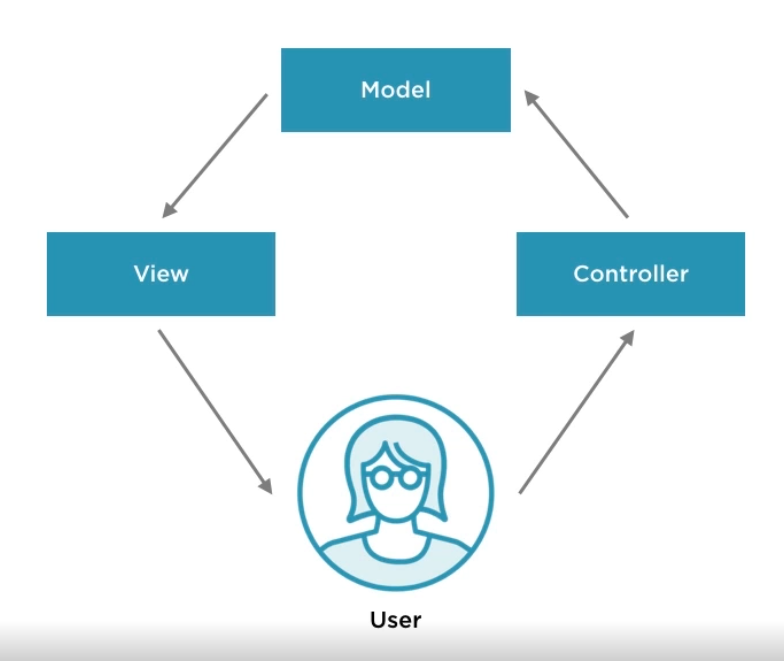
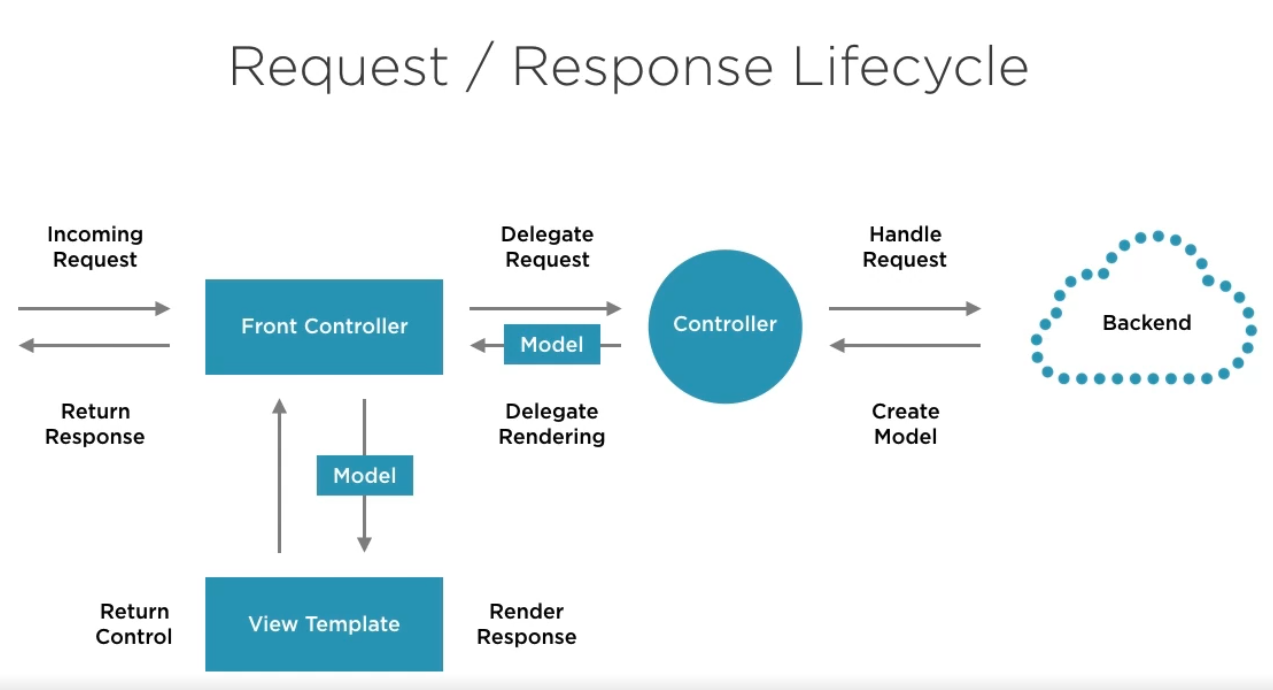
**SPRING MVC**

**Spring MVC is used for java web development.**



**Spring MVC lifecycle**



* Incoming request from **browser** hits our **front controller** [ dispatcher servlet]. This is just a dispatcher which decides which controller will handle the request.
* The **Controller** handles the request and sends to the backend.
* The **backends service** consist of database business logic etc. Now backend gets the data **creates** **a model** and hands it back.
* Model is just of data which we want to represent on our screen.
* Once data via **model** comes to **Controller**, the controller will give this to someone else to do the rendering (which is separate from business logic).
* Now once Model comes to **front controller** it will pass to **view template** (can be any JSP etc).
* Once JSP page render the request then it will **return back to front controller**
* Via front controller it is return to browser.

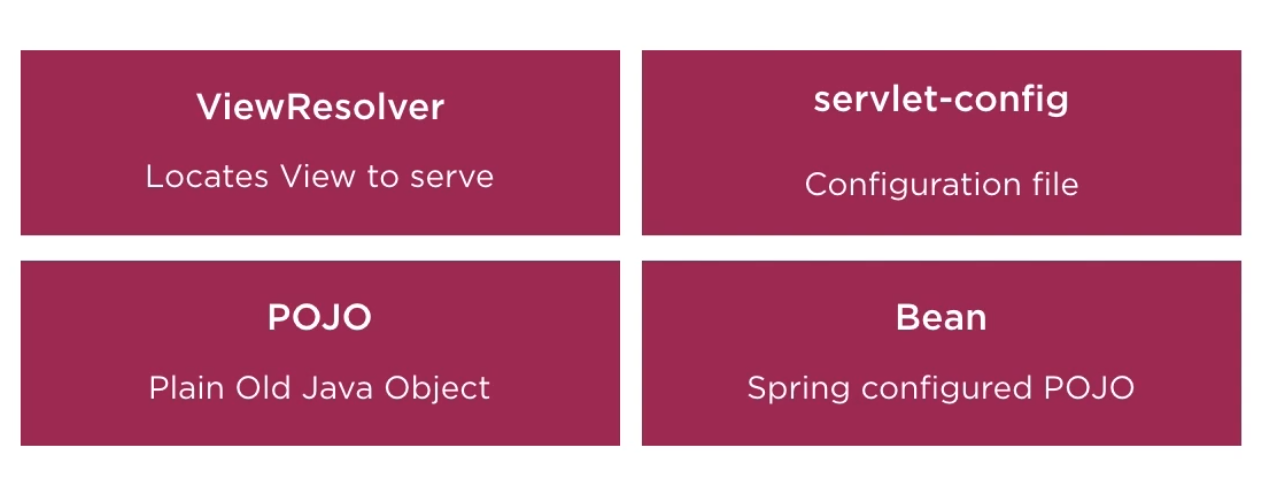
**What SPRING MVC provides:**



* Initially most of java application was written with java-server pages for frontend and had basic controller-based architecture.
* Later people were looking for alternative of self-service and come REST services. Full featured.
* As javascript took over SPA is used widely for front-ends.
* Spring MVC provides full support to REST and made it a great support to SPA apps.

**VOCABULARY WE SHOULD KNOW**





**SPA –** Single page application like angular.

**Dispatcher Servlet-** Entry point in our application, this is the place where spring MVC webapp begins its configuration. We always start with dispatcherservlet**.**

**Controller –** It is an implementation of Command pattern handler.

**Request Mapping –** Everything inside spring MVC is associated with request mapping. URL and type of request that we are making.

**View Resolver –** It is use to locate the view and serve it back. It can be used to find a template,jsp page , html page etc.

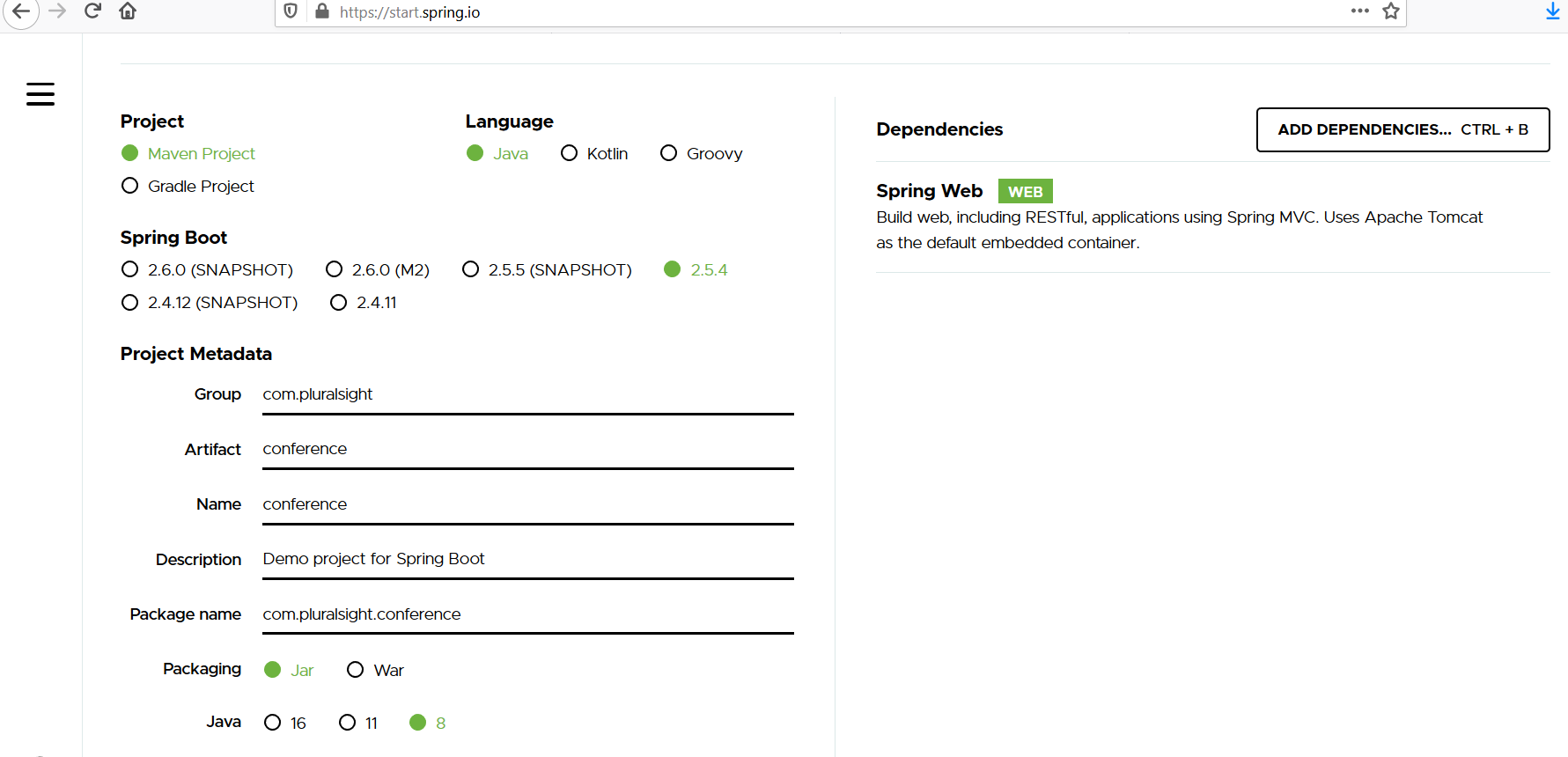
**Servlet-config** – this use to be web.xml file coupled with servlet-config file. Now everything can be done using java configuration.

**POJO** – All spring bean are simple POJOs (no -args cons and getter and setter).

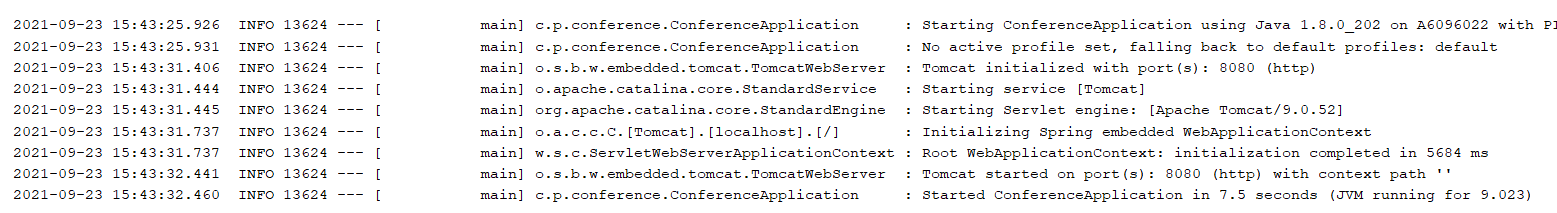
**Bean** – Spring configured POJO.

**STARTING OUR SPRING MVC PROJECT**

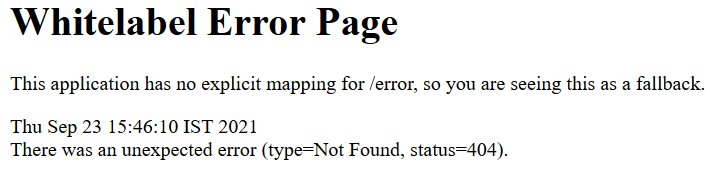
**Fully self-contained jar as a spring boot application**



* As we already added a dependency of Spring Web **it automatically has embedded tomcat in our container. PACKAGING IS JAR.**
* Once we generate the spring boot application. Then we need to extract the zip file.
* Open intellij and import project as maven.
* Now as this is a web project when will run the ApplicationMain then it will initialize embedded tomcat and start it at port 8080.



* But when in browser we do localhost:8080 then it will give error as we don’t have any index page to show.



* We need to create a index.html page in resource-> static directory
* Then just restart the application.

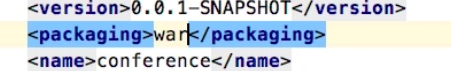
**Above approach of self -contained spring boot works well when we want to have restful services.**

**But deploying JSP pages (existing project) will not work well with self-contained jar. Hence, we need to package it as a WAR and deploy to tomcat server.**

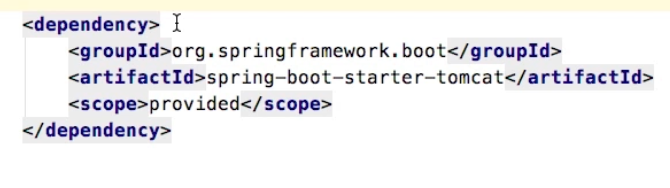
**PACKAGING AS WAR AND DEPLOY TO TOMCAT SERVER**

**IN INTELLIJ ULTIMATE**

* Download tomcat version
* Put the tomcat location into File -> setting -> Build execution deployment 🡪 application server
* Now we will convert the self-contained (self executable) jar into **packaging war** so that we can deploy in tomcat.
* Make changes to POM and add package info below version.



* We will also add dependency of springboot.starter.tomcat explicitly as we are deploying on standalone tomcat and to remove the dependency in internal package application (which we added tomcat dependency during creating springboot starter.)

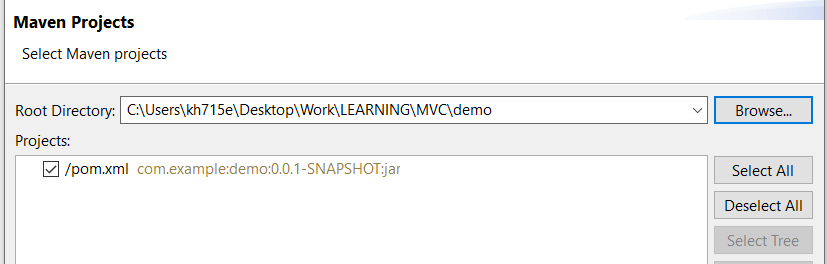


* Now in order to deploy war in standalone tomcat, we need to make changes to the configuration.
* Add a new configuration select tomcat 🡪locale , then server 🡪 make sure our tomcat version is there and correct JRE is there.
* Now go to deployment , press + and add our .war and change application context.

Finally, we need to create a webapp directory parallel to java and create a index.html. The index.html is created in webapp because when we are packaging and deploying as a WAR then it will not look for root in resource🡪static rather it will look for webapp.

**IN STS:**

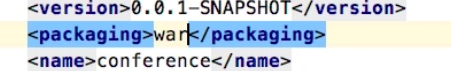
* **Have a workspace location and copy the zip of spring boot into that location.**
* **Go to file🡪 import🡪 Maven 🡪 Existing maven project**



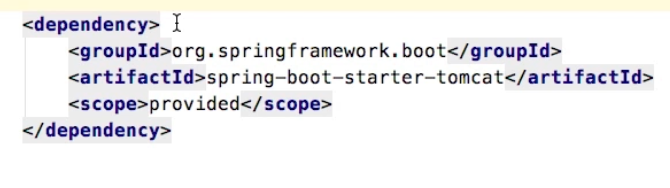
* **Once project is imported in STS , first go to windows🡪 preferences🡪java🡪compiler, please check 1.8 is there.**
* **Then go to java-> Installed JREs and check location for jre and jdk.**
* **If we are running just spring boot integrated one just do run as spring boot application.**

**NOW TO CHANGE TO STANDALONE TOMCAT AND PACKAGE AS WAR**

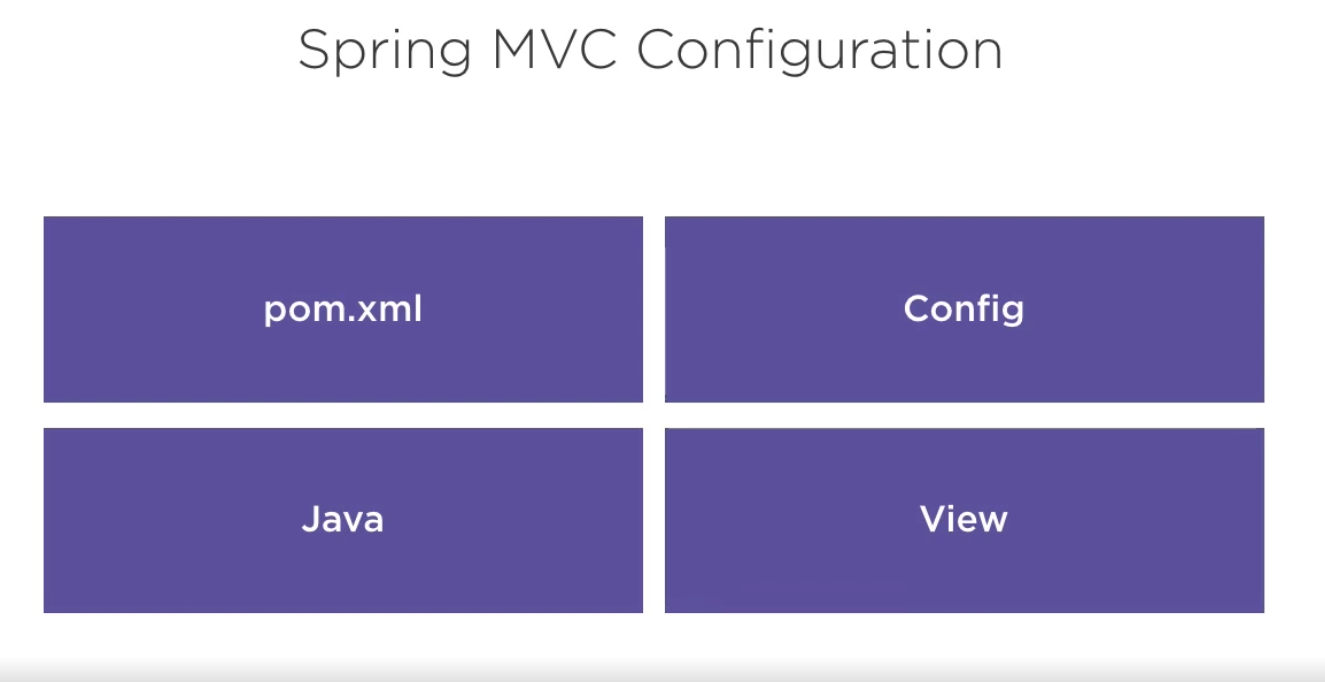
* **Now we need to add a index.html in webapp so create folder and go to java\_build\_path🡪 source folder and add webapp.**
* **Now go to project facets and select java,javascript and dynamic web 4.0.**
* **Now go to window 🡪 preference 🡪 server🡪 runtime env and add tomcat location**
* Now we will convert the self-contained (self executable) jar into **packaging war** so that we can deploy in tomcat.
* Make changes to POM and add package info below version.



* We will also add dependency of springboot.starter.tomcat explicitly as we are deploying on standalone tomcat and to remove the dependency in internal package application (which we added tomcat dependency during creating springboot starter.)



* **Then do run as maven install [ it will clean build and run], we will get**
* 
* **So select jdk in windows ->preferences instead of jre.**
* **Again do maven install and now it will build our war successfully.**
* **Once war is build run as server**
* **If getting any project level error**



* **First, we need pom.xml file to install any dependency which we can create from start.spring.io**
* **Then we have a configuration section either java, xml web.xml etc**
* **Then we have our java files controller, model possibly configuration(if we are using java).**
* **Then we have view can be anything, JSP , angular etc**