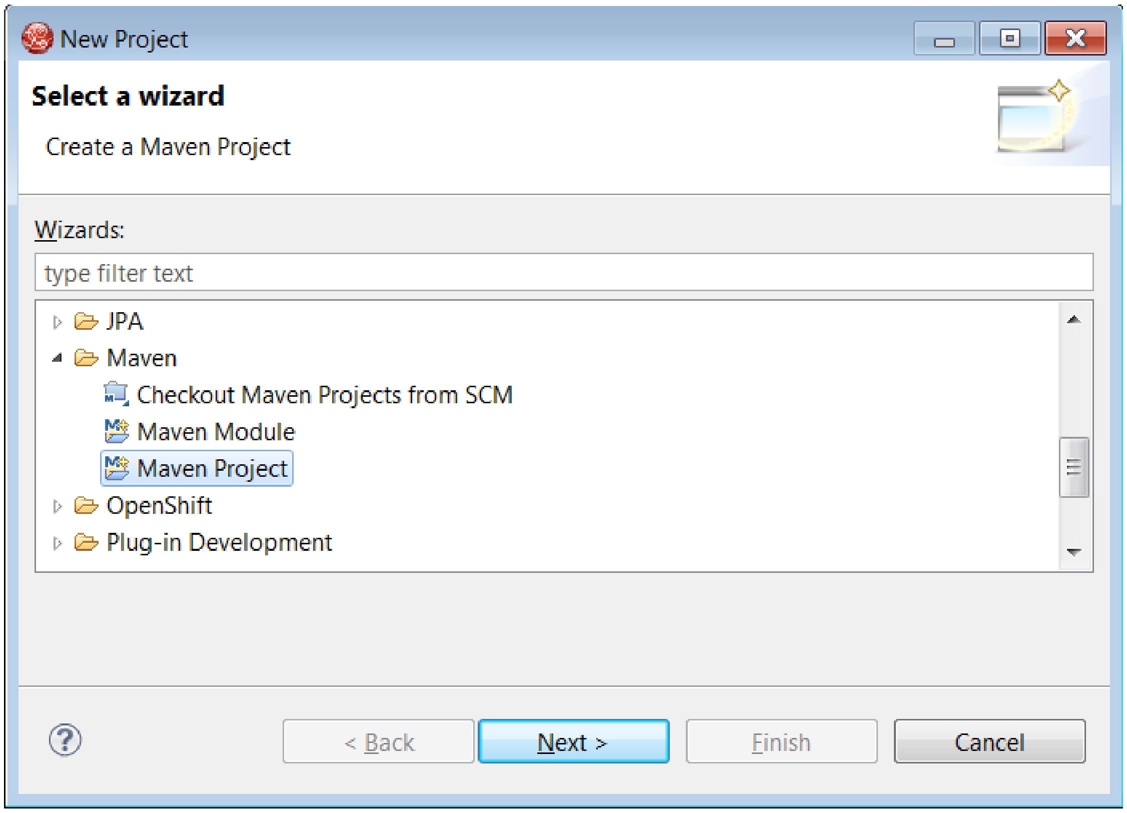
Spring MVC Project Creation

1. Prerequisites

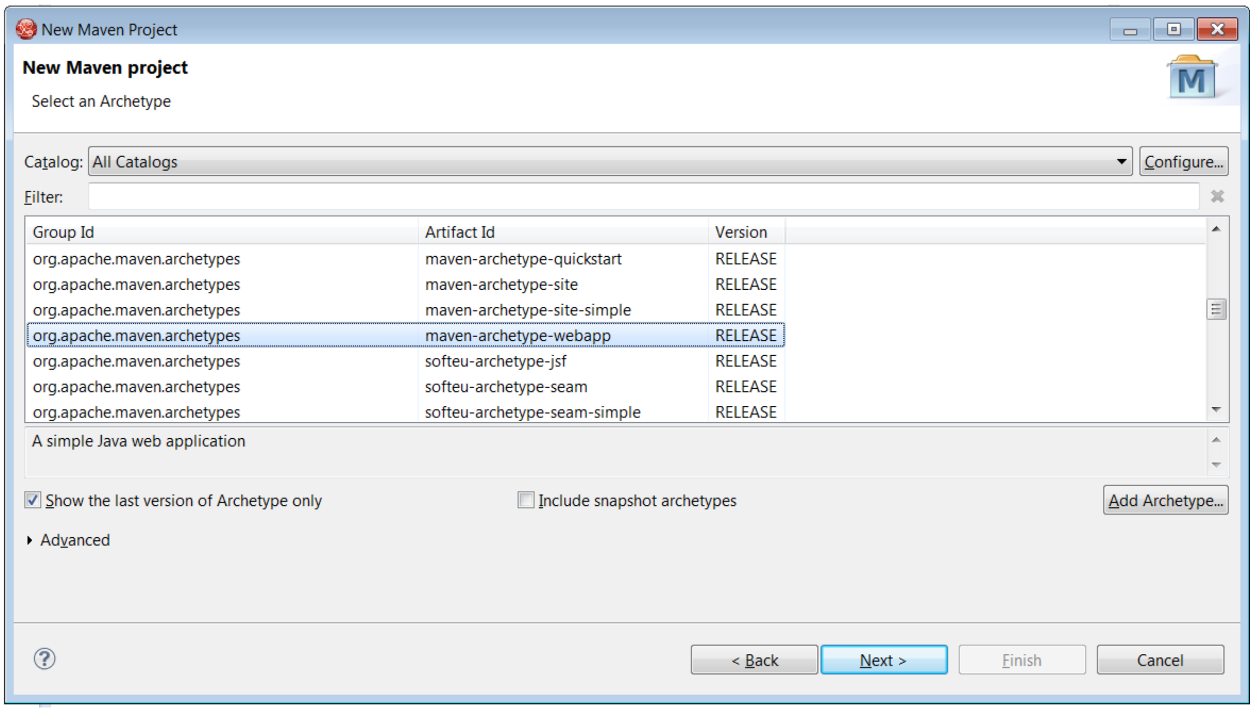
- Java version 1.7
- JBoss Developer Studio
- Apache Tomcat 7.0.x

2. Create a Maven Project

- **File** -> **New** -> **Other**… It will open the eclipse select wizard.



- **Select Maven -> Maven Project and click Next.**
- **Again click Next on New Maven Project dialog.**
- **Select an Archetype either by filtering webapp or selecting from the available artifacts and click Next.**


New Maven Project

New Maven project

Select an Archetype

Catalog: All Catalogs

Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-quickstart	RELEASE
org.apache.maven.archetypes	maven-archetype-site	RELEASE
org.apache.maven.archetypes	maven-archetype-site-simple	RELEASE
org.apache.maven.archetypes	maven-archetype-webapp	RELEASE
org.apache.maven.archetypes	softeu-archetype-jsf	RELEASE
org.apache.maven.archetypes	softeu-archetype-seam	RELEASE
org.apache.maven.archetypes	softeu-archetype-seam-simple	RELEASE

A simple Java web application

[✓] Show the last version of Archetype only

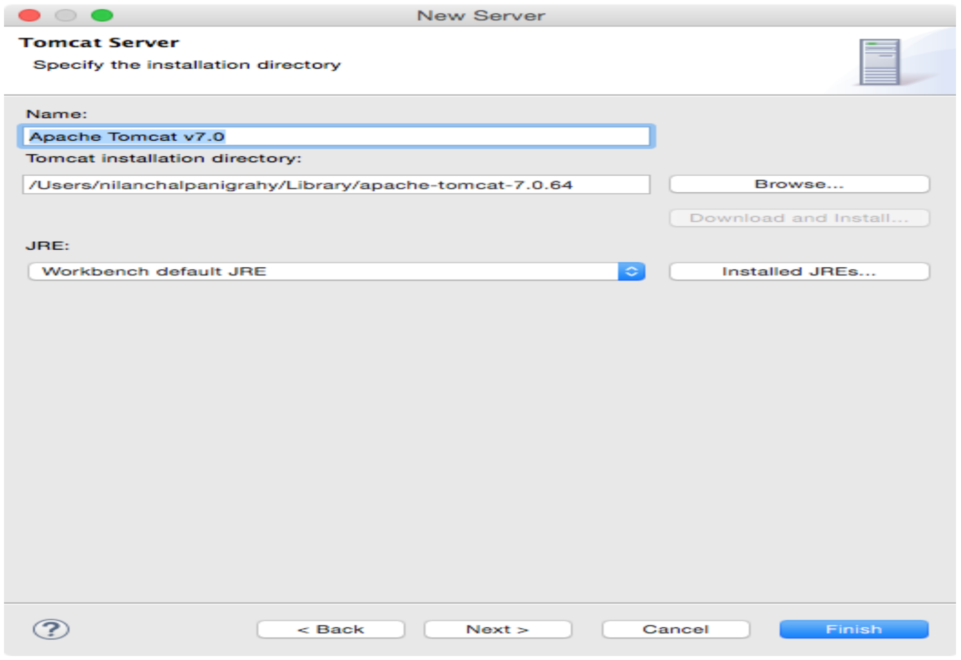
[ ] Include snapshot archetypes

- **Provide the details for Group Id and Artifact Id and click Finish. As a general practice, the group id is domain name and artifact id is the name of the application.**
- **This will create a basic Maven project template in eclipse.**

3. Create a Server Instance

- **Apache Tomcat application server for deploying our Spring MVC application. Now let us add a server instance on eclipse.**

- File **->** New **->** Other**… It will open the eclipse select wizard.**


New Server

Tomcat Server

Specify the installation directory

Name: Apache Tomcat v7.0

Tomcat installation directory: /Users/nilanchalpanigrahy/Library/apache-tomcat-7.0.64

JRE: Workbench default JRE

- **Select Server -> Server and click Next.**
- **Select Apache -> Tomcat v7.0 Server and click Next. Here we are using Apache Tomcat version 7.0. However, the similar steps will work for other tomcat versions.**
- **Browse and select the Apache Tomcat v7.0 server installation directory and click Finish.**

4. Update Project Build Path

If we might notice an error on our project. This is due to the project Build path problem. To fix this you need to right click on project -> **Properties** to open project Java Build Path settings.

Select **Libraries** tab and click on **Add Library**…-> **Server Runtime** -> **Apache Tomcat** -> **Finish**.

5. Configure Spring Dependency

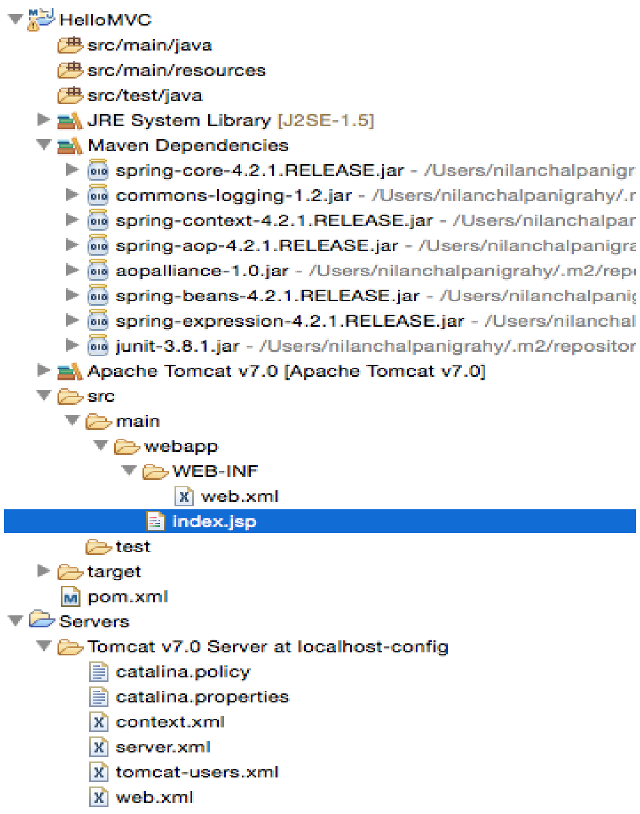
Before we develop Spring MVC web application, we need to configure the project by adding the required Maven dependencies. Add the following Maven dependencies into your pom.xml file.

pom.xml

```
<project
        xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
```

```
	<modelVersion>4.0.0</modelVersion>
	<groupId>com.javatechig</groupId>
	<artifactId>HelloMVC</artifactId>
	<packaging>war</packaging>
	<version>0.0.1-SNAPSHOT</version>
	<name>HelloMVC Maven Webapp</name>
	<url>http://maven.apache.org</url>

	<!--Spring library version -->
	<properties>
		<spring.version>4.2.1.RELEASE</spring.version>
	</properties>

	<dependencies>

		<!-- Spring dependencies -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-core</artifactId>
			<version>${spring.version}</version>
		</dependency>

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

		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-web</artifactId>
			<version>${spring.version}</version>
		</dependency>
		<!--End Spring dependencies -->

		<dependency>
			<groupId>junit</groupId>
			<artifactId>junit</artifactId>
			<version>3.8.1</version>
			<scope>test</scope>
		</dependency>
	</dependencies>
	<build>
		<finalName>HelloMVC</finalName>
	</build>
</project>
```

With this the basic Spring project configuration is complete. Your project structure should look like as the following screenshot.



```
▼ HelloMVC
    src/main/java
    src/main/resources
    src/test/java
  ▶ JRE System Library [J2SE-1.5]
  ▼ Maven Dependencies
    ▶ spring-core-4.2.1.RELEASE.jar - /Users/nilanchalpanigr
    ▶ commons-logging-1.2.jar - /Users/nilanchalpanigrahy/.r
    ▶ spring-context-4.2.1.RELEASE.jar - /Users/nilanchalpar
    ▶ spring-aop-4.2.1.RELEASE.jar - /Users/nilanchalpanigra
    ▶ aopalliance-1.0.jar - /Users/nilanchalpanigrahy/.m2/rep
    ▶ spring-beans-4.2.1.RELEASE.jar - /Users/nilanchalpani
    ▶ spring-expression-4.2.1.RELEASE.jar - /Users/nilanchal
    ▶ junit-3.8.1.jar - /Users/nilanchalpanigrahy/.m2/repositor
  ▶ Apache Tomcat v7.0 [Apache Tomcat v7.0]
  ▼ src
    ▼ main
      ▼ webapp
        ▼ WEB-INF
            web.xml
          index.jsp
      test
  ▶ target
    pom.xml
▼ Servers
  ▼ Tomcat v7.0 Server at localhost-config
      catalina.policy
      catalina.properties
      context.xml
      server.xml
      tomcat-users.xml
      web.xml
```

The following steps will take you through the rest of the steps to create and deploy Spring MVC HelloWorld app.

6. Configure Dispatcher Servlet

The DispatcherServlet must be configured as normal in web.xml to bootstrap a Spring WebApplicationContext. Edit the default web.xml file and add the following.

web.xml

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
	<display-name>Archetype Created Web Application</display-name>
	<context-param>
		<param-name>contextConfigLocation</param-name>
		<param-value>/WEB-INF/dispatcher-servlet.xml</param-value>
	</context-param>

	<listener>
```

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

	<servlet>
		<servlet-name>dispatcher</servlet-name>
		<servlet-class>
			org.springframework.web.servlet.DispatcherServlet
		</servlet-class>
		<load-on-startup>1</load-on-startup>
	</servlet>

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

7. Mapping Requests

Create an xml file dispatcher-servlet.xml under the same directory of web.xml.

dispatcher-servlet.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

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

	<bean
	class="org.springframework.web.servlet.view.InternalResourceViewResolve
r">
		<property name="prefix">
			<value>/WEB-INF/views/</value>
		</property>
		<property name="suffix">
			<value>.jsp</value>
		</property>
	</bean>
</beans>
```

In the above xml file, base-package specifies the package of the controllers. prefix specifies the directory of views, and it is set to be /WEB-INF/views/, which means views directory should be created under WEB-INF. suffix specifies the file extension of views. For example, given a view hello, the view will be located as /WEB-INF/views/hello.jsp.

8. Create Controller class

Create the HelloWorldController under src/main/java/ directory.

HelloWorldController.java

```
package com.demo.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

@Controller
public class HelloWorldController {

      @RequestMapping("/hello")
      public ModelAndView welcomeMessage(
                  @RequestParam(value = "name", required = false) String
name) {
            // Name of your jsp file as parameter
            ModelAndView view = new ModelAndView("hello");
            view.addObject("name", name);
            return view;
      }
}
```

In the code above, @RequestMapping annotation maps web requests onto specific handler classes and/or handler methods, in this case, welcomeMessage(). It provides a consistent style between Servlet environments, with the semantics adapting to the concrete environment.

RequestParam indicates that a method parameter should be bound to a web request parameter. In this case, we also make it not required and give it a default value. The ModelAndView(“hello”) determines that hello is the target view.

9. Working with Views

Edit the default index.jsp and add the following code snippets.

index.jsp

```
<html>
<body>
    <h2>Hello World</h2>
    <h3><a href="hello?name=Sally">Click here...</a></h3>
</body>
```

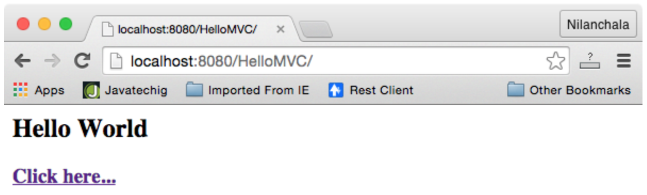
```
</html>
```

Add a JSP files `hello.jsp` file under `/WEB-INF/views/` directory.

hello.jsp

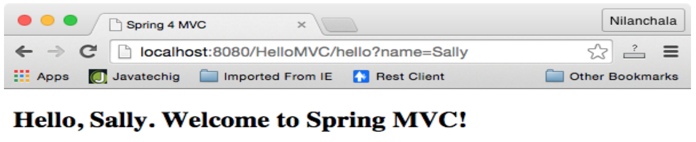
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
      pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Spring 4 MVC</title>
<%@ page isELIgnored="false"%>
</head>
<body>
      <h2>Hello, ${name}. Welcome to Spring MVC!</h2>
</body>
</html>
```

Now we are done with our first HelloWorld example. To deploy it on Tomcat application server, Right click on the project -> select **Run as**. Choose **Tomcat server**, select **Next** and **Finish**. It will deploy the project and to see the output visit http://localhost:8080/HelloMVC on your browser. you will see the following output on your browser.



Hello World

Click here...



Hello, Sally. Welcome to Spring MVC!