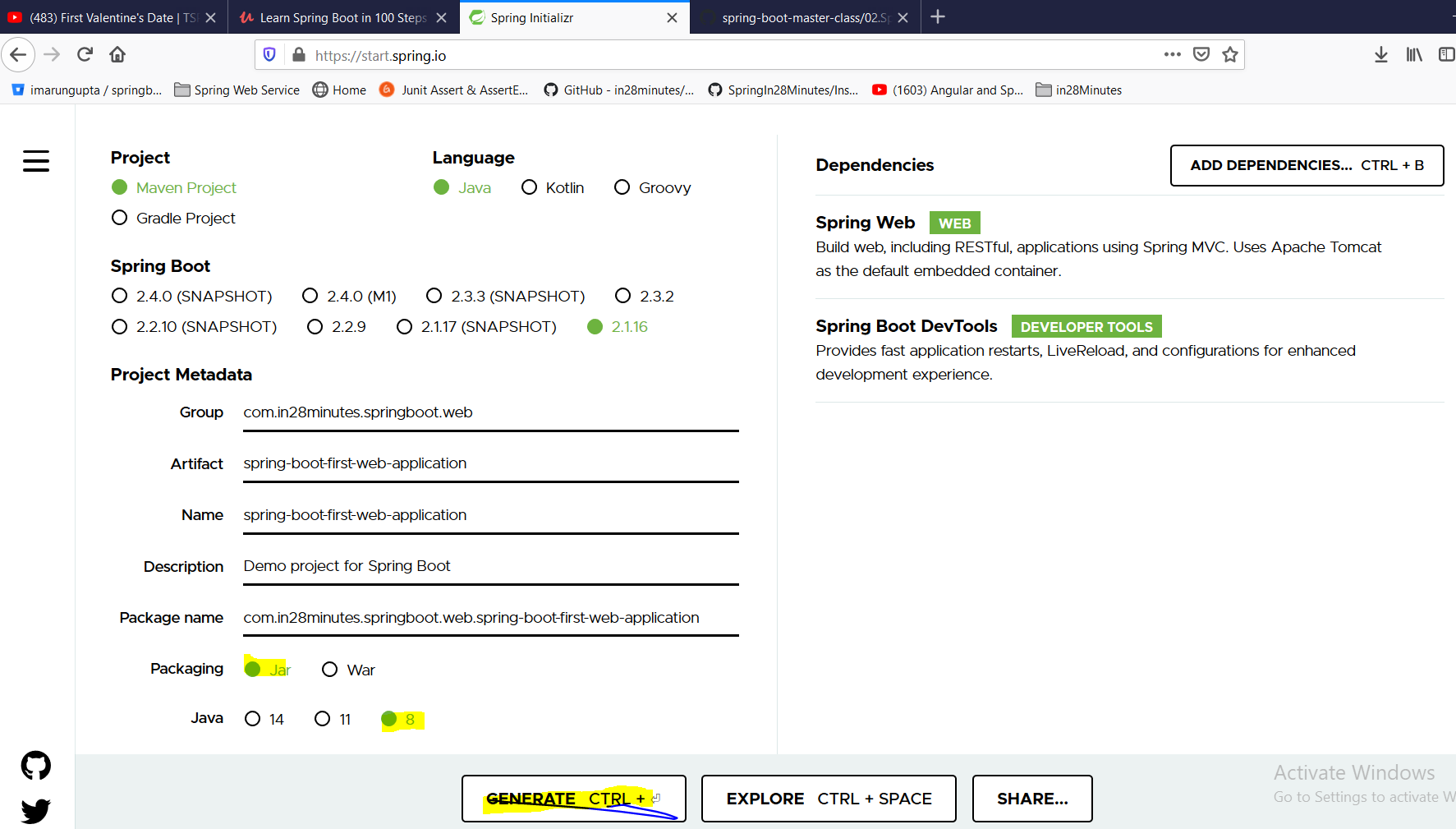


**Step 01: Part 1 Basic Spring Boot Web Application Setup**

1- We would use spring initializer to setup our project.

2- We will get a sample web application from <https://start.spring.io/>

3- One of the important projects that the spring boot provides is called spring initializer hosted at <https://start.spring.io/>. This helps us to setup web application



**Group**: defines the package of the project

**Artifact**: Defines the name of the project

**Packaging** : jar

**Dependencies** : Spring Web and Spring Boot DevTools

**Spring Web** Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container

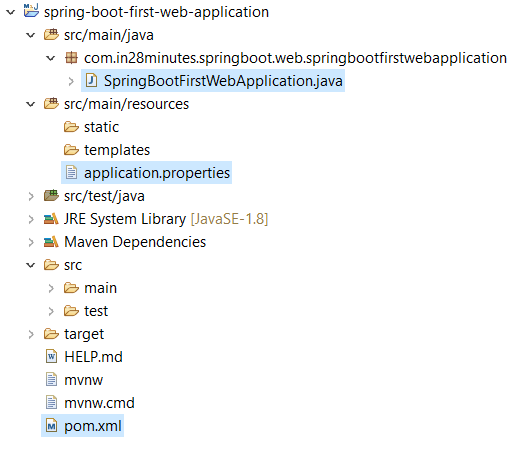
**Spring Boot DevTools Developer Tools** Provides fast application restarts, LiveReload, and configurations for enhanced development experience. This devtools makes the development faster and make the developer productive.

As soon as we click on Generate, it will generate zip folder which will contain a maven project.

Now unzip and import it as existing maven project into eclipse.

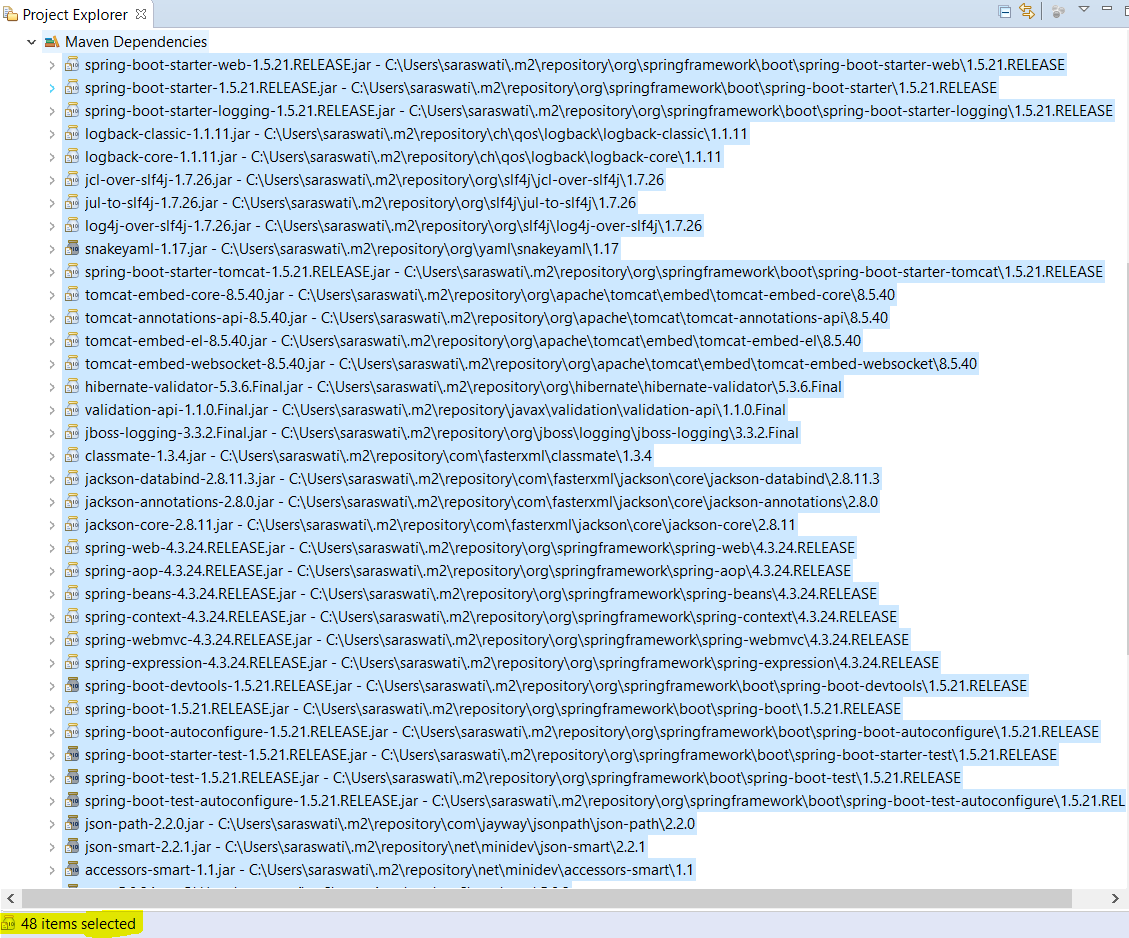
Now if we expand the project in eclipse we would see three important file

* 1. SpringBootFirstWebApplication.java
  2. application.properties
  3. POM.xml (Project Object Model) : Describes the characteristics of the project



* 1. **Pom.xml**

|  |
| --- |
| <?xmlversion=*"1.0"*encoding=*"UTF-8"*?>  <projectxmlns=*"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 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <!-- Project Detail -->  <modelVersion>4.0.0</modelVersion>  <groupId>com.in28minutes.springboot.web</groupId>  <artifactId>spring-boot-first-web-application</artifactId>  <version>0.0.1-SNAPSHOT</version>  <name>spring-boot-first-web-application</name>  <description>Demo project for Spring Boot</description>  <! -- adding parent: It is very similar to java, just inherit parent ability into this POM.xml. This parent would manage all the dependencies version accordingly without any conflict -->  <parent>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-parent</artifactId>  <version>1.5.21.RELEASE</version>  <relativePath/><!-- lookup parent from repository -->  </parent>  <!-- java version -->  <properties>  <java.version>1.8</java.version>  </properties>  <!-- Adding dependencies web for MVC and rest, devtools for auto server stsrt, test for unit and Integration testing-->  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-devtools</artifactId>  <scope>runtime</scope>  <optional>true</optional>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  </dependencies>  <!— - plugin to run the maven project and helps in creating jar or war files for spring boot projects- - 🡪  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build>  </project> |
| 1- spring-boot-starter-parent  2- spring-boot-starter-web  3- spring-boot-devtools  4- spring-boot-starter-test  5- spring-boot-maven-plugin |



All jars got downloaded and managed by Pom.xml (Maven) as soon as we add dependency in the dependency section of POM

And also it will download something called transitive dependencies. For example I am using spring dependency and for to be able to work it need some other dependency like slf4j. So now I need spring as well as its dependency slf4j as dependencies.

So the fallowing things we have seen so far in the POM.xml.

1. Parent
2. Dependency
3. Plugin
4. Java version

2- Now the next important file of the project is SpringBootFirstWebApplication.java

This class is annotated with @**SpringBootApplication** and inside this class we have one main method. This class basically spring boot launcher class.

|  |
| --- |
| package com.in28minutes.springboot.web.springbootfirstwebapplication;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class SpringBootFirstWebApplication {  public static void main(String[] args) {  SpringApplication.run(SpringBootFirstWebApplication.class, args);  }  } |

**@SpringBootApplication**: It does two things

1. It initializes the spring framework (component scan)and
2. It initializes spring boot auto configuration. Spring boot run on top of spring framework

|  |
| --- |
| [-->[Open Declaration](eclipse-open:%E2%98%82=spring-boot-first-web-application/C:\/Users\/saraswati\/.m2\/repository\/org\/springframework\/boot\/spring-boot-autoconfigure\/1.5.21.RELEASE\/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication)](eclipse-open:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication)[org](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg).[springframework](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework).[boot](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot).[autoconfigure](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure).SpringBootApplication  @[SpringBootConfiguration](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot%5C/1.5.21.RELEASE%5C/spring-boot-1.5.21.RELEASE.jar%3Corg.springframework.boot(SpringBootConfiguration.class%E2%98%83SpringBootConfiguration) @[EnableAutoConfiguration](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(EnableAutoConfiguration.class%E2%98%83EnableAutoConfiguration) @[ComponentScan](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan.class%E2%98%83ComponentScan)([excludeFilters](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan.class%E2%98%83ComponentScan~excludeFilters)={@[Filter](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter)([type](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~type)=[CUSTOM](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(FilterType.class%E2%98%83FilterType%5ECUSTOM), [classes](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~classes)={[TypeExcludeFilter](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot%5C/1.5.21.RELEASE%5C/spring-boot-1.5.21.RELEASE.jar%3Corg.springframework.boot.context(TypeExcludeFilter.class%E2%98%83TypeExcludeFilter).class}), @[Filter](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter)([type](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~type)=[CUSTOM](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(FilterType.class%E2%98%83FilterType%5ECUSTOM), [classes](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.24.RELEASE%5C/spring-context-4.3.24.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~classes)={[AutoConfigurationExcludeFilter](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(AutoConfigurationExcludeFilter.class%E2%98%83AutoConfigurationExcludeFilter).class})}) @[Target](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Target.class%E2%98%83Target)([value](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Target.class%E2%98%83Target~value)={[TYPE](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(ElementType.class%E2%98%83ElementType%5ETYPE)}) @[Retention](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Retention.class%E2%98%83Retention)([value](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Retention.class%E2%98%83Retention~value)=[RUNTIME](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(RetentionPolicy.class%E2%98%83RetentionPolicy%5ERUNTIME)) @[Documented](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Documented.class%E2%98%83Documented) @[Inherited](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Inherited.class%E2%98%83Inherited)  Indicates a [configuration](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82Configuration) class that declares one or more [@Bean](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82Bean) methods and also triggers [auto-configuration](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82EnableAutoConfiguration) and [component scanning](eclipse-javadoc:%E2%98%82=spring-boot-first-web-application/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.5.21.RELEASE%5C/spring-boot-autoconfigure-1.5.21.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82ComponentScan). This is a convenience annotation that is equivalent to declaring @Configuration, @EnableAutoConfiguration and @ComponentScan.  **Since:**1.2.0  **Author:**Phillip WebbStephane Nicoll |

To launch this spring boot application just right click and run as – java application which will start the embedded tomcat server and launch the spring boot application.

Now the third important file in this project is application.properties

**application.properties**

This application.properties can be used as configuration file

Let’s say if we want to run our application not default port 8080 but on 9000 then we can configure or change this port in this file (server.port=9000)

**So far we have learnt the following things**

* Creating maven project
* spring boot starter parent
* spring boot starter web
* @SpringBootApplication
* Auto configuration
* Devtools
* spring boot maven plugin

**Step02-**

<https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step02.md>

What You Will Learn during this Step:

* @RequestMapping(value = "/login", method = RequestMethod.GET)
* <http://localhost:8080/login>
* Why @ResponseBody?
* Important of RequestMapping method
* How do web applications work? Request and Response
* Browser sends Http Request to Web Server
* Code in Web Server => Input:HttpRequest, Output: HttpResponse
* Web Server responds with Http Response

Useful Snippets and References

First Snippet

@Controller

public class LoginController {

@RequestMapping(value = "/login")

@ResponseBody

public String sayHello() {

return "Hello World dummy";

}

}

### todo.txt

Spring Boot Starter Parent

Spring Boot Starter Web

@SpringBootApplication

Auto Configuration

|  |
| --- |
| **package com.in28minutes.springboot.web.springbootfirstwebapplication.controller;**  **import org.springframework.stereotype.Controller;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.ResponseBody;**  **import org.springframework.web.bind.annotation.RestController;**  **//@Controller**  **@RestController**  **publicclass LoginController {**    **@RequestMapping("/login")**  **//@ResponseBody**  **public String loginMessage() {**  **return"Hello: My first spring boot web app with RestController";**  **}**  **}** |

**Note**: We don’t need to use @ResponseBody annotation while using @RestController because @RestController is the combination of (@Controller + @ResponseBody) i.e. [@RestController=(@Controller + @ResponseBody)]

Console Log: When we put correct URL and but @ResponseBody is missing at method level: click enters or refresh page then:

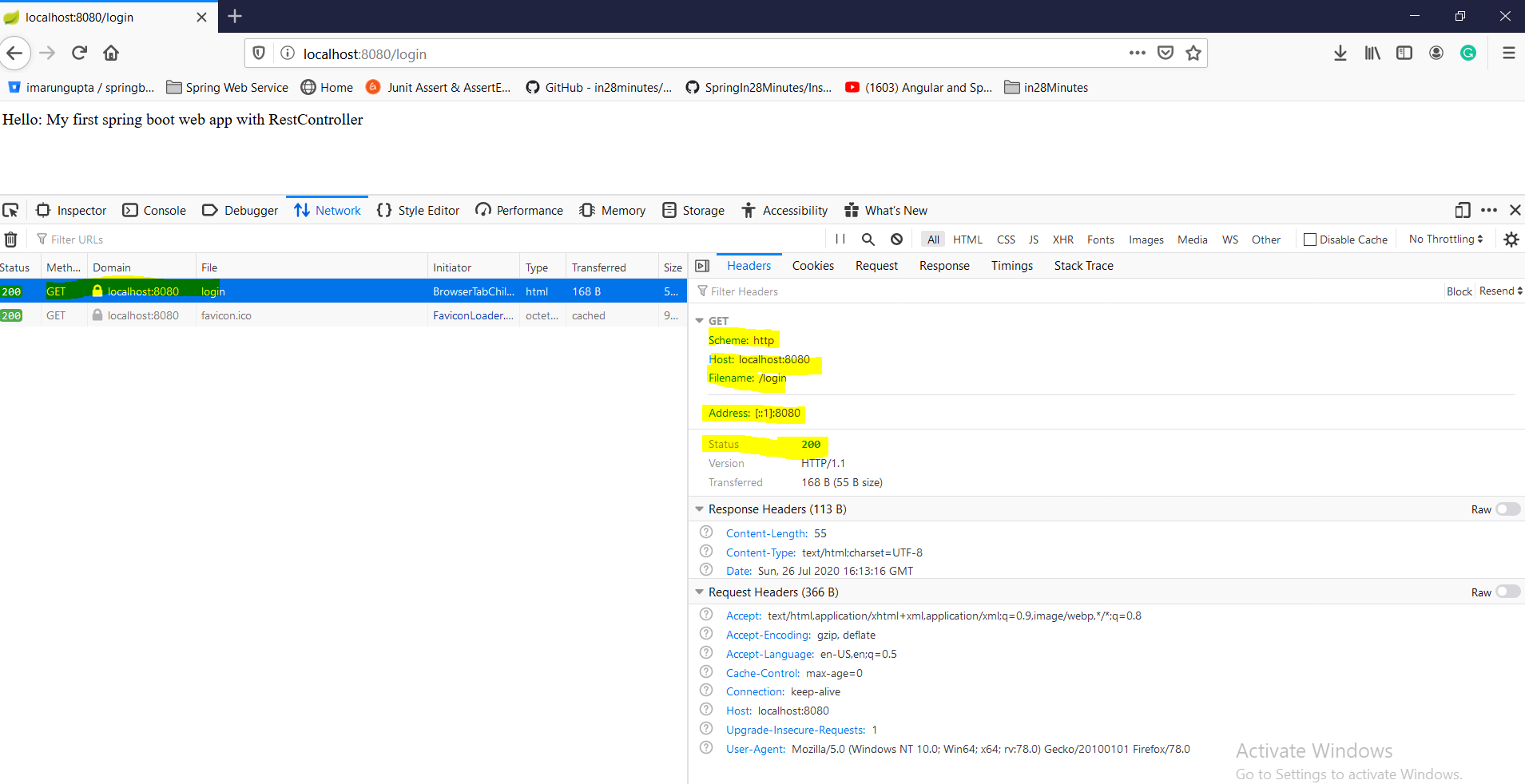
|  |
| --- |
| 2020-07-26 21:22:01.450 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Initializing servlet 'dispatcherServlet'  2020-07-26 21:22:01.451 INFO 8448 --- [nio-9090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring FrameworkServlet 'dispatcherServlet'  2020-07-26 21:22:01.451 INFO 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization started  2020-07-26 21:22:01.451 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Using MultipartResolver [org.springframework.web.multipart.support.StandardServletMultipartResolver@2822282a]  2020-07-26 21:22:01.451 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate LocaleResolver with name 'localeResolver': using default [org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver@2c1da9f6]  2020-07-26 21:22:01.451 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate ThemeResolver with name 'themeResolver': using default [org.springframework.web.servlet.theme.FixedThemeResolver@691f2211]  2020-07-26 21:22:01.452 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate RequestToViewNameTranslator with name 'viewNameTranslator': using default [org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator@61662c46]  2020-07-26 21:22:01.453 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate FlashMapManager with name 'flashMapManager': using default [org.springframework.web.servlet.support.SessionFlashMapManager@583c2b38]  2020-07-26 21:22:01.453 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Published WebApplicationContext of servlet 'dispatcherServlet' as ServletContext attribute with name [org.springframework.web.servlet.FrameworkServlet.CONTEXT.dispatcherServlet]  2020-07-26 21:22:01.453 INFO 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization completed in 2 ms  2020-07-26 21:22:01.453 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Servlet 'dispatcherServlet' configured successfully  2020-07-26 21:22:01.453 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : **DispatcherServlet with name 'dispatcherServlet' processing GET request for [/login]**  2020-07-26 21:22:01.454 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Looking up handler method for path /login  2020-07-26 21:22:01.454 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : **Returning handler method [public java.lang.String com.in28minutes.springboot.web.springbootfirstwebapplication.controller.LoginController.loginMessage()]**  2020-07-26 21:22:01.454 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Last-Modified value for [/login] is: -1  2020-07-26 21:22:01.457 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.v.ContentNegotiatingViewResolver : Requested media types are [text/html, application/xhtml+xml, image/webp, application/xml;q=0.9, \*/\*;q=0.8] based on Accept header types and producible media types [\*/\*])  2020-07-26 21:22:01.457 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.servlet.view.BeanNameViewResolver : **No matching bean found for view name 'Hello: My first spring boot web app with RestController'**  2020-07-26 21:22:01.460 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.v.ContentNegotiatingViewResolver : **Returning [org.springframework.web.servlet.view.InternalResourceView: name 'Hello: My first spring boot web app with RestController'; URL [Hello: My first spring boot web app with RestController]] based on requested media type 'text/html'**  2020-07-26 21:22:01.460 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : **Rendering view [org.springframework.web.servlet.view.InternalResourceView: name 'Hello: My first spring boot web app with RestController'; URL [Hello: My first spring boot web app with RestController]] in DispatcherServlet with name 'dispatcherServlet'**  2020-07-26 21:22:01.462 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.servlet.view.InternalResourceView : Forwarding to resource [Hello: My first spring boot web app with RestController] in InternalResourceView 'Hello: My first spring boot web app with RestController'  2020-07-26 21:22:01.466 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : **DispatcherServlet with name 'dispatcherServlet' processing GET request for [/Hello: My first spring boot web app with RestController]**  2020-07-26 21:22:01.466 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : **Looking up handler method for path /Hello: My first spring boot web app with RestController**  2020-07-26 21:22:01.468 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : **Did not find handler method for [/Hello: My first spring boot web app with RestController]**  2020-07-26 21:22:01.468 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Matching patterns for request [/Hello: My first spring boot web app with RestController] are [/\*\*]  2020-07-26 21:22:01.468 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : URI Template variables for request [/Hello: My first spring boot web app with RestController] are {}  2020-07-26 21:22:01.468 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapping [/Hello: My first spring boot web app with RestController] to HandlerExecutionChain with handler [ResourceHttpRequestHandler [locations=[ServletContext resource [/], class path resource [META-INF/resources/], class path resource [resources/], class path resource [static/], class path resource [public/]], resolvers=[org.springframework.web.servlet.resource.PathResourceResolver@8208b4]]] and 1 interceptor  2020-07-26 21:22:01.469 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Last-Modified value for [/Hello: My first spring boot web app with RestController] is: -1  2020-07-26 21:22:01.470 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Null ModelAndView returned to DispatcherServlet with name 'dispatcherServlet': assuming HandlerAdapter completed request handling  2020-07-26 21:22:01.470 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Successfully completed request  2020-07-26 21:22:01.471 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Successfully completed request  2020-07-26 21:22:01.471 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : DispatcherServlet with name 'dispatcherServlet' processing GET request for [/error]  2020-07-26 21:22:01.471 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Looking up handler method for path /error  2020-07-26 21:22:01.473 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Returning handler method [public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)]  2020-07-26 21:22:01.473 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Last-Modified value for [/error] is: -1  2020-07-26 21:22:01.483 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.v.ContentNegotiatingViewResolver : Requested media types are [text/html, text/html;q=0.8] based on Accept header types and producible media types [text/html])  2020-07-26 21:22:01.485 DEBUG 8448 --- [nio-9090-exec-1] o.s.w.s.v.ContentNegotiatingViewResolver : Returning [org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$SpelView@610f3009] based on requested media type 'text/html'  2020-07-26 21:22:01.485 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Rendering view [org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration$SpelView@610f3009] in DispatcherServlet with name 'dispatcherServlet'  2020-07-26 21:22:01.566 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Successfully completed request |

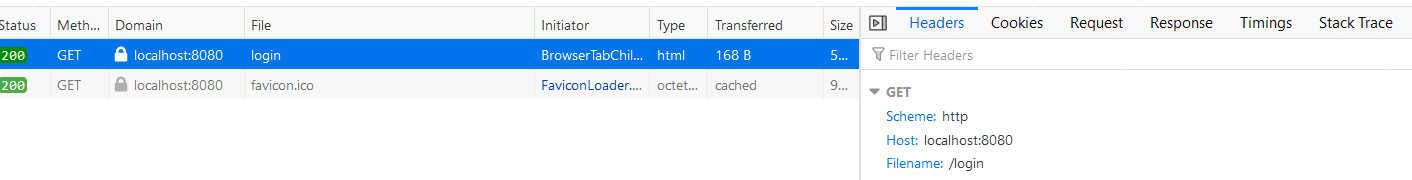
So here we will get whiteLevel error and will not get any response

But when we include @ResponseBody then we will get proper response: compare both the logs

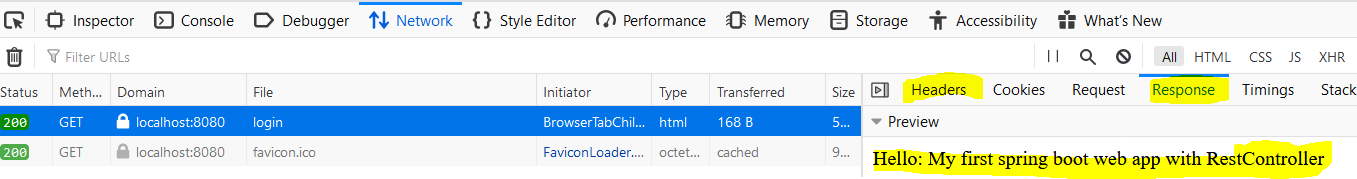
|  |
| --- |
| 2020-07-26 21:15:07.682 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : **Initializing servlet 'dispatcherServlet'**  2020-07-26 21:15:07.682 INFO 8448 --- [nio-9090-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring FrameworkServlet 'dispatcherServlet'  2020-07-26 21:15:07.682 INFO 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization started  2020-07-26 21:15:07.682 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Using MultipartResolver [org.springframework.web.multipart.support.StandardServletMultipartResolver@340dec05]  2020-07-26 21:15:07.682 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate LocaleResolver with name 'localeResolver': using default [org.springframework.web.servlet.i18n.AcceptHeaderLocaleResolver@30327e5c]  2020-07-26 21:15:07.683 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate ThemeResolver with name 'themeResolver': using default [org.springframework.web.servlet.theme.FixedThemeResolver@7530e63e]  2020-07-26 21:15:07.684 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate RequestToViewNameTranslator with name 'viewNameTranslator': using default [org.springframework.web.servlet.view.DefaultRequestToViewNameTranslator@1f39eec1]  2020-07-26 21:15:07.685 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Unable to locate FlashMapManager with name 'flashMapManager': using default [org.springframework.web.servlet.support.SessionFlashMapManager@448f54e7]  2020-07-26 21:15:07.685 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Published WebApplicationContext of servlet 'dispatcherServlet' as ServletContext attribute with name [org.springframework.web.servlet.FrameworkServlet.CONTEXT.dispatcherServlet]  2020-07-26 21:15:07.685 INFO 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : FrameworkServlet 'dispatcherServlet': initialization completed in 3 ms  2020-07-26 21:15:07.685 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Servlet 'dispatcherServlet' configured successfully  2020-07-26 21:15:07.685 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : **DispatcherServlet with name 'dispatcherServlet' processing GET request for [/login]**  2020-07-26 21:15:07.686 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : **Looking up handler method for path /login**  2020-07-26 21:15:07.686 DEBUG 8448 --- [nio-9090-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping **: Returning handler method [public java.lang.String com.in28minutes.springboot.web.springbootfirstwebapplication.controller.LoginController.loginMessage()]**  2020-07-26 21:15:07.686 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Last-Modified value for [/login] is: -1  2020-07-26 21:15:07.703 DEBUG 8448 --- [nio-9090-exec-1] m.m.a.RequestResponseBodyMethodProcessor : **Written [Hello: My first spring boot web app with RestController] as "text/html" using [org.springframework.http.converter.StringHttpMessageConverter@c4ee0e4]**  2020-07-26 21:15:07.703 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Null ModelAndView returned to DispatcherServlet with name 'dispatcherServlet': assuming HandlerAdapter completed request handling  2020-07-26 21:15:07.703 DEBUG 8448 --- [nio-9090-exec-1] o.s.web.servlet.DispatcherServlet : Successfully completed request |

**What basically happens when we hit url(**[**http://localhost:8080/login**](http://localhost:8080/login) **),what happens in the background?**





When we hit the url : it goes the server <http://localhost:8080/login> and in the response section we get the response as shown below.



**Here's the Quick Start with Top 10 FAQ**

<https://github.com/in28minutes/in28minutes-initiatives/blob/master/The-in28Minutes-TroubleshootingGuide-And-FAQ/quick-start.md>

**Step03-**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step03.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step03.md)

|  |
| --- |
| What You Will Learn during this Step:   * Demystifying some of the magic * Spring Boot Starter Parent * Spring Boot Starter Web * Embedded Tomcat * Dev Tools   **Spring Boot Starter Parent**: hover the mouse on pom.xml parent >Ctrl+click> it will take you to the another pom.xml parent (it is like inheritance class A extends B) which will contain [spring-boot-dependencies], default java version 1.6, Spring core, Maven plugin for running the project, creating jar or war.  Again hover the mouse on its parent >ctrl+click> it will take us to the another pom.xml where the versions of all the dependencies(jars) is mentioned.  In this way spring-boot-starter-parent manages the dependencies and its versions.  **Spring Boot Starter Web: Basically it does two things**   1. All the dependencies that we need to run the web and rest application (like spring core, spring mvc, validation, logging, embedded tomcat, Jackson data binding etc.) To get more go to pom.xml>hover the mouse on spring-boot-starter-web> ctrl+click and here we will see all the things under this. 2. **Embedded Tomcat server,** which is now the part of application. Like earlier Tomcat server were sitting outside where we were creating war file and then pushing into tomcat server. That is now we are not doing here in spring boot. So here we are including tomcat in application that is why it is called embedded tomcat. So the tomcat server is now the part of application, which allows us to do the configure in other server e.g. if we need to deploy this application jar on Linux server then we don’t need configure tomcat server separately because tomcat server is already embedded in the application. 3. **Dev Tools:** Provides fast application restarts, LiveReload, and configurations for enhanced development experience. This devtools makes the development faster and make the developer productive. Basically the dev tool keeps monitoring the application folders and as soon as we do any changes in any of the project file, it reloaded the application into the tomcat server, and we don’t need to restart tomcat server explicitly. Generally the dynamic part like java file will be reloaded again and again if we do any changes.   This tool will be only for dynamic part like java but if we have made any changes in pom.xml (added any new dependency) then in this case we need to restart the server. |

## ****Step04-****

## <https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step04.md>

|  |
| --- |
| Untill now we had only controller, now we will use JSP and for that we will write view (jsp). Three things we need to add into so far existing project mentioned in yellow colour. What You Will Learn during this Step:  Your First JSP  There is a bit of setup before we get there!  Introduction to View Resolver  Useful Snippets and References  First Snippet - /src/main/webapp/WEB-INF/jsp/login.jsp  <html>  <head>  <title>Yahoo!!</title>  </head>  <body>  My First JSP!!!  </body>  </html>  Second Snippet - /src/main/resources/application.properties  spring.mvc.view.prefix: /WEB-INF/jsp/  spring.mvc.view.suffix: .jsp  logging.level.: DEBUG  Third Snippet : To enable jsp support in embedded tomcat server!  <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  Exercises  Create a new jsp and a new controller method to redirect to it!  Play around!  Files List  **pom.xml**  spring-boot-starter-parent  <java.version>1.8</java.version>  spring-boot-starter-web  spring-boot-devtools    <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  <scope>provided</scope>  </dependency>  spring-boot-starter-test  spring-boot-maven-plugin    src/main/java/com/in28minutes/springboot/web/controller/LoginController.java  package com.in28minutes.springboot.web.controller;  import org.springframework.stereotype.Controller;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.ResponseBody;  @Controller  public class LoginController {    @RequestMapping("/login")  //@ResponseBody  public String loginMessage(){  return "login";  }  }  **Note: @ResponseBody annotation made commented because it used to print method response into browser. But here we are printing response from view (jsp) that is why we need not to use this annotation.**  **src/main/resources/application.properties**  spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp  logging.level.org.springframework.web=DEBUG  **src/main/webapp/WEB-INF/jsp/login.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  My First JSP!!  </body>  </html>  **todo.txt**  Spring Boot Starter Parent  Spring Boot Starter Web  @SpringBootApplication  Auto Configuration  Dispatcher Servlet  /login => "login"  "login" => src/main/webapp/WEB-INF/jsp/login.jsp  Search for a view named "login"  /login => LoginController |

## ****Step05-****

## <https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step05.md>

|  |
| --- |
| What You Will Learn during this Step:   * You first GET Parameter. * Problem with using GET   **pom.xml**  Same dependency  **src/main/java/com/in28minutes/springboot/web/controller/LoginController.java**  package com.in28minutes.springboot.web.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  @Controller  public class LoginController {  //Model  @RequestMapping("/login")  public String loginMessage(@RequestParam String name, **ModelMap** model){  model.put("name", name);  return "login";  }  }  **Note: Model is used to pass data from controller to view(jsp)The same name we will have in model class as well as in jsp(where this value need to display)**  **src/main/resources/application.properties**  spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp  logging.level.org.springframework.web=DEBUG  **src/main/webapp/WEB-INF/jsp/login.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  My First JSP!! Welcome ${name}!  </body>  </html>  **todo.txt**  Spring Boot Starter Parent  Spring Boot Starter Web  @SpringBootApplication  Auto Configuration  Dispatcher Servlet  /login => "login"  "login" => src/main/webapp/WEB-INF/jsp/login.jsp  Search for a view named "login"  /login => LoginController |

## So as here we can see that if we want to send some data from the controller to the view, in that case we will use model to map controller data to the view (jsp). So controller control the entire flow and once the controller has some data, then it put into the model and redirects to the view and view used to model to render the data on the screen. That is why it called MVC (Model-View-Controller)

|  |
| --- |
| public String loginMessage(@RequestParam String name, **ModelMap** model){  model.put("namekey", name);  return "login"; Here we have created name as string request parameter using @RequestParam annotation and putting this request parameter value into model and using model put method we are putting this value into model key and now this model will be available into view hence finally using this key we print the request parameter value into jsp using expression language${ }.My First JSP!!! welcome:${namekey} |

## 

## 

## So far:

|  |  |
| --- | --- |
| Spring Boot Starter ParentSpring Boot Starter Webtomcat-embed-jasperspring-boot-devtoolsspring-boot-starter-testspring-boot-maven-plugin@SpringBootApplicationAuto Configuration | @Controller@RequestMapping("/mylogin")RequestParam String name,ModelMap model@RunWith(SpringRunner.class)@SpringBootTest |

## ****Step06-****

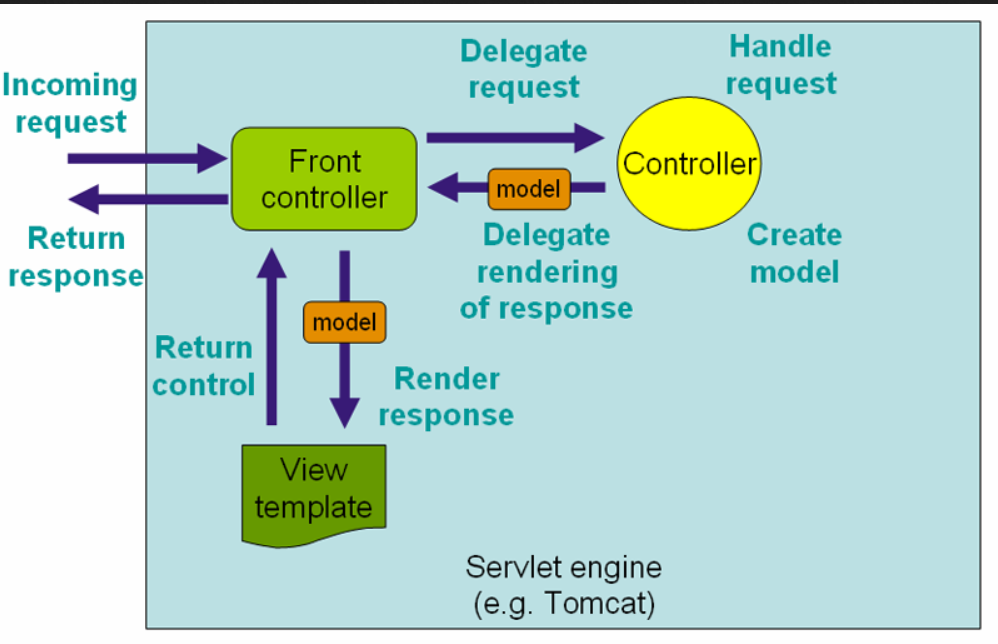
## <https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step06.md>

What You Will Learn during this Step:

* Understand importance of DispatcherServlet.

Spring MVC Request Flow

* DispatcherServlet is nothing but front controller which first receives all the request (whether it is login or error or anything) comes from web application. i.e. DispatcherServlet receives HTTP Request.
* After that it search that what are the mapping(like /login) available and/or param(like- ?name=arun) in request comes in the form of URL
* DispatcherServlet identifies the right Controller (LoginController) based on the URL (/login).
* Controller executes Business Logic.
* Controller returns a) Model b) View Name Back to DispatcherServlet.
* DispatcherServlet identifies the correct view (ViewResolver- i.e. logical jsp name).
* DispatcherServlet makes the model available to view (jsp) and executes it using prefix and suffix configure in the application.properties file.
* DispatcherServlet returns HTTP Response Back.
* Flow : <http://docs.spring.io/spring-framework/docs/2.0.8/reference/images/mvc.png>

****

In POM.xml: We have four main tags

1- Parent tags

2- Properties tags

3- dependencies tags

4- build tag

## ****Step07-****

## <https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step07.md>

|  |
| --- |
| What You Will Learn during this Step:   * Lets get the name from the user(request url using @RequestParam) in a form   First Snippet  @RequestMapping(value = "/login", method = RequestMethod.GET)  public String showLoginPage(ModelMap model, @RequestParam String name) {  return "login";  }  @RequestMapping(value = "/login", method = RequestMethod.POST)  public String handleLogin(ModelMap model, @RequestParam String name) {  model.put("name", name);  return "welcome";  }  Second Snippet  <form method="POST">  Name : <input name="name" type="text" /><input type="submit" />  </form>  **Files List**  pom.xml: Same what we have used in the above example. No change.  src/main/java/com/in28minutes/springboot/web/controller/LoginController.java  src/main/java/com/in28minutes/springboot/web/SpringBootFirstWebApplication.java  src/main/resources/application.properties  src/main/webapp/WEB-INF/jsp/login.jsp  src/main/webapp/WEB-INF/jsp/welcome.jsp  src/test/java/com/in28minutes/springboot/web/SpringBootFirstWebApplicationTests.java  src/main/java/com/in28minutes/springboot/web/controller/LoginController.java  package com.in28minutes.springboot.web.controller;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RequestParam;  @Controller  public class LoginController {    @RequestMapping(value="/login", method = RequestMethod.GET)  public String showLoginPage(ModelMap model){  return "login";  }  // This method just redirecting login page  @RequestMapping(value="/login", method = RequestMethod.POST)  public String showWelcomePage(ModelMap model, @RequestParam String name){  model.put("name", name);  return "welcome";  }  }  src/main/resources/application.properties  spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp  logging.level.org.springframework.web=DEBUG  **src/main/webapp/WEB-INF/jsp/login.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  <form method="post">  Name : <input type="text" name="name" />  Password : <input type="password" name="password" />  <input type="submit" />  </form>  </body>  </html>  **Note: Created a form and input tag to accept request parameter**  **src/main/webapp/WEB-INF/jsp/welcome.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Welcome ${name}!!  </body>  </html>  **todo.txt**  Spring Boot Starter Parent  Spring Boot Starter Web  @SpringBootApplication  Auto Configuration  Dispatcher Servlet  /login => "login"  "login" => src/main/webapp/WEB-INF/jsp/login.jsp  Search for a view named "login"  /login => LoginController |

## So far we have learnt how to send request parameter through the URL and learnt how to print the same on JSP. Now we will learn an alternative way to send request parameter through form.

## Get method is not secure because whatever we will type in the text box it will show in the request URL. Let’s say if we are passing name=Ranga and password= dummy and click on submit then in the request URL we will get like as:

## http://localhost:8080/login?name=Ranga&password=dummy

## 

## While hit this URL then there is a number of router between browser and server which trough this URL goes and some can read and inject other things.

## Now when we hit <http://localhost:8080/getLogin>

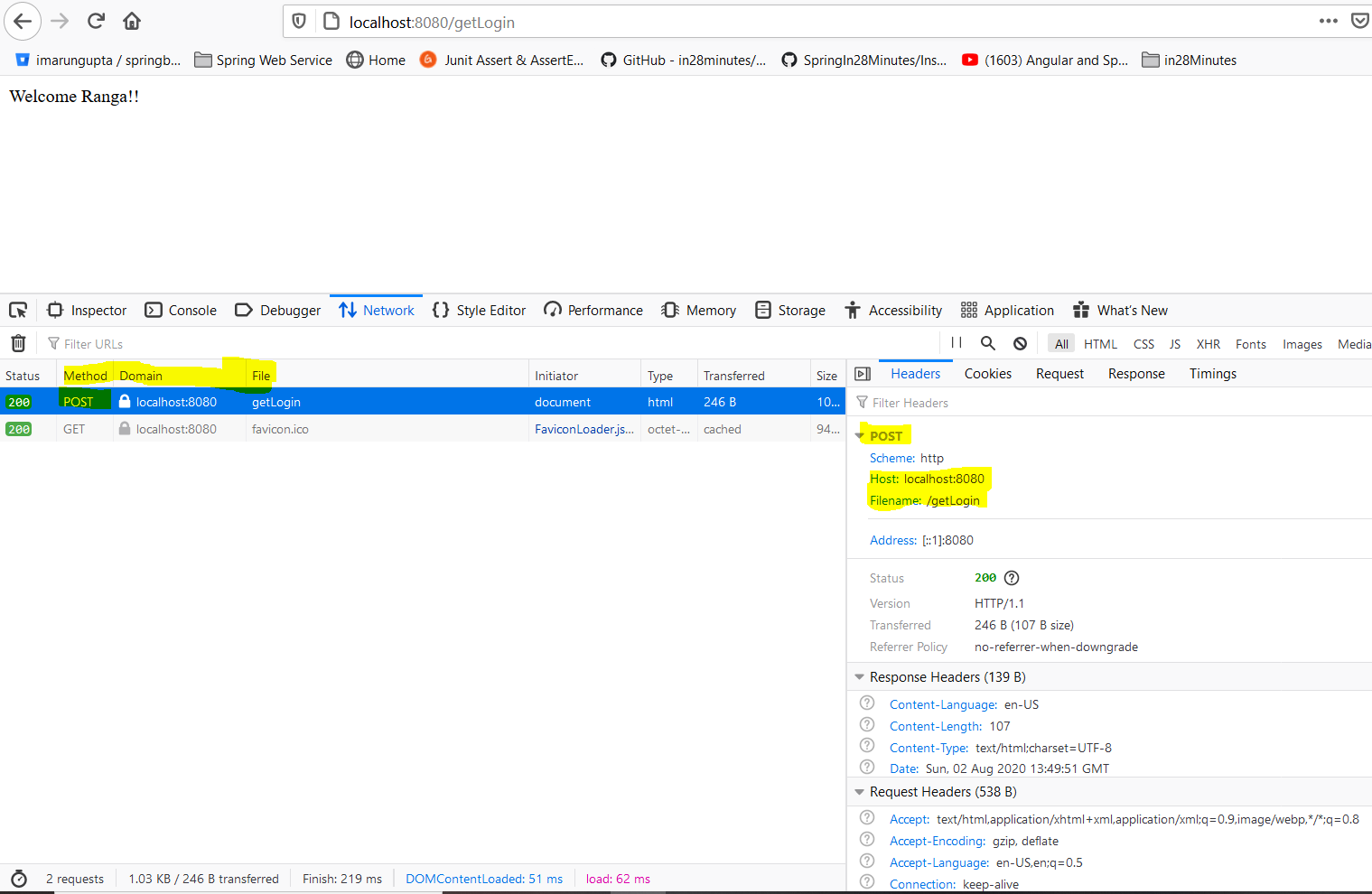
## 

## In the above screen Method is Get and in the header section we see detail.

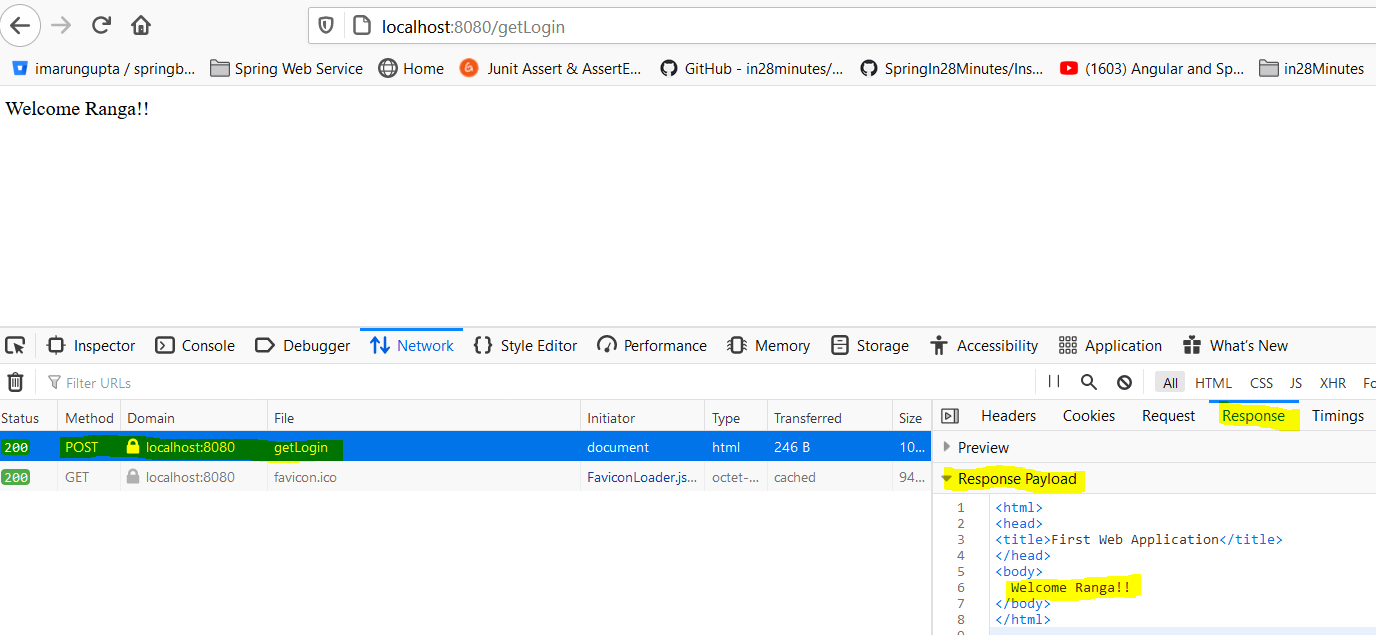
## 

In the Response tab we get response payload (html) and preview.

Now let enter name and password and click on Submit Query.



And in Response tab: Here method is POST



**But post method is secure because it does not send the request parameter with the Request URL. So make sure that when we are sending any parameter to the server, use only post method.**

**So as we have seen that we have two methods in the LoginController class one method is having Get request method and another method is having POST request method. So once we hit the URL then first method having Get request method gets called because Get is default method and its gets called automatically once hit the request URL. After that we get login page. Now in the login form the method type is POST so once we click on submit button the handler start matching the mapping URL (/getLogin) and method POST and hence it called another method annotated with POST method. This is the logic behind of matching and calling particular method based on the URL mapped and request method.**

## ****Step08-****

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step08.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step08.md)

|  |
| --- |
| **What You Will Learn during this Step:**   1. Add validation for userid and password 2. Hard coded validation!!   First Snippet  package com.in28minutes.springboot.web.service;  import org.springframework.stereotype.Component;  @Component  public class LoginService {  public boolean validateUser(String user, String password) {  return user.equalsIgnoreCase("in28Minutes") && password.equals("dummy");  }  }  Second Snippet  @Autowired  private LoginService service;  @RequestMapping(value = "/login", method = RequestMethod.POST)  public String handleLogin(ModelMap model, @RequestParam String name,  @RequestParam String password) {  boolean isValidUser = service.validateUser(name, password);  if (isValidUser) {  model.put("name", name);  return "welcome";  } else {  model.put("errorMessage", "Invalid Credentials!!");  return "login";  }  }  Files List  pom.xml: Same dependencies-No change  In POM.xml: We have four main tags  1- Parent tag- spring-boot-starter-parent  2- **Properties**- java versiontags  3 -Dependencies-a- spring-boot-starter-web  tomcat-embed-jasper  spring-boot-devtools  spring-boot-starter-test  4-build maven plugin- spring-boot-maven-plugin  File – Java , properties and jsp   * src/main/java/com/in28minutes/springboot/web/controller/LoginController.java * src/main/java/com/in28minutes/springboot/web/service/LoginService.java * src/main/java/com/in28minutes/springboot/web/SpringBootFirstWebApplication.java * src/main/resources/application.properties * src/main/webapp/WEB-INF/jsp/login.jsp * src/main/webapp/WEB-INF/jsp/welcome.jsp * src/test/java/com/in28minutes/springboot/web/SpringBootFirstWebApplicationTests.java   src/main/java/com/in28minutes/springboot/web/controller/LoginController.java  package com.in28minutes.springboot.web.controller;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.ModelMap;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RequestParam;  import com.in28minutes.springboot.web.service.LoginService;  @Controller  public class LoginController {    @Autowired  LoginService service;    @RequestMapping(value="/login", method = RequestMethod.GET)  public String showLoginPage(ModelMap model){  return "login";  }  @RequestMapping(value="/login", method = RequestMethod.POST)  public String showWelcomePage(ModelMap model, @RequestParam String name, @RequestParam String password){  boolean isValidUser = service.validateUser(name, password);  if (!isValidUser) {  model.put("errorMessage", "Invalid Credentials");  return "login";  }  model.put("name", name);  model.put("password", password);    return "welcome";  }  }  src/main/java/com/in28minutes/springboot/web/service/LoginService.java  package com.in28minutes.springboot.web.service;  import org.springframework.stereotype.Component;  @Component  public class LoginService {  public boolean validateUser(String userid, String password) {  // in28minutes, dummy  return userid.equalsIgnoreCase("in28minutes") && password.equalsIgnoreCase("dummy");  }  }  src/main/resources/application.properties  spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp  logging.level.org.springframework.web=DEBUG  src/main/webapp/WEB-INF/jsp/login.jsp  <html>  <head>  <title>First Web Application</title>  </head>  <body>  <font color="red">${errorMessage}</font>  <form method="post">  Name : <input type="text" name="name" />  Password : <input type="password" name="password" />  <input type="submit" />  </form>  </body>  </html>  src/main/webapp/WEB-INF/jsp/welcome.jsp  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Welcome ${name}!!  </body>  </html>  **todo.txt**  Component, Service, Repository, Controller  Autowired  ComponentScan  Spring Boot Starter Parent  Spring Boot Starter Web  @SpringBootApplication  Auto Configuration  Dispatcher Servlet  /login => "login"  "login" => src/main/webapp/WEB-INF/jsp/login.jsp  Search for a view named "login"  /login => LoginController |

## ****Step09-****

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step09.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step09.md)

What You Will Learn during this Step:

* Magic of Spring
* Learn about Spring Auto-wiring and Dependency Management.
* @Autowired, @Component, @Repository, @Controller, @ComponentScan

In the last step we have create one component service and we Autowired it into LoginController. Let’s learn some more about auto wiring and dependency injection. Spring is one of most imp dependency injection framework. Before dependency injection it was very difficult to do unit testing. The only way was that we had to deploy the code into server and then test it. Hence everything was tightly coupled (like earlier we had to write fully [ClassName obj = new ClassName();]. So to make it loosely coupled spring brought dependency injection which makes it very easy to do unit testing.

If we mention **@Component** annotation at any class then it means we are requesting spring framework to manage its object initiation and creation. Similarly we have another annotation like [@Service, @Repository @Controller etc.] if we mention these annotation at class level then spring will take care of its object creation and initiation.

**@ Controller** is a specialization of @Component annotation allowing for implementation classes to be autodetected through classpath scanning i.e. @Controller is typically used specially in case of MVC but @Component @Service, @Repository is being used mostly everywhere.

**@Component** or **@Service** can be used for business service. @Repository can be used in terms of data store or database operations. So these four annotations are being managed by spring itself.

**Now let’s talk about @Autowired:** It is generally used inside the class [like inside controller class] so once the bean is created, spring will Autowired automatically wherever it is needed [like inside the controller class]

**@ComponentScan**: Generally **@SpringBootApplication** annotated class Indicates a [configuration](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.4.0.RELEASE%5C/spring-boot-autoconfigure-1.4.0.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82Configuration) class which triggers [auto-configuration](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.4.0.RELEASE%5C/spring-boot-autoconfigure-1.4.0.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82EnableAutoConfiguration) and [component scanning](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/boot%5C/spring-boot-autoconfigure%5C/1.4.0.RELEASE%5C/spring-boot-autoconfigure-1.4.0.RELEASE.jar%3Corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication%E2%98%82ComponentScan). This is a convenience annotation that is equivalent to declaring @Configuration, @EnableAutoConfiguration and @ComponentScan. Generally scanning happens in the same package defined for @SpringBootApplication class for all the @Component, @Service, @Repository, and @Controller. But if want to scan these annotation in another different package then we have to explicitly mention the package name to be scan. If we hover the mouse on @SpringBootApplication we will found that @ComponentScan comes under @SpringBootApplication and by default it does the scanning into main package and its sub packages.

|  |
| --- |
| package com.in28minutes.springboot.web;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class SpringBootFirstFormWebApplication {  public static void main(String[] args) {  SpringApplication.run(SpringBootFirstFormWebApplication.class, args);  }  }  So here the default scanning will be done for com.in28minutes.springboot.web So the @Component will be scanned and initialized for auotwiring if it is available in the com.in28minutes.springboot.web package.  But if @Component is available in some other package then we need to explicitly mention to be scanned.  **The best way is we always mention base package so that the entire package associated with it will be the part of this and will be available for auto scan.**  So to make available auto scan in the entire package we will mention the packaging like given below.  @ComponentScan (com.in28minutes). This declaration will scan this package as well as its associated packages. |

This is the main basic of spring.

## ****Step-11****

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step11.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step11.md)

**Let’s discuss about Architecture of web applications**

**Java web application is based on two model architecture.**

**1) Model1- In this model all the layer (Model, View, Controller) was being written in jsp only. The request of browser was sent to Jsp, the request was being processed in jsp only and finally model data was being sent in the view written in the jsp. In this way all the layer and logic was written in the JSP only and which was very difficult to maintain.**

**2) Model2- In the model2 we have three separate layers (Model, View, Controller). In this architecture the request of browser was sent to java Servlet and Servlet used to process the request and after processing the request the model (business data) was sent to view layer (jsp).**

**3)Model2 Enhancement- In model2 all the request was sent to on single Servlet to get response and let say if we have another request to be processed then we need to write another Servlet and need to map this Servlet response in the view accordingly. In short way we can say that In Modle2 we had only JSP-Servlet concept and there were no concept of on centralized front controller.**

**In this enhance model2, the centralized front controller (DispatcherServlet) concept came, which was mapping and handling the controller and view.**

**Java web application architecture we used to have following layers. (1) Web or view Layer 2) Business or model layer 3) Data Layer 3) integration layer.**

## ****Step10-****

## <https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step10.md>

What we will do:

* Create TodoController and list-todos.jsp
* Make TodoService a @Service and inject it.
* We will perform add, update and delete operation.
* For performing above operation we need to have some business logic. So to perform business logic we will use in memory data base like array list.
* We will include these operation in TodoService.java file

Pending for Next Step

* ${name} is not available in list-todos.jsp
* in28Minutes is hardcoded in TodoController

**Files List**

pom.xml: Same dependencies-No change

In POM.xml: We have four main tags

1- Parent tag- spring-boot-starter-parent

2- **Properties**- java version tags

3 -Dependencies-a- spring-boot-starter-web

tomcat-embed-jasper

spring-boot-devtools

spring-boot-starter-test

4-build maven plugin- spring-boot-maven-plugin

File – Java , properties and jsp

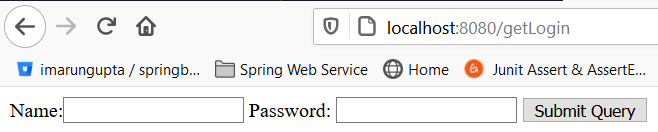
* 1. /src/main/java/com/in28minutes/springboot/web/model/Todo.java (Model class having variable and getters,setters)
  2. /src/main/java/com/in28minutes/springboot/web/service/TodoService.java (Business logic having add, update, delete, retrieve operations)
  3. /src/main/java/com/in28minutes/springboot/web/controller/TodoController.java(Here TodoService will be injected to get all the operation values in controller class
  4. src/main/webapp/WEB-INF/jsp/todo-list.jsp( This jsp will used to display todo list using model object passed from controller class

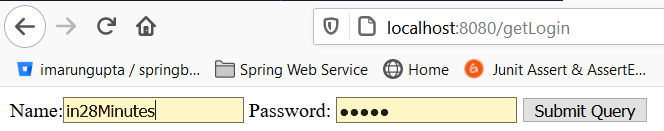
## Snippets

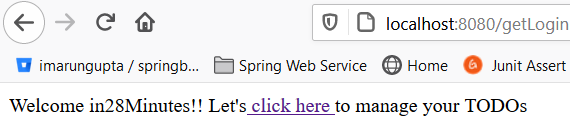
Snippet - /src/main/java/com/in28minutes/springboot/web/model/Todo.java

|  |
| --- |
| package com.in28minutes.springboot.web.model;  import java.util.Date;  public class Todo {  private int id;  private String user;  private String desc;  private Date targetDate;  private boolean isDone;  public Todo(int id, String user, String desc, Date targetDate,  boolean isDone) {  super();  this.id = id;  this.user = user;  this.desc = desc;  this.targetDate = targetDate;  this.isDone = isDone;  }  // Getters & Setters  @Override  public int hashCode() {  final int prime = 31;  int result = 1;  result = prime \* result + id;  return result;  }  @Override  public boolean equals(Object obj) {  if (this == obj) {  return true;  }  if (obj == null) {  return false;  }  if (getClass() != obj.getClass()) {  return false;  }  Todo other = (Todo) obj;  if (id != other.id) {  return false;  }  return true;  }  @Override  public String toString() {  return String.format(  "Todo [id=%s, user=%s, desc=%s, targetDate=%s, isDone=%s]", id,  user, desc, targetDate, isDone);  }  } |
| Snippet - /src/main/java/com/in28minutes/springboot/web/service/TodoService.java  package com.in28minutes.springboot.web.service;  import java.util.ArrayList;  import java.util.Date;  import java.util.Iterator;  import java.util.List;  import org.springframework.stereotype.Service;  import com.in28minutes.springboot.web.model.Todo;  @Service  public class TodoService {  private static List<Todo> todos = new ArrayList<Todo>();  private static int todoCount = 3;  static {  todos.add(new Todo(1, "in28Minutes", "Learn Spring MVC", new Date(),  false));  todos.add(new Todo(2, "in28Minutes", "Learn Struts", new Date(), false));  todos.add(new Todo(3, "in28Minutes", "Learn Hibernate", new Date(),  false));  }  public List<Todo> retrieveTodos(String user) {  List<Todo> filteredTodos = new ArrayList<Todo>();  for (Todo todo : todos) {  if (todo.getUser().equals(user)) {  filteredTodos.add(todo);  }  }  return filteredTodos;  }  public void addTodo(String name, String desc, Date targetDate,  boolean isDone) {  todos.add(new Todo(++todoCount, name, desc, targetDate, isDone));  }  public void deleteTodo(int id) {  Iterator<Todo> iterator = todos.iterator();  while (iterator.hasNext()) {  Todo todo = iterator.next();  if (todo.getId() == id) {  iterator.remove();  }  }  }  } |

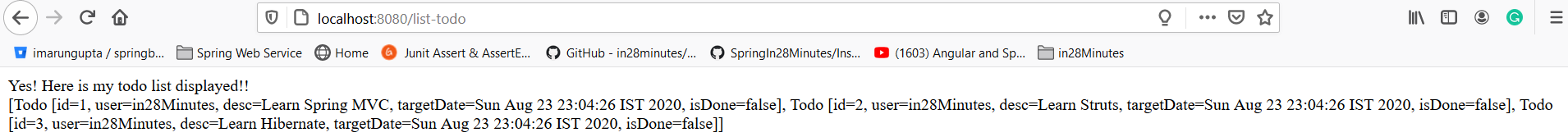
|  |
| --- |
| **package** com.in28minutes.springboot.web.controller;  **import** org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** com.in28minutes.springboot.web.service.TodoService;  @Controller  **publicclass** TodoController {    @Autowired  TodoService service;  @RequestMapping(value="/list-todo", method=RequestMethod.***GET***)  **public** String showTodos(ModelMap model) {  System.***out***.println("Inside list todo mentod controller");  model.put("todo", service.retrieveTodos("in28Minutes"));  **return**"todo-list";  }  } |
| <html>  <head>  <title>First Web Application</title>  </head>  <body>  Yes! Here is my todo list displayed!!</br>  ${todo}</br>  </body>  </html> |

****

****

****

**List of todo:**

****

## ****Step12-****

## [spring-boot-master-class](https://github.com/in28minutes/spring-boot-master-class)/[02.Spring-Boot-Web-Application](https://github.com/in28minutes/spring-boot-master-class/tree/master/02.Spring-Boot-Web-Application)/****Step12.md****

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step12.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step12.md)

## What You Will Learn during this Step:

* Session vs Model vs Request.
* Be cautious about what you use Session for.
* @SessionAttributes("name") and how it works?
* Why use Model? "Adding elements directly to the HttpServletRequest (as request attributes) would seem to serve the same purpose. The reason to do this is obvious when taking a look at one of the requirements we have set for the MVC framework: It should be as view-agnostic as possible, which means we’d like to be able to incorporate view technologies not bound to the HttpServletRequest as well." - Rod Johnson et. al’s book Professional Java Development with the Spring Framework
* Spring documentation states that the @SessionAttributes annotation “list the names of model attributes which should be transparently stored in the session or some conversational storage.”

Useful Snippets and References

First Snippet : @SessionAttributes("name")

Files List

pom.xml: Same dependencies-No change

In POM.xml: We have four main tags

1- Parent tag- spring-boot-starter-parent

2- **Properties**- java versiontags

3 -Dependencies-a- spring-boot-starter-web

tomcat-embed-jasper

spring-boot-devtools

spring-boot-starter-test

4-build maven plugin- spring-boot-maven-plugin

File – Java, properties and jsp

1. src/main/java/com/in28minutes/springboot/web/SpringBootFirstWebApplication.java
2. src/main/resources/application.properties
3. src/main/java/com/in28minutes/springboot/web/controller/LoginController.java
4. src/main/java/com/in28minutes/springboot/web/controller/TodoController.java
5. src/main/java/com/in28minutes/springboot/web/model/Todo.java
6. src/main/java/com/in28minutes/springboot/web/service/LoginService.java
7. src/main/java/com/in28minutes/springboot/web/service/TodoService.java
8. src/main/webapp/WEB-INF/jsp/login.jsp
9. src/main/webapp/WEB-INF/jsp/welcome.jsp
10. src/main/webapp/WEB-INF/jsp/list-todos.jsp
11. src/test/java/com/in28minutes/springboot/web/SpringBootFirstWebApplicationTests.java

|  |
| --- |
| **Spring Boot Launcher class: To launch the application**  package com.in28minutes.springboot.web;  import org.springframework.boot.SpringApplication;  import org.springframework.boot.autoconfigure.SpringBootApplication;  @SpringBootApplication  public class SpringBootFirstFormWebApplication {  public static void main(String[] args) {  SpringApplication.run(SpringBootFirstFormWebApplication.class, args);  }  } |
| **Application.properties: Mentioned the view prefix, suffix and logging level**  spring.mvc.view.prefix=/WEB-INF/jsp/  spring.mvc.view.suffix=.jsp  logging.level.org.springframework.web=DEBUG |
| **Creating LoginService.java : this service will be used to authenticate user credential and return true if credential matches**  **package** com.in28minutes.springboot.web.service;  **import** org.springframework.stereotype.Component;  @Component  **publicclass** LoginService {  **publicboolean** validateUser(String userid, String password) {  **return** userid.equalsIgnoreCase("in28Minutes") &&password.equalsIgnoreCase("dummy");  }  } |
| **Create a** LoginController.java class to consume LoginService.java and rendering the welcome page after successfully login.  **package com.in28minutes.springboot.web.controller;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.beans.factory.parsing.PassThroughSourceExtractor;**  **import org.springframework.stereotype.Controller;**  **import org.springframework.ui.ModelMap;**  **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.bind.annotation.SessionAttributes;**  **import com.in28minutes.springboot.web.service.LoginService;**  **@Controller**  **@SessionAttributes("nameKey")**  **public class LoginController {**    **@Autowired**  **LoginService loginservice;**    **@RequestMapping(value="/getLogin", method=RequestMethod.GET)**  **public String loginMessage( ModelMap model) {**  **System.out.println("inside login controll");**  **return"login";**  **}**  **@RequestMapping(value="/getLogin", method=RequestMethod.POST)**  **public String showWelcomePage(@RequestParam String name,@RequestParam String password,ModelMap model) {**  **boolean isValidate=loginservice.validateUser(name, password);**  **if(!isValidate) {**  **model.put("errorMsg", "Invalid Credentials");**  **return"login";**  **}else**  **model.put("nameKey", name);**  **model.put("passKey", password);**    **return"welcome";**  **}**  **}**  In the controller class we have Autowired loginService component class for validating credential. If credential does not matches then return login page with error message. Else return welcome page and putting username and password into model object to be displayed on the view page.  Since request object or model object scope is available till the current request so we have kept name into SessionAttribute object so that we could use name object till application session scope. |
| **Login.jsp**  <html>  <head>  <title>SpringBootFirstWebApplication</title>  </head>  <body>  <fontcolor=*"red"*>${errorMsg}</font>  <formaction=*""*method=*"post"*>  Name:<inputtype=*"text"*name=*"name"*id=*"name"*/>  Password: <inputtype=*"password"*name=*"password"*id=*"password"*/>  <inputtype=*"submit"*>  </form>  </body>  </html> |
| **Welcome.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Welcome ${nameKey}!! Let's<a href=*"/list-todo"*> click here </a>to manage your TODOs  </body>  </html> |
| **Creating Model class :**  **package** com.in28minutes.springboot.web.model;  **import** java.util.Date;  **public class** Todo {  **private int** id;  **private** String user;  **private** String desc;  **private** Date targetDate;  **private boolean** isDone;  **public** Todo(**int** id, String user, String desc, Date targetDate,  **boolean** isDone) {  **super**();  **this**.id = id;  **this**.user = user;  **this**.desc = desc;  **this**.targetDate = targetDate;  **this**.isDone = isDone;  System.***out***.println("Called TODO Model class");  }  // Getters & Setters  @Override  **publicint** hashCode() {  **finalint**prime = 31;  **int**result = 1;  result = prime \* result + id;  **return**result;  }  @Override  **publicboolean** equals(Object obj) {  **if** (**this** == obj) {  **returntrue**;  }  **if** (obj == **null**) {  **returnfalse**;  }  **if** (getClass() != obj.getClass()) {  **returnfalse**;  }  Todoother = (Todo) obj;  **if** (id != other.id) {  **returnfalse**;  }  **returntrue**;  }  @Override  **public** String toString() {  **return** String.*format*(  "Todo [id=%s, user=%s, desc=%s, targetDate=%s, isDone=%s]", id,  user, desc, targetDate, isDone);  }  } |
| **TodoService.java class used to add id, user, desc, targetDate into static list.**  **package com.in28minutes.springboot.web.service;**  **import java.util.ArrayList;**  **import java.util.Date;**  **import java.util.Iterator;**  **import java.util.List;**  **import org.springframework.stereotype.Service;**  **import com.in28minutes.springboot.web.model.Todo;**  **@Service**  **public class TodoService {**  **private static List<Todo> todos = new ArrayList<Todo>();**  **private static int todoCount = 3;**  **// Static block to add id,user,desc,targetDate into list object.**  **static {**  **todos.add(new Todo(1, "in28Minutes", "Learn Spring MVC", new Date(),false));**  **todos.add(new Todo(2, "in28Minutes", "Learn Struts", new Date(), false));**  **todos.add(new Todo(3, "in28Minutes", "Learn Hibernate", new Date(),false));**  **}**  **// Method to retrieve the values from static list and putting into filteredTodos list which will be used in the controller class to display.**  **public List<Todo> retrieveTodos(String user) {**  **List<Todo> filteredTodos = new ArrayList<Todo>();**  **for (Todo todo : todos) {**  **if (todo.getUser().equals(user)) {**  **filteredTodos.add(todo);**  **}**  **}**  **return filteredTodos;**  **}**  **// Method to add new record into static todo list object**  **public void addTodo(String name, String desc, Date targetDate,**  **boolean isDone) {**  **todos.add(new Todo(++todoCount, name, desc, targetDate, isDone));**  **}**  **public void deleteTodo(int id) {**  **Iterator<Todo> iterator = todos.iterator();**  **while (iterator.hasNext()) {**  **Todo todo = iterator.next();**  **if (todo.getId() == id) {**  **iterator.remove();**  **}**  **}**  **}**  **}** |
| TodoController.java class  **package** com.in28minutes.springboot.web.controller;  **import**org.springframework.beans.factory.annotation.Autowired;  **import** org.springframework.stereotype.Controller;  **import** org.springframework.ui.ModelMap;  **import** org.springframework.web.bind.annotation.RequestMapping;  **import** org.springframework.web.bind.annotation.RequestMethod;  **import** org.springframework.web.bind.annotation.SessionAttributes;  **import** com.in28minutes.springboot.web.service.TodoService;  @Controller  @SessionAttributes("nameKey")  **publicclass** TodoController {    @Autowired  TodoService service;  @RequestMapping(value="/list-todo", method=RequestMethod.***GET***)  **public** String showTodos(ModelMap model) {  System.***out***.println("Inside list todo mentod controller");  String userFromSession = (String) model.get("nameKey");  System.***out***.println("userFromSession"+userFromSession);  //model.put("todo", service.retrieveTodos("in28Minutes"));  model.put("todo", service.retrieveTodos(userFromSession));    **return**"todo-list";  }  }   * 1. **Autowired TodoService component class to consume their methods**   2. **Get the nameKey from SessionAttributes**   3. **Retrieving the todo list and putting into model object.**   4. **Finally return the todo-list view where all the todo list is being displayed** |
| **Welcome.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Welcome ${nameKey}!! Let's<ahref=*"/list-todo"*> click here </a>to manage your TODOs  </body>  </html>  Here as we can see that we have created on href link [click here] which will call to /list-todo method and in this method it is returning [todo-list.jsp] to view the list of todos. |
| **Todo-list.jsp**  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Yes! Here is my todo list displayed!!</br>  ${todo}</br>  your name is${nameKey}  </body>  </html> |
|  |
| **If we provide correct credential then:** |
| **No on clicking on** click herewe get list of todos |

## ****Step13-****

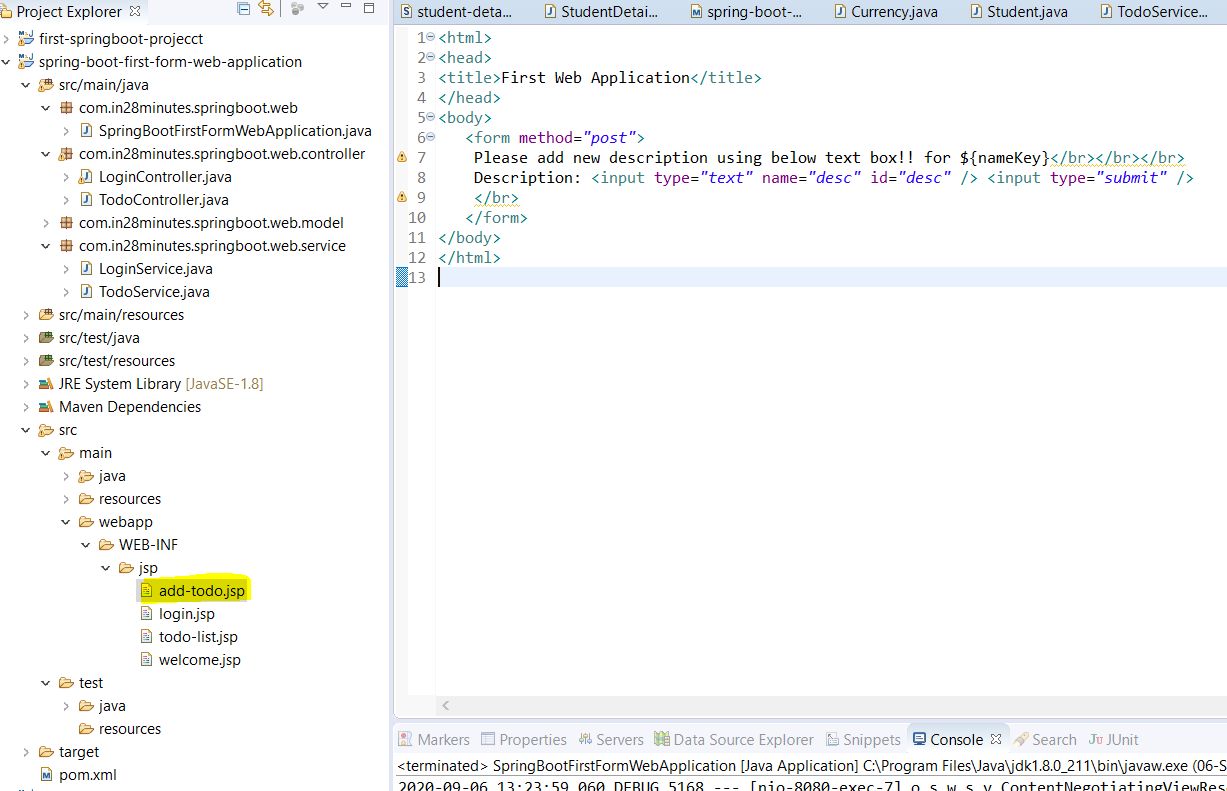
## https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step13.md

Now the next requirement is to add new description: And for that

* 1. In the **list-todo.jsp** create a link to add **<Add a Todo>**
  2. Create a jsp page(add-todo.jsp)having one text box and submit button to add todo which will be render on clicking of **<Add a Todo>** link
  3. In the **TodoService.java create a method to add description in the todo list.**
  4. Now in the TodoController class consume this method which will add desc in the todo list and after adding display the list-todo.jsp having newly added description as well.

|  |
| --- |
| Todo-list.jsl  <html>  <head>  <title>First Web Application</title>  </head>  <body>  Yes! Here is my todo list displayed!!</br>  ${todo}</br>  your name is${nameKey}  </br>  <a href=*"/add-todo"*>Add a Todo</a>  </body>  </html> |
| Add-todo.jsp  <html>  <head>  <title>First Web Application</title>  </head>  <body>  <form method=*"post"*>  Please add new description using below text box!! for ${nameKey}</br></br></br>  Description: <input type=*"text"*name=*"desc"*id=*"desc"*/><input type=*"submit"*/>  </br>  </form>  </body>  </html> |
| TodoService.java  **publicvoid**addTodo(String name, String desc, Date targetDate,  **boolean**isDone) {  *todos*.add(**new** Todo(++*todoCount*, name, desc, targetDate, isDone));  } |
| TodoController.java  @RequestMapping(value="/add-todo", method=RequestMethod.***GET***)  **public**String showAddTodoPage(ModelMap model) {  **return**"add-todo";    }  @RequestMapping(value="/add-todo", method=RequestMethod.***POST***)  **public** String addTodos(ModelMap model, @RequestParam String desc) {  System.***out***.println("Inside list todo mentod controller");  String userFromSession = (String) model.get("nameKey");  System.***out***.println("userFromSession"+userFromSession);  //model.put("todo", service.retrieveTodos("in28Minutes"));  service.addTodo(userFromSession, desc, **new** Date(), **false**);    //return "todo-list";  /\*  \* Here the above line we were trying to render the todo-list page having todo  \* list with newly added list. But the list was empty because here it was just  \* rendering jsp only so the solution is given below. Instead of using jsp page  \* we will use the request mapping url which populate the call the method to get  \* todo-list and display the same into todo-listjsp  \*/  **return**"redirect:/list-todo";  } |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |



## ****Step14.md****

## https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step14.md

## What we will do:

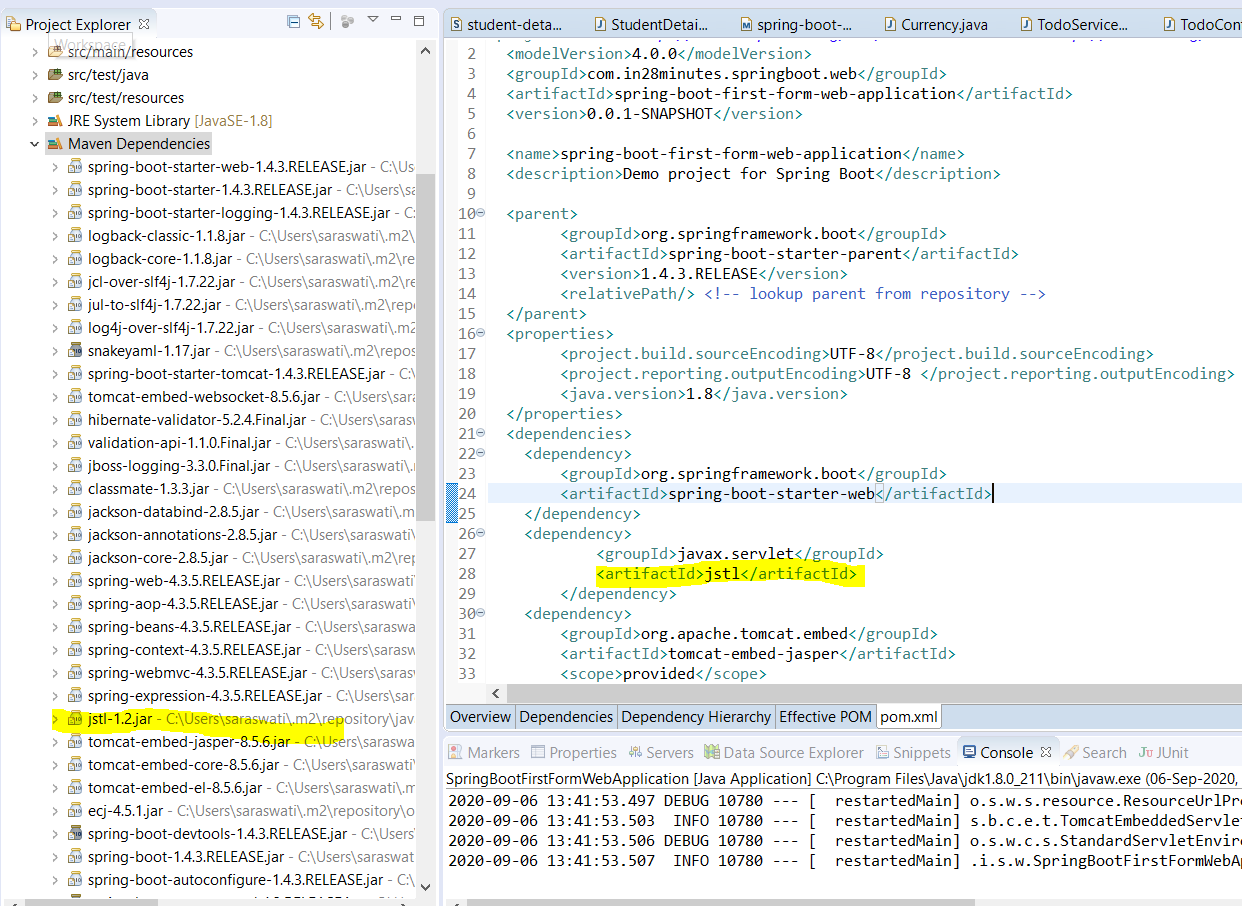
* Display Todos in a table using **JSTL Tags (Java Standard Tag Library). It provides lots of utility tag in jsp**
* <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c"%>
* Add Dependency for jstl
* To enable JSTL tag add below dependency in POM.xml

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

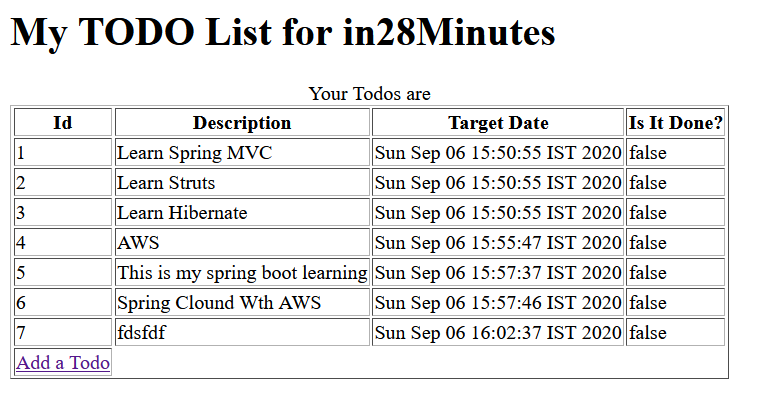


Note: After adding any dependency we need to restart the server.

Here JSTL mail role is to put the entire list into tabular format i.e. for better view of list of records. So mainly the changes will be happened in todo-list.jsp page as given below

* 1. Define jstl tag library on top of the jsp page.
  2. <%@ taglib uri="<http://java.sun.com/jsp/jstl/core>" prefix="c"%>
  3. First we will create one table and will create Table caption, table header and row inside the header.

|  |
| --- |
| <%@tagliburi=*"http://java.sun.com/jsp/jstl/core"*prefix=*"c"*%>  <html>  <head>  <title>First Web Application TODO for ${nameKey}</title>  </head>  <body>  <h1>My TODO List for ${nameKey}</h1>  <tableborder=*"1"*>  <caption>Your Todos are</caption>  <thead>  <tr>  <th>Id</th>  <th>Description</th>  <th>Target Date</th>  <th>Is It Done?</th>  </tr>  </thead>  <tbody>  <!-- use JStl for loop todo list. We will use forEach loop-->  <!-- forEach items in todo list create one variable todoItem -->  <!-- using var name access the model class variable just like calling getter method to get values -->  <c:forEach items=*"*${todo}*"*var=*"todoItem"*>  <tr>  <td>${todoItem.id}</td>  <td>${todoItem.desc}</td>  <td>${todoItem.targetDate}</td>  <td>${todoItem.done}</td>  </tr>  </c:forEach>  </tbody>  <tr>  <td><ahref=*"/add-todo"*>Add a Todo</a></td>  </tr>  </table>  <%-- Yes! Here is my todo list displayed!!</br>  ${todo}</br>  your name is${nameKey} --%>    <!-- <a href="/add-todo">Add a Todo</a> -->  </body>  </html> |



**Step15.md**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step15.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/02.Spring-Boot-Web-Application/Step15.md)

What we will do:

* Add bootstrap to give basic formatting to the page: We use bootstrap classes container, table and table-striped.
* We will use webjars

It is already auto configured by Spring Boot : o.s.w.s.handler.SimpleUrlHandlerMapping :

Mapped URL path [/webjars/\*\*] onto handler of type [class.org.springframework.web.servlet.resource.ResourceHttpRequestHandler]

Useful Snippets

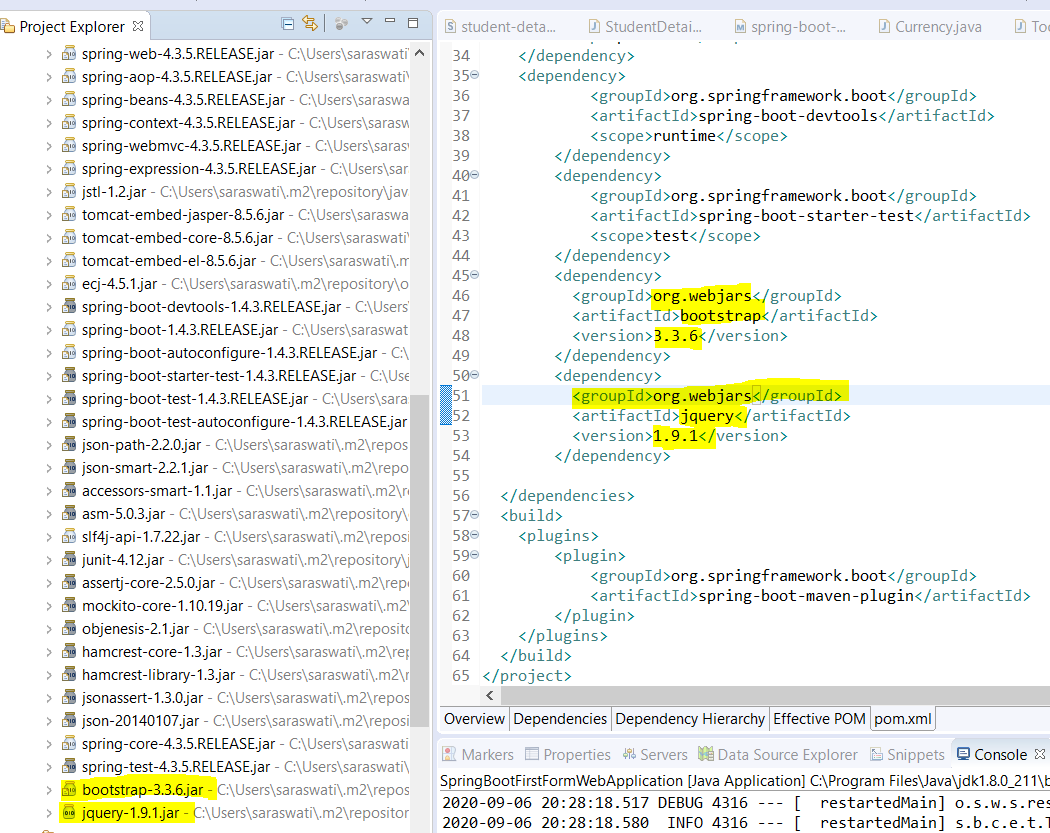
|  |
| --- |
| <dependency>  <groupId>org.webjars</groupId>  <artifactId>bootstrap</artifactId>  <version>3.3.6</version>  </dependency>  <dependency>  <groupId>org.webjars</groupId>  <artifactId>jquery</artifactId>  <version>1.9.1</version>  </dependency>    <script src="webjars/jquery/1.9.1/jquery.min.js"></script>  <script src="webjars/bootstrap/3.3.6/js/bootstrap.min.js"></script>  <link href="webjars/bootstrap/3.3.6/css/bootstrap.min.css"rel="stylesheet"> |

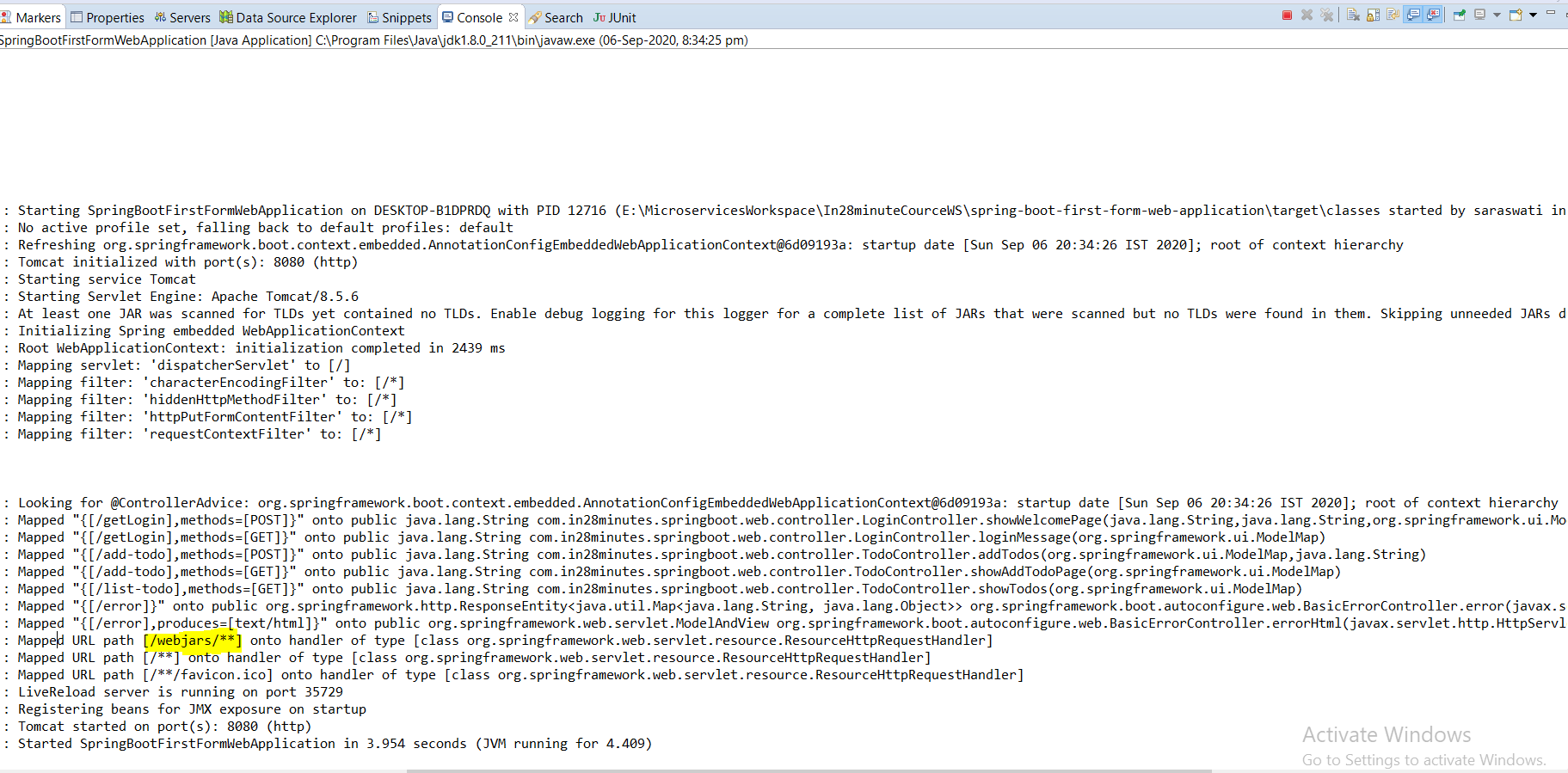
**WebJars concept** Generally if we want use spring boot framework then usually we need to add spring boot parent and some of its dependency which get downloaded automatically **similarly** if we want to use java script (e.g. JQuery) or CSS file in our application then we need to download the source file and need to put into WEB-INF folder so that we can use its features into html or JSP file.

Now if we want to upgrade this java script and CSS in future then we need to download the latest version and need to put physically in the application inside WEB-INF folder of the application.

To avoid this headache, WebJars concept came into picture. Now it came into maven dependency which we can put into POM.xml and whenever we want to upgrade, we just need to change version into POM.xml file and everything will be take care automatically. Using this WebJars concept we can upgrade the version of these static files (java script and CSS)

After adding the above two dependency as soon as we restart the server these two dependency gets added in the maven dependency as shown in the below image.





**As here in console we can see that WebJars is auto configured which spring boot take care for this auto configuration.**

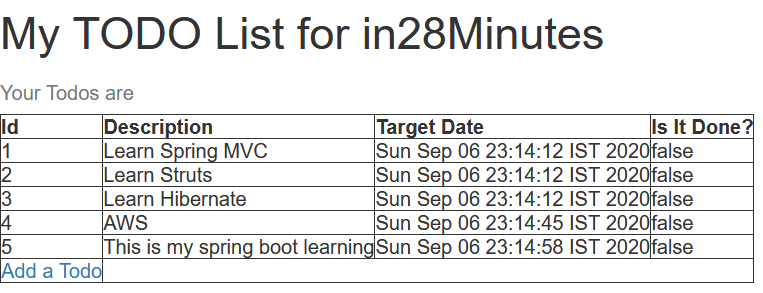
To be able to use bootstrap we need JQuery that is why we have added in the POM.xml and to be able to use these we need to link boot strap and JQuery in our JSP (todo-list.jsp). The best practice is that we will add these scripts at the bottom of the page. And CSS will be added inside the <head> tag of the JSP. So that during the page load CSS could be loaded ASAP and java script could be loaded as late as possible.

The path of the java script and CSS file can be seen in maven dependency by expanding the particular jar.

|  |  |
| --- | --- |
|  |  |

So to add boot strap we need to add these two java script and one CSS as mentioned in the JSP.

**Now after adding these two script and css file, once we reload the page then the outcome will look like as below.**



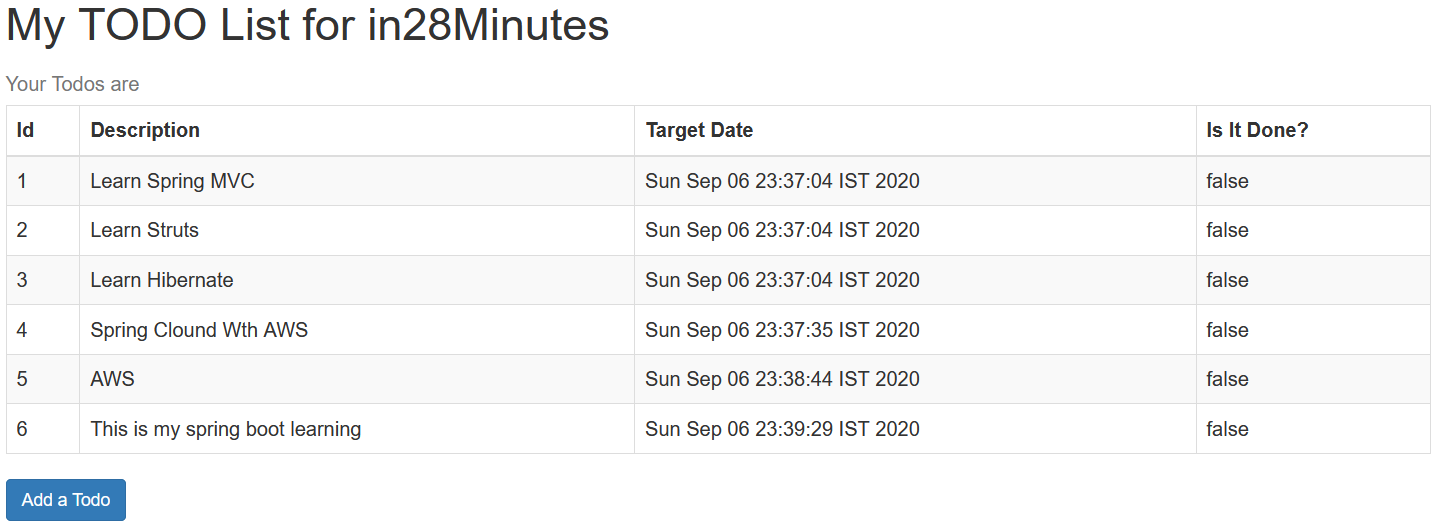
Now to make it more good looking let’s use some more boot strap classes.

* One of the important thing about boot strap is everything will put inside <div> tag and inside this tag we will use CSS class =”container” [<divclass=*"container"*>]
* The other thing inside <table> tag we can use the class=”table” i.e. <table class=”table”>. To make it better we can use class=”table-striped” and for table border we can use class=”*table-bordered”*

So all the table class can be mentioned as <tableclass=*"table table-striped table-bordered"*>

* For button <div><aclass=*"btn btn-primary"*href=*"/add-todo"*>Add a Todo</a></div>

|  |
| --- |
| <%@tagliburi=*"http://java.sun.com/jsp/jstl/core"*prefix=*"c"*%>  <html>  <head>  <title>First Web Application TODO for ${nameKey}</title>  <linkhref=*"webjars/bootstrap/3.3.6/css/bootstrap.min.css"*  rel=*"stylesheet"*>  </head>  <body>  <divclass=*"container"*>  <h1class=*"text-justify"*>My TODO List for ${nameKey}</h1>  <tableclass=*"table table-striped table-bordered"*>  <caption>Your Todos are</caption>  <thead>  <tr>  <th>Id</th>  <th>Description</th>  <th>Target Date</th>  <th>Is It Done?</th>  </tr>  </thead>  <tbody>  **<!-- use JStl for loop todo list. We will use forEach loop-->**  **<!-- forEach items in todo list create one variable todoItem -->**  **<!-- using var name access the model class variable just like calling getter method to get values -->**  <c:forEachitems=*"*${todo}*"*var=*"todoItem"*>  <tr>  <td>${todoItem.id}</td>  <td>${todoItem.desc}</td>  <td>${todoItem.targetDate}</td>  <td>${todoItem.done}</td>  </tr>  </c:forEach>  </tbody>  </table>  <div>  <aclass=*"btn btn-primary"*href=*"/add-todo"*>Add a Todo</a>  </div>  <scriptsrc=*"webjars/jquery/1.9.1/jquery.min.js"*></script>  <scriptsrc=*"webjars/bootstrap/3.3.6/js/bootstrap.min.js"*></script>  </div>  </body>  </html> |



**Hence using Bootstrap CSS framework we have formatted our page very well.**