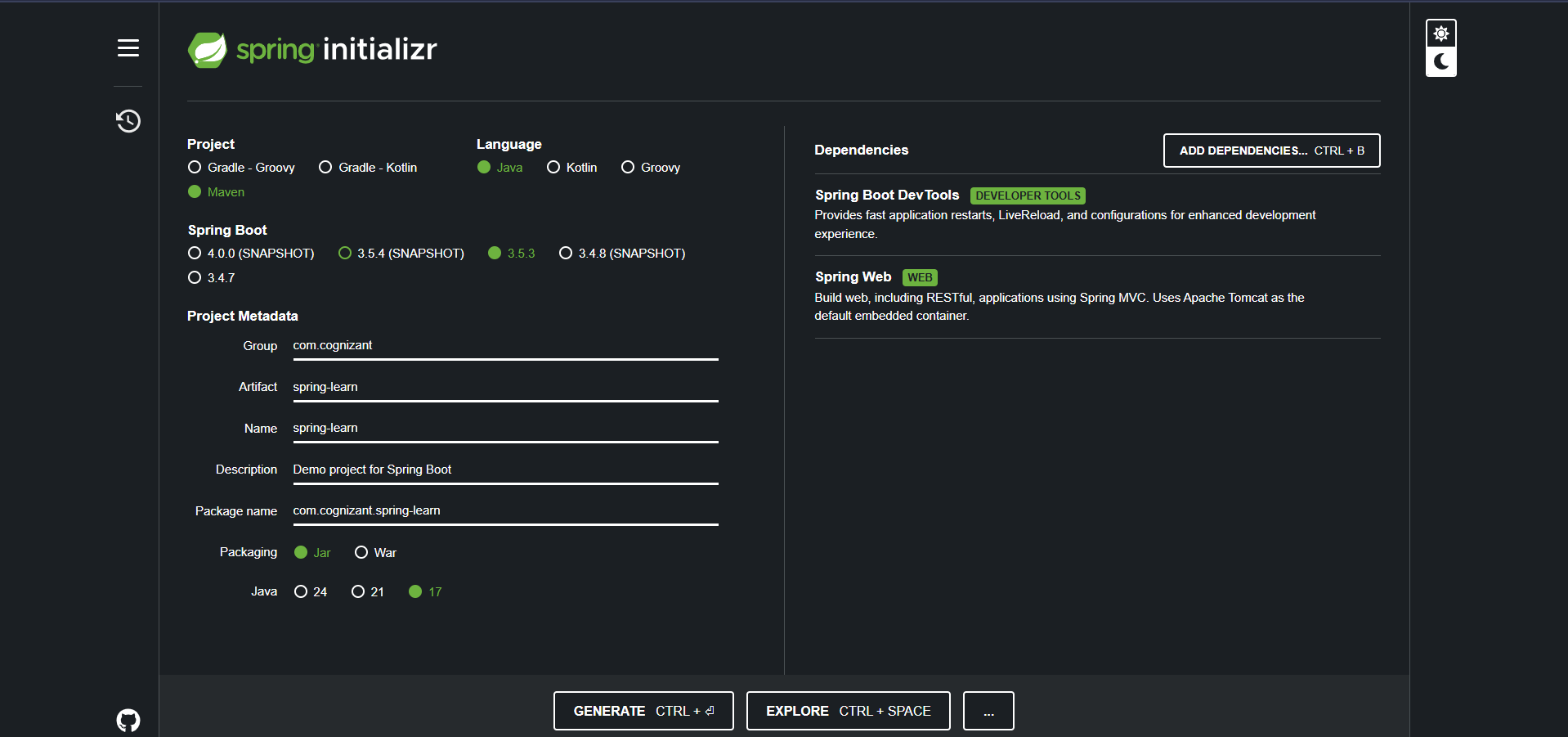
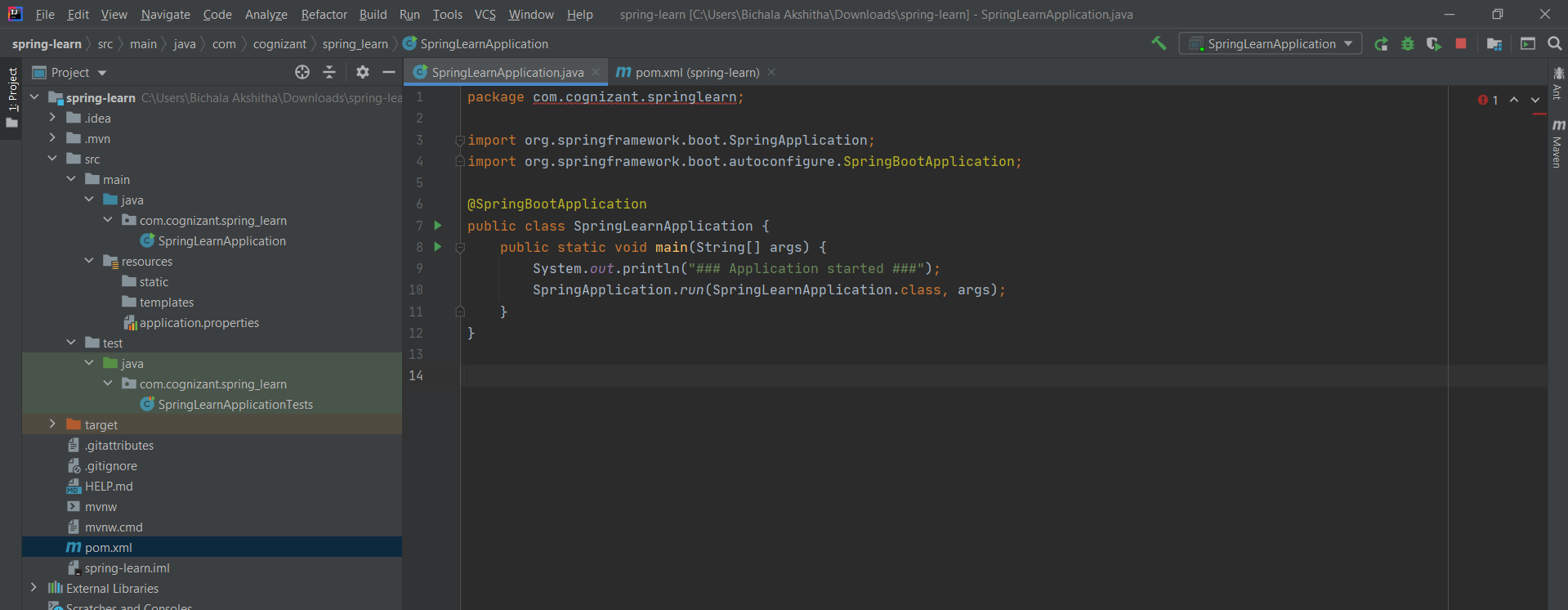
**WEEK-4**

**Hands on 1**

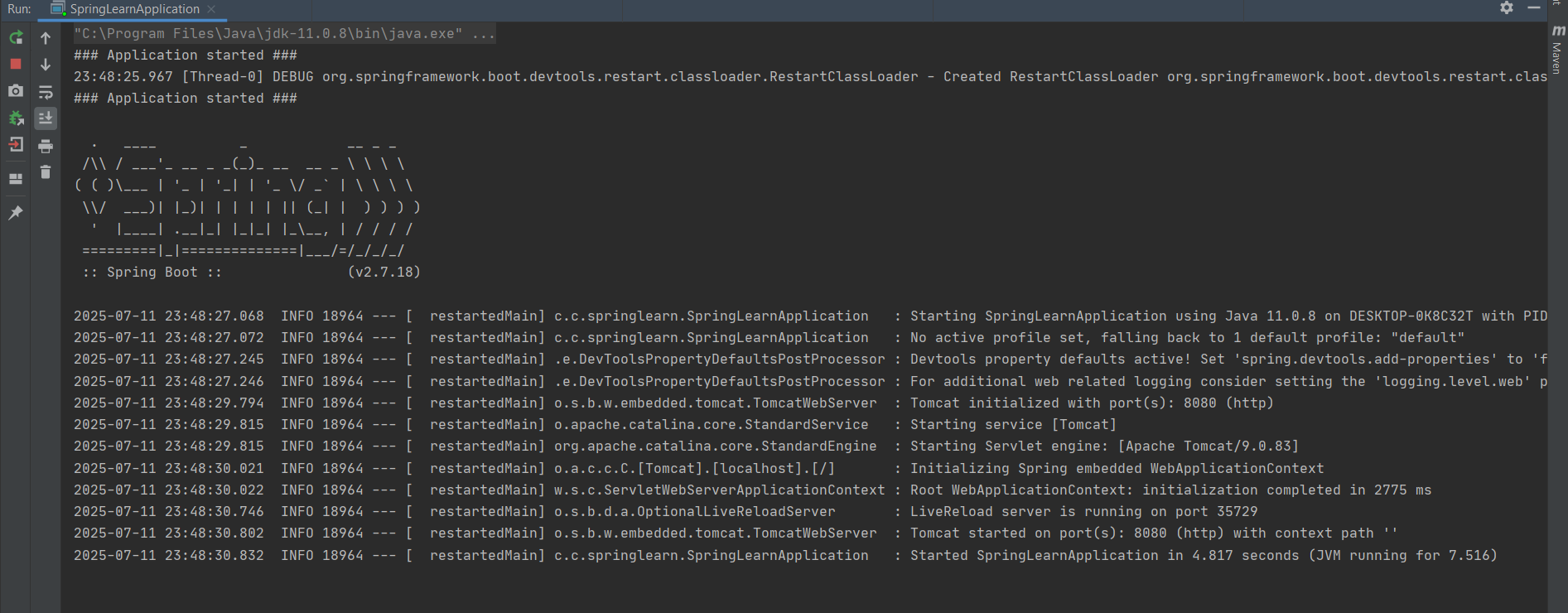
**Create a Spring Web Project using Maven**   
  
Follow steps below to create a project: 

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. 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
8. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
9. Include logs to verify if main() method of SpringLearnApplication.
10. Run the SpringLearnApplication class.



****

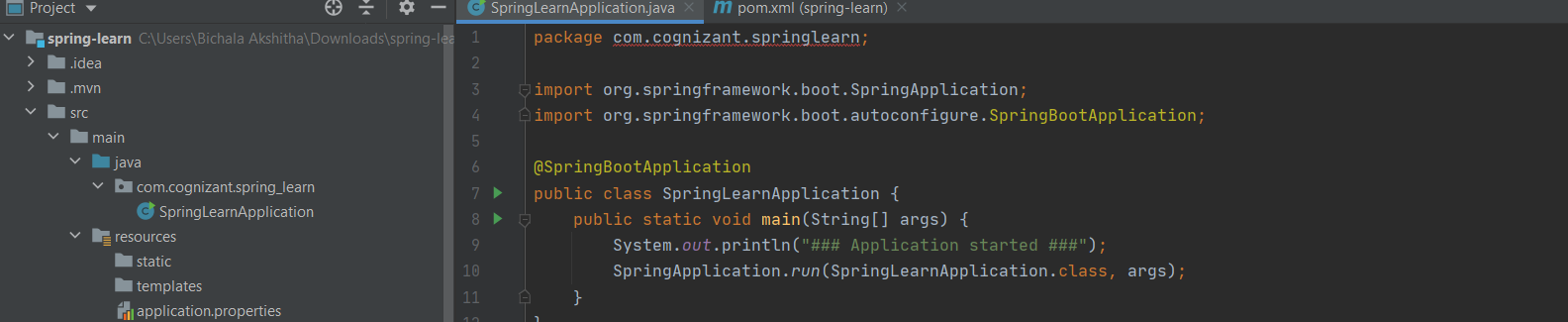
**Output:**

****

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

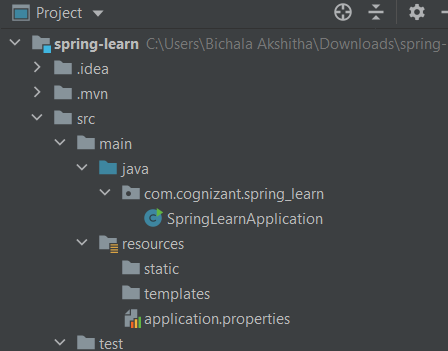
**1. src/main/java - Folder with application code**

This folder holds the application code. In this case, the main class SpringLearnApplication.java is the entry point. Any controllers or services we write will also be under this package structure



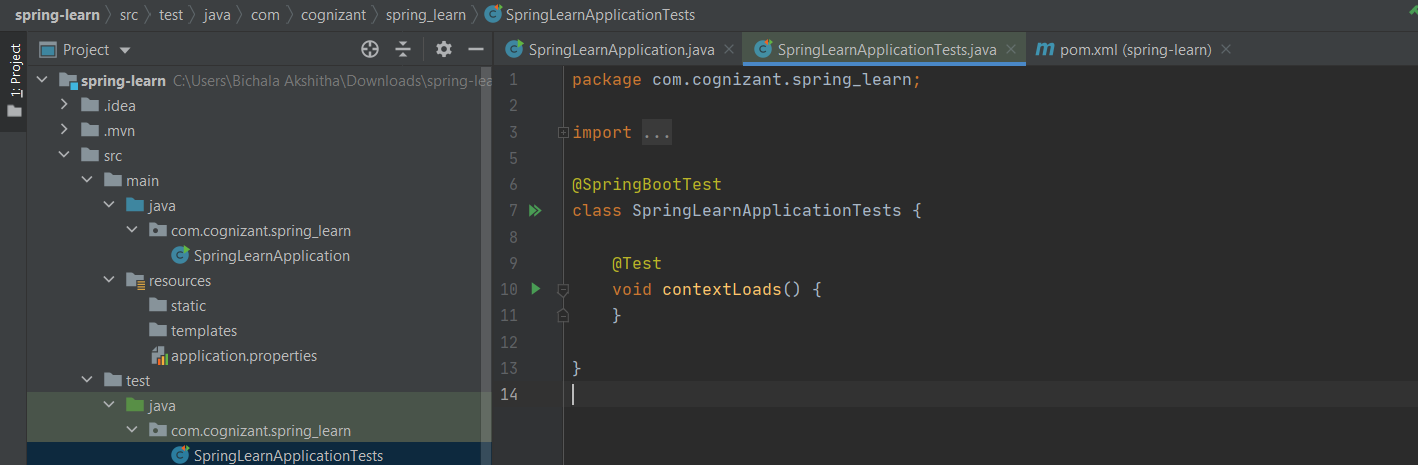
**2. src/main/resources - Folder for application configuration**

This folder contains application-level configurations like properties, logging, and message files. Currently, it’s empty, but by default, Spring looks for application.properties here.

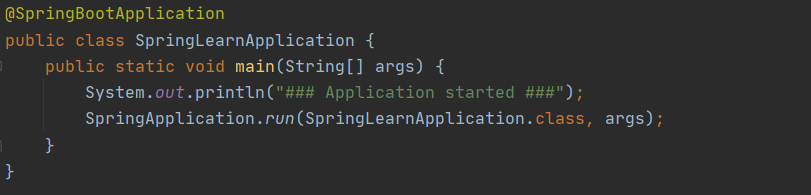
****

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

This folder is meant for test cases. Spring Boot generates a basic test class, and we can add JUnit tests for our controllers, services, and repositories here.

****

**4. SpringLearnApplication.java - Walkthrough the main() method.**

This is the entry point of the Spring Boot app. The main() method uses SpringApplication.run() to bootstrap the entire application. The @SpringBootApplication annotation enables auto-configuration, component scanning, and makes it a Spring Boot app.****

**5. Purpose of @SpringBootApplication annotation**

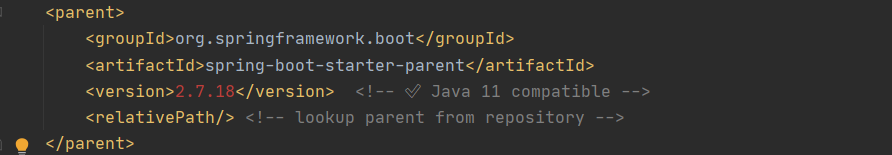
@SpringBootApplication combines three key Spring annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan. It sets up the context, scans packages, and auto-configures beans.

**6. pom.xml**

**1. Walkthrough all the configuration defined in XML file**

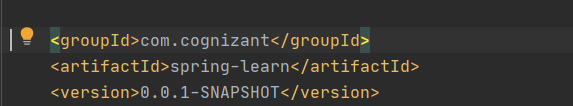
**a. Parent Definition:**

This defines the base Spring Boot version and imports default settings.

****

**b. Project Info:**

This uniquely identifies the project in Maven

****

**c. Java Version:**This ensures the project is compiled using Java 11.

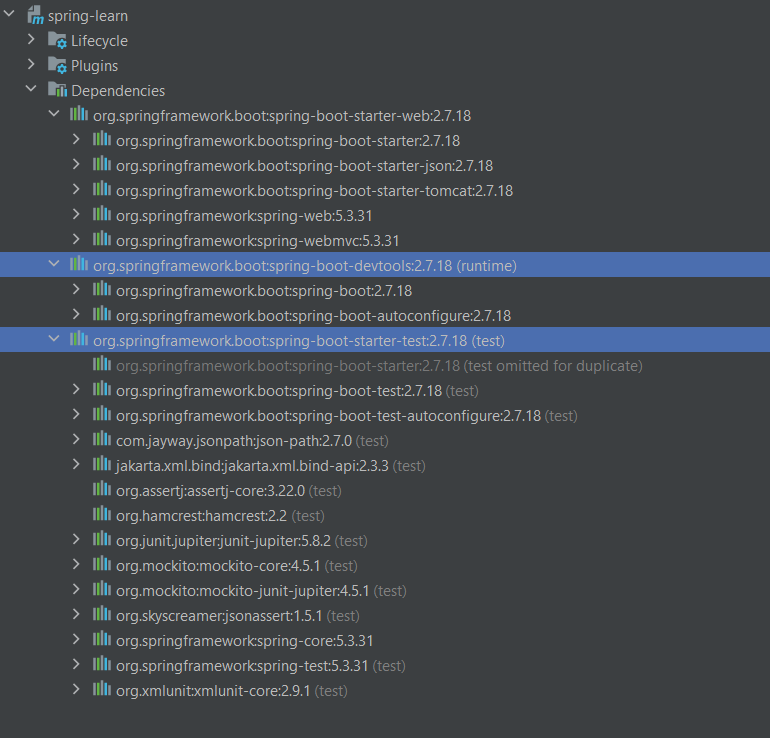
****

**d. Dependencies:**

Spring Web adds REST/HTTP support. DevTools helps with auto-restart during development. The test starter includes JUnit and Mockito.****

**2. Open 'Dependency Hierarchy' and show the dependency tree**

In the Dependency Hierarchy, we can see that the three declared dependencies—Web, DevTools, and Test—pull in a variety of transitive libraries such as Tomcat, Jackson, Spring MVC, and testing frameworks. This structure helps manage complex dependency trees easily.

****

**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.

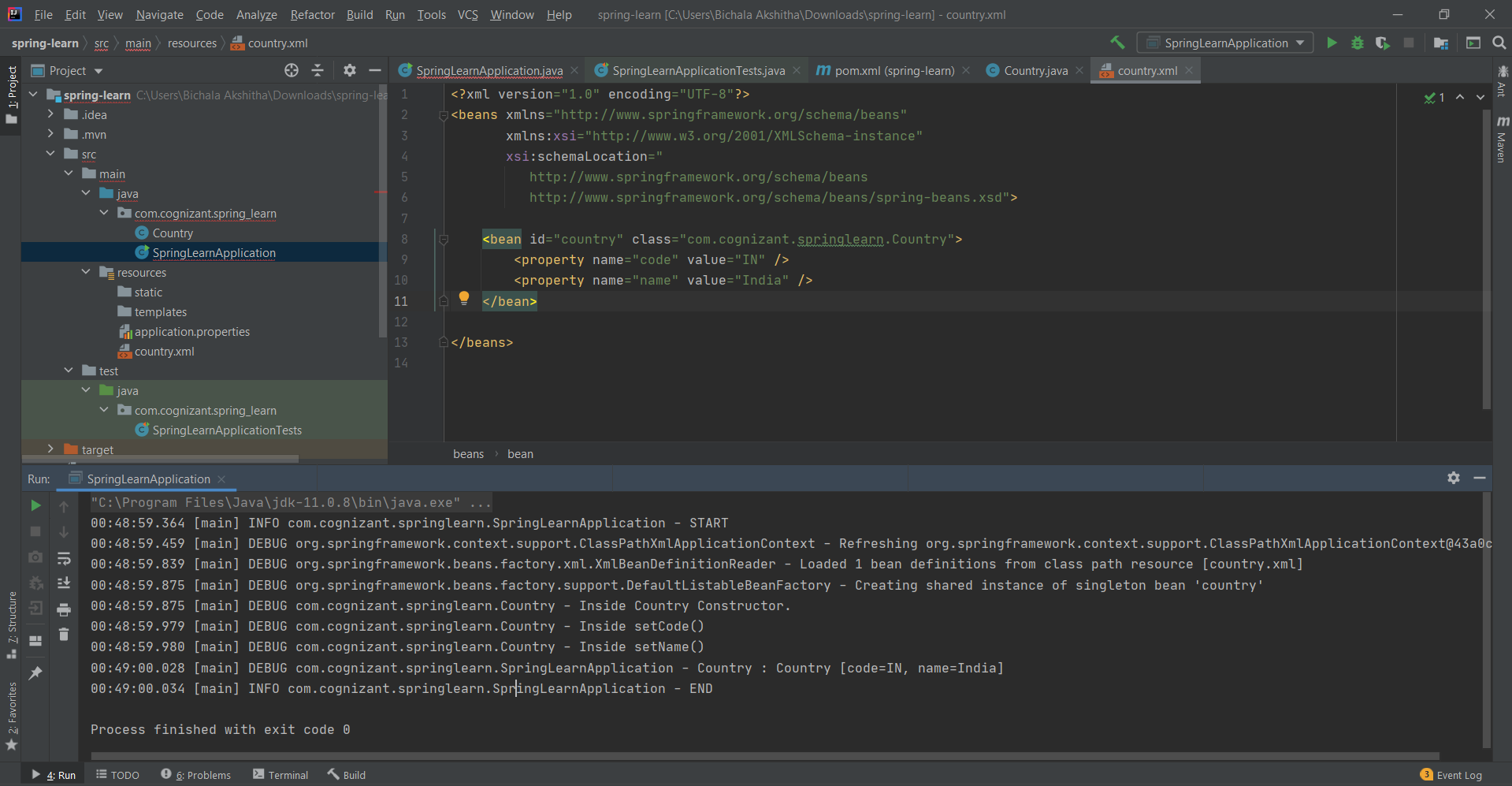
ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

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.

**Output:**

****

SME to provide more detailing about the following aspects:

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

**🔹 bean tag**

Defines an object (bean) managed by the Spring container.

**🔹 id attribute**

Unique name to refer to the bean ("country" in this case).

**🔹 class attribute**

Fully qualified name of the Java class (com.cognizant.springlearn.Country).

**🔹 property tag**

Used to inject values into fields of the bean.

**🔹 name and value attributes**

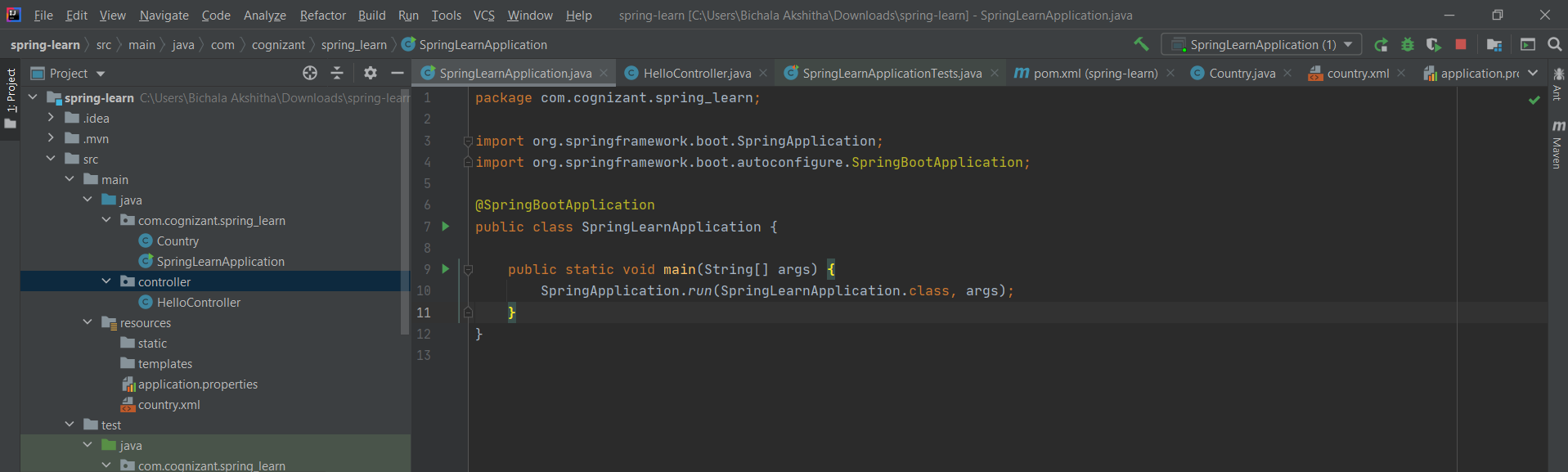
* name: matches the field in the Java class.
* value: literal value to inject.
* ApplicationContext, ClassPathXmlApplicationContext

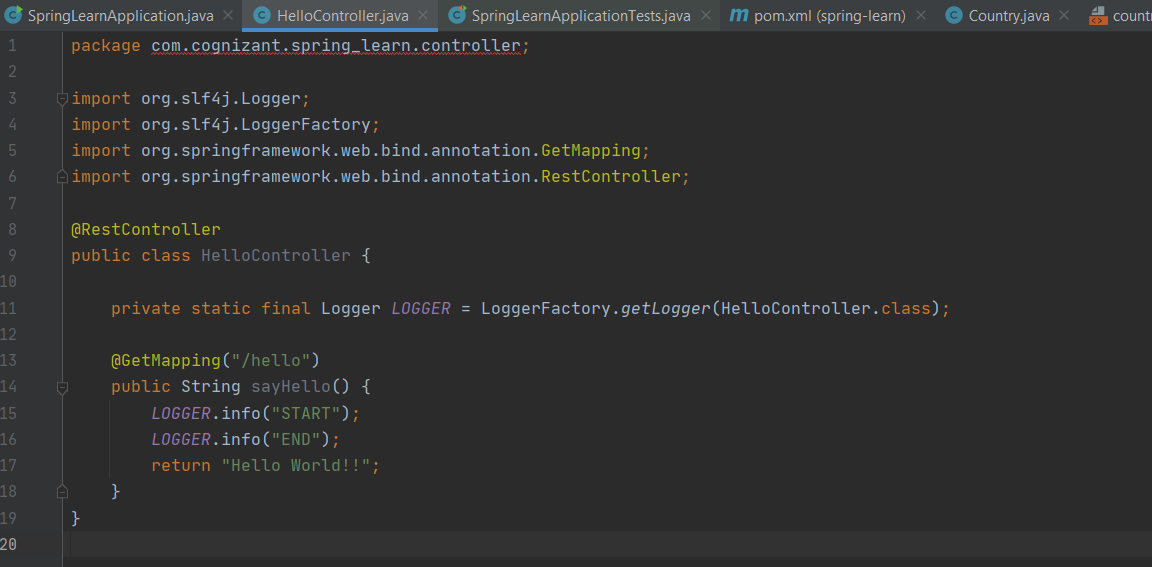
**ApplicationContext vs ClassPathXmlApplicationContext**

* ApplicationContext: The Spring container interface.
* ClassPathXmlApplicationContext: Loads XML config from classpath (like resources/).
* What exactly happens when context.getBean() is invoked
  + Spring reads country.xml.
  + Finds the bean definition for "country".
  + Instantiates the Country class.
  + Sets the code and name using setters.
  + Returns the fully initialized object.

**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!!

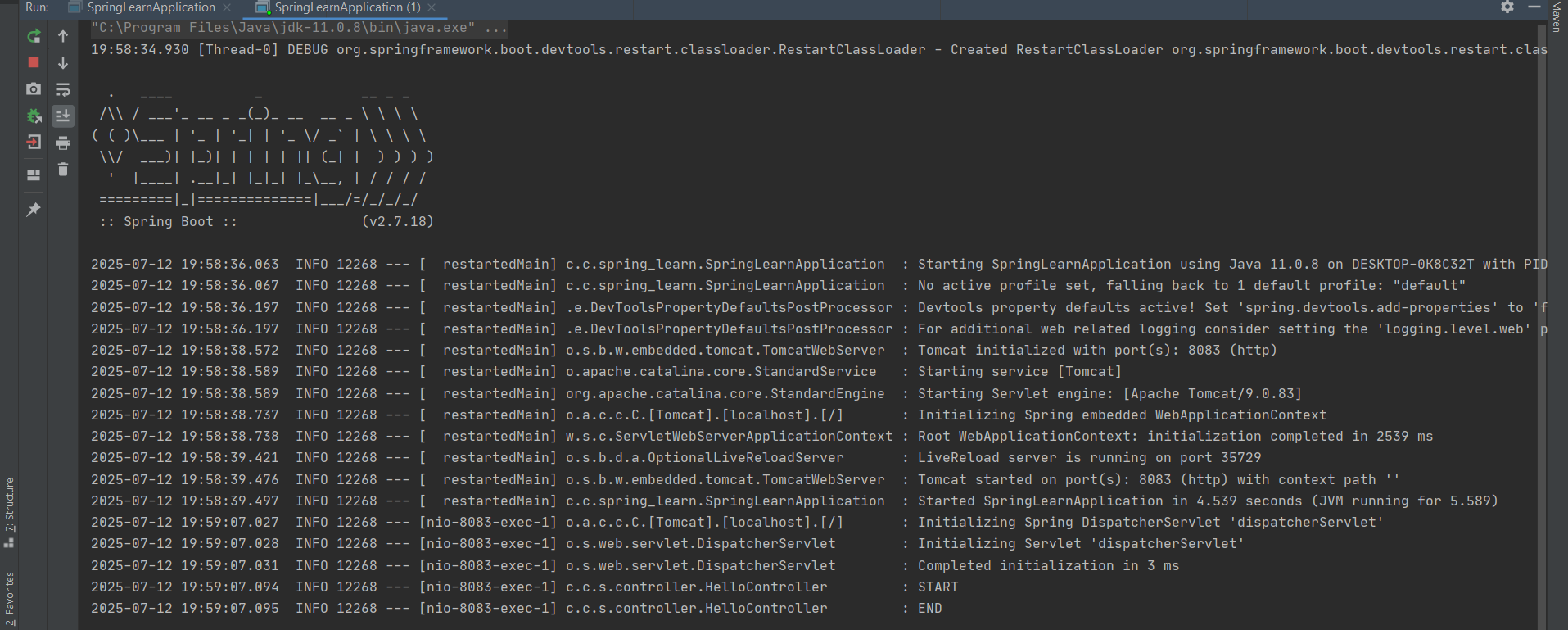
**SpringLearnApplication.java:**



**HelloController.java:**  


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

**Output:**

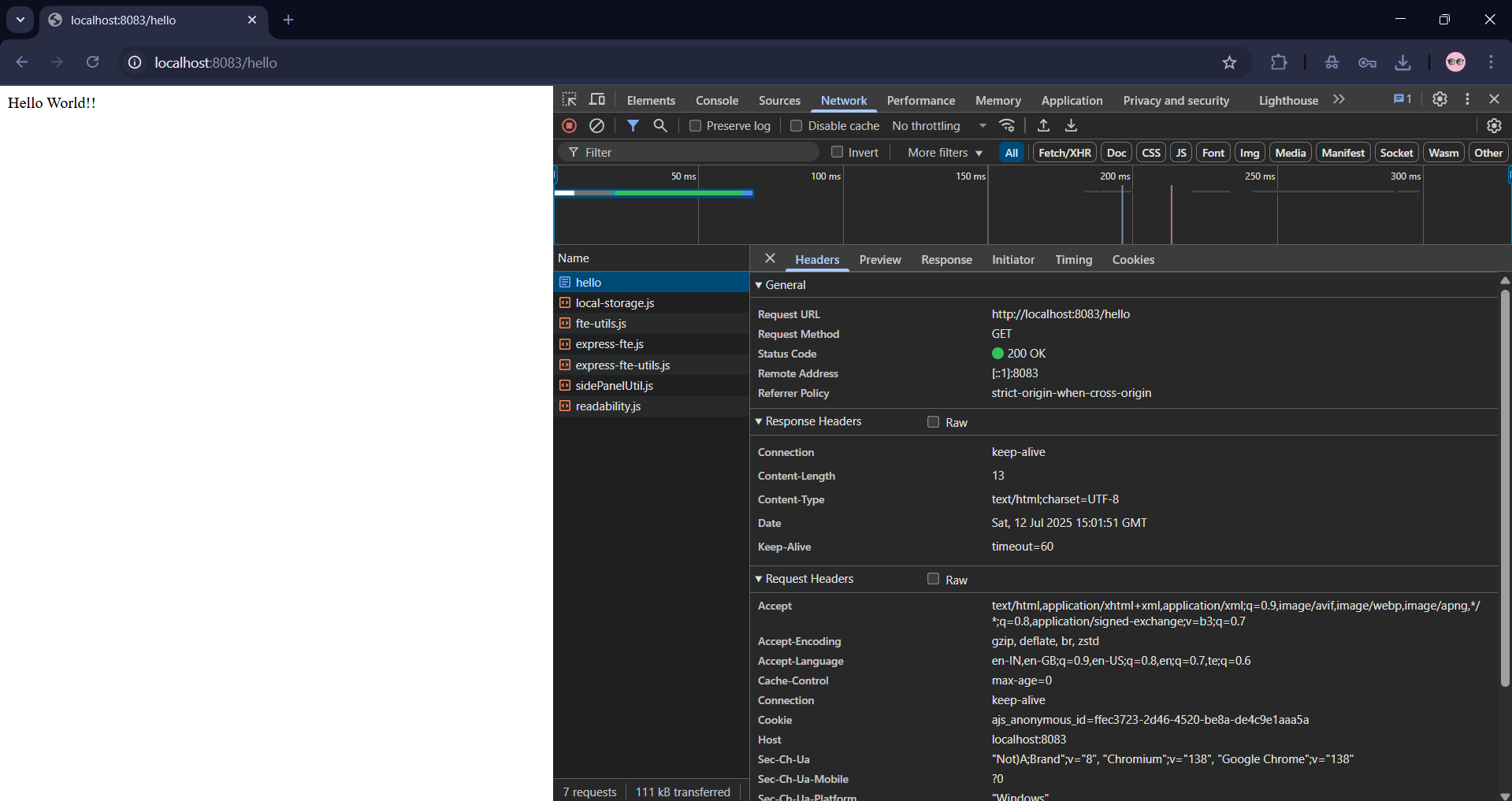
  
  
Try the URL http://localhost:8083/hello in both chrome browser and postman.

**Output:**

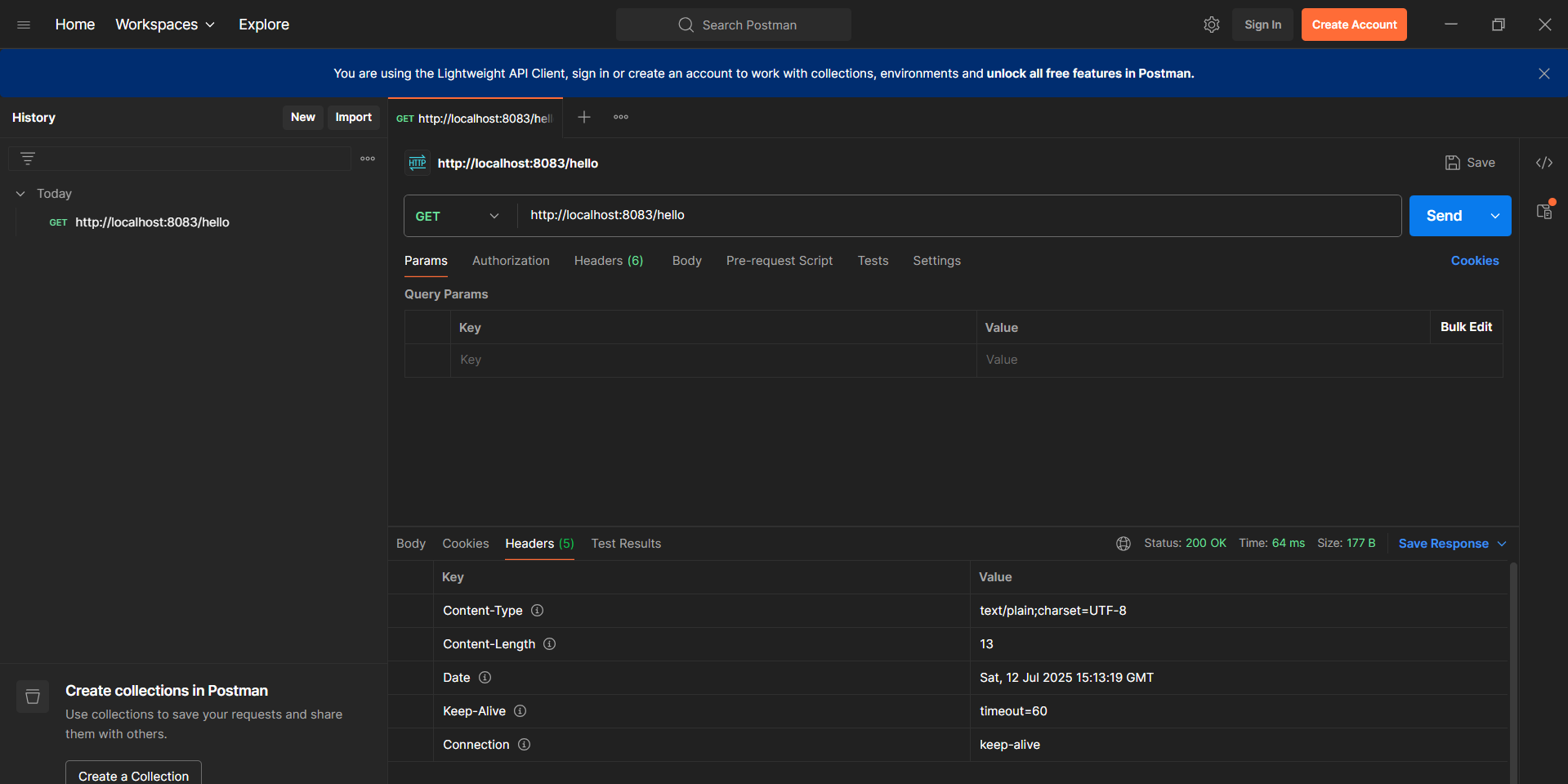
  
  
SME to explain the following aspects:

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

**Output:**

****

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



**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:

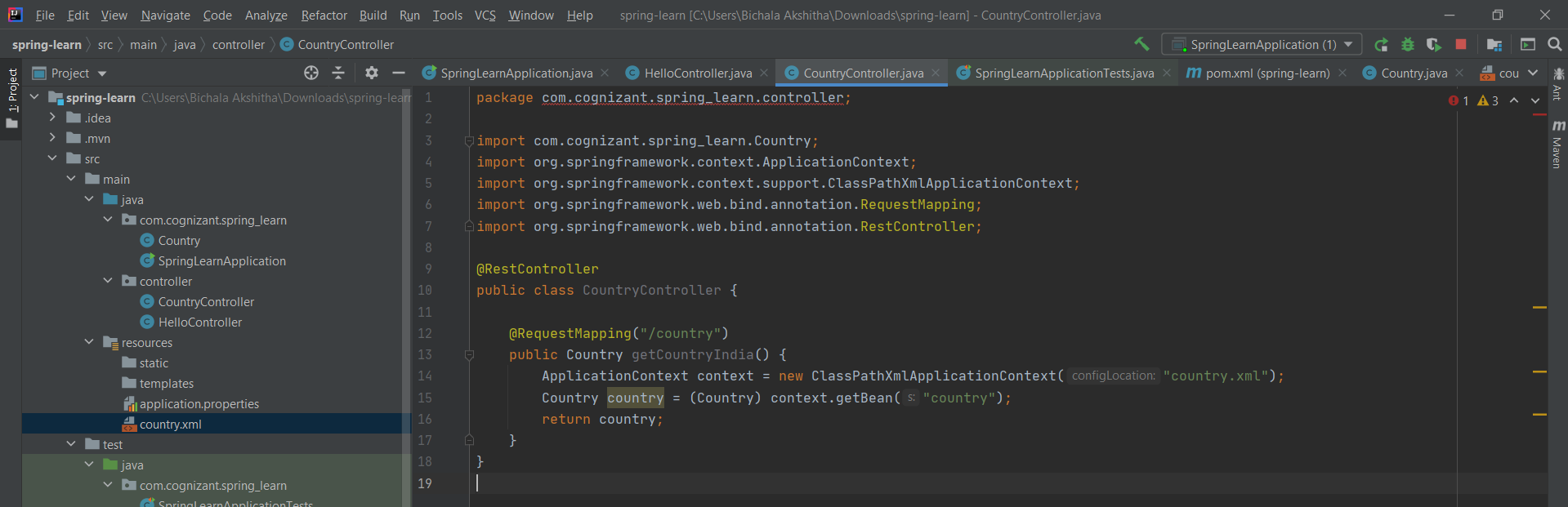
{

  "code": "IN",

  "name": "India"

}

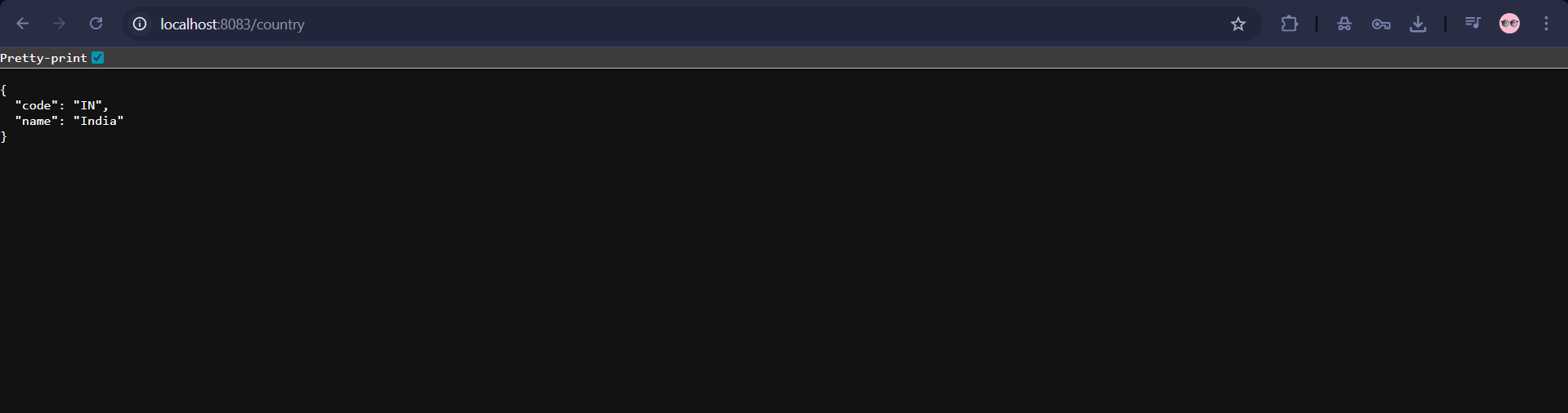
**CountryController.java:**



**Country.java:**



**Output:**

****

SME to explain the following aspects:

* **What happens in the controller method?**

@RequestMapping("/country") maps the HTTP GET request.

Spring loads country.xml using ClassPathXmlApplicationContext.

It retrieves the Country bean (India) and returns it.

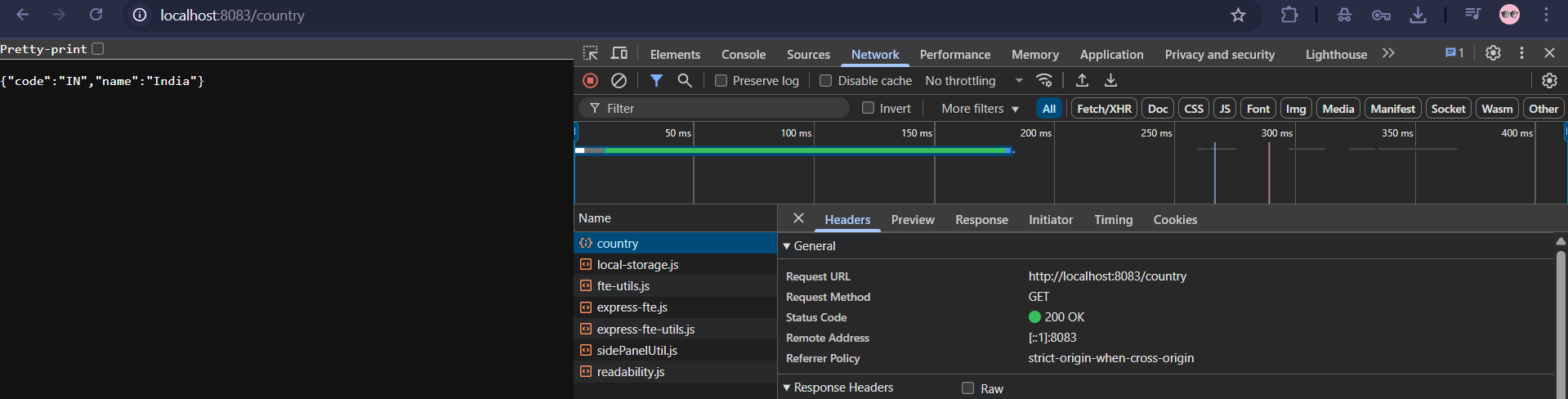
* **How the bean is converted into JSON reponse?**

Spring Boot uses Jackson (auto-included via spring-boot-starter-web).

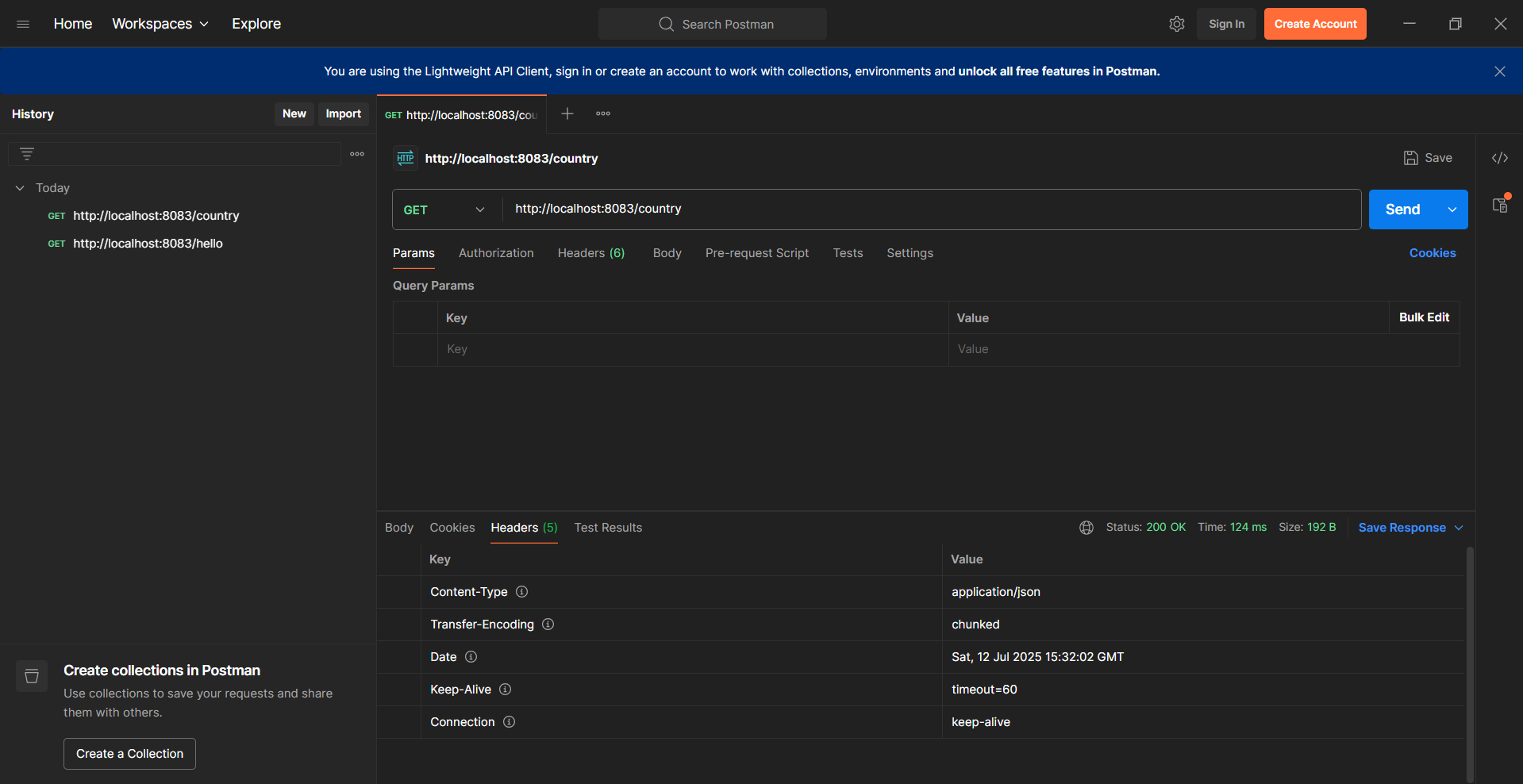
Jackson automatically converts any Java object (POJO) into JSON in REST responses.

No manual serialization is needed.

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

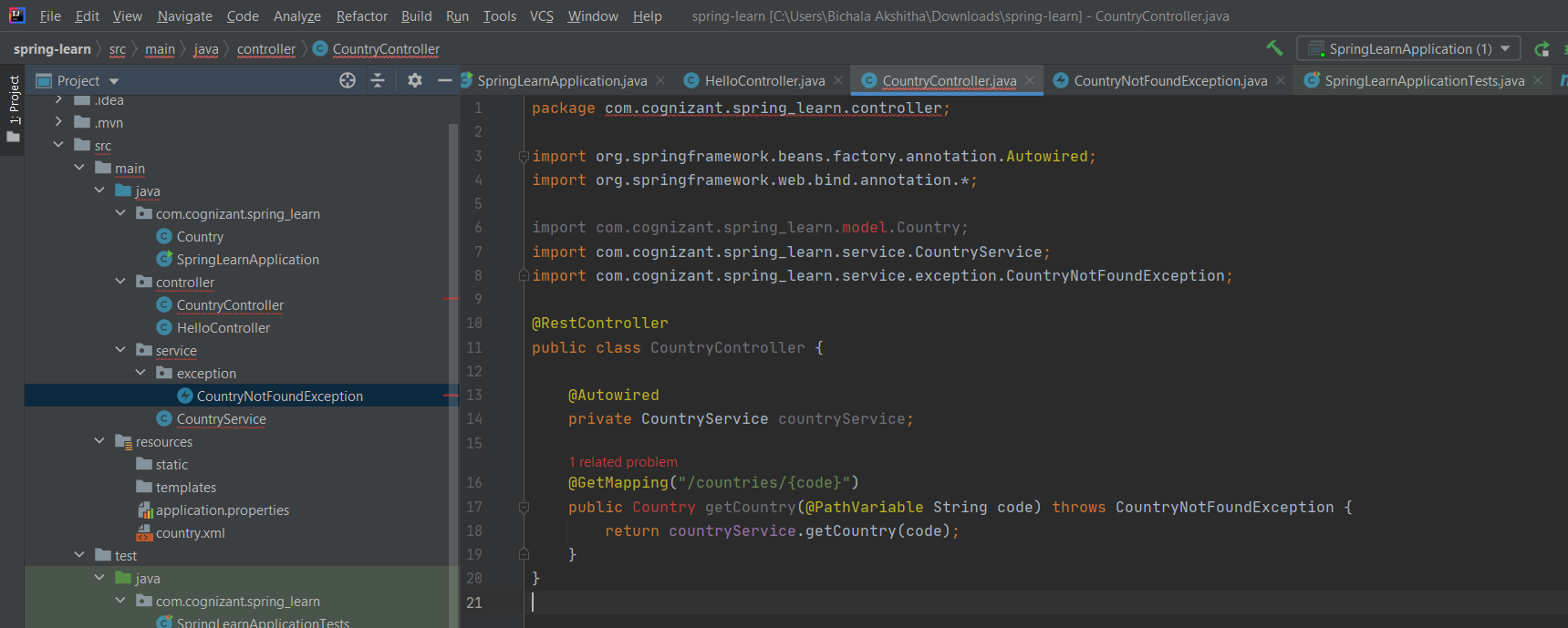
****

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

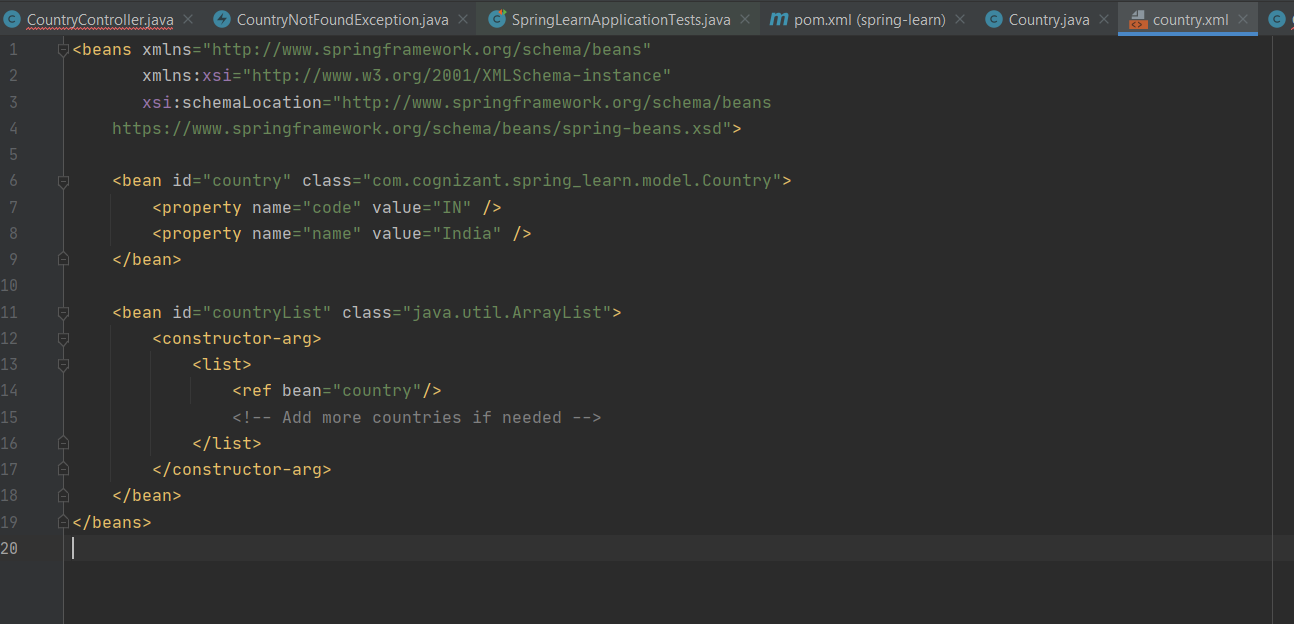
****

**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)

**CountryController.java:**



**country.xml:**

  
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:

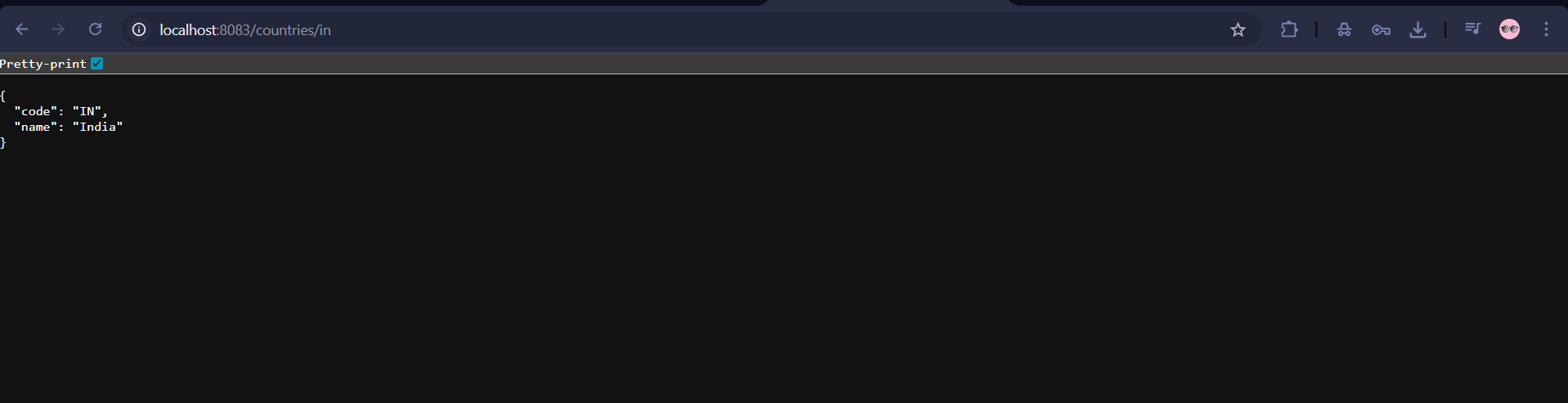
{

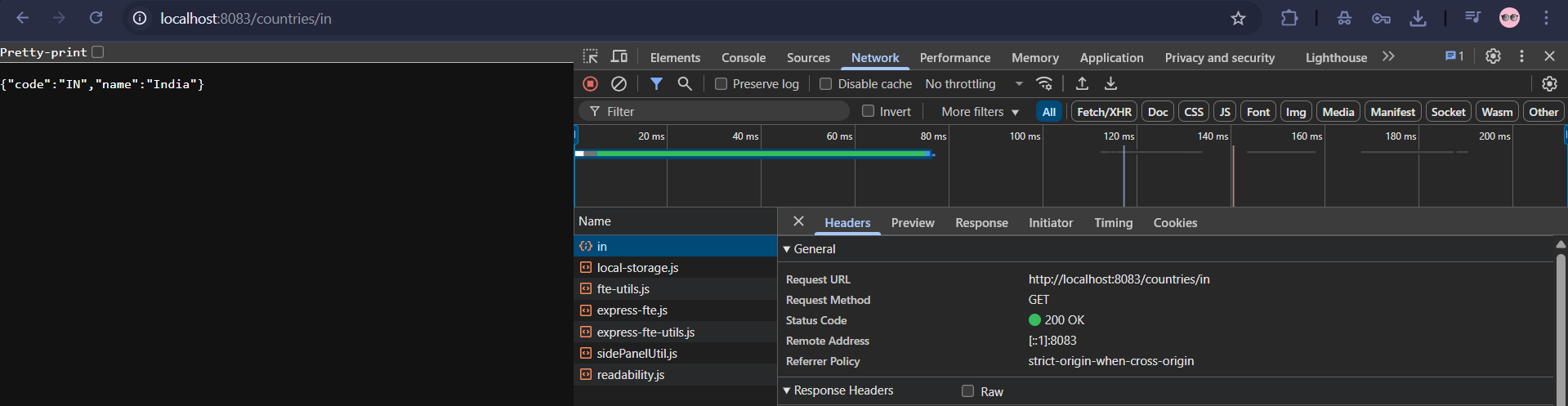
  "code": "IN",

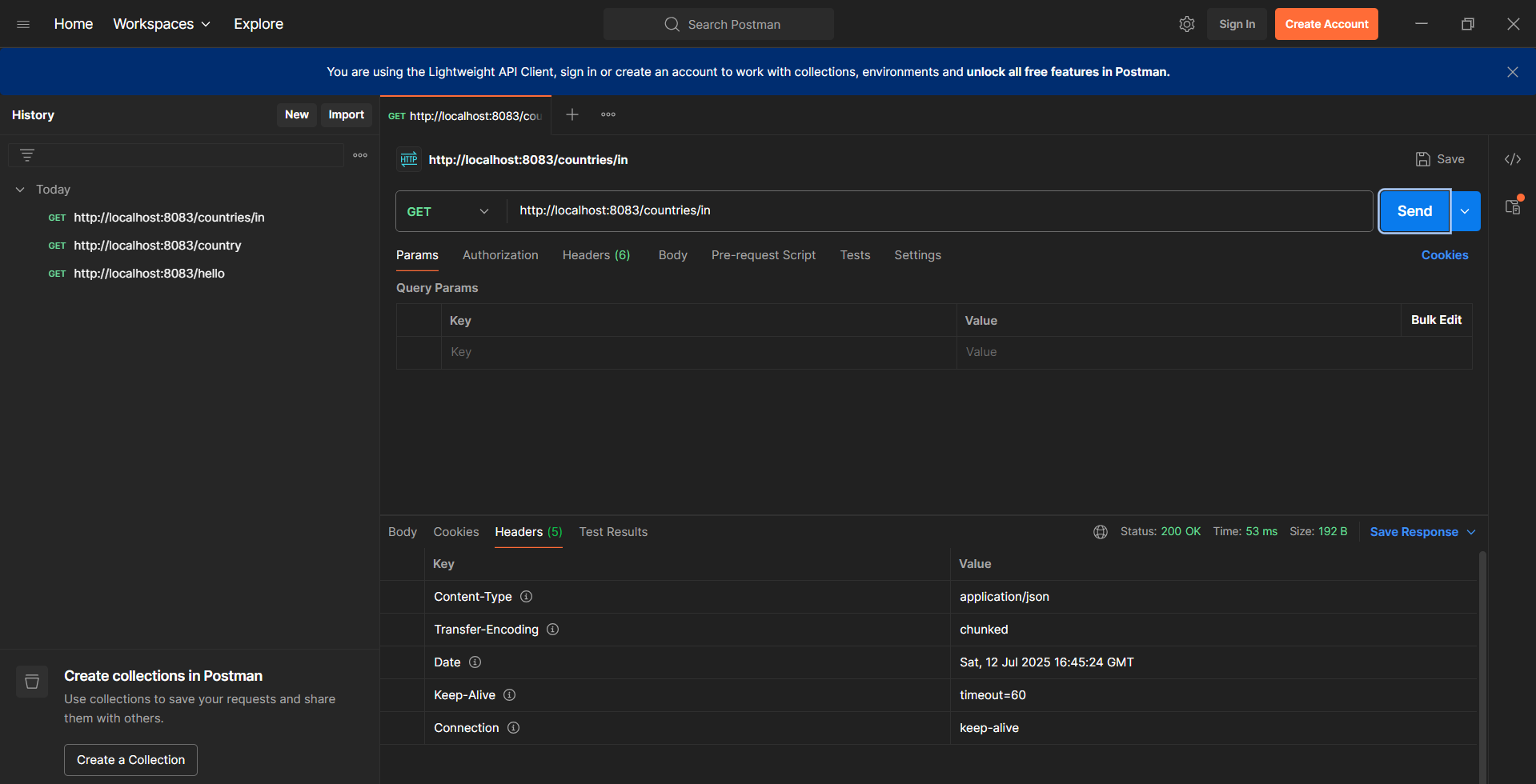
  "name": "India"

}

**Output:**

****

****

****

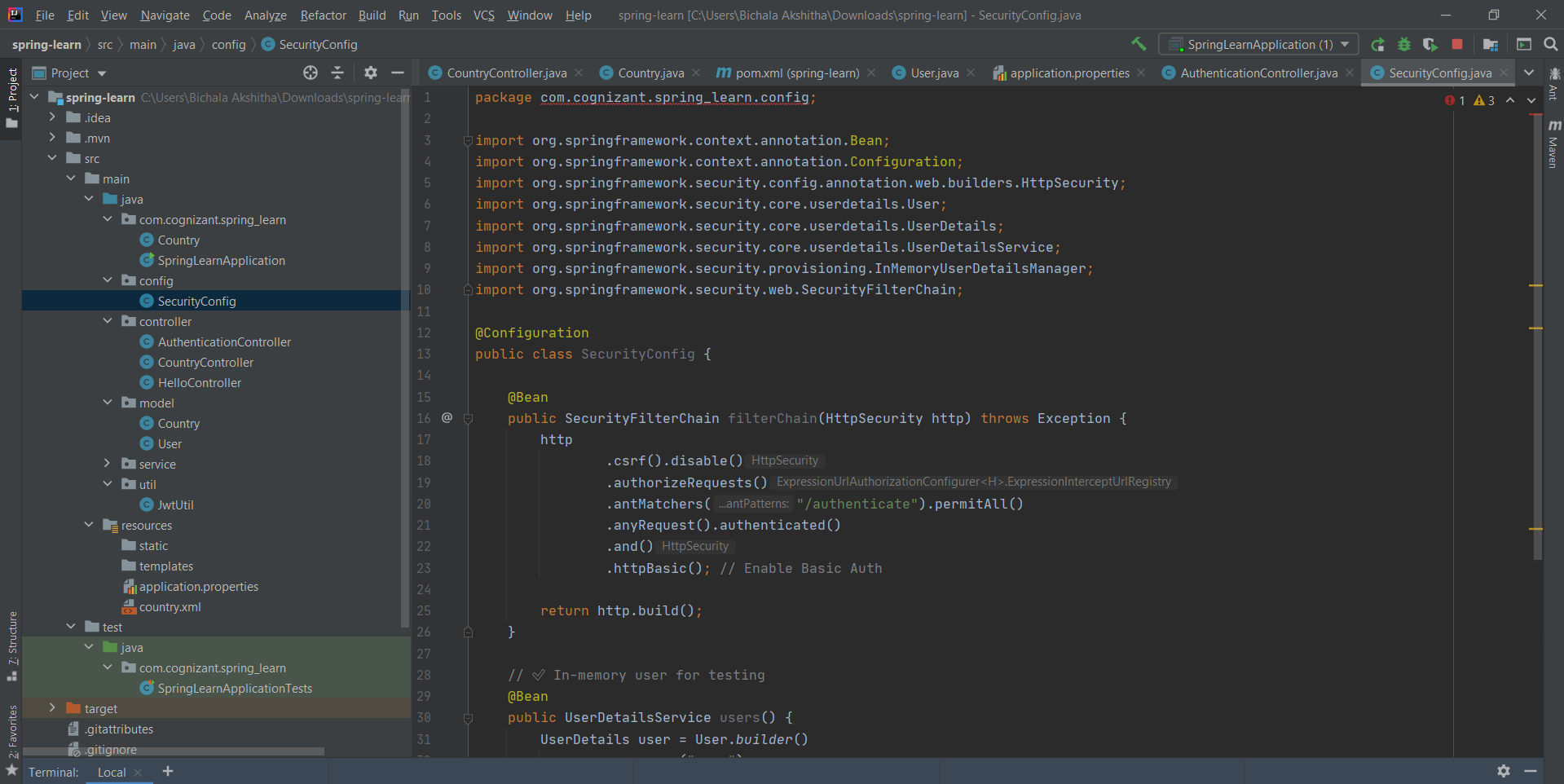
**Create authentication service that returns JWT**As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token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

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

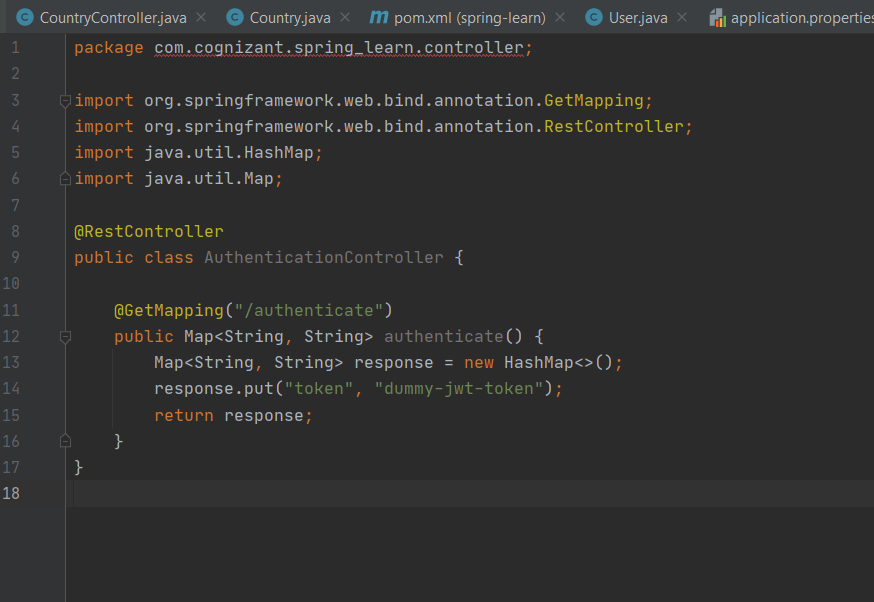
Response

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

**SecurityConfig.java:**



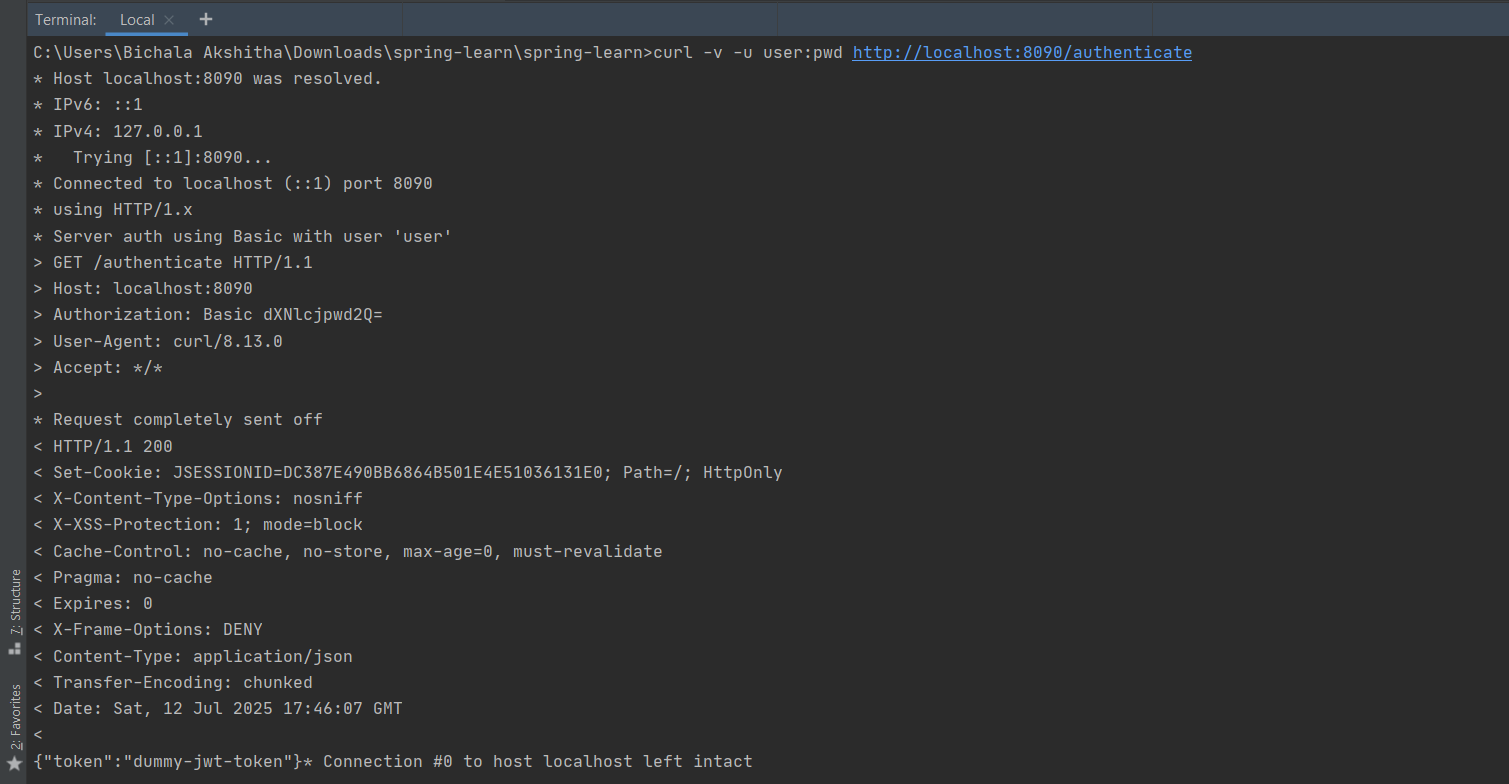
**AuthenticationController.java:**



**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.cognizant</groupId>  
 <artifactId>spring-learn</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <name>spring-learn</name>  
 <description>Spring Learn REST Web Service Project</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.7.18</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
  
 <properties>  
 <java.version>11</java.version>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Boot Web -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt</artifactId>  
 <version>0.9.1</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
  
  
 <!-- Spring Boot DevTools -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
 <!-- Spring Security -->  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </dependency>  
  
 <!-- JAXB for XML Parsing -->  
 <dependency>  
 <groupId>javax.xml.bind</groupId>  
 <artifactId>jaxb-api</artifactId>  
 </dependency>  
  
 <!-- Jackson XML -->  
 <dependency>  
 <groupId>com.fasterxml.jackson.dataformat</groupId>  
 <artifactId>jackson-dataformat-xml</artifactId>  
 </dependency>  
  
 <!-- JUnit 5 for Testing -->  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Spring Boot Maven Plugin -->  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

**Output:**

  
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

Let incorporate the above as separate hands on exercises.

**Output: **