JAVA CONFIGURATION (NO XML)

**Hello World using Java configuration.**

1 - Create the maven webapp project.

2 - Delete the web.xml file.

**Error:**

web.xml is missing and is set to true pom.xml

**Solution:**

Update the pom.xml file and add the following in the element:

<plugins>

<plugin>

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

<version>2.2</version>

<configuration>

<failOnMissingWebXml>false</failOnMissingWebXml>

</configuration>

</plugin>

</plugins>

**Error:**

The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path

**Solution:**

Add the following dependency

<dependency>

<groupId>javax</groupId>

<artifactId>javaee-api</artifactId>

<version>7.0</version>

<scope>provided</scope>

</dependency>

3 - Add the spring-webmvc dependency in the project.

* Just above the dependencies element add the properties element

<properties>

<spring.version>4.3.4.RELEASE</spring.version>

</properties>

* Then add the following dependency for working with spring-webmvc

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

4 - Update the pom.xml file to work with the latest version of Java

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

Now right click on the project select Maven -> Update the project

5 - Change the project facet for the following error:

**Error**

Java compiler level does not match the version of the installed Java project facet.

**Solution**

Right Click on Project -> Properties -> Project Facets -> Uncheck Dynamic Web Module and make sure the java version is 1.8

6 - Inside the main directory create another directory with the name java

7 - Create the initializer and config package using the convention of groupid.artifactid.XXX by right clicking of src/main/java

net.kzn.collaborationbackend.initializer

net.kzn.collaborationbackend.config

8 - Create the MVCWebApplicationInitializer class inside the initializer package with the following definition

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

public class MvcWebApplicationInitializer extends AbstractAnnotationConfigDispatcherServletInitializer {

@Override

protected Class<?>[] getRootConfigClasses() {

return new Class[] {};

}

@Override

protected Class<?>[] getServletConfigClasses() {

return new Class[] {MvcConfig.class};

}

@Override

protected String[] getServletMappings() {

return new String[] {"/"};

}

}

9 - Now create the MvcConfig class inside the config package with the following class definition.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.ViewResolver;

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

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;

@Configuration

@EnableWebMvc

@ComponentScan("net.kzn.collaborationbackend")

public class MvcConfig extends WebMvcConfigurerAdapter {

// Configuration to load the static resources

@Override

public void addResourceHandlers(ResourceHandlerRegistry registry) {

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

}

// Configuration for view resolver

@Bean

public ViewResolver configureViewResolver() {

InternalResourceViewResolver viewResolve = new InternalResourceViewResolver();

viewResolve.setPrefix("/WEB-INF/views/");

viewResolve.setSuffix(".jsp");

return viewResolve;

}

// Use the DefaultServletHandlerConfigurer

@Override

public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer){

configurer.enable();

}

}

10 - Create a folder inside WEB-INF with the name as views and copy the index.jsp file inside the views directory.

11 - We will create a test controller only to serve and check that the url mapping are mapped properly to the handler method. For this create another package with groupid.artifactid.controller.

import org.springframework.stereotype.Controller;

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

@Controller

public class TestController {

@RequestMapping(value = {"/","/index"})

public String index() {

return "index";

}

}

12 - Deploy the application on your favorite container that could be Tomcat, Jetty, Glassfish etc. Run your application on the server.