**What is a web.xml ?**

It is a config file for our web-app. It’s a deployment descriptor.

* Here we defines classes, resource and the configuration of the application.
* The web server uses these information to serve a web request.
* When the web server receives a web request, it uses the deployment descriptor’s url mapping to send the request to a specific servlet which can handle the request.

**Who is our web server ?**

Tomcat

**web.xml does so many things for us. Is it a important file ?**

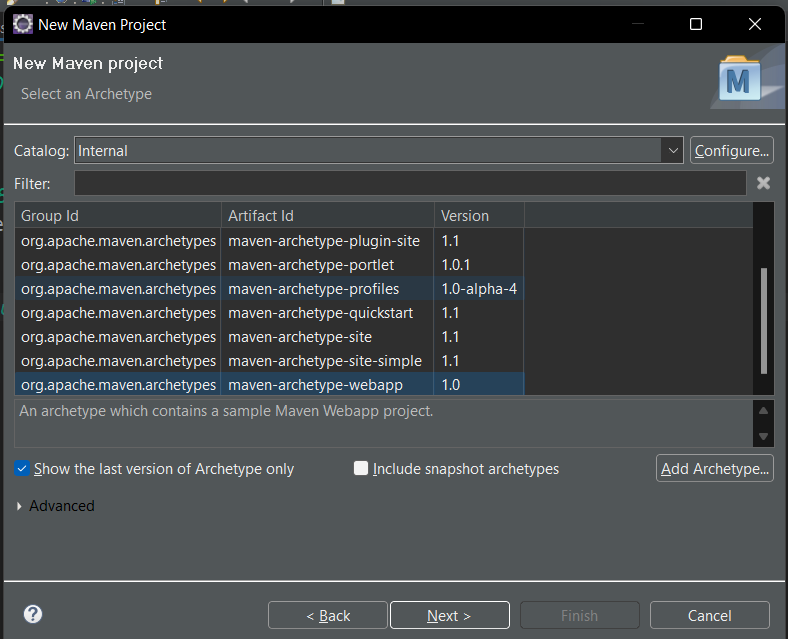
Yes, it is pretty important.

**What if we delete the web.xml ?** **Can we run/deploy a web app without having web.xml file ?**

Yes, this file is optional from **servlet 3.0**. Now an another question is arise that where will we configure all the important information. So answer is very simple – **Java Annotation**

Now, we are going to create another web project in which we will eliminate the **web.xml** file.

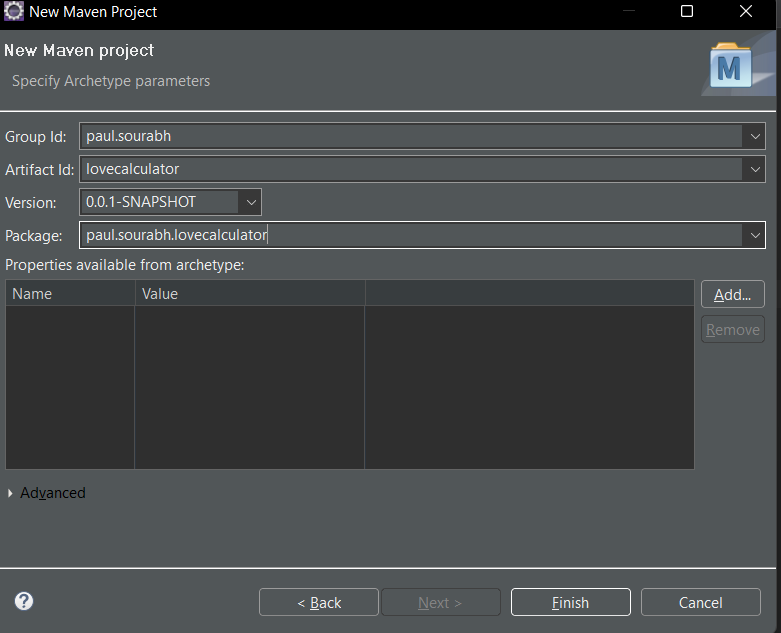
**Step 1 :** Create a Maven Project.

* File / New / Maven Project
* Choose **maven-archetype-webapp** and hit next.****
* As the next step, we need to provide 3 things i.e. **Group id, Artifact id and Package**

Where,

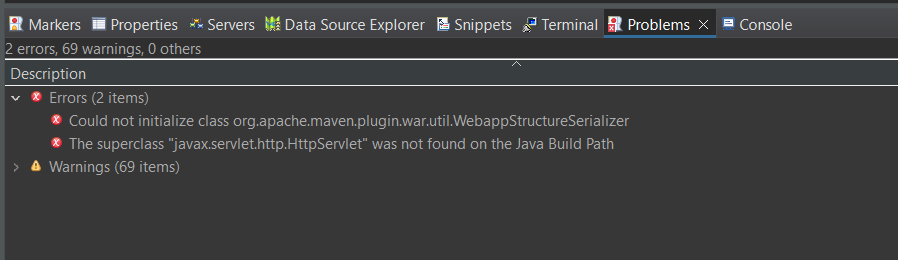
**Group id** is the Company Name.

**Artifact id** is the Project Name.



**Step 2 :** Resolving errors in a fresh maven project.

* Yes, we you creates a new maven project it can come up with few errors. Now, here are two errors that can be arise.



* **Could not initialize class org.apache.maven.plugin.war.util.WebappStructureSerializer**
  + This error can be resolve by adding a below plugin in the **pom.xml**
    - Open pom.xml
    - Inside **<build>** **tag** paste the below code :

<plugins>

<plugin>

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

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

<version>3.3.1</version>

</plugin>

</plugins>

* + - As a last step, update the project : Right Click on project / Maven / Update project **Or** alt + f5
* **The superclass "javax.servlet.http.HttpServlet" was not found on the Java Build Path**
  + Right Click On Project / Properties / Project Facets / Runtime / Choose Appropriate Tomcat Server Version / Apply

**Step 3 :** Now, run your project to make sure project is running perfectly with no error. If your project printing “Hello World”. Congrats !

**Step 4 :** Add spring webmvc maven dependency. To do this, just add the below code into your **<dependencies>** tag. After adding if you check the **Libraries** folder you will find that maven has automatically added all the jar file into this folder.

<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->

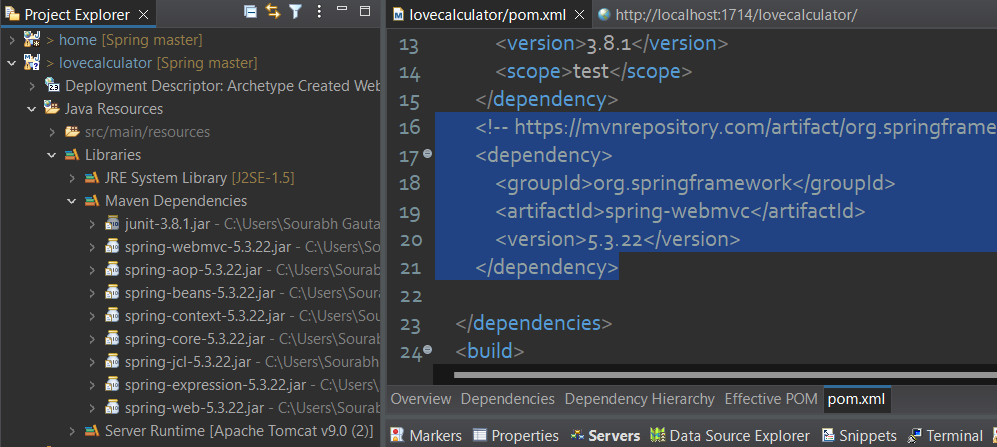
<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.22</version>

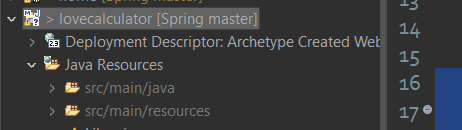
</dependency>

****

You guys can be surprised because spring downloaded **spring-aop**, **spring-core** and many more dependencies with **spring-mvc**. It is because spring-mvc is totally dependent on this jar as well.

**Step 5 :** Your Project 🡪 Expand Java Resources 🡪 Right Click On src/main/resources 🡪 Show In 🡪 System Explorer 🡪 Create new folder with name **java** using **ctrl + shift + N**

Now refresh your project ( to refresh select your project and press f5 ) your project should have new folder named **java** into **Java Resources**



**Step 6 :** Now create a package **com.lc.config** into **src/main/java** and then create a class in it name **LCApplicationInitializer** and it should implement the interface called **WebApplicationInitializer**.

**WebApplicationInitializer** is a functional interface i.e. it is only have one single method called **onStartup**

public interface WebApplicationInitializer {

void onStartup(ServletContext servletContext) throws ServletException;

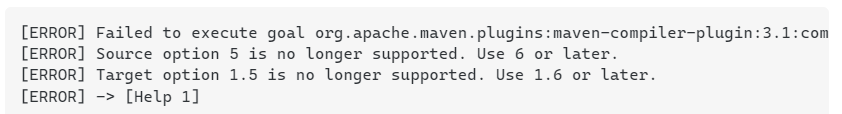
}

Now, override this method inside your class and add a below print statement in it.

System.***out***.println("onStartup method called");

**Step 7 :** Testing the project. To test the project, Right Click On Project 🡪 Run as 🡪 Maven Install. If it is showing Build Success message then testing is pass. But you may face the below errors.

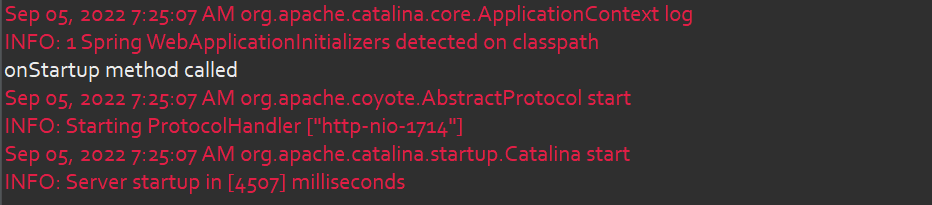
* Error 1



If you are getting the above error, you need to update your Java Version inside build path.

**Project / Build Path / Configure Build Path / Libraries / Delete the Existing One / Add Library / JRE System Library / Alternate Or Workspace Default**

**Step 8 :** Run the project. Project 🡪 Run as 🡪 Run on server.



If you carefully watch your console. You will find that spring automatically called the **onStartup()** method hence it is printing the message. That mean, if your class is implementing the **WebApplicationInitializer** interface so spring will automatically detect this class and called the method **onStartup().**

**Step 9 :** Create a new xml file inside src/main/resource folder with name **config.xml.**

**config.xml :**

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

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

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

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

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

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

*http://www.springframework.org/schema/mvc*

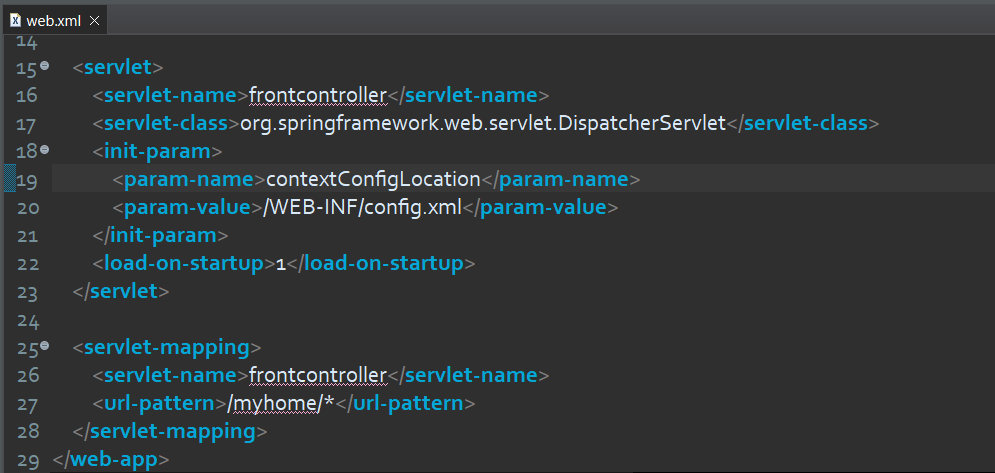
*http://www.springframework.org/schema/mvc/spring-mvc.xsd*

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

*https://www.springframework.org/schema/context/spring-context.xsd "*>

</**beans**>

**Step 9 :** Configuring Front Controller.

****

In web.xml approach, this is how we were configuring the front controller as shown in the above image. But we are in the mission of eliminating web.xml so configuring front controller should be done inside a java class.

To do so,

Add the below code into **onStartup()** method of **LCApplicationInitializer** class

public void onStartup(ServletContext servletContext) throws ServletException {

//create DispatcherServlet object

XmlWebApplicationContext webApplicationContext = new XmlWebApplicationContext();

webApplicationContext.setConfigLocation("classpath:config.xml");

DispatcherServlet ds = new DispatcherServlet(webApplicationContext);

//Register DispatcherServlet object with ServletContext

ServletRegistration.Dynamic sc = servletContext.addServlet("frontcontroller", ds);

sc.setLoadOnStartup(1);

sc.addMapping("/myhome/\*");

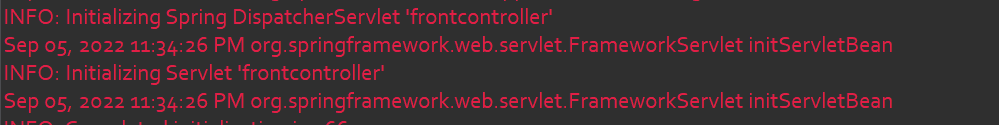
}

Before move ahead, I’ll suggest you to compare this code with the image of web.xml above. This complete code being eliminating the web.xml

Please compare which line of this code is doing the same job as of which tag in web.xml

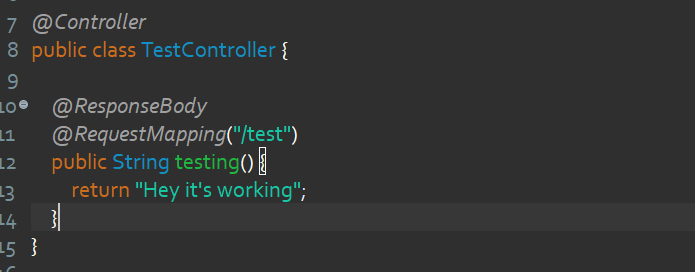
**Step 10 :** Run the application again.

When you run the application this time tomcat should initialize the DispatcherServlet and to verify whether DispatcherServlet is initialized or not. Just look into your console and find your DispatcherServlet name i.e. “frontcontroller”.

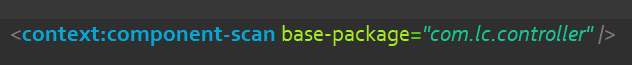


**Step 11 :** Creating a TestController.

* Create a new package com.ic.config into src/main/java
* Create a class with TestController.java in it with the below body.



* Add <component-scan> tag into **config.xml**



* Now run your project and try to access the below url, if it is accessible then congrats! You Rock!

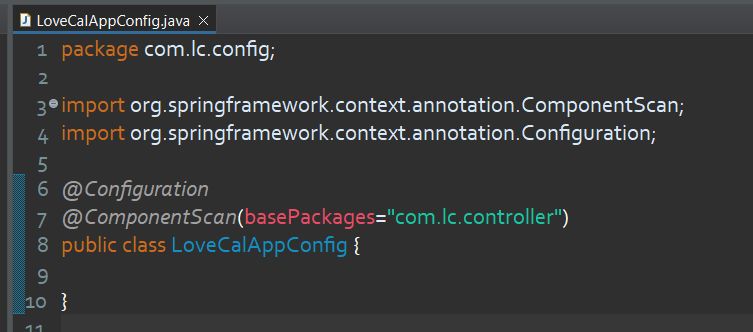
[**http://localhost:1714/lovecalculator/myhome/test**](http://localhost:1714/lovecalculator/myhome/test)

So far we have removed the complete **web.xml** in 11 steps. But our job is not done yet because we have still an xml file inside our project i.e. **config.xml**

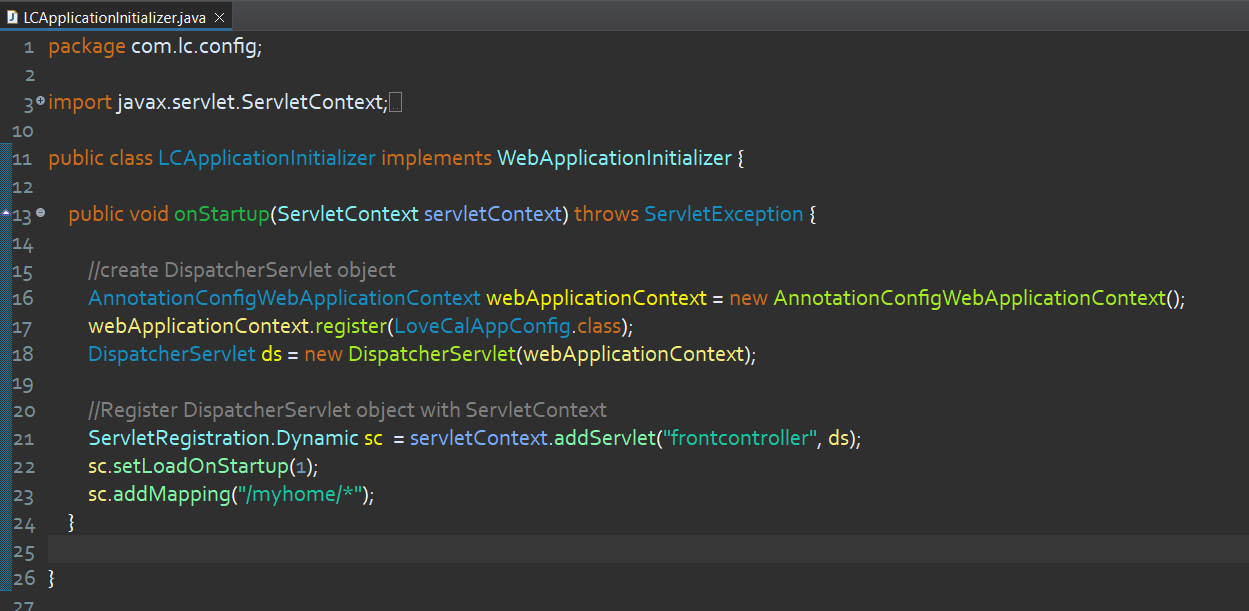
If you studied spring core you must have an idea how we going to eliminate **config.xml** i.e. by creating a configuration java class using **@Configuration** annotation.

So we are not going to deep dive in this. Follow the below steps :

1. Create a class LoveCalAppConfig with below body inside the package **com.lc.config**

****

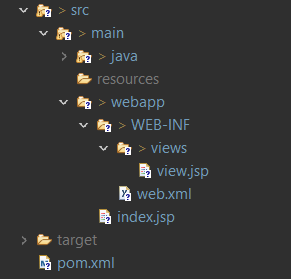
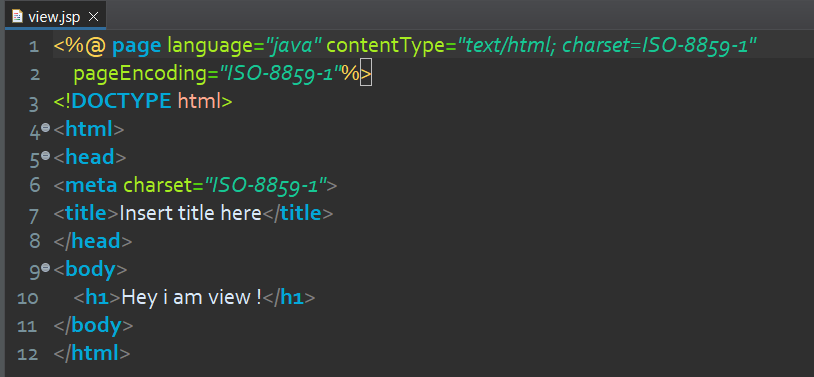
1. Update **LCApplicationInitializer** class as shown below :

****

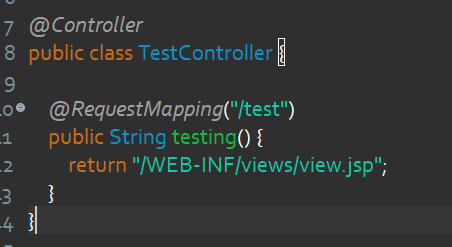
1. Now delete the **config.xml**
2. Run the project and visit the [**http://localhost:1714/lovecalculator/myhome/test**](http://localhost:1714/lovecalculator/myhome/test) if this link is accessible then congrats again.

It’s time to add some Views in our project.

Let’s add a View called **view.jsp**

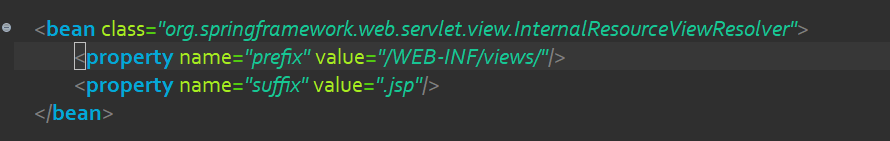
Now, update your **TestController** by removing **@Response** and give path of **view.jsp** like below :



Now run the project and visit the [**http://localhost:1714/lovecalculator/myhome/test**](http://localhost:1714/lovecalculator/myhome/test) and View should be rendered.

**Talk about View Resolver !**

If you remember, when we were learning about View Resolver. We had created it’s object through **<bean>** tag.



But this time we don’t have any config.xml file so to make the object of View Resolver will be done inside the LoveCalAppConfig class that we have already created.

**LoveCalAppConfig.java :**

*@Configuration*

*@ComponentScan*(basePackages="com.lc.controller")

public class LoveCalAppConfig {

*@Bean*

InternalResourceViewResolver viewResolver() {

InternalResourceViewResolver vr = new InternalResourceViewResolver();

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

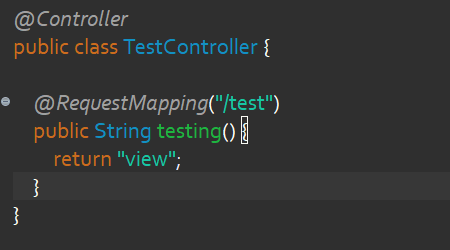
vr.setSuffix(".jsp");

return vr;

}

}

Now update your TestController by removing suffix and prefix from it.



Now still your project should display the same output by accessing this link - [**http://localhost:1714/lovecalculator/myhome/test**](http://localhost:1714/lovecalculator/myhome/test)