**Step List**

* **Step 01: Setup and Launch Spring Boot Application with Maven and Eclipse**
* **Step 02: Creating your first RestController**
* **Step 03: Understanding Spring Boot Magic : Spring Boot Starter Web**
* **Step 04: Understanding Spring Boot Magic : Spring Boot Starter Parent**
* **Step 05: Spring Boot vs Spring**
* **Step 06: Create all Services for Survey and Questions**
* **Step 07: What is REST? Creating REST Service with @GetMapping and @PathVariable**
* **Step 08: Second REST Service to retrieve a specific question**
* **Step 09: Spring Boot Developer Tools and LiveReload : Develop faster!**
* **Step 10: Create a REST Service to add a new question to survey : @PostMapping, Postman**
* **Step 11: Understand Content Negotiation. Deliver XML Responses from the REST Services**
* **Step 12: Spring Initializr : Create Spring Boot Projects on the fly!**
* **Step 13: Spring Boot Actuator : Monitor your Spring Boot applications!**
* **Step 14: Understanding Embedded servlet containers : Switch to Jetty or Undertow**
* **Step 15: Adding Dynamic Configuration to your application : YAML & More..**
* **Step 16: Basics of Profiles**
* **Step 17: Advanced Application Configuration with Type Safe Configuration Properties**
* **Step 18: Spring Boot Starter : Spring Data JPA with CommandLineRunner**
* **Step 19: In Memory Database H2 Console and add a new JPA Repository Method**
* **Step 20: Spring Boot Starter : Introduction to Spring Data Rest**
* **Step 21: Spring Boot Integration Test**
* **Step 22: Adding Integration Test for POST Request**
* **Step 23: Small Refactoring to organise ourselves**
* **Step 24: Writing Unit Tests with Spring Boot and Mockito**
* **Step 25: Writing Unit test for createTodo**
* **Step 26: Securing our services with Basic Authentication using Spring Security**
* **Step 27: Configure Spring Security user roles for survey and other services**
* **Step 28: A Deep Dive into Spring Boot Auto Configuration**

## Starting with Spring Initializr

**For all Spring applications, you should start with the**[**Spring Initializr**](https://start.spring.io/)**. The Initializr offers a fast way to pull in all the dependencies you need for an application and does a lot of the setup for you. This example needs only the Spring Web dependency.**

**You can get a Maven build file with the necessary dependencies directly from the**[**Spring Initializr**](https://start.spring.io/#!type=maven-project&language=java&platformVersion=2.4.2.RELEASE&packaging=jar&jvmVersion=1.8&groupId=com.example&artifactId=testing-web&name=testing-web&description=Demo%20project%20for%20Spring%20Boot&packageName=com.example.testing-web&dependencies=web)**. The following listing shows the pom.xml file that is created when you choose Maven:**

**<?xml version="1.0" encoding="UTF-8"?>**

**<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"**

**xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">**

**<modelVersion>4.0.0</modelVersion>**

**<parent>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-parent</artifactId>**

**<version>2.4.2</version>**

**<relativePath/> <!-- lookup parent from repository -->**

**</parent>**

**<groupId>com.example</groupId>**

**<artifactId>testing-web</artifactId>**

**<version>0.0.1-SNAPSHOT</version>**

**<name>testing-web</name>**

**<description>Demo project for Spring Boot</description>**

**<properties>**

**<java.version>1.8</java.version>**

**</properties>**

**<dependencies>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-web</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-test</artifactId>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<plugins>**

**<plugin>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-maven-plugin</artifactId>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

## Create a Simple Application

**Create a new controller for your spring application. The following listing (from src/main/java/com/example/testingweb/HomeController.java) shows how to do so:**

**package com.example.testingweb;**

**import org.springframework.stereotype.Controller;**

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

**import org.springframework.web.bind.annotation.ResponseBody;**

**@Controller**

**public class HomeController {**

**@RequestMapping("/")**

**public @ResponseBody String greeting() {**

**return "Hello, World";**

**}**

**}COPY**

|  |  |
| --- | --- |
|  | **The preceding example does not specify GET versus PUT, POST, and so forth. By default @RequestMapping maps all HTTP operations. You can use @GetMapping or @RequestMapping(method=GET) to narrow this mapping.** |

## Run the Application

**The Spring Initializr creates an application class (a class with a main() method) for you. For this guide, you need not modify this class. The following listing (from src/main/java/com/example/testingweb/TestingWebApplication.java) shows the application class that the Spring Initializr created:**

**package com.example.testingweb;**

**import org.springframework.boot.SpringApplication;**

**import org.springframework.boot.autoconfigure.SpringBootApplication;**

**@SpringBootApplication**

**public class TestingWebApplication {**

**public static void main(String[] args) {**

**SpringApplication.run(TestingWebApplication.class, args);**

**}**

**}COPY**

**@SpringBootApplication is a convenience annotation that adds all of the following:**

* **@Configuration: Tags the class as a source of bean definitions for the application context.**
* **@EnableAutoConfiguration: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.**
* **@EnableWebMvc: Flags the application as a web application and activates key behaviors, such as setting up a  DispatcherServlet. Spring Boot adds it automatically when it sees spring-webmvc on the classpath.**
* **@ComponentScan: Tells Spring to look for other components, configurations, and services in the the  com.example.testingweb package, letting it find the HelloController class.**
* **@SpringBootTest**

**The main() method uses Spring Boot’s SpringApplication.run() method to launch an application. Did you notice that there is not a single line of XML? There is no web.xml file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure. Spring Boot handles all of that for you.**

**Logging output is displayed. The service should be up and running within a few seconds.**

**The @SpringBootTest annotation tells Spring Boot to look for a main configuration class (one with @SpringBootApplication, for instance) and use that to start a Spring application context. You can run this test in your IDE or on the command line (by running ./mvnw test or ./gradlew test), and it should pass. To convince yourself that the context is creating your controller, you could add an assertion, as the following example (from src/test/java/com/example/testingweb/SmokeTest.java) shows:**

**package com.example.testingweb;**

**import static org.assertj.core.api.Assertions.assertThat;**

**import org.junit.jupiter.api.Test;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.boot.test.context.SpringBootTest;**

**@SpringBootTest**

**public class SmokeTest {**

**@Autowired**

**private HomeController controller;**

**@Test**

**public void contextLoads() throws Exception {**

**assertThat(controller).isNotNull();**

**}**

**}COPY**

**Spring interprets the @Autowired annotation, and the controller is injected before the test methods are run. We use**[**AssertJ**](https://joel-costigliola.github.io/assertj/)**(which provides assertThat() and other methods) to express the test assertions.**

|  |  |
| --- | --- |
|  | **A nice feature of the Spring Test support is that the application context is cached between tests. That way, if you have multiple methods in a test case or multiple test cases with the same configuration, they incur the cost of starting the application only once. You can control the cache by using the**[**@DirtiesContext**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/test/annotation/DirtiesContext.html)**annotation.** |

**It is nice to have a sanity check, but you should also write some tests that assert the behavior of your application. To do that, you could start the application and listen for a connection (as it would do in production) and then send an HTTP request and assert the response. The following listing (from src/test/java/com/example/testingweb/HttpRequestTest.java) shows how to do so:**

**package com.example.testingweb;**

**import org.junit.jupiter.api.Test;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.boot.test.context.SpringBootTest;**

**import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;**

**import org.springframework.boot.test.web.client.TestRestTemplate;**

**import org.springframework.boot.web.server.LocalServerPort;**

**import static org.assertj.core.api.Assertions.assertThat;**

**@SpringBootTest(webEnvironment = WebEnvironment.RANDOM\_PORT)**

**public class HttpRequestTest {**

**@LocalServerPort**

**private int port;**

**@Autowired**

**private TestRestTemplate restTemplate;**

**@Test**

**public void greetingShouldReturnDefaultMessage() throws Exception {**

**assertThat(this.restTemplate.getForObject("http://localhost:" + port + "/",**

**String.class)).contains("Hello, World");**

**}**

**}COPY**

**Note the use of webEnvironment=RANDOM\_PORT to start the server with a random port (useful to avoid conflicts in test environments) and the injection of the port with @LocalServerPort. Also, note that Spring Boot has automatically provided a TestRestTemplate for you. All you have to do is add @Autowired to it.**

**Another useful approach is to not start the server at all but to test only the layer below that, where Spring handles the incoming HTTP request and hands it off to your controller. That way, almost of the full stack is used, and your code will be called in exactly the same way as if it were processing a real HTTP request but without the cost of starting the server. To do that, use Spring’s MockMvc and ask for that to be injected for you by using the @AutoConfigureMockMvc annotation on the test case. The following listing (from src/test/java/com/example/testingweb/TestingWebApplicationTest.java) shows how to do so:**

**package com.example.testingweb;**

**import static org.hamcrest.Matchers.containsString;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;**

**import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;**

**import org.junit.jupiter.api.Test;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;**

**import org.springframework.boot.test.context.SpringBootTest;**

**import org.springframework.test.web.servlet.MockMvc;**

**@SpringBootTest**

**@AutoConfigureMockMvc**

**public class TestingWebApplicationTest {**

**@Autowired**

**private MockMvc mockMvc;**

**@Test**

**public void shouldReturnDefaultMessage() throws Exception {**

**this.mockMvc.perform(get("/")).andDo(print()).andExpect(status().isOk())**

**.andExpect(content().string(containsString("Hello, World")));**

**}**

**}COPY**

**In this test, the full Spring application context is started but without the server. We can narrow the tests to only the web layer by using @WebMvcTest, as the following listing (from src/test/java/com/example/testingweb/WebLayerTest.java) shows:**

**@WebMvcTest**

**public class WebLayerTest {**

**@Autowired**

**private MockMvc mockMvc;**

**@Test**

**public void shouldReturnDefaultMessage() throws Exception {**

**this.mockMvc.perform(get("/")).andDo(print()).andExpect(status().isOk())**

**.andExpect(content().string(containsString("Hello, World")));**

**}**

**}COPY**

**The test assertion is the same as in the previous case. However, in this test, Spring Boot instantiates only the web layer rather than the whole context. In an application with multiple controllers, you can even ask for only one to be instantiated by using, for example, @WebMvcTest(HomeController.class).**

**So far, our HomeController is simple and has no dependencies. We could make it more realistic by introducing an extra component to store the greeting (perhaps in a new controller). The following example (from src/main/java/com/example/testingweb/GreetingController.java) shows how to do so:**

**package com.example.testingweb;**

**import org.springframework.stereotype.Controller;**

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

**import org.springframework.web.bind.annotation.ResponseBody;**

**@Controller**

**public class GreetingController {**

**private final GreetingService service;**

**public GreetingController(GreetingService service) {**

**this.service = service;**

**}**

**@RequestMapping("/greeting")**

**public @ResponseBody String greeting() {**

**return service.greet();**

**}**

**}COPY**

**Then create a greeting service, as the following listing (from src/main/java/com/example/testingweb/GreetingService.java) shows:**

**package com.example.testingweb;**

**import org.springframework.stereotype.Service;**

**@Service**

**public class GreetingService {**

**public String greet() {**

**return "Hello, World";**

**}**

**}COPY**

**Spring automatically injects the service dependency into the controller (because of the constructor signature). The following listing (from src/test/java/com/example/testingweb/WebMockTest.java) shows how to test this controller with @WebMvcTest:**

**package com.example.testingweb;**

**import static org.hamcrest.Matchers.containsString;**

**import static org.mockito.Mockito.when;**

**import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;**

**import static org.springframework.test.web.servlet.result.MockMvcResultHandlers.print;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;**

**import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;**

**import org.junit.jupiter.api.Test;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;**

**import org.springframework.boot.test.mock.mockito.MockBean;**

**import org.springframework.test.web.servlet.MockMvc;**

**@WebMvcTest(GreetingController.class)**

**public class WebMockTest {**

**@Autowired**

**private MockMvc mockMvc;**

**@MockBean**

**private GreetingService service;**

**@Test**

**public void greetingShouldReturnMessageFromService() throws Exception {**

**when(service.greet()).thenReturn("Hello, Mock");**

**this.mockMvc.perform(get("/greeting")).andDo(print()).andExpect(status().isOk())**

**.andExpect(content().string(containsString("Hello, Mock")));**

**}**

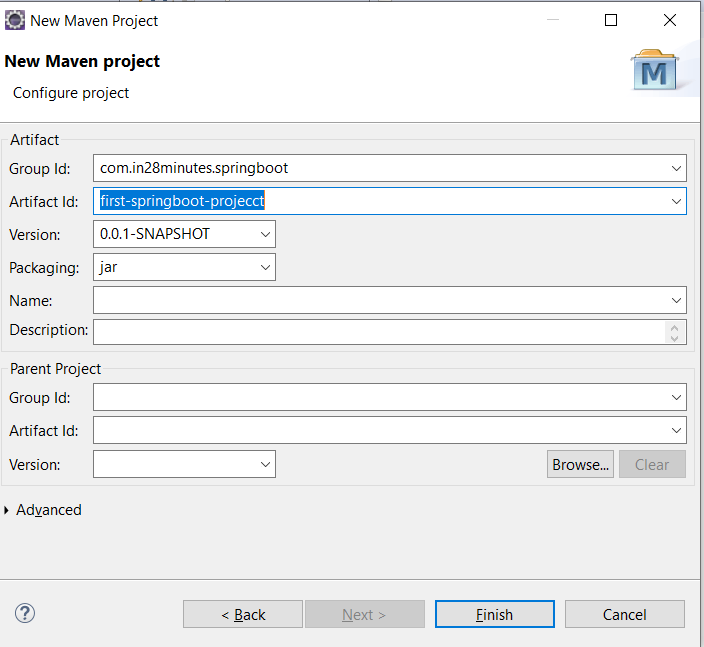
**}COPY**

**We use @MockBean to create and inject a mock for the GreetingService (if you do not do so, the application context cannot start), and we set its expectations using Mockito.**

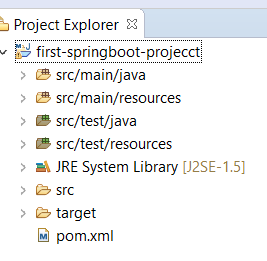
## Summary

**Congratulations! You have developed a Spring application and tested it with JUnit and Spring MockMvc and have used Spring Boot to isolate the web layer and load a special application context.**

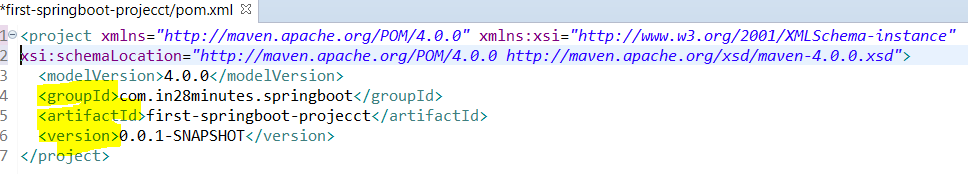
1. **Create a workspace: Workspace is basically is a folder where all the files belong to the project is stored.**
2. **Create a maven project and give the project name**
3. **Name of the project will be : GroupId+ArtifactId = Project name**
4. **GroupId is nothing but package name and ArtifatId is nothing but project name**
5. **Choose the version as snapshot and packaging as jar**

****

1. **Click finish and we will get project on eclipse explorer with the following project structure.**

****

1. **Here all the folder will be empty and we will get Project Object Model(POM) for maven configuration, where we will define all the dependencies of our project.**
2. **Src-main-java folder will have all the source code and src-test-java will have unit test or integration testing code.**
3. **Initially the POM will look like give below.**

****

**Here groupId , artifactId and version is exactly same what we have given during project creation.**

|  |
| --- |
| **<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 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>**  **<modelVersion>4.0.0</modelVersion>**  **<groupId>com.in28minutes.springboot</groupId>**  **<artifactId>first-springboot-projecct</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **</project>** |

**8- SNAPSHOT represents project under development**

**Note: Since java project needs some jar file as dependencies of the project. So Maven manages these dependencies of the project. For example we are creating spring boot project so we need spring jar and for logging we will use logging jar (log4j or some other slf4j.jar). So before maven we used to download the jar manually and put it to our project workspace class path. So include spring dependency or any other dependency, we just need to find out dependency name under the <dependencies> section into POM.xml file.**

**As soon as we update the project, Maven will download all the jars and will put into classpath of the project to be used.**

|  |
| --- |
| **<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 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>**  **<modelVersion>4.0.0</modelVersion>**  **<groupId>com.in28minutes.springboot</groupId>**  **<artifactId>first-springboot-projecct</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **<dependencies>**  **<!-- https://mvnrepository.com/artifact/org.springframework/spring-core -->**  **<dependency>**  **<groupId>org.springframework</groupId>**  **<artifactId>spring-core</artifactId>**  **<version>5.2.6.RELEASE</version>**  **</dependency>**  **</dependencies>**  **</project>** |

****

**As we can see that it downloaded the spring core jar and other jar as part of spring core jar of 5.2.6-Release verion in Maven Dependencies folder.**

****

**Here we can see that common-logging-1.2.jar also got downloaded as the part of spring core jar. Since we are using spring core and it needs common logging. So as we can see that we don’t need to download manually and put into project class path. Maven manages all the dependencies automatically. So in short Maven is nothing but build tool to build the project with its dependencies.**

1. **Now let’s focus on creating spring boot project: Five steps given below.**
2. **Add the Spring Boot starter parent in POM: One of the great things of this spring boot starter parent is that we don’t need to mention of spring version any more, it takes care of versions of all spring dependencies. As in the above we saw that while including dependency name, we need to mention the version of the dependencies**
3. **Include Spring Boot Starter Web: Before Spring Boot if we need to develop any web based project then we need to include lots of dependeies like, springcore, Spring MVC etc. But this Spring Boot Starter Web would bring all the decencies automatically.**
4. **Include Java Version 8: As Spring Boot project requires at least java version 8.**
5. **Spring Boot Plugin: This plugin would help us to launch the Spring Boot application**
6. **Create Spring Boot Application Launcher: This is the java file used to launch the Spring Boot app.**

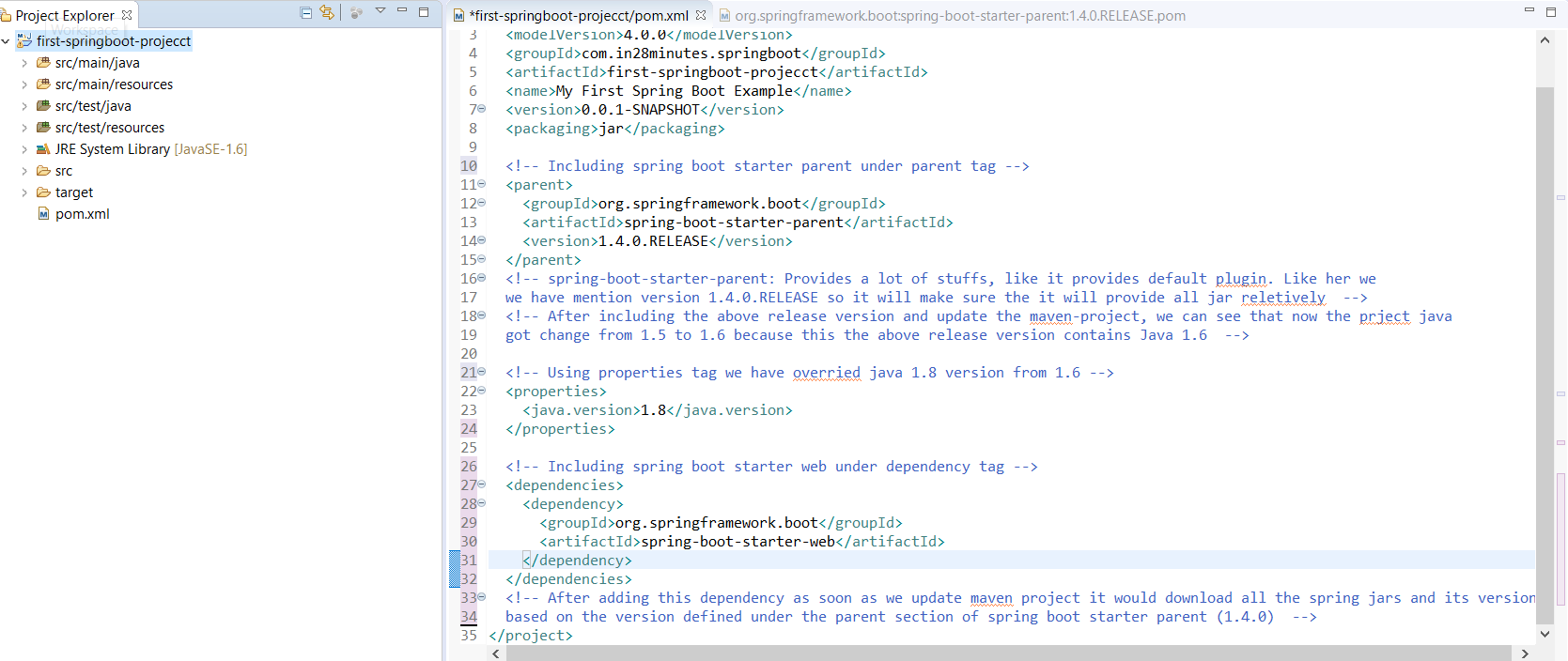
|  |
| --- |
| **<projectxmlns=*"http://maven.apache.org/POM/4.0.0"*xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"***  **xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>**  **<modelVersion>4.0.0</modelVersion>**  **<groupId>com.in28minutes.springboot</groupId>**  **<artifactId>first-springboot-projecct</artifactId>**  **<name>My First Spring Boot Example</name>**  **<version>0.0.1-SNAPSHOT</version>**  **<packaging>jar</packaging>**  **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<version>1.4.0.RELEASE</version>**  **</parent>**  **<! -- Spring-boot-starter-parent: Provides a lot of stuffs, like it provides default plugin. Like here we have mention version 1.4.0.RELEASE so it will make sure the it will provide all jar relatively -->**  **<!-- After including the above release version and update the maven-project, we can see that now the project java version got change from 1.5 to 1.6 because this the above release version contains Java 1.6 -->**  **<!-- Using properties tag we have override java 1.8 version from 1.6 -->**  **<properties>**  **<java.version>1.8</java.version>**  **</properties>**  **<! -- Including spring boot starter web under dependency tag for web based application. So if we want develop web application using spring boot then we need to include this-->**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **</dependency>**  **</dependencies>**  **<!-- Now add Spring Boot Plugin which is noting but maven plugin to run the maven project: -->**  **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot.maven.plugin</artifactId>**  **</plugin>**  **</plugins>**  **</build>**  **</project>** |

**Let’s have a look of 1.4.0.RELEASE POM.xml below**

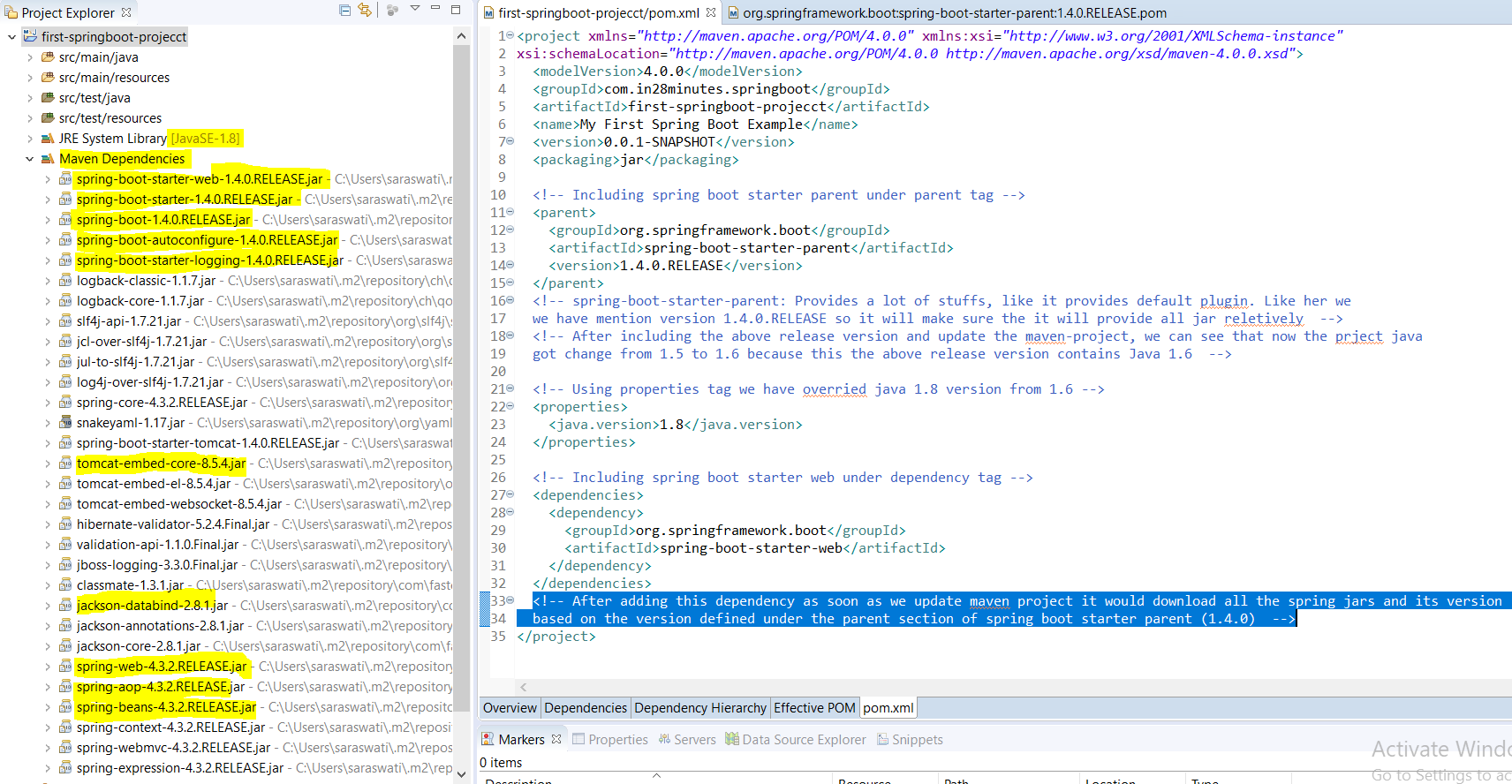
**In the below POM we have java 1.6 is mentioned. Now to override this version include java 1.8 under property section of Project POM.xml**

|  |
| --- |
| **<?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 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>**  **<modelVersion>4.0.0</modelVersion>**  **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-dependencies</artifactId>**  **<version>1.4.0.RELEASE</version>**  **<relativePath>../../spring-boot-dependencies</relativePath>**  **</parent>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<packaging>pom</packaging>**  **<name>Spring Boot Starter Parent</name>**  **<description>Parent pom providing dependency and plugin management for applications**  **built with Maven</description>**  **<url>http://projects.spring.io/spring-boot/</url>**  **<organization>**  **<name>Pivotal Software, Inc.</name>**  **<url>http://www.spring.io</url>**  **</organization>**  **<properties>**  **<java.version>1.6</java.version>**  **<resource.delimiter>@</resource.delimiter><!-- delimiter that doesn't clash with Spring ${} placeholders -->**  **<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**  **<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>**  **<maven.compiler.source>${java.version}</maven.compiler.source>**  **<maven.compiler.target>${java.version}</maven.compiler.target>**  **</properties>**  **<dependencyManagement>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework</groupId>**  **<artifactId>spring-core</artifactId>**  **<version>${spring.version}</version>**  **<exclusions>**  **<exclusion>**  **<groupId>commons-logging</groupId>**  **<artifactId>commons-logging</artifactId>**  **</exclusion>**  **</exclusions>**  **</dependency>**  **</dependencies>**  **</dependencyManagement>**  **<build>**  **<!-- Turn on filtering by default for application properties -->**  **<resources>**  **<resource>**  **<directory>${basedir}/src/main/resources</directory>**  **<filtering>true</filtering>**  **<includes>**  **<include>\*\*/application\*.yml</include>**  **<include>\*\*/application\*.properties</include>**  **</includes>**  **</resource>**  **<resource>**  **<directory>${basedir}/src/main/resources</directory>**  **<excludes>**  **<exclude>\*\*/application\*.yml</exclude>**  **<exclude>\*\*/application\*.properties</exclude>**  **</excludes>**  **</resource>**  **</resources>**  **<pluginManagement>**  **<plugins>**  **<!-- Apply more sensible defaults for user projects -->**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-failsafe-plugin</artifactId>**  **<executions>**  **<execution>**  **<goals>**  **<goal>integration-test</goal>**  **<goal>verify</goal>**  **</goals>**  **</execution>**  **</executions>**  **</plugin>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-jar-plugin</artifactId>**  **<configuration>**  **<archive>**  **<manifest>**  **<mainClass>${start-class}</mainClass>**  **<addDefaultImplementationEntries>true</addDefaultImplementationEntries>**  **</manifest>**  **</archive>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-surefire-plugin</artifactId>**  **<configuration>**  **<includes>**  **<include>\*\*/\*Tests.java</include>**  **<include>\*\*/\*Test.java</include>**  **</includes>**  **<excludes>**  **<exclude>\*\*/Abstract\*.java</exclude>**  **</excludes>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-war-plugin</artifactId>**  **<configuration>**  **<failOnMissingWebXml>false</failOnMissingWebXml>**  **<archive>**  **<manifest>**  **<mainClass>${start-class}</mainClass>**  **<addDefaultImplementationEntries>true</addDefaultImplementationEntries>**  **</manifest>**  **</archive>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.codehaus.mojo</groupId>**  **<artifactId>exec-maven-plugin</artifactId>**  **<configuration>**  **<mainClass>${start-class}</mainClass>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-resources-plugin</artifactId>**  **<version>2.6</version>**  **<configuration>**  **<delimiters>**  **<delimiter>${resource.delimiter}</delimiter>**  **</delimiters>**  **<useDefaultDelimiters>false</useDefaultDelimiters>**  **</configuration>**  **</plugin>**  **<plugin>**  **<groupId>pl.project13.maven</groupId>**  **<artifactId>git-commit-id-plugin</artifactId>**  **<version>2.1.11</version>**  **<executions>**  **<execution>**  **<goals>**  **<goal>revision</goal>**  **</goals>**  **</execution>**  **</executions>**  **<configuration>**  **<verbose>true</verbose>**  **<dateFormat>yyyy-MM-dd'T'HH:mm:ssZ</dateFormat>**  **<generateGitPropertiesFile>true</generateGitPropertiesFile>**    **<generateGitPropertiesFilename>${project.build.outputDirectory}/git.properties</generateGitPropertiesFilename>**  **</configuration>**  **</plugin>**  **<!-- Support our own plugin -->**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **<executions>**  **<execution>**  **<goals>**  **<goal>repackage</goal>**  **</goals>**  **</execution>**  **</executions>**  **<configuration>**  **<mainClass>${start-class}</mainClass>**  **</configuration>**  **</plugin>**  **<!-- Support shade packaging (if the user does not want to use our plugin) -->**  **<plugin>**  **<groupId>org.apache.maven.plugins</groupId>**  **<artifactId>maven-shade-plugin</artifactId>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **<version>1.4.0.BUILD-SNAPSHOT</version>**  **</dependency>**  **</dependencies>**  **<configuration>**  **<keepDependenciesWithProvidedScope>true</keepDependenciesWithProvidedScope>**  **<createDependencyReducedPom>true</createDependencyReducedPom>**  **<filters>**  **<filter>**  **<artifact>\*:\*</artifact>**  **<excludes>**  **<exclude>META-INF/\*.SF</exclude>**  **<exclude>META-INF/\*.DSA</exclude>**  **<exclude>META-INF/\*.RSA</exclude>**  **</excludes>**  **</filter>**  **</filters>**  **</configuration>**  **<executions>**  **<execution>**  **<phase>package</phase>**  **<goals>**  **<goal>shade</goal>**  **</goals>**  **<configuration>**  **<transformers>**  **<transformerimplementation=*"org.apache.maven.plugins.shade.resource.AppendingTransformer"*>**  **<resource>META-INF/spring.handlers</resource>**  **</transformer>**  **<transformerimplementation=*"org.springframework.boot.maven.PropertiesMergingResourceTransformer"*>**  **<resource>META-INF/spring.factories</resource>**  **</transformer>**  **<transformerimplementation=*"org.apache.maven.plugins.shade.resource.AppendingTransformer"*>**  **<resource>META-INF/spring.schemas</resource>**  **</transformer>**  **<transformerimplementation=*"org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"*/>**  **<transformerimplementation=*"org.apache.maven.plugins.shade.resource.ManifestResourceTransformer"*>**  **<mainClass>${start-class}</mainClass>**  **</transformer>**  **</transformers>**  **</configuration>**  **</execution>**  **</executions>**  **</plugin>**  **</plugins>**  **</pluginManagement>**  **</build>**  **</project>** |

1. **After adding dependency and before doing maven update project. No Maven dependcies are added**

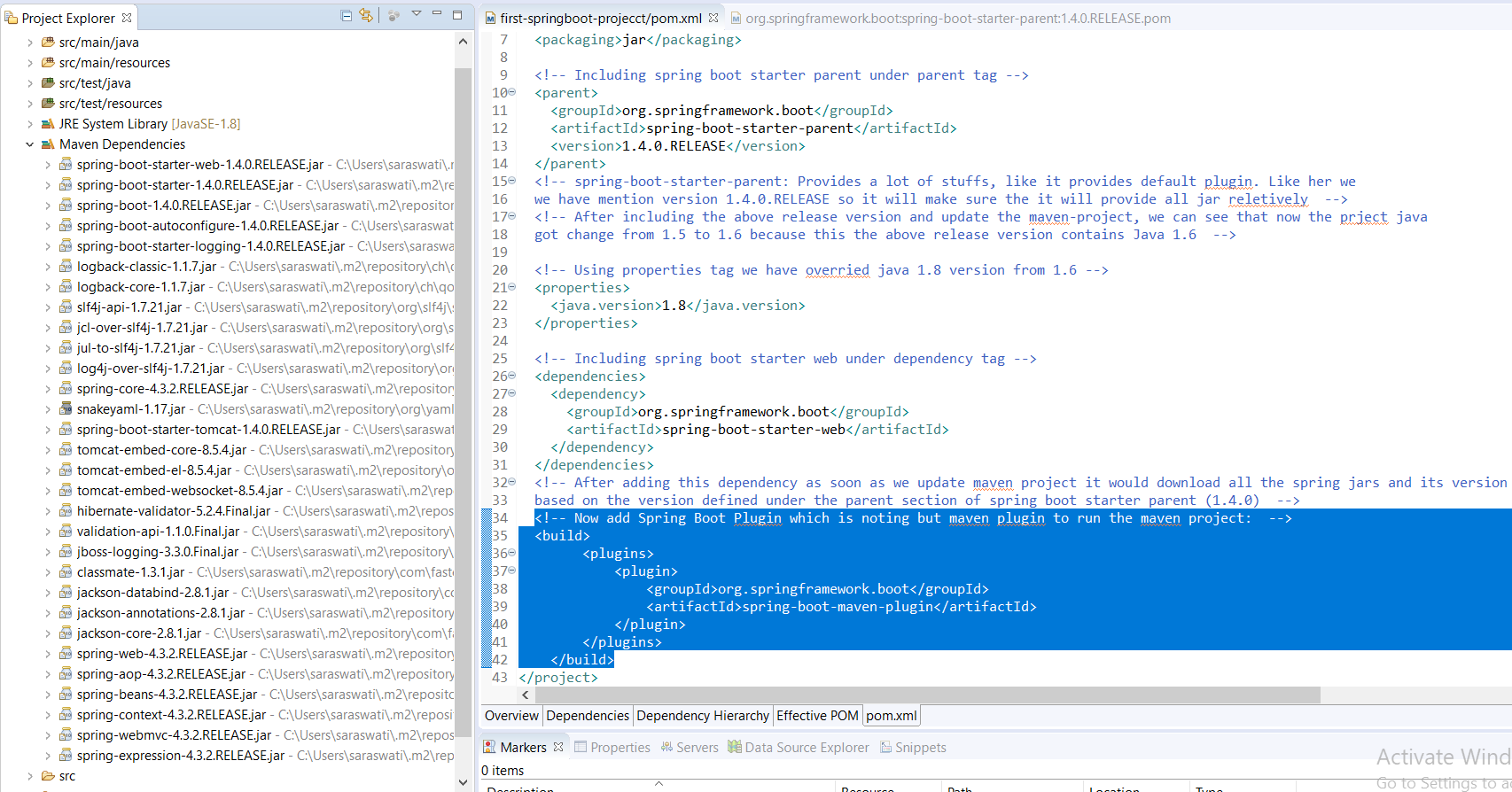
****

1. **After adding this dependency as soon as we update maven project it would download all the spring jars and its version based on the version defined under the parent section of spring boot starter parent (1.4.0) and java version also will be changed.**

****

**So 1.4.0.Release of spring boot starter parent will download the following version of spring 4.3.2 of core, web, aop, beans and etc. Jackson-databind-2.8, Embedded Tomcat 8.5 , logging jars and so on.**

1. **Now include Spring Boot Plugin: This plugin would be added under <build>under<plugins>under<plugin> tag**

****

|  |
| --- |
| **<!-- Now add Spring Boot Plugin which is noting but mavenplugin to run the maven project: --**  **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **</plugin>**  **</plugins>**  **</build>** |

**Now we have completed POM.xml (1 to 4 steps) now the only last 5th step is create spring boot application launcher.**

1. **Creating Spring Boot Application Launcher (java File)**

|  |
| --- |
| **src/main/java/com/in28minutes/springboot/Application.java**  **package com.in28minutes.springboot;**  **import org.springframework.boot.SpringApplication;**  **import org.springframework.boot.autoconfigure.SpringBootApplication;**  **import org.springframework.context.ApplicationContext;**  **@SpringBootApplication**  **public class Application {**  **public static void main(String[] args) {**  **ApplicationContext ctx = SpringApplication.run(Application.class, args);**  **}**  **}** |

**This launcher class having main method. Here we are creating ApplicationContext and using Spring boot utility class SpringApplication and its method run we are launching this launcher class (Application.class)**

**@SpringBootApplication: This annotation creates a magic and do a lot of stuffs to launch the launcher class.**

|  |
| --- |
| [**[Open Declaration](eclipse-open:%E2%98%82=first-springboot-projecct/C:\/Users\/saraswati\/.m2\/repository\/org\/springframework\/boot\/spring-boot-autoconfigure\/1.4.0.RELEASE\/spring-boot-autoconfigure-1.4.0.RELEASE.jar%3corg.springframework.boot.autoconfigure(SpringBootApplication.class%E2%98%83SpringBootApplication)**](eclipse-open:%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)[**org**](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**](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**](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**](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**  **@**[**SpringBootConfiguration**](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%5C/1.4.0.RELEASE%5C/spring-boot-1.4.0.RELEASE.jar%3Corg.springframework.boot(SpringBootConfiguration.class%E2%98%83SpringBootConfiguration) **@**[**EnableAutoConfiguration**](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(EnableAutoConfiguration.class%E2%98%83EnableAutoConfiguration) **@**[**ComponentScan**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan.class%E2%98%83ComponentScan)**(**[**excludeFilters**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan.class%E2%98%83ComponentScan~excludeFilters)**={@**[**Filter**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter)**(**[**type**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~type)**=**[**CUSTOM**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(FilterType.class%E2%98%83FilterType%5ECUSTOM)**,** [**classes**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.context.annotation(ComponentScan$Filter.class%E2%98%83Filter~classes)**={**[**TypeExcludeFilter**](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%5C/1.4.0.RELEASE%5C/spring-boot-1.4.0.RELEASE.jar%3Corg.springframework.boot.context(TypeExcludeFilter.class%E2%98%83TypeExcludeFilter)**.class})}) @**[**Target**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=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 that declares one or more** [**@Bean**](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%82Bean) **methods and also 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.**  **Since:**  **1.2.0**  **Author:**  **Phillip WebbStephane Nicoll** |

**@SpringBootApplication is a convenience annotation that adds all of the following:**

**@Configuration: Tags the class as a source of bean definitions for the application context.**

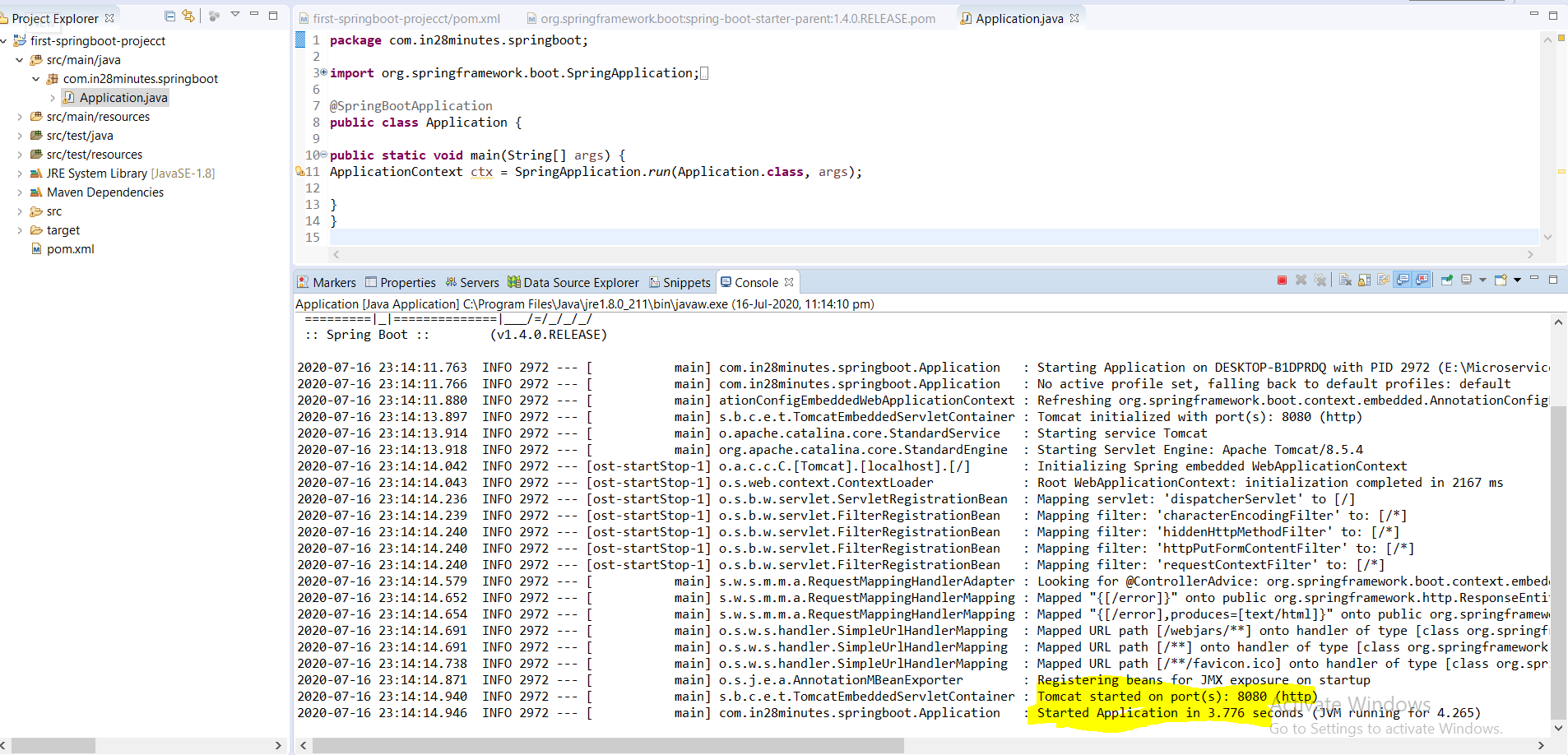
**@EnableAutoConfiguration: Tells Spring Boot to start adding beans based on classpath settings, other beans, and various property settings.**

**@EnableWebMvc: Flags the application as a web application and activates key behaviors, such as setting up a DispatcherServlet. Spring Boot adds it automatically when it sees spring-webmvc on the classpath.**

**@ComponentScan: Tells Spring to look for other components, configurations, and services in the the com.example.testingweb package, letting it find the HelloController class.**

**The main() method uses Spring Boot’s SpringApplication.run() method to launch an application. Did you notice that there is not a single line of XML? There is no web.xml file, either. This web application is 100% pure Java and you did not have to deal with configuring any plumbing or infrastructure. Spring Boot handles all of that for you.**

**Now right click on the Application.java and run as java application: It will just start the embedded tomcat server and start the application up and running.**

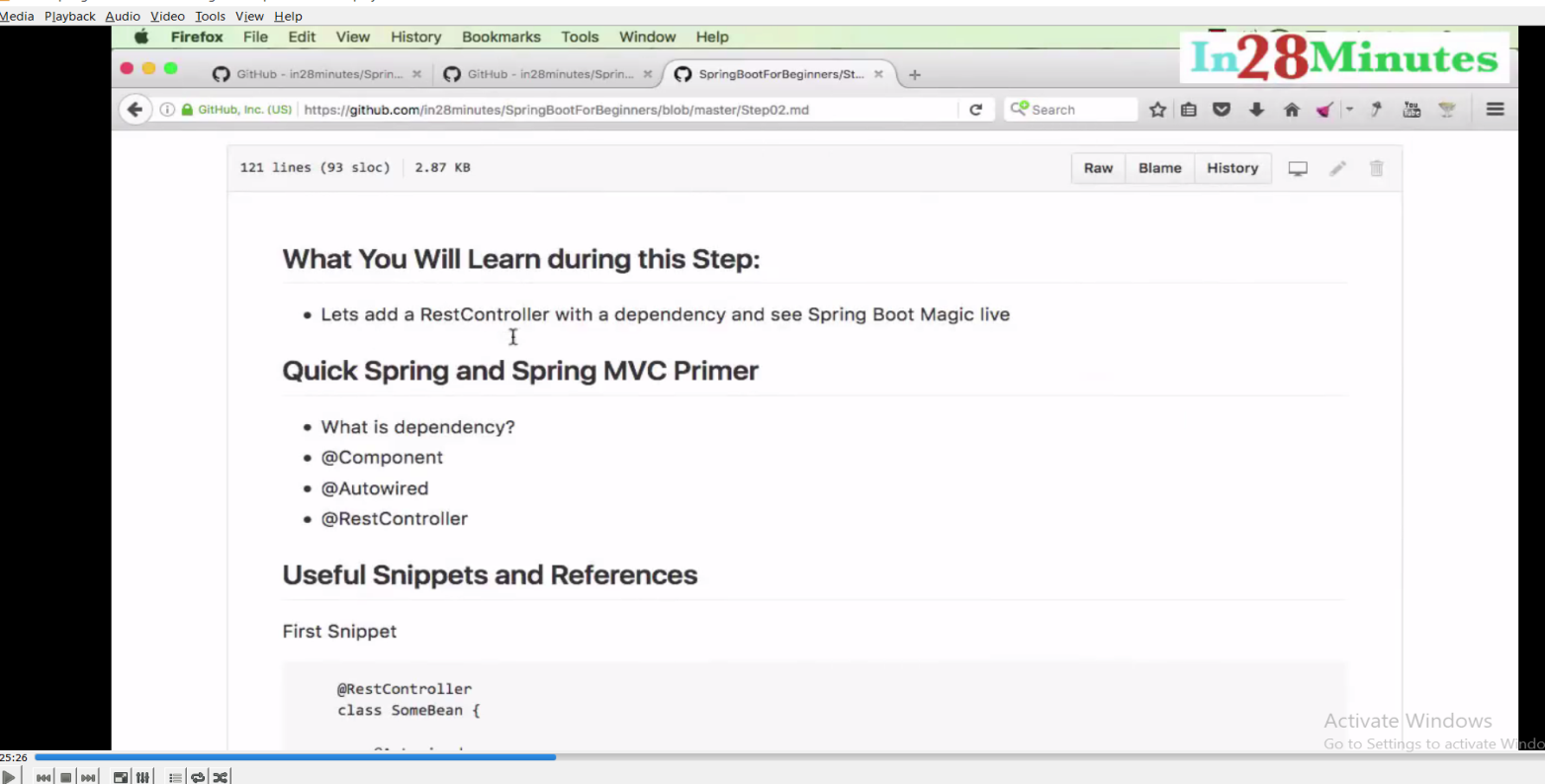
****

**So basically five basic step needs to start the spring boot application up and running.**

1. **Include Spring Boot Starter Parent in POM.xml**
2. **Include Java version 8 in POM.xml**
3. **Include Spring Boot Starter Web in POM.xml**
4. **Include Spring Boot Maven Plugin in POM.xml**
5. **Create Spring Boot Application.java as spring boot application Launcher**

**Note we have not created any controller so far. We have the server in running status that is it.**

|  |
| --- |
| **. \_\_\_\_ \_ \_\_ \_ \_**  **/\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \**  **( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \**  **\\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )**  **' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /**  **=========|\_|==============|\_\_\_/=/\_/\_/\_/**  **:: Spring Boot :: (v1.4.0.RELEASE)**  **2020-07-16 23:14:11.763 INFO 2972 --- [ main] com.in28minutes.springboot.Application : Starting Application on DESKTOP-B1DPRDQ with PID 2972 (E:\MicroservicesWorkspace\In28minuteCourceWS\first-springboot-projecct\target\classes started by saraswati in E:\MicroservicesWorkspace\In28minuteCourceWS\first-springboot-projecct)**  **2020-07-16 23:14:11.766 INFO 2972 --- [ main] com.in28minutes.springboot.Application : No active profile set, falling back to default profiles: default**  **2020-07-16 23:14:11.880 INFO 2972 --- [ main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@78b66d36: startup date [Thu Jul 16 23:14:11 IST 2020]; root of context hierarchy**  **2020-07-16 23:14:13.897 INFO 2972 --- [ main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port(s): 8080 (http)**  **2020-07-16 23:14:13.914 INFO 2972 --- [ main] o.apache.catalina.core.StandardService : Starting service Tomcat**  **2020-07-16 23:14:13.918 INFO 2972 --- [ main] org.apache.catalina.core.StandardEngine : Starting Servlet Engine: Apache Tomcat/8.5.4**  **2020-07-16 23:14:14.042 INFO 2972 --- [ost-startStop-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext**  **2020-07-16 23:14:14.043 INFO 2972 --- [ost-startStop-1] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 2167 ms**  **2020-07-16 23:14:14.236 INFO 2972 --- [ost-startStop-1] o.s.b.w.servlet.ServletRegistrationBean : Mapping servlet: 'dispatcherServlet' to [/]**  **2020-07-16 23:14:14.239 INFO 2972 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'characterEncodingFilter' to: [/\*]**  **2020-07-16 23:14:14.240 INFO 2972 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'hiddenHttpMethodFilter' to: [/\*]**  **2020-07-16 23:14:14.240 INFO 2972 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'httpPutFormContentFilter' to: [/\*]**  **2020-07-16 23:14:14.240 INFO 2972 --- [ost-startStop-1] o.s.b.w.servlet.FilterRegistrationBean : Mapping filter: 'requestContextFilter' to: [/\*]**  **2020-07-16 23:14:14.579 INFO 2972 --- [main] s.w.s.m.m.a.RequestMappingHandlerAdapter : Looking for @ControllerAdvice: org.springframework.boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext@78b66d36: startup date [Thu Jul 16 23:14:11 IST 2020]; root of context hierarchy**  **2020-07-16 23:14:14.652 INFO 2972 --- [Main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error]}" onto public org.springframework.http.ResponseEntity<java.util.Map<java.lang.String, java.lang.Object>> org.springframework.boot.autoconfigure.web.BasicErrorController.error(javax.servlet.http.HttpServletRequest)**  **2020-07-16 23:14:14.654 INFO 2972 --- [main] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped "{[/error],produces=[text/html]}" onto public org.springframework.web.servlet.ModelAndView org.springframework.boot.autoconfigure.web.BasicErrorController.errorHtml(javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)**  **2020-07-16 23:14:14.691 INFO 2972 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/webjars/\*\*] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]**  **2020-07-16 23:14:14.691 INFO 2972 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/\*\*] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]**  **2020-07-16 23:14:14.738 INFO 2972 --- [main] o.s.w.s.handler.SimpleUrlHandlerMapping : Mapped URL path [/\*\*/favicon.ico] onto handler of type [class org.springframework.web.servlet.resource.ResourceHttpRequestHandler]**  **2020-07-16 23:14:14.871 INFO 2972 --- [main] o.s.j.e.a.AnnotationMBeanExporter : Registering beans for JMX exposure on startup**  **2020-07-16 23:14:14.940 INFO 2972 [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)**  **2020-07-16 23:14:14.946 INFO 2972 --- [main] com.in28minutes.springboot.Application : Started Application in 3.776 seconds (JVM running for 4.265)** |

****

**Now let’s add Controller in the Spring Boot Application.**

**Step02.md: Creating first Rest Controller**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step02.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step02.md)

**Let’s add a RestController : In SB Using @RestController annotation at the Class level we can create a controller class.**

**@RestController :**

|  |
| --- |
| [[Open Declaration](eclipse-open:%E2%98%82=first-springboot-projecct/C:\/Users\/saraswati\/.m2\/repository\/org\/springframework\/spring-web\/4.3.2.RELEASE\/spring-web-4.3.2.RELEASE.jar%3corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController)](eclipse-open:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController)[org](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg).[springframework](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework).[web](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web).[bind](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind).[annotation](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation).RestController **@**[**Controller**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-context%5C/4.3.2.RELEASE%5C/spring-context-4.3.2.RELEASE.jar%3Corg.springframework.stereotype(Controller.class%E2%98%83Controller) **@**[**ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(ResponseBody.class%E2%98%83ResponseBody) **@**[**Target**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Documented.class%E2%98%83Documented)  **A convenience annotation that is itself annotated with** [**@Controller**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController%E2%98%82Controller) **and** [**@ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController%E2%98%82ResponseBody)**.**  **Types that carry this annotation are treated as controllers where** [**@RequestMapping**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController%E2%98%82RequestMapping) **methods assume** [**@ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestController.class%E2%98%83RestController%E2%98%82ResponseBody) **semantics by default.**  **NOTE:@RestController is processed if an appropriate HandlerMapping-HandlerAdapter pair is configured such as the RequestMappingHandlerMapping-RequestMappingHandlerAdapter pair which are the default in the MVC Java config and the MVC namespace. In particular @RestController is not supported with the DefaultAnnotationHandlerMapping-AnnotationMethodHandlerAdapter pair both of which are also deprecated.**  **Since: 4.0** |

**@RestControllerAdvice**

|  |
| --- |
| [-->[Open Declaration](eclipse-open:%E2%98%82=first-springboot-projecct/C:\/Users\/saraswati\/.m2\/repository\/org\/springframework\/spring-web\/4.3.2.RELEASE\/spring-web-4.3.2.RELEASE.jar%3corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice)](eclipse-open:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice)[org](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg).[springframework](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework).[web](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web).[bind](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind).[annotation](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation).RestControllerAdvice **@**[**ControllerAdvice**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(ControllerAdvice.class%E2%98%83ControllerAdvice) **@**[**ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(ResponseBody.class%E2%98%83ResponseBody) **@**[**Target**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/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=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jre1.8.0_211%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Documented.class%E2%98%83Documented)  **A convenience annotation that is itself annotated with** [**@ControllerAdvice**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice%E2%98%82ControllerAdvice) **and** [**@ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice%E2%98%82ResponseBody)**.**  **Types that carry this annotation are treated as controller advice where** [**@ExceptionHandler**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice%E2%98%82ExceptionHandler) **methods assume** [**@ResponseBody**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Users%5C/saraswati%5C/.m2%5C/repository%5C/org%5C/springframework%5C/spring-web%5C/4.3.2.RELEASE%5C/spring-web-4.3.2.RELEASE.jar%3Corg.springframework.web.bind.annotation(RestControllerAdvice.class%E2%98%83RestControllerAdvice%E2%98%82ResponseBody) **semantics by default.**  **NOTE:@RestControllerAdvice is processed if an appropriate HandlerMapping-HandlerAdapter pair is configured such as the RequestMappingHandlerMapping-RequestMappingHandlerAdapter pair which are the default in the MVC Java config and the MVC namespace. In particular @RestControllerAdvice is not supported with the DefaultAnnotationHandlerMapping-AnnotationMethodHandlerAdapter pair both of which are also deprecated.**  **Since: 4.3** |

**@RequestMapping: Annotation for mapping web requests onto specific handler classes and/orhandler methods.**

|  |
| --- |
| **package com.in28minutes.springboot;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **@RestController**  **publicclass WelcomeController {**  **@RequestMapping("/welcome")**  **public String welcome() {**  **return"Welcome to spring boot 28 min";**  **}**  **}** |

**URL:** [**http://localhost:8080/welcome**](http://localhost:8080/welcome)

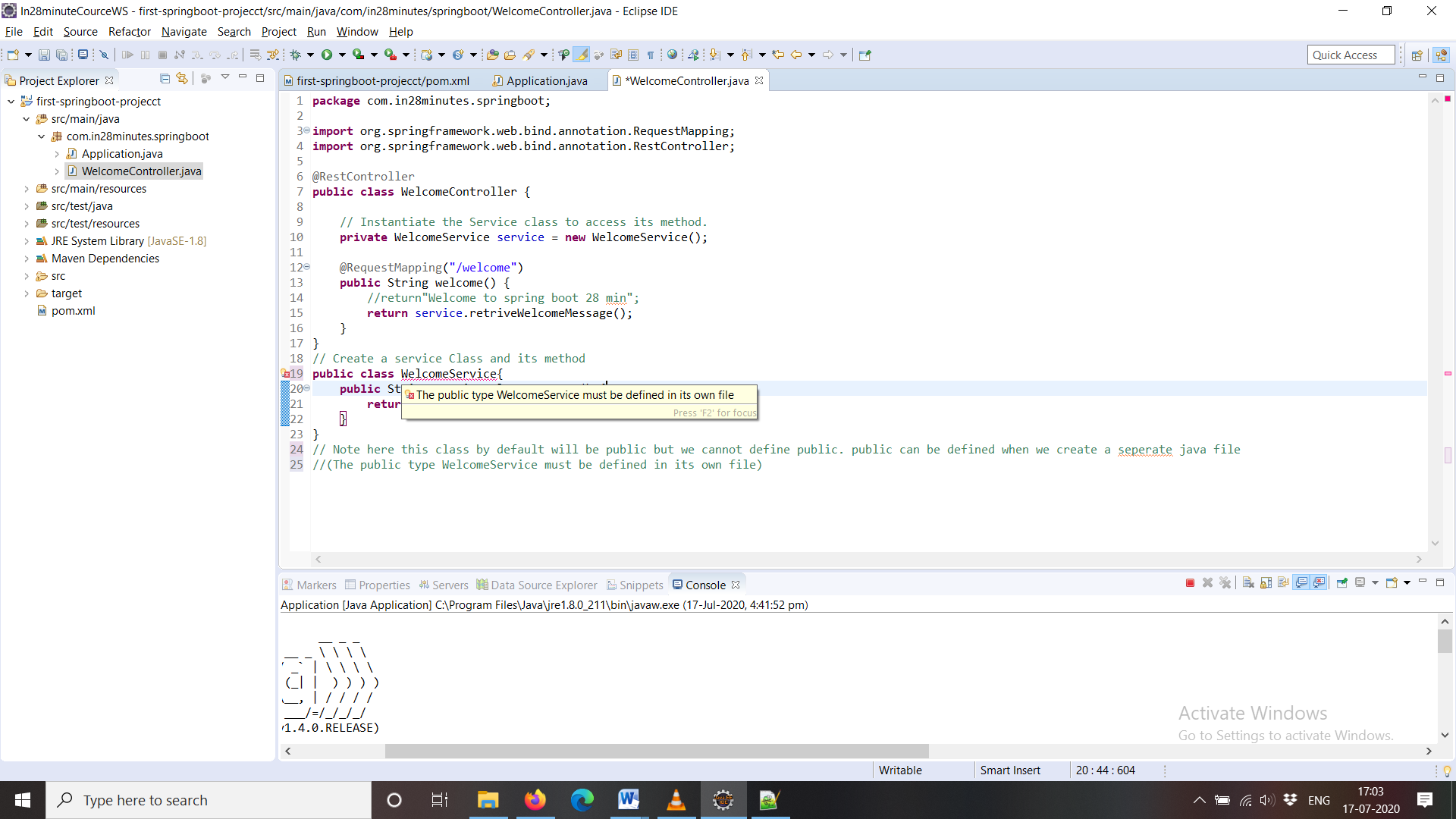
**In the console log we can see the below log**

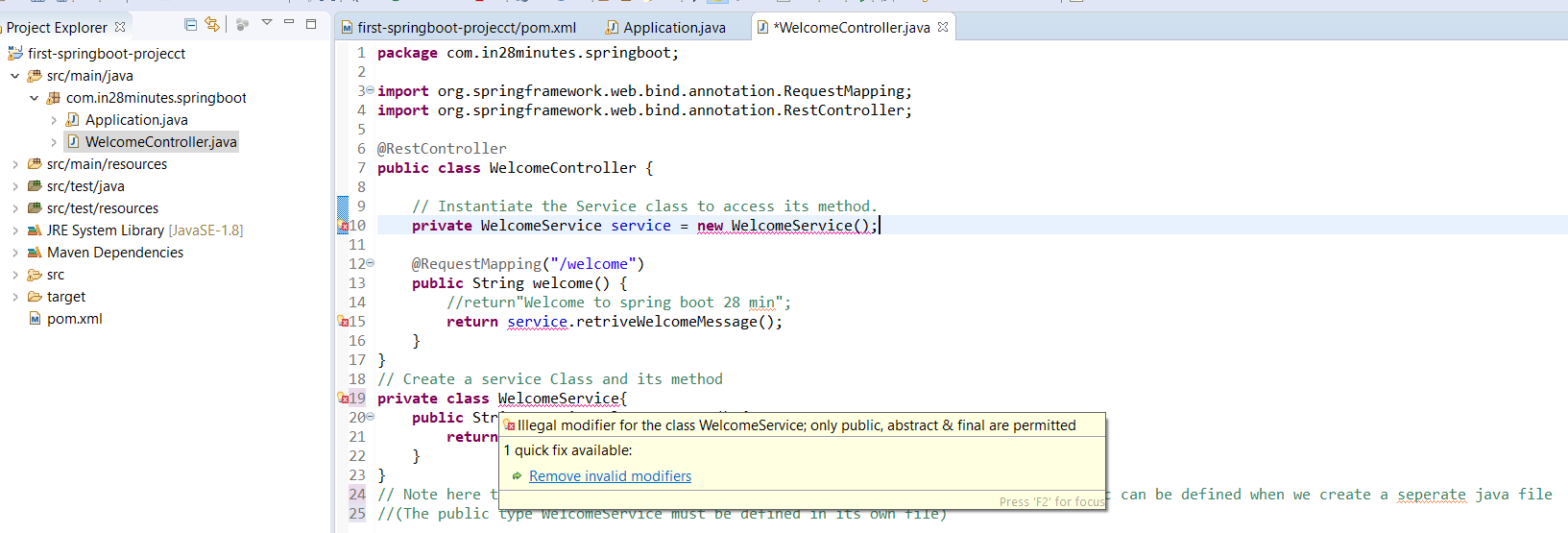
**RequestMappingHandlerMapping : Mapped "{[/welcome]}" onto**

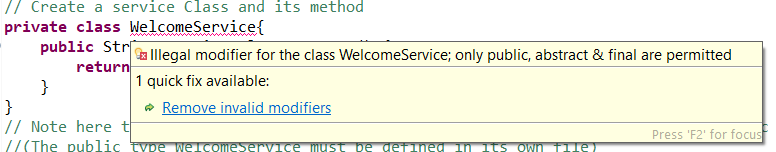
**public java.lang.String com.in28minutes.springboot.WelcomeController.welcome()**

**Note: Here the welcome message is displaying using welcome method using the below URL:** [**http://localhost:8080/welcome**](http://localhost:8080/welcome)

**Now if want to display this welcome message from some service:**

****

****

****

|  |
| --- |
| **package com.in28minutes.springboot;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **@RestController**  **publicclass WelcomeController {**  **// Instantiate the Service class to access its method.**  **private WelcomeService service = new WelcomeService();**  **@RequestMapping("/welcome")**  **public String welcome() {**  **//return"Welcome to spring boot 28 min";**  **returnservice.retriveWelcomeMessage();**  **}**  **}**  **// Create a service Class and its method**  **class WelcomeService{**  **public String retriveWelcomeMessage() {**  **return" Good Morning from Servie class";**  **}**  **}**  **// Note here this class by default will be public and we cannot define public. public can be defined when we create a seperate java file**  **//(The public type WelcomeService must be defined in its own file)** |

**So here WelcomeService is nothing but dependency of the WelcomeController. WelcomeController is dependent on WelcomeService to get message. Here WelcomeController take care of creating instatiation of WelcomeService. But this dependency is hard dependency we cannot change it in future.**

**To manage this dependency as flexible dependency spring brought one concept, where spring will manage all the beans and create and instance the dependent class. In this case we just need to inject the dependency on the WelcomeController class and we can access the dependent class functionality.**

**Ques: Now the question arise that, which is the class or component that spring need to manage: So answer is we can use @Component or @Service annotation at the class level of dependent class (WelcomeService) . Now the spring will manage this bean and will manage the instantiation of the object of the dependent class (WelcomeService). Now whenever we start the application, spring automatically will instantiate and create the object of these classes annotated with @Component or @Service. Now where we want to use these classes we just need to inject. This inject is nothing but call Auto wiring and @Autowired annotation is used for auto wiring. Once we Autowired, we don’t need to create object using new(). Spring will take care of object creation and Autowired automatically.**

**Same example using @Component and @Autowired annotation**

* **@RestController**
* **@RequestMapping**
* **@Autowired**
* **@Component / @Service**

|  |
| --- |
| **package com.in28minutes.springboot;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.stereotype.Component;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **@RestController**  **publicclass WelcomeController {**    **@Autowired**  **private WelcomeService service;**  **@RequestMapping("/welcome")**  **public String welcome() {**  **return service.retriveWelcomeMessage();**  **}**  **}**  **@Component // Created a service Class and its method**  **class WelcomeService{**  **publicString retriveWelcomeMessage() {**  **return" Good Morning from Servie class using @component and @Autowoire";**  **}**  **}** |

**When the application starts:**

1. **Spring creates the object of component class WelcomeService and Controller class WelcomeController.**
2. **When it goes inside of the controller, it finds that something is Autowired and then spring incject here in controller class and then it launch controller class**

**Now few question arises:**

1. **How does Spring Framework knows, where to search for the beans to manage (instantiation and creation of object)**
2. **What is the annotations Spring looks for?**
3. **What if spring does not find what it is looking for?**
4. **How the spring search the particular class (Controller or Component class) in the project package and what are the annotations that spring looks for in the particular classes and what if spring does not find what it is looking for?**

**So let start from last Q, if we remove @Component from WelcomeService class.**

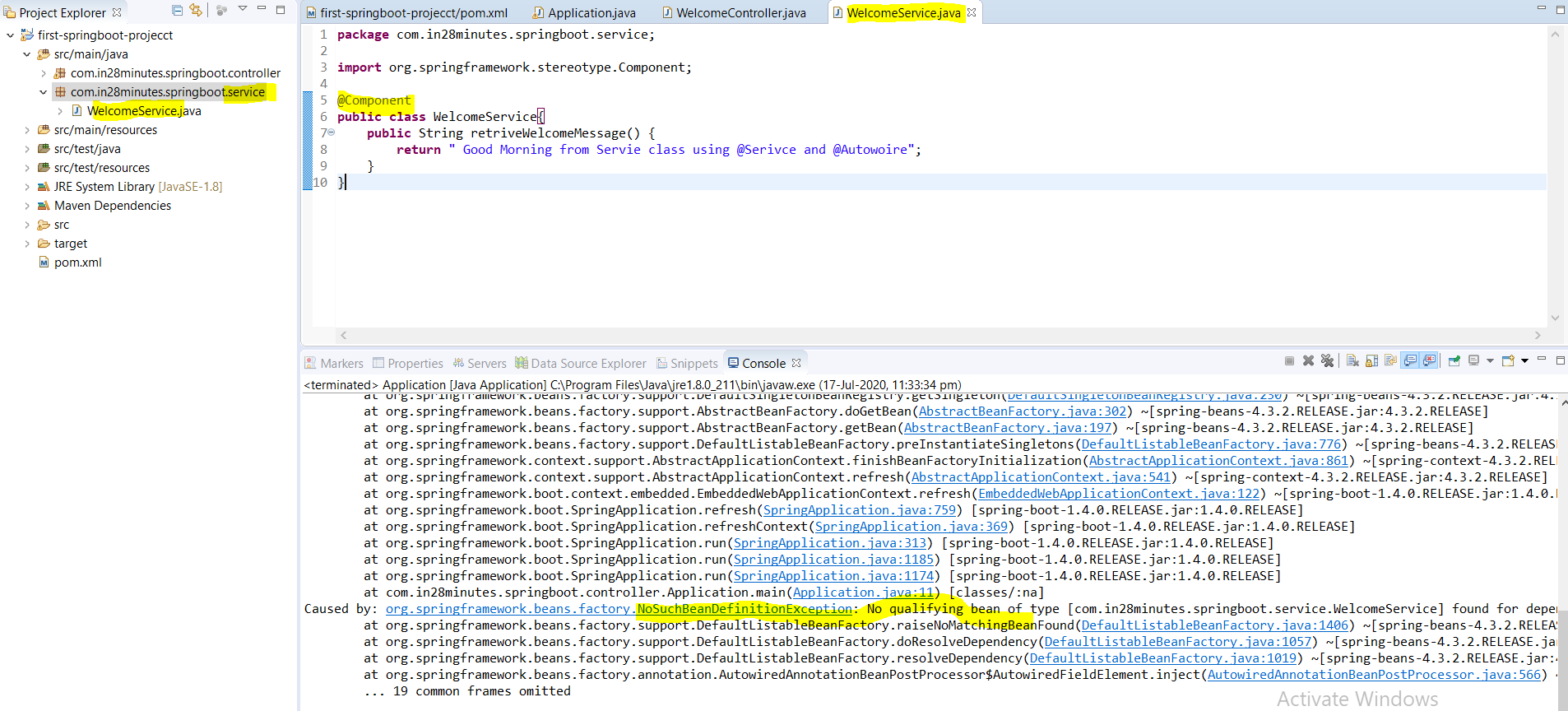
|  |
| --- |
| **Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of type [com.in28minutes.springboot.WelcomeService] found for dependency [com.in28minutes.springboot.WelcomeService]: expected at least 1 bean which qualifies as autowire candidate for this dependency. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}** |

**So we saw that during the application startup only it scan and try to search the dependent class and when it does not found the particular class annotated with auto wired then it does not proceed further to start server also. It stop there itself there only if does not find what it is looking for. NoSuchBeanDefinitionException occurs when it does not found annotation @Component or @Service for annotated with @Autowired.**

**Now: How does Spring Framework knows where to search for the beans to manage (instantiation and creation of object)?**

**For understand let’s do some changes in the project.**

**Move the component class into other package from the controller class and run the application**

****

**As we can see that again we got the same exception, because the service class annotated with @Component moved into other package and spring could not found @Autowired annotated dependent class in the same package or child package or in same class. So when spring finds the component class in the same package it does the auto scan and do all the stuff (manage bean, injection etc.). But once it gets moved into different package then auto scan does not work. In this case to enable auto scan at application we need to @ComponentScan in the Application.java launcher along with the base package name. Component scan happens at Application Launch.**

**@ComponentScan(“com.in28minutes”)**

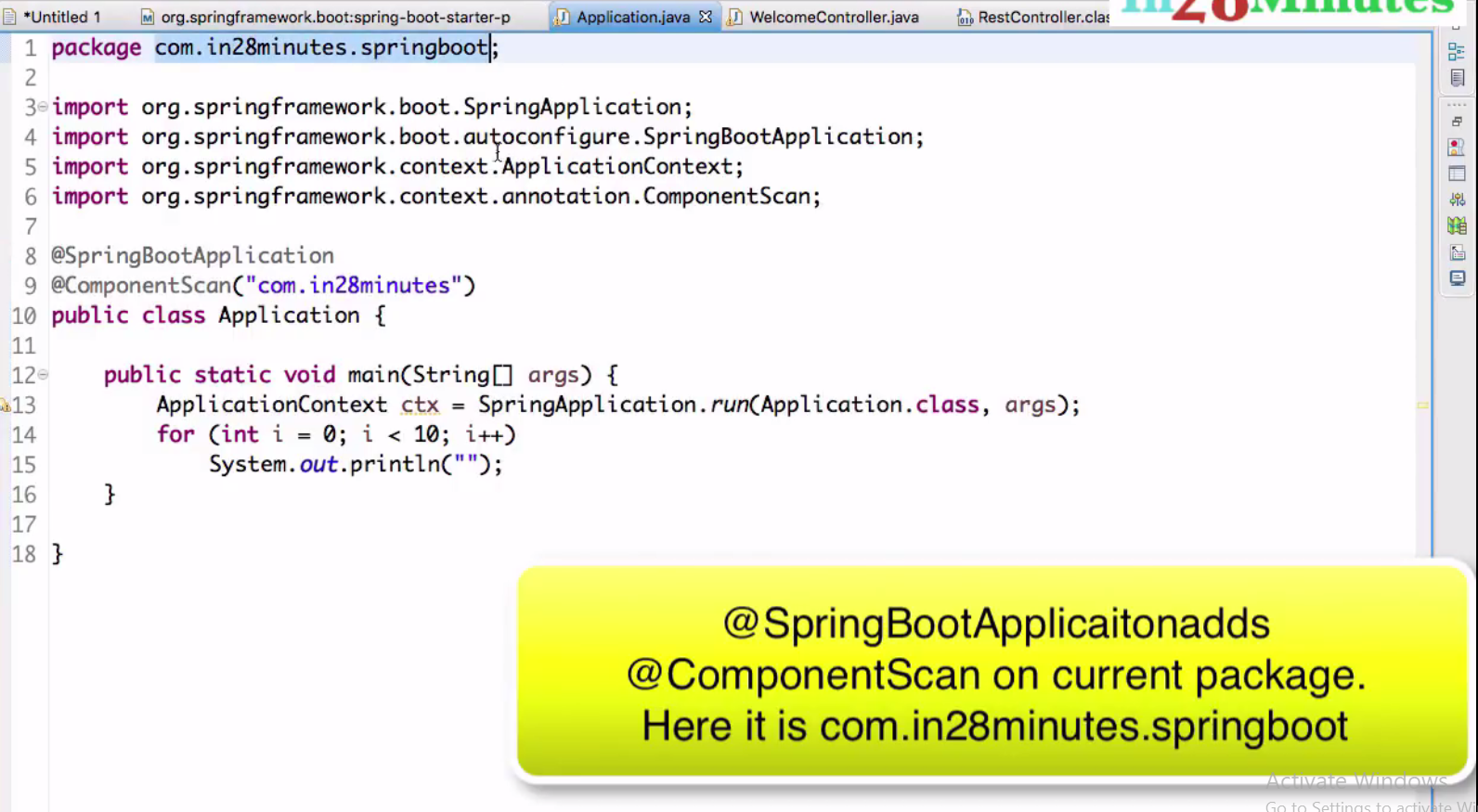
|  |
| --- |
| **package com.in28minutes.springboot.controller;**  **import org.springframework.boot.SpringApplication;**  **import org.springframework.boot.autoconfigure.SpringBootApplication;**  **import org.springframework.context.ApplicationContext;**  **import org.springframework.context.annotation.ComponentScan;**  **@SpringBootApplication**  **@ComponentScan("com.in28minutes")**  **public class Application {**  **public static void main(String[] args) {**  **ApplicationContext ctx = SpringApplication.run(Application.class, args);**  **}**  **}** |

****

**Now we can see that everything looks good. Server got started at application launch because Spring now able to find the particular annotation what it was looking for.**

**So the logic is Once the Spring start component scan of the defined package, it instantiate and create the object of all the component class and where ever it find @Autowired , it inject that particular component there.**

**This is the way spring framework work. By default component scan happen in the current package and if it does not found then it start scanning in other packages. If all the component classes are in the same package then we don’t need to define component scan[@ComponentScan ("com.in28minutes")] at application launcher explicitly.**

****

**Q: What are the component Spring looks for :**

**Ans: @Controller, @RestController (Extenuation of @Controller), @Component, @Service, @Repository and so on. Generally four (@Controller, @Component, @Service, @Repository), so far we have used three out or these four components.**

**spring-boot-starter-parent:**

**<artifactId>spring-boot-starter-parent</artifactId>**

**<packaging>pom</packaging>**

**<name>Spring Boot Starter Parent</name>**

**<description>Parent pom providing dependency and plugin management for applications built with Maven</description>**

**Step03.md: Understanding Spring Boot Magic:**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step03.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step03.md)

**spring-boot-starter-web**

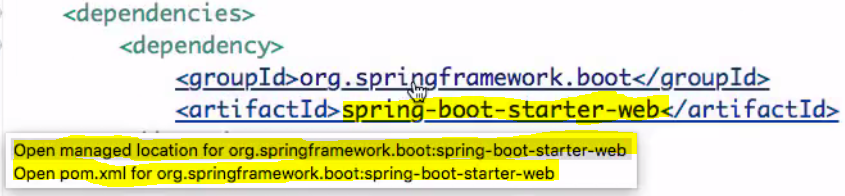
**Now we will talk about some spring boot magic like what is happening behind [spring-boot-starter-web]. We saw that Tomcat is default embedded container.**

**This is either use in developing Rest services or in Spring MVC based web application. Now a days this (spring-boot-starter-web) being used mostly in developing Rest services.**

**Few questions that comes in our mind:**

1. **What version of spring-boot-starter-web is used?**
2. **What dependencies do the spring-boot-starter-web brings in?**
3. **What auto configurations do the spring-boot-starter-web brings in?**

**To know about this > Go TO POM.xml > Hover the mouse on this dependency [spring-boot-starter-web] and press control**

****

**The first option: Open managed location: shows the version of spring-boot-starter-web: This opens up the parent POM which is basically manages spring boot dependencies. The version what it shows here is nothing but the same version of [spring-boot-starter-parent], that we had already mentioned in the spring boot parent section.**

**The second option: Here all the default dependencies are mentioned which gets downloaded as a part of spring-boot-starter-web like (Embedded Tomcat , Jackson Binding , log4j,hibernate validator, spring MVC, spring core…see in the maven dependencies ). So these are default dependences that spring-boot-starter-web bring in.**

|  |
| --- |
| **<?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 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>**  **<modelVersion>4.0.0</modelVersion>**  **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starters</artifactId>**  **<version>1.4.0.RELEASE</version>**  **</parent>**  **<artifactId>spring-boot-starter-web</artifactId>**  **<name>Spring Boot Web Starter</name>**  **<description>Starter for building web, including RESTful, applications using Spring**  **MVC. Uses Tomcat as the default embedded container</description>**  **<url>http://projects.spring.io/spring-boot/</url>**  **<organization>**  **<name>Pivotal Software, Inc.</name>**  **<url>http://www.spring.io</url>**  **</organization>**  **<properties>**  **<main.basedir>${basedir}/../..</main.basedir>**  **</properties>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-tomcat</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.hibernate</groupId>**  **<artifactId>hibernate-validator</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>com.fasterxml.jackson.core</groupId>**  **<artifactId>jackson-databind</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework</groupId>**  **<artifactId>spring-web</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework</groupId>**  **<artifactId>spring-webmvc</artifactId>**  **</dependency>**  **</dependencies>**  **</project>** |

**We have answered of two questions. Now the last question**

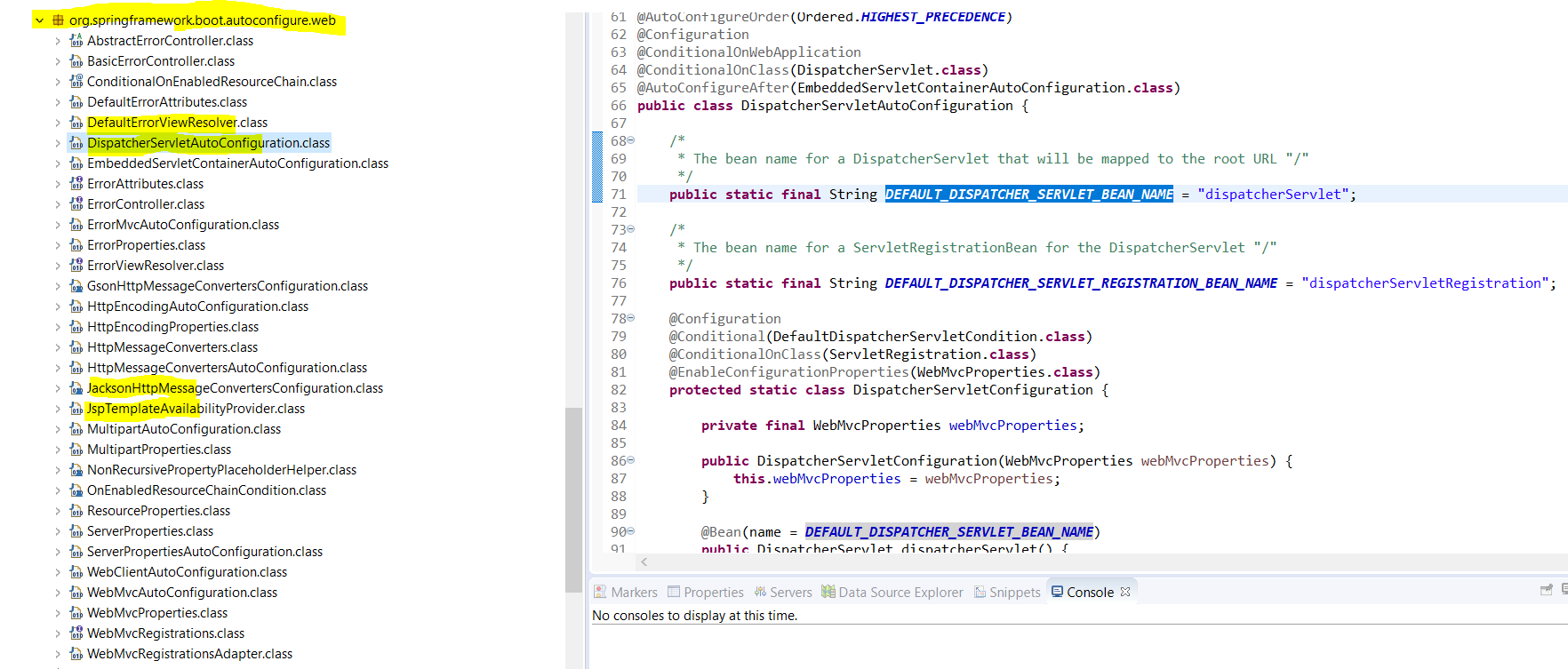
1. **What auto configurations do the spring-boot-starter-web brings in?**

**Earlier when we used to develop spring application then we need to do manually configure like following.**

1. **Dispatcher servlet & Application controller class**
2. **HandlerMapping for controller class and view page**
3. **Error handling , url mapping and so on**

**But in spring boot these things are taking care by Spring Boot itself. We don’t need to configure these things manually in the spring boot application. In the Maven dependency we get one spring-boot-autpconfigure.jar which actually does auto configure and many more things.**

**To get more GOTO spring boot application Maven Dependency > Select Spring-boot-autoconfigure-1.4.0.Release.jar > For Web autoconfigure Go to org.springframework.boot.autoconfigure.web**

****

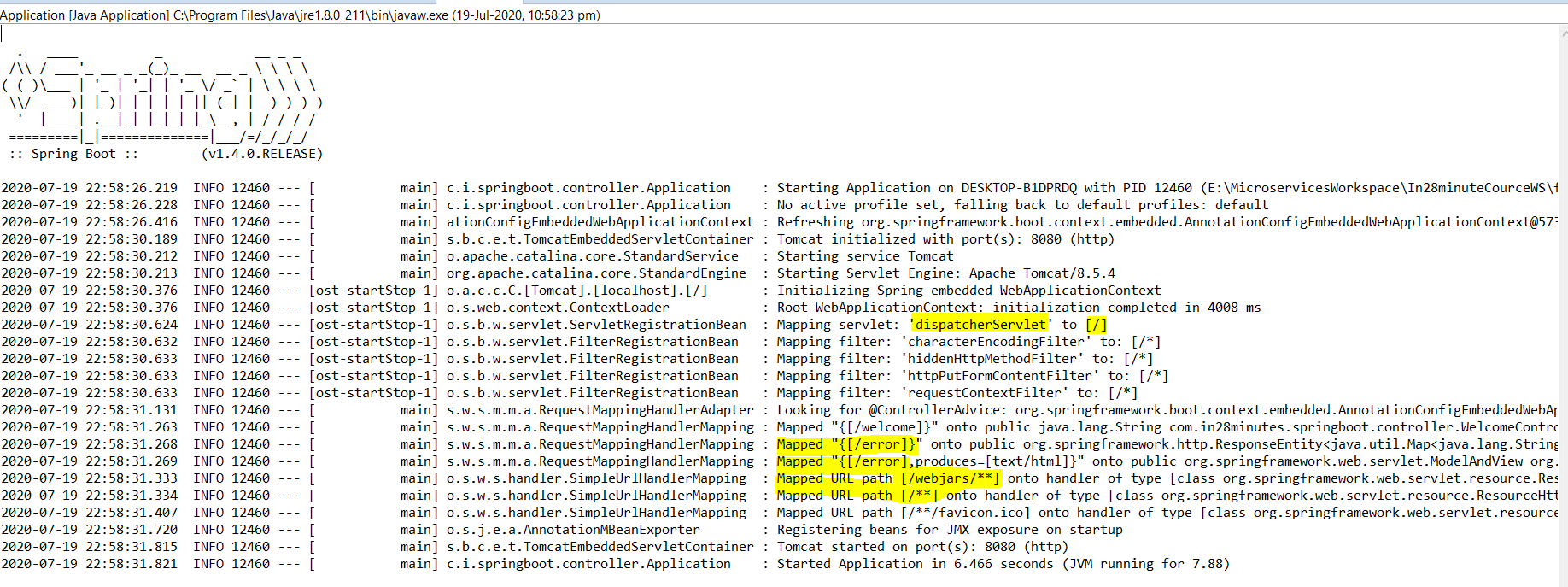
**While running Tomcat server we can see all the autoconfigure things in the server console.**

1. **Contains all the dependencies related to spring web and rest base development like spring-core, spring-web, Jackson-databind, embedded tomcat, data-validator.**
2. **Autoconfiguration for :**

**Dispatcher servlet & Application controller class**

**HandlerMapping for controller class and view page**

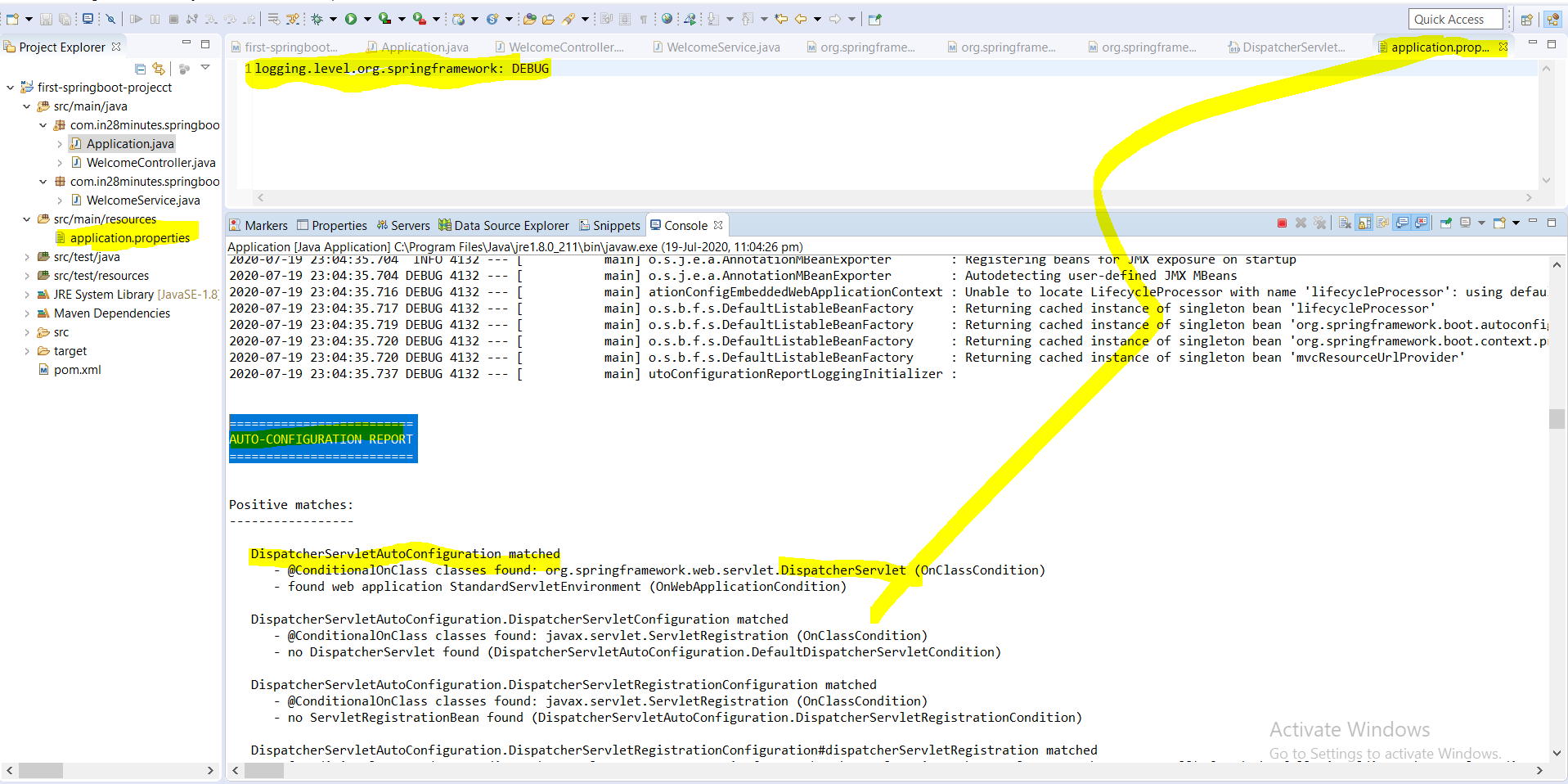
**Error handling, url mapping and so on**

****

**As we can see the dispatcher servlet and some error mapped which we have not configure, even though they are coming here because spring boot takes care these configuration itself. Developer doesn’t need to configure these things.**

**For more detail of Auto configure :**

1. **Create one application.properties file inside resource package.**
2. **Put : logging.level.org.springframework: DEBUG**
3. **Start the server in debug mode**
4. **In server console search ====AUTO-CONFIGURATION REPORT=========================**

****

**So here we can see the list of the auto configuration (positive matches) done by the spring boot and also the list of configuration (negative matches) not done by the spring boot.**

**So these all about spring-boot-starter-web:**

**Now let’s try to understand:**

**Step04.md : spring-boot-starter-parent**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step04.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step04.md)

**What You Will Learn during this Step:**

* **Understand Starter Parent**
* **How to override things defined in Starter Parent**
* **Other starter projects**

**Spring-boot-starter-parent: It defines the following**

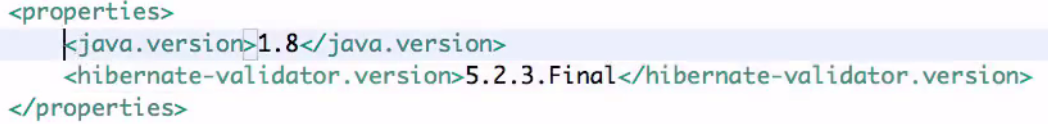
1. **Dependency versioning**
2. **Default plugins**
3. **Java version**

**So if open up the spring-boot-starter-parent then we will get the above defined things. In other word Parent pom providing dependency and plug-in management for applications built with Maven.**

**One of the great thing of the spring boot project is that it makes sure that all these versions work together really well without any compatibility issue, means spring boot ensures all these versions of framework are compatible with each other.**

**This is the great relief of developer while upgrading application spring version from lower version to higher or any version then spring boot will take care all the compatibility.**

**If anything like version of default plug-in we want to change then we can override using <properties> tag as we have done in case of java version from 1.6 to 1.8**

****

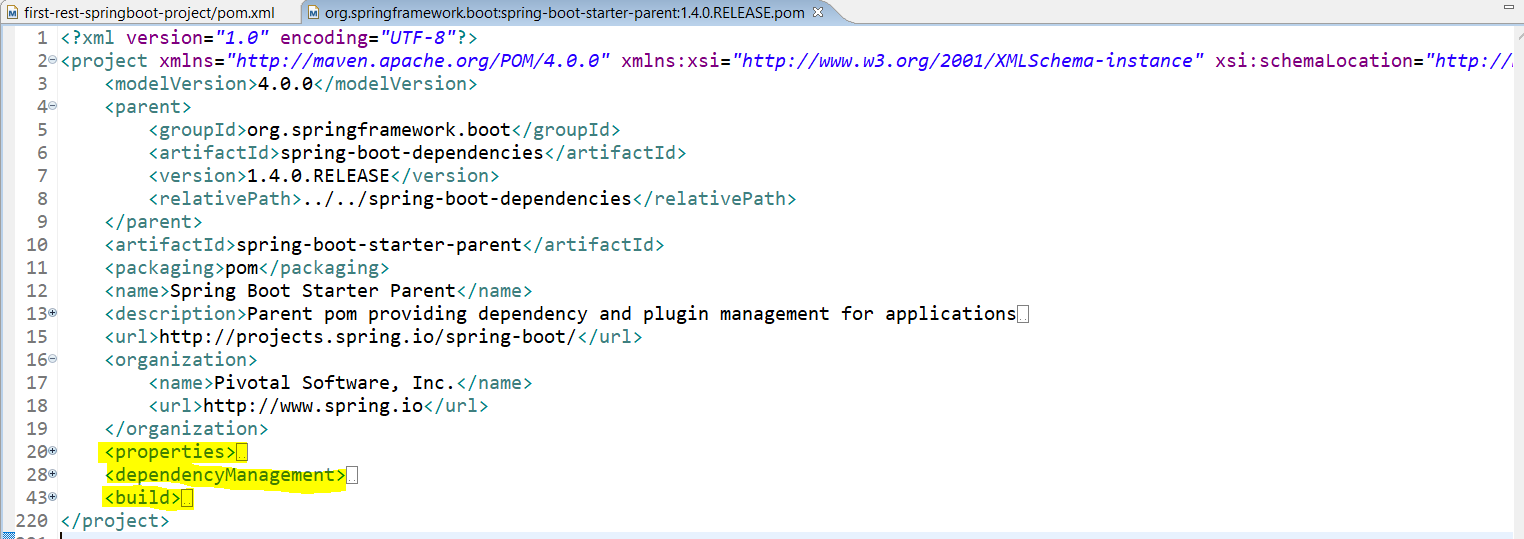
**Here we have override java version and hibernate Validator version inside properties tag. But it is not recommended to override the version provided by the spring boot , otherwise we have to be very cautious when overriding version of dependencies managed by spring boot starter parent because spring boot already make sure that all the version are compatible.**

**Other Starter Projects**

* **spring-boot-starter-web-services : For Soap based web service**
* **spring-boot-starter-test**
* **spring-boot-starter-jdbc**
* **spring-boot-starter-security**
* **spring-boot-starter-data-jpa**
* **spring-boot-starter-data-rest**
* **More at** [**https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter**](https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#using-boot-starter)

**To know more about spring-boot-starter-parent: Go to POM > Press ctrl > hover the mouse and click**

|  |
| --- |
| **<artifactId>spring-boot-starter-parent</artifactId>**  **<packaging>pom</packaging>**  **<name>Spring Boot Starter Parent</name>**  **<description>Parent pom providing dependency and plugin management for applications built with Maven</description>** |
| **Versioning- using <properties> tag** |
| **Plug-in : inside <build> tag** |

****

**That is all about spring-boot-starter-parent.**

**Step05.md: Spring Boot Vs Spring**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step05.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step05.md)

**What You Will Learn during this Step:**

* **Spring Boot vs. Spring**
* **What Spring Boot is Not!**
* **The core concept of spring is Dependency Injection and the things like Auotwiring, Components, and ComponentScan.**
* **The other thing is to reduce the plumbing code in the enterprise application**

**Spring Boot vs Spring**

### Spring

* **Spring is just a dependency injection framework. Spring focuses on the "plumbing" of enterprise applications so that teams can focus on application-level business logic, without unnecessary ties to specific deployment environments.**
* **If we go to first half of the 2000 decade! EJBs**
* **EJB was ruling in the development and EJBs were NOT easy to develop.**
* **We had to Write a lot of xml and plumbing code to get EJBs running**
* **And the most important thing was Impossible to Unit Test**
* **If want to include JDBC then for - Writing simple JDBC Code involved a lot of plumbing**

**Spring framework started with aim of making Java EE development simpler.**

* **Goals of spring is**
* **Make applications testable. i.e. easier to write unit tests**
* **Reduce plumbing code of JDBC and JMS**
* **Simple architecture. Minus EJB.**
* **Integrates well with other popular frameworks.**

### Applications with Spring Framework

* **Over the next few years, a number of applications were developed with Spring Framework**
* **Testable but**
* **Lot of configuration (XML and Java)**
* **Developing Spring Based application need configuration of a lot of beans!**
* **Integration with other frameworks need configuration as well!**
* **In the last few years, focus is moving from monolith applications to microservices. We need to be able to start project quickly. Minimum or Zero start up time**
* **Framework Setup**
* **Deployment - Configurability**
* **Logging, Transaction Management**
* **Monitoring**
* **Web Server Configuration**

**Spring Boot**

* **Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.**
* **We take an opinionated view of the spring platform and third-party libraries so you can get started with minimum fuss.**
* **Example Problem Statements**
* **You want to add Hibernate to your project. You dont worry about configuring a data source and a session factory. I will do if for you!**
* **Goals** 
  + **Provide quick start for projects with Spring.**
  + **Be opinionated but provide options.**
  + **Provide a range of non-functional features that are common to large classes of projects (e.g. embedded servers, security, metrics, health checks, externalized configuration).**

#### What Spring Boot is NOT?

* **It’s not an app or a web server**
* **Does not implement any specific framework - for example, JPA or JMS**
* **Does not generate code**

**Step06.md (create all service for survey and questions)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step06.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step06.md)

**Now here from onwards we will focus more on business logic for our Rest Service.**

**Create a rest service for survey questions.**

## What You Will Learn during this Step:

* **Create a REST Service for Retrieving all questions for a survey**
* **Autowire SurveyService**
* **Create @GetMapping("/surveys/{surveyId}/questions")**
* **Use @PathVariable String surveyId**
* [**http://localhost:8080/surveys/Survey1/questions/**](http://localhost:8080/surveys/Survey1/questions/)
* **How does the Bean get converted to a JSON?**
* **Auto Configuration: If Jackson jar is on the class path, message converters are auto created! (Search in log :Creating shared instance of singleton bean 'mappingJackson2HttpMessageConverter')**

## Some Theory

* **What is REST?**
* **Architectural style for the web. REST specifies a set of constraints or limitations, which specify, what should be done and what should not do. It is not technology or protocol. Below are the following constraints.** 
  + **There must be Client & Server: Server is (service provider) should be different from a client (service consumer).**
    - **Enables loose coupling and independent evolution of server and client as new technologies emerge.**
  + **Each service should be stateless i.e. it should not be depended on any intermediate changes, every time we will get same data if it did not modify at the server level.**
  + **Each Resource has a resource identifier. (Ex: ("/surveys/{surveyId}/questions") so that it could be identified very easily.**
  + **It should be possible to cache response so that for the same request we don’t need to hit the server resource every time and performance can be improved.**
  + **Consumer of the service may not have a direct connection to the Service Provider. Response might be sent from a middle layer cache.**
  + **A resource can have multiple representations. Resource can modified through a message in any of the these representations.**

**What You Will Learn during this Step:**

* **We want to prepare for creating a Rest Service**
* **Survey**
* **Question**
* **SurveyService**
* **We use hard-coded data to get started**

**Files List:**

1. **POM.xml: similar to previous step** 
   * + - **Spring-boot-starter-parent**
       - **Java-version 1.8**
       - **Spring-boot-starter-web**
       - **Spring-boot-maven-plugin**
2. **src/main/java/com/in28minutes/springboot/Application.java**
3. **src/main/java/com/in28minutes/springboot/model/Question.java**

**Here in this model class we will have following variables and question will have answer of multiple option.**

**private String id;**

**private String description;**

**private String correctAnswer;**

**private List<String> options;**

1. **src/main/java/com/in28minutes/springboot/model/Survey.java**

**The survey will have id , title ,description and several list of questions**

**private String id;**

**private String title;**

**private String description;**

**private List<Question> questions;**

1. **src/main/java/com/in28minutes/springboot/service/SurveyService.java**

**Here in this class we will write main business logic.**

* + - * **We will create the list of questions**
      * **We will create the list of surveys**
      * **Method to retrieve all surveys (public List<Survey> retrieveAllSurveys()**
      * **Method to retrieve particular survey based on surveyId (public Survey retrieveSurvey(String surveyId)**
      * **Method to retrieve list of questions based on survey id -public List<Question> retrieveQuestions(String**

**surveyId)**

* + - * **Method to retrieve particular question based on survey id and question id [public Question**

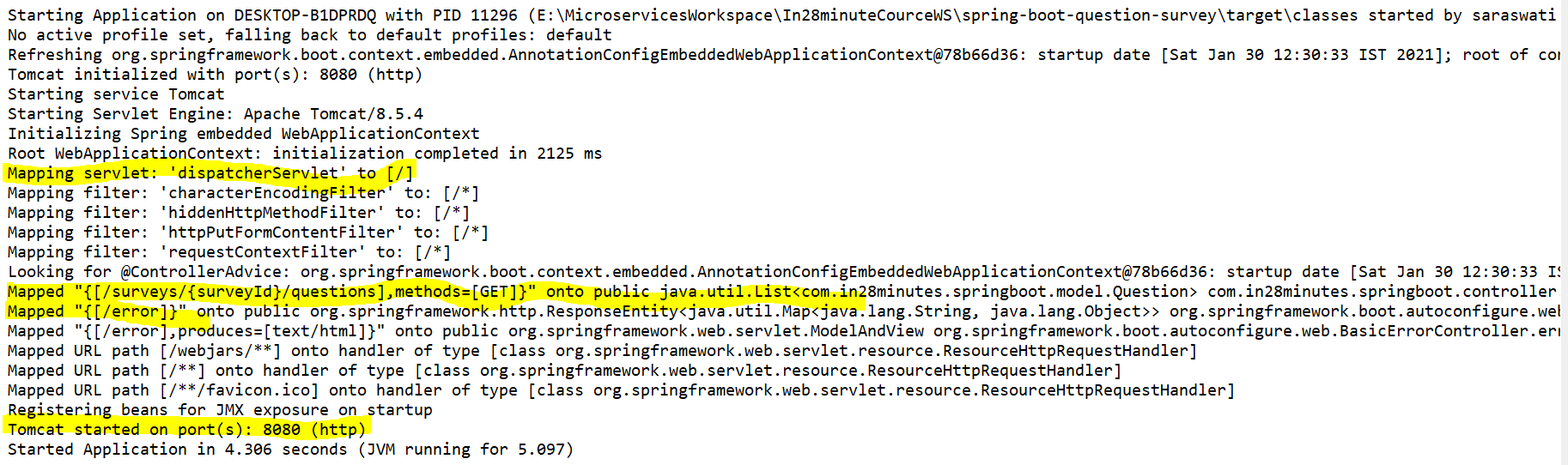
**retrieveQuestion(String surveyId, String questionId)]**

* + - * **Method to add particular question into the particular seuvey public Question addQuestion(String**

**surveyId, Question question)**

1. **src/main/java/com/in28minutes/springboot/WelcomeController.java**

|  |
| --- |
| **For code snippet :**  [**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step06.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step06.md) |

****

|  |  |
| --- | --- |
| **Question.java**  **package com.in28minutes.springboot.model;**  **import java.util.List;**  **public class Question {**  **private String id;**  **private String description;**  **private String correctAnswer;**  **private List<String> options;**  **public Question() { }**  **public Question(String id, String description, String correctAnswer,**  **List<String> options) {**  **super();**  **this.id = id;**  **this.description = description;**  **this.correctAnswer = correctAnswer;**  **this.options = options;**  **}// Getters and setters**  **@Override**  **public String toString() {**  **return String.*format*("Question [id=%s, description=%s, correctAnswer=%s, options=%s]", description, correctAnswer, options);**  **}**  **// Override hashCode and equals method**  **}** | **Survey.java**  **package com.in28minutes.springboot.model;**  **import java.util.List;**  **public class Survey {**  **private String id;**  **private String title;**  **private String description;**  **private List<Question> questions;**  **public Survey(String id, String title, String description,**  **List<Question> questions) {**  **super();**  **this.id = id;**  **this.title = title;**  **this.description = description;**  **this.questions = questions;**  **}**  **// getters and setters**  **public void setQuestions(List<Question> questions) {**  **this.questions = questions;**  **}**  **@Override**  **public String toString() {**  **return "Survey [id=" + id + ", title=" + title + ", description="**  **+ description + ", questions=" + questions + "]";**  **}**  **}** |
| **package com.in28minutes.springboot.service;**  **import java.math.BigInteger;**  **import java.security.SecureRandom;**  **import java.util.ArrayList;**  **import java.util.Arrays;**  **import java.util.List;**  **import org.springframework.stereotype.Component;**  **import com.in28minutes.springboot.model.Question;**  **import com.in28minutes.springboot.model.Survey;**  **@Component**  **public class SurveyService {**  **private static List<Survey> surveys = new ArrayList<>();**  **static {**  **Question question1 = new Question("Question1","Largest Country in the World", "Russia", Arrays.asList("India", "Russia", "United States", "China"));**  **Question question2 = new Question("Question2","Most Populus Country in the World", "China", Arrays.asList("India", "Russia", "United States", "China"));**  **Question question3 = new Question("Question3","Highest GDP in the World", "United States", Arrays.asList("India", "Russia", "United States", "China"));**  **Question question4 = new Question("Question4","Second largest english speaking country", "India", Arrays.asList("India", "Russia", "United States", "China"));**  **List<Question> questions = new ArrayList<>(Arrays.asList(question1,**  **question2, question3, question4));**  **Survey survey = new Survey("Survey1", "My Favorite Survey",**  **"Description of the Survey", questions);**    **Survey survey1 = new Survey("Survey2", "My Second Survey", "Desc of 2nd Survey", questions);**  **surveys.add(survey);**  **surveys.add(survey1);**  **}**  **public List<Survey> retrieveAllSurveys() {**  **return surveys;**  **}**  **public Survey retrieveSurvey(String surveyId) {**  **for (Survey survey : surveys) {**  **if (survey.getId().equals(surveyId)) {**  **return survey;**  **}**  **}**  **return null;**  **}** | **public List<Question> retrieveQuestions(String surveyId) {**  **Survey survey = retrieveSurvey(surveyId);**  **if (survey == null) {**  **return null;**  **}**  **return survey.getQuestions();**  **}**  **public Question retrieveQuestion(String surveyId, String questionId) {**  **Survey survey = retrieveSurvey(surveyId);**  **if (survey == null) {**  **return null;**  **}**  **for (Question question : survey.getQuestions()) {**  **if (question.getId().equals(questionId)) {**  **return question;**  **}**  **}**  **return null;**  **}**  **private SecureRandom random = new SecureRandom();**  **public Question addQuestion(String surveyId, Question question) {**  **Survey survey = retrieveSurvey(surveyId);**  **if (survey == null) {**  **return null;**  **}**  **String randomId = new BigInteger(130, random).toString(32);**  **question.setId(randomId);**  **survey.getQuestions().add(question);**  **return question;**  **}**  **}** |
| **Controller class**  **package com.in28minutes.springboot.controller;**  **import java.util.Arrays;**  **import java.util.List;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.web.bind.annotation.GetMapping;**  **import org.springframework.web.bind.annotation.PathVariable;**  **import org.springframework.web.bind.annotation.PostMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **import com.in28minutes.springboot.model.Question;**  **import com.in28minutes.springboot.model.Survey;**  **import com.in28minutes.springboot.service.SurveyService;**  **@RestController**  **class SurveyController {**  **@Autowired**  **private SurveyService surveyService;**  **@GetMapping("/surveys/{surveyId}/questions")**  **public List<Question> retrieveQuestions(@PathVariable String surveyId) {**  **return surveyService.retrieveQuestions(surveyId);**  **}**    **@GetMapping("/surveys")**  **public List<Survey> retrieveAllSurvery(){**  **return surveyService.retrieveAllSurveys();**  **}**    **@GetMapping("/surveys/{surveyId}")**  **public Survey retrieveSurvey(@PathVariable String surveyId){**  **//return surveyService.retrieveAllSurveys();**  **return surveyService.retrieveSurvey(surveyId);**  **}**    **@GetMapping("/surveys/{surveyId}/questions/{questionId}")**  **public Question retrieveParticularQuestion(@PathVariable String surveyId, @PathVariable String questionId) {**  **return surveyService.retrieveQuestion(surveyId, questionId);**  **}**  **@PostMapping("/surveys/{surveyId}/questions")**  **public Question addParticularQuestion(@PathVariable String surveyId, @PathVariable Question questionId) {**    **questionId = new Question("MyOwnQuestion",**  **"What is the capital of India?", "Delhi", Arrays**  **.asList("New Delhi", "Kolkata", "Lucknow", "Bangalore"));**    **return surveyService.addQuestion("Survey1", questionId);**  **}**  **}** | |

**So here the numbers of annotations used are:**

**@RestController**

**@GetMapping**

**@PostMapping**

**@PathVariable (used to pass variable)**

**Step07.md (Creating Rest Service with @GetMapping and @PathVariable)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step07.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step07.md)

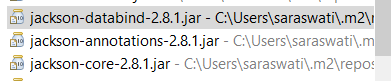
**Question: Here in most of the cases we are return List, now the question comes how this list gets converted into JSON?**

* **How does the Bean get converted to a JSON?**
* **Auto Configuration: If Jackson jar is on the class path, message converters are auto created! (Search in log :Creating shared instance of singleton bean 'mappingJackson2HttpMessageConverter')**

**It happens because of message converter, so if we restart the application server and look at the log then we would see that there are number of message converters which are present inside the log. These message converters are auto configured by spring boot. Here we are returning the list (which is nothing but an object) which we want to convert into JSON (or JSON to object or JSON to xml or XML to JSON). This conversion is done by message converter configured in spring boot.**

**Here we have Jackson-databind is one of the framework which are being used in spring boot which converts object to Json and Json to Object. So whenever spring boot find this Jackson-databind.jar into the class path it automatically converts the object returned by the RestController by using message converter. This happens automatically, we don’t need to think about this conversion.**

**This Jackson-datbind.jar comes with spring-boot-starter-web**

****

**This is one of the transitive dependencies of the spring boot project.**

**Step08.md (Second REST service to retrieve a specific question )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step08.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step08.md)

**Request methods:**

**GET : Retrieve details of the resource “/survey/{surveyId}/questions”**

**POST : Create a new resource “/survey/{surveyId}/questions”**

**PUT : Update an existing resource “/survey/{surveyId}/questions”**

**PATCH : Update part of the resource “/survey/{surveyId}/questions”**

**DELETE: Delete a resource “/survey/{surveyId}/questions”**

**Step09.md (Spring Boot Developer Tools & Live Reload: Develop Faster! )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step09.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step09.md)

## What You Will Learn during this Step:

* **I hate the fact that I've to stop and start the server each time. Can somebody save me?**
* **Yeah. Spring Boot Developer Tools**
* **By default, any entry on the classpath that points to a folder will be monitored for changes.**
* **These will not trigger restart - /META-INF/maven, /META-INF/resources ,/resources ,/static ,/public or /templates**
* **Folders can be configured : spring.devtools.restart.exclude=static/,public/**
* **Additional Paths : spring.devtools.restart.additional-paths**
* **LiveReload**[**http://livereload.com/extensions/**](http://livereload.com/extensions/)
* **Technology in progress!! So, expect a few problems!**
* **Programming Tip**
* **Become an expert at your IDE -**[**https://www.youtube.com/watch?v=dN9GYsG1v\_c**](https://www.youtube.com/watch?v=dN9GYsG1v_c)

## Useful Snippets and References

**First Snippet**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-devtools</artifactId>**

**<optional>true</optional>**

**</dependency>**

**Step10.md (Spring Boot Developer Tools & Live Reload: Develop Faster! )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step10.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step10.md)

## What You Will Learn during this Step:

* **Create a REST Service to add a new question to survey**
* **@PostMapping("/surveys/{surveyId}/questions")**
* **@RequestBody Question question**
* **What should be Response Status for create?**
* **ResponseEntity.created(location).build()**
* **ResponseEntity.noContent().build()**
* **Using Postman :**[**https://www.getpostman.com**](https://www.getpostman.com/)
* **URL to POST to -**[**http://localhost:8080/surveys/Survey1/questions**](http://localhost:8080/surveys/Survey1/questions)

## Useful Snippets and References

**Sample Body for POST Request**

**{"description":"Second Most Populous Country in the World","correctAnswer":"India","options":["India","Russia","United States","China"]}**

**Q1- What should be the structure of request body?**

**The structure of request body should be exactly same as we retrieve the question using get method so the sample request is shown in the sample body above.**

**Q2- What will it be mapped to Question object?**

**To map this request body we will use @RequestBody annotation**

**Q3- What should be return?**

**When creation of new question gets success then it should return URI of the resource in the Response header and status will say created. To build the URI we have used ServletUriComponentsBuilder which is one of utility class in the spring framework which that extracts information from the current HttpServletRequest to build resource URI.**

**Q4- What should be the response status?**

**The response status will be shown as created. And to create the response, status will be created by using ResponseEntity (it is Extension of HttpEntity that adds an HttpStatus status code.Used in RestTemplate as well @Controller methods.). Since in the POST request we used to save data into data base by creating new record so in this case instead of returning any particular return type (Class object or String or other) we used to return Http status using below line of code.**

**return ResponseEntity.status(HttpStatus.CREATED).build();**

**Note : Here in the POST we are sending two things surveyId and Question detail. SurveyId we are sending through @PathVariable. Since Question will contain several information like id, description, correctAnswer, optionList in Question object since an object can have one or several variable, so to map the question object we will use @RequestBody annotation as shown below in the code snippet.**

**First Snippet**

**@PostMapping("/surveys/{surveyId}/questions")**

**ResponseEntity<?> add(@PathVariable String surveyId, @RequestBody Question question) {**

**Question newQuestion = surveyService.addQuestion(surveyId, question);**

**if (newQuestion == null) {**

**return ResponseEntity.noContent().build();// Return No content found response code(204). So now if we hit incorrect Url then we will not get Internal server 500 error**

**}**

**URI location = ServletUriComponentsBuilder.fromCurrentRequest()**

**.path("/{id}").buildAndExpand(createdTodo.getId()).toUri();**

**return ResponseEntity.created(location).build();**

**// return ResponseEntity.*status*(HttpStatus.*CREATED*).build()**

**}**

|  |
| --- |
|  |
|  |
| Handeled null in the code to get proper response instead of getting internal server error |

## ResponseEntity

**ResponseEntity is utility class which helps in creating HTTP response including headers, body, and status. While @ResponseBody puts the return value into the body of the response, ResponseEntity also allows us to add headers and status code.Extension of**[**HttpEntity**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpEntity.html)**that adds a**[**HttpStatus**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/HttpStatus.html)**status code. Used in RestTemplate as well @Controller methods.**

**Here in the above code snippet we don’t need to write the code to Get URI to get location and passing the same location to get HttpStatus.**

**return ResponseEntity.*status*(HttpStatus.*CREATED*).build()**

**System.*out*.println ("Status:” + ResponseEntity.*created* (location).build ());**

**Above line will return just HttpStatus code only. If we put the above into SOP then will get the result as given below.**

**Status::::<201 Created,{Location=[http://localhost:8080/surveys/Survey1/questions/3okbgeufgge8odammc63spv6o2]}>**

**return ResponseEntity.*created*(location).build();**

**System.*out*.println("Status::::" +ResponseEntity.*status*(HttpStatus.*CREATED*).build());**

**Here apart from status it will return the location where the record got created**

**Status ::::< 201 Created, {}>**

**Note: So whenever we want to use POST request to create a new resource then we would create a POST request to a specific URL and we would return an http status of created.**

**Now the number of annotation we have used so far:**

**So here the numbers of annotations used are:**

**@RestController**

**@GetMapping**

**@PostMapping**

**@PathVariable (used to pass variable)**

**@RequestBody Question question**

**ResponseEntity Cl**

**Step11.md (Understand Content Negotiation & Deliver XML response )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step11.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step11.md)

**What You Will Learn during this Step:**

* **Understand Content Negotiation**
* **Accept:application/xml**
* **Deliver XML Responses from the REST Services**
* [**http://localhost:8080/surveys/Survey1/questions/**](http://localhost:8080/surveys/Survey1/questions/)

**Useful Snippets and References**

**First Snippet**

**<dependency>**

**<groupId>com.fasterxml.jackson.dataformat</groupId>**

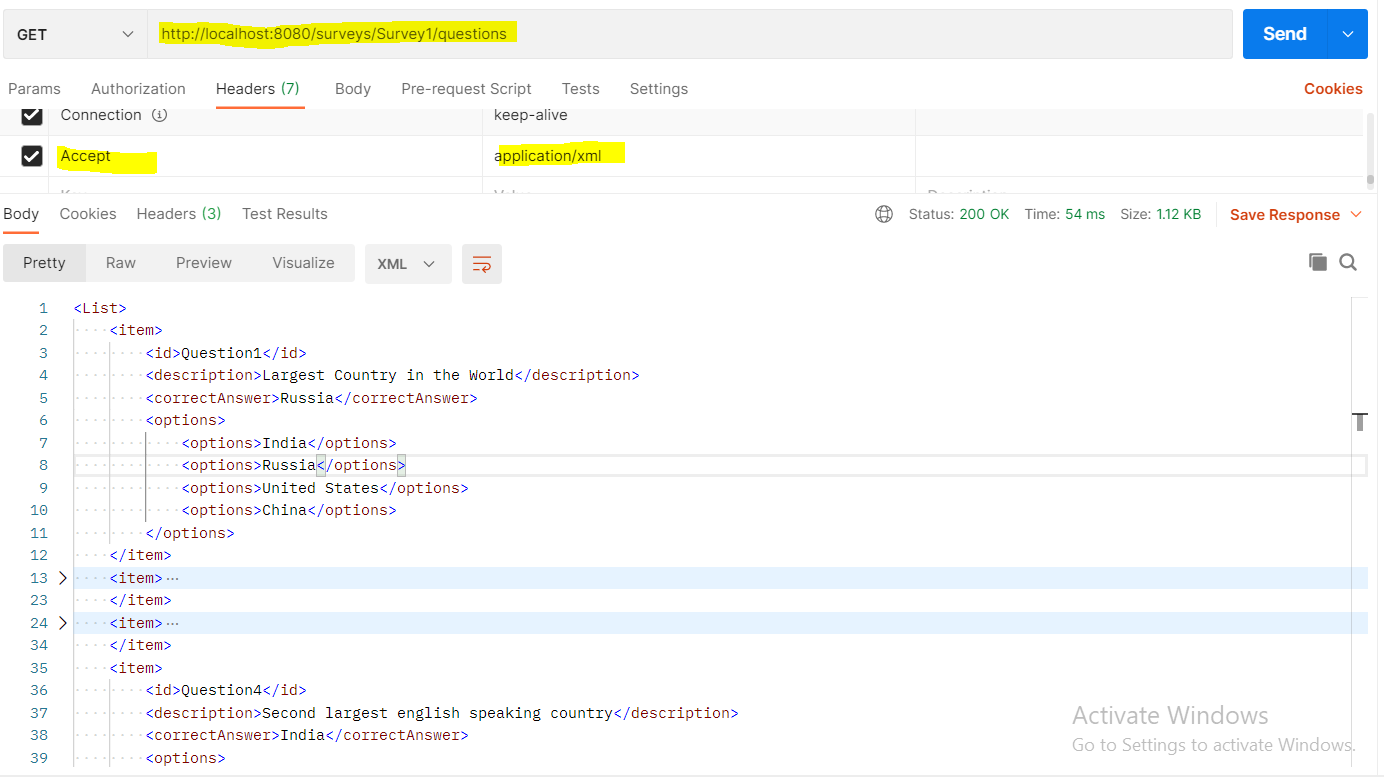
**<artifactId>jackson-dataformat-xml</artifactId>**

**</dependency>**

**Content Negotiation is nothing but the format of the response which the service consumer (client) demands to the service provider. They can demand the response in the format like JSON, XML etc.**

**To create the response in xml in the REST service we just need to include one dependency [Jackson-dataformat-xml] in the POM.xml.**

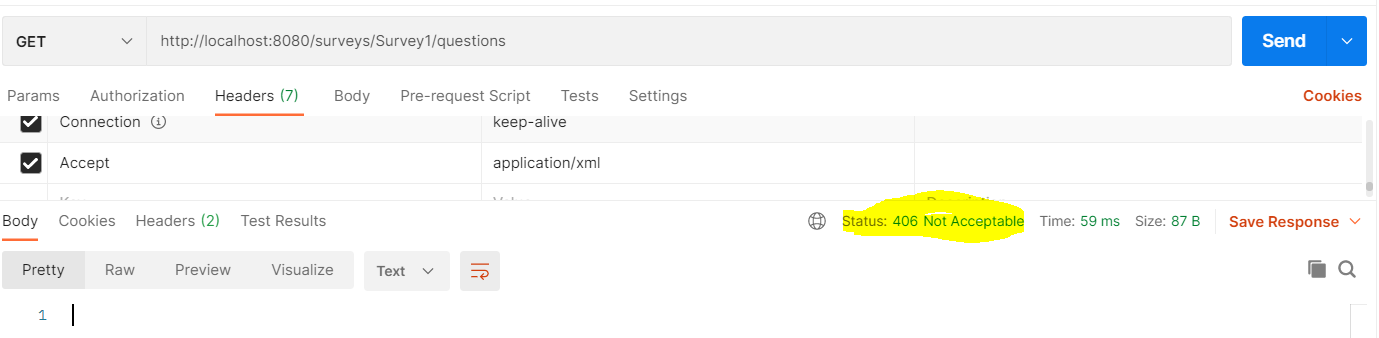
**While testing in Postman we have to write key – accept and value - application/xml**

****

|  |  |
| --- | --- |
| **<List>**  **<item>**  **<id>Question1</id>**  **<description>Largest Country in the World</description>**  **<correctAnswer>Russia</correctAnswer>**  **<options>**  **<options>India</options>**  **<options>Russia</options>**  **<options>United States</options>**  **<options>China</options>**  **</options>**  **</item>**  **<item>**  **<id>Question2</id>**  **<description>Most Populus Country in the World</description>**  **<correctAnswer>China</correctAnswer>**  **<options>**  **<options>India</options>**  **<options>Russia</options>**  **<options>United States</options>**  **<options>China</options>**  **</options>**  **</item>** | **<item>**  **<id>Question3</id>**  **<description>Highest GDP in the World</description>**  **<correctAnswer>United States</correctAnswer>**  **<options>**  **<options>India</options>**  **<options>Russia</options>**  **<options>United States</options>**  **<options>China</options>**  **</options>**  **</item>**  **<item>**  **<id>Question4</id>**  **<description>Second largest english speaking country</description>**  **<correctAnswer>India</correctAnswer>**  **<options>**  **<options>India</options>**  **<options>Russia</options>**  **<options>United States</options>**  **<options>China</options>**  **</options>**  **</item>**  **</List>** |

**So as soon as we include dependency the spring boot automatically configure the xml conversion automatically and hence we can deliver the response in JSON or XML.**

**Note: If we do not include the xml conversion dependency and try to get xml format in the Postman then we will get 406- Not Acceptable response as shown below in the image.**

****

**Step12.md( Spring Initializr to create spring boot project: we have already learnt)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step12.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step12.md)

**What You Will Learn during this Step:**

* **Spring Initializr**
* [**https://start.spring.io**](https://start.spring.io/)
* **Create a few projects!**

**Step13.md( Spring Boot Actuator : Monitor your Spring Boot Application)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step13.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step13.md)

**What You Will Learn during this Step:**

* **Spring Boot Actuator is kind of monitoring application of your Rest services. Un-till now we have created several Rest services, so want to expose metrics, status of the system whether it has free space, what are the beans are configured, what the property that are exposed, what are mapping identifier we are providing . All these things can be exposed outside the world using Spring Boot Actuator. In-short ----**
* **/env, /metrics, /trace, /dump, /shutdown, /beans, / autoconfig, /configprops, /mappings**

**HAL Browser**

**It is a kind of user friendly UI provided for Spring Boot Actuator to display all the things in user friendly manner. In other words we can say that Spring Boot Actuator provides the data and HAL browser is the one through which we can browse the data in user friendly environment.**

* [**http://localhost:8080/actuator/**](http://localhost:8080/actuator/)
* **Execute individual REST Services for each of above**
* **Programming Tip**
* **Use static code analysis -**[**https://www.youtube.com/watch?v=rB\_BaftN3nE**](https://www.youtube.com/watch?v=rB_BaftN3nE)

**Useful Snippets and References: To get these two functionality we need to add the below two dependencies in POM.xml**

**First Snippet**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-actuator</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.data</groupId>**

**<artifactId>spring-data-rest-hal-browser</artifactId>**

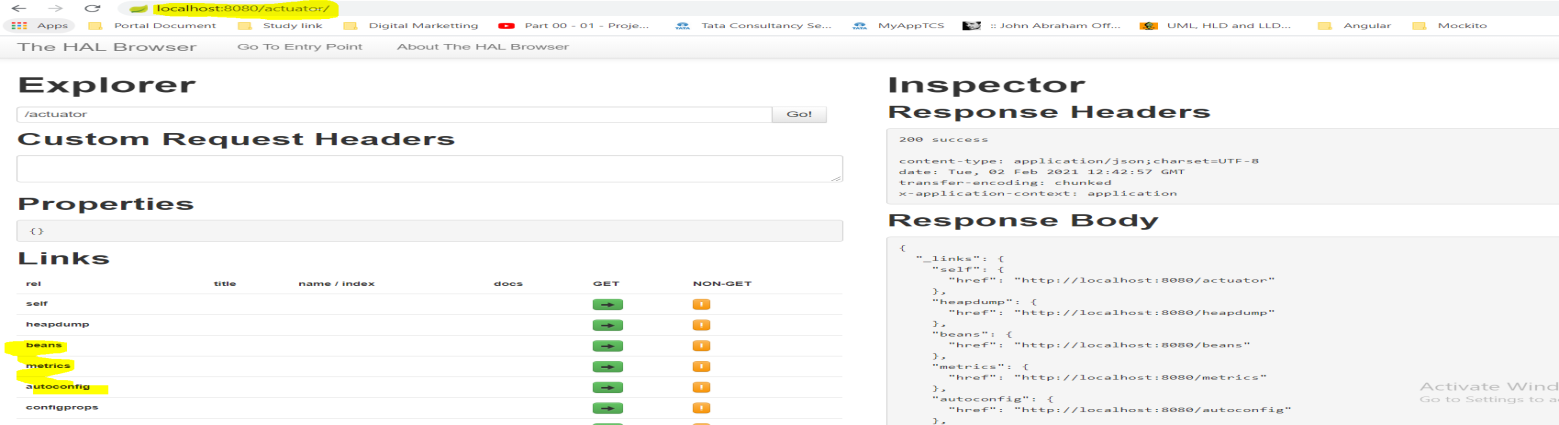
**</dependency>**

**Note: Spring Boot configure automatically once we add this two dependency in the pom.xml. To get more info we can see the log when we start the tomcat server.**

|  |
| --- |
|  |

**After that we can see all the metrics and data using HAL browser. To open the HAL browser just type the URL:**

[**http://localhost:8080/actuator/**](http://localhost:8080/actuator/)

****

**As we can see that the actuator is returning the list of links like**

**self !**

**heapdump:** [**http://localhost:8080/heapdump**](http://localhost:8080/heapdump) **!**

**beans: if click on the beans It will display all the beans configured in the application.**

**metrics :** [**http://localhost:8080/metrics**](http://localhost:8080/metrics) **: show all the request number that we have fired in the browser. For example for GET Or Post request submitted, it displays the number of time it got hit in the browser.**

**autoconfig:** [**http://localhost:8080/autoconfig**](http://localhost:8080/autoconfig)**( Here we can see the positiveMatches shows which are active in the application and NegativeMatches shows which not found and that is why not active in application.**

**configprops:** [**http://localhost:8080/configprops**](http://localhost:8080/configprops)

**env: It will display all the information related to machine like JVM, disc space and others**

**info :** [**http://localhost:8080/info**](http://localhost:8080/info)

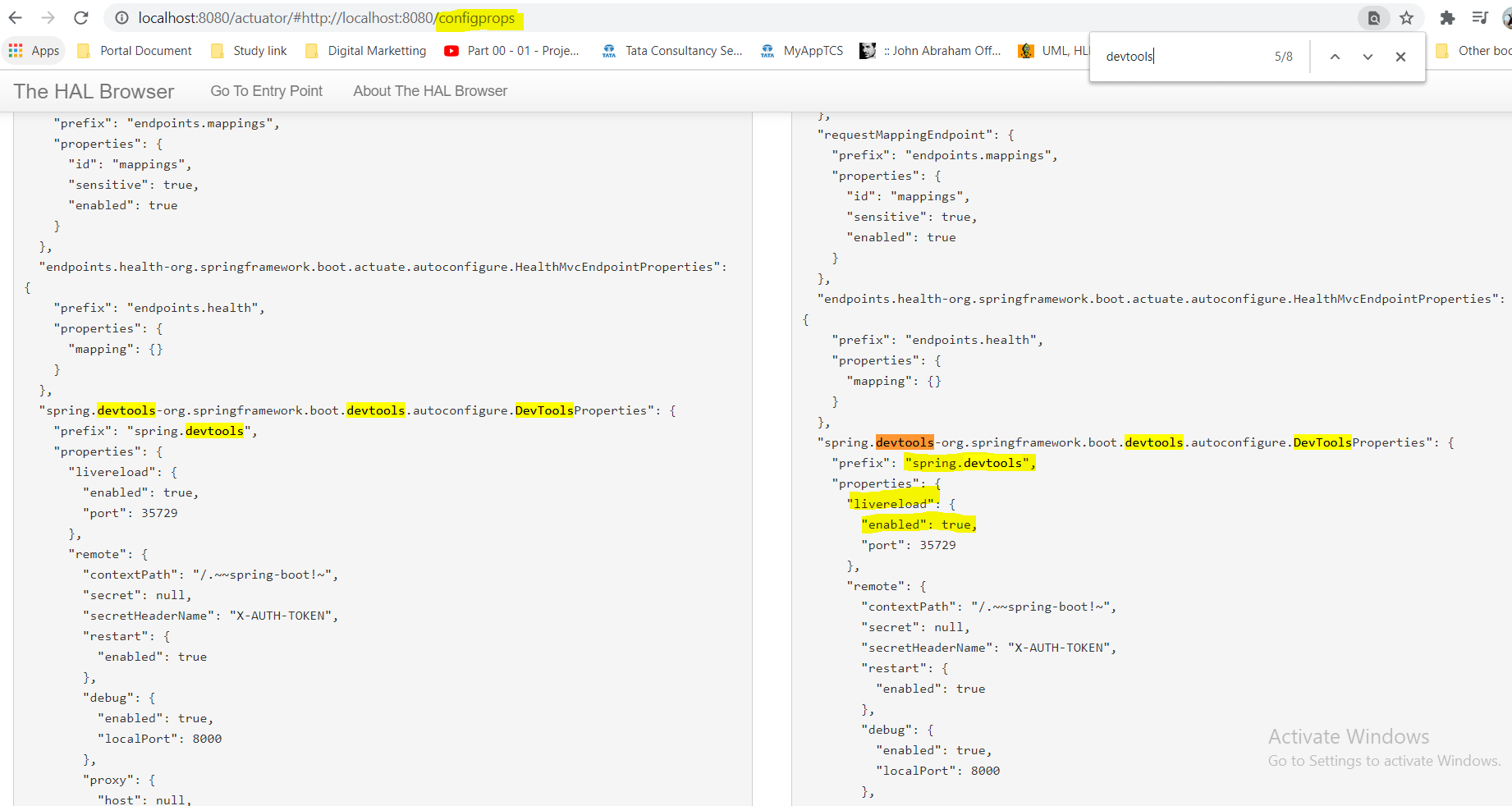
**dump:** [**http://localhost:8080/dump**](http://localhost:8080/dump)

**health**

**mappings: Display all the mapping identifier mentioned in the application.**

**trace:** [**http://localhost:8080/trace**](http://localhost:8080/trace) **: show the request detail that we have fired few moment back.**

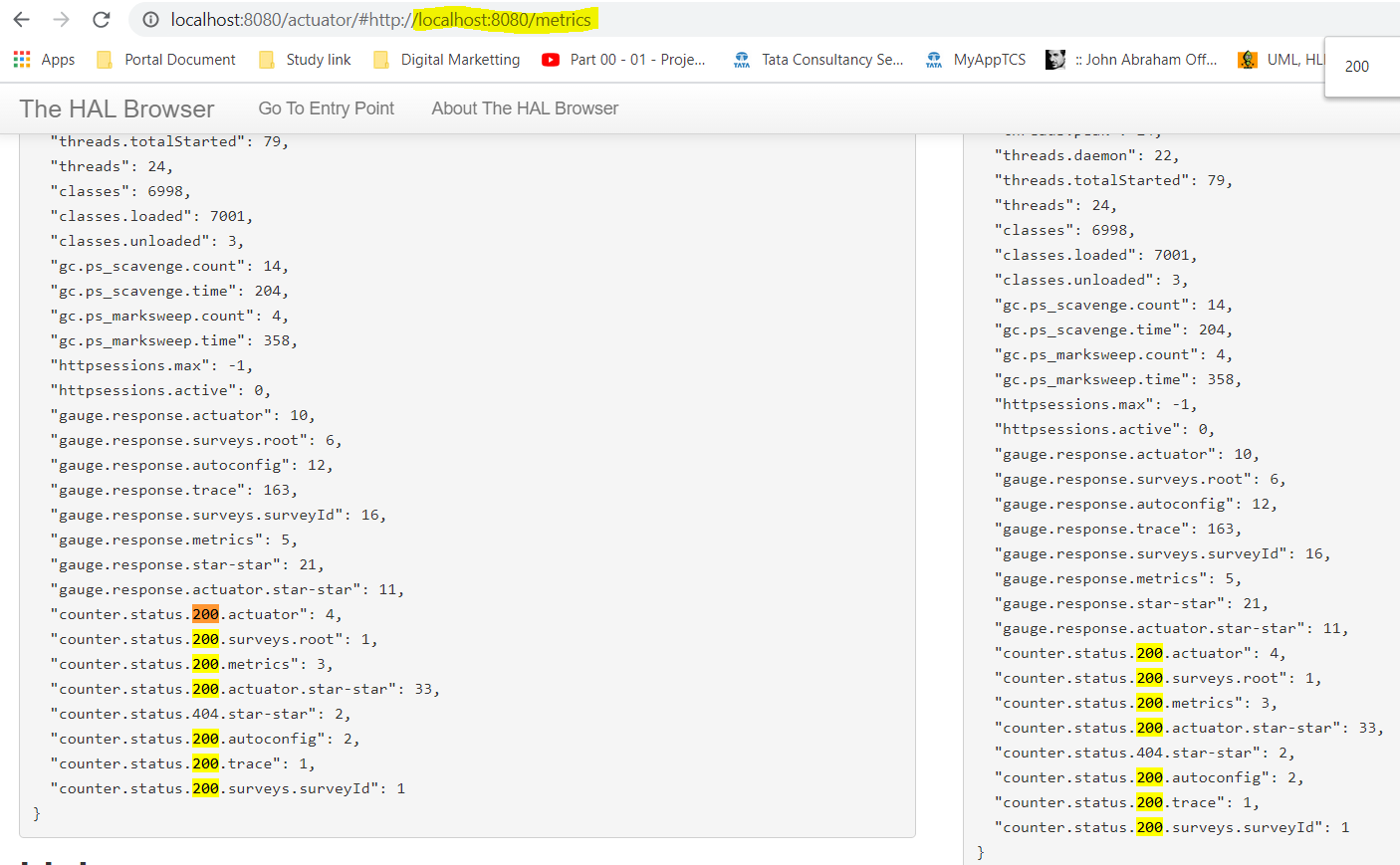
|  |
| --- |
|  |
|  |
| **We can our own url what we used to in Postman like GET req or POST req.** |

****

**Here with this information we can change the property values by mentioning it into application.properties file**

**Like**

**Spring.devtools. livereload=false (if we mention this into application.property file then the live reload will be disabled)**

****

**This actuator / HAL browser is very important when we deploy application in the production environment to monitor the application.**

**Step14.md(Unerstanding Embedded Servlet Container:switch to jetty or undertow)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step14.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step14.md)

**What You Will Learn during this Step:**

* **Embedded servlet containers**
* **Default Tomcat**
* **We did not install Tomcat. Did we? Magic is done by Spring Boot!**
* **Switching to Jetty or Undertow**
* **Configuration**
* **server.port**
* **Programming Tip**
* **Always review code :**[**https://www.youtube.com/watch?v=hVJGu0xdXII**](https://www.youtube.com/watch?v=hVJGu0xdXII)

## Useful Snippets and References

**First Snippet**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-web</artifactId>**

**<exclusions>**

**<exclusion>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-tomcat</artifactId>**

**</exclusion>**

**</exclusions>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

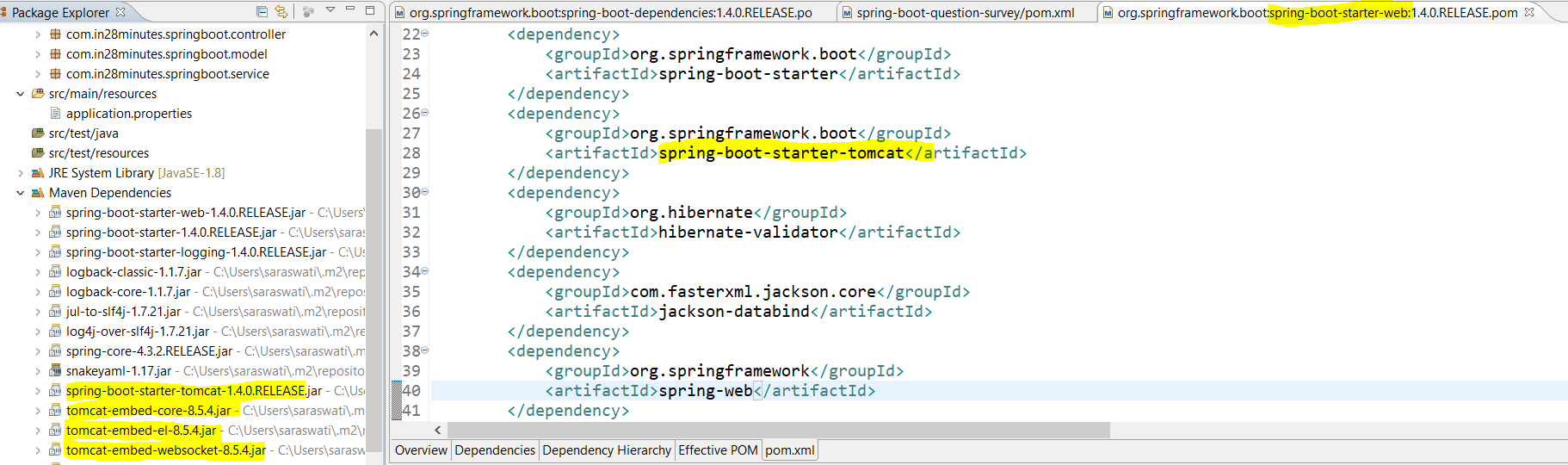
**<artifactId>spring-boot-starter-jetty</artifactId>**

**</dependency>**

**Generally before spring boot we used to create war file and used to deploy into web server. But spring boot now changed this picture, now spring boot comes with embedded Servlet containers. For example tomcat is the default embedded Servlet container. If we look into maven dependencies in the Eclipse we would see the tomcat directly embedded into our spring boot and now it comes as the part of spring boot application dependency. So only things we need to run spring boot application is JDK , because the tomcat configuration now is the part of spring boot application that is why we don’t need to deploy the spring boot application into tomcat server to run the application, spring boot application gets deployed into tomcat server automatically and just ready to run.**

**This tomcat comes default with spring boot, if we see in the pom.xml we will not see tomcat server dependency in the pom.xml. This tomcat server dependency is included in spring-boot-starter-web, that is why it comes as default embedded Servlet container in the spring boot.**

|  |
| --- |
| **Pom.xml>spring-boot-starter-web> then we will see tomcat dependency here**    **<dependency>**  **<groupId>org.apache.tomcat.embed</groupId>**  **<artifactId>tomcat-embed-core</artifactId>**  **<version>${tomcat.version}</version>**  **</dependency>** |

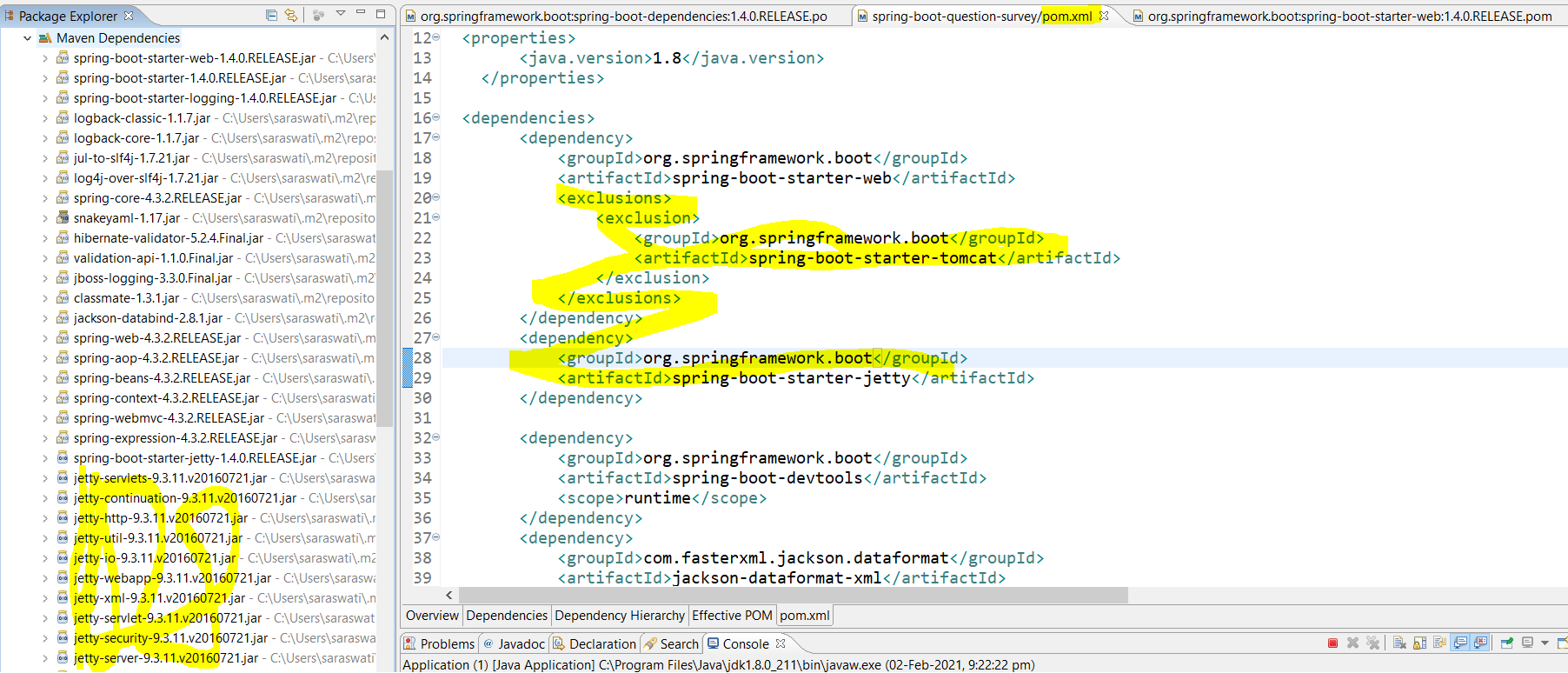
****

**If we want to use other embedded Servlet container, then we need to exclude tomcat server and we need to include other Servlet container (for example let’s include jetty Servlet container).**

|  |
| --- |
| **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **<exclusions>**  **<exclusion>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-tomcat</artifactId>**  **</exclusion>**  **</exclusions>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-jetty</artifactId>**  **</dependency>** |

**So here we have disabled the default tomcat server using <exclusions><exclusion>--</exclusion></exclusions>tag and included the jetty Servlet container. So now instead of tomcat we now will have jetty Servlet container in our spring boot application. Now as soon as we include the above changes in the pom.xml we will see the jetty server dependencies in the maven dependencies.**

**The same we can see in the below screen shot. In this way we can exclude and include any kind of Servlet container in the spring boot application.**

****

|  |
| --- |
| **Pom.xml**  **<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">**  **<modelVersion>4.0.0</modelVersion>**  **<groupId>com.springrest.springboot</groupId>**  **<artifactId>first-rest-springboot-project</artifactId>**  **<version>0.0.1-SNAPSHOT</version>**  **<parent>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-parent</artifactId>**  **<version>1.4.0.RELEASE</version>**  **</parent>**  **<properties>**  **<java.version>1.8</java.version>**  **</properties>**  **<dependencies>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-web</artifactId>**  **<exclusions>**  **<exclusion>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-tomcat</artifactId>**  **</exclusion>**  **</exclusions>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-jetty</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-devtools</artifactId>**  **<scope>runtime</scope>**  **</dependency>**  **<dependency>**  **<groupId>com.fasterxml.jackson.dataformat</groupId>**  **<artifactId>jackson-dataformat-xml</artifactId>**  **</dependency>**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-actuator</artifactId>**  **</dependency>**    **<dependency>**  **<groupId>org.springframework.data</groupId>**  **<artifactId>spring-data-rest-hal-browser</artifactId>**  **</dependency>**  **</dependencies>**  **<build>**  **<plugins>**  **<plugin>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-maven-plugin</artifactId>**  **</plugin>**  **</plugins>**  **</build>**  **</project>** |

**Similarly we can include undertow Servlet container for more information please click on below link.**

[**https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto-embedded-web-servers**](https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto-embedded-web-servers)

****

**For more detail of spring boot application:**

[**https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto-spring-boot-application**](https://docs.spring.io/spring-boot/docs/current/reference/html/howto.html#howto-spring-boot-application)

**Step15.md(Dynamic Configuration in your application)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step15.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step15.md)

**What You Will Learn during this Step:**

* **Using Dynamic Configuration in your application**
* **Customize Welcome Message**
* **Different ways of configuration**
* **--welcome.message="SomethingElse" in Program Arguments**
* **--spring.config.location=classpath:/default.properties**
* **We will learn about profiles in next step**
* **Using Placeholders**
* **YAML**

## Snippets First Snippet

**logging:**

**level:**

**org.springframework: DEBUG**

**app:**

**name: In28Minutes**

**description: ${app.name} is your first Spring Boot application**

**welcome:**

**message: Welcome to your first Spring Boot app!**

**Second Snippet**

**placeholder technique @Value("${welcome.message}")**

|  |  |
| --- | --- |
| **src/main/resources/application.properties**  **logging.level.org.springframework: INFO**  **logging.level.org.springframework.web.servlet: DEBUG**  **app.name=in28Minutes**  **welcome.message=Welcome message from property file! Welcome to ${app.name}** | **src/main/resources/application.yaml**  **logging:**  **level:**  **org.springframework: INFO**  **org.springframework.web.servlet: DEBUG** |

**Here in the application.properties file we have declared two message property app.name which is further being used in welcome.message. And here we have kept key, app.name property inside placeholder ${key} so that the value corresponding to this key app.name will be replaced with its value. This is called placeholder technique.**

**Now let see how to inject this application.properties value inside Controller class.**

|  |  |
| --- | --- |
| **src/main/java/com/in28minutes/springboot/WelcomeService.java**  **package com.in28minutes.springboot;**  **import org.springframework.beans.factory.annotation.Value;**  **import org.springframework.stereotype.Component;**  **//Spring to manage this bean and create an instance of this**  **@Component**  **public class WelcomeService {**  **@Value("${welcome.message}")**  **private String welcomeMessage;**  **public String retrieveWelcomeMessage() {**  **//Complex Method**  **return welcomeMessage;**  **}**  **}** | **src/main/java/com/in28minutes/springboot/WelcomeController.java**  **package com.in28minutes.springboot;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.web.bind.annotation.RequestMapping;**  **import org.springframework.web.bind.annotation.RestController;**  **@RestController**  **public class WelcomeController {**  **//Auto wiring**  **@Autowired**  **private WelcomeService service;**  **@RequestMapping("/welcome")**  **public String welcome() {**  **return service.retrieveWelcomeMessage();**  **}**  **}** |
| **As here we can see that we have used @Value annotation to inject or Autowired property file value into the component class. Now the value of @Value("${welcome.message}") will be inject into private string welcomeMessage which can be used further anywhere according to need** | **Here we are just autowiring the component class and calling the method to get welcome message.** |

**In this way dynamically we can inject Autowired the property value into the controller class.**

**The other way , we can get this welcome message from program argument. To configure this welcome message into program argument, fallow the following steps:**

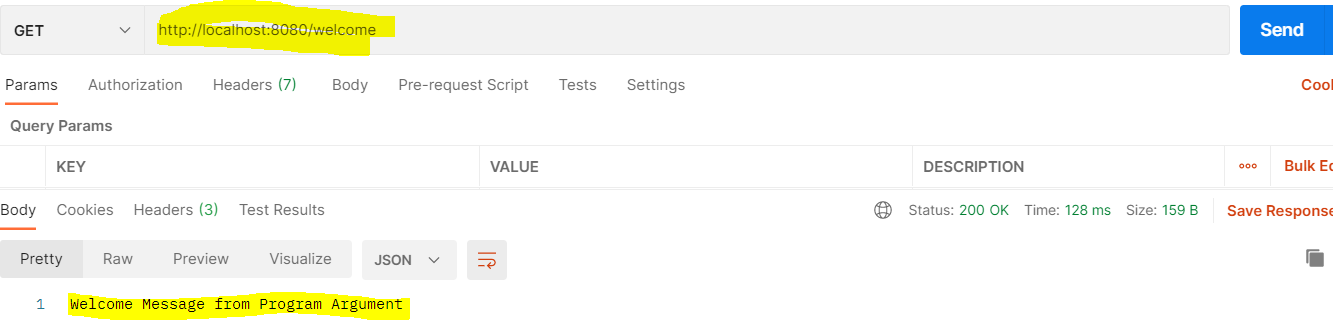
**Right click on the spring boot project**

**Go to run configuration**

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

**Here we have passed the message in the program argument, now this message will be passed to JVM and this property will be having higher priority than application.properties used in the spring boot application. Because this property value will be injected dynamically at run time.**

**Now if run the spring boot application then the welcome message will be displayed from program argument instead of application.properties.**

****

**So now we can say that the command line argument has highest priority that application.properties configuration. And this way we can override the value of application.properties file with program argument. And we can pass different argument in the different environment (dev, test, preprod & prod).**

**And we can say that spring boot provide multiple ways to provide the application properties values to the java class.**

**Now let’s talk about YAML configuration:**

**We already know about application.properties used in the spring boot application and we already know how to declare key and value into application.properties. So far we have seen two types to declare application properties key and value.**

**welcome.message: Getting welcome message from application properties**

**or**

**welcome.message= Getting welcome message from application properties**

**Similar to this we can have YAML property file also in the spring boot application. The only different is that the ext of this file would be .yaml instead of .properties.**

**The advantage of YAML is that it is more readable.**

|  |  |
| --- | --- |
| **src/main/resources/application.properties**  **logging.level.org.springframework: INFo**  **logging.level.org.springframework.web.servlet: DEBUG** | **src/main/resources/application.yaml**  **logging:**  **level:**  **org.springframework: INFO**  **org.springframework.web.servlet: DEBUG** |

**So here in the application.properties file we need to declare same thing again even if name is same, while in case of yaml we just need to mention the main property name. So in this way we can see that YAML is more readable.**

**Step16.md(Spring Profiles for Dev, Test , Prod )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step16.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step16.md)

**What You Will Learn during this Step:**

* **Understand Basics of Profiles**

**Setting a profile: Two ways to setup profiles**

* **Using -Dspring.profiles.active=prod in VM Arguments**
* **spring.profiles.active=prod (in application.properties)**

**Using a profile**

* **application-{profile-name}.properties**
* **@Profile("dev") on a bean**

**Usage**

* **Configure Resources - Databases, Queues, External Services**

**As we have discussed in the previous step for improving application configuration, here in this step we will discuss more about application configuration, the profiles concept in the spring boot application.**

**Profiles are the spring feature and very well supported by spring boot also. Here we will see that how we can setup and use a profile in application according to different environment.**

**The easiest way to set up a profile according to environment just by putting below line in application.properties file.**

**[spring.profiles.active =profileName]**

|  |  |
| --- | --- |
| **src/main/resources/application-dev.properties**  **logging.level.org.springframework: TRACE** | **src/main/resources/application-prod.properties**  **logging.level.org.springframework: INFO** |
| **src/main/resources/application.properties**  **spring.profiles.active=dev**  **logging.level.org.springframework: DEBUG**  **app.name=in28Minutes**  **welcome.message=Welcome message from property file!** | **src/main/resources/application.properties**  **spring.profiles.active=prod**  **logging.level.org.springframework: DEBUG**  **app.name=in28Minutes**  **welcome.message=Welcome message from property file!** |

**As here we can see that we have one application.properties, then we have two application property file for dev (application-dev.properties) and prod (application-prod.properties) environment. And in the application.properties we have mentioned [spring.profiles.active=dev], which means it will activate the profile (application-dev.properties) only and whenever we deploy the application in the server then only[application-dev.properties] profile will be activate and TRACE logging level log will be shown in the logger.**

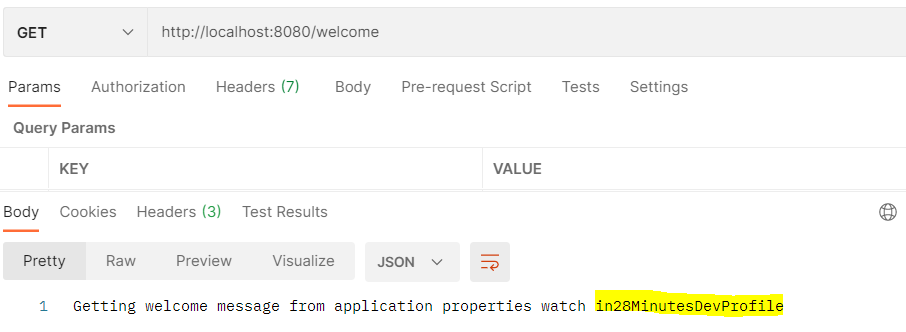
**Another side in the application.properties we have profile [spring.profiles.active=prod] which means it will activate (application-prod.properties) profile only and whenever we will deploy the application in the server then only INFO level log will be active in the logger.**

**And more things that we keep in mind that it will overwrite all the property with the activated profile if any of the application.properties file key matches with activated profile in application.properties.**

**For example**

|  |  |
| --- | --- |
| **application.properties**  **spring.profiles.active= dev**  **logging.level.org.springframework: INFO**  **#logging.level.org.springframework.web.servlet: DEBUG**  **app.name: in28MinutesAPP**  **welcome.message: Getting welcome message from application properties watch ${app.name}** | **application-dev.properties**  **logging.level.org.springframework: TRACE**  **app.name: in28MinutesDev**  **welcome.message: Getting welcome message from application properties watch ${app.name}** |

**Now if we run the application the application.properties file welcome message will be overwritten by dev profile welcome message.**

****

**So here we can see that the welcome message we are getting from dev profile instead of getting application.properties.**

**Another example:**

|  |  |
| --- | --- |
| **application-prod.properties**  **app.name: in28MinutesProdProfile**  **welcome.message: Getting welcome message from application properties watch ${app.name}** | **application.properties**  **spring.profiles.active= prod**  **logging.level.org.springframework: INFO**  **app.name: in28MinutesApp**  **welcome.message: Getting welcome message from application properties watch ${app.name}** |

**Here prod profile is activated in the application.properties so first application.properties will be loaded once we deploy the application and in turn the prod profile will be loaded, hence the welcome message will be override with the prod welcome message.**

****

**This is the one way we can setup spring profiles and use in the application based on the different environment using application.properties file.**

**There is another way of configuring spring profile is by Run configuration Using -Dspring.profiles.active=prod in VM Arguments**

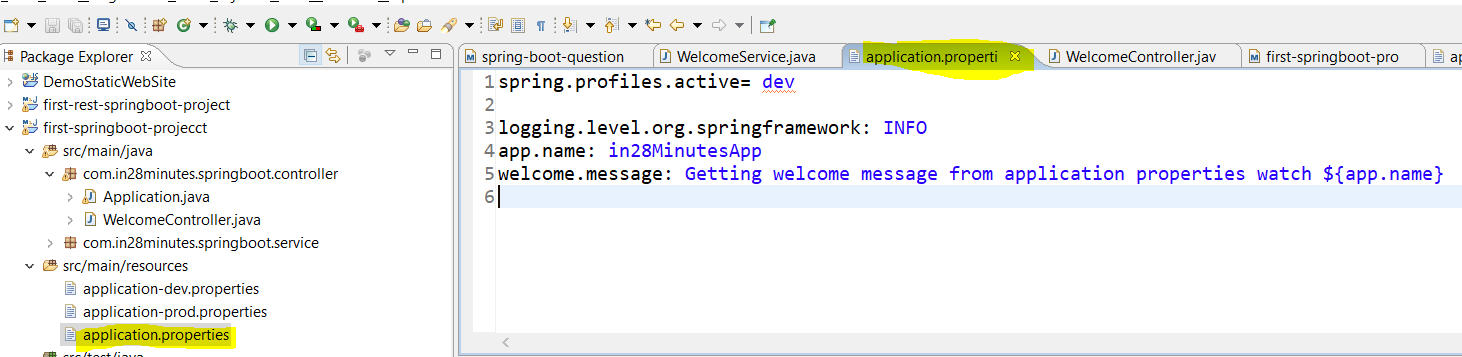
|  |  |
| --- | --- |
|  |  |

**Right click on the application > go to Run Configuration > go to Arguments > in VM arguments paste the below line**

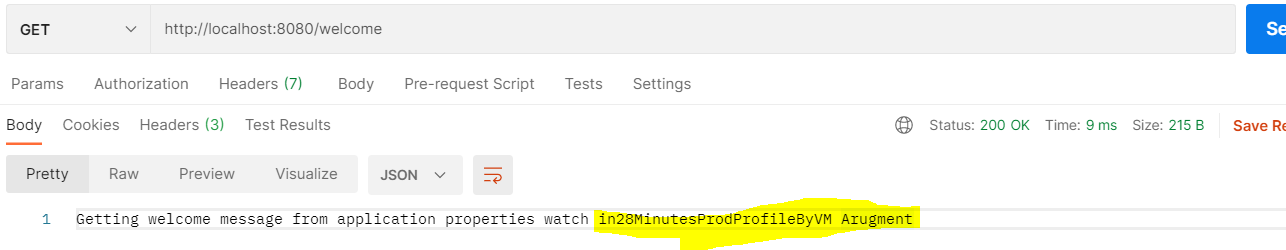
* **-Dspring.profiles.active=prod**

**As we can see that here we have mentioned prod profile in the VM argument so it will load this prod profile during the run time. Now if we run the application then the prod profile will be loaded from VM Argument.**

**Let’s say we have activated the dev profile in application.properties file and prod profile in the VM argument then in this case the highest priority will be for prod profile.**

****

**Now let’s run the application and see which profile will be having highest priority.**

****

**So yes here prod profile got highest priority and hence the profile setup in the VM argument having highest priority.**

**Note: We can create the bean or object of the method based on the active profile.**

|  |  |
| --- | --- |
| **@Profile("prod")**  **@Bean**  **public String profileClass() {**  **return "Hello";**  **}** | **@Profile("dev")**  **@Bean**  **public String profileClass() {**  **return "Hello";**  **}** |

**Here we can see that we have two method annotated with @Bean and @Profile annotation. So the bean creation will be taken care by spring container itself. But one thing we should note here that the bean will be created for the active profile only. As per the above scenario the prod profile is active in VM argument using Run configuration so the bean will be created for the class annotated with @Profile(“prod”) only. This concept we can use in case of unit testing, like if we want to do the unit testing in test environment only then we can use the @Profile(“test”) and bean will be created for the active profile i.e. for test env only and when we deploy the same code in production then bean will not be created in production for those class which are annotated with @Profile(“test”). In this way we can do the unit testing in the particular environment and can help in improving performance by stop creating bean for unnecessary methods.**

|  |  |
| --- | --- |
| @ ConfigurationProperties **@**[**Target**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Target.class%E2%98%83Target)**(**[**value**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Target.class%E2%98%83Target~value)**={**[**METHOD**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(ElementType.class%E2%98%83ElementType%5EMETHOD)**,** [**TYPE**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(ElementType.class%E2%98%83ElementType%5ETYPE)**}) @**[**Retention**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Retention.class%E2%98%83Retention)**(**[**value**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Retention.class%E2%98%83Retention~value)**=**[**RUNTIME**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(RetentionPolicy.class%E2%98%83RetentionPolicy%5ERUNTIME)**) @**[**Documented**](eclipse-javadoc:%E2%98%82=first-springboot-projecct/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang.annotation(Documented.class%E2%98%83Documented)  **Annotation for externalized configuration. Add this to a class definition or a @Bean method in a @Configuration class if you want to bind and validate some external Properties (e.g. from application .properties file).**  **Note that contrary to @Value, SpEL expressions are not evaluated since property values are externalized.** | **@RestController**  **@RequestMapping**  **@PathVariable**  **@RequestBody**  **@RequestParam**  **@Component**  **@Value**  **@Profile** @ConfigurationProperties **@Bean**  **@Autowired**  **@GetMapping, @PostMapping**  **ResponseEntity Utility class**  **ServletUriComponentsBuilder utility class** |

**Step17.md(Advanced Application Configuration: Type Safe configuration of properties )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step17.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step17.md)

**In the last step we have seen that how we have injected/Autowired the application.properties values using @Value annotation in the java file wherever it was needed.Obiviously this is one of the greatest feature provided by the spring to inject/Autowired or configure application properties values using @Value annotation in the java class. But let’s say we want to inject 100 or 1000 of properties value in the java class, and then in that case we need to write 100 or 1000 time @Value to inject or Autowired the value in the java file which is little bit difficult to maintain. One more thing is that it is not type safe also.**

**In this step we will talk about better configuration management than @Value.**

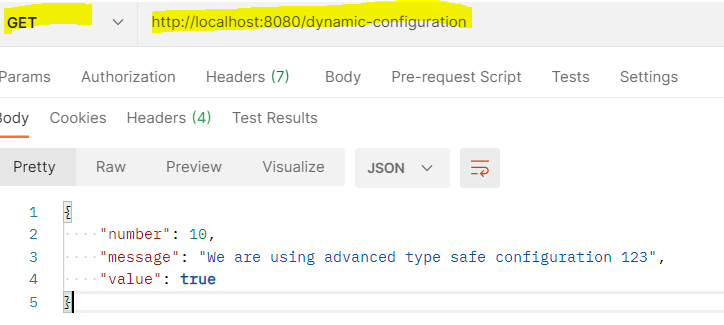
* **Let’s create a @Component class with name BasicConfiguration and add three property kind of things variable in the class.**
* **Add @Component and to make this component class as configuration file let’s add annotation @ConfigurationProperties**
* **Now to make use of this ConfigurationProperties java file will create method in the controller class which returns Map**

|  |  |
| --- | --- |
| **package com.in28minutes.springboot.configuration;**  **import org.springframework.boot.context.properties.ConfigurationProperties;**  **import org.springframework.stereotype.Component;**  **@Component**  **@ConfigurationProperties("basic")**  **public class BasicConfiguration {**  **private boolean value;**  **private String message;**  **private int number;**  **// generate getters and setters** | **@RequestMapping("/dynamic-configuration")**  **public Map dynamicConfiguration() {**    **Map map = new HashMap();**  **map.put("value", config.isValue());**  **map.put("message", config.getMessage());**  **map.put("number", config.getNumber());**  **return map;**  **}** |

**Now we will set the value of (value, message & number) in the application.properties file like as given below**

|  |  |
| --- | --- |
| **application.properties**  **basic.value= true**  **basic.message= We are using advanced type safe configuration 123**  **basic.numner= 10**  **# the above declartion in case of YAML will like below**  **basic:**  **value:true**  **message:We are using advanced type safe configuration 123**  **number:10** | **Note: Here in the property file the name declared in the @ConfigurationProperties and Component class (BasicConfiguration) should be exactly same and the value will be set using as shown in the in the application.properties file.** |

**Now if run the application using** [**http://localhost:8080/dynamic-configuration**](http://localhost:8080/dynamic-configuration)

****

**Note: As in the java class we have defined the data type of each variable and in the property file we have mentioned the values according to the type defined in the java file. In case if we try to give other than the type declared in the java file then we get and an exception and in that case tomcat server itself will not start. So we can say that type of configuration is type-safe also.**

**Let’s say in the number instead of giving any integer value we are trying to pass string value then we will get conversion exception as shown below.**

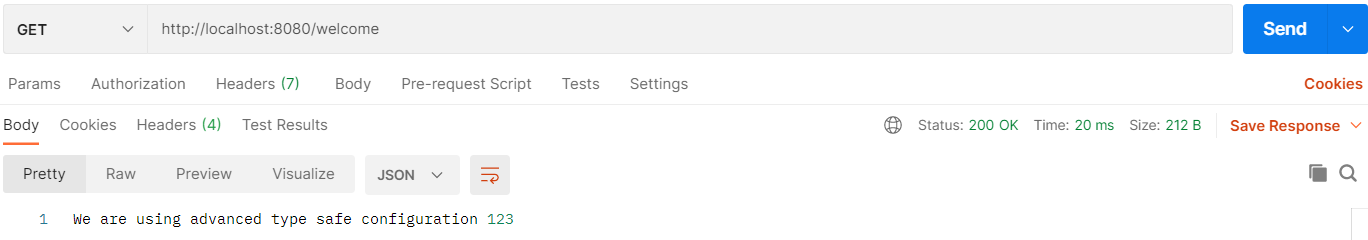
|  |
| --- |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **APPLICATION FAILED TO START**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Description:**  **Binding to target com.in28minutes.springboot.configuration.BasicConfiguration@78d67949 failed:**  **Property: basic.number**  **Value: Hi**  **Reason: Failed to convert property value of type [java.lang.String] to required type [int] for property 'number'; nested exception is org.springframework.core.convert.ConverterNotFoundException: No converter found capable of converting from type [java.lang.String] to type [int]**  **Action:**  **Update your application's configuration** |

**So as we can see that we have tried to configure String (Hi) instead of integer, then it throws number conversion exception with reason and Action and other description as shown above.**

**Note2: If we do not mention any value in the properties file then it will take the default value based on the type declared in the java configuration file. So in this way we can provide the default value with this configuration.**

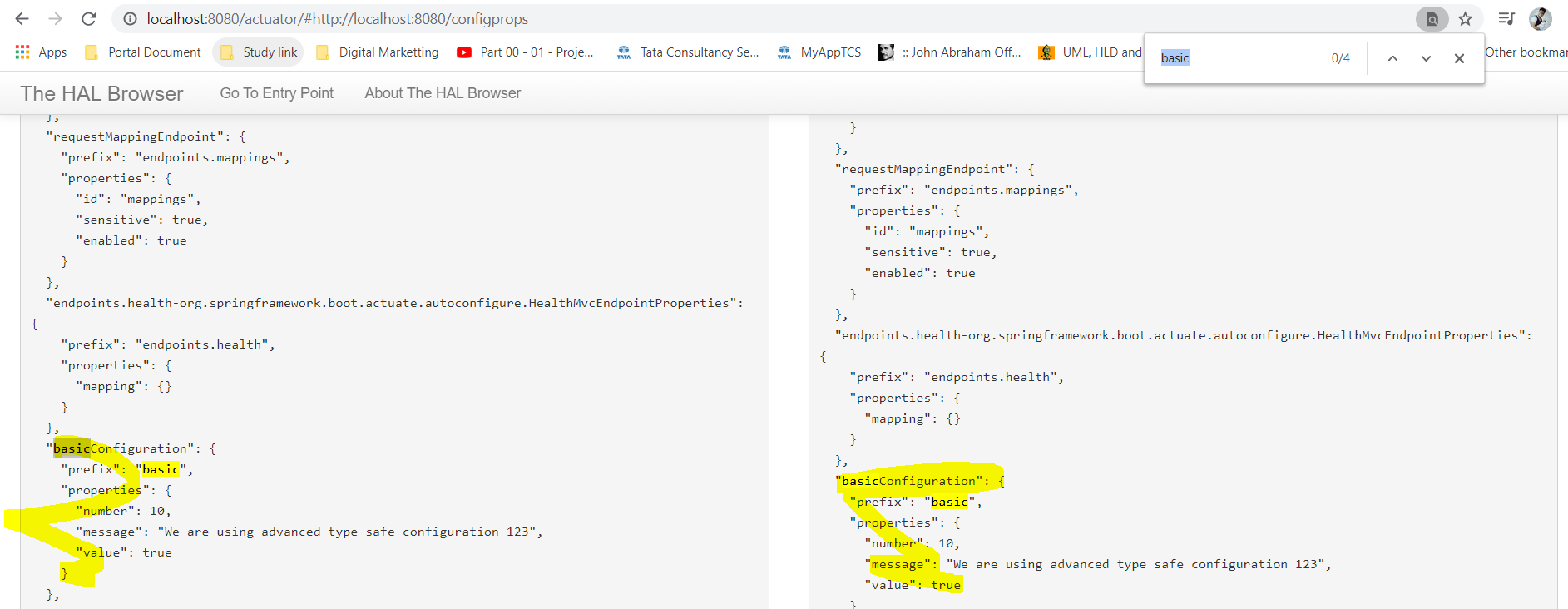
**So we have seen that we have one centralized class where we have declared all property file kind of key and just set the value in the application.properties file using the key and now we don’t need to use @Value to get the value from the property file. To get the value we have to just Autowired the configuration class(BasicConfiguraion) and using getter method we can retrieve the value form the property file**

|  |  |
| --- | --- |
| **package com.in28minutes.springboot.service;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.beans.factory.annotation.Value;**  **import org.springframework.stereotype.Component;**  **import com.in28minutes.springboot.configuration.BasicConfiguration;**  **@Component**  **public class WelcomeService{**  **//@Value("${welcome.message}")**  **//private String welcomeMessage;**    **@Autowired**  **private BasicConfiguration config;**  **public String *retriveWelcomeMessage*() {**  **// return welcomeMessage;**  **return config.getMessage();**  **}**  **}** | **@RequestMapping("/welcome")**  **public String welcome() {**    **return service.*retriveWelcomeMessage*();**  **}**  **@RequestMapping("/dynamic-configuration")**  **public Map dynamicConfiguration() {**  **Map map = new HashMap();**  **map.put("value", config.isValue());**  **map.put("message", config.getMessage());**  **map.put("number", config.getNumber());**  **return map;**  **}** |

****

**Hence we have got the succefully message from the property file just using getter method without using @Value.**

## Now if we open <http://localhost:8080/actuator> then we can see the configuration with the value mentioned in the property file

****

**What You have Learned during this Step:**

* **Even better configuration management than @Value**
* **Type-safe Configuration Properties**
* [**http://localhost:8080/dynamic-configuration**](http://localhost:8080/dynamic-configuration)
* **Also look at**[**http://localhost:8080/actuator/#http://localhost:8080/configprops**](http://localhost:8080/actuator/#http://localhost:8080/configprops)

**Useful Snippets and References**

|  |  |
| --- | --- |
| **First Snippet**  **package com.in28minutes.springboot.configuration;**  **import org.springframework.boot.context.properties.ConfigurationProperties;**  **import org.springframework.stereotype.Component;**  **@Component**  **@ConfigurationProperties("basic")**  **public class BasicConfiguration {**  **private boolean value;**  **private String message;**  **private int number;**  **public boolean isValue() {**  **return value;**  **}**  **public void setValue(boolean value) {**  **this.value = value;**  **}**  **public String getMessage() {**  **return message;**  **}**  **public void setMessage(String message) {**  **this.message = message;**  **}**  **public int getNumber() {**  **return number;**  **}**  **public void setNumber(int number) {**  **this.number = number;**  **}**  **}** | **Second Snippet**  **@Autowired**  **private BasicConfiguration configuration;**  **@RequestMapping("/dynamic-configuration")**  **public Map dynamicConfiguration() {**  **// Not the best practice to use a map to store differnt types!**  **Map map = new HashMap();**  **map.put("message", configuration.getMessage());**  **map.put("number", configuration.getNumber());**  **map.put("key", configuration.isValue());**  **return map;**  **}**  **Third Snippet**  **basic.value: true**  **basic.message: Dynamic Message**  **basic.number: 100**  **Fourth Snippet**  **basic:**  **value: true**  **message: Dynamic Message YAML**  **number: 100** |

**Exercises - Understand Type Safety**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**APPLICATION FAILED TO START**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Description: Binding to target com.in28minutes.springboot.configuration.BasicConfiguration@391b8545 failed:**

**Property: basic.number**

**Value: ABC**

**Reason: Failed to convert property value of type [java.lang.String] to required type [int] for property 'number'; nested exception is org.springframework.core.convert.ConverterNotFoundException: No converter found capable of converting from type [java.lang.String] to type [int]**

**Action: Update your application's configuration**

**Step18.md(Spring boot starter: Spring Data JPAwith CommandLineRunner )** [**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step18.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step17.md)

**In the last three steps we have talked much more about configurations. Now let’s talk about data side of the spring. In this step we will talk about spring data JPA. JPA is java persistence API; it helps us to map an object to a database. Spring data JPA provides one more level of abstraction on top of JPA and it makes even easier to store things in the database.**

**As we know spring boot mostly provides starter for the things to include into spring boot application and spring boot does the configuration automatically for the most of starter. Similarly spring boot provides the starter for spring boot JPA which, when we configure, it makes the things very easy to save in database.**

* + 1. **We will create a simple JPA entity(table) called User with three details.**
    2. **Add the spring-boot-starter-jpa and in-memory database (h2). Embedded databases are typically used in unit tests.**

|  |  |
| --- | --- |
| **Add dependency in POM.xml**  **First Snippet - Add H2 Later after showing the erro**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-data-jpa</artifactId>**  **</dependency>**  **Once we include the above dependency in the POM.xml and try to start server then we will get an error in server start, because as we know once we include spring-boot-data-jpa in the POM.xml and start server then spting boot start auto configuration and try to configure database, but it does not find any kind of the database in the application that is why it does not start server.**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **APPLICATION FAILED TO START**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Description:**  **Cannot determine embedded database driver class for database type NONE**  **Action:**  **If you want an embedded database please put a supported one on the classpath. If you have database settings to be loaded from a particular profile you may need to active it (the profiles "prod" are currently active).**  **So let’s add embedded database (in-memory database) by using below dependency**  **<dependency>**  **<groupId>com.h2database</groupId>**  **<artifactId>h2</artifactId>**  **</dependency>**  **Now the spring do the autoconfigurtiaon of JPA and points to H2 datasource.** | **Second Snippet**  **package com.in28minutes.springboot.jpa;**  **import javax.persistence.Entity;**  **import javax.persistence.GeneratedValue;**  **import javax.persistence.GenerationType;**  **import javax.persistence.Id;**  **@Entity**  **public class User {**  **@Id**  **@GeneratedValue(strategy = GenerationType.AUTO)**  **private Long id;**  **private String name;// Not perfect!! Should be a proper object!**  **private String role;// Not perfect!! An enum should be a better choice!**  **protected User() {**  **}**  **public User(String name, String role) {**  **super();**  **this.name = name;**  **this.role = role;**  **}**  **public Long getId() {**  **return id;**  **}**  **public String getName() {**  **return name;**  **}**  **public String getRole() {**  **return role;**  **}**  **@Override**  **public String toString() {**  **return String.format("User [id=%s, name=%s, role=%s]", id, name, role);**  **}**  **}** |

* + 1. **We will create a class UserCommandLineRunner which implements CommandLineRunner interface, this interface contains method run(String..args), which we will override here in UserCommandLineRunner Class to save the things into database.**
    2. **Create Logger from Slf4**
    3. **So we will create some user and few methods to find-out the users.**
    4. **Finally creates an interface with name UserRepository which extends CurdRepository which is Interface for generic CRUD operations on a repository for a specific type.**
    5. **For more info:** [**https://www.baeldung.com/spring-data-repositories**](https://www.baeldung.com/spring-data-repositories)
    6. **or** [**https://www.tutorialspoint.com/difference-between-crudrepository-and-jparepository-in-java**](https://www.tutorialspoint.com/difference-between-crudrepository-and-jparepository-in-java)
    7. **Difference between CommandLineRunner and ApplicationRunner.**

[**https://www.tutorialspoint.com/spring\_boot/spring\_boot\_runners.htm**](https://www.tutorialspoint.com/spring_boot/spring_boot_runners.htm)

|  |  |
| --- | --- |
| **Third Snippet**  **package com.in28minutes.springboot.jpa;**  **import org.slf4j.Logger;**  **import org.slf4j.LoggerFactory;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.boot.CommandLineRunner;**  **import org.springframework.stereotype.Component;**  **@Component**  **public class UserCommandLineRunner implements CommandLineRunner {**  **private static final Logger log = LoggerFactory**  **.getLogger(UserCommandLineRunner.class);**  **@Autowired**  **private UserRepository repository;**  **@Override**  **public void run(String... args) {**  **// save a couple of customers**  **repository.save(new User("Ranga", "Admin"));**  **repository.save(new User("Ravi", "User"));**  **repository.save(new User("Satish", "Admin"));**  **repository.save(new User("Raghu", "User"));**  **log.info("-------------------------------");**  **log.info("Finding all users");**  **log.info("-------------------------------");**  **for (User user : repository.findAll()) {**  **log.info(user.toString());**  **}**  **}**  **}** Command Line Runner **Command Line Runner is an interface. It is used to execute the code after the Spring Boot application started.**  **This interface provides access to application arguments as string array.**  **Spring boot’s**[**CommandLineRunner**](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/CommandLineRunner.html)**interface is used to run a code block only once in application’s lifetime – after application is initialized.**  **So if we want to initialize something during the application startup then this command line runner is very useful.** | **Fourth Snippet**  **package com.in28minutes.springboot.jpa;**  **import java.util.List;**  **import org.springframework.data.repository.CrudRepository;**  **public interface UserRepository extends CrudRepository<User, Long> {**  **}**  **public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {**  **<S extends T> S save(S entity);**  **T findOne(ID primaryKey);**  **Iterable<T> findAll();**  **Long count();**  **void delete(T entity);**  **boolean exists(ID primaryKey);**  **}**  **Notice the typical CRUD functionality:**   * ***save(…) – s*ave an *Iterable* of entities. Here, we can pass multiple objects to save them in a batch** * ***findOne(…)* – get a single entity based on passed primary key value** * ***findAll()* – get an *Iterable* of all available entities in database** * ***count() – r*eturn the count of total entities in a table** * ***delete(…)* – delete an entity based on the passed object** * **exists(…) – verify if an entity exists based on the passed primary key value**   **This interface looks quite generic and simple, but actually, it provides all basic query abstractions needed in an application.** |
|  |  |

**COURSE UPDATE : H2 Database URL**

**With the latest versions of Spring Boot (2.3+), the H2 database name is randomly generated each time you restart the server.**

**You can find the database name and URL from the console log.**

**RECOMMENDED:**

**Make the database URL a constant by configuring this in application.properties.**

1. **spring.datasource.url=jdbc:h2:mem:testdb**
2. **spring.data.jpa.repositories.bootstrap-mode=default**

**DEBUGGING GUIDE (If you have problems)**

**JPA Hibernate Debugging Guide:**[**https://github.com/in28minutes/in28minutes-initiatives/blob/master/The-in28Minutes-TroubleshootingGuide-And-FAQ/jpa-and-hibernate.md**](https://github.com/in28minutes/in28minutes-initiatives/blob/master/The-in28Minutes-TroubleshootingGuide-And-FAQ/jpa-and-hibernate.md)

**Why do we need to configure bootstrap-mode? Details here -**[**https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.3-Release-Notes#bootstrapmode-for-jpa-repositories**](https://github.com/spring-projects/spring-boot/wiki/Spring-Boot-2.3-Release-Notes#bootstrapmode-for-jpa-repositories)

**Step19.md(In-Memory Database H2 Console and add new JSP Repository method )** [**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step19.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step19.md)

## What You Will Learn during this Step:

* **Look at H2 Console :**[**http://localhost:8080/h2-console**](http://localhost:8080/h2-console)
* **Use db url jdbc:h2:mem:testdb**
* **Add findByRole method**

## Some Notes

* **Useful Properties**
* **spring.datasource.driver-class-name=com.mysql.jdbc.Driver**
* **spring.datasource.url=jdbc:mysql://localhost:3306/test**
* **spring.datasource.username=root**
* **spring.datasource.password=admin**
* **spring.datasource.initialize=true**
* **spring.jpa.hibernate.ddl-auto=update**
* **spring.jpa.show-sql=true**

## Useful Snippets and References

**First Snippet**

**log.info("-------------------------------");**

**log.info("Finding user with id 1");**

**log.info("-------------------------------");**

**User user = repository.findOne(1L);**

**log.info(user.toString());**

**log.info("-------------------------------");**

**log.info("Finding all Admins");**

**log.info("-------------------------------");**

**for (User admin : repository.findByRole("Admin")) {**

**log.info(admin.toString());**

**// Do something...**

**}**

**Second Snippet**

**package com.in28minutes.springboot.jpa;**

**import java.util.List;**

**import org.springframework.data.repository.CrudRepository;**

**public interface UserRepository extends CrudRepository<User, Long> {**

**List<User> findByRole(String description);**

**}**

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

**Step20.md(Spring boot starter: Introduction to Spring Data Rest: )** [**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step20.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step20.md)

**Step21.md(Spring boot Integration Test: )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step21.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step21.md)

**What is JUnit and Unit Testing?**

**As we know that the test is done to check the functionality as expected. As we know that generally the application has 1000 or more that line of code, and if we waiting them to deploy and then, in that case we would have to check big-2 chunks of functionality and that is called screen testing or system testing.**

**Unit testing is different from that. Unit testing is the testing of method or class.**

**Let’s say we have one class name class1 and this class has several methods like (method1, method2, method3…) so the unit testing can be performed at method level or some time at the group of method or we can write the unit test for specific class also. But most of the time we write the unit testing for specific method.**

**JUNIT- is framework which helps us to call the specific method and check whether the output is as expected or not.**

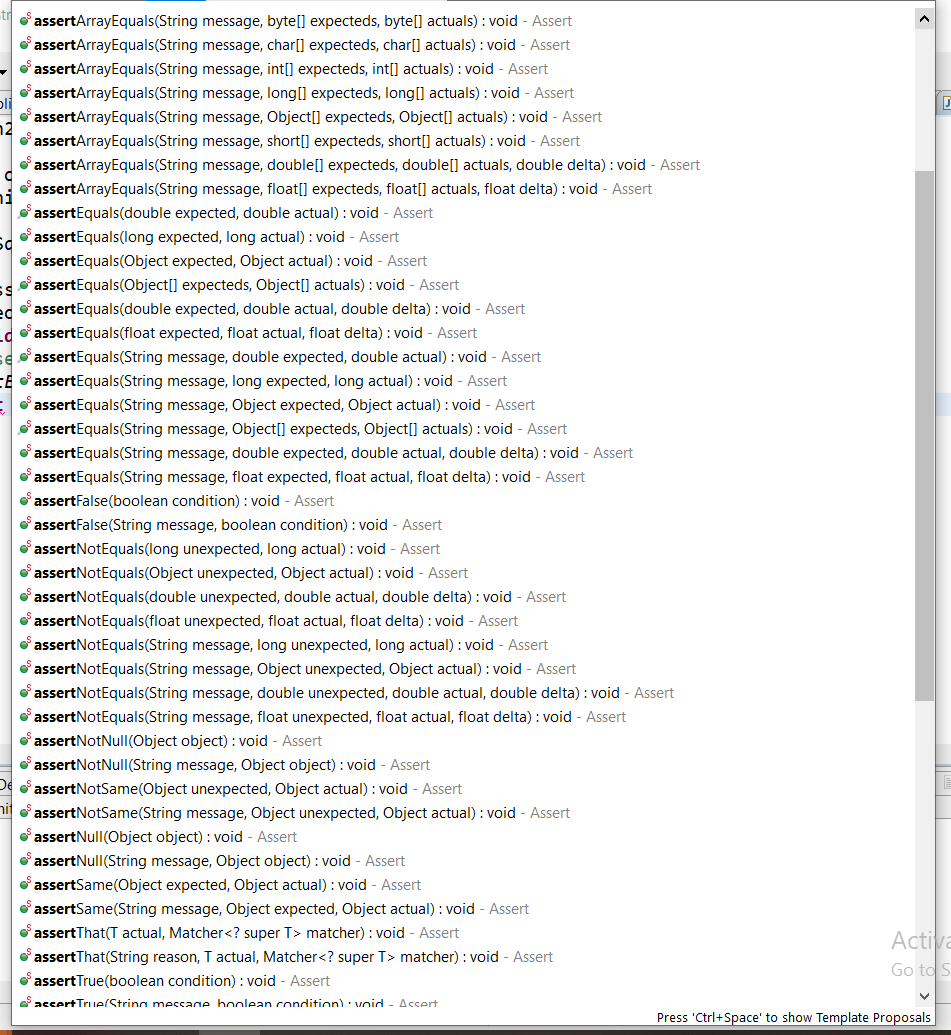
**Let’s we have one method [public int square(int x)] and we call this method in JUNIT test case and pass the number and will check whether the result is as expected or not e.g. square(3)==9 then test succeeds and if square(3)==8 then it fails. Junit allows us to call this method to test. So Junit test is nothing but testing a specific method and check where the method is giving result as expected or not.**

**One of the most advantages of using unit testing is that these are automated test, so we can keep running these test cases automatically any number of times or whenever there is a change in code and whenever test gets failed I would get notified either by email or by any other way. So unit testing falls under automated testing.**

|  |  |
| --- | --- |
| **package com.in28minutes.springboot.controller;**  **public class SquareClass {**  **public int square(int x) {**  **return x \* x;**  **}**  **public int sum(int[] numbers) {**  **int sum = 0;**  **for (int i : numbers) {**  **sum += i;**  **}**  **return sum;**  **}**  **}** | **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.\*;**  **import org.junit.Test;**  **public class SquareTest {**  **SquareClass c = new SquareClass();**  **@Test**  **public void testSquare() {**  ***assertEquals(144, c.square(12));***  ***assertEquals*(6, c.sum(new int[] { 2, 2, 2 }));**  **}**  **}** |

**For more detail:** [**https://www.javatpoint.com/junit-tutorial**](https://www.javatpoint.com/junit-tutorial)

|  |
| --- |
| **The Junit 4.x framework is annotation based, so let's see the annotations that can be used while writing the test cases.**  **@Test annotation specifies that method is the test method.**  **@Test(timeout=1000) annotation specifies that method will be failed if it takes longer than 1000 milliseconds (1 second).**  **@BeforeClass annotation specifies that method will be invoked only once, before starting all the tests.**  **@Before annotation specifies that method will be invoked before each test.**  **@After annotation specifies that method will be invoked after each test.**  **@AfterClass annotation specifies that method will be invoked only once, after finishing all the tests.** |
| **Methods of Assert class : The common methods of Assert class are as follows:**  **void assertEquals(boolean expected,boolean actual): checks that two primitives/objects are equal. It is overloaded.**  **void assertTrue(boolean condition): checks that a condition is true.**  **void assertFalse(boolean condition): checks that a condition is false.**  **void assertNull(Object obj): checks that object is null.**  **void assertNotNull(Object obj): checks that object is not null.** |

****

|  |  |
| --- | --- |
| **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.\*;**  **import org.junit.Before;**  **import org.junit.Test;**  **public class SquareTest {**  **SquareClass c = new SquareClass();**  **@Before**  **public void before() {**  **System.*out*.println("before");**  **}**  **@Test(timeout = 1000)**  **public void testSquare() {**  **System.*out*.println("Test1");**  ***assertEquals*(144, c.square(12));**  **}**  **@Test**  **public void testSum() {**  **System.*out*.println("Test2");**  ***assertEquals*(6, c.sum(new int[] { 2, 2, 2 }));**  **}**  **}**  **Output :**  **before**  **Test2**  **before**  **Test1**  **As we can see that the method annotated with @Before is running before every @Test annotated method. So if we want any method to be run before every test method then we can use @Before** | **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.\*;**  **import org.junit.After;**  **import org.junit.Before;**  **import org.junit.Test;**  **public class SquareTest {**  **SquareClass c = new SquareClass();**  **@After**  **public void after() {**  **System.*out*.println("after");**  **}**  **@Test(timeout = 1000)**  **public void testSquare() {**  **System.*out*.println("Test1");**  ***assertEquals*(144, c.square(12));**  **}**  **@Test**  **public void testSum() {**  **System.*out*.println("Test2");**  ***assertEquals*(6, c.sum(new int[] { 2, 2, 2 }));**  **}**  **}**  **Output:**  **Test2**  **after**  **Test1**  **after**  **As we can see that the method annotated with @After is running after every @Test annotated method. So if we want any method to be run after every test method then we can use @After** |
| **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.\*;**  **import org.junit.After;**  **import org.junit.Before;**  **import org.junit.Test;**  **public class SquareTest {**  **SquareClass c = new SquareClass();**  **@Before**  **public void before() {**  **System.*out*.println("before");**  **}**  **@After**  **public void after() {**  **System.*out*.println("after");**  **}**  **@Test(timeout = 1000)**  **public void testSquare() {**  **System.*out*.println("Test1");**  ***assertEquals*(144, c.square(12));**  **}**  **@Test**  **public void testSum() {**  **System.*out*.println("Test2");**  ***assertEquals*(6, c.sum(new int[] { 2, 2, 2 }));**  **}**  **}** | **Output:**  **before**  **Test2**  **after**  **before**  **Test1**  **after**  **So we can see that @Before method is running before every @Test method and @After method runs after every @Test method.**  **So in this way we can use @Before and @After in the Junit test case** |

**Integration Testing:**

**Here we will do the integration testing for the URL [http://localhost:8080/surveys/Survey1/questions/Question1]. So this is the URL that wants to test in this integration testing. The aim of the IT is to go across all the layers. To start integration testing we have to add few code snippet in the spring application.**

|  |
| --- |
| **First Snippet**  **<dependency>**  **<groupId>org.springframework.boot</groupId>**  **<artifactId>spring-boot-starter-test</artifactId>**  **<scope>test</scope>**  **</dependency>**  **This dependency helps us to write unit test. Here we have mentioned scope test, because we don’t want unit testing to be available in production code. So when we build the jar then this test jar would not be available in production code. Because we don’t want the test code in production.** |

|  |  |
| --- | --- |
| **UnRefactored Snippet**  **package com.in28minutes.springboot.controller;**  **import java.util.Arrays;**  **import org.junit.Test;**  **import org.junit.runner.RunWith;**  **import org.skyscreamer.jsonassert.JSONAssert;**  **import org.springframework.boot.context.embedded.LocalServerPort;**  **import org.springframework.boot.test.context.SpringBootTest;**  **import org.springframework.boot.test.web.client.TestRestTemplate;**  **import org.springframework.http.HttpEntity;**  **import org.springframework.http.HttpHeaders;**  **import org.springframework.http.HttpMethod;**  **import org.springframework.http.MediaType;**  **import org.springframework.http.ResponseEntity;**  **import org.springframework.test.context.junit4.SpringRunner;**  **import com.in28minutes.springboot.Application;**  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes = Application.class,**  **webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)**  **public class SurveyControllerIT {**  **@LocalServerPort**  **private int port;**  **@Test**  **public void testRetrieveSurveyQuestion() {**  **String url = "http://localhost:" + port + "/surveys/Survey1/questions/Question1";**  **TestRestTemplate restTemplate = new TestRestTemplate();**  **HttpHeaders headers = new HttpHeaders();**  **headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));**  **HttpEntity<String> entity = new HttpEntity<String>(null, headers);**  **ResponseEntity<String> response = restTemplate.exchange(url ,HttpMethod.GET, entity, String.class);**    **String expected = "{id:Question1,description:Largest Country in the World,correctAnswer:Russia}";**  **JSONAssert.assertEquals(expected, response.getBody(), false);**  **}**  **}** | **Refactored Snippet**  **package com.in28minutes.springboot.controller;**  **import java.util.Arrays;**  **import org.junit.Before;**  **import org.junit.Test;**  **import org.junit.runner.RunWith;**  **import org.skyscreamer.jsonassert.JSONAssert;**  **import org.springframework.boot.context.embedded.LocalServerPort;**  **import org.springframework.boot.test.context.SpringBootTest;**  **import org.springframework.boot.test.web.client.TestRestTemplate;**  **import org.springframework.http.HttpEntity;**  **import org.springframework.http.HttpHeaders;**  **import org.springframework.http.HttpMethod;**  **import org.springframework.http.MediaType;**  **import org.springframework.http.ResponseEntity;**  **import org.springframework.test.context.junit4.SpringRunner;**  **import com.in28minutes.springboot.Application;**  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes = Application.class, webEnvironment =**  **SpringBootTest.WebEnvironment.RANDOM\_PORT)**  **public class SurveyControllerIT {**  **@LocalServerPort**  **private int port;**  **private TestRestTemplate template = new TestRestTemplate();**  **HttpHeaders headers = new HttpHeaders();**  **@Before**  **public void setupJSONAcceptType() {**  **headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));**  **}**  **@Test**  **public void retrieveSurveyQuestion() throws Exception {**  **String expected = "{id:Question1,description:Largest Country in the World, correctAnswer : Russia, options:[India ,Russia, United States,China]}";**  **ResponseEntity<String> response = template.exchange( createUrl("/surveys/Survey1/questions/Question1"),**  **HttpMethod.GET, new HttpEntity<String>("DUMMY\_DOESNT\_MATTER",**  **headers), String.class);**  **JSONAssert.assertEquals(expected, response.getBody(), false);**  **}**  **private String createUrl(String uri) {**  **return "http://localhost:" + port + uri;**  **}**  **}** |

**So basically we are doing the following things in the integration testing.**

**Part 1- Initialize and launch spring boot application for integration testing using below annotation at class level**

1. **@RunWith(SpringRunner.class)**
2. **@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)**
3. **@LocalServerPort**

**private int port;**

**Part 2- Invoke the URL /surveys/Survey1/questions/Question1**

1. **Create the URL to be invoked using Http GET Or POST method (http://domainName:port/requestMappingURL Name)**

**e.g. String url = "http://localhost:" + port + "/surveys/Survey1/questions";**

1. **Create an object TestRestTemplate or inject using @Autowired annotation**
2. **Create HttpHeaders object and set Accept the media type (Application\_JSON or xml) according to requirement**
3. **Now invoke exchange( CreatedURL, HttpEntity\_reqWith\_Body\_n\_Header,HttpGetOrPOST, ParameterizedTypeReference ) method using TestRestTemplate object and pass the required parameter . This method will return ResponseEntity response.**
4. **Now create a sample response and test with *assertTrue*(response.getBody().contains(sampleQuestion));**

**If the ResponseEntity response contains the matching sentence with sampleQuestion then Unit test will be passed otherwise will failed.**

|  |
| --- |
| **package com.in28minutes.springboot.controller;**  **import java.util.Arrays;**  **import org.junit.Test;**  **import org.junit.runner.RunWith;**  **import org.skyscreamer.jsonassert.JSONAssert;**  **import org.springframework.boot.context.embedded.LocalServerPort;**  **import org.springframework.boot.test.context.SpringBootTest;**  **import org.springframework.boot.test.web.client.TestRestTemplate;**  **import org.springframework.http.HttpEntity;**  **import org.springframework.http.HttpHeaders;**  **import org.springframework.http.HttpMethod;**  **import org.springframework.http.MediaType;**  **import org.springframework.http.ResponseEntity;**  **import org.springframework.test.context.junit4.SpringRunner;**  **import com.in28minutes.springboot.Application;**  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes = Application.class,**  **webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*)**  **public class SurveyControllerIT\_1 {**  **@LocalServerPort**  **private int port;**  **@Test**  **public void testRetrieveSurveyQuestion() {**  **String url = "http://localhost:" + port + "/surveys/Survey1/questions/Question1";**  **TestRestTemplate restTemplate = new TestRestTemplate();**  **HttpHeaders headers = new HttpHeaders();**  **headers.setAccept(Arrays.*asList*(MediaType.*APPLICATION\_JSON*));**  **HttpEntity<String> entity = new HttpEntity<String>(null, headers);**  **ResponseEntity<String> response = restTemplate.exchange(url, HttpMethod.*GET*, entity, String.class);**  **String expected = "{id:Question1,description:Largest Country in the World,correctAnswer:Russia}";**  **JSONAssert.*assertEquals*(expected, response.getBody(), false);**  **}**  **}** |

**@RunWith(SpringRunner.class) :\_ Provides a bridge between Spring Boot test features and JUnit. Whenever we are using any Spring Boot testing features in our JUnit tests, this annotation will be required. It creates Spring TestContext Framework.**

**SpringRunner.class : SpringRunner is an alias for the SpringJUnit4ClassRunner, which joins JUnit testing library with the Spring TestContext Framework. We use it with @RunWith (SpringRunner.class).With SpringRunner; we can implement standard JUnit 4-based unit and integration tests.**

**@SpringBootTest: The @SpringBootTest annotation tells Spring Boot to look for a main configuration class (one with @SpringBootApplication, for instance) and use that to start a spring application context.**

**Note the use of webEnvironment=RANDOM\_PORT to start the server with a random port (useful to avoid conflicts in test environments as in case of continuous testing several test used to run so need different port to avoid confilct) and the injection of this random port is done with @LocalServerPort. Also, note that Spring Boot has automatically provided a TestRestTemplate for you. All you have to do is add @Autowired to it.**

**TestRestTemplate: This is the alternative of**[**Rest Template**](https://docs.spring.io/spring-framework/docs/5.3.3/javadoc-api/org/springframework/web/client/RestTemplate.html?is-external=true)**that is suitable for integration tests. Spring TestRestTemplate can be used to consume HTTP POST REST API, this can be used to invoke the Rest service URL. This is the utility class from spring boot. Make sure, you have spring-boot-starter-test dependency in your project to get access to**[**TestRestTemplate**](https://docs.spring.io/spring-boot/docs/current/api/org/springframework/boot/test/web/client/TestRestTemplate.html)**class in runtime. If you are using the @SpringBootTest annotation, a TestRestTemplate is automatically available and can be @Autowired into your test.**

**HttpHeaders[HttpHeaders headers = new HttpHeaders();]**

**HTTP headers let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon ( : ), then by its value. ... Response headers hold additional information about the response, like its location or about the server providing it.**

**HttpHeaders headers = new HttpHeaders();**

**headers. setContentType(MediaType. APPLICATION\_JSON);**

**headers. set("Authorization", "Bearer "+accessToken);**

**HttpEntity<String> entity = new HttpEntity<String>(requestJsonBody,headers);**

**HttpEntity[HttpEntity<String> entity = new HttpEntity<String>(“reqestBody”, headers);]**

|  |
| --- |
| **Represents an HTTP request or response entity, consisting of headers and body. Typically used in combination with the**[**RestTemplate**](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/client/RestTemplate.html)**, like so:**  **HttpHeaders headers = new HttpHeaders();**  **headers.setContentType(MediaType.TEXT\_PLAIN);**  **HttpEntity<String> entity = new HttpEntity<String>(helloWorld, headers);**  **URI location = template.postForLocation("https://example.com", entity);** |

**TestRestTemplate.exchange()[ResponseEntity<String> response=restTemplate.exchange(url, HttpMethod.*GET*, entity, String.class)]**

**ResponseEntity<String> response = restTemplate.exchange(url ,HttpMethod.GET, entity, String.class);**

**ResponseEntity represents the whole HTTP response: status code, headers, and body. Here we are calling TestRestTemplate exchange method to get Http response, We are passing request URL ,request Method(GET) , http entity having request body & headers and string.class.**

**Note: Since here we are trying to fetch only one question that is why we have passed String.class, in case of retrieving list of question we will have to pass List as shown in the below example**

**Step21.md(Adding Integration Test for Get Request: Getting List of questions )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step21.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step21.md)

|  |
| --- |
| **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.assertTrue;**  **import java.util.Arrays;**  **import java.util.List;**  **import org.junit.Before;**  **import org.junit.Test;**  **import org.junit.runner.RunWith;**  **import org.skyscreamer.jsonassert.JSONAssert;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.boot.context.embedded.LocalServerPort;**  **import org.springframework.boot.test.context.SpringBootTest;**  **import org.springframework.boot.test.web.client.TestRestTemplate;**  **import org.springframework.core.ParameterizedTypeReference;**  **import org.springframework.http.HttpEntity;**  **import org.springframework.http.HttpHeaders;**  **import org.springframework.http.HttpMethod;**  **import org.springframework.http.MediaType;**  **import org.springframework.http.ResponseEntity;**  **import org.springframework.test.context.junit4.SpringRunner;**  **import com.in28minutes.springboot.Application;**  **import com.in28minutes.springboot.model.Question;**  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)**  **public class SurveyControllerIT {**  **@LocalServerPort**  **private int port;**  **// Generate the Random port number**  **@Autowired**  **private TestRestTemplate template;**  **// private TestRestTemplate template = new TestRestTemplate();**  **@Test**  **public void retrieveAllSurveyQuestions() throws Exception {**  **String url = "http://localhost:" + port + "/surveys/Survey1/questions";**  **// Create the rest url which will be used inside TestRestTemplate to be invoked.**  **TestRestTemplate restTemplate = new TestRestTemplate();**  **HttpHeaders headers = new HttpHeaders();**  **headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));**  **// Setting up response type which we will get**  **HttpEntity<String> reqBody\_n\_Header = new HttpEntity<String>("NoReqBodyAsItIsGetReq", headers);**  **ResponseEntity<List<Question>> response = restTemplate.exchange(url, HttpMethod.GET,**  **reqBody\_n\_Header, new ParameterizedTypeReference<List<Question>>() {});**  **// Passing Rest URL,**  **Question sampleQuestion = new Question("Question2", "Largest Country in the World", "Russia",**  **Arrays.asList("India", "Russia", "United States", "China"));**  **assertTrue(response.getBody().contains(sampleQuestion));**  **}**  **}**  **Here instead of string.class we have passed list of questions in ParameterizedTypeReference and in response we will get a list of question and in the unit test assertTure method we are checking the condition if the sampleQuestion present in the response.** |

**JSONAssert.*assertEquals*(expected, response.getBody(), false);**

**Using assertEquals method we are just comparing the response with the expected**

**ParameterizedTypeReference**

[**https://shuaibabdulla40.medium.com/need-and-usage-of-rest-template-parameterizedtypereference-13a3a3977f6f**](https://shuaibabdulla40.medium.com/need-and-usage-of-rest-template-parameterizedtypereference-13a3a3977f6f)

|  |
| --- |
| **Execute the HTTP method to the given URI template, writing the given request entity to the request, and returns the response as**[**ResponseEntity**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html)**. The given**[**ParameterizedTypeReference**](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/core/ParameterizedTypeReference.html)**is used to pass generic type information:**  **I would like to describe the usage of the ParameterizedTypeReference in the Rest template.**  **In a test, I wish to hit an endpoint that returns a list of a type. Currently, I have**  **@Test public void when\_endpoint\_is\_hit\_then\_return\_list(){  //Given  ParameterizedTypeReference<List<Example>> responseType = new ParameterizedTypeReference<List<Example>>() {};**  **String targetUrl = "/path/to/hit/" + expectedExample.getParameterOfList(); //When //This works however**  **highlights an unchecked assignment of List to List<Example>  List<Example> actualExample = restTemplate.getForObject(targetUrl, List.class); //This does not work  List<Example> actualExample = restTemplate.getForObject(targetUrl, responseType); //This does not work either  List<Example> actualExample = restTemplate.exchange(targetUrl, HttpMethod.GET, null, new**  **ParameterizedTypeReference<List<Example>>() {}); //Then  //Assert Results }**  **The problem for the getForObject method is the ParameterizedTypeReference makes the getForObject method not resolve, as the types do not match up.**  **The problem for the exchange method is incompatible types. Required List but ‘exchange’ was inferred to ResponseEntity: no instance(s) of type variable(s) exist so that ResponseEntity conforms to List**  ***Solution***  **Execute the HTTP method to the given URI template, writing the given request entity to the request, and returns the response as ResponseEntity. The given ParameterizedTypeReference is used to pass generic type information:**  **ParameterizedTypeReference<List<MyBean>> myBean = new ParameterizedTypeReference<List<MyBean>>() {}; ResponseEntity<List<MyBean>> response = template.exchange("http://example.com",HttpMethod.GET, null, myBean);**  **So in your case, you can:**  **ResponseEntity<List<Example>> actualExample = restTemplate.exchange(targetUrl, HttpMethod.GET, null, new ParameterizedTypeReference<List<Example>>() {}); List<Example> exampleList = actualExample.getBody();** |

**Step22.md(Adding Integration Test for POST Request: Adding question into List of questions )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step22.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step22.md)

|  |
| --- |
| **//NEEDS REFACTORING**  **@Test**  **public void retrieveAllSurveyQuestions() throws Exception {**  **// Create the Post URL to be invoked in the TestRestTemplate**  **// Create HttpHeaders object(headers) and set the MediaType.Application\_JSON using headers.setAccept(….)**  **// Create the request body which we will pass into TestRestTemplate**  **// Create the HttpEntity where we will pass RequestBody and httpHeadders to get httpEntity which further will be**  **passed into TestRestTemplate**  **// Create the object of TestRestTemplate testRestTemplate as shown below in the code**  **// Invoke the exchange() method of the TestRestTemplate utility class and pass the required parameter to invoke post**  **request exchange(URL, HttpMethod.POST, httpEntity(having ReqBody and header) , responseType(String or class**  **type)).**  **// The exchange method will return ResponseEntity object of string type because here the response type we have**  **Mentioned as String. Since we are invoking POST method so it will return the location list in response. So next**  **// Extract the actualLocation from ResponseEntity String actualLocation =**  **response.getHeaders().get(HttpHeaders.*LOCATION*).get(0);**  **// Now let’s check that actualLocation contains the POST method request URL or not. Unit test will succeed if actual**  **location contains the POST method request URL.** |
| **@Test**  **public void createSurveyQuestion() throws Exception {**  **// Create the Rest URL for POST method : @PostMapping("/surveys/{surveyId}/questions")**  **String url = "http://localhost:" + port + "/surveys/Survey1/questions";**  **// Create the HttpHeaders for setting up media type (APPLICATION\_JSON)**  **HttpHeaders headers = new HttpHeaders();**  **headers.setAccept(Arrays.*asList*(MediaType.*APPLICATION\_JSON*));**  **// Creating Request Body to be passed into HttpEntity**  **Question questionResBody = new Question("QuestionIDGenrateAuto", "IntegrationTestQuestion", "Russia",**  **Arrays.*asList*("India", "Russia", "United States", "China"));**  **// Creating HttpEntity where we will pass request body and http header to get entity which will further pass into**  **TestRestTemplate**  **HttpEntity<Question> entity = new HttpEntity<Question>(questionResBody, headers);**  **// Creating TestRestTemplate using which we will use to invoke rest url with**  **// http Entity and POST method to get response**  **TestRestTemplate tesRestTemplate = new TestRestTemplate();**  **ResponseEntity<String> response = tesRestTemplate.exchange(url, HttpMethod.*POST*, entity, String.class);**  **// The above line will return the URI of the next questions as a response so we will give the list of the location from the response**  **System.*out*.println("Response:::" + response);**  **// Response:::<201 Created,{Connection=[keep-alive],**  **// Location=[http://localhost:63475/surveys/Survey1/questions/qedeje4kepeicgqlldcn6qovhl],**  **// Content-Length=[0], X-Application-Context=[application:0], Date=[Mon, 15 Feb**  **// 2021 11:23:47 GMT]}>**  **String actualLocation = response.getHeaders().get(HttpHeaders.*LOCATION*).get(0);**  **System.*out*.println("actualLocation:::" + actualLocation);**  **// Now we will check that the actualLocation contains the post request url[/surveys/Survey1/questions]**  ***assertTrue*(actualLocation.contains("/surveys/Survey1/questions"));**  **}** |

**Below Log is from SurveyContoller post method :**

**URI:::1111http://localhost:63475/surveys/Survey1/questions/qedeje4kepeicgqlldcn6qovhl**

**Status:::qqqqq:<201 Created,{Location=[http://localhost:63475/surveys/Survey1/questions/qedeje4kepeicgqlldcn6qovhl]}>**

**Status::::<201 Created,{}>**

**The below log is from TestIntegration Controller for POST method**

**Response:::<201 Created,{Connection=[keep-alive], Location=[http://localhost:63475/surveys/Survey1/questions/qedeje4kepeicgqlldcn6qovhl], Content-Length=[0], X-Application-Context=[application:0], Date=[Mon, 15 Feb 2021 11:23:47 GMT]}>**

**actualLocation:::http://localhost:63554/surveys/Survey1/questions/qedeje4kepeicgqlldcn6qovhl**

|  |  |
| --- | --- |
| **Before Refactoring Snippet**  **//NEEDS REFACTORING**  **@Test**  **public void retrieveAllSurveyQuestions() throws Exception {**  **String url = "http://localhost:" + port + "/surveys/Survey1/questions";**  **TestRestTemplate restTemplate = new TestRestTemplate();**  **HttpHeaders headers = new HttpHeaders();**  **headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));**  **ResponseEntity<List<Question>> response = restTemplate.exchange(url,**  **HttpMethod.POST, new HttpEntity<String>("DUMMY\_DOESNT\_MATTER",headers), new ParameterizedTypeReference<List<Question>>() {});**  **Question sampleQuestion = new Question("Question1",**  **"Largest Country in the World", "Russia", Arrays.asList("India", "Russia", "United States", "China"));**  **assertTrue(response.getBody().contains(sampleQuestion));**  **}**  **Wrong Code : Try to correct the above code for POST method.** | **After Refactoring Snippet - We will discuss this in Step 23**  **@Test**  **public void createSurveyQuestion() throws Exception {**  **Question question = new Question("DOESN'T MATTER", "Smallest Number","1", Arrays.asList("1", "2", "3", "4"));**  **ResponseEntity<String> response = template.exchange(**  **createUrl("/surveys/Survey1/questions/"), HttpMethod.POST,**  **new HttpEntity<Question>(question, headers), String.class);**  **assertThat(response.getHeaders().get(HttpHeaders.LOCATION).get(0), containsString("/surveys/Survey1/questions/"));**  **}** |

**Unit Testing Using Mockito:**

**Step24.md(Adding Integration Test for POST Request: Adding question into List of questions )**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step24.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step24.md)

## What You Will Learn during this Step:

* **Unit Testing with stubs**
* **First Mock with Mockito**
* **Mockito Annotation**
* **Advanced Mockito Example**
* **Write a Unit Test for retrieving a specific question from a survey.**
* **Different between Unit Test and Integration Test**
* **Basics of Mocking**
* **MockMvc framework**
* **@MockBean**
* **Programming Tip**

|  |  |
| --- | --- |
| **package com.in28minutes.springboot.controller;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import com.in28minutes.springboot.service.DataService;**  **public class SquareMockitoClass {**  **@Autowired**  **private DataService dataService;**  **int findGreatestData() {**  **int[] data = dataService.retriveAllData();**  **int greatest = Integer.MAX\_VALUE;**  **for (int value : data) {**  **if (value > greatest)**  **greatest = value;**  **}**  **return greatest;**  **}}** | **package com.in28minutes.springboot.service;**  **import org.springframework.stereotype.Component;**  **@Component**  **public interface DataService {**  **int[] retriveAllData();**  **}**  **Here we have created one DataService component class and one class to use this service method for getting greatest number.**  **Now to test these two method let’s do some unit testing using stubs** |

**Create the Junit test class using the below step**

**Right Click on Project> Provide the Test Class Name > Select the class to be tested.**

|  |  |
| --- | --- |
|  | **package com.in28minutes.springboot.service;**  **import org.springframework.stereotype.Component;**  **@Component**  **public interface DataService {**  **int[] retriveAllData();**  **}** |
| **Let’s create the implementation class of the DataService interface:**  **package com.in28minutes.springboot.controller;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import com.in28minutes.springboot.service.DataService;**  **public class SquareStubClass {**  **// Generate the consructor to accest the data service**  **public SquareStubClass(DataService dataService) {**  **super();**  **this.dataService = dataService;**  **}**  **@Autowired**  **private DataService dataService;**  **int findGreatestData() {**  **int[] data = dataService.retriveAllData();**  **int greatest = Integer.MIN\_VALUE;**  **for (int value : data) {**  **if (value > greatest)**  **greatest = value;**  **}**  **System.out.println("Greatest ::" + greatest);**  **return greatest;**  **}**  **}**  **// Implement the interface method**  **class DataServiceStub implements DataService {**  **// Method having 3 data**  **@Override**  **public int[] retriveAllData() {**  **// TODO Auto-generated method stub**  **return new int[] { 4, 11, 25 };**  **}**  **// Method having only one data**  **@Override**  **public int[] retriveAllData() {**  **// TODO Auto-generated method stub**  **return new int[] { 4 };**  **}**  **// Method having no data**  **@Override**  **public int[] retriveAllData() {**  **// TODO Auto-generated method stub**  **return new int[] { };**  **}**  **}** | **Click Finish and initially we will get the below generated code.**  **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.*assertEquals*;**  **import org.junit.Test;**  **public class SqueareMockitoTest {**  **@Test**  **public void testfindGreatestData() {**  **SquareStubClass stubClass = new SquareMockitoClass(new DataServiceStub());**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(25, result);**    **}**  **}**  **Here as we can see that to test the greatest value we have to write stubs. Disadvantage of unit testing using Stubs**   1. **We have to write a lot of codes like we have written Class DataServiceStub** 2. **If we want to test some other values then we need to write several methods and keep calling in the Junit test calls to test.** 3. **If we want to test different scenario then we have to write code like as given below.**   **// Method having only one data**  **// Method having no data**  **So we need to create different kind of stubs to do DataService testing.**  **To solve this problem Mock came into existence. Using mock we really don’t need to create different kind of stubs for testing.**  **Mock makes very easy to create dynamically different classes and make that the data return that would want.** |

**Let’s create a mock and try to run the above class using Mockito.**

|  |  |
| --- | --- |
| **@Test**  **public void testfindGreatestMockData\_three() {**  **// Creates mock object of given class or interface.**  **// Create a mock for the class which needs to be mocked**  **DataService dataServiceMock = *mock*(DataService.class);**  **/\***  **Call the implemented method of interface DataService, which will return an array of Integer. But we don't need to call the below method to get array of integer, here in case of mock we will use when(retriveAllData is called)thenReturn new Array of Integer dataServiceMock.retriveAllData();**  **\*/**  ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] {**  **12, 2, 33 });**  **// SquareStubClass mockitoClass = new SquareStubClass(new DataServiceStub());**  **SquareStubClass mockitoClass = new SquareStubClass(dataServiceMock);**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(33, result);** } | **@Test**  **public void testfindGreatestMockData\_two() {**  **// Creates mock object of given class or interface.**  **// Create a mock for the class which needs to be mocked**  **DataService dataServiceMock = *mock*(DataService.class);**  ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] { 12, 11 });**  **SquareStubClass mockitoClass = new SquareStubClass(dataServiceMock);**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(12, result);** } **@Test**  **public void testfindGreatestMockData\_One() {**  **// Creates mock object of given class or interface.**  **// Create a mock for the class which needs to be mocked**  **DataService dataServiceMock = *mock*(DataService.class);**  ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] { 12});**  **SquareStubClass mockitoClass = new SquareStubClass(dataServiceMock);**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(12, result);**  **}** |

**So as above we can see that just need to provide differtnt type of pattern to be tested and we don’t need to implement these methods of the different type of pattern. Using mockito we can provide the implementation at run time.**

**Here we have just the below thing in Mockito.**

* **Created mock object of the class or interface which needs to be mocked.**
* **Using mock object called the method of the mocking class or interface.**
* **Using When(call the class method to be mocked using mock object)thenReturn(provide the statement or argument to be returned)**
* **Then using assertEqual method just check the result**
* **Note : Whenever using mock then there are two things that we need to take care**
* **1- How do we send the mock out to the class we want to test: For example here we are sending the mock object into the SquareStubClass using constructor (which is nothing but constructor injection). So mocking data sending means injecting dependency. Other type of Dependncy inection is setter type DI.**

**Junit Testing using Mockito Annotations - @Mock, @Injection, @RunWith (MockitoJUnitRunner.class)**

**As above we have seen that we have used mock(class to be mocked) keyword.**

**Like [DataService dataServiceMock = *mock* (DataService.class);]. But now instead of using mock keyword we can just use annotation @Mock on top of the class when needs to be mocked**

|  |
| --- |
| **@Mock**  **DataService dataServiceMock;** |

**After this we need to inject this Mock object into the Consturctor or Setter method. Like in the old code the mock object was injected like as given below:**

**SquareStubClass mockitoClass = new SquareStubClass (dataServiceMock);**

**But now instead of injecting like above we can just use @InjectMocks annotation at the top of injector class as given below.**

|  |
| --- |
| **@InjectMocks**  **SquareStubClass mockitoClass;** |

**Note: Now after using these two annotations if we try to run the above code then Junit test will be failed because to use the above annotation inside the @Test method we need to use below annotation at the calss level @RunWith(MockitoJUnitRunner.class)**

## So finally the Mockito testing using annotation will be given as below.

|  |
| --- |
| **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.*assertEquals*;**  **import org.junit.Test;**  **import org.junit.runner.RunWith;**  **import org.mockito.InjectMocks;**  **import org.mockito.Mock;**  **import org.mockito.runners.MockitoJUnitRunner;**  **import static org.mockito.Mockito.*mock*;**  **import static org.mockito.Mockito.*when*;**  **import com.in28minutes.springboot.service.DataService;**  **@RunWith(MockitoJUnitRunner.class)**  **public class SqueareMockitoTest {**  **@Mock**  **DataService dataServiceMock;**  **@InjectMocks**  **SquareStubClass mockitoClass;**  **@Test**  **public void testfindGreatestMockData() {**  **// Creates mock object of given class or interface.**  **//DataService dataServiceMock = mock(DataService.class);**    ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] { 12, 2, 33 });**  **//SquareStubClass mockitoClass = new SquareStubClass(dataServiceMock);**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(33, result);**  **}**  **@Test**  **public void testfindGreatestMockData\_two() {**  **// Creates mock object of given class or interface.**  **// DataService dataServiceMock = mock(DataService.class);**  ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] { 12, 11 });**  **// SquareStubClass mockitoClass = new SquareStubClass(dataServiceMock);**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(12, result);**  **}**  **}** |

**So here we have used three annotations (@RunWith (MockitoJUnitRunner.class) @Mock, @InjectMocks). So when test class runs then first @RunWith annotation create the object of @Mock annotated calss and @InjectMock annotated class and inject the mock class object gets injected into @InjectMock annotated class.**

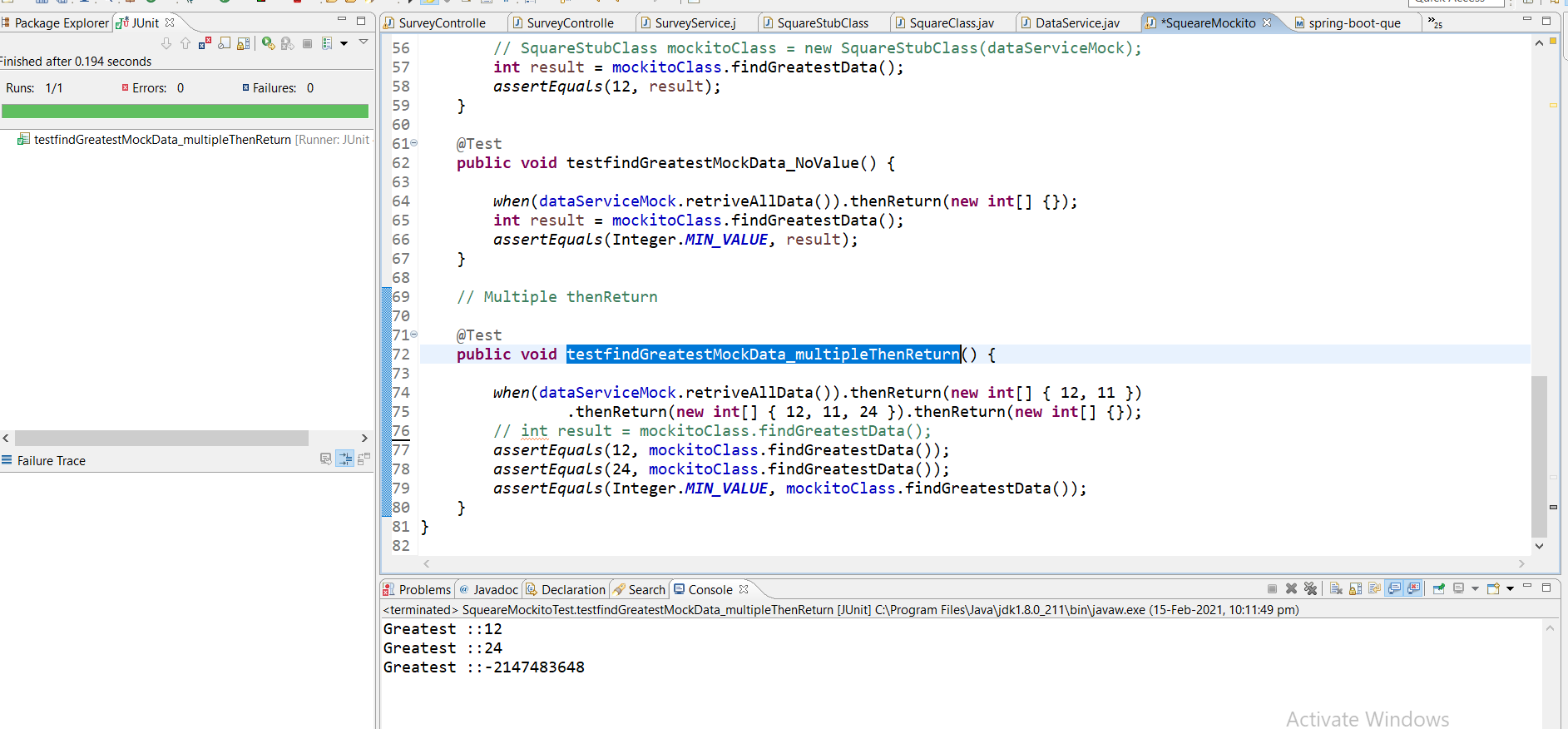
|  |
| --- |
| **@Test**  **public void testfindGreatestMockData\_NoValue() {**    ***when*(dataServiceMock.retriveAllData()).thenReturn(new int[] {});**  **int result = mockitoClass.findGreatestData();**  ***assertEquals*(Integer.*MIN\_VALUE*, result);**  **}** |

**So Using Mockito Junit testing becomes very easy.**

**Mocking List Interface: (Using multiple thenReturn() with when: Using Mockito.anyType())**

**Here let’s mock the List Interface**

|  |
| --- |
| **@Test**  **public void testfindGreatestMockData\_multipleThenReturn() {**  **when(dataServiceMock.retriveAllData()).thenReturn(new int[] { 12, 11 })**  **.thenReturn(new int[] { 12, 11, 24 }).thenReturn(new int[] {});**  **// int result = mockitoClass.findGreatestData();**  **assertEquals(12, mockitoClass.findGreatestData());**  **assertEquals(24, mockitoClass.findGreatestData());**  **assertEquals(Integer.MIN\_VALUE, mockitoClass.findGreatestData());**  **}** |

****

|  |  |
| --- | --- |
| **@Test**  **public void testListAny() {**  **List mockList = *mock*(List.class);**  ***when*(mockList.size()).thenReturn(10);**  ***assertEquals*(10, mockList.size());**  **}** | **@Test**  **public void testListAny() {**  **List mockList = *mock*(List.class);**  ***when*(mockList.size()).thenReturn(10).thenReturn(20);**  ***assertEquals*(10, mockList.size());**  ***assertEquals*(20, mockList.size());**  **}** |
| **Here if want to return some string on calling first postion of list then**  **@Test**  **public void testListAny() {**  **List mockList = *mock*(List.class);**  ***when*(mockList.size()).thenReturn(10).thenReturn(20);**  ***assertEquals*(10, mockList.size());**  ***assertEquals*(20, mockList.size());**  ***when*(mockList.get(0)).thenReturn("SomeString");**  ***assertEquals*("SomeString", mockList.get(0));**  **}** | **If we want to return string on any integer postion**  **@Test**  **public void testListAny() {**  **List mockList = *mock*(List.class);**  ***when*(mockList.size()).thenReturn(10).thenReturn(20);**  ***assertEquals*(10, mockList.size());**  ***assertEquals*(20, mockList.size());**  ***when*(mockList.get(0)).thenReturn("SomeString");**  ***assertEquals*("SomeString", mockList.get(0));**  ***when*(mockList.get(Mockito.*anyInt*())).thenReturn("My Name");**  ***assertEquals*("My Name", mockList.get(11000));**  **}** |

## So as here we can see that we can set the particular things like string at anyInt. So if we want set any default string or any thing then we can use the above example.

**Step24.md(Writing Unit Test with Spring Boot and Mockito)**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step24.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step24.md)

**What You Will Learn during this Step:**

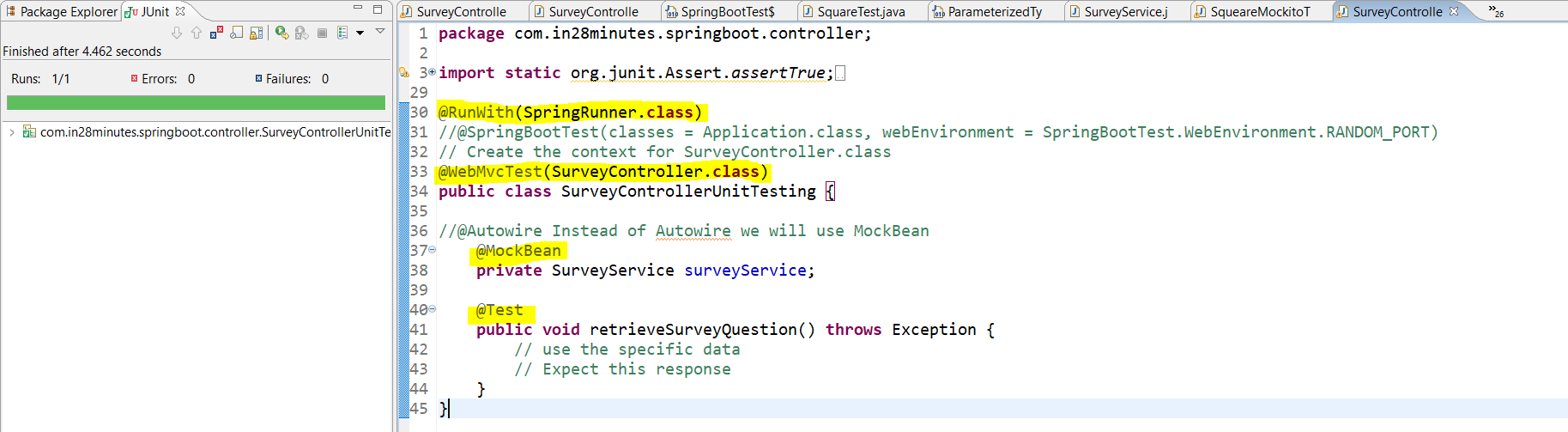
* **Write a Unit Test for retrieving a specific question from a survey.**
* **Different between Unit Test and Integration Test**
* **Basics of Mocking**
* **MockMvc framework**
* **@MockBean**

|  |  |
| --- | --- |
| **Integration Testing**  **@RunWith(SpringRunner.class)**  **@SpringBootTest(classes = Application.class, webEnvironment =**  **SpringBootTest.WebEnvironment.*RANDOM\_PORT*)**  **public class SurveyControllerUnitTesting {**  **@LocalServerPort**  **private int port;**  **@Autowired**  **private TestRestTemplate template;**  **HttpHeaders headers = new HttpHeaders();**  **@Before**  **public void setupJSONAcceptType() {**  **headers.setAccept(Arrays.*asList*(MediaType.*APPLICATION\_JSON*));**  **}**  **@Test**  **public void retrieveSurveyQuestion() throws Exception {**  **}**  **@Test**  **public void retrieveAllSurveyQuestions() throws Exception {**  **}**  **}** | **As here in case of Intgrtion testing we have used the following things listed below.**  **@RunWith(SpringRunner.class) : To Launch the context**  **@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT*)**  **classes = Application.class**  **Here insdie the @SpringBootTest we have mentioned Application.class which tells the spring boot frame work to create context at the entire application level and launch the entire spring boot application.**  **webEnvironment = SpringBootTest.WebEnvironment.*RANDOM\_PORT***  **@LocalServerPort**  **private int port;**  **This is being used to create random port at run time so that multiple unit testing can be executed simultaneously on dofferent port and we could not get any exception related to server in use** |

**So in case of Integration testing we will have to create conext at application level so that we can launch entir spring boot application and can do end to end testing From Controller 🡪 Service 🡪 To JPA and other stuffs.**

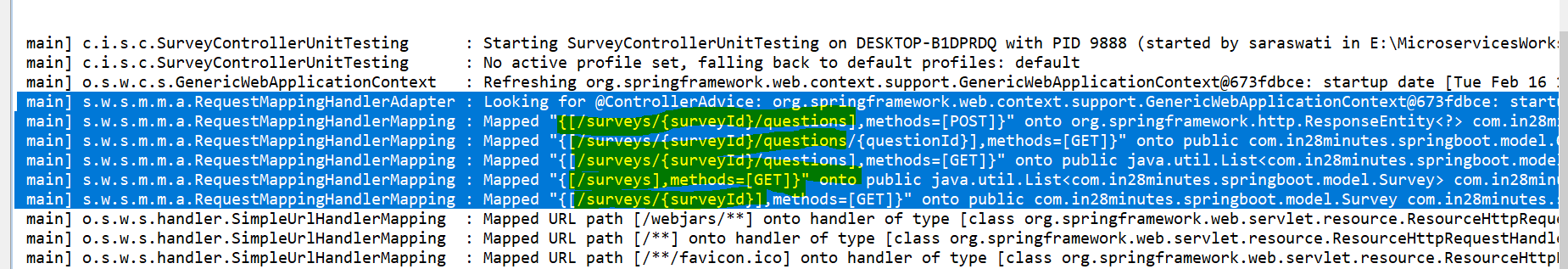
**But- In case of unit test we don’t need to launch entire application, we just only need to launch the specific class or controller that we want to test. And in case of mock we don’t want to get the data from Service 🡪 DB to test. We just need to mock the Service class with our own mock data.**

|  |  |
| --- | --- |
| **@RunWith(SpringRunner.class)**  **//@SpringBootTest(classes = Application.class, webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)**  **// Create the context for SurveyController.class**  **public class SurveyControllerUnitTesting {**  **//@Autowire Instead of Autowire we will use MockBean**  **@MockBean**  **private SurveyService surveyService;**  **@Test**  **public void retrieveSurveyQuestion() throws Exception {**  **// use the specific data**  **// Expect this response**  **}**  **}** | **So here in case of Unit testing.**  **We will launch the class to which unit testing needs to be done.So we will create the context at class level.**  **To lauch the specific class we will use @WebMvcTest(SurveyController.class)**  **Then using mock we will test the unit testing. To enable mock we will use @MockBean at the using in the mock.**  **Now the SpringRunner.class would pick the @MockBean annotated class (SurveyService) and Autowired into @WebMvcTest(SurveyController.class) automatically.** |

****

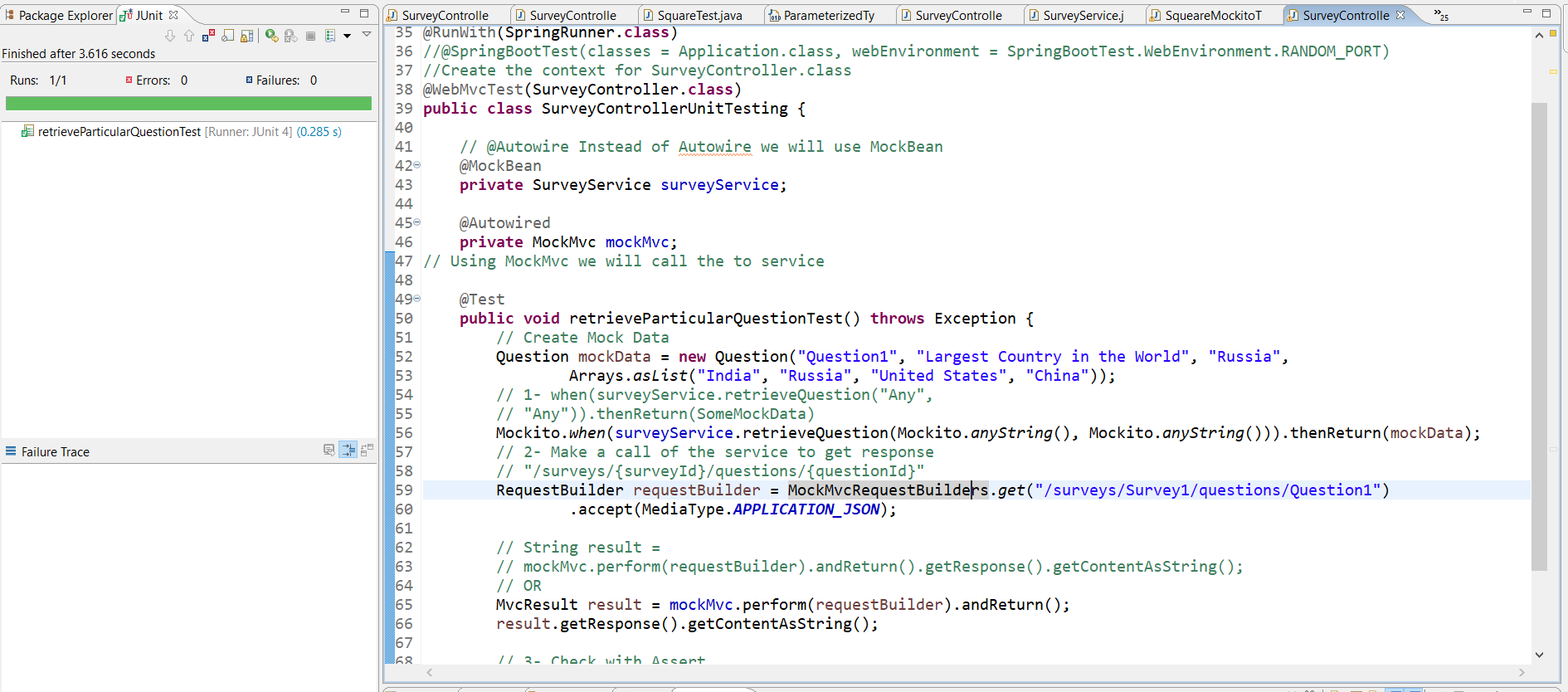
**Now if we see in the log then we will see that it would load all the Rest Url mentioned in the SurveyController class, while the URL mentioned into another controller class would not be loaded because here we launching SurveyController class only using @WebMvcTest (SurveyController.class) and it takes less time to run in compare to integration testing.**

**@WebMvcTest will not perform the component scan unlike (@SpringBootTest (classes= Application.class)) and thus it would not load other beans available in the spring boot application. It would the load the only bean of the class mentioned inside @WebMvcTest.**

****

**So till now we got to know that how lauch the spring boot application Unit testing. Now we will with the coding to test the controller class. So basically inside the @Test annotated class we will do the following things.**

|  |  |
| --- | --- |
| **@Autowired**  **private SurveyService surveyService;**  **@GetMapping("/surveys/{surveyId}/questions/{questionId}")**  **public Question retrieveParticularQuestion(@PathVariable String surveyId, @PathVariable String questionId) {**  **return surveyService.retrieveQuestion(surveyId, questionId);**  **}** | **As here we have Autowired SurveyService class and then using object of this class we are calling retrieveQuestion to get the particular question based on SurveyId and questionID.** |
| **While in case of Unit testing:**  **Instead of using @Autowired we will use @MockBean on the SurveyService class to create the object for Mocking.**   1. **Annotate the class @RunWith(SpringRunner.class) @WebMvcTest(SurveyController.class) to launch the specified class context** 2. **Annotate the Service calls with @MockBean to be mocked and to Autowired in the SurveyController class** 3. **Created and annotate with @Autowired MockMvc class just to make a service call and retrun the Request Builder having response.** 4. **Created a MockData to be returned in mocking** 5. **Now Start Moking using : when(surveyService.retrieveQuestion("Any", "Any")).thenReturn(MockData)** 6. **Make a call of the service to get response "/surveys/{surveyId}/questions/{questionId}"** 7. **Check with AssertEqual(MockData, resultData)** | |
| **package com.in28minutes.springboot.controller;**  **import static org.junit.Assert.assertTrue;**  **// All Imports**  **@RunWith(SpringRunner.class)**  **@WebMvcTest(SurveyController.class)**  **public class SurveyControllerUnitTesting {**  **// @Autowire Instead of Autowire we will use MockBean**  **@MockBean**  **private SurveyService surveyService;**  **@Autowired**  **private MockMvc mockMvc;**  **// Using MockMvc we will call the to service**  **@Test**  **public void retrieveParticularQuestionTest() throws Exception {**  **// Create Mock Data**  **Question mockData = new Question("Question1", "Largest Country in the World", "Russia",**  **Arrays.asList("India", "Russia", "United States", "China"));**  **//1- when(surveyService.retrieveQuestion("Any", "Any")).thenReturn(mockData)**  **Mockito.when(surveyService.retrieveQuestion(Mockito.anyString(), Mockito.anyString())).thenReturn(mockData);**  **//2- Make a call of the service to get response "/surveys/{surveyId}/questions/{questionId}"**  **RequestBuilder *requestBuilder* = MockMvcRequestBuilders.get("/surveys/Survey1/questions/Question1")**  **.accept(MediaType.APPLICATION\_JSON);**  **// String result = mockMvc.perform(requestBuilder).andReturn().getResponse().getContentAsString();**  **// OR**  **MvcResult result = mockMvc.perform(*requestBuilder*).andReturn();**  **result.getResponse().getContentAsString();**  **//3- Check with Assert**  **String expected = "{id:Question1,description:Largest Country in the World,correctAnswer:Russia}";**  **JSONAssert.assertEquals(expected, result.getResponse().getContentAsString(), false);**  **}**  **}** | |

****

**Now if we see in the console then we will the below logs**

|  |
| --- |
| **MockHttpServletRequest:**  **HTTP Method = GET**  **Request URI = /surveys/Survey1/questions/Question1**  **Parameters = {}**  **Headers = {Accept=[application/json]}**  **Handler:**  **Type = com.in28minutes.springboot.controller.SurveyController**  **Method = public com.in28minutes.springboot.model.Question com.in28minutes.springboot.controller.SurveyController.retrieveParticularQuestion(java.lang.String,java.lang.String)**  **Async:**  **Async started = false**  **Async result = null**  **Resolved Exception:**  **Type = null**  **ModelAndView:**  **View name = null**  **View = null**  **Model = null**  **FlashMap:**  **Attributes = null**  **MockHttpServletResponse:**  **Status = 200**  **Error message = null**  **Headers = {Content-Type=[application/json;charset=UTF-8]}**  **Content type = application/json;charset=UTF-8**  **Body = {"id":"Question1","description":"Largest Country in the World","correctAnswer":"Russia","options":["India","Russia","United States","China"]}**  **Forwarded URL = null**  **Redirected URL = null**  **Cookies = []**  **2021-02-16 17:34:48.366 INFO 11156 --- [ Thread-2] o.s.w.c.s.GenericWebApplicationContext : Closing org.springframework.web.context.support.GenericWebApplicationContext@2a8448fa: startup date [Tue Feb 16 17:34:45 IST 2021]; root of context hierarchy** |

## Step25.md

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step25.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step25.md)

|  |
| --- |
| What You Will Learn during this Step:  * **Exercise from previous step** * **Unit test for createTodo**  Useful Snippets and References **First Snippet**  **@Test**  **public void retrieveSurveyQuestions() throws Exception {**  **List<Question> mockList = Arrays.asList(**  **new Question("Question1", "First Alphabet", "A", Arrays.asList(**  **"A", "B", "C", "D")),**  **new Question("Question2", "Last Alphabet", "Z", Arrays.asList(**  **"A", "X", "Y", "Z")));**  **when(service.retrieveQuestions(anyString())).thenReturn(mockList);**  **MvcResult result = mvc.perform(MockMvcRequestBuilders.get("/surveys/Survey1/questions")**  **.accept(MediaType.APPLICATION\_JSON))**  **.andExpect(status().isOk()).andReturn();**  **String expected = "["**  **+ "{id:Question1,description:First Alphabet,correctAnswer:A,options:[A,B,C,D]},"**  **+ "{id:Question2,description:Last Alphabet,correctAnswer:Z,options:[A,X,Y,Z]}"**  **+ "]";**  **JSONAssert.assertEquals(expected, result.getResponse().getContentAsString(), false);**  **}**  **Second Snippet**  **@Test**  **public void createSurveyQuestion() throws Exception {**  **Question mockQuestion = new Question("1", "Smallest Number", "1",**  **Arrays.asList("1", "2", "3", "4"));**  **String questionJson = "{\"description\":\"Smallest Number\",\"correctAnswer\":\"1\",\"options\":[\"1\",\"2\",\"3\",\"4\"]}";**  **//surveyService.addQuestion to respond back with mockQuestion**  **Mockito.when(surveyService.addQuestion(Mockito.anyString(), Mockito**  **.any(Question.class))).thenReturn(mockQuestion);**  **//Send question as body to /surveys/Survey1/questions**  **RequestBuilder requestBuilder = MockMvcRequestBuilders.post("/surveys/Survey1/questions")**  **.accept(MediaType.APPLICATION\_JSON).content(questionJson)**  **.contentType(MediaType.APPLICATION\_JSON);**  **MvcResult result = mockMvc.perform(requestBuilder).andReturn();**  **MockHttpServletResponse response = result.getResponse();**  **assertEquals(HttpStatus.CREATED.value(), response.getStatus());**  **assertEquals("http://localhost/surveys/Survey1/questions/1", response.getHeader(HttpHeaders.LOCATION));**  **}** |

**Step-26:**

[**https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step26.md**](https://github.com/in28minutes/spring-boot-master-class/blob/master/05.Spring-Boot-Advanced/Step26.md)

## What You Will Learn during this Step:

* **Securing our services with Basic Authentication using Spring Security**
* **Executing Requests using Basic Authentication with Postman**
* **default user name is user**
* **default security password is printed in console**

## Useful Snippets and References

**First Snippet**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-security</artifactId>**

**</dependency>**

**Second Snippet**

**Using default security password:**

**Third Snippet : Executing a GET to**[**http://localhost:8080/surveys/Survey1/questions/**](http://localhost:8080/surveys/Survey1/questions/)

**{**

**"timestamp": 1483514297025,**

**"status": 401,**

**"error": "Unauthorized",**

**"message": "Full authentication is required to access this resource",**

**"path": "/surveys/Survey1/questions/"**