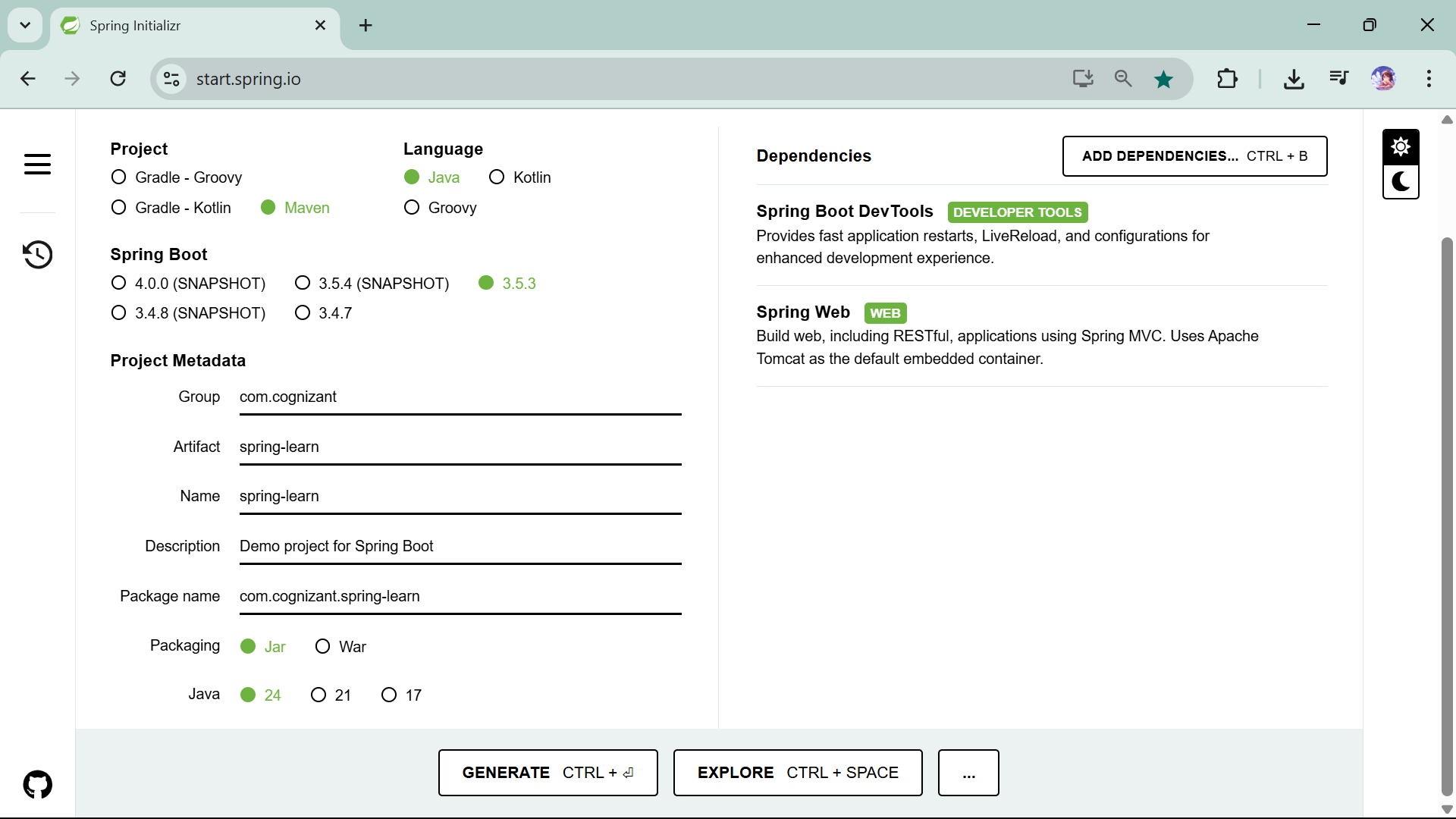
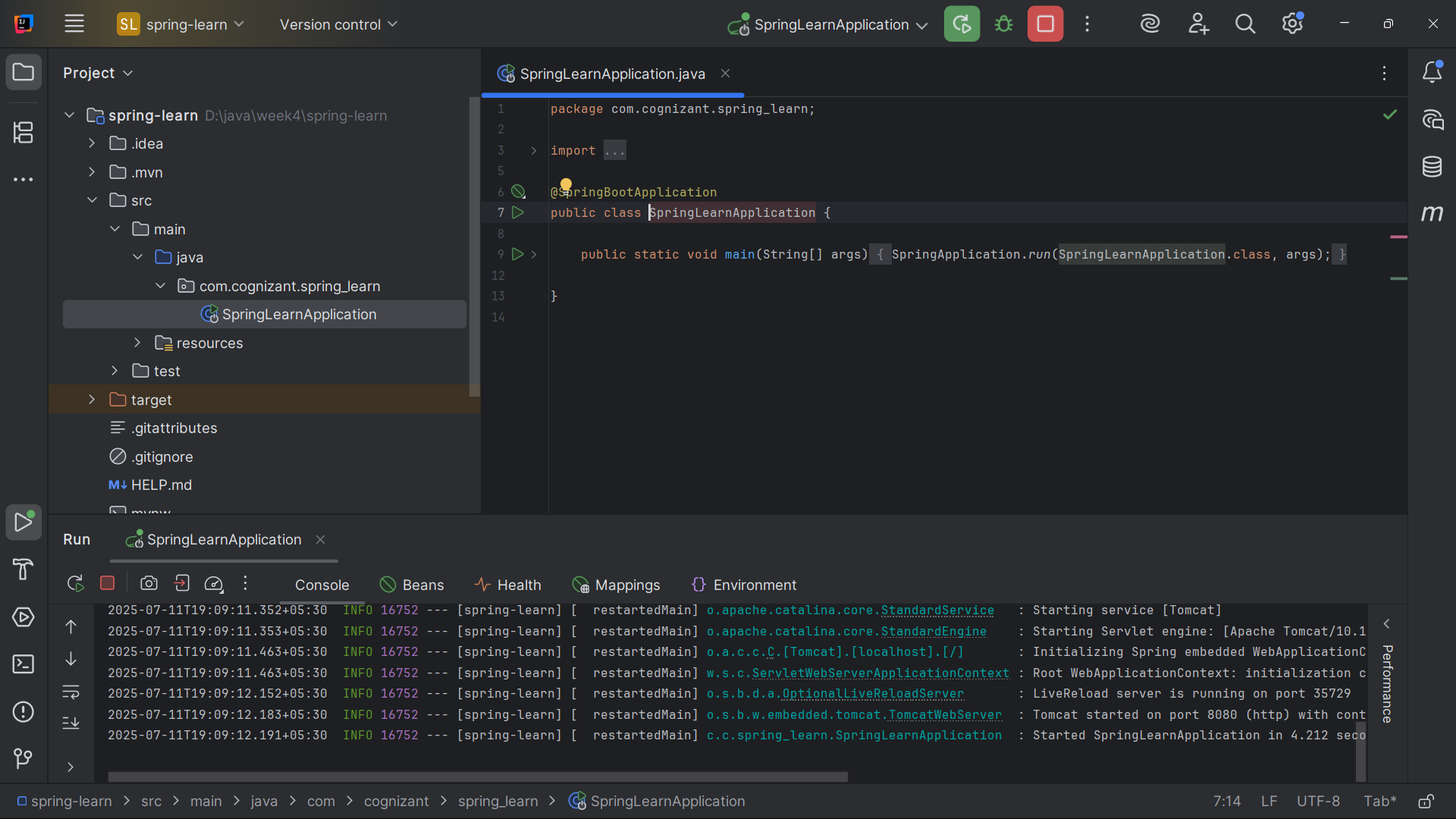
**Spring REST using Spring Boot 3**

**Create a Spring Web Project using Maven**

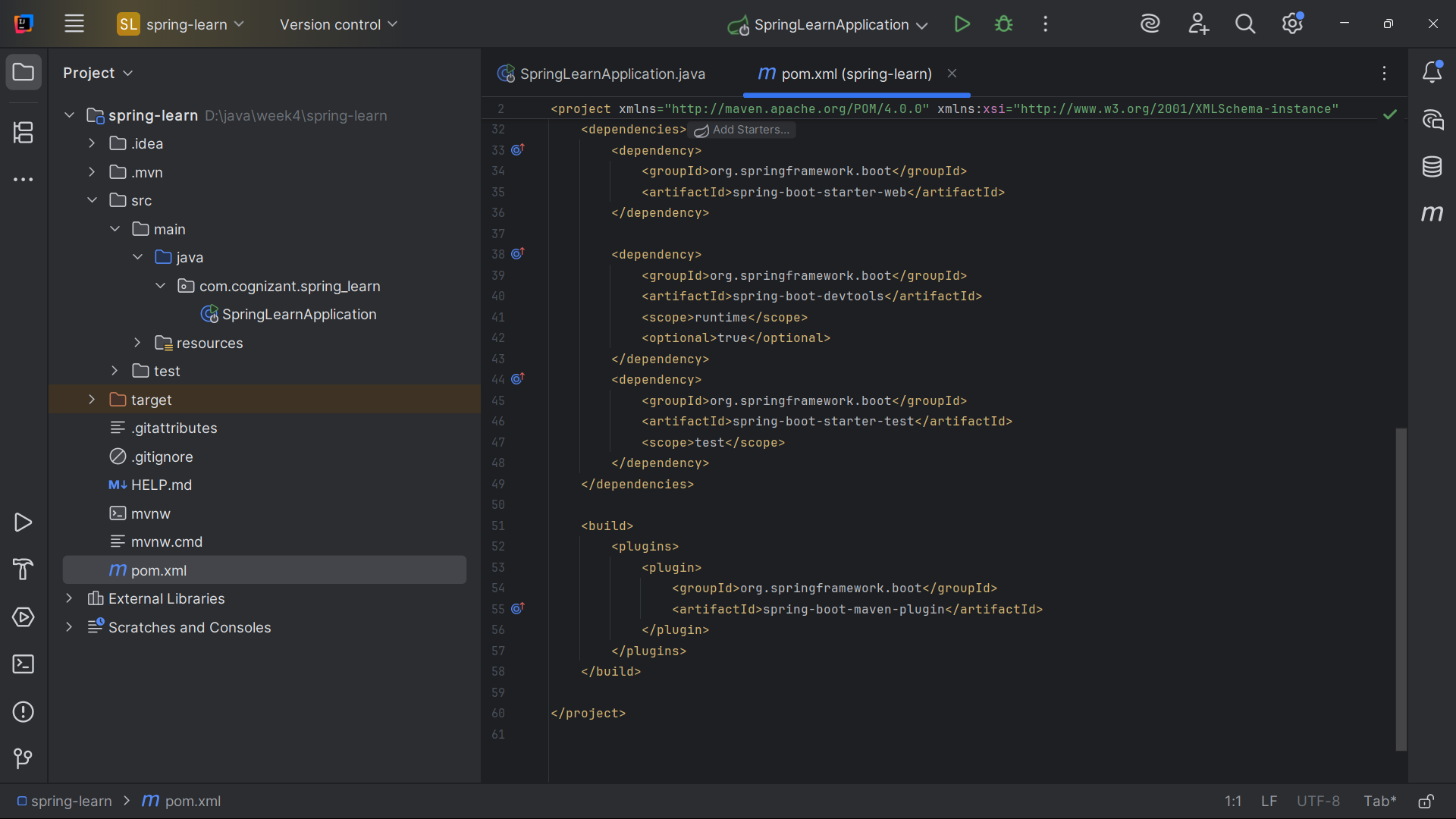
//Creating Project on start.spring.io



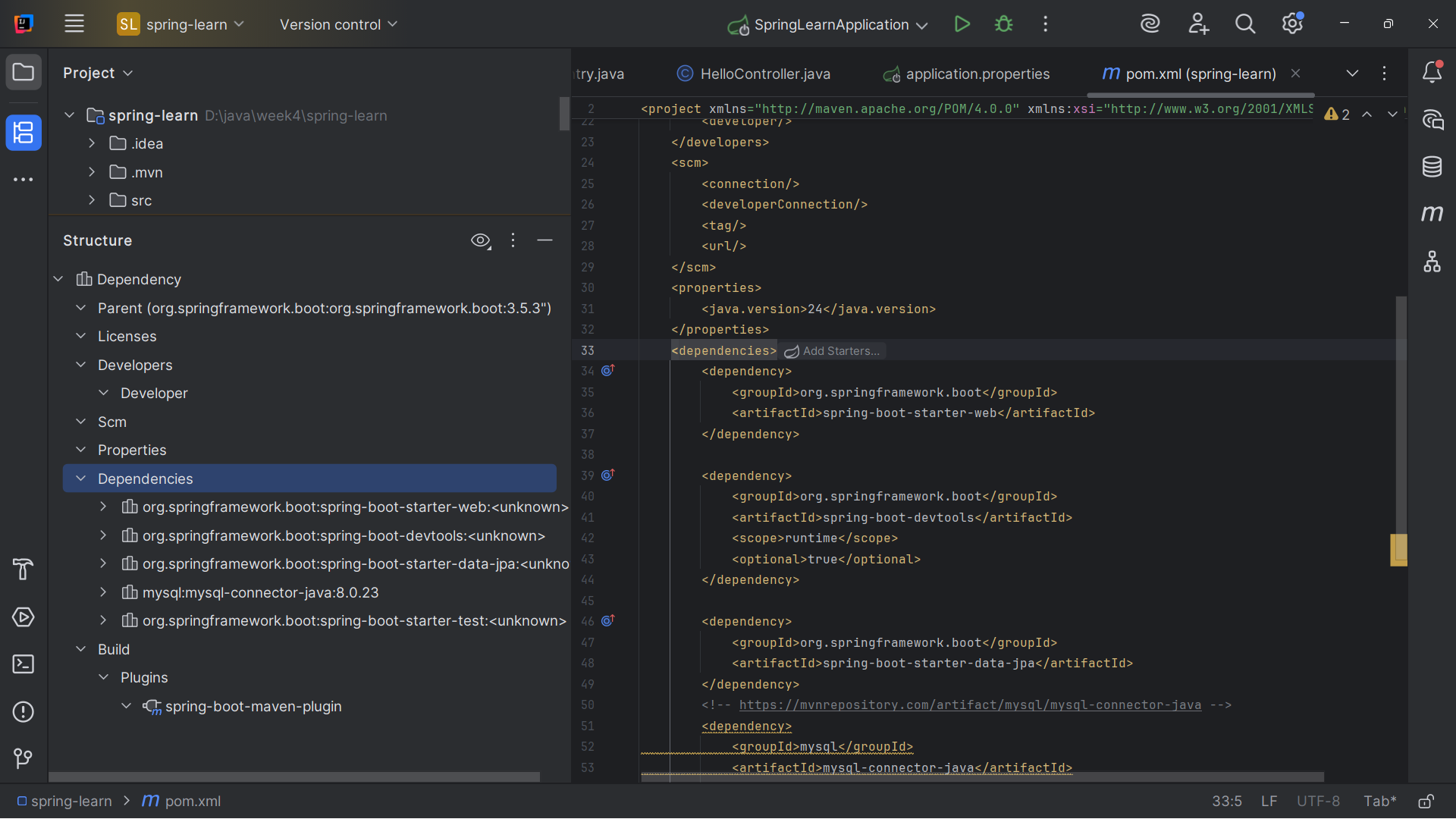
//SpringLearnApplication



//pom.xml



//Dependency Hierarchy



Purpose of @SpringBootApplication annotation:

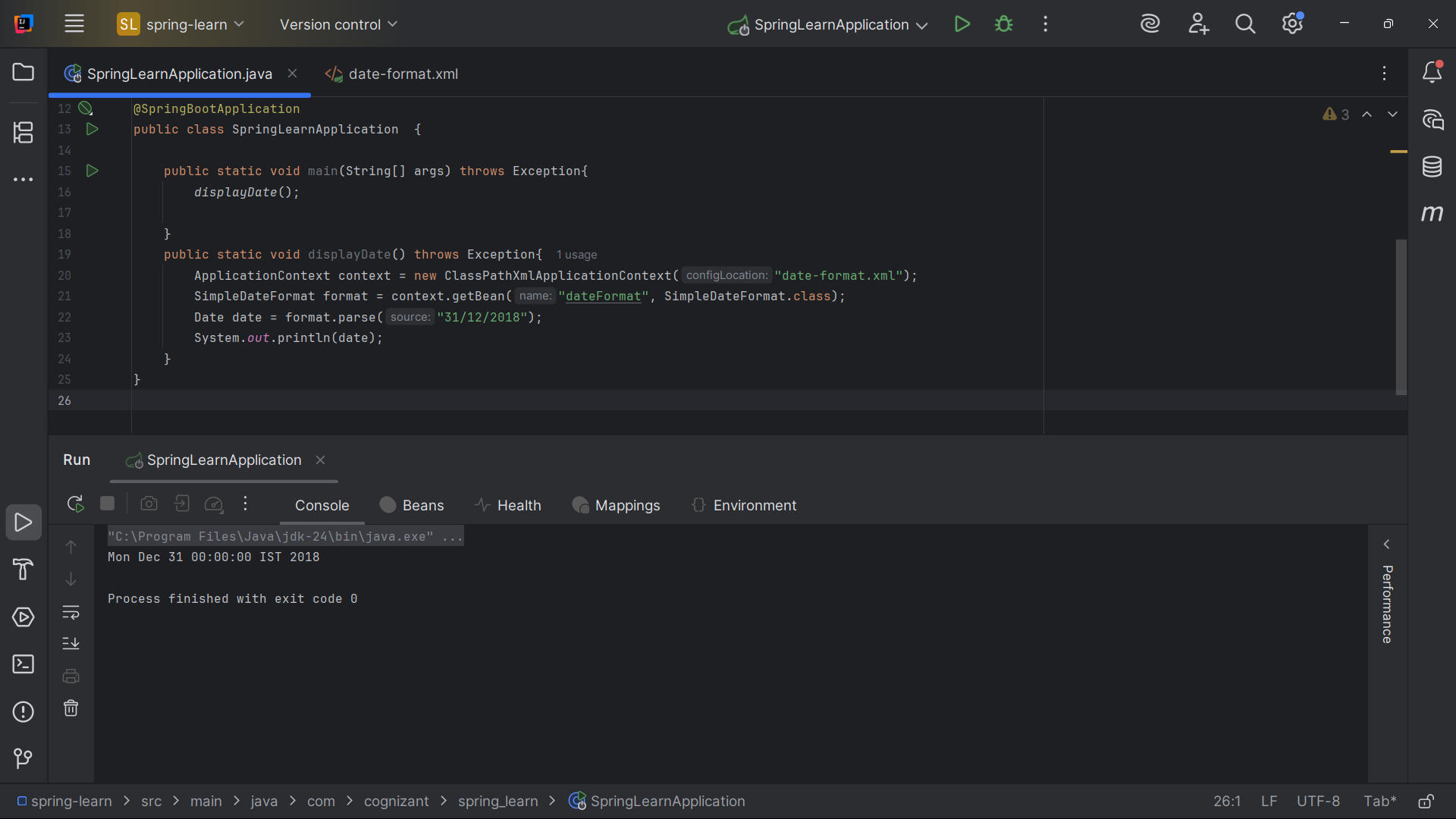
* It serves as a central convenience annotation in Spring Boot, simplifying the setup and configuration of a Spring Boot application
* It essentially combines three key annotations into a single, powerful annotation:
  + @Configuration: This indicates that the annotated class is a source of bean definitions for the Spring application context. It allows you to define and configure Spring beans within this class.
  + @EnableAutoConfiguration:This enables Spring Boot's auto-configuration mechanism. Spring Boot automatically configures your application based on the dependencies present in your classpath. For example, if you have Spring Web MVC in your classpath, it will automatically configure a DispatcherServlet.
  + @ComponentScan:This enables component scanning, which automatically discovers and registers Spring components (like @Component, @Service, @Repository, @Controller, etc.) as beans in the Spring application context. By default, it scans the package where the @SpringBootApplication class is located and its sub-packages.

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

**// SpringLearnApplication Class**

package com.cognizant.spring\_learn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 public static void main(String[] args) throws Exception{  
 *displayDate*();  
  
 }  
 public static void displayDate() throws Exception{  
 ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
 SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
 Date date = format.parse("31/12/2018");  
 System.*out*.println(date);  
 }  
}

**OUTPUT**



**Hello World RESTful Web Service**

**//HelloController class**

package com.cognizant.spring\_learn.controller;  
  
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 *log* = LoggerFactory.*getLogger*(HelloController.class);  
 @GetMapping("/hello")  
 public String sayHello(){  
 *log*.info("Start");  
 String s= "Hello World!";  
 *log*.info("End");  
 return s;  
 }  
}

**//Main class**

package com.cognizant.spring\_learn;

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

@SpringBootApplication  
public class SpringLearnApplication {

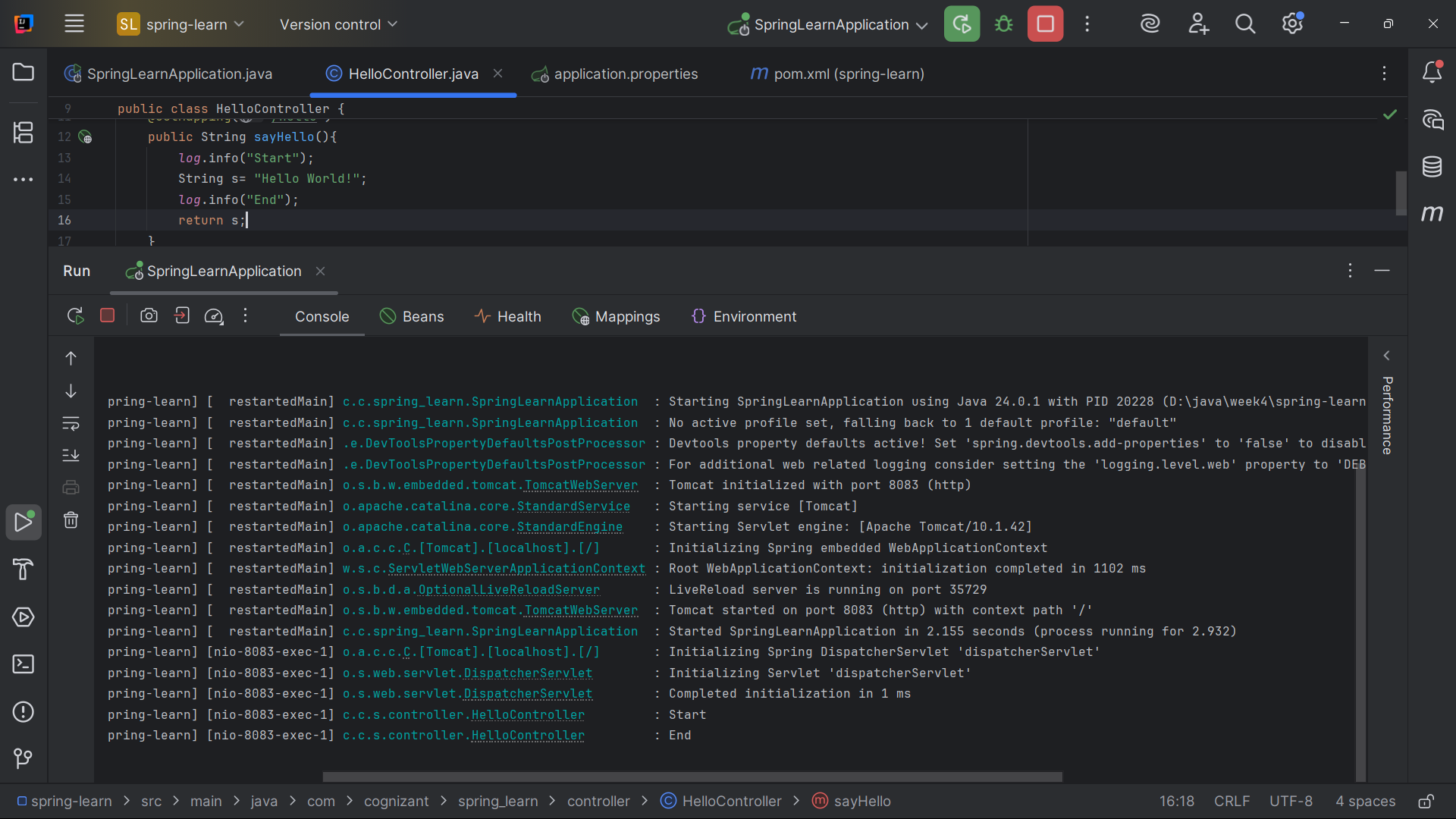
public static void main(String[] args) {  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
}

}

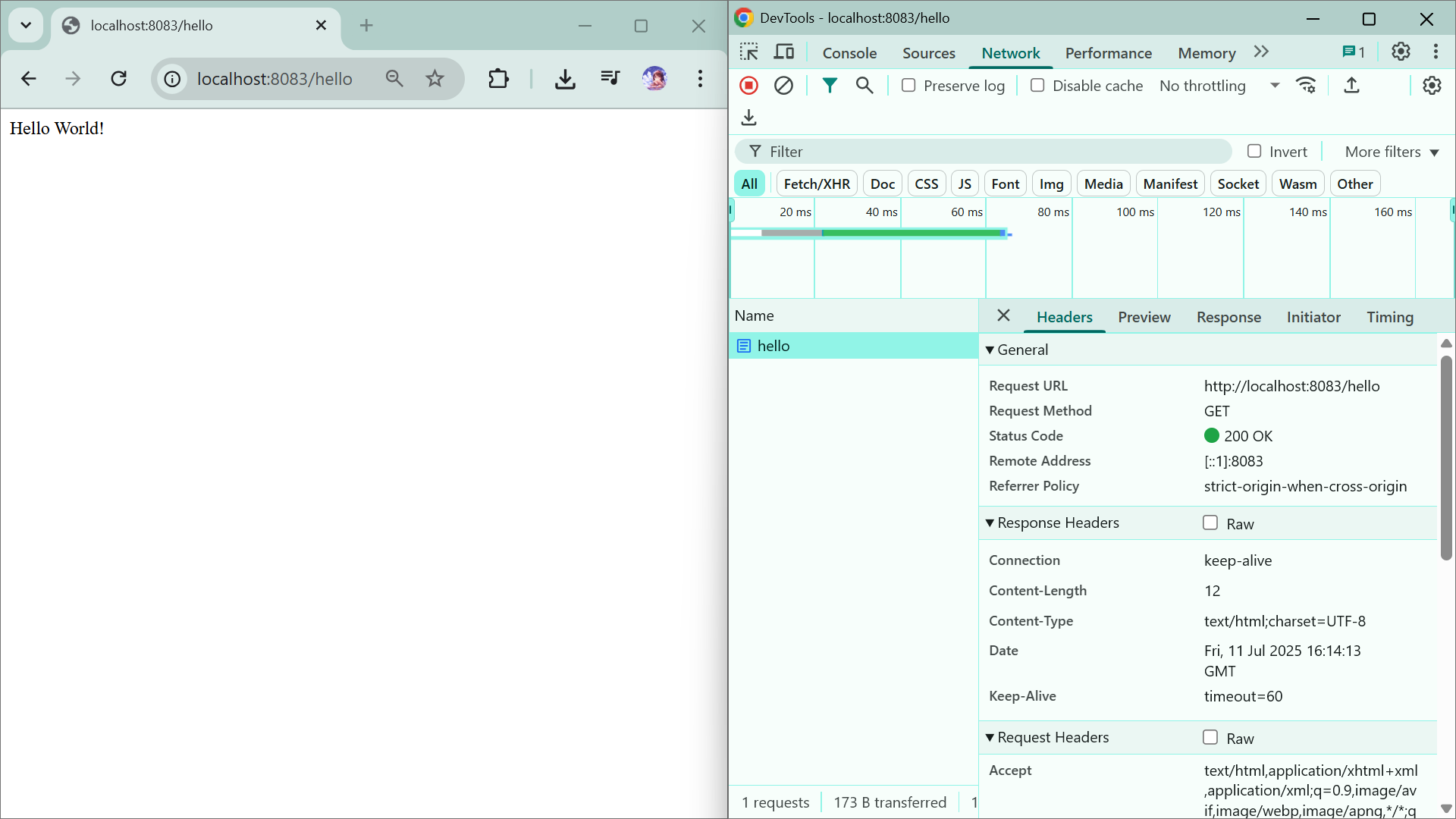
**//application.properties – for server port change**

spring.application.name=spring-learn  
server.port=8083

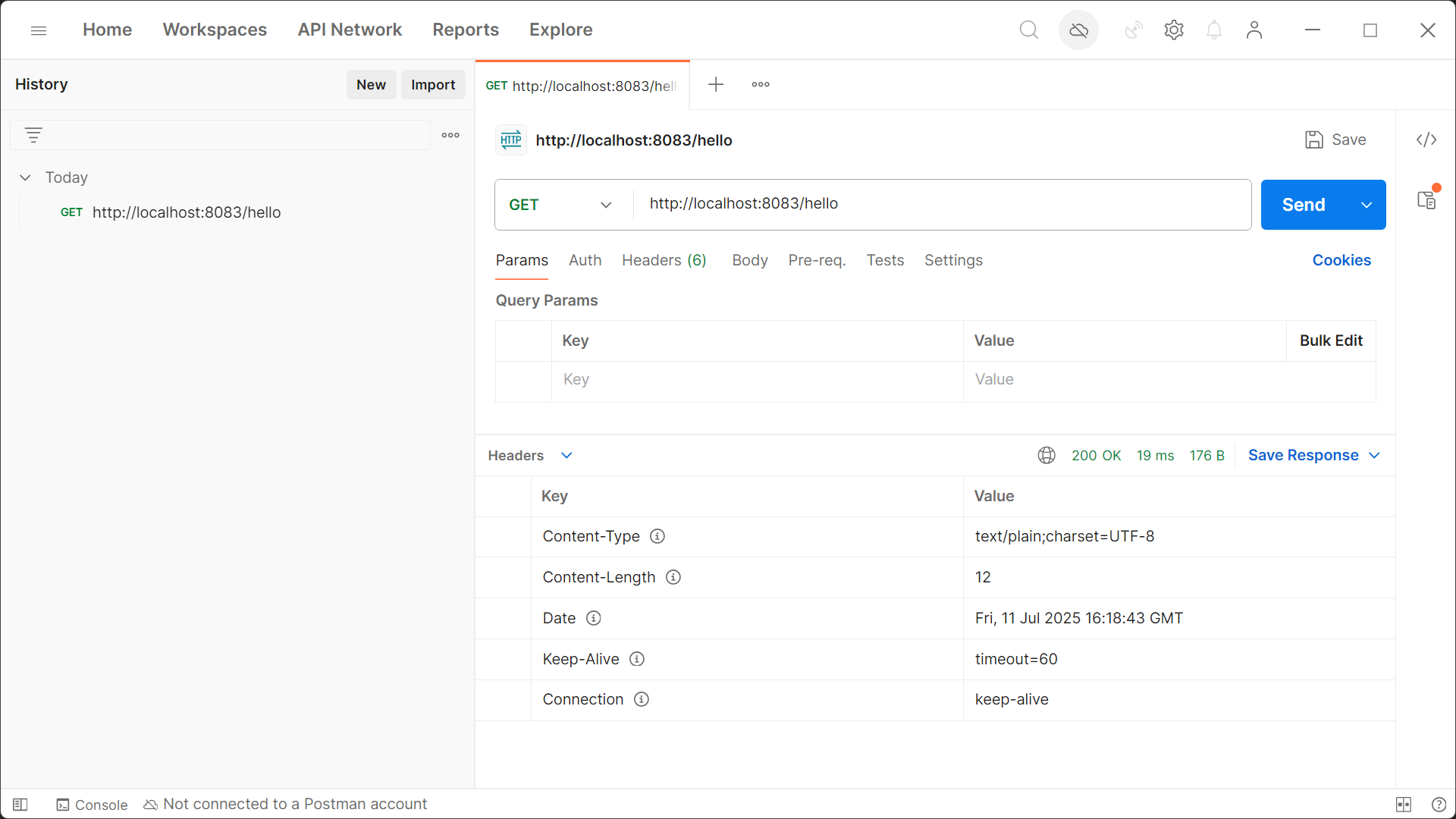
**OUTPUT**



//web browser- Developer tools-Headers



//Postman-"Headers" tab



**REST - Country Web Service**

**//country.java**

package com.cognizant.spring\_learn.model;  
import jakarta.persistence.Column;  
import jakarta.persistence.Entity;  
import jakarta.persistence.Id;  
import jakarta.persistence.Table;  
  
@Entity  
@Table(name="country")  
public class country{  
  
 @Id  
 @Column(name="co\_code")  
 private String code;  
  
 @Column(name="co\_name")  
 private String name;  
  
 // getters and setters  
  
 public String getCode() {  
 return code;  
 }  
  
 public void setCode(String code) {  
 this.code = code;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 // toString()  
 @Override  
 public String toString() {  
 return "Country{" + "code=" + code + ", name=" + name + '}';  
 }  
  
}

**//country.xml**

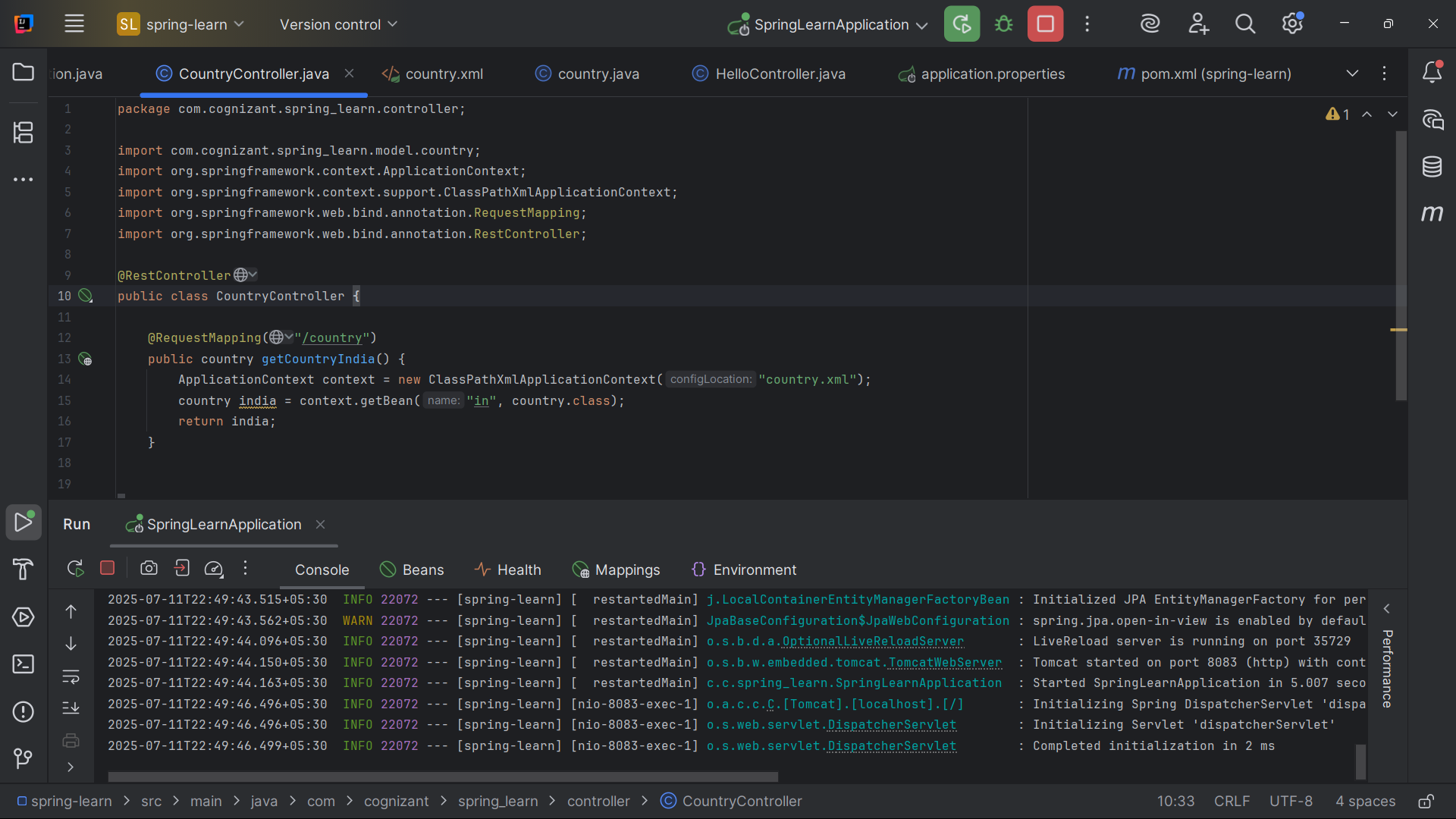
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">  
<bean id="in" class="com.cognizant.spring\_learn.model.country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
</bean>  
</beans>

**// CountryController**

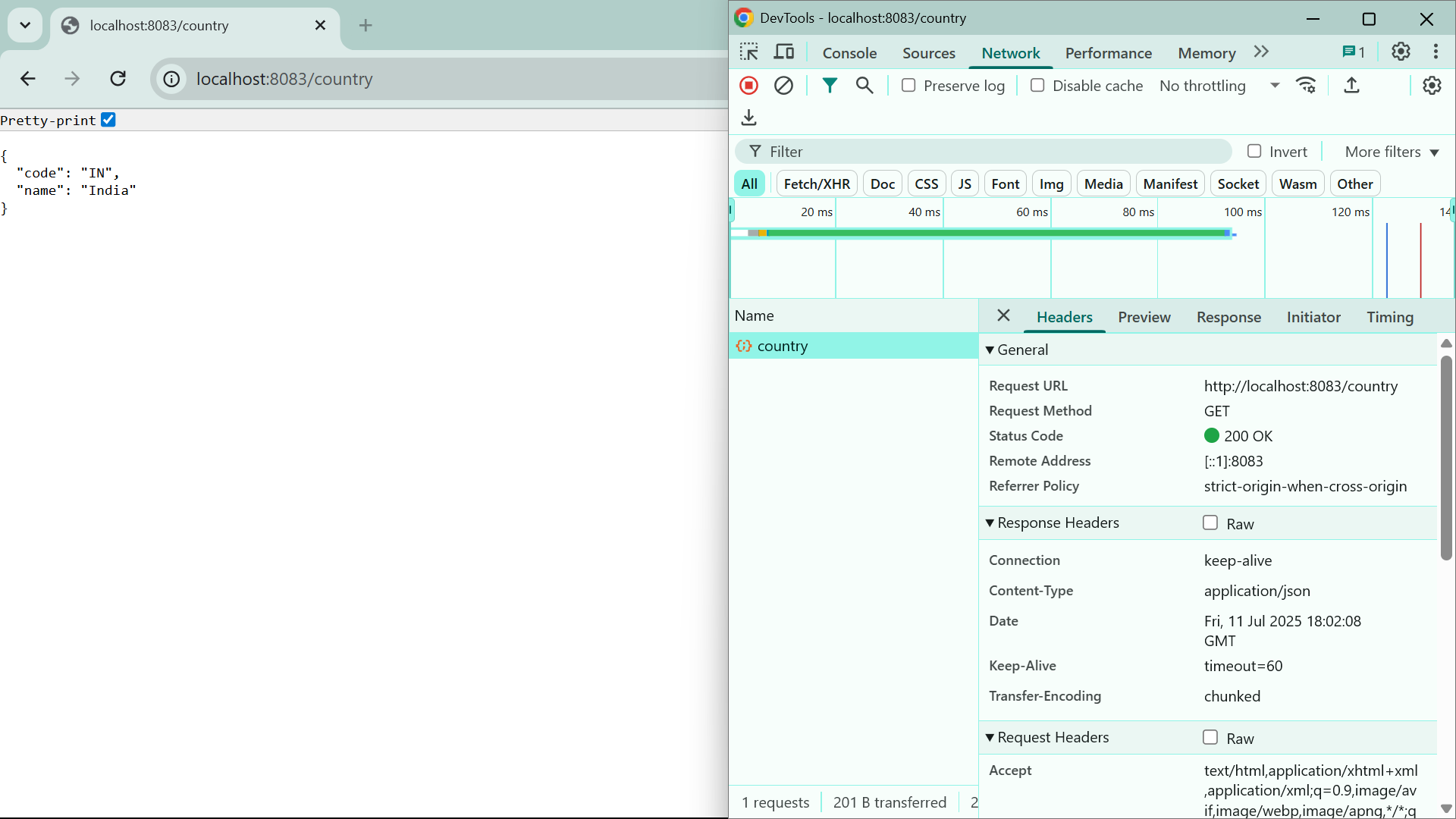
package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.model.country;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class CountryController {  
  
 @RequestMapping("/country")  
 public country getCountryIndia() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 country india = context.getBean("in", country.class);  
 return india;  
 }  
}

**OUTPUT**

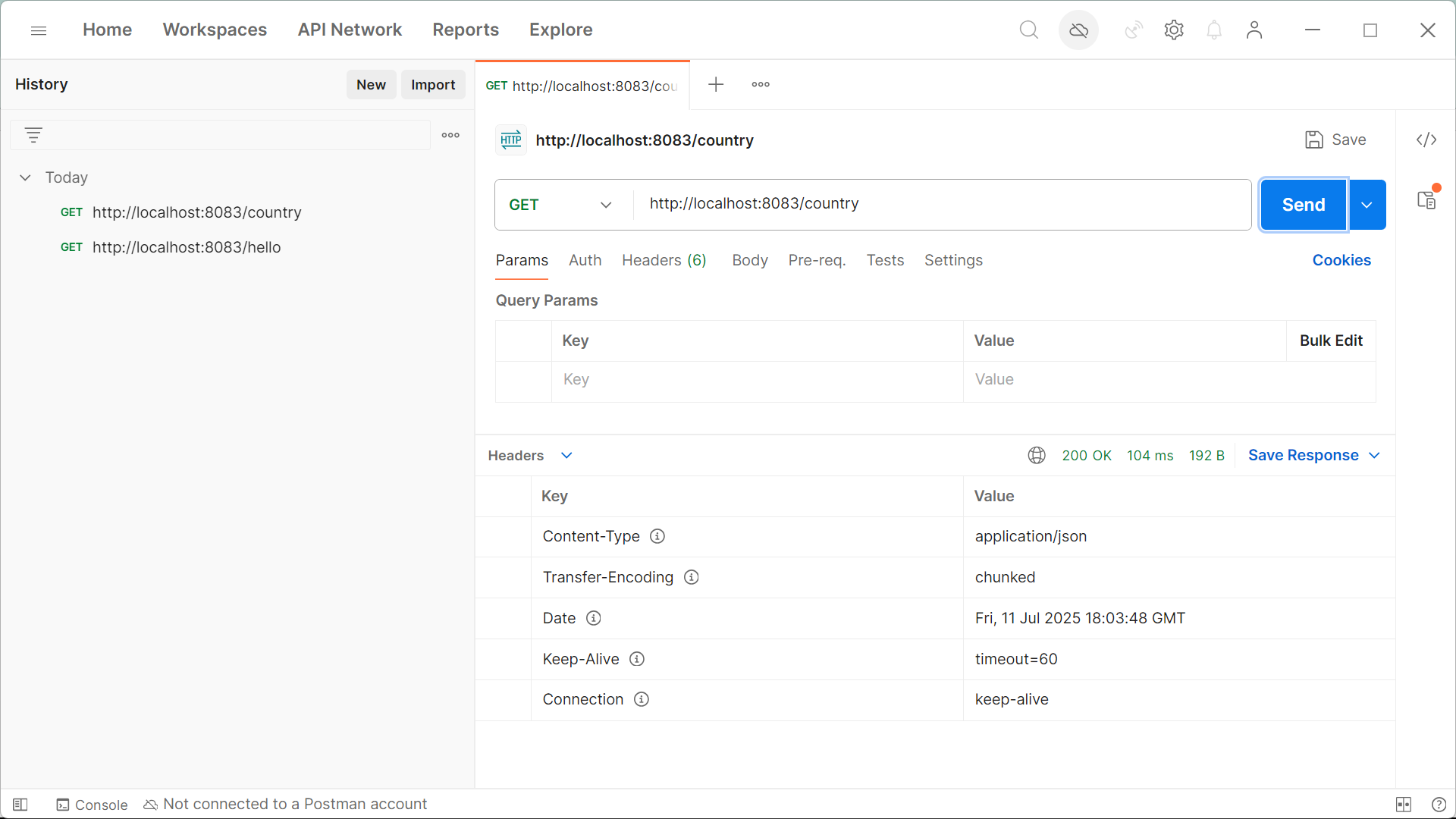
//console



//web brower



//Postman



**REST - Get country based on country code**

**//country.xml**

<bean id="countryList" class="java.util.ArrayList">  
 <constructor-arg>  
 <list>  
 <bean class="com.cognizant.spring\_learn.model.country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
 <bean class="com.cognizant.spring\_learn.model.country">  
 <property name="code" value="US"/>  
 <property name="name" value="United States"/>  
 </bean>  
 <bean class="com.cognizant.spring\_learn.model.country">  
 <property name="code" value="CN"/>  
 <property name="name" value="China"/>  
 </bean>  
 </list>  
 </constructor-arg>  
</bean>

**// CountryController**

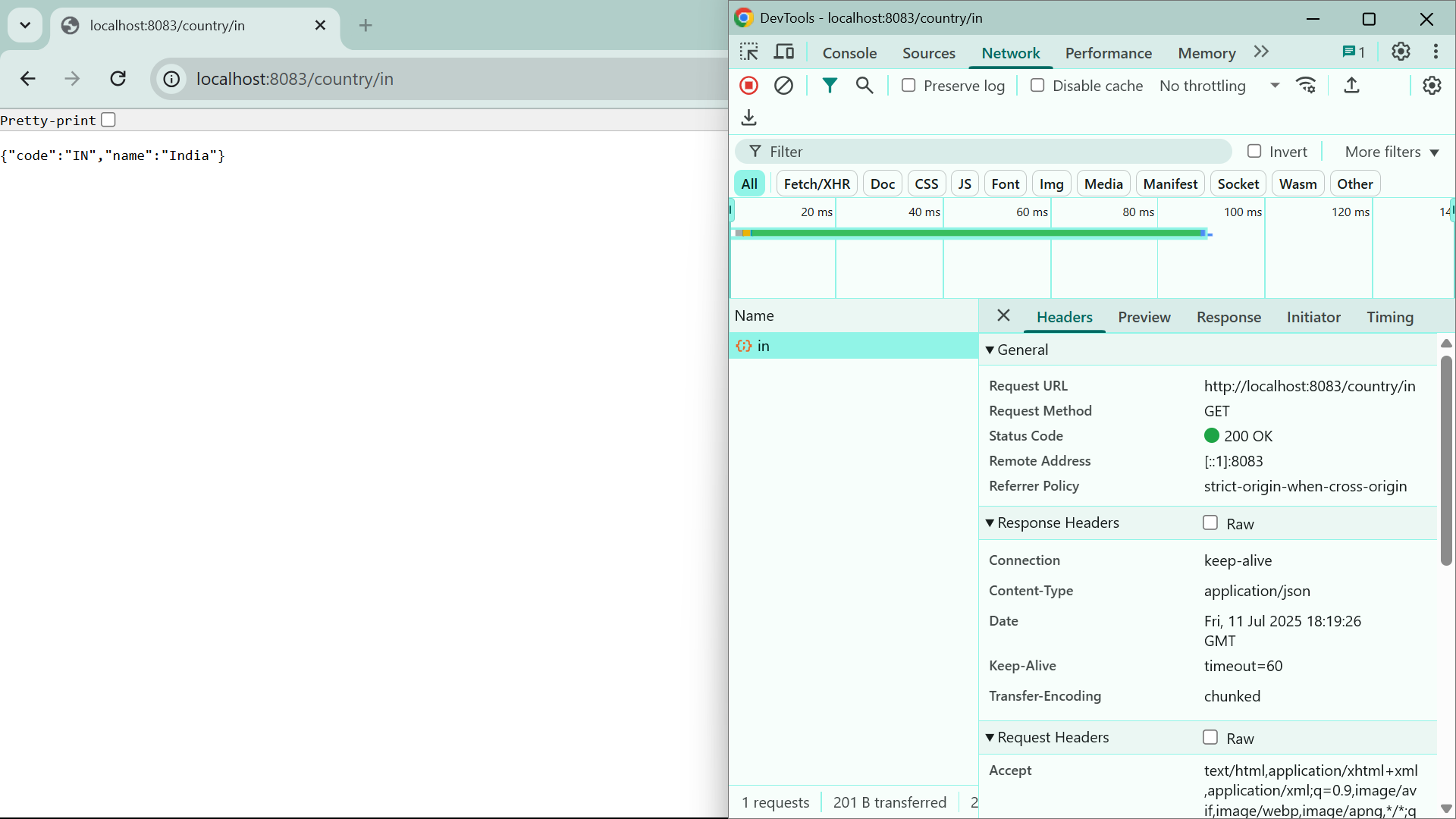
@RestController  
public class CountryController {  
 @Autowired  
 private CountryService countryService;

@GetMapping("/country/{code}")  
 public country getCountry(@PathVariable String code){  
 return countryService.getCountry(code);  
 }  
}

**//CountryService**

package com.cognizant.spring\_learn.service;  
  
import com.cognizant.spring\_learn.model.country;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.springframework.stereotype.Service;  
import org.springframework.web.bind.annotation.PathVariable;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 public country getCountry(String code) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 List<country> countryList = (List<country>) context.getBean("countryList");  
  
 return countryList.stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElse(null); // or throw a custom exception if preferred  
 }  
}

**OUTPUT**



**JWT-Handson**

**Create authentication service that returns JWT**

**// JwtUtil Class**

package com.cognizant.spring\_learn.controller;  
  
import org.springframework.stereotype.Component;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
  
@Component  
public class JwtUtil {  
 private final String SECRET = "mysecret";  
  
 public String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.*currentTimeMillis*()))  
 .setExpiration(new Date(System.*currentTimeMillis*() + 10 \* 60 \* 1000)) // 10 min  
 .signWith(SignatureAlgorithm.*HS256*, SECRET)  
 .compact();  
 }  
}

**// AuthenticationController**

package com.cognizant.spring\_learn.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import javax.servlet.http.HttpServletRequest;  
import java.nio.charset.StandardCharsets;  
import java.util.Base64;  
import java.util.HashMap;  
import java.util.Map;  
  
@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @GetMapping("/authenticate")  
 public Map<String, String> authenticate(HttpServletRequest request) {  
 // Step 2: Read and decode Authorization header  
 String authHeader = request.getHeader("Authorization");  
 if (authHeader == null || !authHeader.startsWith("Basic ")) {  
 throw new RuntimeException("Missing or invalid Authorization header");  
 }  
  
 String base64Credentials = authHeader.substring("Basic ".length());  
 byte[] decodedBytes = Base64.*getDecoder*().decode(base64Credentials);  
 String credentials = new String(decodedBytes, StandardCharsets.*UTF\_8*);  
  
 // credentials = "user:pwd"  
 String[] values = credentials.split(":", 2);  
 String username = values[0];  
 String password = values[1];  
  
 // Step 3: Validate credentials manually (for demo purpose)  
 if (!("user".equals(username) && "pwd".equals(password))) {  
 throw new RuntimeException("Invalid credentials");  
 }  
  
 // Step 3: Generate token  
 String token = jwtUtil.generateToken(username);  
 Map<String, String> response = new HashMap<>();  
 response.put("token", token);  
 return response;  
 }  
}

**// SecurityConfig class**

package com.cognizant.spring\_learn.security;  
  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeRequests()  
 .antMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated();  
 }  
}

**OUTPUT**

