public DataSource getDataSource() throws NamingException{

JndiTemplate jndiTemplate = new JndiTemplate();

DataSource dataSource = (DataSource) jndiTemplate.lookup("java:comp/env/jdbc/springmvc");

return dataSource;

}

}

***WebInitializer.java***

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

## Creating Model Layer

Create a ***User***class under ***com.jackrutorial.model*** and write the following code in it.  
  
***User.java***

package com.jackrutorial.model;

public class User {

private Integer id;

private String firstname;

private String lastname;

private String address;

public User() {

super();

}

public User(Integer id) {

super();

this.id = id;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

}

## Database Access Object Configuration

Create a ***UserDao***interface under ***com.jackrutorial.dao*** and write the following code in it.  
  
***UserDao.java***

package com.jackrutorial.dao;

import java.util.List;

import com.jackrutorial.model.User;

public interface UserDao {

public List listAllUser();

public void addUser(User user);

public void updateUser(User user);

public void delete(User user);

public User findUserById(User user);

}

Create a ***UserDaoImpl***class implements ***UserDao***Interface under ***com.jackrutorial.dao***package and write the following code in it.  
  
***UserDaoImpl.java***

package com.jackrutorial.dao;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;

import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import org.springframework.stereotype.Repository;

import com.jackrutorial.model.User;

@Repository

public class UserDaoImpl implements UserDao {

NamedParameterJdbcTemplate namedParameterJdbcTemplate;

@Autowired

public void setNamedParameterJdbcTemplate(NamedParameterJdbcTemplate namedParameterJdbcTemplate) {

this.namedParameterJdbcTemplate = namedParameterJdbcTemplate;

}

public List listAllUser() {

List list = new ArrayList();

String sql = "SELECT id, firstname, lastname, address FROM users";

list = namedParameterJdbcTemplate.query(sql, getSqlParameterByModel(null), new UserMapper());

return list;

}

private SqlParameterSource getSqlParameterByModel(User user){

MapSqlParameterSource parameterSource = new MapSqlParameterSource();

if(user != null){

parameterSource.addValue("id", user.getId());

parameterSource.addValue("firstname", user.getFirstname());

parameterSource.addValue("lastname", user.getLastname());

parameterSource.addValue("address", user.getAddress());

}

return parameterSource;

}

private static final class UserMapper implements RowMapper{

public User mapRow(ResultSet rs, int rowNum) throws SQLException {

User user = new User();

user.setId(rs.getInt("id"));

user.setFirstname(rs.getString("firstname"));

user.setLastname(rs.getString("lastname"));

user.setAddress(rs.getString("address"));

return user;

}

}

public void addUser(User user) {

String sql = "INSERT INTO users(firstname, lastname, address) VALUES(:firstname, :lastname, :address)";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(user));

}

public void updateUser(User user) {

String sql = "UPDATE users SET firstname=:firstname, lastname=:lastname, address=:address WHERE id =:id";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(user));

}

public void delete(User user) {

String sql = "DELETE FROM users WHERE id=:id";

namedParameterJdbcTemplate.update(sql, getSqlParameterByModel(user));

}

public User findUserById(User user) {

String sql = "SELECT \* FROM users WHERE id =:id";

return namedParameterJdbcTemplate.queryForObject(sql, getSqlParameterByModel(user), new UserMapper());

}

}

## User Service Layer Configuration

Create a ***UserService***interface under ***com.jackrutorial.service*** and write the following code in it.  
  
***UserService.java***

package com.jackrutorial.service;

import java.util.List;

import com.jackrutorial.model.User;

public interface UserService {

public List listAllUser();

public void addUser(User user);

public void updateUser(User user);

public void delete(User user);

public User findUserById(User user);

}

Create a ***UserServiceImpl***class implements ***UserService***Interface under ***com.jackrutorial.service*** package and write the following code in it.  
  
***UserServiceImpl.java***

package com.jackrutorial.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.jackrutorial.dao.UserDao;

import com.jackrutorial.model.User;

@Service

public class UserServiceImpl implements UserService {

UserDao userDao;

@Autowired

public void setUserDao(UserDao userDao) {

this.userDao = userDao;

}

public List listAllUser() {

return userDao.listAllUser();

}

public void addUser(User user) {

userDao.addUser(user);

}

public void updateUser(User user) {

userDao.updateUser(user);

}

public void delete(User user) {

userDao.delete(user);

}

public User findUserById(User user) {

return userDao.findUserById(user);

}

}

## Rest Controller Annotation Configuration

Create a ***UserController***class under ***com.jackrutorial.controller*** package and write the following code in it.  
  
***UserController.java***

package com.jackrutorial.controller;

import java.util.List;

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

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

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

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

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

import com.jackrutorial.model.User;

import com.jackrutorial.service.UserService;

@RestController

public class UserController {

@Autowired

UserService userService;

@RequestMapping(value="/user/", method = RequestMethod.GET, headers="Accept=application/json")

public ResponseEntity<List<User>> listAllUser(){

List<User> list = userService.listAllUser();

if(list.size() == 0){

return new ResponseEntity<List<User>>(HttpStatus.NO\_CONTENT);

}

return new ResponseEntity<List<User>>(list, HttpStatus.OK);

}

@RequestMapping(value="/add/", method = RequestMethod.POST, headers="Accept=application/json")

public ResponseEntity<Void> add(@RequestBody User user){

userService.addUser(user);

HttpHeaders headers = new HttpHeaders();

return new ResponseEntity<Void>(headers, HttpStatus.CREATED);

}

@RequestMapping(value="/update/{id}", method = RequestMethod.PUT, headers="Accept=application/json")

public ResponseEntity<Void> update(@PathVariable("id") int id, @RequestBody User user){

user.setId(id);

userService.updateUser(user);

HttpHeaders headers = new HttpHeaders();

return new ResponseEntity<Void>(headers, HttpStatus.OK);

}

@RequestMapping(value="/delete/{id}", method = RequestMethod.DELETE, headers="Accept=application/json")

public ResponseEntity<Void> delete(@PathVariable("id") int id, @RequestBody User user){

user.setId(id);

userService.delete(user);

HttpHeaders headers = new HttpHeaders();

return new ResponseEntity<Void>(headers, HttpStatus.NO\_CONTENT);

}

}

## Building Spring Restful web services

* Right click this project
* Select Run As -> Maven clean
* Right click this project
* Select Run As -> Maven install

## How to deploy Spring Restful Web Service in Tomcat

*Watch video add Apache Tomcat Server in Eclipse.*

Enter the following to the ***Host***node in the ***TOMCAT\_HOME/conf/server.xml***.  
  
*<Context docBase="****<Project Folder Location>\SpringRESTFulExample\target\SpringRESTFulExample-0.0.1-SNAPSHOT\****" path="/****SpringRESTFulExample****" reloadable="true" >*  
*<Resource*  
*name="jdbc/springmvc"*  
*auth="Container"*  
*type="javax.sql.DataSource"*  
*username="root"*  
*password="root"*  
*driverClassName="com.mysql.jdbc.Driver"*  
*url="jdbc:mysql://localhost:3306/jack\_rutorial\_demo"*  
*maxActive="5"*  
*maxIdle="3"*  
*/>*  
*</Context>*

* ***Start Apache Tomcat from Eclipse Oxygen java***.

## How to test Spring MVC Restful web services using postman

***Add User API***  
Request Method: POST  
URL: <http://localhost:8080/SpringRESTFulExample/add/>  
Body: Type application/json

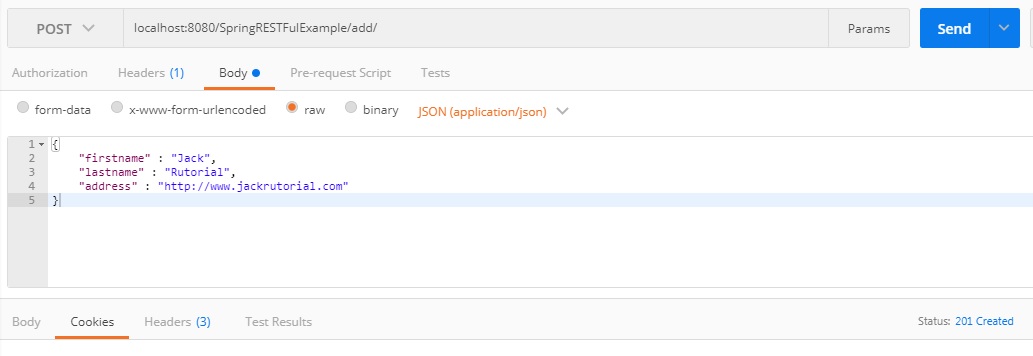
{

"firstname" : "Jack",

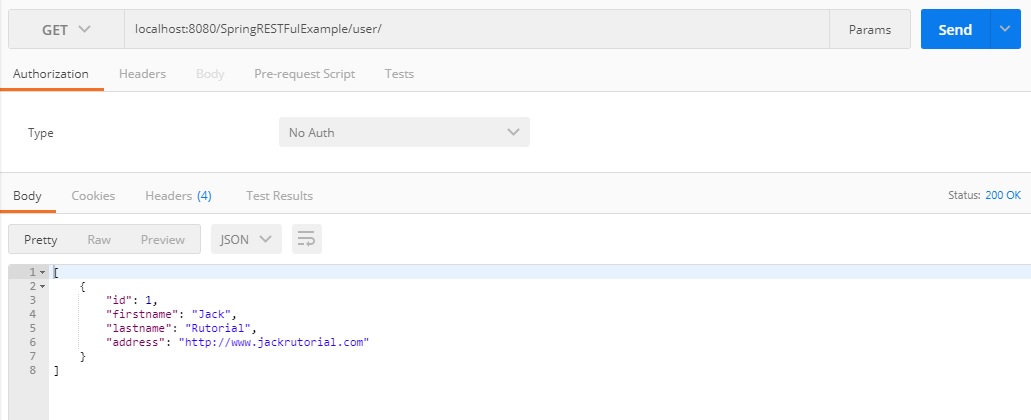
"lastname" : "Rutorial",

"address" : "http://www.jackrutorial.com"

}

[](https://3.bp.blogspot.com/-w23uIWuZJV4/WnlcP_d4gYI/AAAAAAAAAOA/dDj9km057xghd_H8u29NOOe5ZX5KgJOLACLcBGAs/s1600/Add%2BUser%2Bpostman%2B-%2BSpring%2B4%2BRestful%2BWeb%2BServices%2BCRUD%2BJSON%2BExample.jpg)

***GET User API***  
Request Method: GET  
URL: <http://localhost:8080/SpringRESTFulExample/user/>

[](https://4.bp.blogspot.com/-73dv84K85kI/WnlcfXpjdtI/AAAAAAAAAOE/zgC_JtAtHjMqvDLhpjzQJHxDmzTgy7AbwCLcBGAs/s1600/List%2BUser%2Bpostman%2B-%2BSpring%2B4%2BRestful%2BWeb%2BServices%2BCRUD%2BJSON%2BExample.jpg)

***Update User API***  
Request Method: PUT  
URL: <http://localhost:8080/SpringRESTFulExample/update/1>  
Body: Type application/json

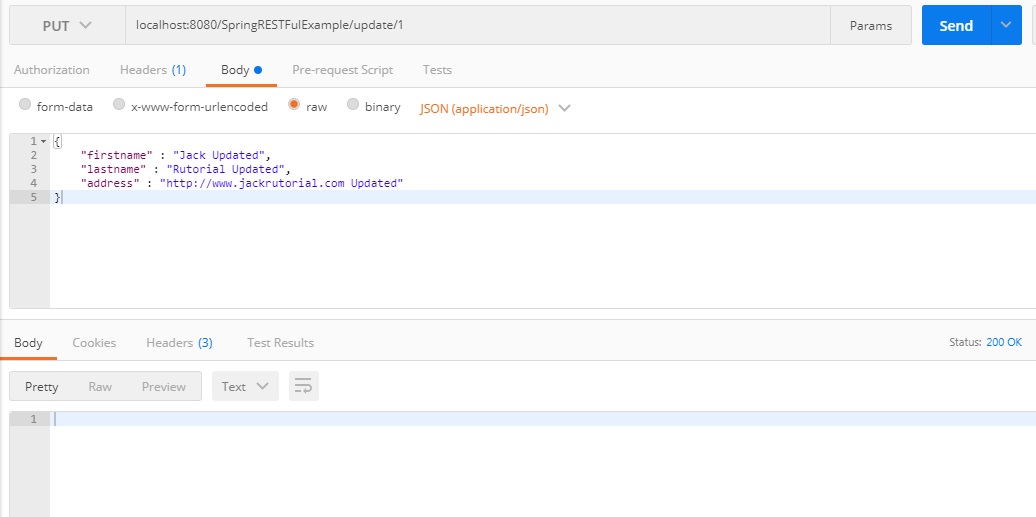
{

"firstname" : "Jack Updated",

"lastname" : "Rutorial Updated",

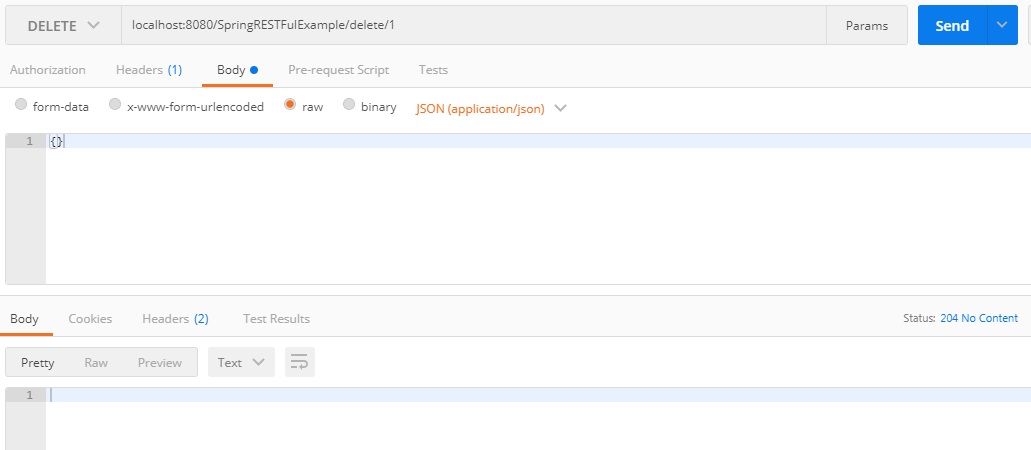
"address" : "http://www.jackrutorial.com Updated"

}

[](https://3.bp.blogspot.com/-08xLazRj06k/Wnlcp34ziWI/AAAAAAAAAOI/5rY3TiKkk48YILW_vklmi_MQ2dFH-0m2ACLcBGAs/s1600/Update%2BUser%2Bpostman%2B-%2BSpring%2B4%2BRestful%2BWeb%2BServices%2BCRUD%2BJSON%2BExample.jpg)

***Delete User API***  
Request Method: DELETE  
URL: <http://localhost:8080/SpringRESTFulExample/delete/1>  
Body: Type application/json

{}

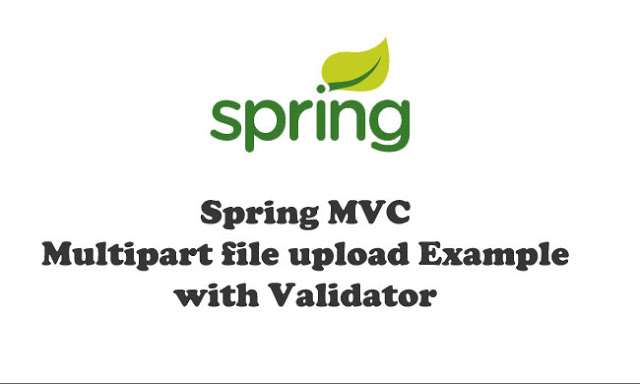
[](https://3.bp.blogspot.com/-8BQcgCl_rJw/Wnlc1-UG_kI/AAAAAAAAAOQ/VlOfwFv09RwQVFB0gghSwT3aVzRxQh6LQCLcBGAs/s1600/Delete%2BUser%2Bpostman%2B-%2BSpring%2B4%2BRestful%2BWeb%2BServices%2BCRUD%2BJSON%2BExample.jpg)

# Spring MVC Multipart file upload Example with Validator - Spring MVC Tutorial

In [File Upload](https://www.jackrutorial.com/search/label/file%20upload?max-results=6) [Java](https://www.jackrutorial.com/search/label/java?max-results=6)Published On [May 17, 2018](https://www.jackrutorial.com/2018/05/spring-mvc-multipart-file-upload-example.html)

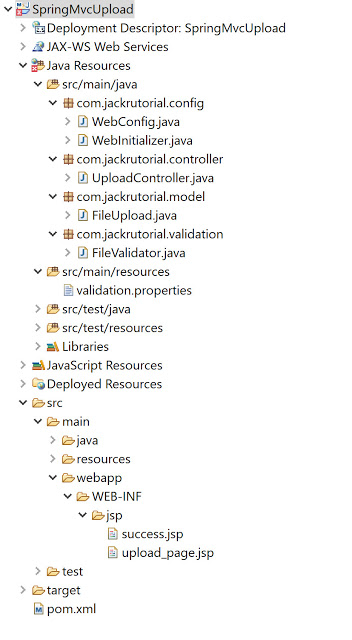
## Overview

In this tutorial, we show you how to create ***Multiple File Uploads*** in [Spring MVC](https://www.jackrutorial.com/search/label/spring%20mvc%20tutorial) and validating it. We use the ***commons-fileupload 1.3.1*** lib with method ***CommonsMultipartFile*** to upload multiple files and the ***validation-api 1.1.0.Final*** lib to validate the uploaded files.  
***Follow the steps mentioned below to buid this example***

[](https://3.bp.blogspot.com/-CuRm1osV0fs/Wv24Er-eeCI/AAAAAAAAA7w/e4wGm8NXA6UO8I3OgoxNePlFr-cRUwTaACLcBGAs/s1600/Spring-MVC-Multipart-file-upload-Example-with-Validator.jpg)

## Watch video Tutorials

## Project Structure

[](https://1.bp.blogspot.com/-89UTKg_MuvA/Wv2e4K3-WRI/AAAAAAAAA64/1BFeclwQqKYKEIfcYeh2J9MVGsi5VHGPACLcBGAs/s1600/Project%2BStructure%2BSpring%2BMVC%2BMultipart%2Bfile%2Bupload%2BExample.jpg)

## How to Create the Maven Project

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "com.jackrutorial" in the "Group ID:" field
* Type "SpringMvcUpload" in the "Artifact Id:" field
* Packaging -> War
* Click Finish.

## Maven Dependencies

We specify the dependency for the ***spring-webmvc version 4.3.0.RELEASE***, ***jstl v1.2*** and ***javax.servlet-api 3.1.0*** to [build Web MVC](https://www.jackrutorial.com/2018/01/spring-mvc-crud-operations-with-spring-jdbc-template-and-mysql-database-tutorial.html). We also include the ***commons-fileupload v1.3.1*** which is used to upload a MultipartFile and the ***validation-api v1.1.0.Final*** to create a Validator.  
The updated ***pom.xml*** file will have the following code: 

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringMvcUpload</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>commons-fileupload</groupId>

<artifactId>commons-fileupload</artifactId>

<version>1.3.1</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.1.0.Final</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

## Configure Web Application

Create a WebConfig class under com.jackrutorial.config package and write the following code in it. 

package com.jackrutorial.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.support.ResourceBundleMessageSource;

import org.springframework.web.multipart.commons.CommonsMultipartResolver;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc

@Configuration

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter {

@Bean

public InternalResourceViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/jsp/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

@Bean

public ResourceBundleMessageSource messageSource() {

ResourceBundleMessageSource rb = new ResourceBundleMessageSource();

rb.setBasenames(new String[] { "validation" });

return rb;

}

@Bean(name="multipartResolver")

public CommonsMultipartResolver getResolver() {

CommonsMultipartResolver commonsMultipartResolver = new CommonsMultipartResolver();

//set max upload size per file is 20mb

commonsMultipartResolver.setMaxUploadSizePerFile(20\*1024\*1024);

return commonsMultipartResolver;

}

}

The ***CommonsMultipartResolver***saves temporary files to the servlet container’s temporary directory. We restrict you to a maximum 20 MB upload file size by set commonsMultipartResolver.setMaxUploadSizePerFile(20\*1024\*1024);  
Create a ***WebInitializer***class extends ***AbstractAnnotationConfigDispatcherServletInitializer*** under com.jackrutorial.config package and write the following code in it.  
***WebInitializer.java***

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class<?>[] getServletConfigClasses() {

// TODO Auto-generated method stub

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

## Model

Create a FileUpload class under com.jackrutorial.model package and write the following code in it. 

package com.jackrutorial.model;

import org.springframework.web.multipart.commons.CommonsMultipartFile;

public class FileUpload {

private CommonsMultipartFile[] files;

public CommonsMultipartFile[] getFiles() {

return files;

}

public void setFiles(CommonsMultipartFile[] files) {

this.files = files;

}

}

## Validator

Create a @Component FileValidator implements the ***org.springframework.validation.Validator***, overrides the two methods supports and validate as follow below. 

package com.jackrutorial.validation;

import org.springframework.stereotype.Component;

import org.springframework.validation.Errors;

import org.springframework.validation.Validator;

import org.springframework.web.multipart.commons.CommonsMultipartFile;

import com.jackrutorial.model.FileUpload;

@Component

public class FileValidator implements Validator {

public boolean supports(Class<?> clazz) {

return FileUpload.class.isAssignableFrom(clazz);

}

public void validate(Object target, Errors errors) {

FileUpload fileUpload = (FileUpload) target;

CommonsMultipartFile[] commonsMultipartFiles = fileUpload.getFiles();

for(CommonsMultipartFile multipartFile : commonsMultipartFiles) {

if(multipartFile.getSize() == 0) {

errors.rejectValue("files", "missing.file");

}

}

}

}

The validator above is used to validate if the uploaded files are not empty. If the files are empty, we reject the value and show the ***"No file choose"*** message. 

## Validator Messages

Create the validation.properties under the ***src/main/resources/*** folder. 

missing.file=No file choose

## Upload Controller

Create a UploadController class under com.jackrutorial.controller package and write the following code in it. 

package com.jackrutorial.controller;

import java.io.File;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

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

import org.springframework.stereotype.Controller;

import org.springframework.util.FileCopyUtils;

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.multipart.commons.CommonsMultipartFile;

import org.springframework.web.servlet.ModelAndView;

import com.jackrutorial.model.FileUpload;

import com.jackrutorial.validation.FileValidator;

@Controller

@RequestMapping(value="/")

public class UploadController {

@Autowired

FileValidator fileValidator;

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

public ModelAndView uploadPage() {

ModelAndView model = new ModelAndView("upload\_page");

model.addObject("formUpload", new FileUpload());

return model;

}

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

public ModelAndView upload(@ModelAttribute("formUpload") FileUpload fileUpload, BindingResult result) throws IOException {

//validate file upload

fileValidator.validate(fileUpload, result);

if(result.hasErrors()) {

return new ModelAndView("upload\_page");

}

return new ModelAndView("success", "fileNames", processUpload(fileUpload));

}

private List<String> processUpload(FileUpload files) throws IOException{

List<String> fileNames = new ArrayList<String>();

CommonsMultipartFile[] commonsMultipartFiles = files.getFiles();

for(CommonsMultipartFile multipartFile : commonsMultipartFiles) {

FileCopyUtils.copy(multipartFile.getBytes(), new File("C:\\upload\\" + multipartFile.getOriginalFilename()));

fileNames.add(multipartFile.getOriginalFilename());

}

return fileNames;

}

}

## Views

Create ***upload\_page.jsp*** file under ***src\main\webapp\WEB-INF\jsp\*** folder and write the following code in it.  
***upload\_page.jsp*** 

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

pageEncoding="ISO-8859-1"%>

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

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

<!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>Upload Page</title>

</head>

<body>

<spring:url value="/upload" var="uploadURL" />

<form:form modelAttribute="formUpload" method="post" action="${uploadURL }" enctype="multipart/form-data" >

<form:input path="files" type="file" multiple="multiple" />

<form:errors path="files" cssStyle="color: red" />

<button type="submit">Upload</button>

</form:form>

</body>

</html>

Create success.jsp file under ***src\main\webapp\WEB-INF\jsp\*** folder and write the following code in it.  
***success.jsp*** 

<%@ 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" %>

<!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>Success</title>

</head>

<body>

<c:forEach items="${fileNames }" var="fileName">

File <b>${fileName }</b> uploaded sucessfully<br/>

</c:forEach>

</body>

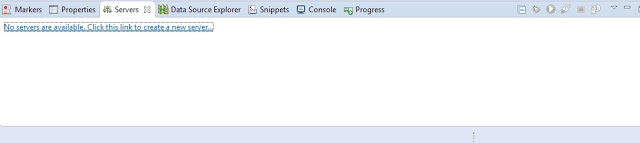
</html>

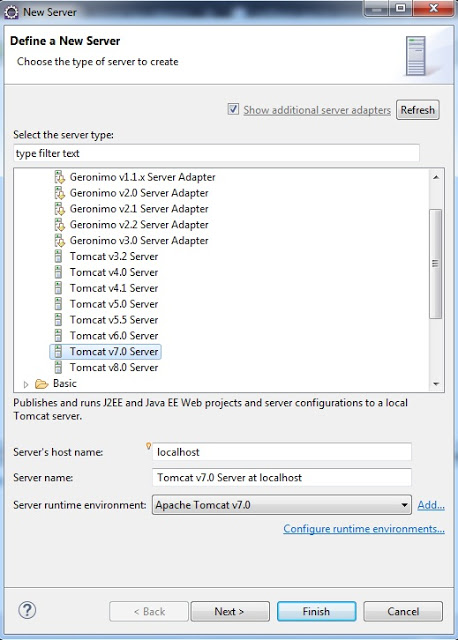
## Buid and Deploy

* Right click this project
* Select Run As -> Maven clean
* Right click this project
* Select Run As -> Maven install

## Configuring Apache Tomcat 7

* Under Servers tab, click link "No servers are available. Click this link to create a new server ...", select Apache tomcat 7
* Click Finish

[](https://2.bp.blogspot.com/-htNsfXUYRZc/Wmiq-_D6jrI/AAAAAAAAAGI/GlTmdzCTvm4_nSloYawfDfZtveO5hbaHwCPcBGAYYCw/s640/configuration%2Bapache%2Btomcat%2Beclipse.jpg)

[](https://4.bp.blogspot.com/-E9eOOzbOQUY/WmirUbsBtkI/AAAAAAAAAGM/hpdZL9HUWlU9jK4i-SWWHxRg_4vNVI_0wCPcBGAYYCw/s640/configuration%2Bapache%2Btomcat%2B7%2Beclipse.jpg)

* Right click "Tomcat v7.0 Server at localhost [Stopped, Republish]", select "Add and Remove ..."
* Add SpringMvcUpload project, then Click Finish
* Open server.xml file under Servers Folder
* Find line

<Context docBase="{WORK\_LOCATION}\SpringMvcUpload\target\SpringMvcUpload-0.0.1-SNAPSHOT\" path="/SpringMvcUpload" reloadable="true" source="org.eclipse.jst.jee.server:SpringMvcCRUDExample" />

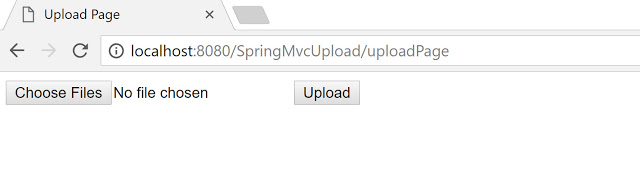
Update its as below: 

<Context docBase="{WORK\_LOCATION}\SpringMvcUpload\target\SpringMvcUpload-0.0.1-SNAPSHOT\" path="/SpringMvcUpload" reloadable="true" />

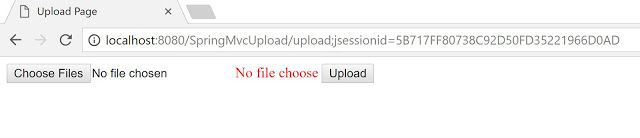
## Run application & Check result

Start Apache Tomcat from Eclipse IDE.  
Type the following URLs in browser's address bar to open the upload page from. 

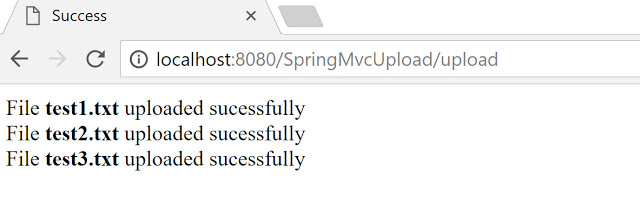
http://localhost:8080/SpringMvcUpload/uploadPage

[](https://2.bp.blogspot.com/-wD5yrEwgDcQ/Wv2zFpBbvrI/AAAAAAAAA7M/QXhtVA1vwWgnZco6rBGCvY9kg9fHscL8QCLcBGAs/s1600/upload%2Bfiles%2Bpage.jpg)

No file choose

[](https://1.bp.blogspot.com/-a9EC6Lk9SgQ/Wv2zZ_j5VYI/AAAAAAAAA7U/1O1VwZxofLIuPJhT4iUa0_j28Knyo0AbgCLcBGAs/s1600/upload%2Bfiles%2Bpage%2Berror.jpg)

Files uploaded sucessfully

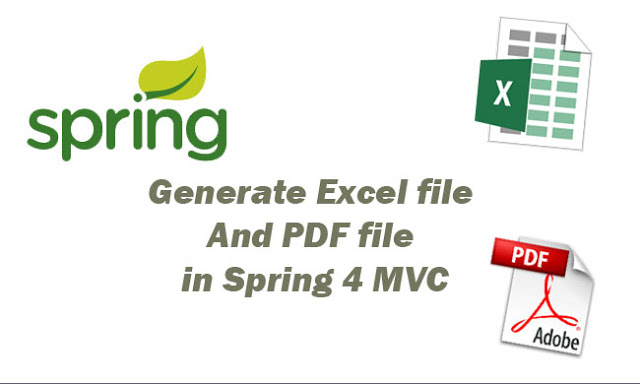
[](https://1.bp.blogspot.com/-Y5Nwt7kRF_Y/Wv2z8iTn7EI/AAAAAAAAA7c/VAKmVa0fyxsKpd8k8qI4j30-wu3nwu1fQCLcBGAs/s1600/upload%2Bfiles%2Bpage%2Bsuccessfully.jpg)

# Generate Excel file And PDF file in Spring 4 MVC Using Apache POI, iText Pdf Tutorial

In [Apache Poi](https://www.jackrutorial.com/search/label/apache%20poi?max-results=6) [Export Excel File](https://www.jackrutorial.com/search/label/export%20excel%20file?max-results=6)Published On [February 13, 2018](https://www.jackrutorial.com/2018/02/generate-excel-file-and-pdf-file-in-spring-4-mvc-using-apache-poi-itext-pdf-tutorial.html)

## Overview

In this tutorial, we show you how to create a simple with Spring 4 MVC to generate Excel file And PDF file with Eclipse Maven, apache poi, itex.

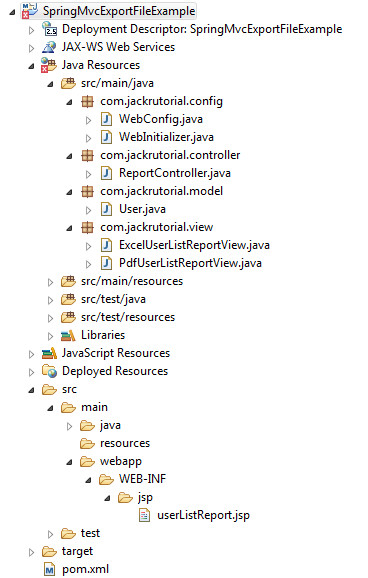
[](https://2.bp.blogspot.com/-m3TEKP3om58/WsObphtQUGI/AAAAAAAAAm8/pT_d-Unwcckqgw1ggtjSlK0TNnD0W5NMwCLcBGAs/s1600/Generate-Excel-file-And-PDF-file-in-Spring-4-MVC.jpg)

***Follow the steps mentioned below to develop this application***

## Watch Tutorial

## Project Structure

The following screenshot shows final structure of the project.

[](https://1.bp.blogspot.com/-EN-CbODqkow/WoKw7SfCYwI/AAAAAAAAAPI/VnS8kJB6UqMXmRy92Po6W7GuyMn40JDbgCLcBGAs/s1600/Project%2BStructure%2BSpring%2BMvc%2BExport%2BFile%2BExample.jpg)

## Create Maven Project

* Launch Eclipse IDE.
* Go to File-> New-> Others... Select Maven Project under Maven category then click Next.
* In New Maven Project wizard, select "Create a simpel project(skip archetype selection)" and click on Next
* In next wizard, type "***com.jackrutorial***" in the "Group ID:" field
* Type "***SpringMvcExportFileExample***" in the "Artifact Id:" field
* Packaging -> War
* Click Finish

## Maven Dependencies

We specify the dependency for the Spring WebMVC,JSTL, Servlet API, Apache POI, itext . The rest dependencies will be automatically resolved by Maven. The updated ***pom.xml***file will have the following code:

<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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.jackrutorial</groupId>

<artifactId>SpringMvcExportFileExample</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.0.RELEASE</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.apache.poi</groupId>

<artifactId>poi</artifactId>

<version>3.15</version>

</dependency>

<dependency>

<groupId>com.lowagie</groupId>

<artifactId>itext</artifactId>

<version>2.1.7</version>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>2.4</version>

<configuration>

<warSourceDirectory>src/main/webapp</warSourceDirectory>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

## Configure Web Application

***Create the com.jackrutorial.config package.***

* Right-click on src/main/java folder, New -> Package
* Enter  the package name as: "***com.jackrutorial.config***"
* Click Finish.

Create a ***WebConfig***class under ***com.jackrutorial.config*** package and write the following code in it.

package com.jackrutorial.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.config.annotation.EnableWebMvc;

import org.springframework.web.servlet.config.annotation.WebMvcConfigurerAdapter;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

import org.springframework.web.servlet.view.JstlView;

@EnableWebMvc

@Configuration

@ComponentScan(basePackages = { "com.jackrutorial" })

public class WebConfig extends WebMvcConfigurerAdapter{

@Bean

public InternalResourceViewResolver viewResolver(){

InternalResourceViewResolver viewResolver = new InternalResourceViewResolver();

viewResolver.setViewClass(JstlView.class);

viewResolver.setPrefix("/WEB-INF/jsp/");

viewResolver.setSuffix(".jsp");

return viewResolver;

}

}

Create a ***WebInitializer***class under ***com.jackrutorial.config*** package and write the following code in it.

package com.jackrutorial.config;

import org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherServletInitializer;

public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class[] getRootConfigClasses() {

return new Class[] { WebConfig.class};

}

@Override

protected Class[] getServletConfigClasses() {

return null;

}

@Override

protected String[] getServletMappings() {

return new String[] { "/" };

}

}

## Creating Model Layer

***Create the com.jackrutorial.model package.***

* Right-click on src/main/java folder, New -> Package
* Enter  the package name as: "***com.jackrutorial.model***"
* Click Finish.

Create a ***User***class under package ***com.jackrutorial.model*** and write the following code in it.

package com.jackrutorial.model;

public class User {

private Integer id;

private String username;

private String firstname;

private String lastname;

public User() {

super();

}

public User(Integer id, String username, String firstname, String lastname) {

super();

this.id = id;

this.username = username;

this.firstname = firstname;

this.lastname = lastname;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getFirstname() {

return firstname;

}

public void setFirstname(String firstname) {

this.firstname = firstname;

}

public String getLastname() {

return lastname;

}

public void setLastname(String lastname) {

this.lastname = lastname;

}

}

## Creating View Layer

***Create the com.jackrutorial.view package.***

* Right-click on src/main/java folder, New -> Package
* Enter  the package name as: "***com.jackrutorial.view***"
* Click Finish.

Create a ***ExcelUserListReportView***class under package ***com.jackrutorial.view*** and write the following code in it.

package com.jackrutorial.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.apache.poi.ss.usermodel.Row;

import org.apache.poi.ss.usermodel.Sheet;

import org.apache.poi.ss.usermodel.Workbook;

import org.springframework.web.servlet.view.document.AbstractXlsView;

import com.jackrutorial.model.User;

public class ExcelUserListReportView extends AbstractXlsView {

@Override

protected void buildExcelDocument(Map<String, Object> model, Workbook workbook, HttpServletRequest request,

HttpServletResponse response) throws Exception {

response.setHeader("Content-disposition", "attachment; filename=\"user\_list.xls\"");

@SuppressWarnings("unchecked")

List<User> list = (List<User>) model.get("userList");

Sheet sheet = workbook.createSheet("User List");

Row header = sheet.createRow(0);

header.createCell(0).setCellValue("ID");

header.createCell(1).setCellValue("USERNAME");

header.createCell(2).setCellValue("FIRST NAME");

header.createCell(3).setCellValue("LAST NAME");

int rowNum = 1;

for(User user : list){

Row row = sheet.createRow(rowNum++);

row.createCell(0).setCellValue(user.getId());

row.createCell(1).setCellValue(user.getUsername());

row.createCell(2).setCellValue(user.getFirstname());

row.createCell(3).setCellValue(user.getLastname());

}

}

}

Create a ***PdfUserListReportView***class under package***com.jackrutorial.view*** and write the following code in it.

package com.jackrutorial.view;

import java.util.List;

import java.util.Map;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.jackrutorial.model.User;

import org.springframework.web.servlet.view.document.AbstractPdfView;

import com.lowagie.text.Document;

import com.lowagie.text.Table;

import com.lowagie.text.pdf.PdfWriter;

public class PdfUserListReportView extends AbstractPdfView {

@Override

protected void buildPdfDocument(Map<String, Object> model, Document document, PdfWriter writer, HttpServletRequest request,

HttpServletResponse response) throws Exception {

response.setHeader("Content-Disposition", "attachment; filename=\"user\_list.pdf\"");

@SuppressWarnings("unchecked")

List<User> list = (List<User>) model.get("userList");

Table table = new Table(4);

table.addCell("ID");

table.addCell("USERNAME");

table.addCell("FIRST NAME");

table.addCell("LAST NAME");

for(User user : list){

table.addCell(String.valueOf(user.getId()));

table.addCell(user.getUsername());

table.addCell(user.getFirstname());

table.addCell(user.getLastname());

}

document.add(table);

}

}

## Creating Controller Layer

Create a ***ReportController***under ***com.jackrutorial.controller*** package and write the following code in it.

package com.jackrutorial.controller;

import java.util.ArrayList;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.stereotype.Controller;

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

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

import org.springframework.web.servlet.ModelAndView;

import com.jackrutorial.model.User;

import com.jackrutorial.view.ExcelUserListReportView;

import com.jackrutorial.view.PdfUserListReportView;

@Controller

@RequestMapping(value="/")

public class ReportController {

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

public ModelAndView userListReport(HttpServletRequest req, HttpServletResponse res){

String typeReport = req.getParameter("type");

List<User> list = new ArrayList<User>();

list.add(new User(1, "username 1", "First name 1", "Last name 1"));

list.add(new User(2, "username 2", "First name 2", "Last name 2"));

list.add(new User(3, "username 3", "First name 3", "Last name 3"));

list.add(new User(4, "username 4", "First name 4", "Last name 4"));

list.add(new User(5, "username 5", "First name 5", "Last name 5"));

if(typeReport != null && typeReport.equals("xls")){

return new ModelAndView(new ExcelUserListReportView(), "userList", list);

} else if(typeReport != null && typeReport.equals("pdf")){

return new ModelAndView(new PdfUserListReportView(), "userList", list);

}

return new ModelAndView("userListReport", "userList", list);

}

}

## Creating JSP Views

***Create jsp folder under src\main\webapp\WEB-INF\ folder.***  
Create ***userListReport.jsp*** file under ***src\main\webapp\WEB-INF\jsp\*** folder and write the following code in it.

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

pageEncoding="ISO-8859-1"%>

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

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

<!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>Report</title>

</head>

<body>

<table border="1" >

<thead>

<tr>

<td>ID</td>

<td>USERNAME</td>

<td>FIRST NAME</td>

<td>LAST NAME</td>

</tr>

</thead>

<tbody>

<c:forEach items="${userList }" var="user" >

<tr>

<td>${user.id }</td>

<td>${user.username }</td>

<td>${user.firstname }</td>

<td>${user.lastname }</td>

</tr>

</c:forEach>

</tbody>

</table>

<spring:url value="/report/?type=xls" var="xlsURL" />

<spring:url value="/report/?type=pdf" var="pdfURL" />

<a href="${xlsURL }">Download Excel</a>

<a href="${pdfURL }">Download PDF</a>

</body>

</html>

## Configuring Apache Tomcat

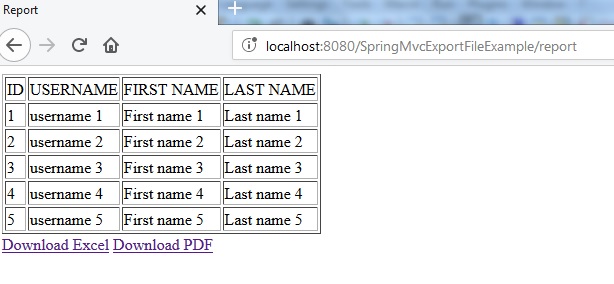
***Watch video add Apache Tomcat Server in Eclipse IDE.***

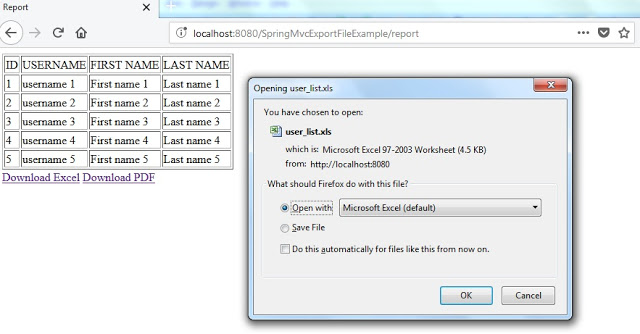
Enter the following to the ***Host***node in the ***TOMCAT\_HOME/conf/server.xml***.  
  
***<Context docBase="[PROJECT\_LOCATION]\SpringMvcExportFileExample\target\SpringMvcExportFileExample-0.0.1-SNAPSHOT\" path="/SpringMvcExportFileExample" reloadable="true" >***

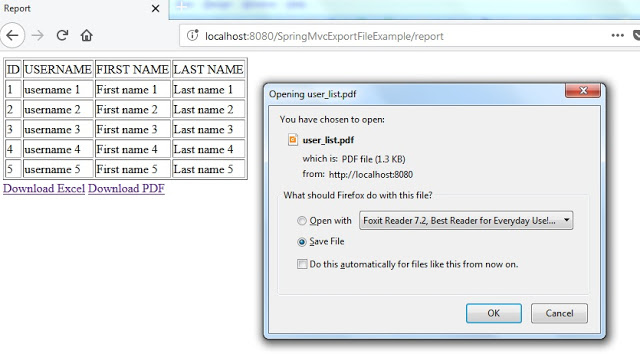
## Run & Check result

* Start Apache Tomcat from Eclipse IDE.
* Type the following URLs in browser's address bar to open the report page.

http://localhost:8080/SpringMvcExportFileExample/report

[[](https://3.bp.blogspot.com/-uE0HjP378oo/WoKyBm3zVHI/AAAAAAAAAPU/h0bHEq67wVwEZwcT9k12g8MKhHgqDrWDQCLcBGAs/s1600/report_page.jpg)](https://3.bp.blogspot.com/-uE0HjP378oo/WoKyBm3zVHI/AAAAAAAAAPU/h0bHEq67wVwEZwcT9k12g8MKhHgqDrWDQCLcBGAs/s1600/report_page.jpg)

[](https://2.bp.blogspot.com/-C0sMquD0cyQ/WoKyKwtvFwI/AAAAAAAAAPY/1X2ENox-UPY0Ub2MGdAu3sjinXexLIePgCLcBGAs/s1600/report_excel.jpg)

[](https://4.bp.blogspot.com/-K07VYUhZexE/WoKyRNqB-RI/AAAAAAAAAPc/jLtzExxYdAw2s6tgJwqk-foA2YlRx4s1wCLcBGAs/s1600/report_pdf.jpg)