WEEK 4

Spring REST using Spring Boot 3

**1.Create a Spring Web Project using Maven**

**DESCRIPTION:**

This hands-on exercise walks through setting up a basic Spring Boot web application using Maven. The project, named spring-learn, was created via [start.spring.io](https://start.spring.io), imported into Eclipse, built from command line using Maven, and executed using the Spring Boot framework. This document includes CLI logs, screenshots, application code, and configuration details.

**CODE:**

pom.xml

<?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>3.5.3</version>

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

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-learn</name>

<description>Spring Web Demo</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</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-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

SpringLearnApplication.java

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringLearnApplication {

public static void main(String[] args) {

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

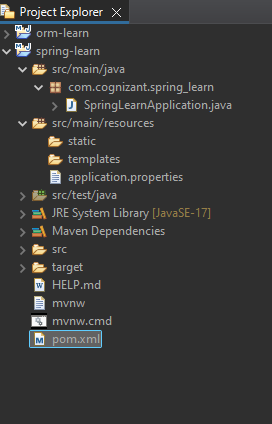
System.***out***.println("SpringLearnApplication started successfully!");

}

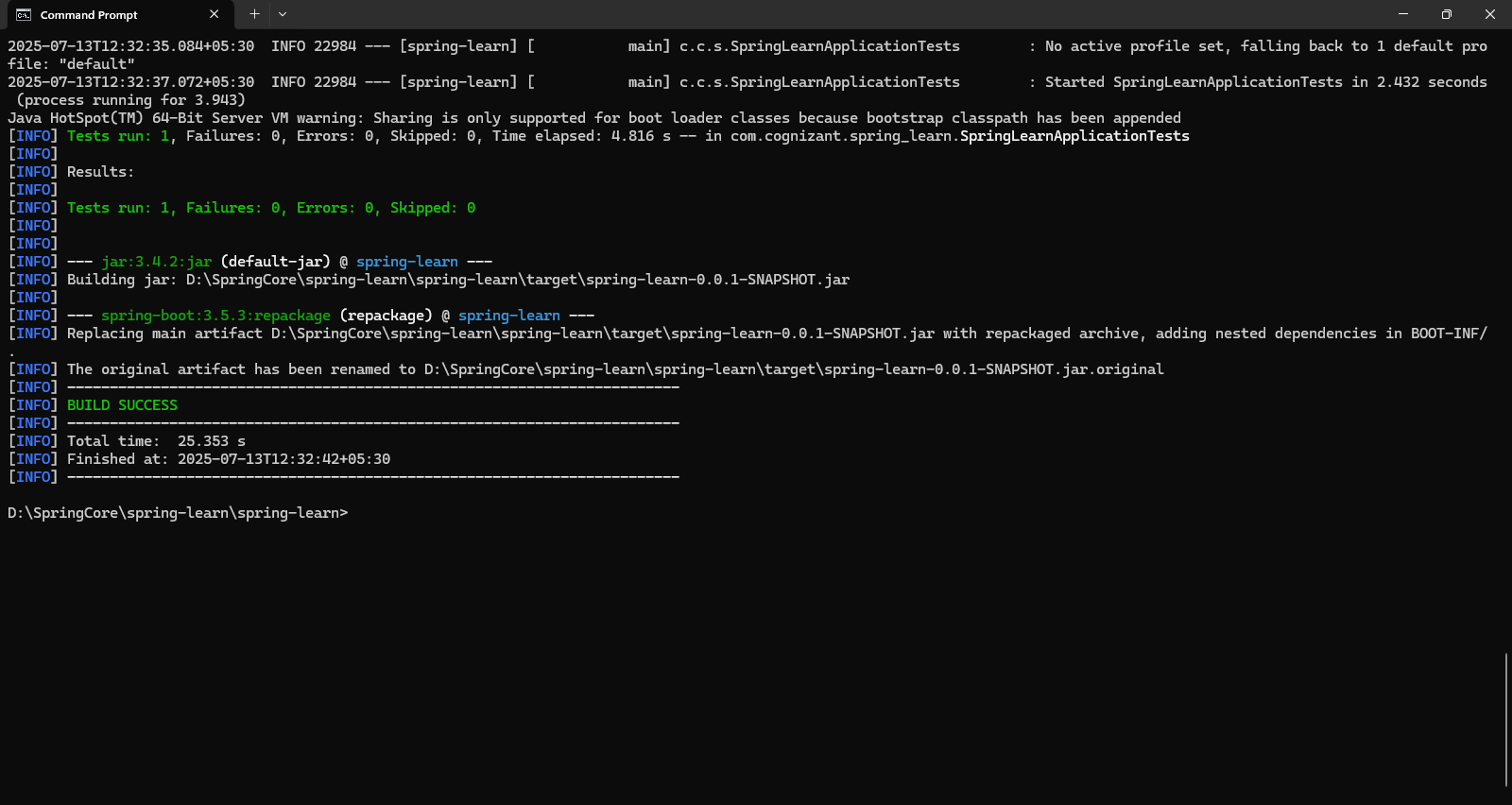
}

**Screenshots:**

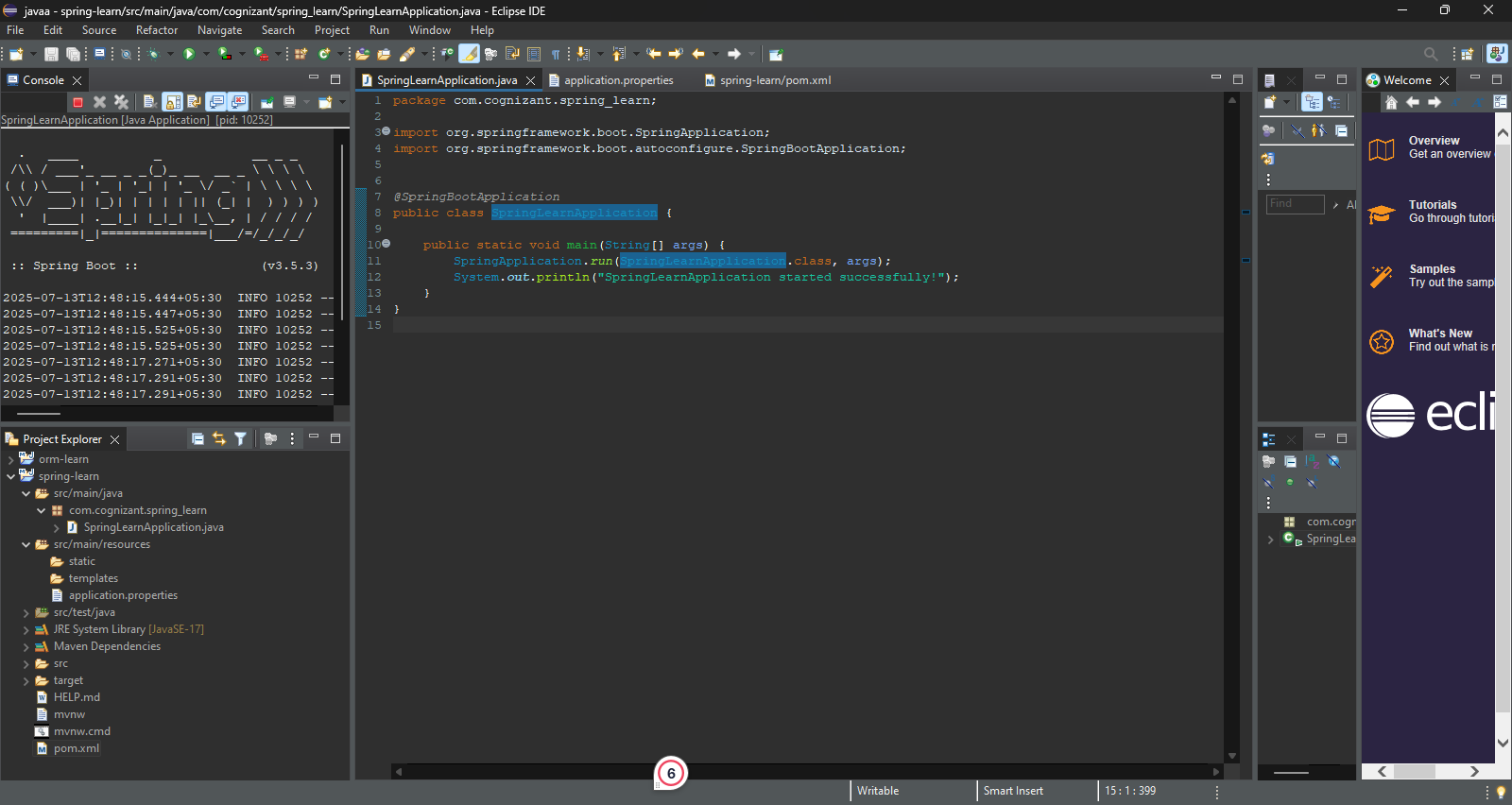
Project Folder Structure in Eclipse



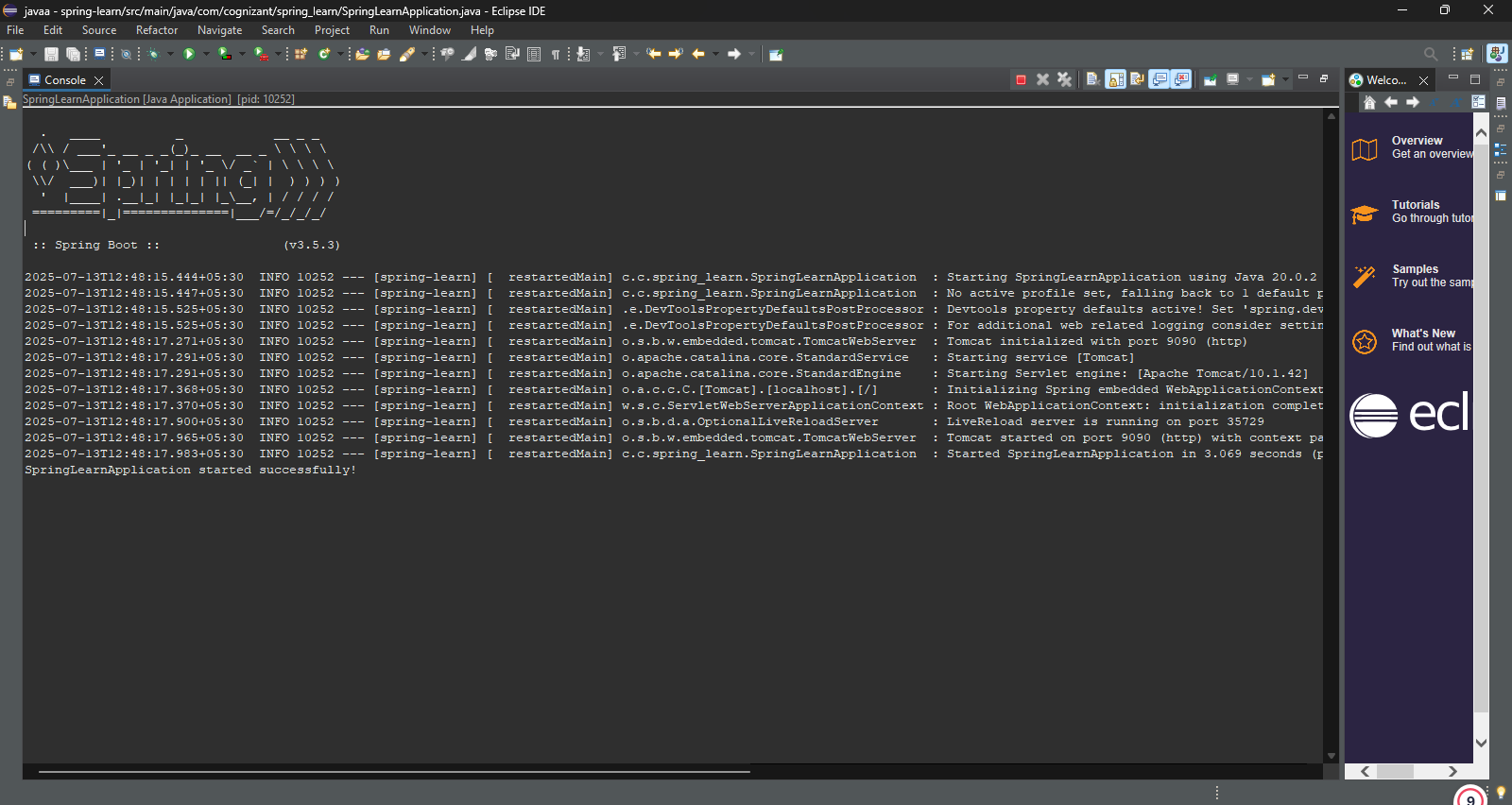
**Maven Build Success**



**SpringLearnApplication.java**



**Console Output**



**Conclusion**

This hands-on helped me understand how to set up a Spring Boot web application using Maven. I created the project, built it using the command line, imported it into Eclipse, and ran it successfully. I also learned how to change the default port and view dependencies. This activity gave me a good starting point for working with Spring Boot projects.

**2.Spring Core – Load Country from Spring Configuration XML**

**Objective:**

To demonstrate how to load bean definitions using Spring's XML-based configuration by defining a Country bean and retrieving it in a Spring application.

**Code:**

**country.xml (under src/main/resources):**

xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="country" class="com.cognizant.spring\_learn.Country">

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

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

</bean>

</beans>

**Country.java**

package com.cognizant.spring\_learn;

public class Country {

private String code;

private String name;

public Country() {

System.out.println("Inside Country Constructor.");

}

public String getCode() {

return code;

}

public void setCode(String code) {

System.out.println("Inside setCode()");

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

System.out.println("Inside setName()");

this.name = name;

}

@Override

public String toString() {

return "Country{code='" + code + "', name='" + name + "'}";

}

}

**SpringLearnApplication.java**package com.cognizant.spring\_learn;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringLearnApplication {

public static void main(String[] args) {

System.out.println("START of main()");

displayCountry();

System.out.println("END of main()");

}

public static void displayCountry() {

System.out.println("Before loading XML");

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

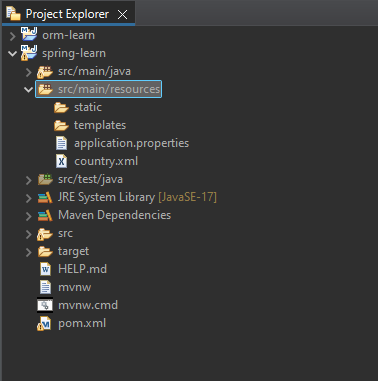
Country country = context.getBean("country", Country.class);

System.out.println("Country: " + country.toString());

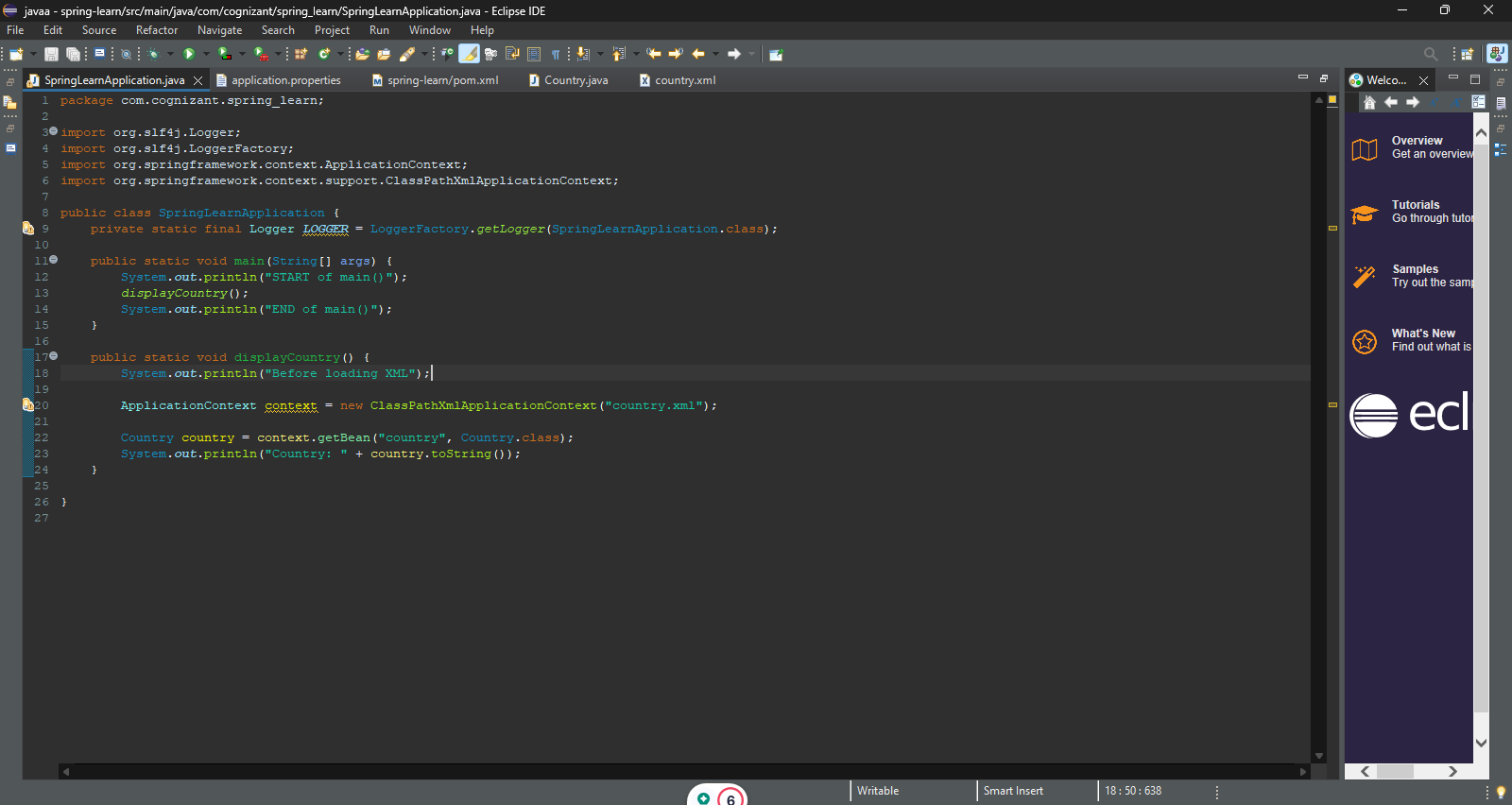
}

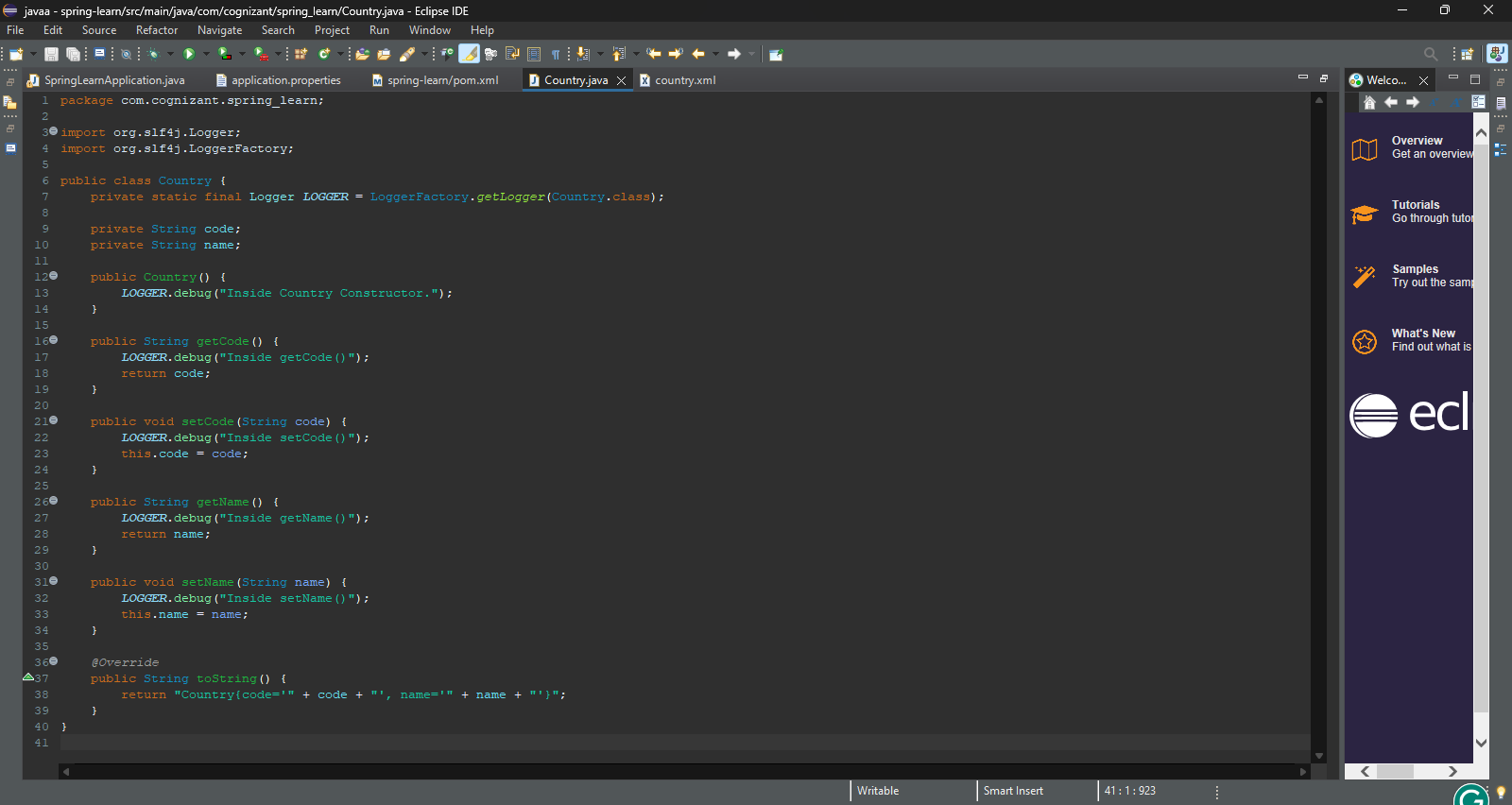
}

**Screenshot:**

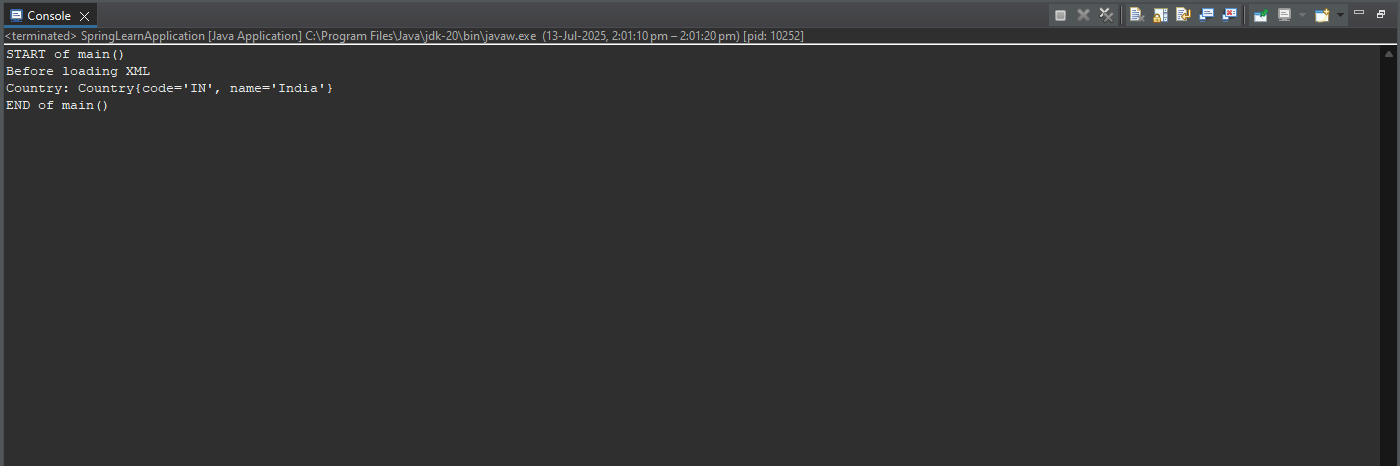


ECLIPSE SCREENSHOT:





**Console output:**



**Conclusion:**

This hands-on demonstrated the use of **Spring XML configuration** to:

* Define a bean (Country)
* Set properties (code and name)
* Retrieve and use it in a Java application using **Spring’s ApplicationContext**

**3.Hello World RESTful Web Service**

**Objective**

To create a simple Spring Boot REST API that returns "Hello World!!" on accessing the /hello endpoint via GET request.

**Tools Used**

* Java 17
* Spring Boot 3.5.3
* Maven
* Eclipse
* Google Chrome (Developer Tools)
* Postman

**Code**

**HelloController.java**

package com.cognizant.spring\_learn.contoller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class HelloController {

private static final Logger LOGGER = LoggerFactory.getLogger(HelloController.class);

@GetMapping("/hello")

public String sayHello() {

LOGGER.info("START of sayHello()");

String message = "Hello World!!";

LOGGER.info("END of sayHello()");

return message;

}}

SpringLearnApplication.java

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(SpringLearnApplication.class);

public static void main(String[] args) {

LOGGER.info("START of main()");

SpringApplication.run(SpringLearnApplication.class, args);

LOGGER.info("END of main()");

}

}

**Request:**

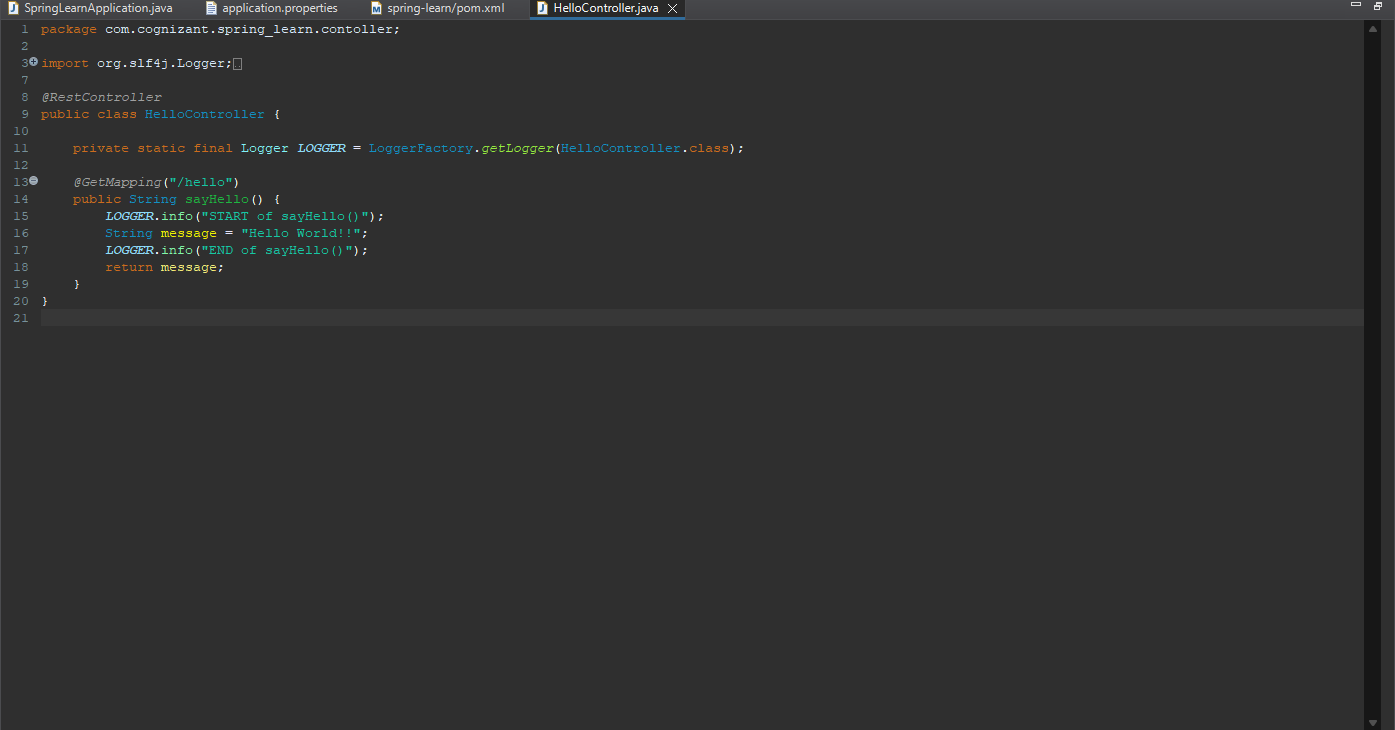
GET http://localhost:8083/hello

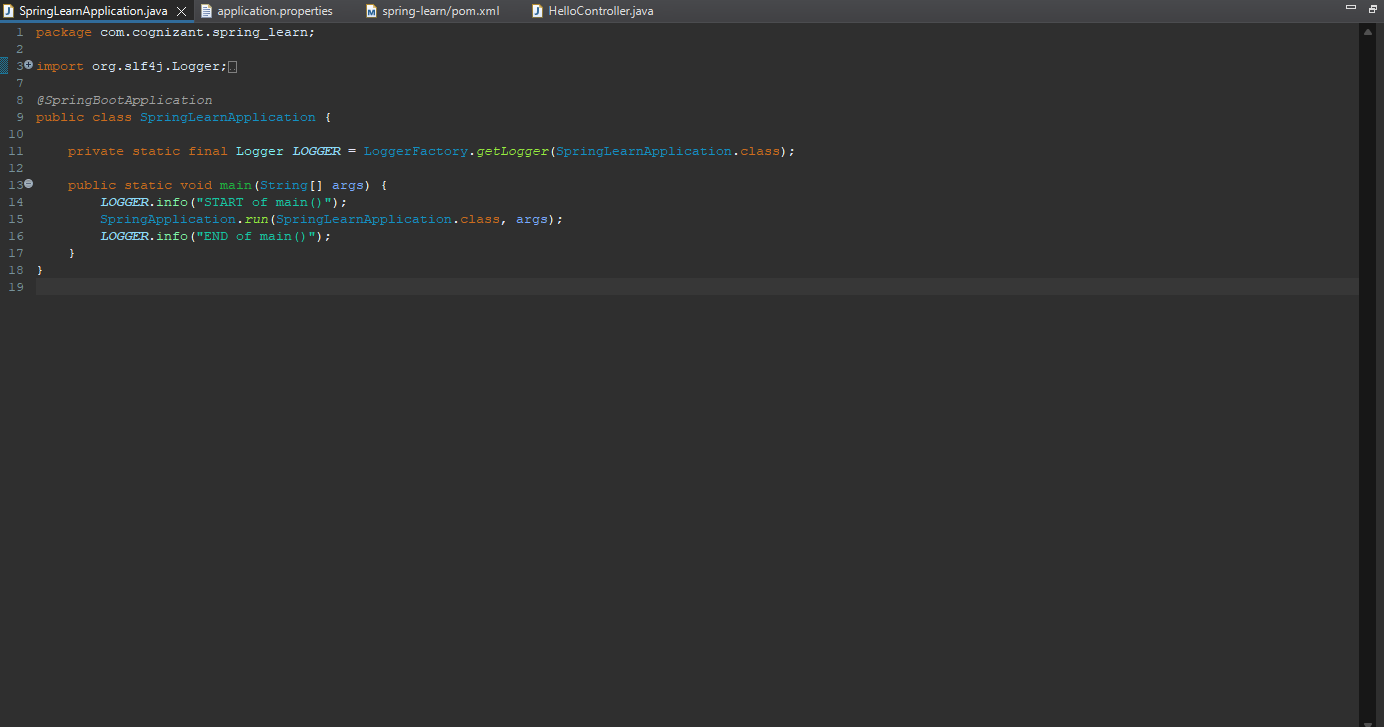
**Response:**

Hello World!!

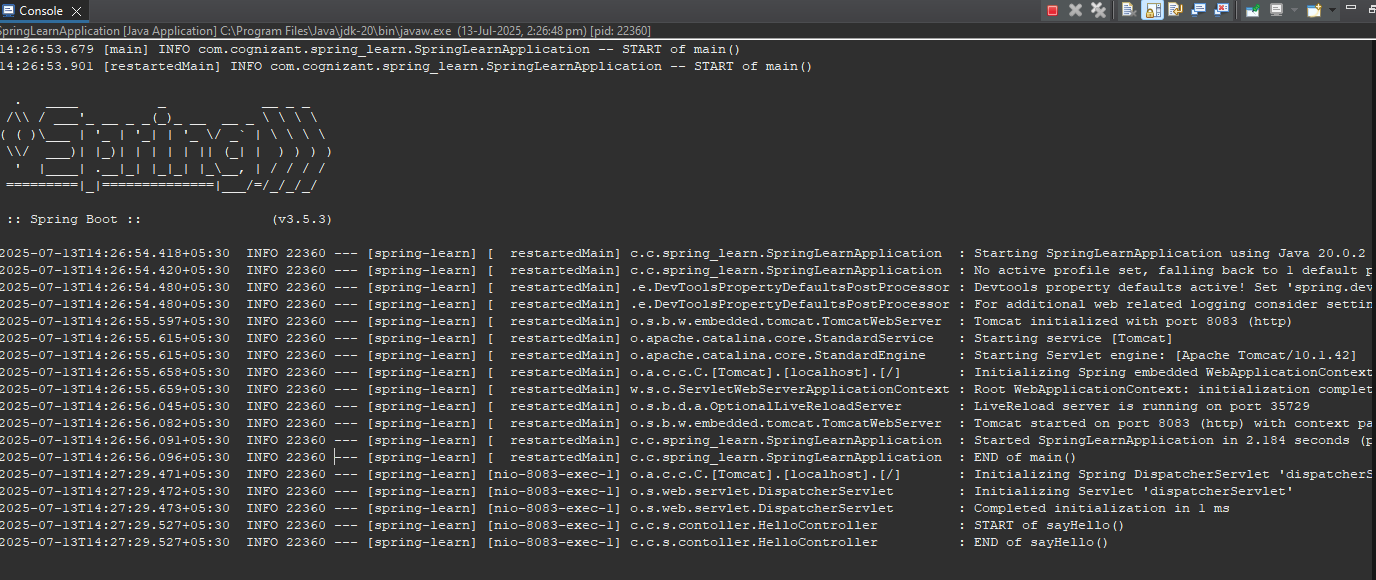
**Screenshot:**

**Eclipse code screenshots:**

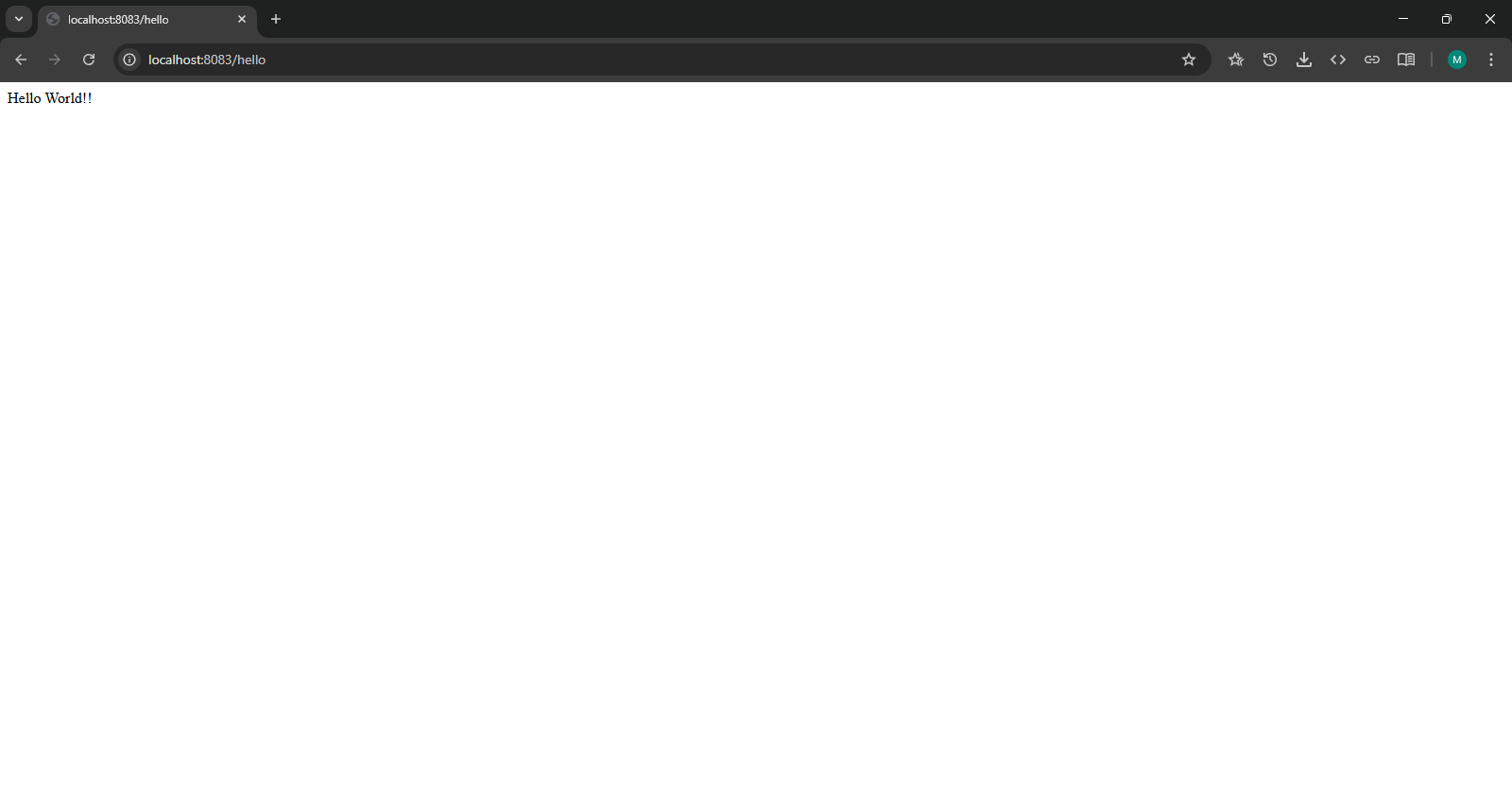




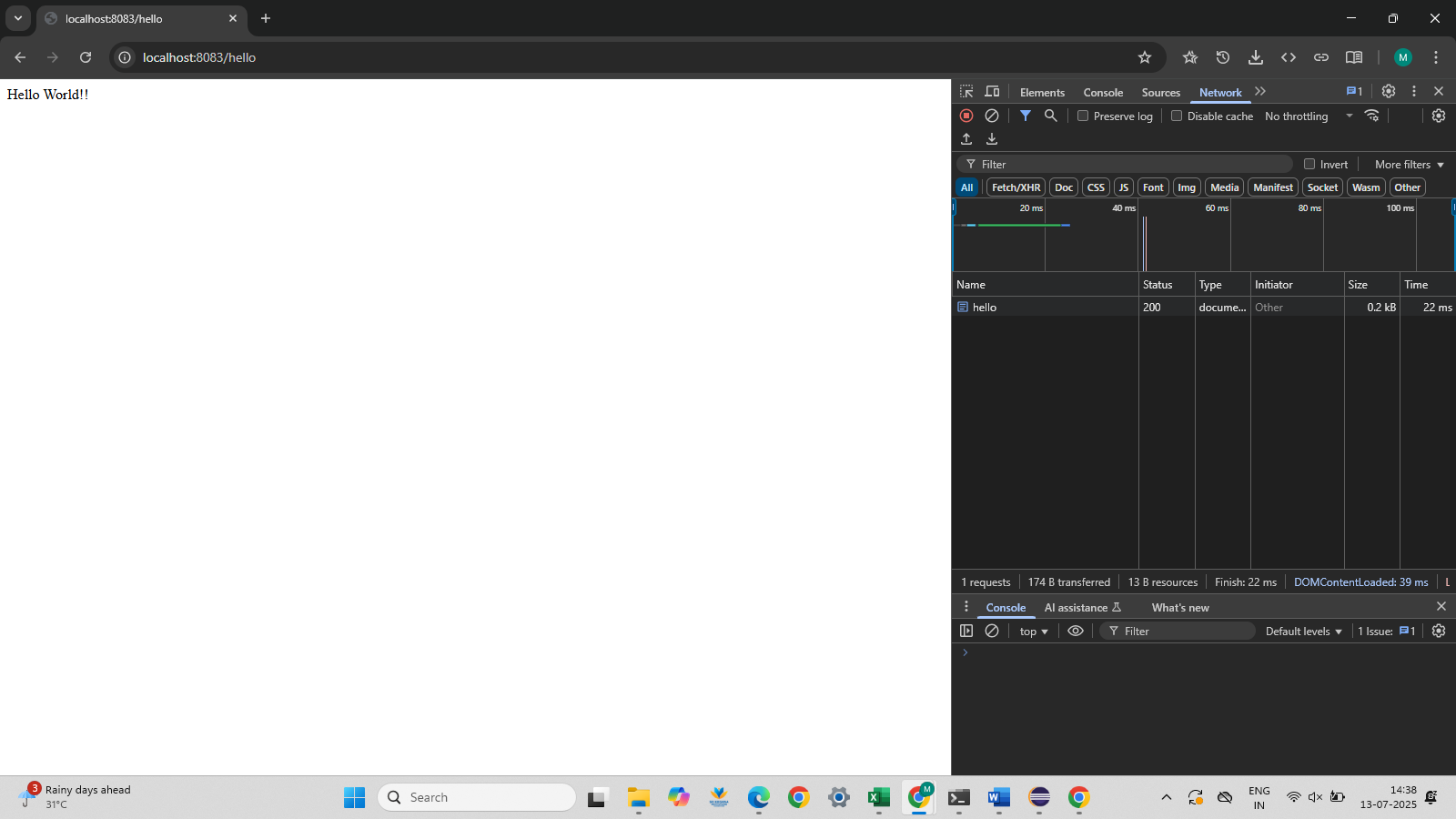
**Console Output:**

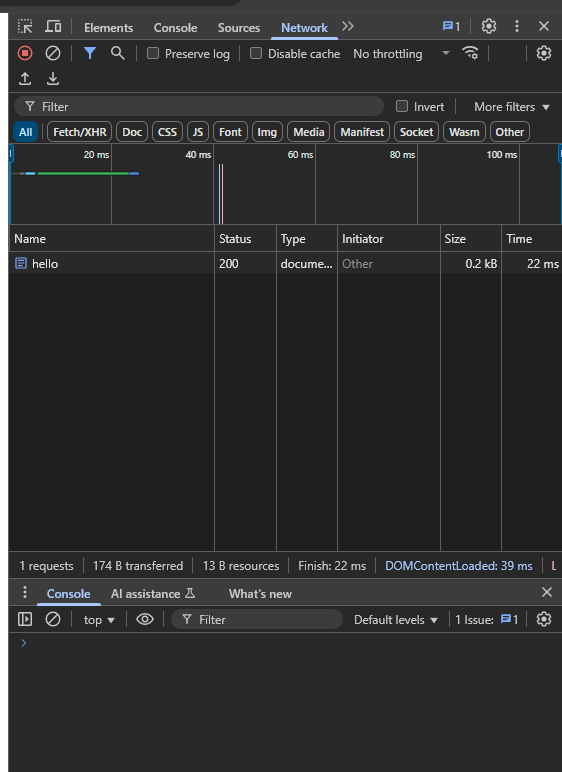


**Browser Output**

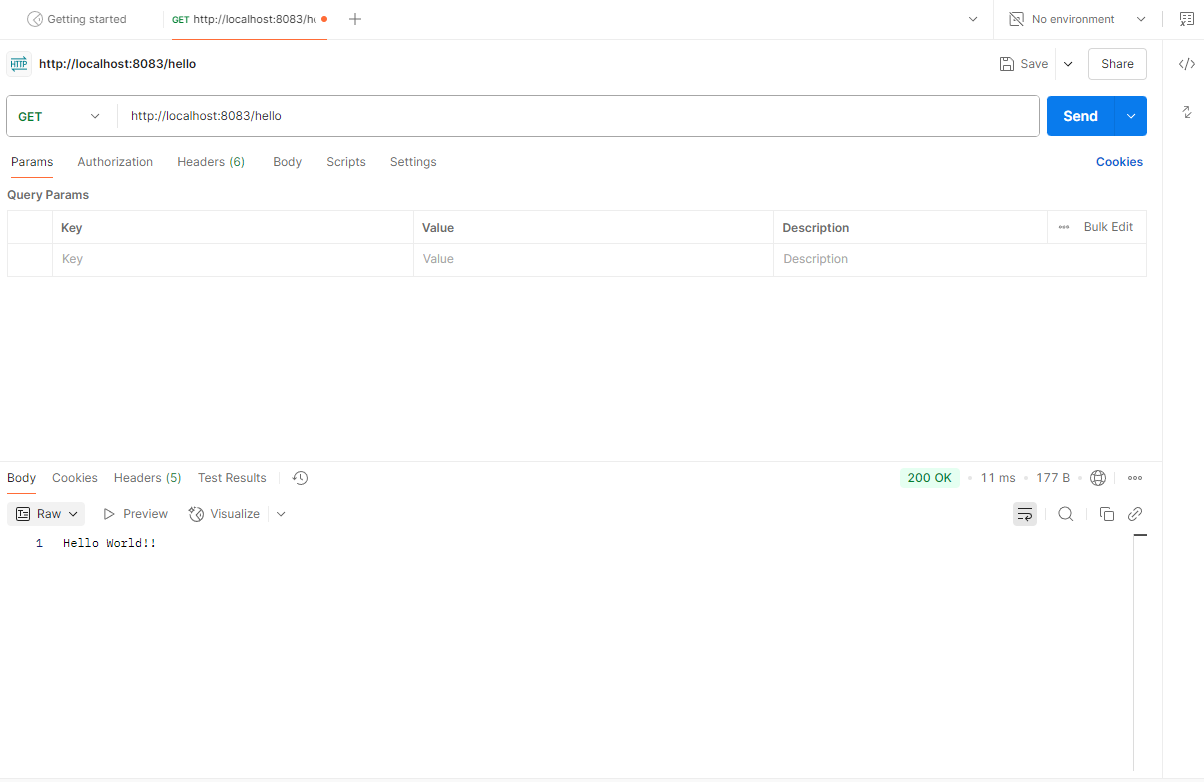


**Chrome Developer Tools – Network Tab**

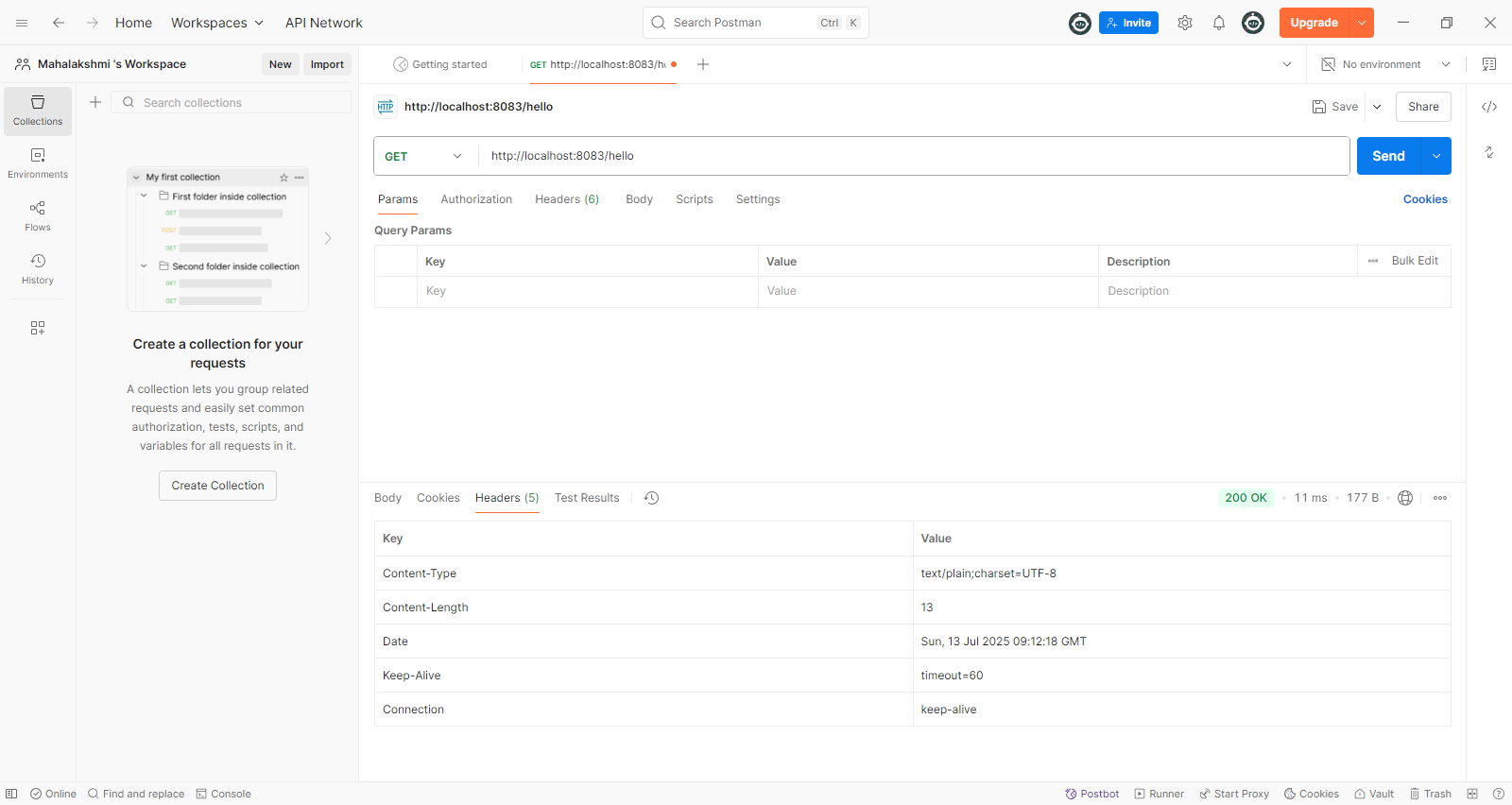




**Postman Output:**



**Postman Headers:**



**Conclusion**

The REST endpoint was successfully developed and tested. The controller returned "Hello World!!" as expected. Developer tools and Postman were used to inspect the HTTP response headers and confirm proper behavior with a 200 OK status.

4.REST - Country Web Service

**Task Description**

Develop a REST service using Spring Web Framework to return country details for India using a Spring XML bean.

**Code:**

CountryController.java

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.Country;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

import org.springframework.web.bind.annotation.RestController;

@RestController

public class CountryController {

private static final Logger LOGGER = LoggerFactory.getLogger(CountryController.class);

@RequestMapping("/country")

public Country getCountryIndia() {

LOGGER.info("START getCountryIndia()");

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

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

LOGGER.info("END getCountryIndia()");

return country;

}

}

**Country.java:**

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country {

private static final Logger LOGGER = LoggerFactory.getLogger(Country.class);

private String code;

private String name;

public Country() {

LOGGER.debug("Inside Country Constructor.");

}

public String getCode() {

LOGGER.debug("Inside getCode()");

return code;

}

public void setCode(String code) {

LOGGER.debug("Inside setCode()");

this.code = code;

}

public String getName() {

LOGGER.debug("Inside getName()");

return name;

}

public void setName(String name) {

LOGGER.debug("Inside setName()");

this.name = name;

}

@Override

public String toString() {

return "Country{code='" + code + "', name='" + name + "'}";}}

**country.xml**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="country" class="com.cognizant.spring\_learn.Country">

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

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

</bean>

</beans>

SpringLearnApplication.java

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(SpringLearnApplication.class);

public static void main(String[] args) {

LOGGER.info("START of main()");

SpringApplication.run(SpringLearnApplication.class, args);

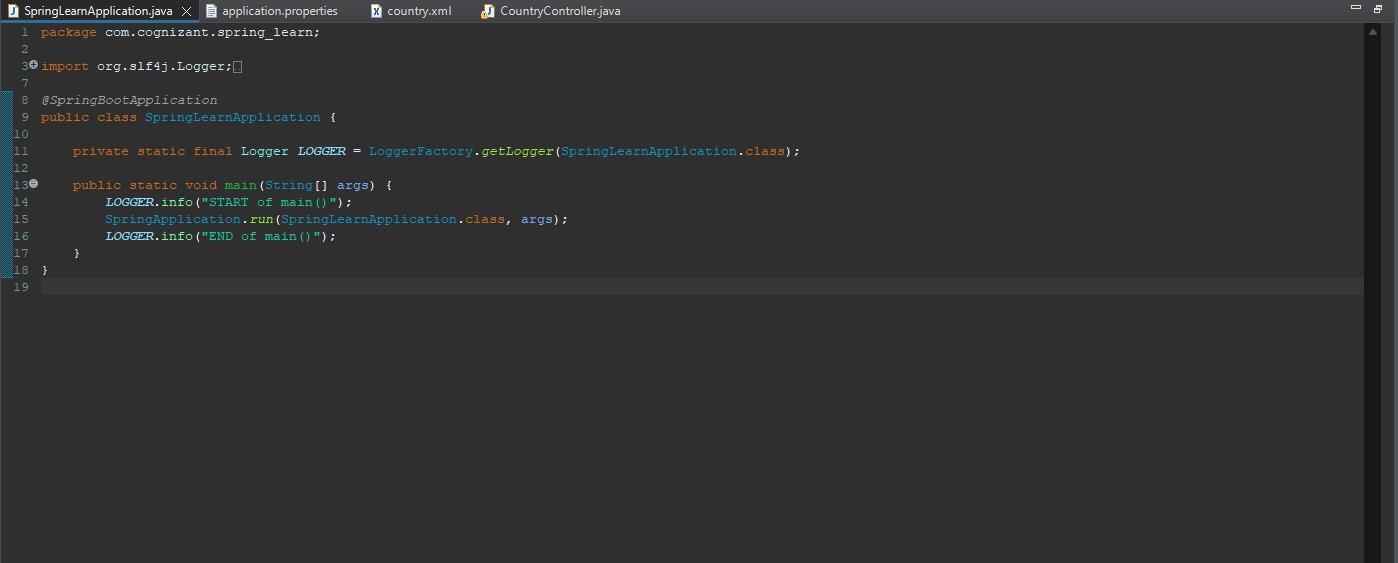
LOGGER.info("END of main()");

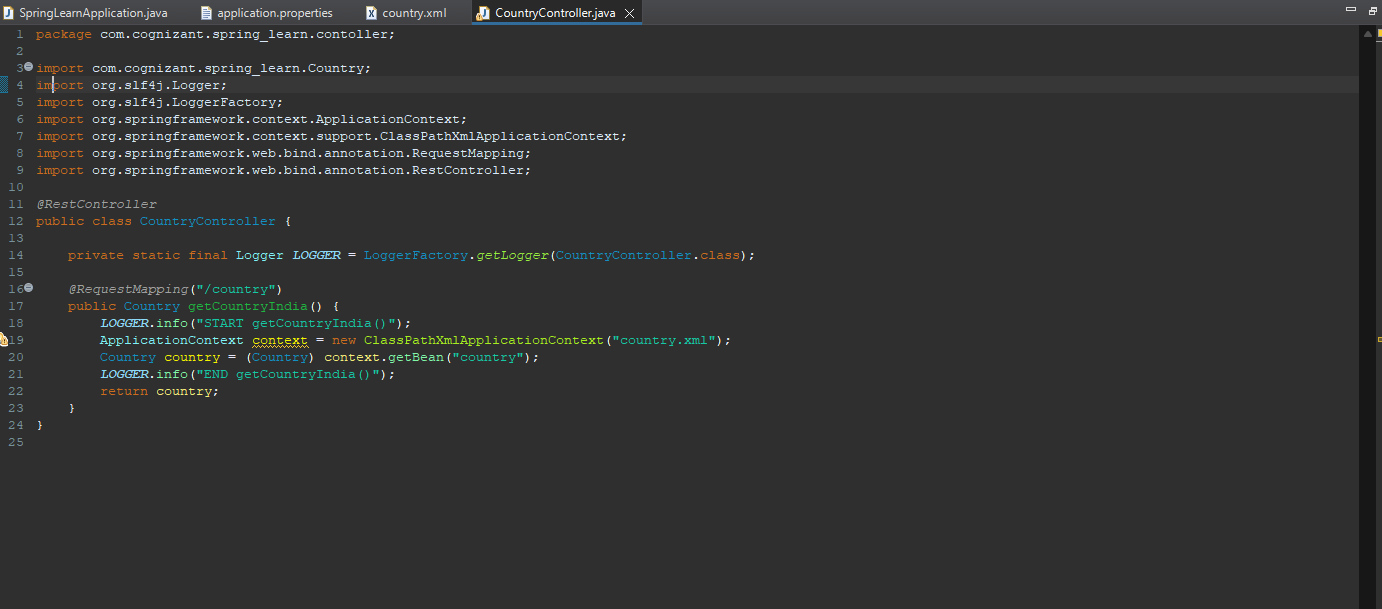
}

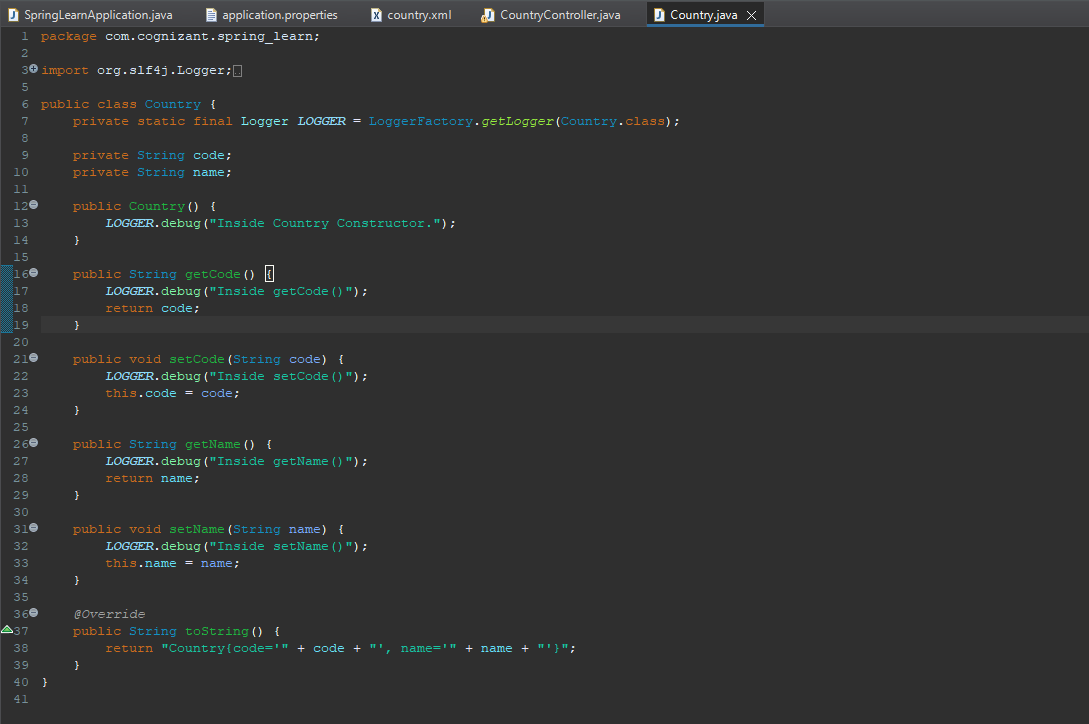
}

**Screenshot:**

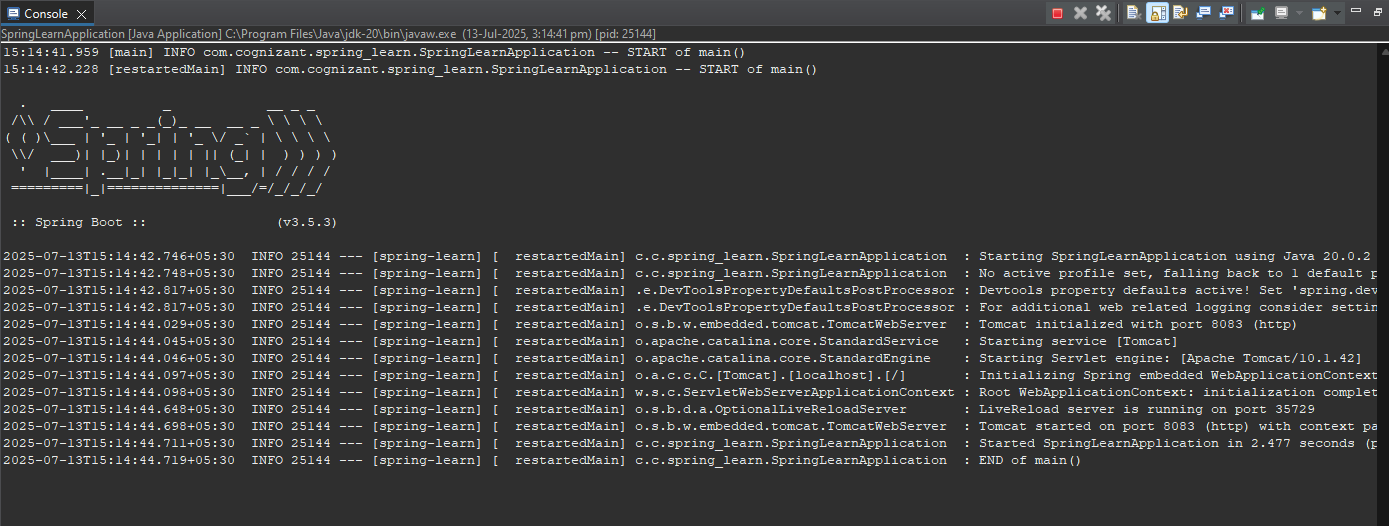
**Eclipse Screenshot:**



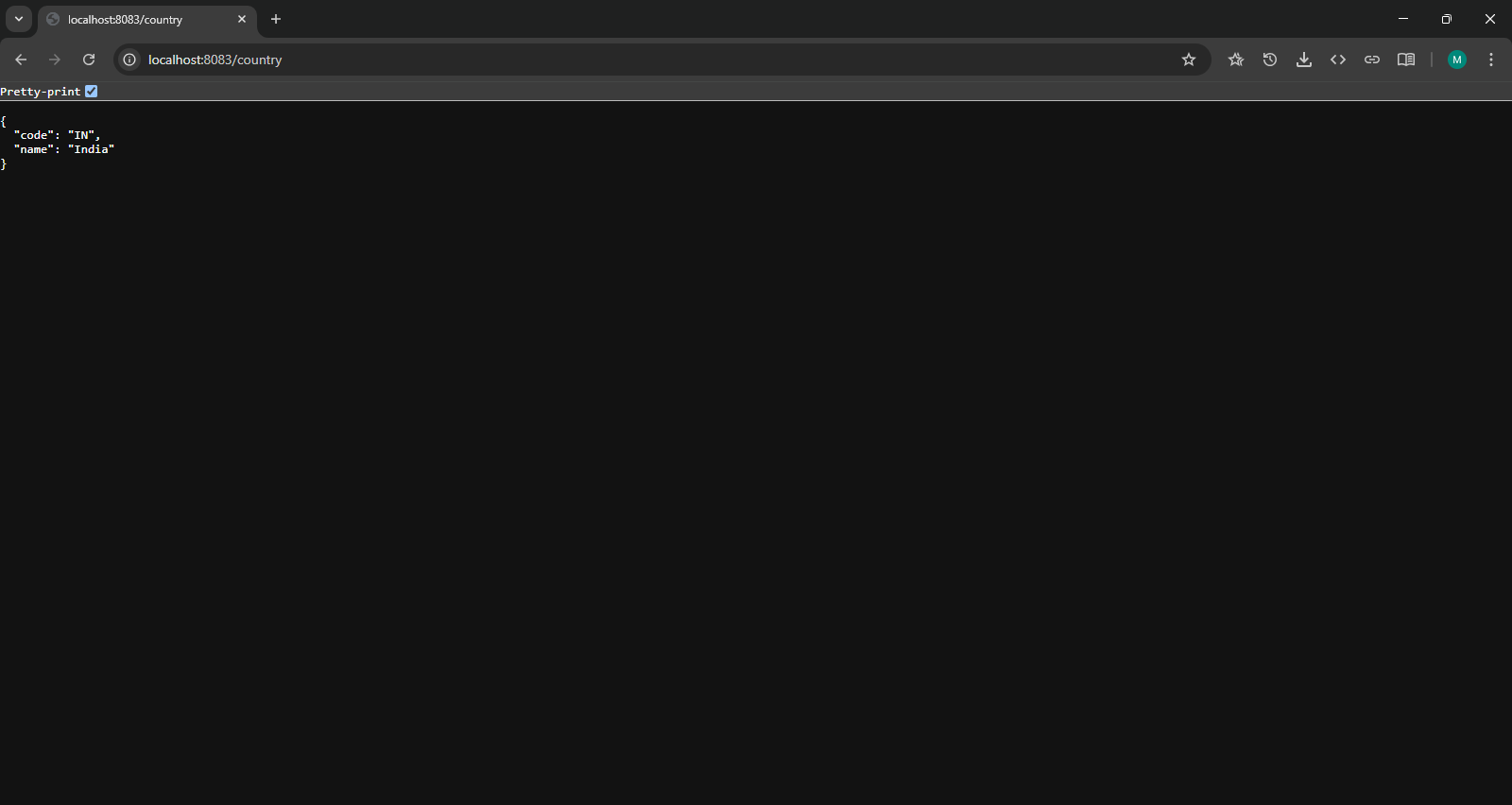




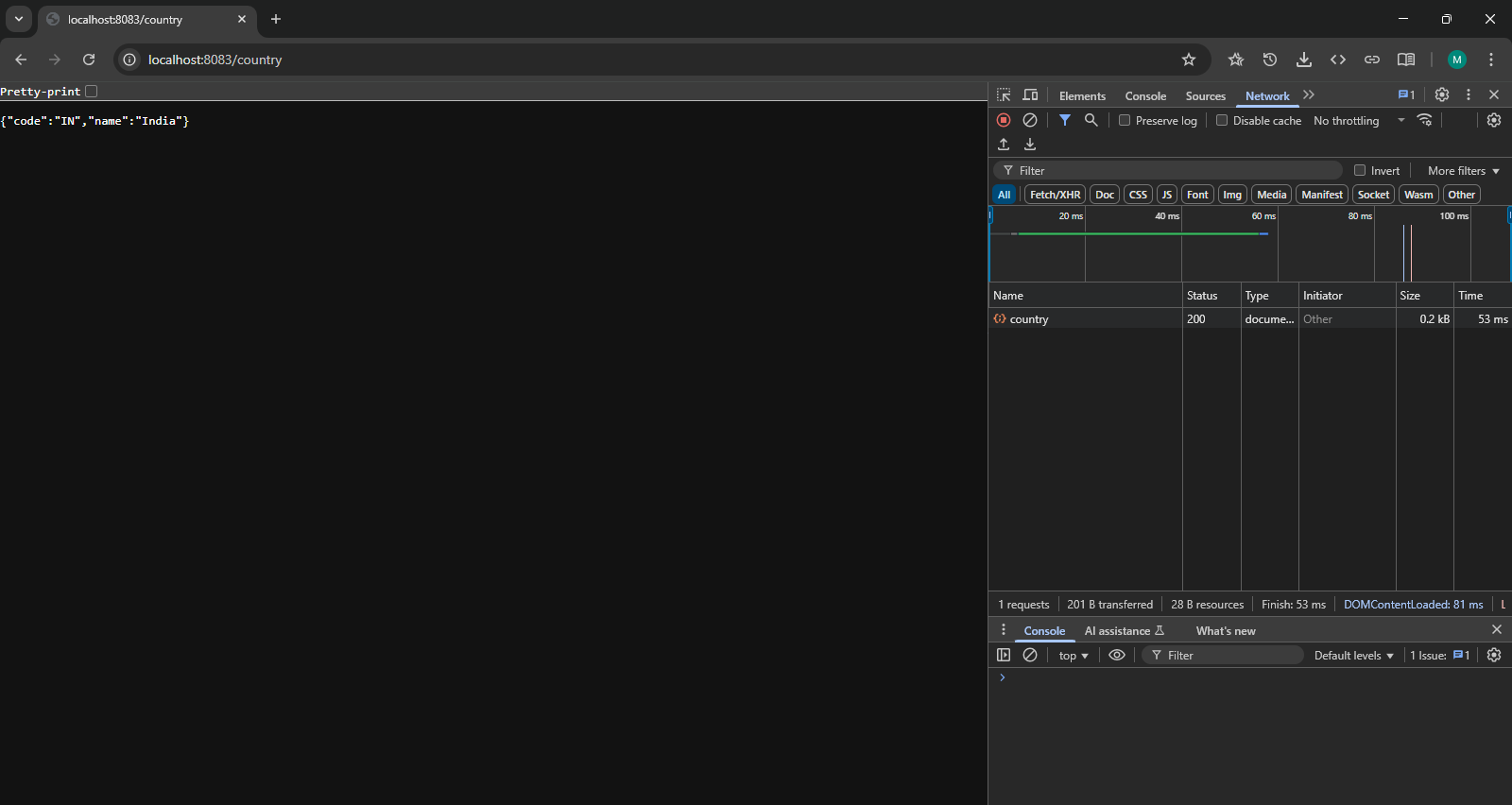
**Console output:**



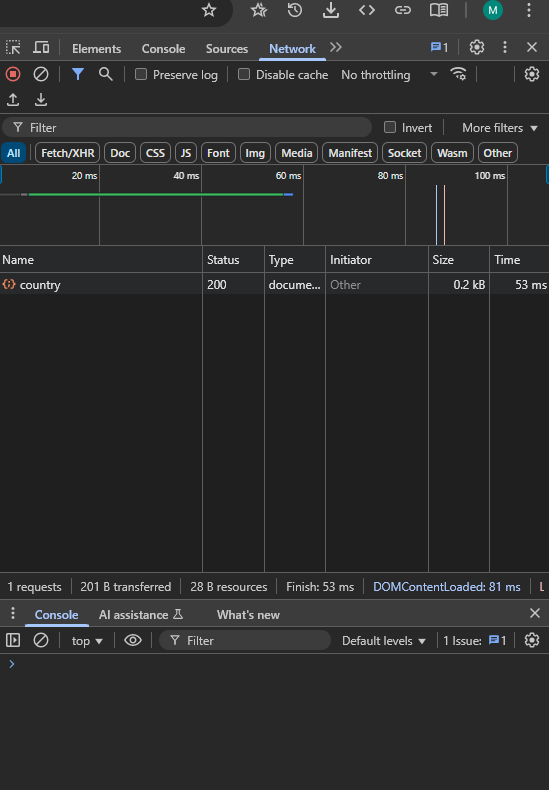
**Browser Output**



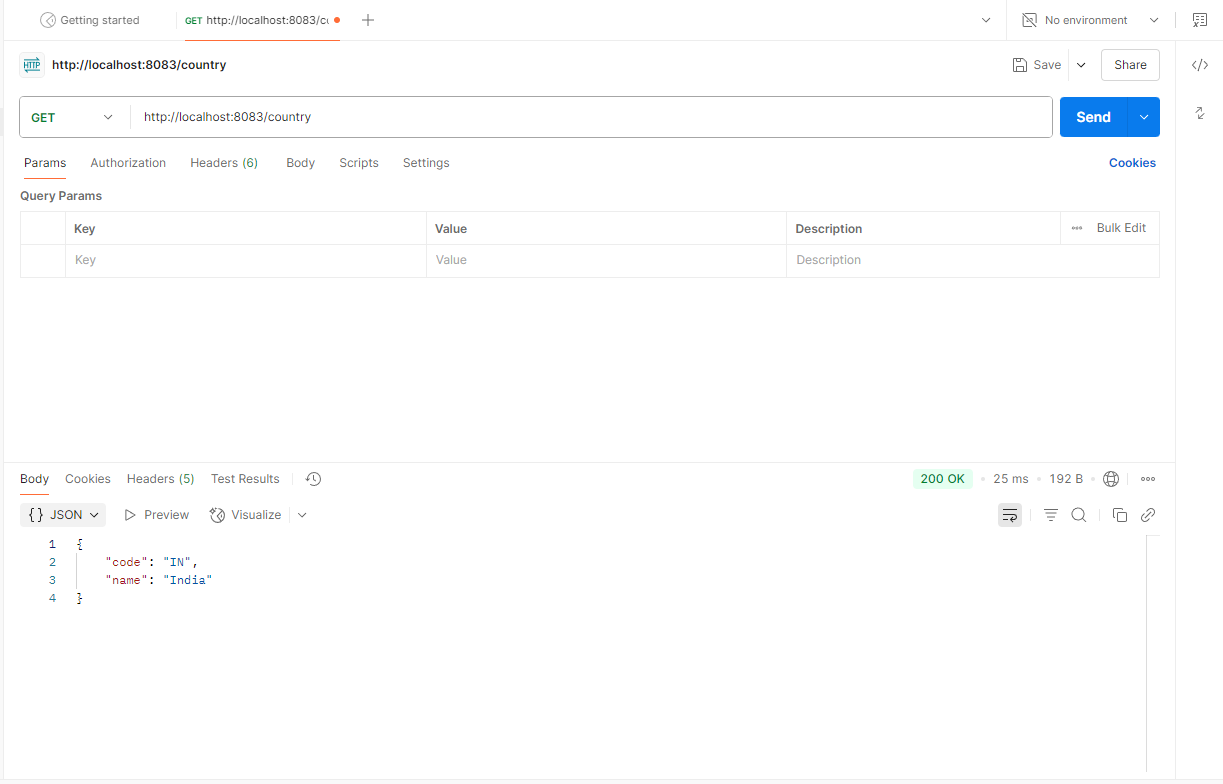
**Developer Tools – Network Tab (Chrome)**



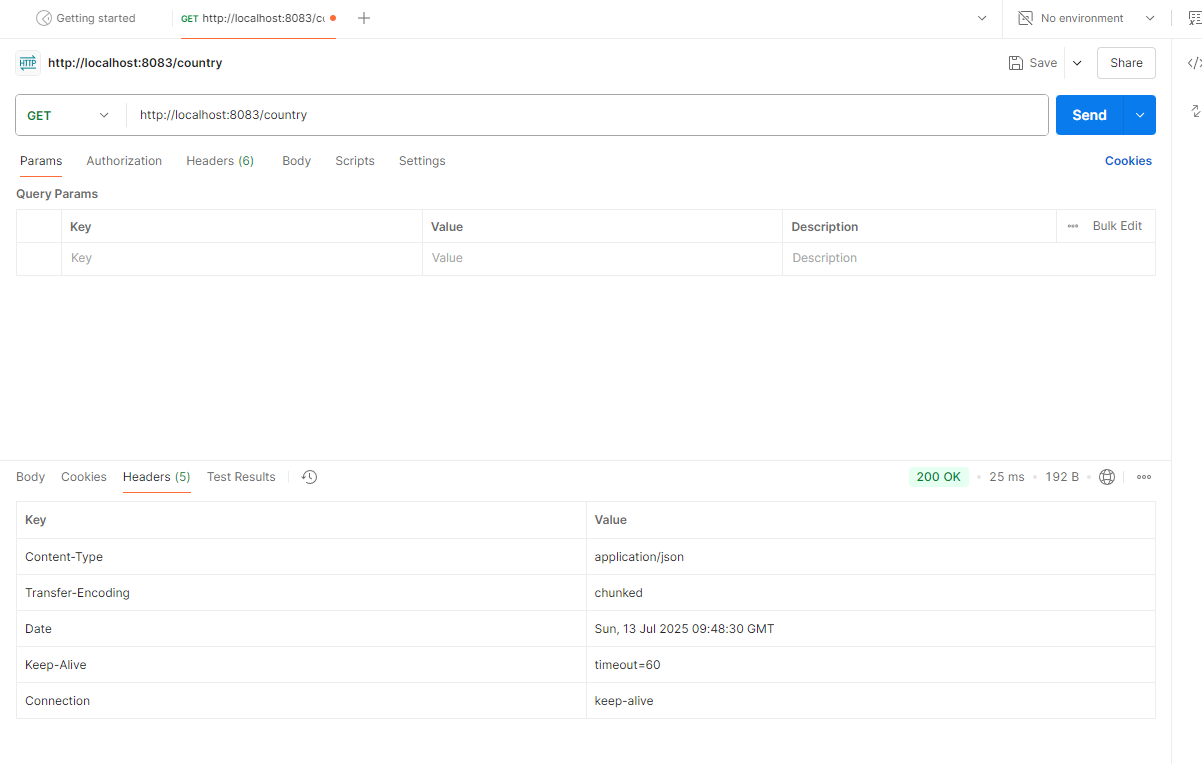
**Developer Tools – Network Tab (Chrome)**



**Postman Output**



**Postman Headers**



**Conclusion**

We created a REST service in Spring Boot that returns India’s country details from an XML-configured bean. The response was tested using both Chrome and Postman, and we understood how Spring handles JSON conversion automatically.

5.REST - Get country based on country code

**Task Description**

Create a Spring Boot REST API that returns a country based on the country code using an XML-configured bean list.

**Code:**

Country.java

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country {

private static final Logger LOGGER = LoggerFactory.getLogger(Country.class);

private String code;

private String name;

public Country() {

LOGGER.debug("Inside Country Constructor.");

}

public String getCode() {

LOGGER.debug("Inside getCode()");

return code;

}

public void setCode(String code) {

LOGGER.debug("Inside setCode()");

this.code = code;

}

public String getName() {

LOGGER.debug("Inside getName()");

return name;

}

public void setName(String name) {

LOGGER.debug("Inside setName()");

this.name = name;

}

@Override

public String toString() {

return "Country{code='" + code + "', name='" + name + "'}";

}

}

CountryController.java

package com.cognizant.spring\_learn.contoller;

import com.cognizant.spring\_learn.Country;

import com.cognizant.spring\_learn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

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

*@RestController*

public class CountryController {

private static final Logger ***LOGGER*** = LoggerFactory.*getLogger*(CountryController.class);

*@Autowired*

private CountryService countryService;

*@GetMapping*("/countries/{code}")

public Country getCountry(*@PathVariable* String code) {

***LOGGER***.info("START getCountry()");

Country country = countryService.getCountry(code);

***LOGGER***.info("END getCountry()");

return country;

}

}

**country.xml:**

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

<**beans** xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*https://www.springframework.org/schema/beans/spring-beans.xsd"*>

<**bean** id=*"country"* class=*"com.cognizant.spring\_learn.Country"*>

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

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

</**bean**>

<**bean** id=*"countryList"* class=*"java.util.ArrayList"*>

<**constructor-arg**>

<**list**>

<**bean** class=*"com.cognizant.spring\_learn.Country"*>

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

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

</**bean**>

<**bean** class=*"com.cognizant.spring\_learn.Country"*>

<**property** name=*"code"* value=*"US"*/>

<**property** name=*"name"* value=*"United States"*/>

</**bean**>

<**bean** class=*"com.cognizant.spring\_learn.Country"*>

<**property** name=*"code"* value=*"CN"*/>

<**property** name=*"name"* value=*"China"*/>

</**bean**>

<**bean** class=*"com.cognizant.spring\_learn.Country"*>

<**property** name=*"code"* value=*"JP"*/>

<**property** name=*"name"* value=*"Japan"*/>

</**bean**>

</**list**>

</**constructor-arg**>

</**bean**>

</**beans**>

CountryService.java

package com.cognizant.spring\_learn.service;

import com.cognizant.spring\_learn.Country;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

import java.util.List;

*@Service*

public class CountryService {

public Country getCountry(String code) {

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

List<Country> countries = (List<Country>) context.getBean("countryList");

return countries.stream()

.filter(c -> c.getCode().equalsIgnoreCase(code))

.findFirst()

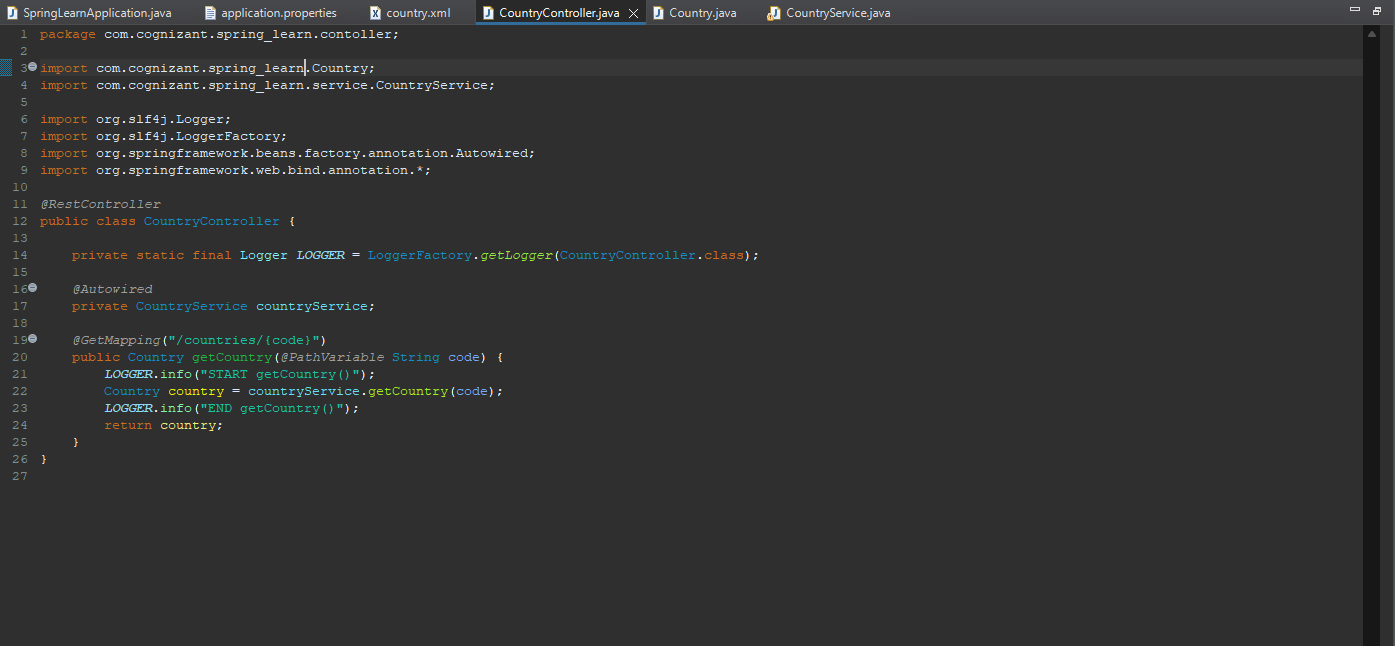
.orElse(null);

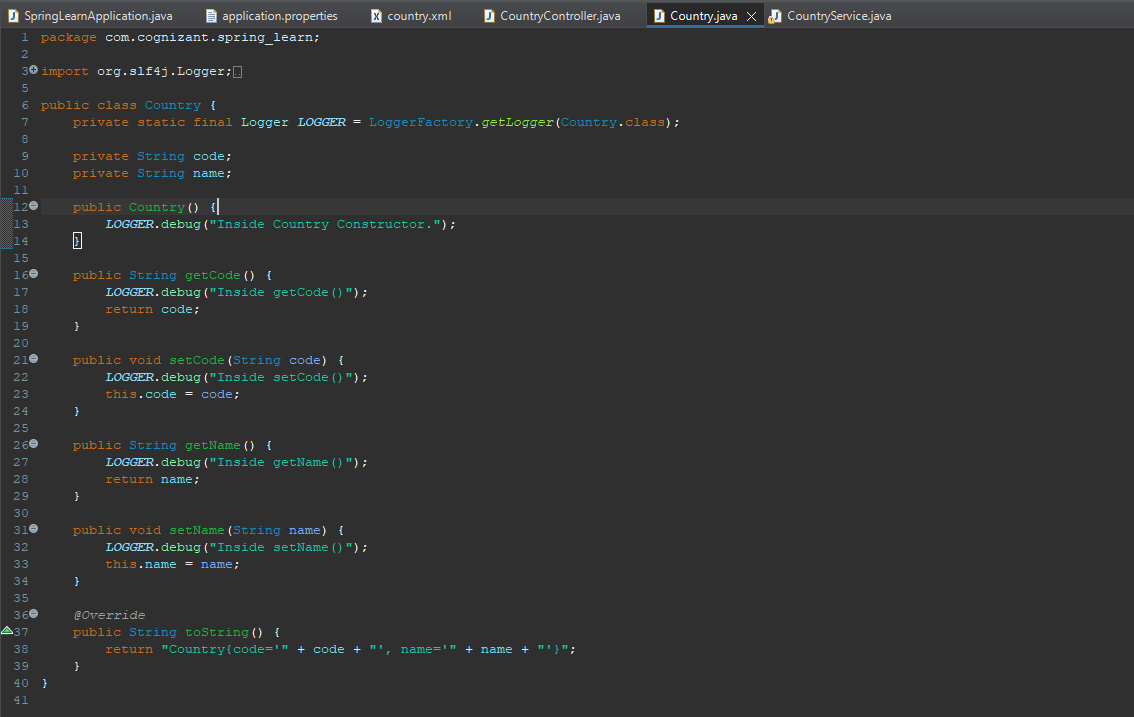
}

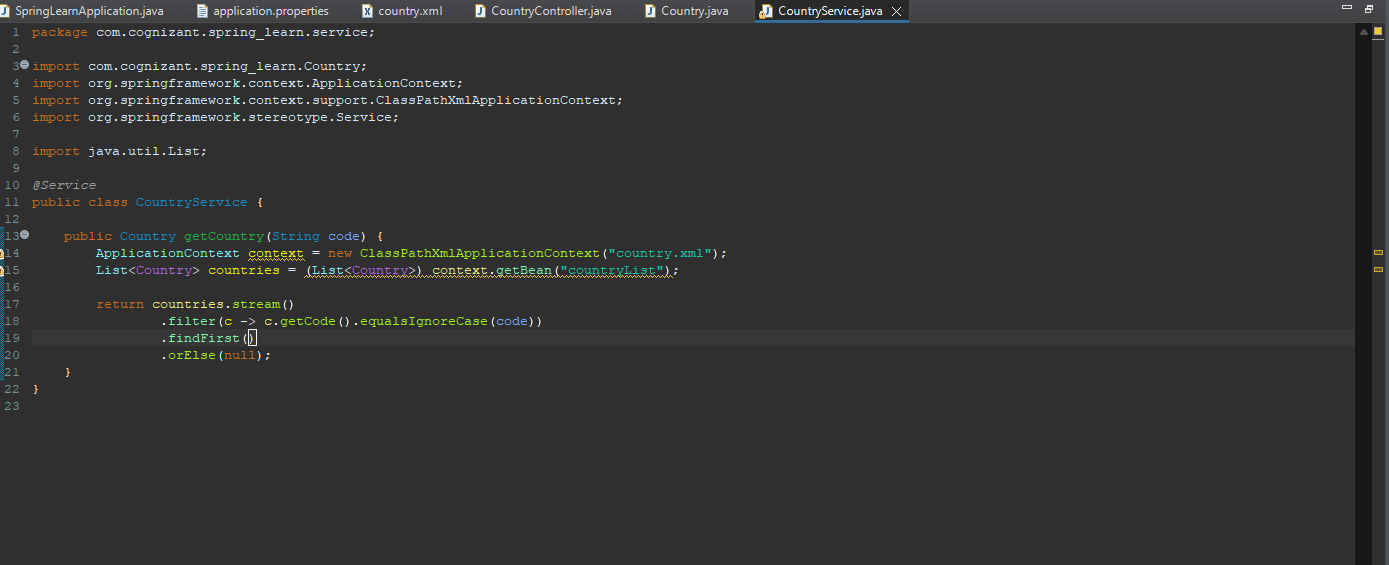
}

**Screenshot:**

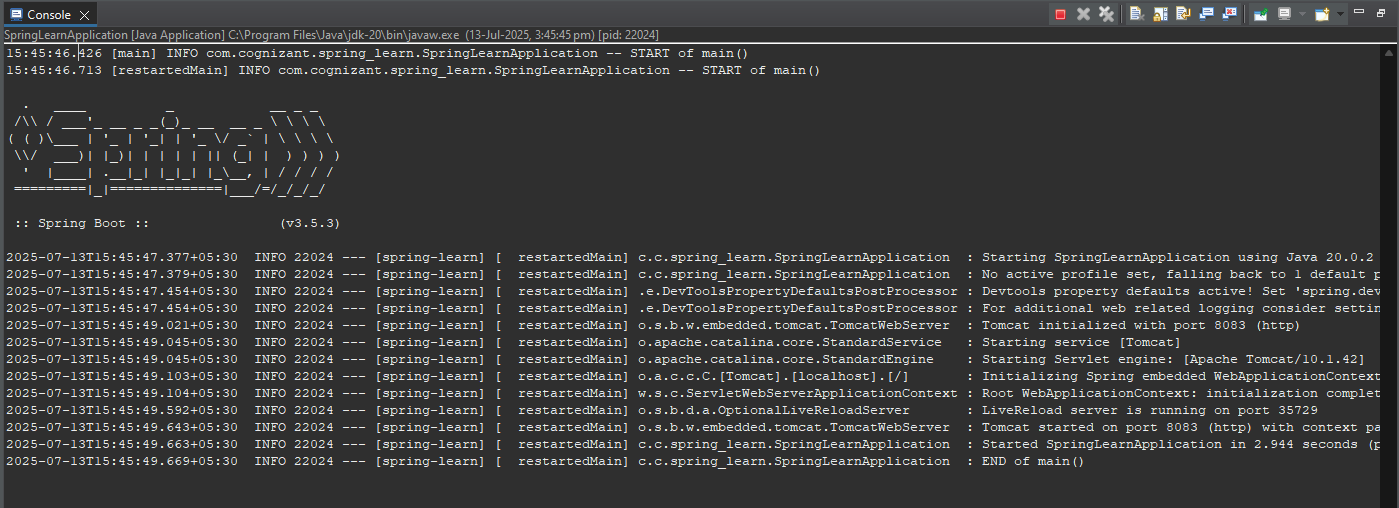
Eclipse code screenshot:





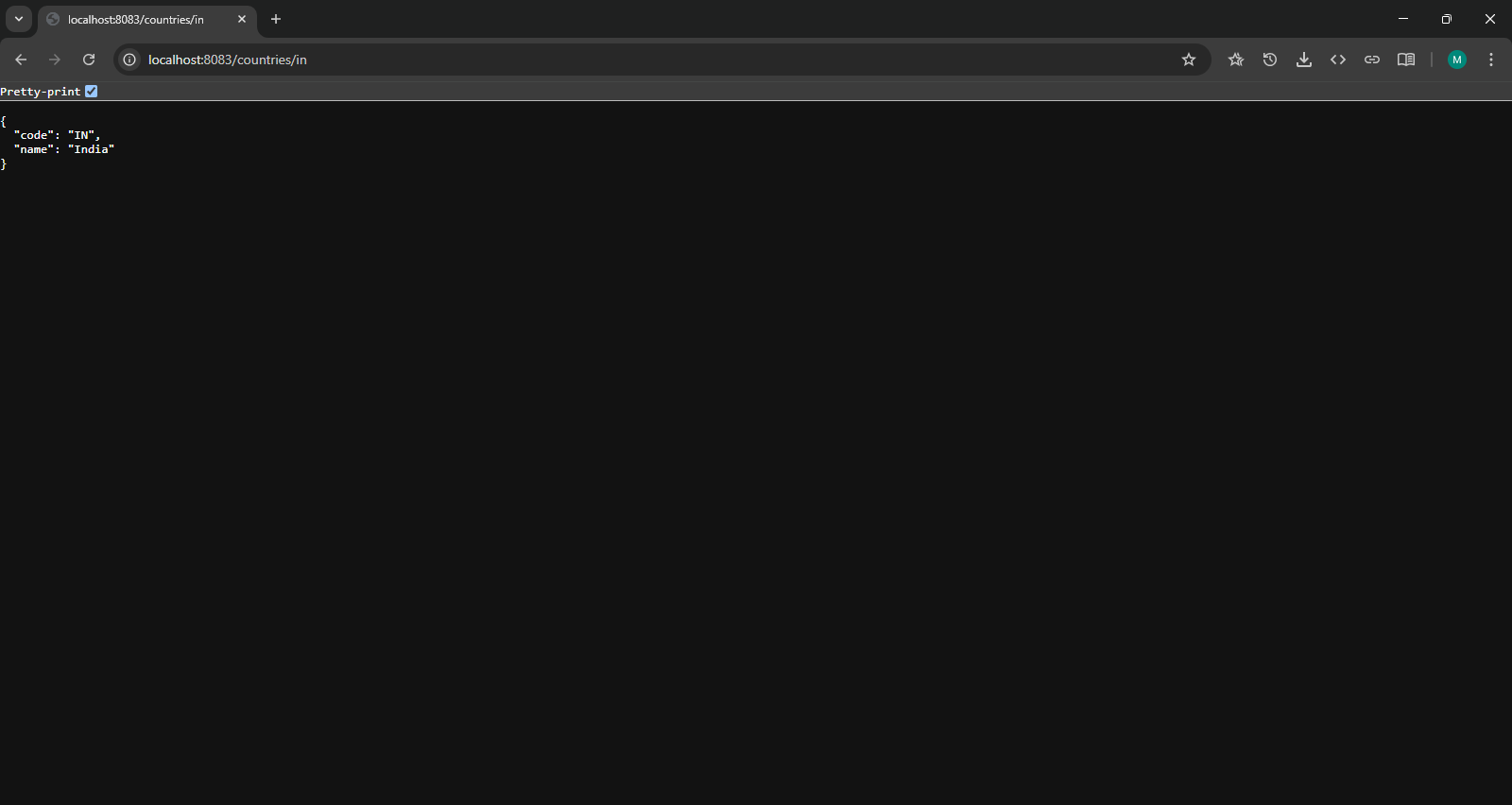


Console output:

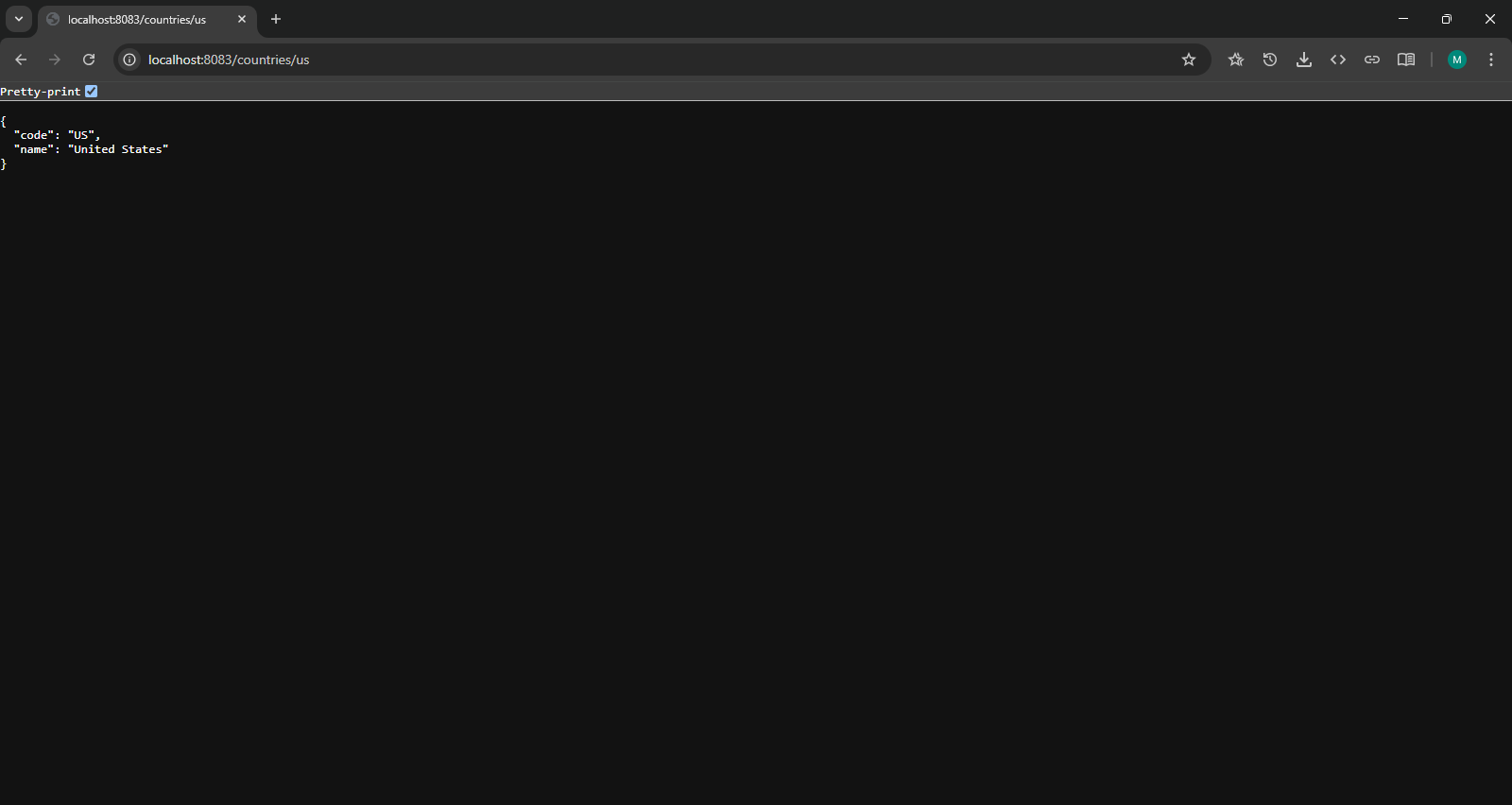


**Output in browser:**

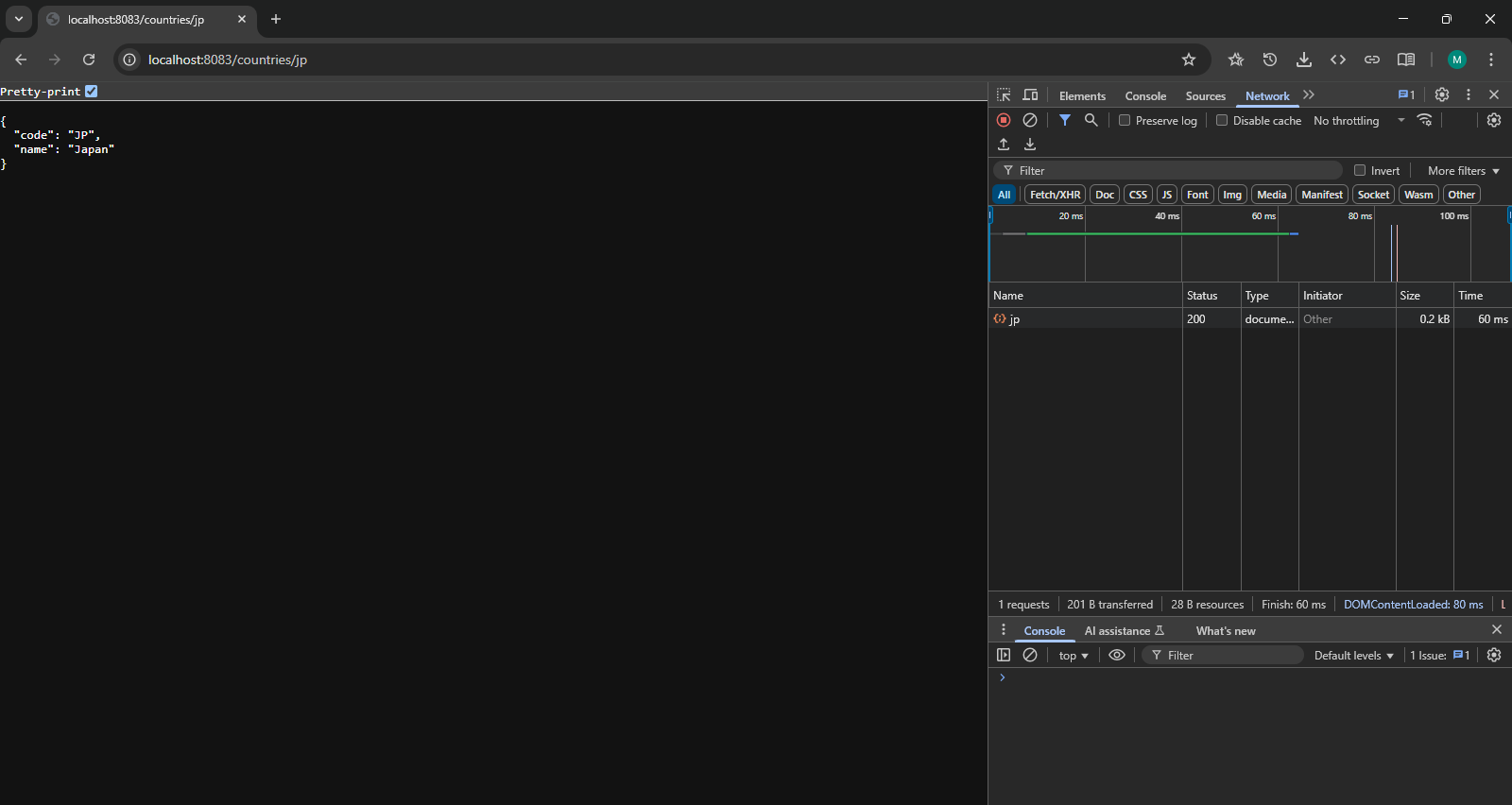
**Country API Response for India (/countries/in)**



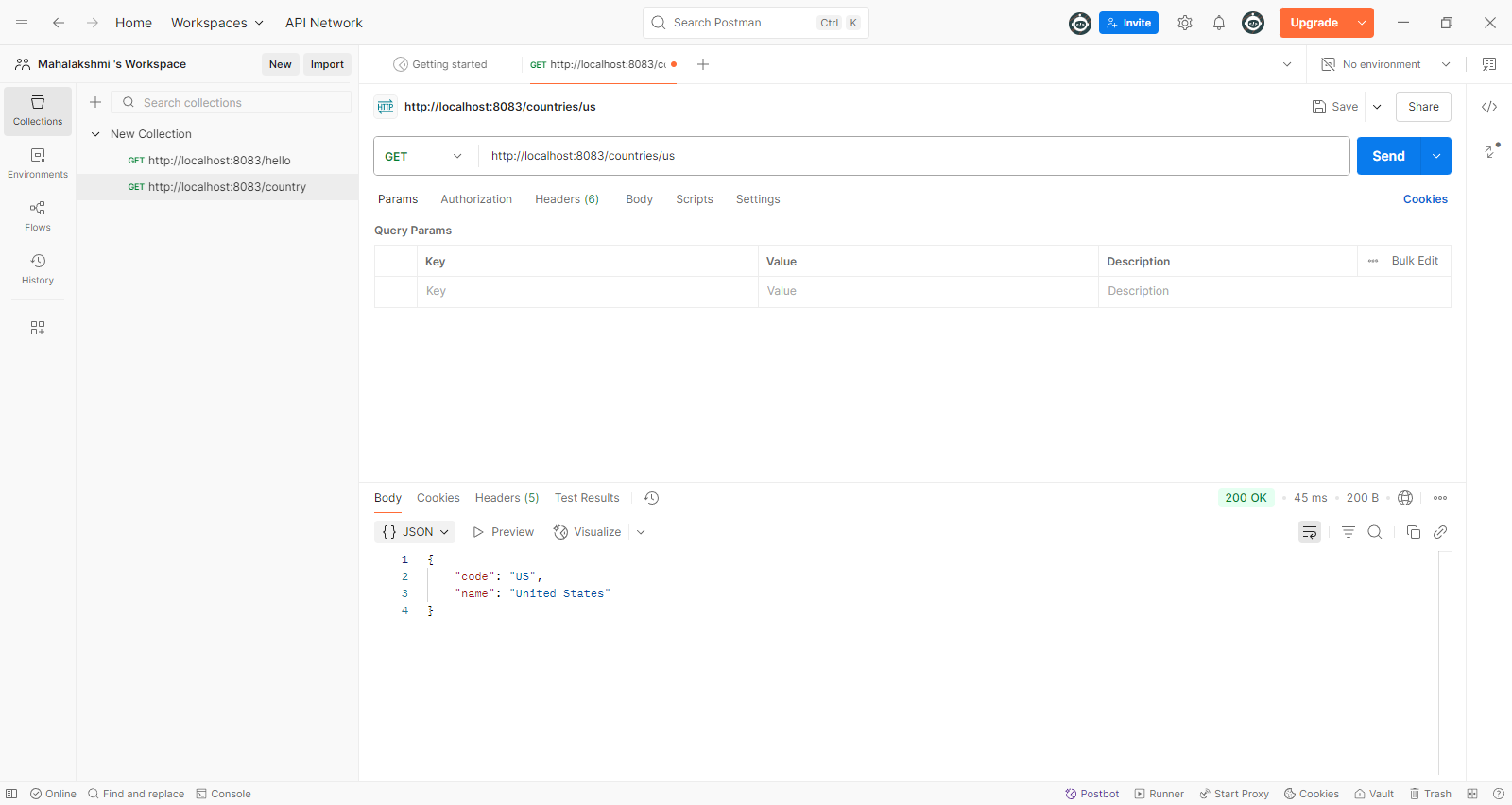
Country API Response for United States (/countries/us)



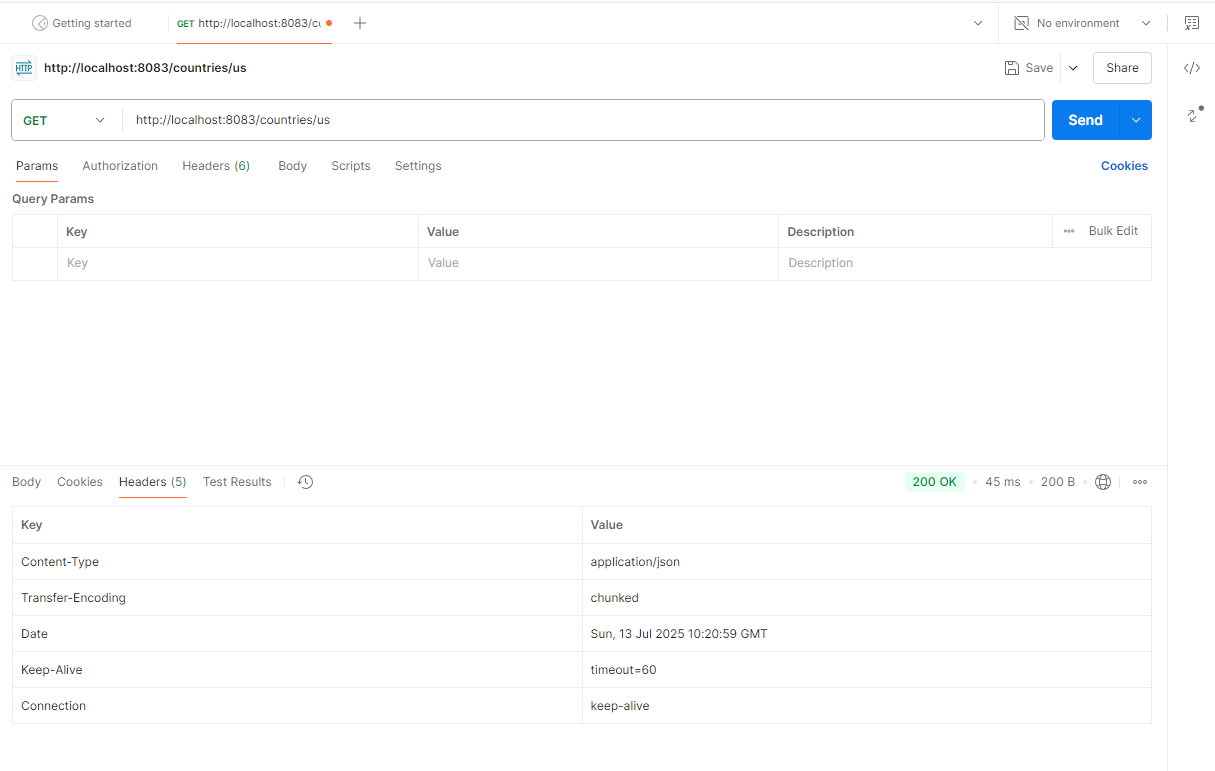
**Developer Tools → Network Tab → Headers:**



**Postman Output:**



**Postman → Headers Tab**



**Conclusion:**

This exercise demonstrated how to pass parameters using URL path variables in Spring Boot REST services, how to load XML-configured beans, and how Spring auto-converts Java objects to JSON format for API responses.

**JWT-handson**

**6.Create authentication service that returns JWT:**

To create a Spring Boot-based RESTful authentication service that validates user credentials using Basic Auth and returns a JWT token.

**Code:**

**JwtUtil.java:**

package com.cognizant.jwtauth.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

import org.springframework.stereotype.Component;

import java.util.Date;

import javax.crypto.SecretKey;

import java.nio.charset.StandardCharsets;

*@Component*

public class JwtUtil {

private static final String ***SECRET\_KEY\_STRING*** = "mySuperSecretKeyThatIsVerySecure123456"; // at least 32 characters

private static final SecretKey ***SECRET\_KEY*** = Keys.*hmacShaKeyFor*(***SECRET\_KEY\_STRING***.getBytes(StandardCharsets.***UTF\_8***));

private static final long ***EXPIRATION\_TIME*** = 1000 \* 60 \* 10; // 10 minutes

public String generateToken(String username) {

return Jwts.*builder*()

.setSubject(username)

.setIssuedAt(new Date(System.*currentTimeMillis*()))

.setExpiration(new Date(System.*currentTimeMillis*() + ***EXPIRATION\_TIME***))

.signWith(***SECRET\_KEY***, *SignatureAlgorithm*.***HS256***)

.compact();

}

}

**SecurityConfig.java**

package com.cognizant.jwtauth.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.Customizer;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.web.SecurityFilterChain;

*@Configuration*

public class SecurityConfig {

*@Bean*

public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

http

.csrf(csrf -> csrf.disable()) // disable CSRF for simplicity

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll() // ✅ allow /authenticate for everyone

.anyRequest().authenticated() // 🔐 all other endpoints require auth

)

.httpBasic(Customizer.*withDefaults*()); // ✅ enable basic auth

return http.build();

}

}

**AuthenticationController.java**

package com.cognizant.jwtauth.controller;

import com.cognizant.jwtauth.util.JwtUtil;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.core.annotation.AuthenticationPrincipal;

import org.springframework.security.core.userdetails.UserDetails;

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

import java.util.Collections;

*@RestController*

public class AuthenticationController {

*@Autowired*

private JwtUtil jwtUtil;

*@GetMapping*("/authenticate")

public ResponseEntity<?> authenticate(*@AuthenticationPrincipal* UserDetails userDetails) {

if (userDetails == null) {

return ResponseEntity.*status*(401).body("Unauthorized");

}

String token = jwtUtil.generateToken(userDetails.getUsername());

return ResponseEntity.*ok*(Collections.*singletonMap*("token", token));

}

}

**JwtauthApplication.java**

package com.cognizant.jwtauth;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JwtauthApplication {

public static void main(String[] args) {

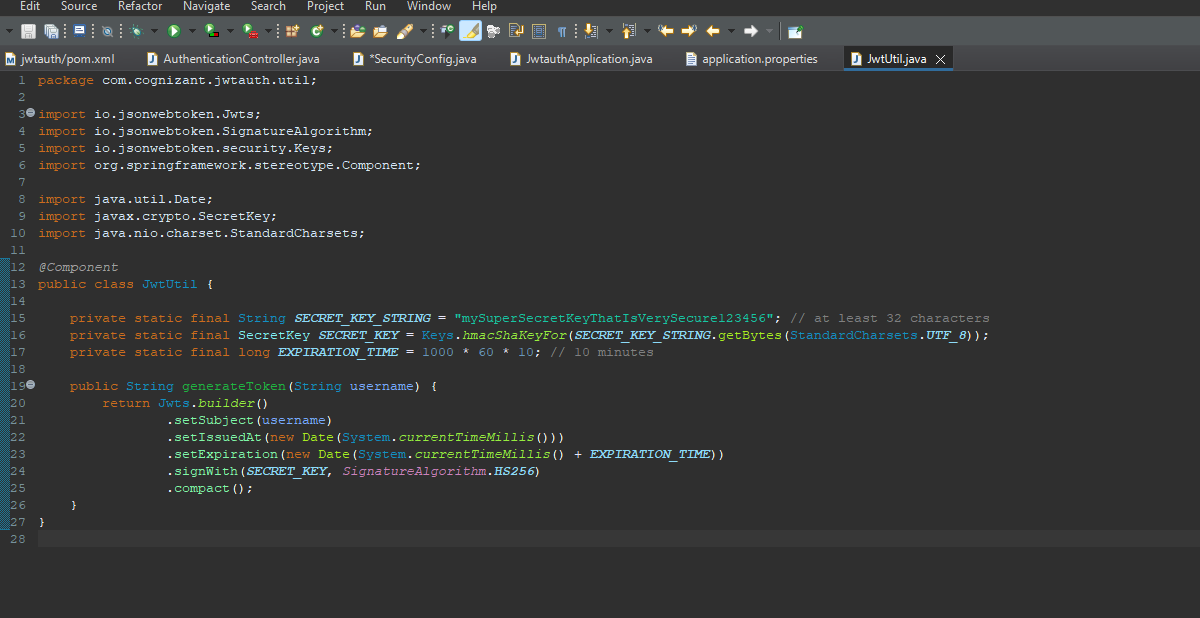
SpringApplication.run(JwtauthApplication.class, args);

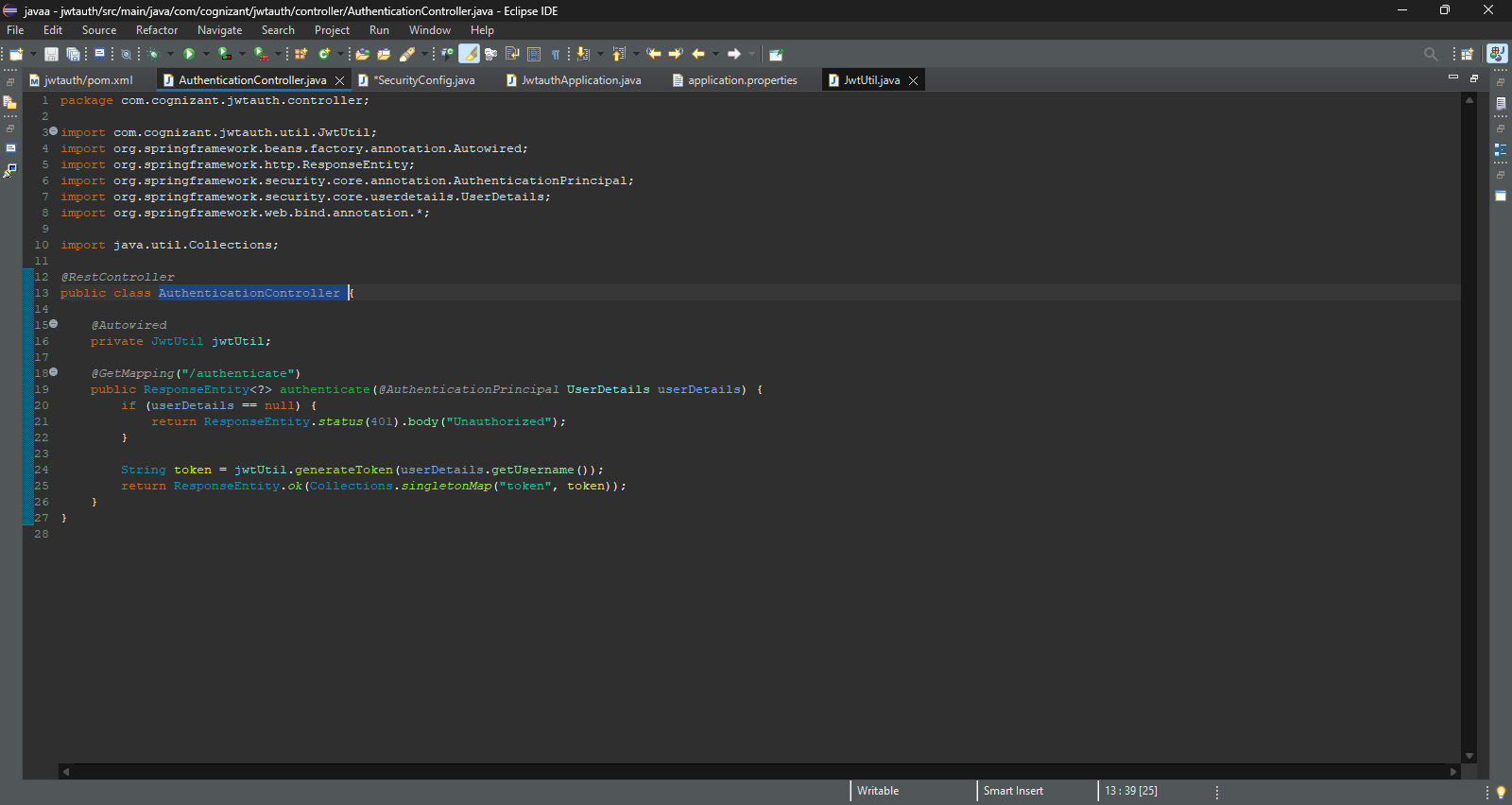
}

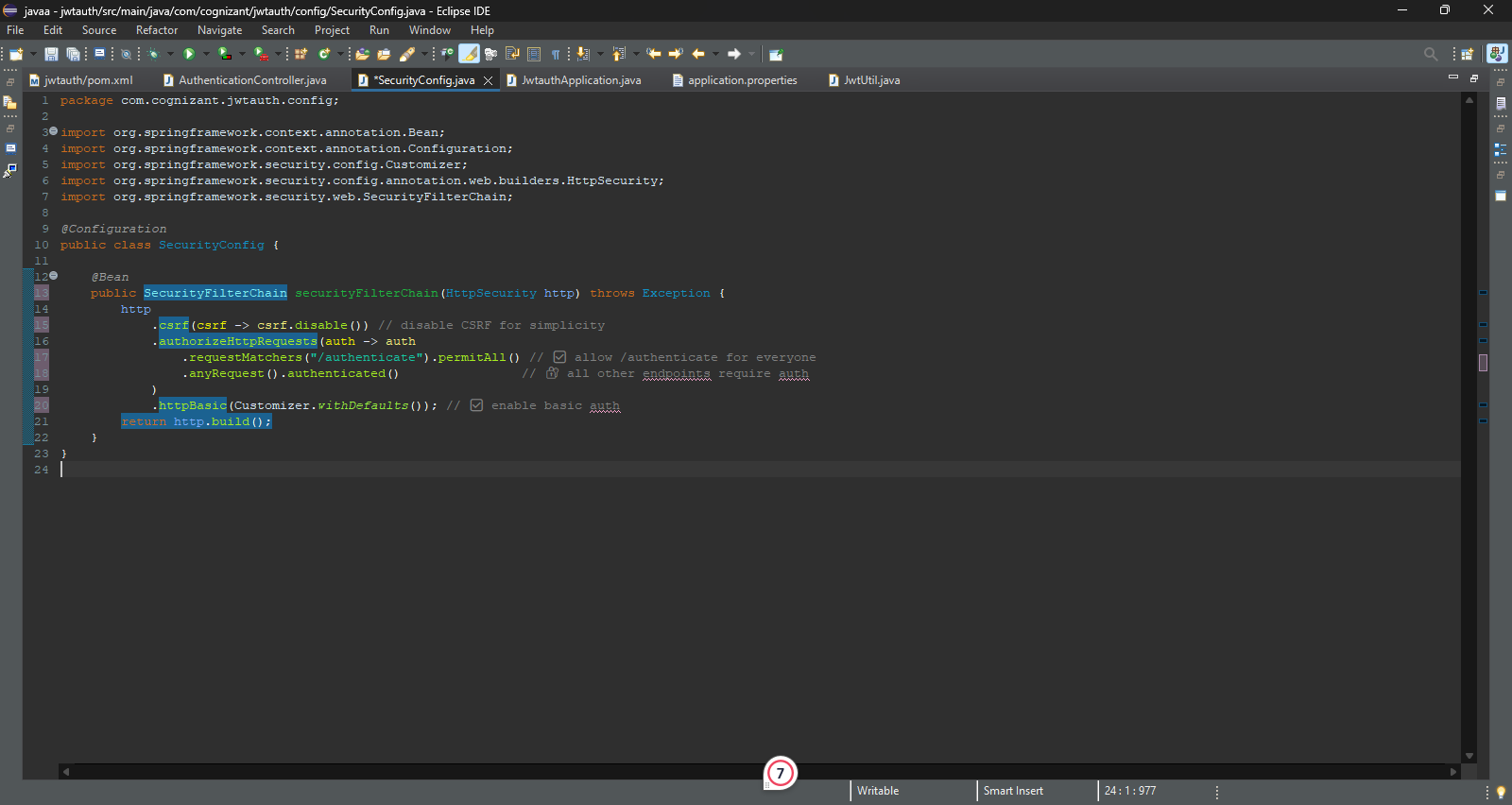
}

**Screenshots:**

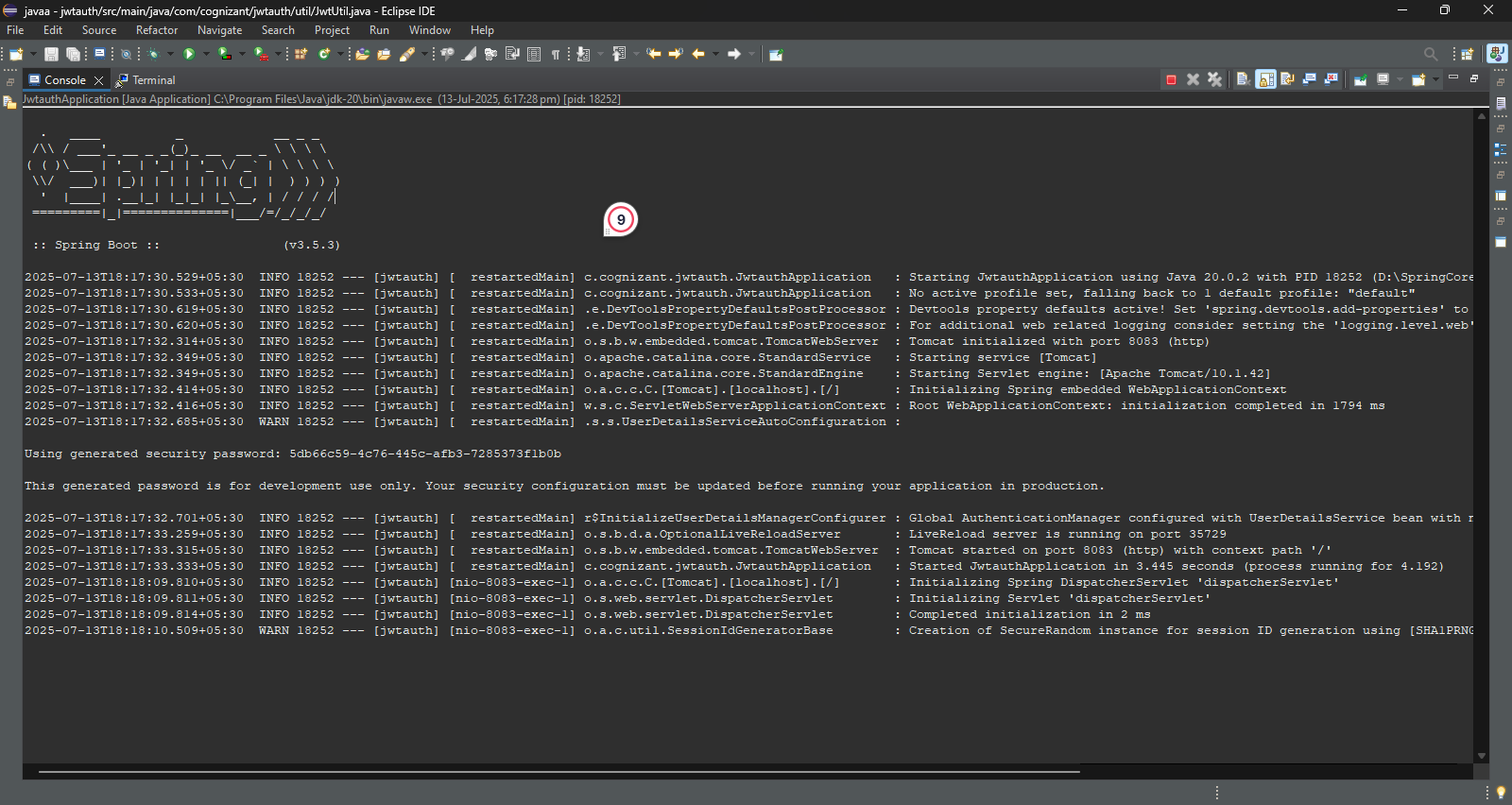
**Eclipse Code Screenshots:**







**CONSOLE OUTPUT:**



**COMMAND:**

curl -s -u user:pwd <http://localhost:8083/authenticate>

**OUTPUT:**

{

"token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNzI4ODA2MDAwLCJleHAiOjE3Mjg4MDY2MDB9.Sd9j3LkX5tBbJwJq8HdY\_xK7LgfK0ZV4Q9Fc8FyYJdU"

}

**CONCLUSION:**

In this task, we successfully implemented a basic JWT Authentication service using Spring Boot. The application validates user credentials provided via Basic Authentication.