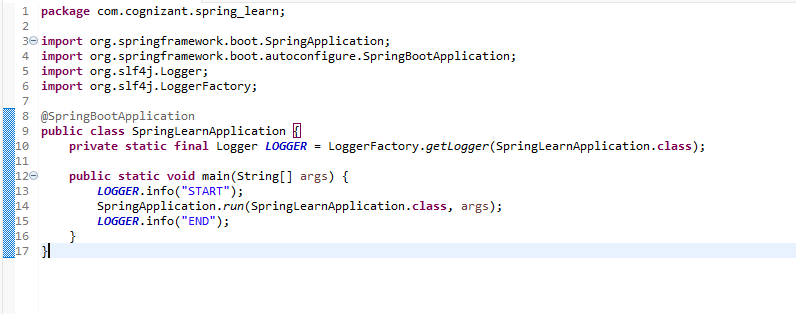
**SKILL LEARNT: SPRING REST USING SPRING BOOT 3 (WEEK 4)**

**EXERCISE 1: CREATE A SPRING WEB PROJECT USING MAVEN**

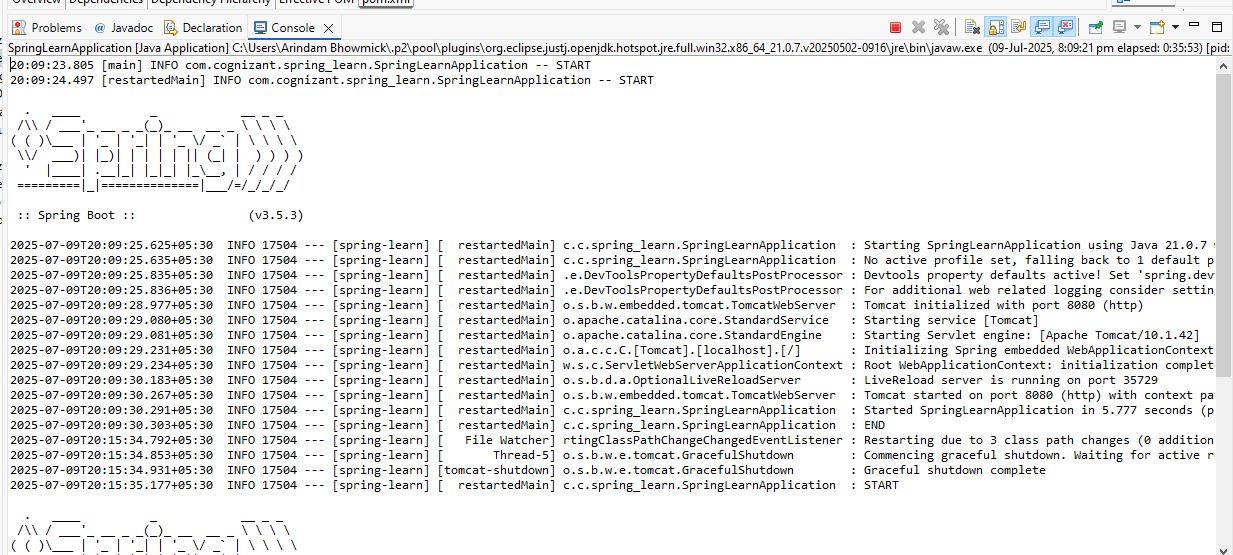
STEP 1: Firstly I visited to Spring Initializr to create the Maven project, I selected project as Maven, Language as Java, Group as com.cognizant, Artifact as spring-learn. I selected two dependencies as stated Spring Boot DevTools, and Spring Web. Then I generated everything as a zip.

STEP 2: Then I uploaded the unzipped folder to my eclipse workspace.

STEP 3: Then under SpringLearnApplication.java, I added logs to verify main() method. The code for SpringLearnApplication.java is given below,



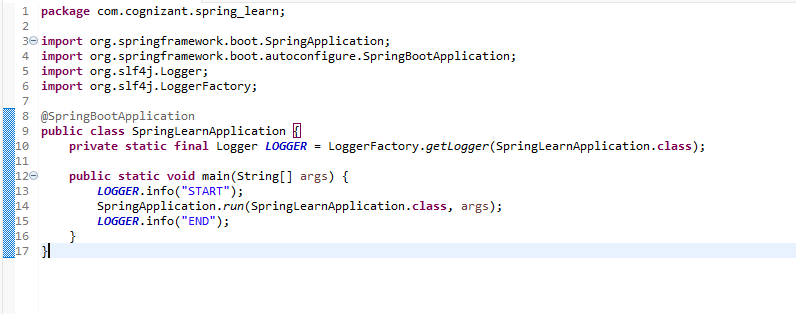
STEP 4: Running the SpringLearnApplication class produces the output.



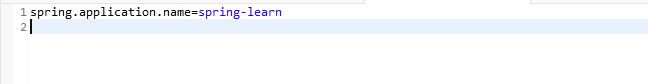


The codes for the following folder structures are:

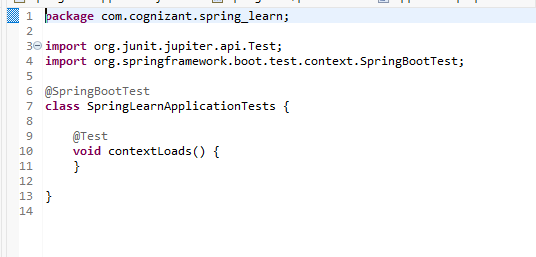
1. src/main/java contains the class named SpringLearnApplication. The code for this is:



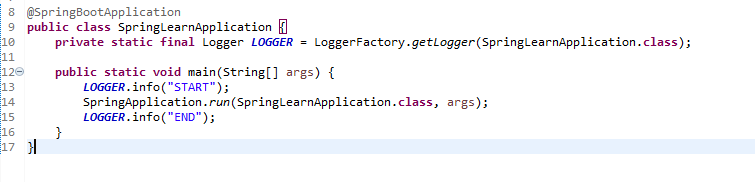
1. src/main/resources contains the folder application.properties. The code for this is shown:



1. src/test/java contains the testing class named SpringLearnApplicationTests.java. The code for this is given:



1. The main method in SpringLearnApplication.java is given as:



@SpringBootApplication: This annotation enables auto configuration, component scanning and sets up Spring Boot.

Main() method: This is the entry point which calls SpringApplication.run() to launch the embedded server.

1. Purpose of SpringBootApplication annotation is:

@SpringBootApplication is a combination of three significant annotations:

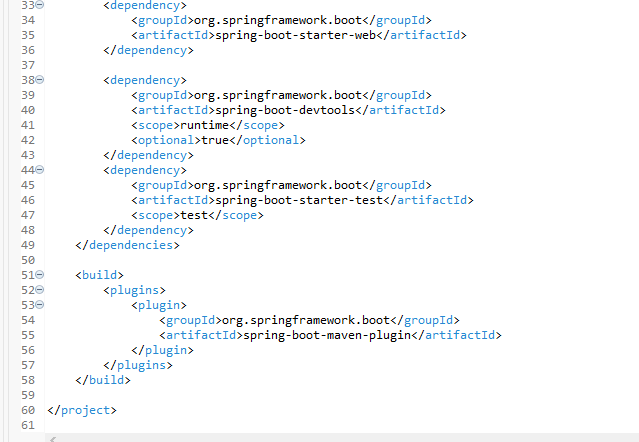
@Configuration: Marks the class as a bean definition source. It eliminates the necessity for explicit configuration classes and enables bean definitions within the application to be picked up automatically.

@EnableAutoConfiguration: Tells Spring Boot to auto-configure the beans and properties from the dependencies of the project (e.g., configuring the web server when Spring Web is on the classpath).

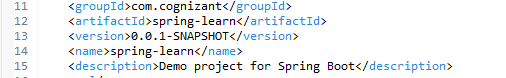
@ComponentScan: It scans for Spring components (such as @Controller, @Service, @Repository, etc.) within the main class package and its sub-packages.

1. The code of pom.xml is shown as

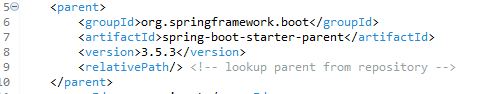




The explanations of each part of project is:



~The above code identifies the project in Maven repositories.



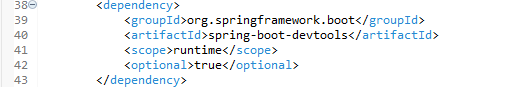
~ The above code inherits defaults from Spring Boot parent, simplifying version management.



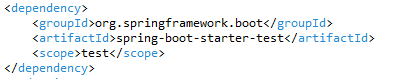
~ The above code sets Java version.



~ The above code provides Spring MVC and embedded Tomcat.

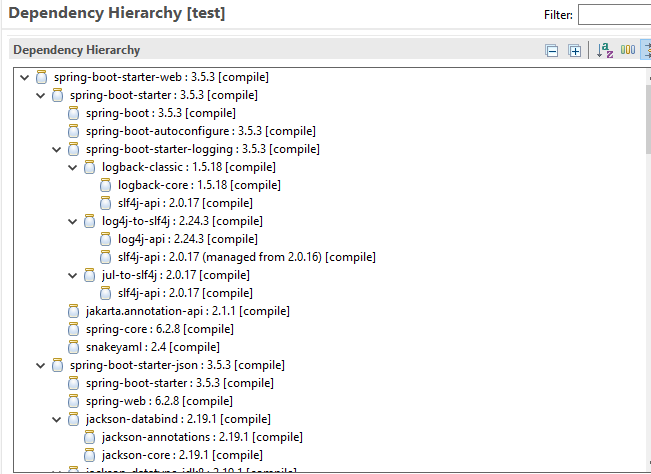


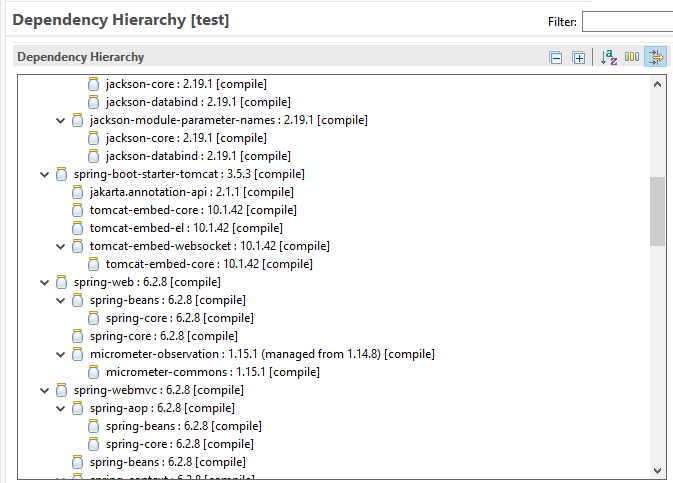
~ The above code enables hot reload and development tools.

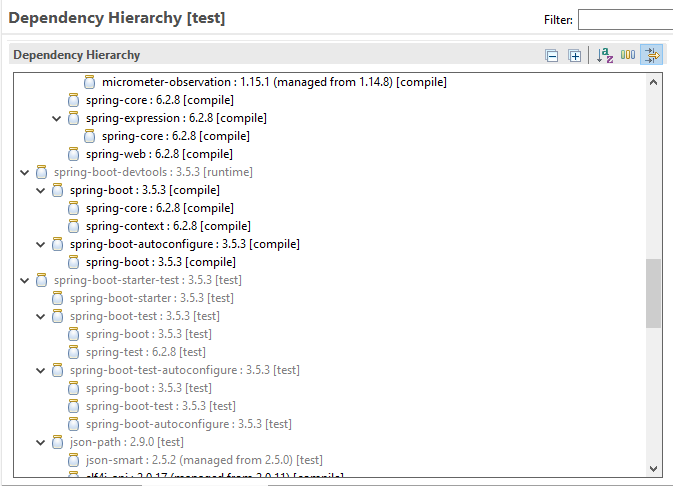


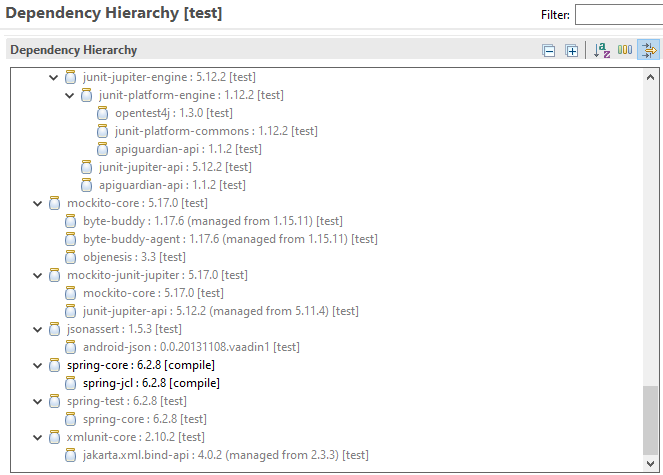
~ The above code adds testing support.

The dependency tree is shown below









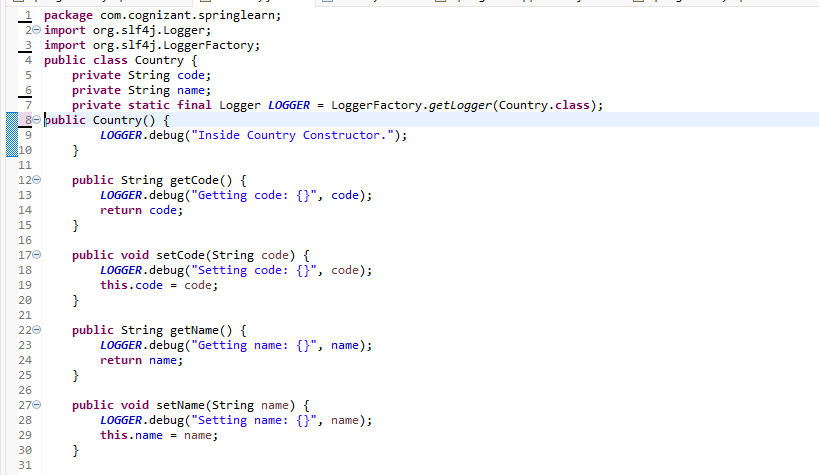
**EXERCISE 2: SPRING CORE. LOAD COUNTRY FROM SPRING CONFIGURATION XML**

STEP 1: Firstly, I created a simple Maven Project with Group Id com.cognizant.springlearn and Artifact id spring-country.

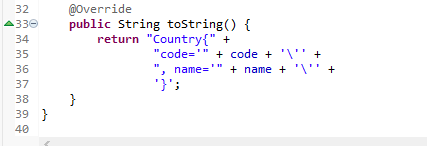
STEP 2: I added dependencies to pom.xml. The code for pom.xml is shown below



STEP 3: Under src/main/java, I created a package named com.cognizant.springlearn under which a class named Country. The code for it is mentioned below



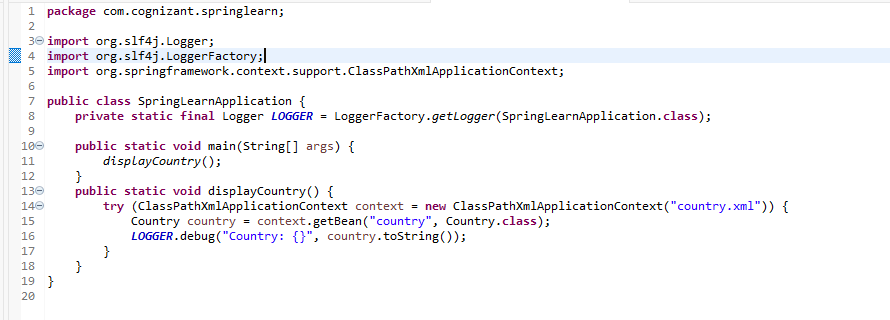
Continued…..



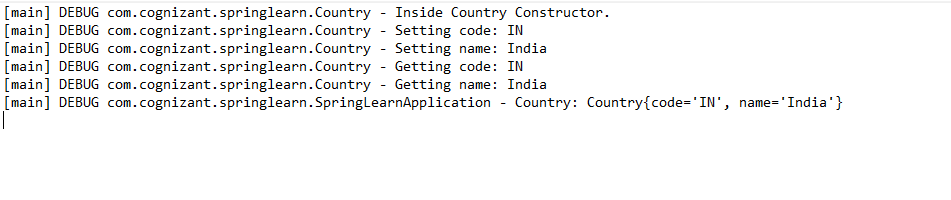
STEP 4: Under src/main/resources, I created a file named country.xml. The code for is given below



STEP 5: Under src/main/java, ad within the packagecom.cognizant.springlearn, I created the class SpringLearnApplication. The code for it is mentioned below



STEP 6: Running the main class i.e. SpringLearnApplication.java produces the output.

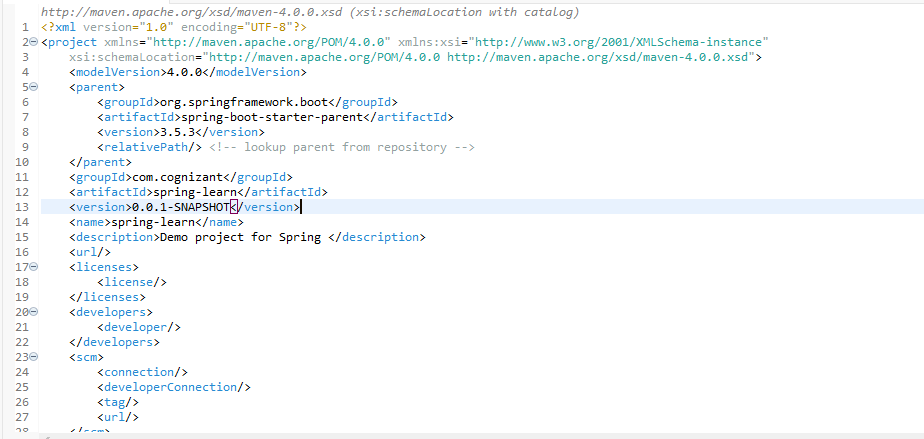


**EXERCISE 3: HELLO WORLD RESTful WEB SERVICE**

STEP 1: Firstly I visited to Spring Initializr to create the Maven project, I selected project as Maven, Language as Java, Group as com.cognizant, Artifact as spring-learn. I selected one dependency Spring Web. Then I generated everything as a zip.

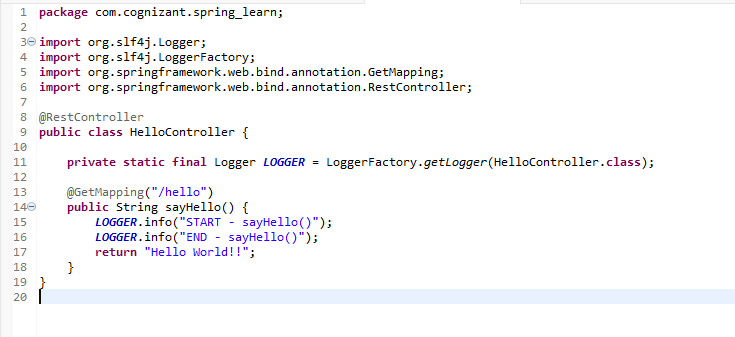
STEP 2: Then I uploaded the unzipped folder to my eclipse workspace.

STEP 3: The code for pom.xml along with the dependencies is

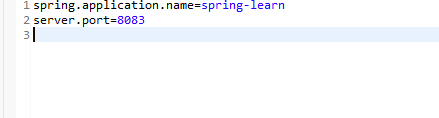




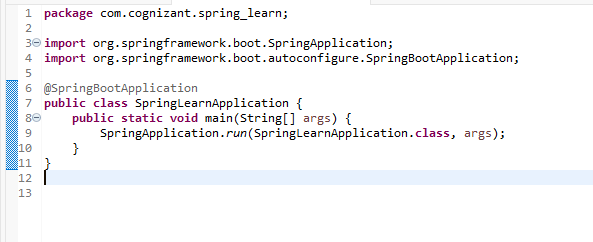
STEP 4: Under src/main/java and within the package com.cognizant.spring\_learn, I created a class named HelloController. The code for it is given below



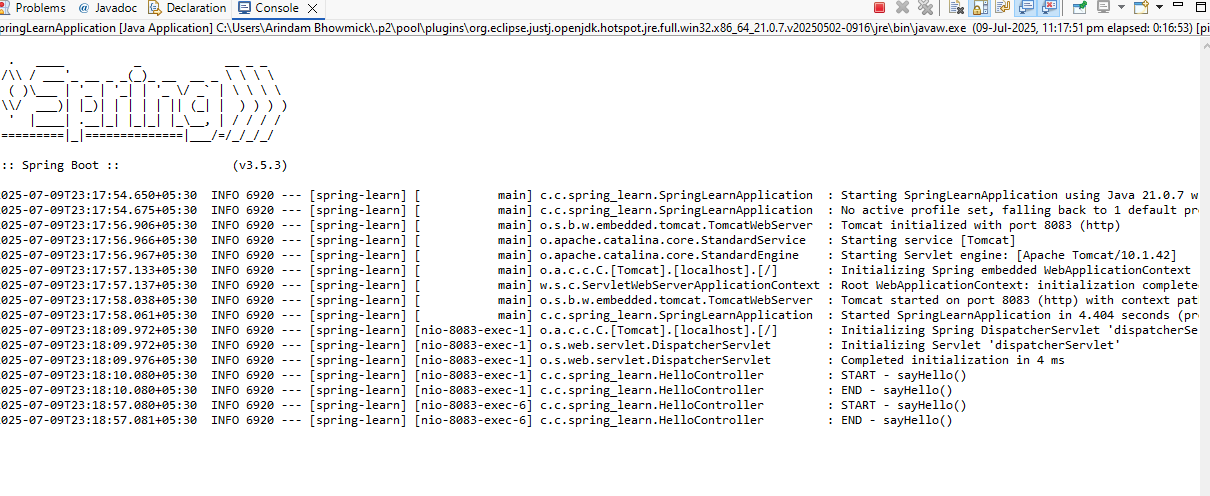
STEP 5: Under src/ main/resources, I updated the application.properties file and added the server port 8083. The code is shown below



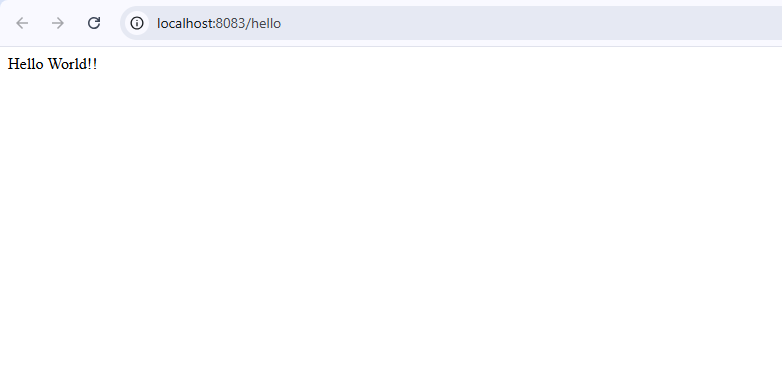
STEP 6: Under src/main/java and with the package com.cognizant.spring\_learn there is the main java class named SpringLearnApplication. The code is given below



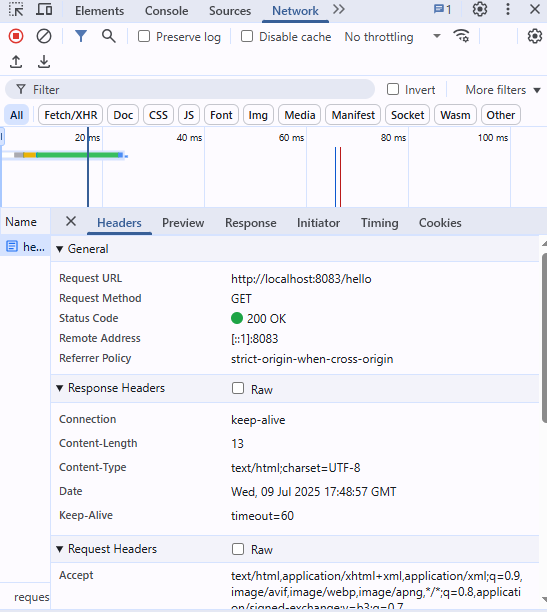
STEP 7: Running the main class produces the output



STEP 8: If in Chrome browser we open port 8083, we can see written “Hello World”



STEP 8: We can view the HTTP headers in Chrome

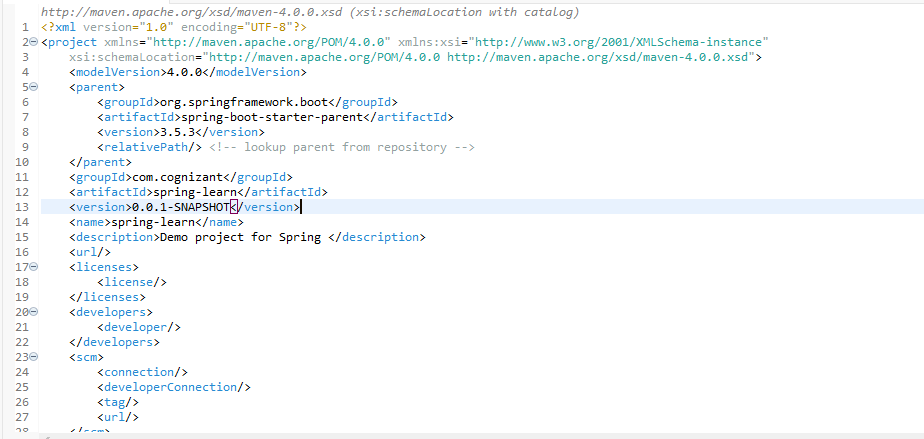


**EXERCISE 4: REST COUNTRY WEB SERVICE**

STEP 1: Firstly I visited to Spring Initializr to create the Maven project, I selected project as Maven, Language as Java, Group as com.cognizant, Artifact as spring-learn. I selected one dependency Spring Web. Then I generated everything as a zip.

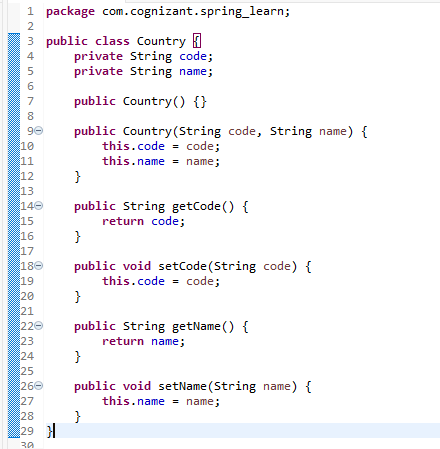
STEP 2: Then I uploaded the unzipped folder to my eclipse workspace.

STEP 3: The code for pom.xml along with the dependencies is

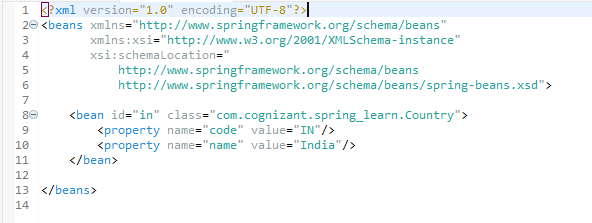




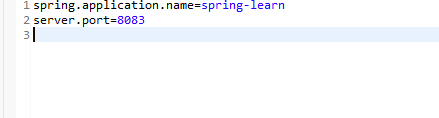
STEP 4: Under src/main/java and within the package com.cognizant.spring\_learn, I created a class named Country. The code for it is given below



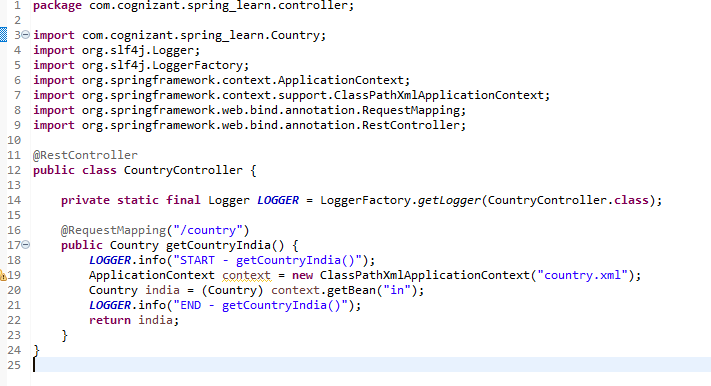
STEP 5: Under src/main/resources, I created a file named country.xml. The code for it is given below



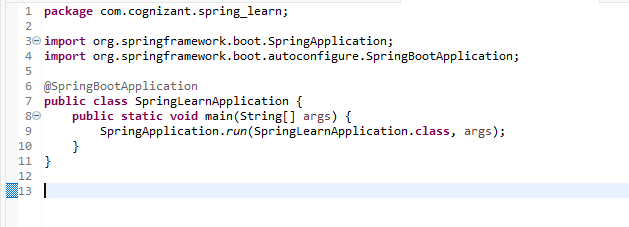
STEP 6: Under src/ main/resources, I updated the application.properties file and added the server port 8083. The code is shown below



STEP 7: Under src/main/java, I created a package named com.cognizant.spring\_learn.controller under which a class named CountryController. The code for it is



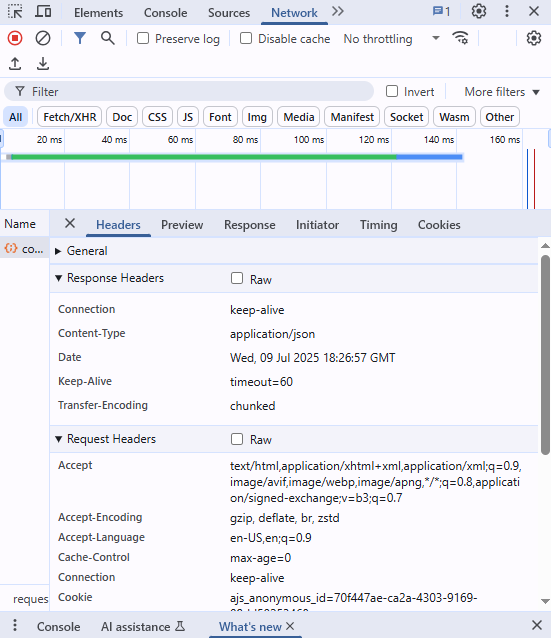
STEP 8: Under src/main/java and with the package com.cognizant.spring\_learn there is the main java class named SpringLearnApplication. The code is given below



STEP 9: If in Chrome browser we open port 8083, we can see the following thing



STEP 10: We can view the HTTP headers in Chrome

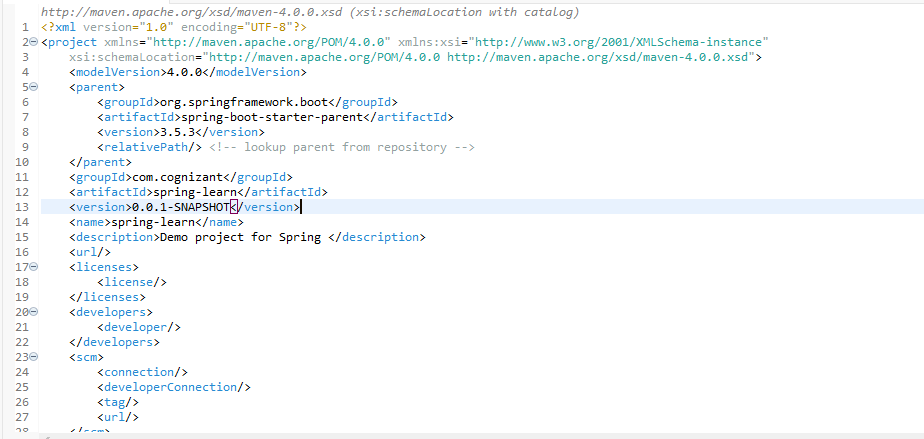


**EXERCISE 5: REST GET COUNTRY BASED ON COUNTRY CODE**

STEP 1: Firstly I visited to Spring Initializr to create the Maven project, I selected project as Maven, Language as Java, Group as com.cognizant, Artifact as spring-learn. I selected one dependency Spring Web. Then I generated everything as a zip.

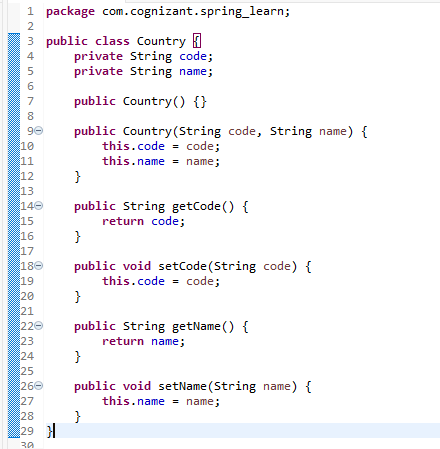
STEP 2: Then I uploaded the unzipped folder to my eclipse workspace.

SEP 3: The code for pom.xml along with the dependencies is





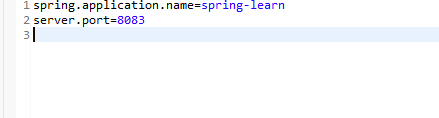
STEP 4: Under src/main/java and within the package com.cognizant.spring\_learn, I created a class named Country. The code for it is given below



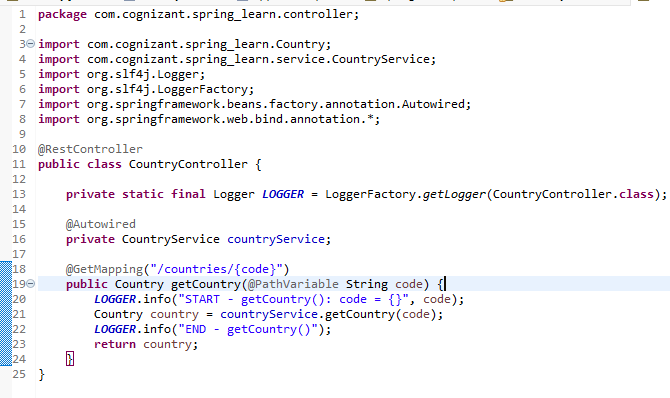
STEP 5: Under src/main/resources, I created a file named country.xml. The code for it is given below



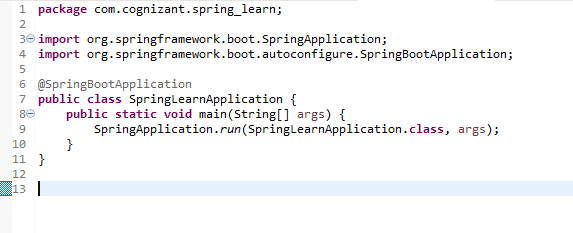
STEP 6: Under src/ main/resources, I updated the application.properties file and added the server port 8083. The code is shown below



STEP 7: Under src/main/java, I created a package named com.cognizant.spring\_learn.controller under which a class named CountryController. The code for it is



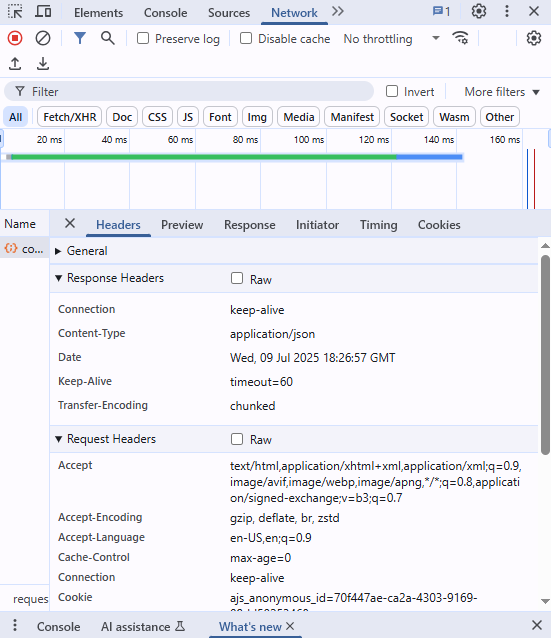
STEP 8: Under src/main/java and with the package com.cognizant.spring\_learn there is the main java class named SpringLearnApplication. The code is given below



STEP 9: If in Chrome browser we open port 8083, we can see the following thing



STEP 10: We can view the HTTP headers in Chrome

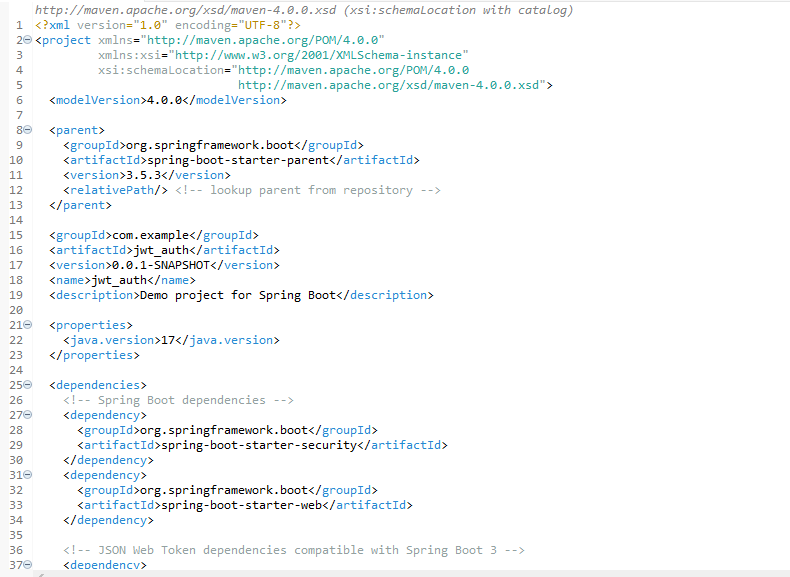


**EXERCISE 6: CREATE AUTHENTICATION SERVICE THAT RETURNS JWT**

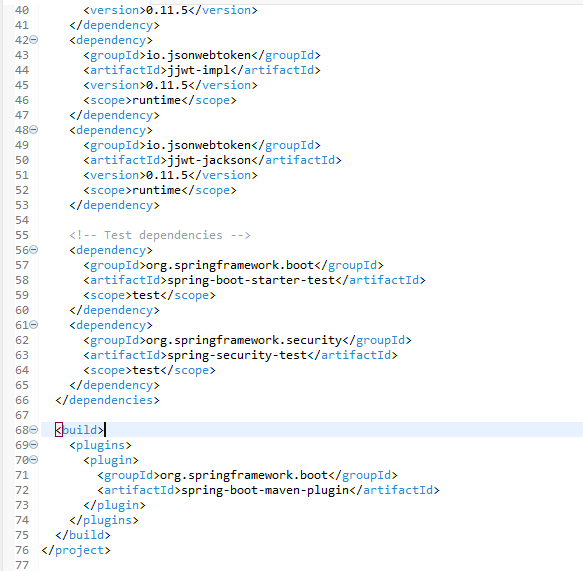
STEP 1: Firstly I visited to Spring Initializr to create the Maven project, I selected project as Maven, Language as Java, Group as com.example, Artifact as jwt-auth. I selected two dependencies as stated Spring Web and Spring Security. Then I generated everything as a zip.

STEP 2: Then I uploaded the unzipped folder to my eclipse workspace.

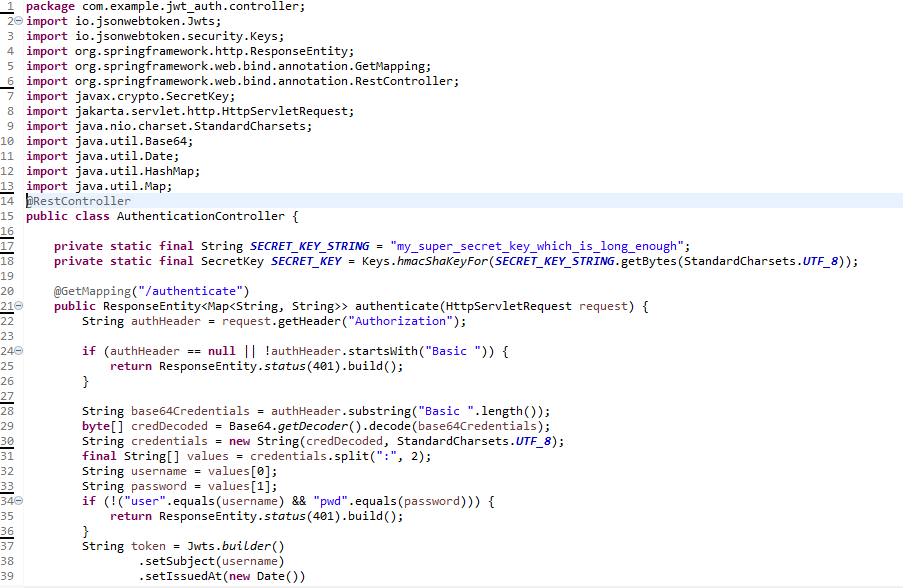
STEP 3: Then I added jjwt dependencies in pom.xml. the code for it is mentioned below



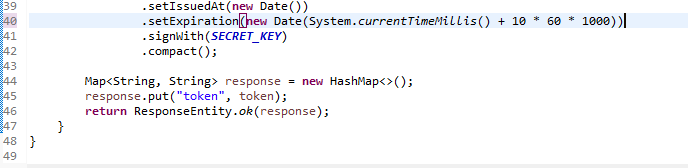
Continued….



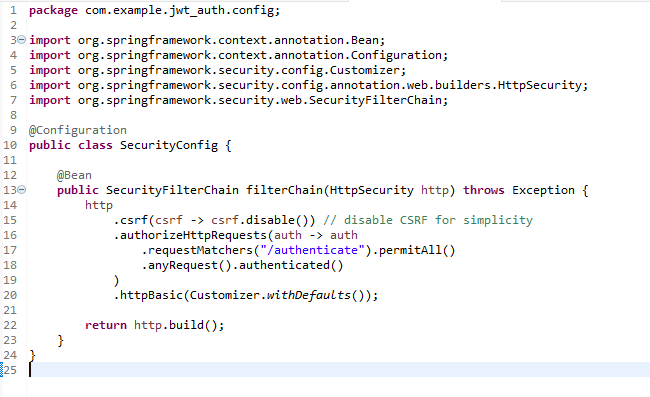
STEP 4: Under src/main/java, I created a package named com.example.jwt\_auth.controller under which a class named AuthenticationController. The code for it is given



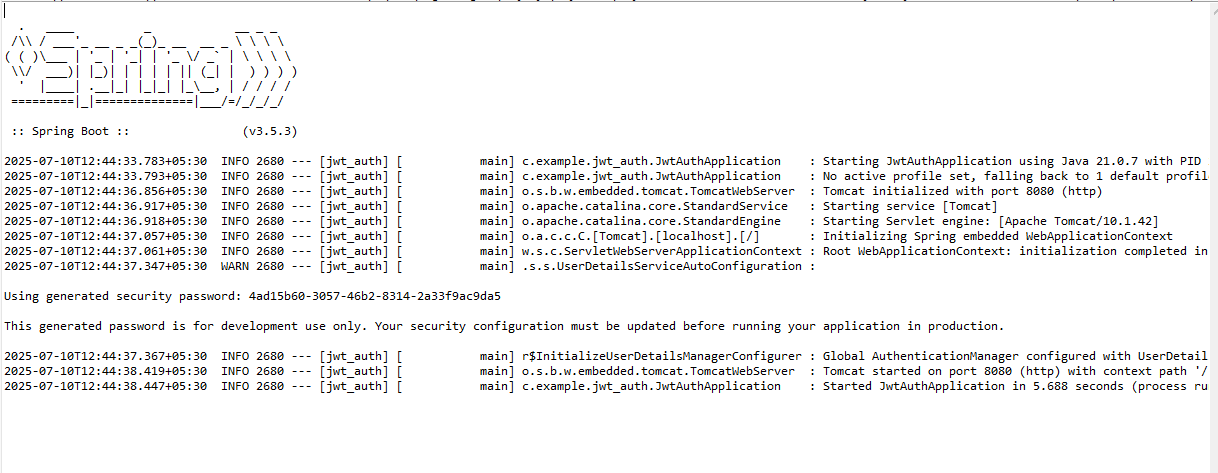
Continued….



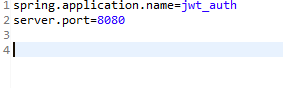
STEP 5: Under src/main/java, I created a package named com.example.jwt\_auth.config under which a class named SecurityConfig. The code for it is given



STEP 6: Running the main java application produces the output.



STEP 7: I also updated the application.properties and added server port 8080



STEP 8: If we run curl -u user:pwd localhost 8080 in out terminal we will get like this

