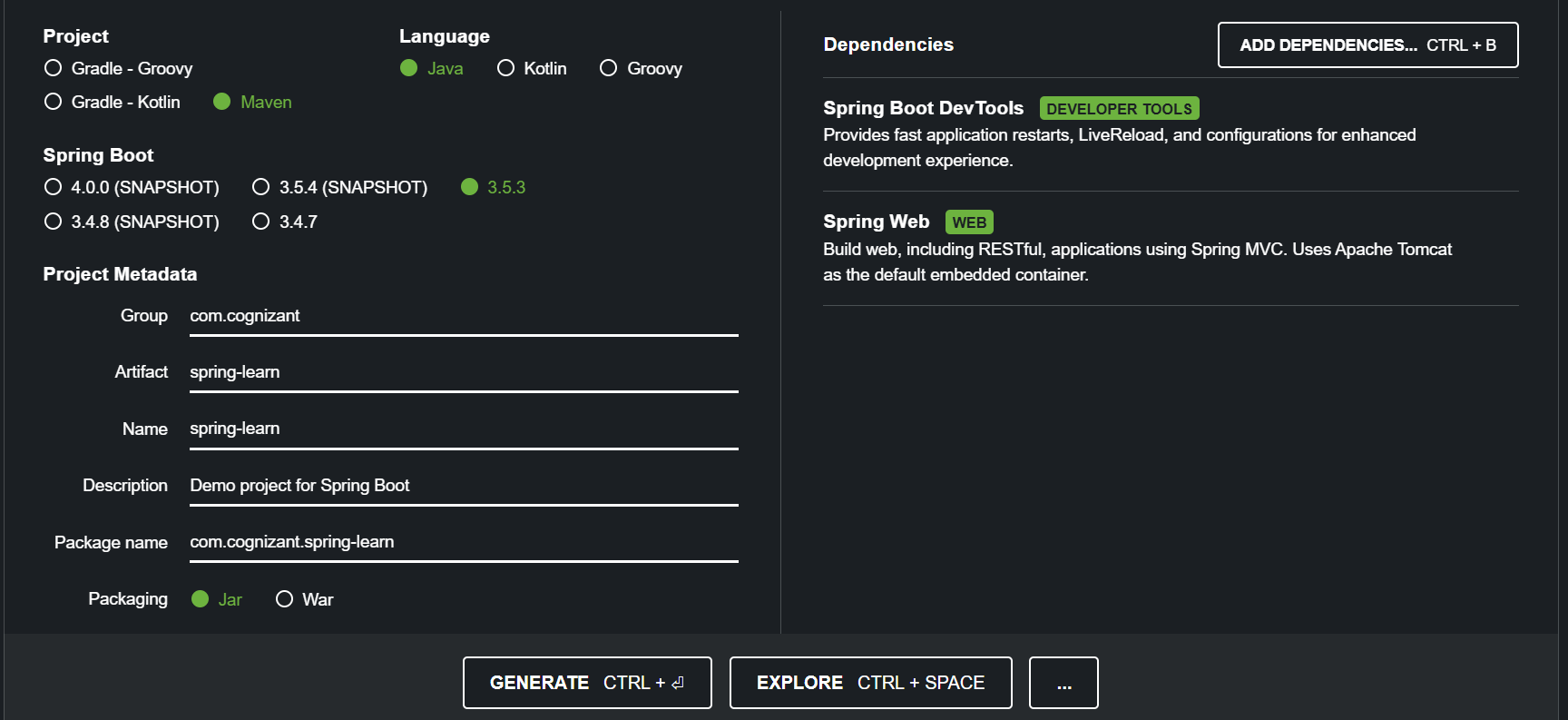
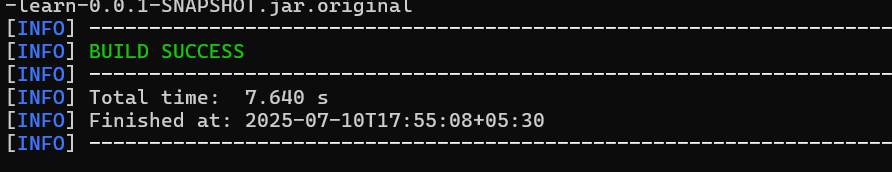
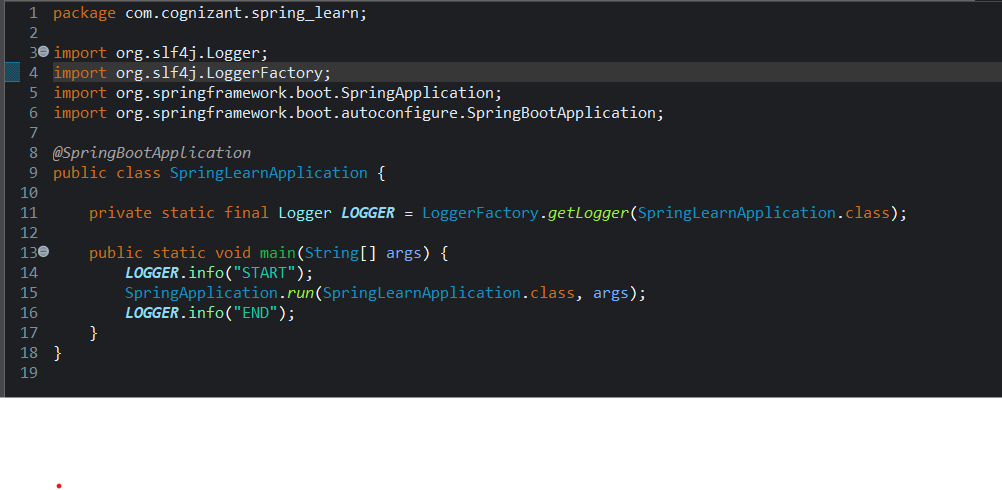
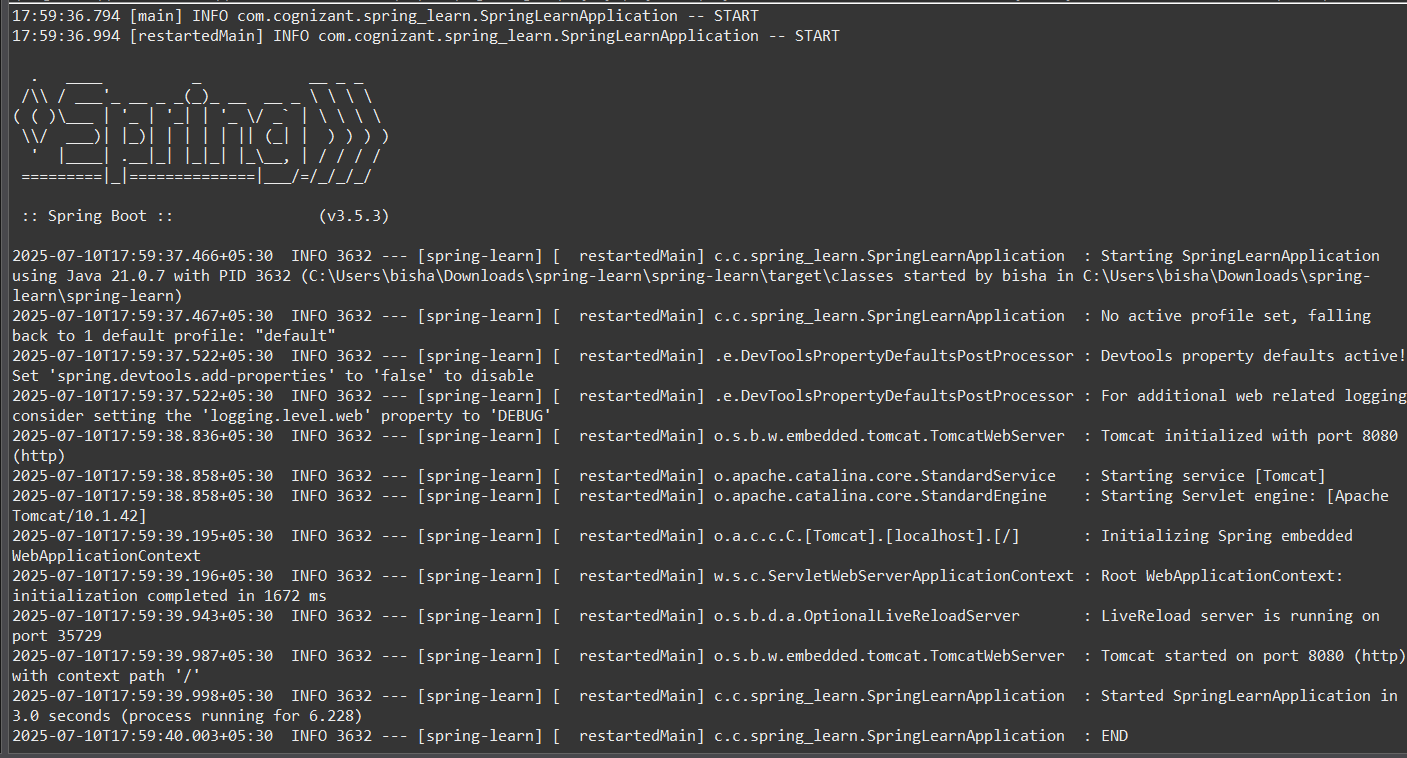
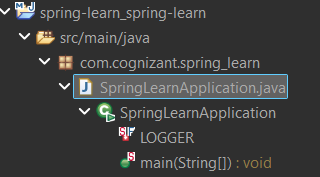
**Hands on 1**

**Create a Spring Web Project using Maven**   
  
Follow steps below to create a project: 

1. Go to <https://start.spring.io/>
2. Change Group as “com.cognizant”
3. Change Artifact Id as “spring-learn”
4. Select Spring Boot DevTools and Spring Web
5. Create and download the project as zip
6. Extract the zip in root folder to Eclipse Workspace
7. 
8. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
9. 
10. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
11. Include logs to verify if main() method of SpringLearnApplication.
12. 
13. Run the SpringLearnApplication class.
14. 

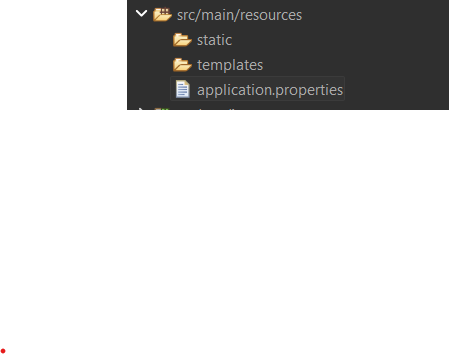
SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code



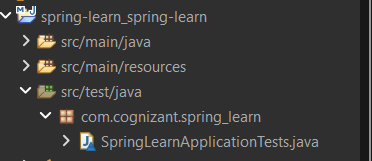
1. src/main/resources - Folder for application configuration

This folder is used for Spring Boot configuration files. The application.properties file is where we set up server port, logging, DB config, etc

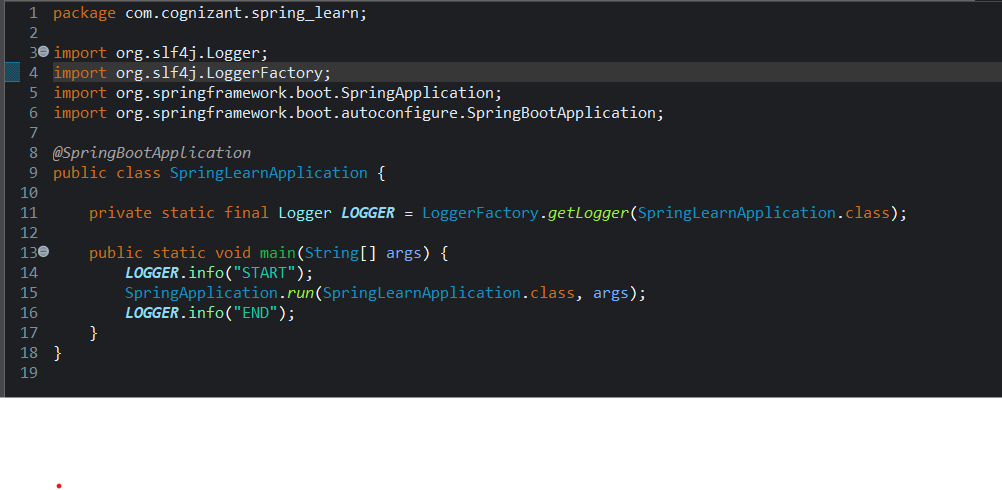


1. src/test/java - Folder with code for testing the application

This folder is for writing test cases to validate the business logic. Spring Boot uses JUnit by default for testing.



1. SpringLearnApplication.java - Walkthrough the main() method.

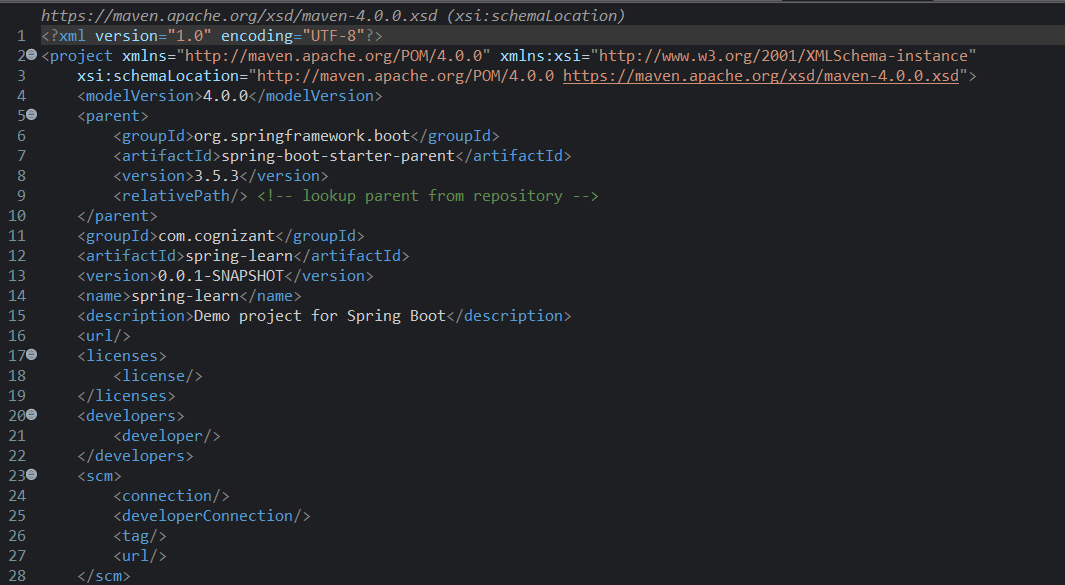


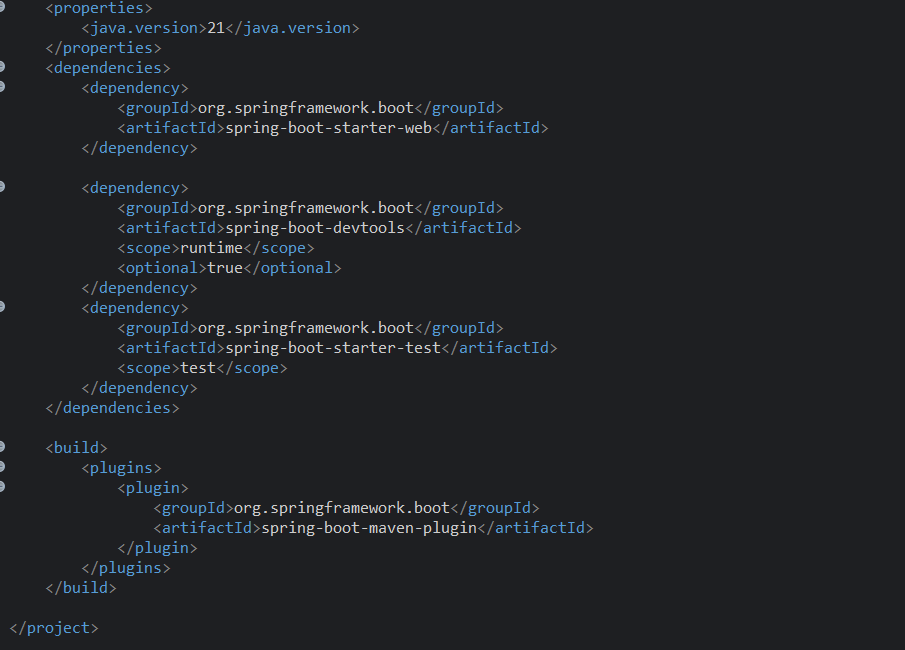
1. Purpose of @SpringBootApplication annotation

This annotation is used for:

@Configuration : Marks class as a configuration source. @EnableAutoConfiguration : Enables Spring Boot’s auto configuration. @ComponentScan : Scans the package for components, services, etc.

1. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.





**Hands on 4**

**Spring Core – Load Country from Spring Configuration XML**   
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country. 

|  |  |
| --- | --- |
| **Code** | **Name** |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.  
  
Steps to implement

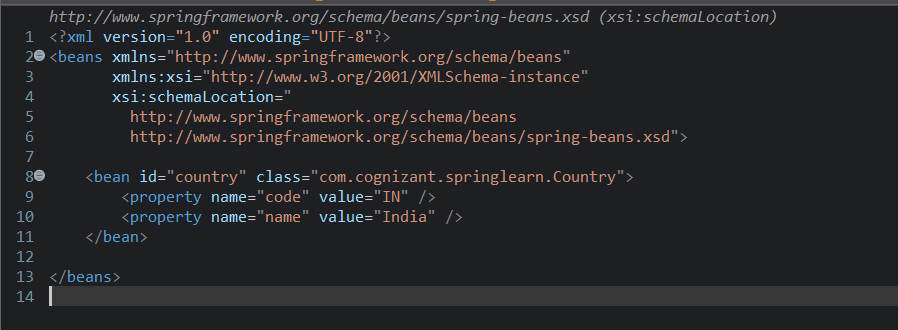
* Pick any one of your choice country to configure in Spring XML configuration named country.xml.
* Create a bean tag in spring configuration for country and set the property and values

    <bean id="country" class="com.cognizant.springlearn.Country">

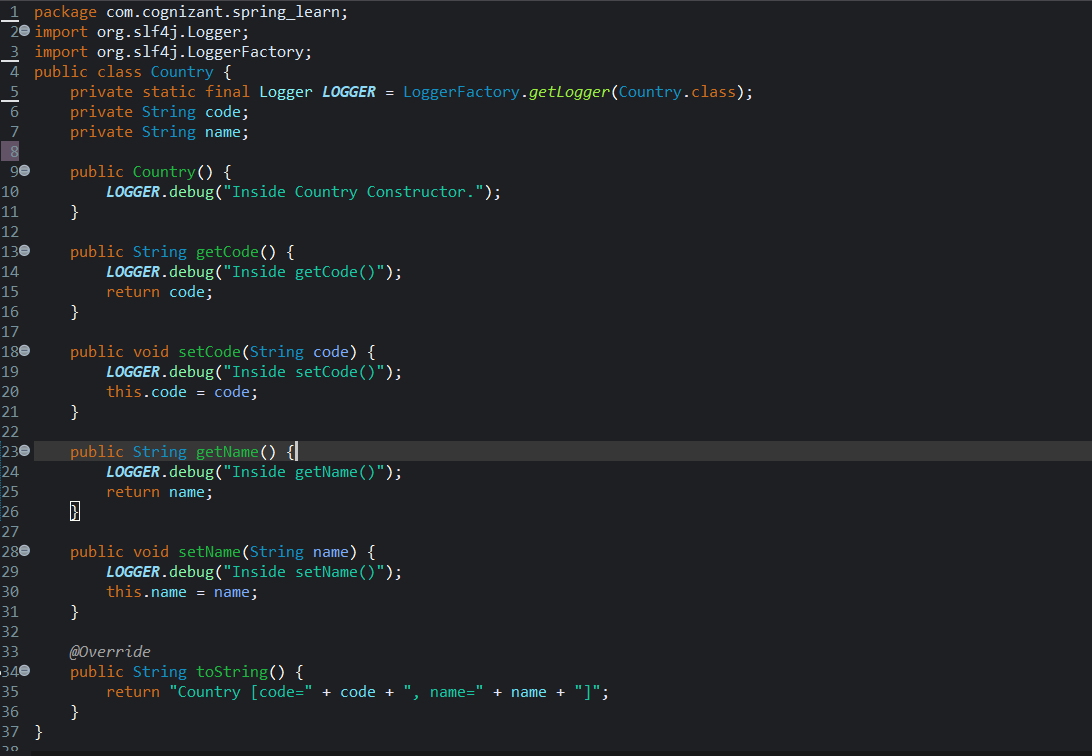
        <property name="code" value="IN" />

        <property name="name" value="India" />

    </bean>



* Create Country class with following aspects:
  + Instance variables for code and name
  + Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”
  + Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.
  + Generate toString() method



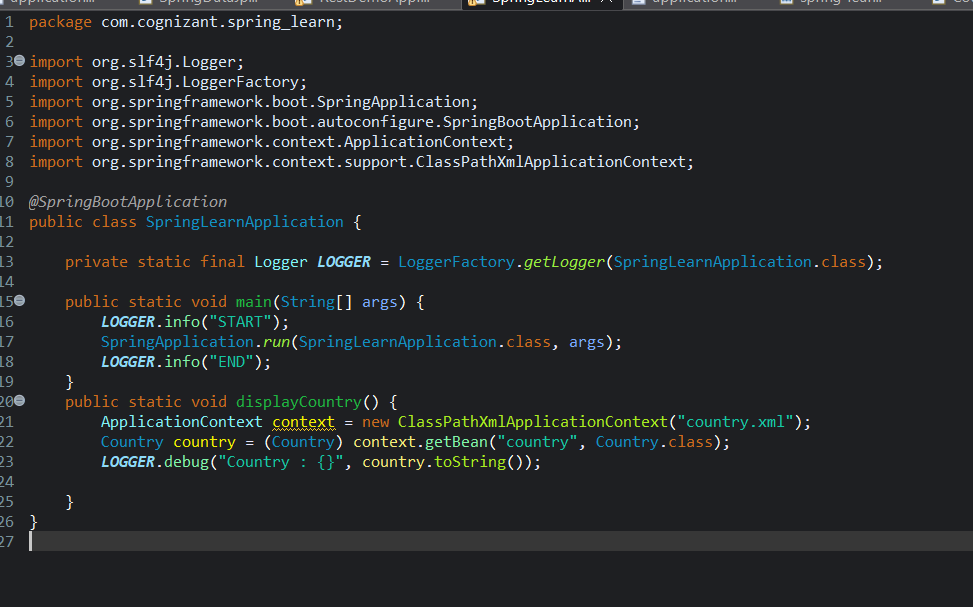
* Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = (Country) context.getBean("country", Country.class);

LOGGER.debug("Country : {}", country.toString());

* Invoke displayCountry() method in main() method of SpringLearnApplication.java.
* Execute main() method and check the logs to find out which constructors and methods were invoked.



SME to provide more detailing about the following aspects:

* bean tag, id attribute, class attribute, property tag, name attribute, value attribute

<bean>: Declares a Spring bean.(id:) Unique name to reference this bean ("country"). (class:) Fully qualified Java class to instantiate. <property>: Used to set values of fields in the bean via setter injection. (name:) Matches Java property name (e.g., code, name).(value:) Value to assign (e.g., IN, India).

* ApplicationContext, ClassPathXmlApplicationContext

ApplicationContext-it is the core spring container interface to access beans

ClassPathXmlApplicationContext-it loads bean definations from Xml file on the classpath

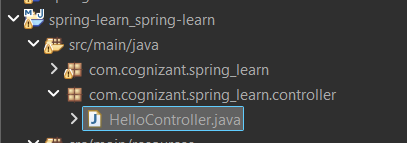
* What exactly happens when context.getBean() is invoked

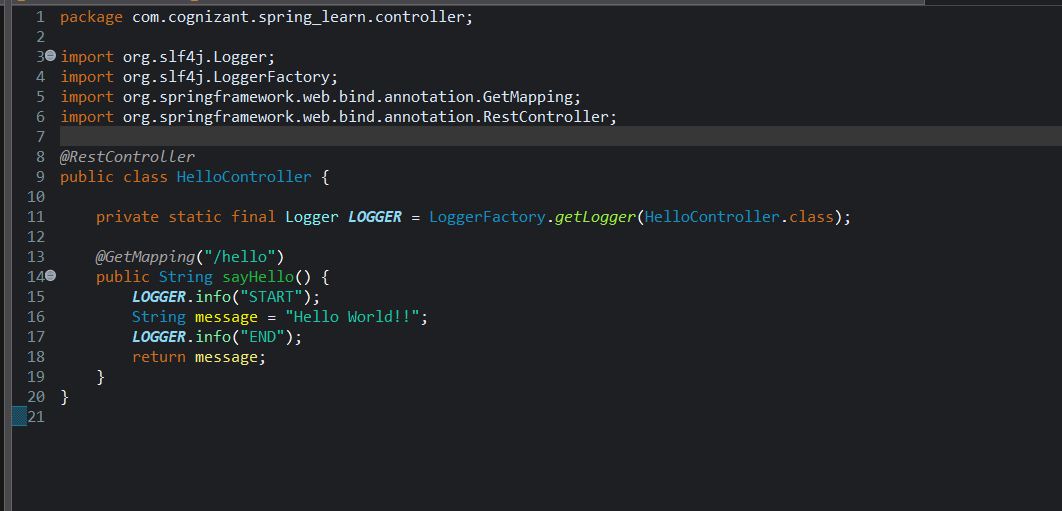
When context.getBean() in invoked it reads the bean definition from country.xml, sets its properties via setters,return the bean instance cast to country

Assesment-3

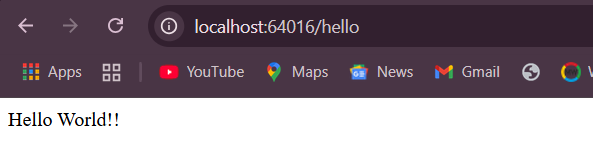
2.spring-rest-handson

**Hello World RESTful Web Service**   
  
Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:  
  
**Method:** GET  
**URL:** /hello  
**Controller:** com.cognizant.spring-learn.controller.HelloController

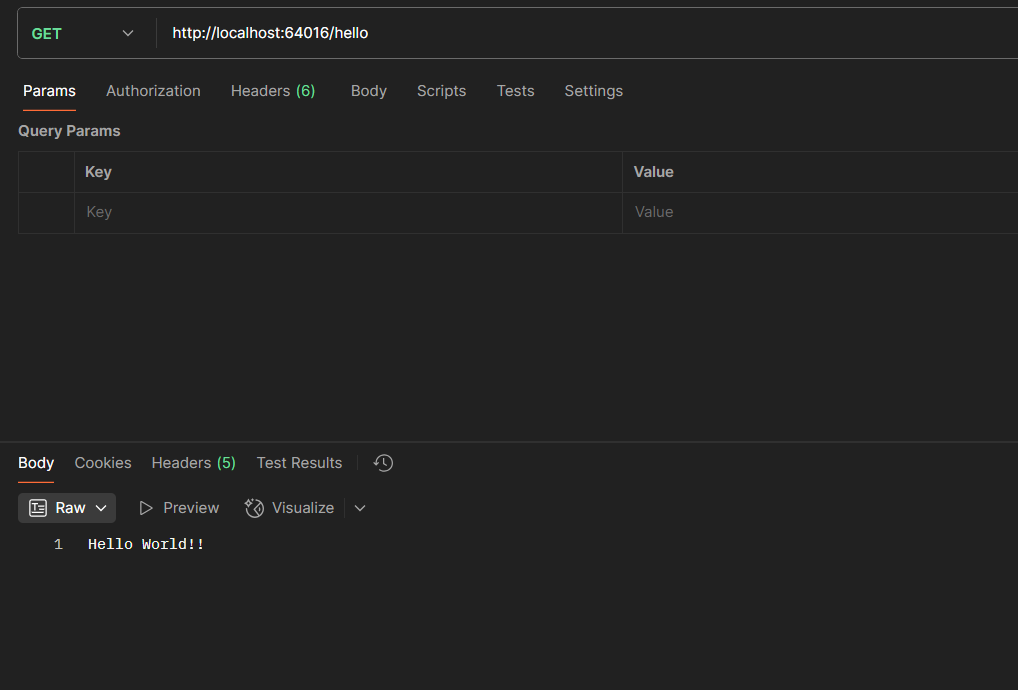
  
**Method Signature:** public String sayHello()

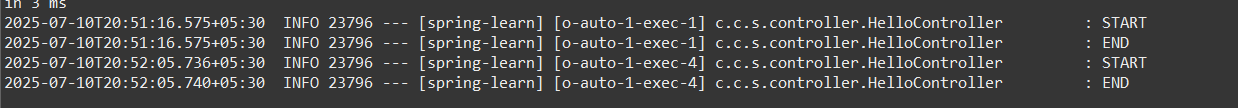
  
**Method Implementation:** return hard coded string "Hello World!!"  
**Sample Request**: http://localhost:8083/hello  
**Sample Response:** Hello World!!

IN BROWSER:

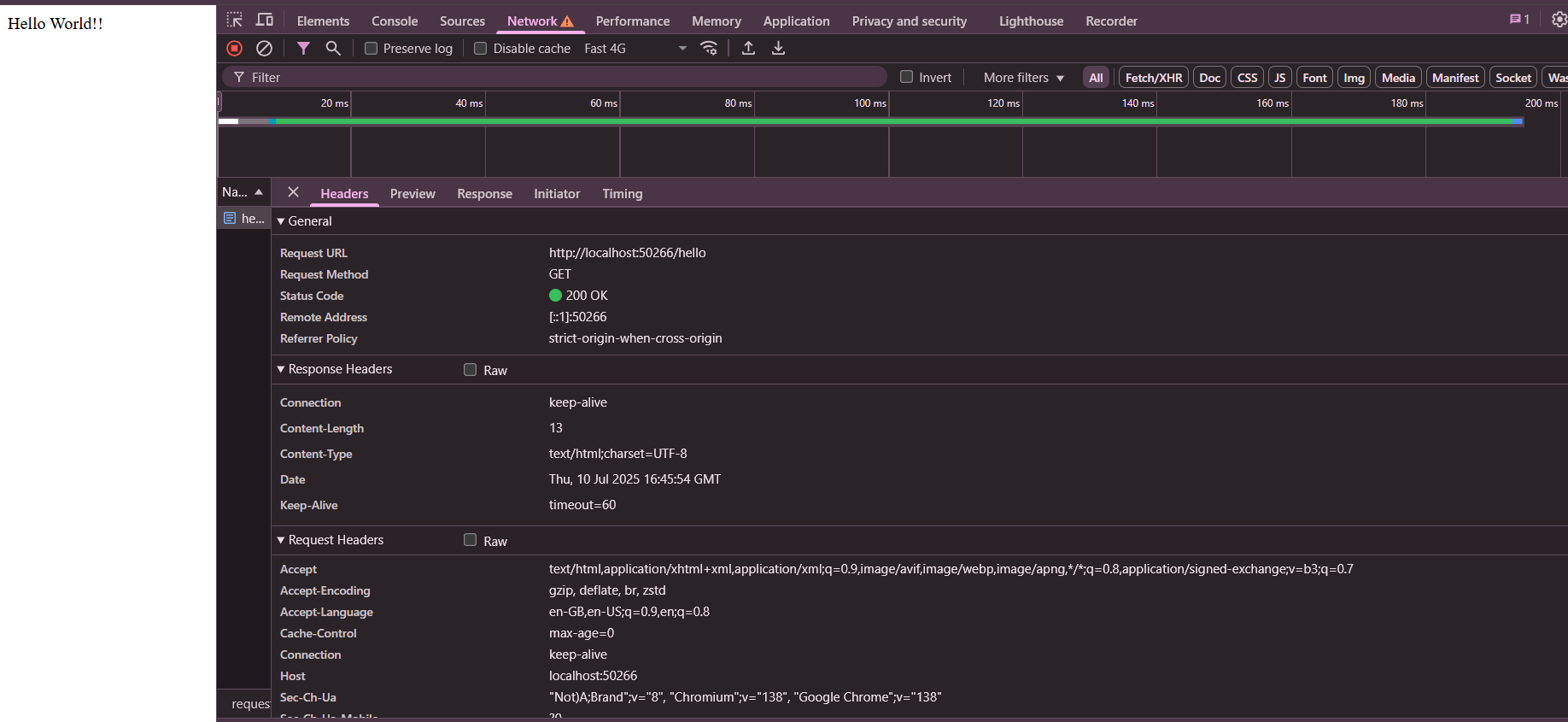


IN POSTMAN:

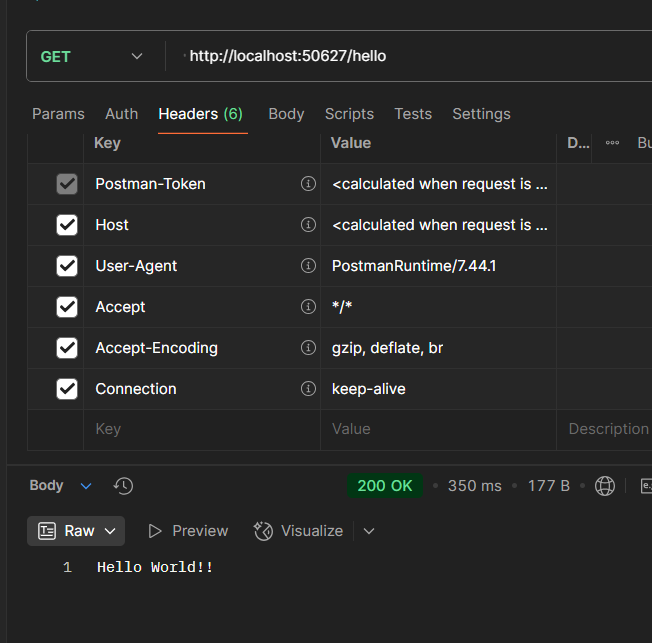
  
  
**IMPORTANT NOTE**: Don't forget to include start and end log in the sayHello() method.

  
  
Try the URL http://localhost:8083/hello in both chrome browser and postman.  
  
SME to explain the following aspects:

* In network tab of developer tools show the HTTP header details received



* In postman click on "Headers" tab to view the HTTP header details received

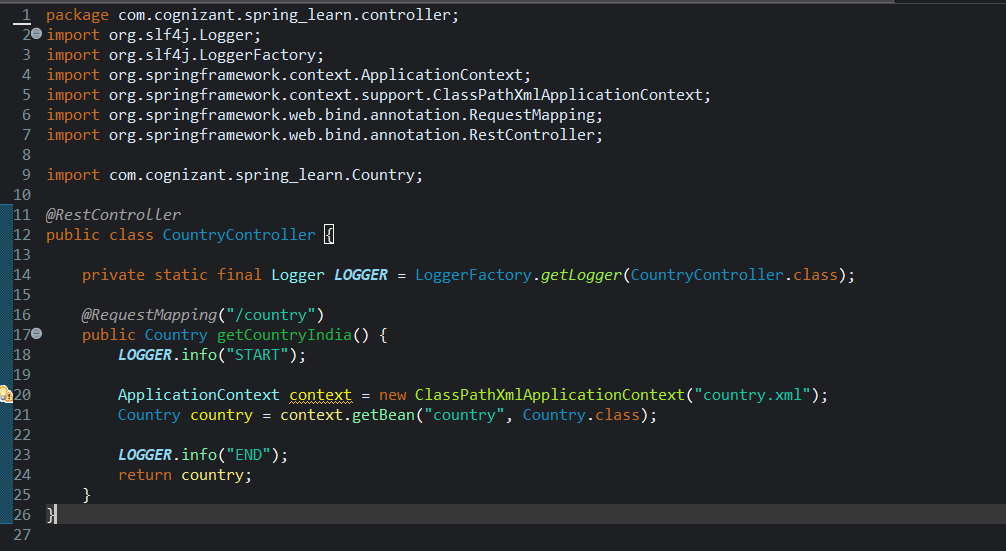


**Assesment-4**

2.spring-rest-handson

**REST - Country Web Service**   
  
Write a REST service that returns India country details in the earlier created spring learn application.  
  
**URL**: /country  
**Controller**: com.cognizant.spring-learn.controller.CountryController

  
**Method Annotation**: @RequestMapping  
**Method Name**: getCountryIndia()  
**Method Implementation**: Load India bean from spring xml configuration and return

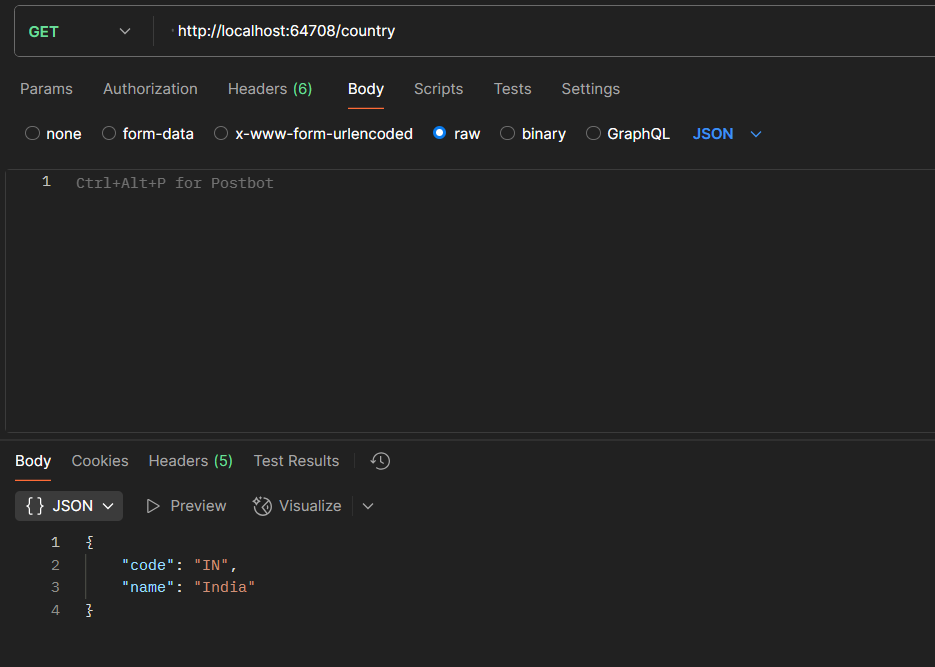
  
**Sample Request**: http://localhost:8083/country  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}



SME to explain the following aspects:

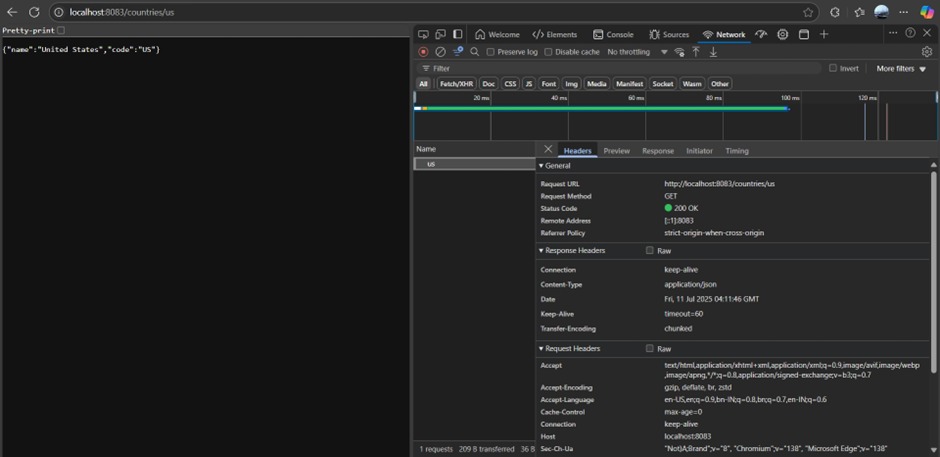
* What happens in the controller method?

getCountryIndia() is mapped to /country using @RequestMapping,it loads the bean country from country.xml file using ApplicationContext.

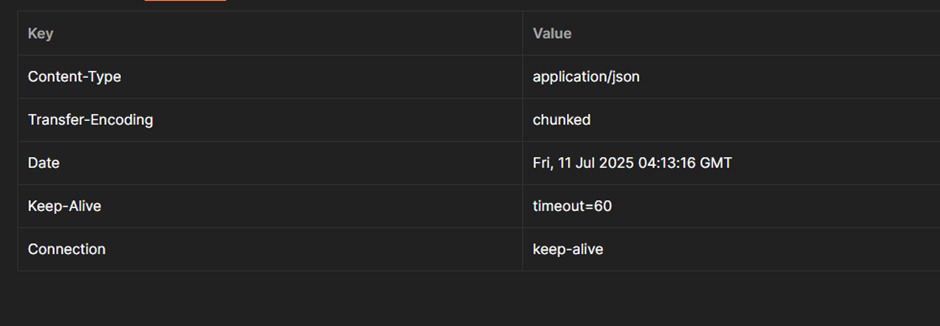
* How the bean is converted into JSON reponse?

SpringBoot uses JSON automatically,in which @RestController implies @ResponseBody where spring automatically converts any POJO into JSON.

* In network tab of developer tools show the HTTP header details received

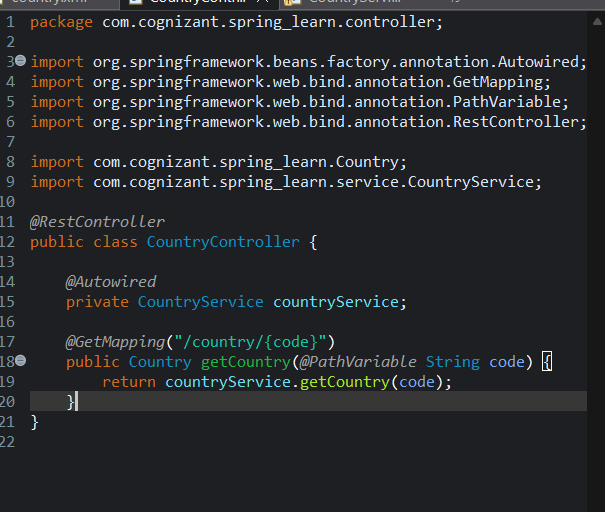


* In postman click on "Headers" tab to view the HTTP header details received

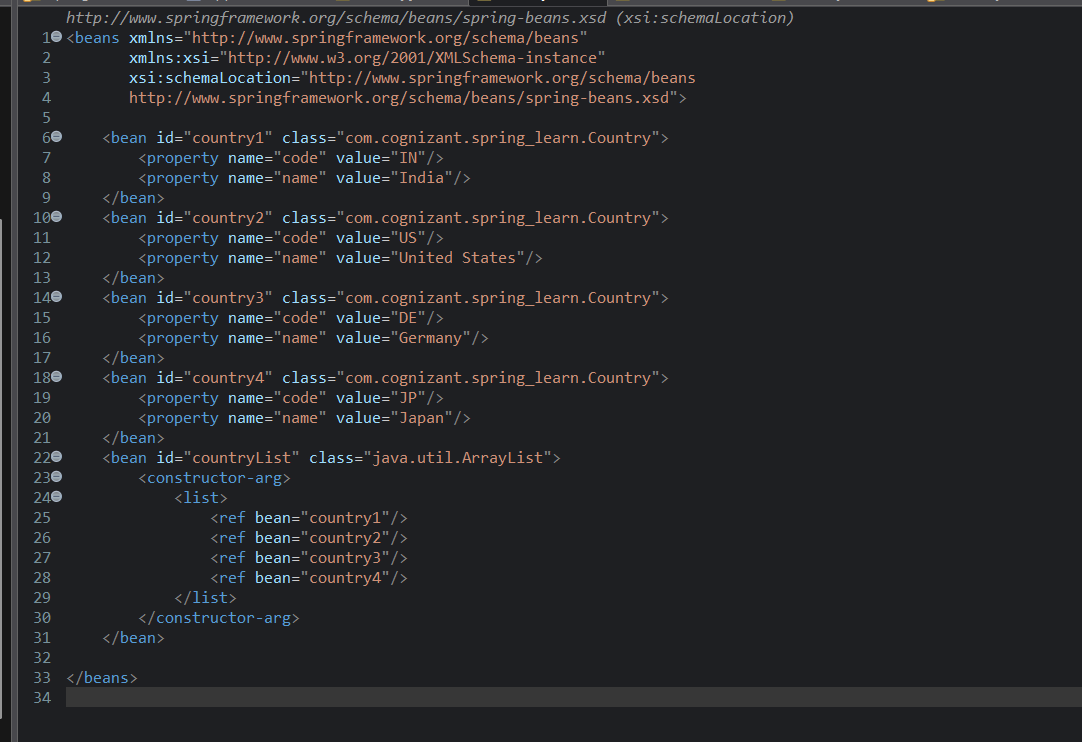


**Assesment-5**

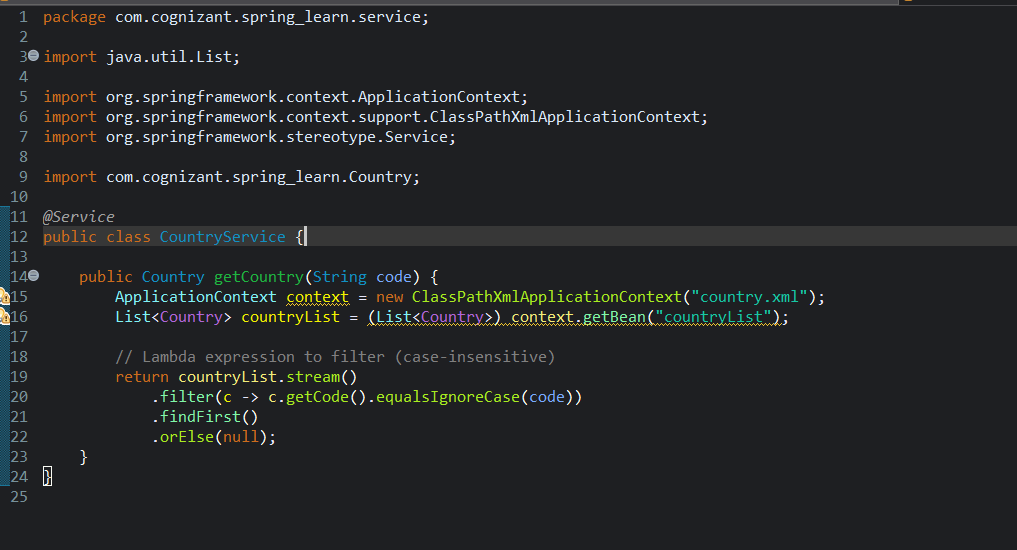
**REST - Get country based on country code**   
  
Write a REST service that returns a specific country based on country code. The country code should be case insensitive.  
  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation:** @GetMapping("/countries/{code}")  
**Method Name**: getCountry(String code)

  
**Method Implemetation**: Invoke countryService.getCountry(code)   
**Service Method:**com.cognizant.spring-learn.service.CountryService.getCountry(String code)  
  
**Service Method Implementation**:

* Get the country code using @PathVariable
* Get country list from country.xml



* Iterate through the country list
* Make a case insensitive matching of country code and return the country.
* Lambda expression can also be used instead of iterating the country list



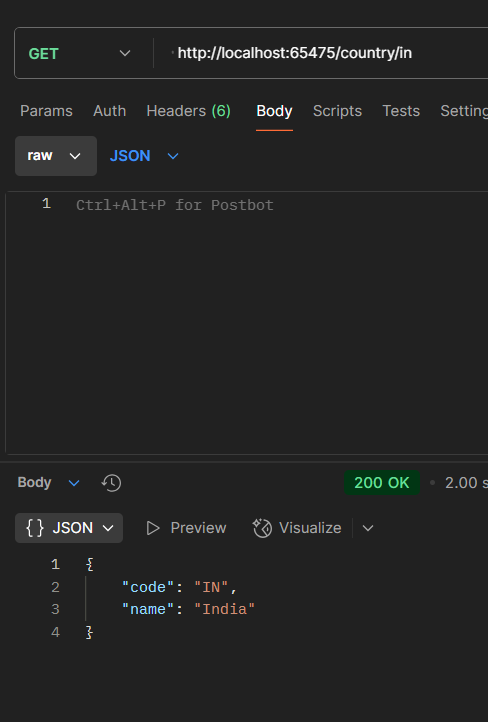
**Sample Request**: http://localhost:8083/country/in  
  
**Sample Response**:

{

  "code": "IN",

  "name": "India"

}



Assessment-6

**Create authentication service that returns JWT**   
  
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

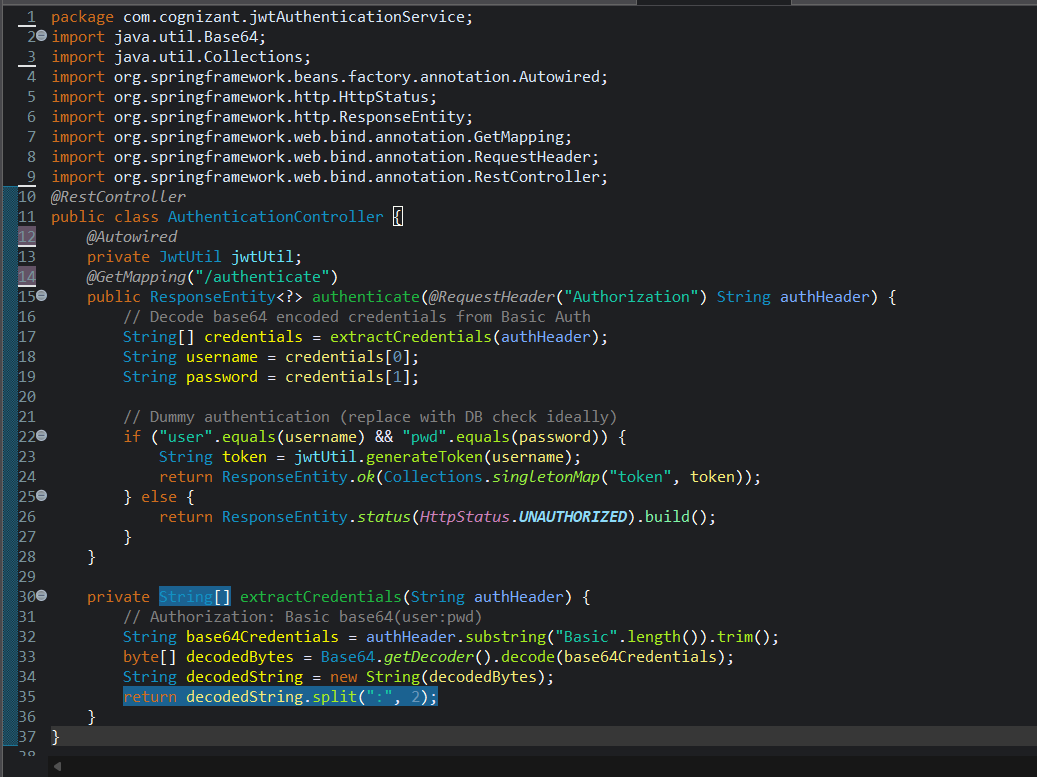
curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

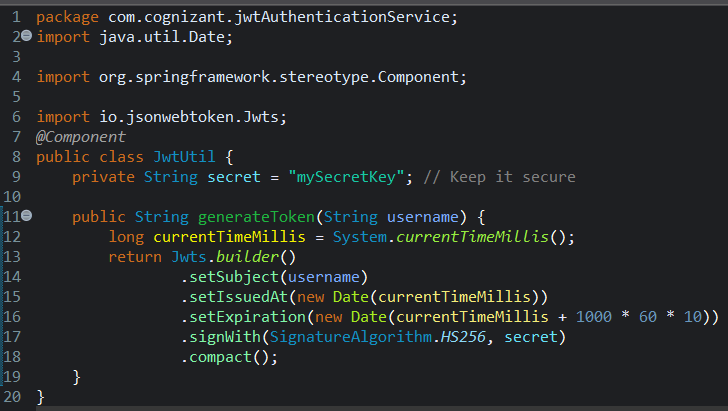
{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This can be incorporated as three major steps:

* Create authentication controller and configure it in SecurityConfig



* Read Authorization header and decode the username and password



* Generate token based on the user retrieved in the previous step

