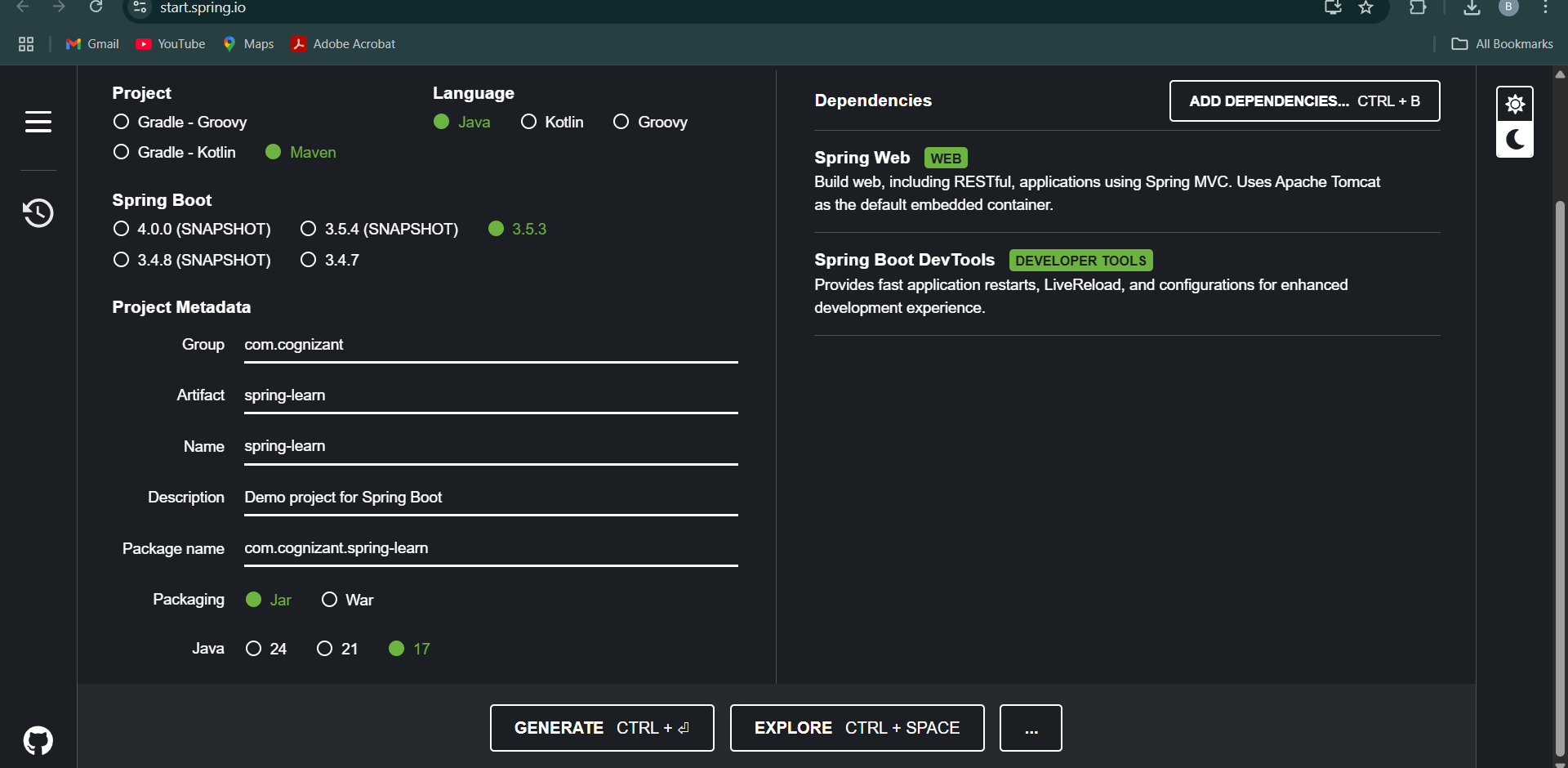
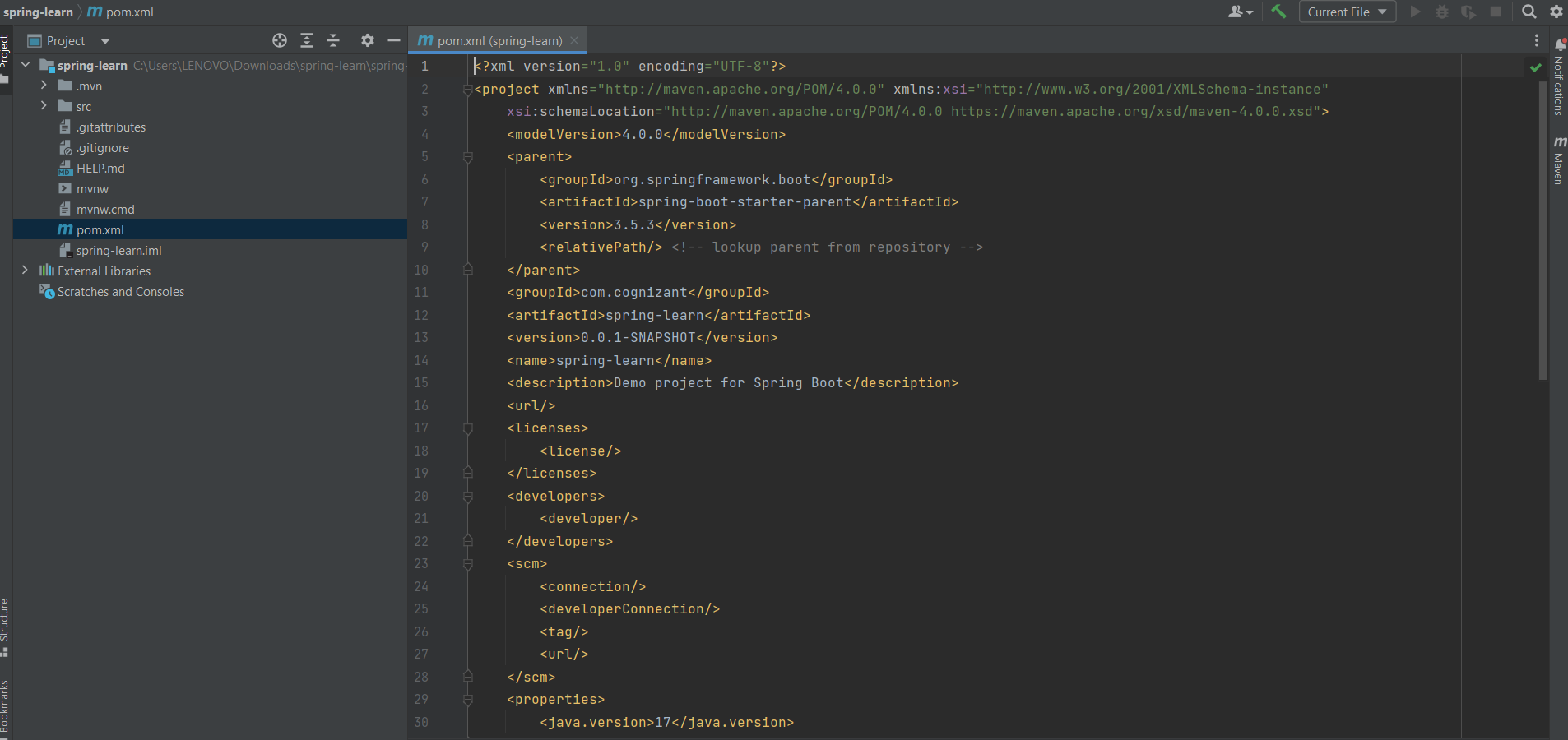
**EXERCISE 1 :**

**Create a Spring Web Project using Maven**

Step 1: Created project using Spring Initializr

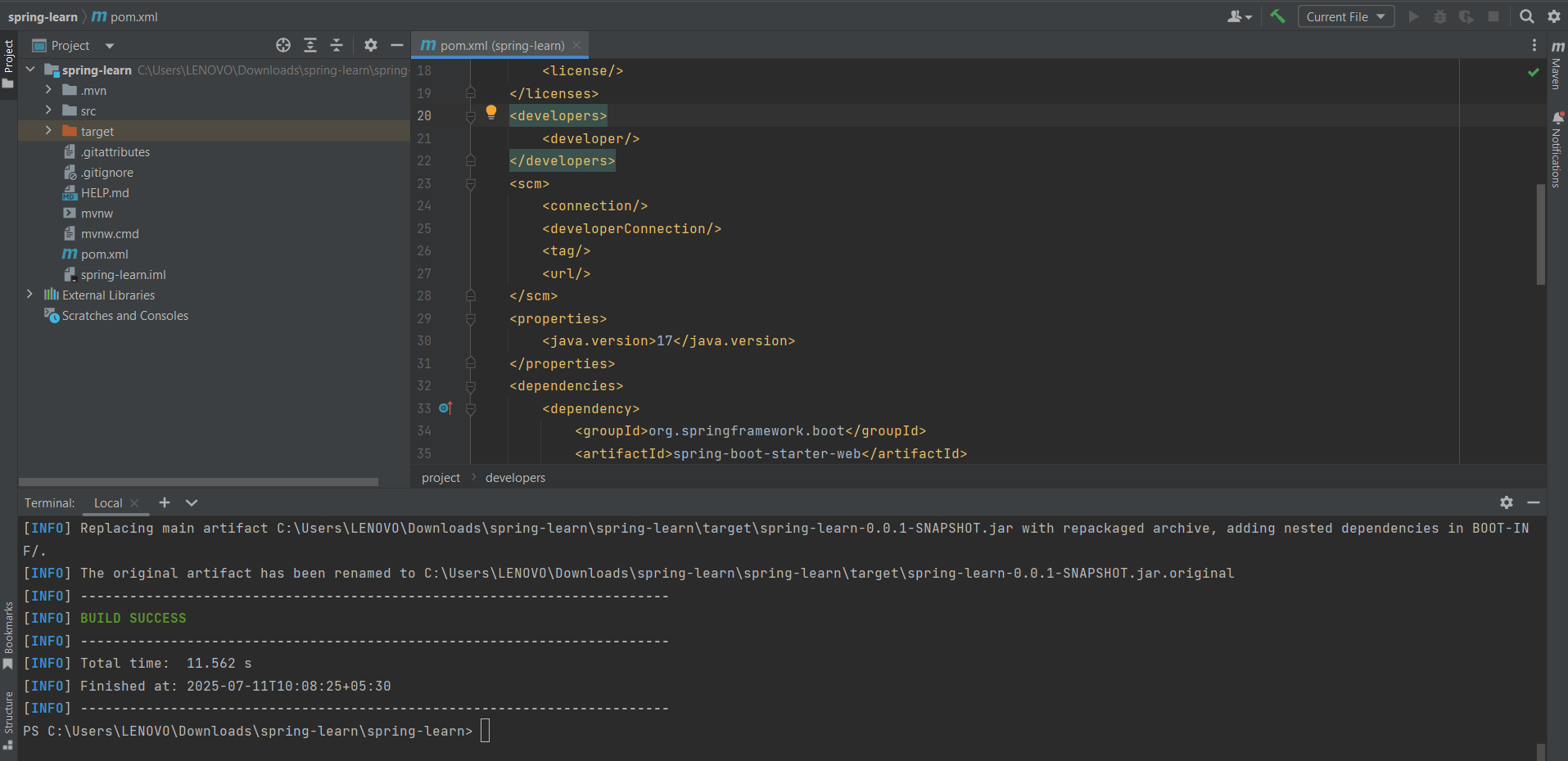


Step 2: Extracted project into INTELLIJ workspace



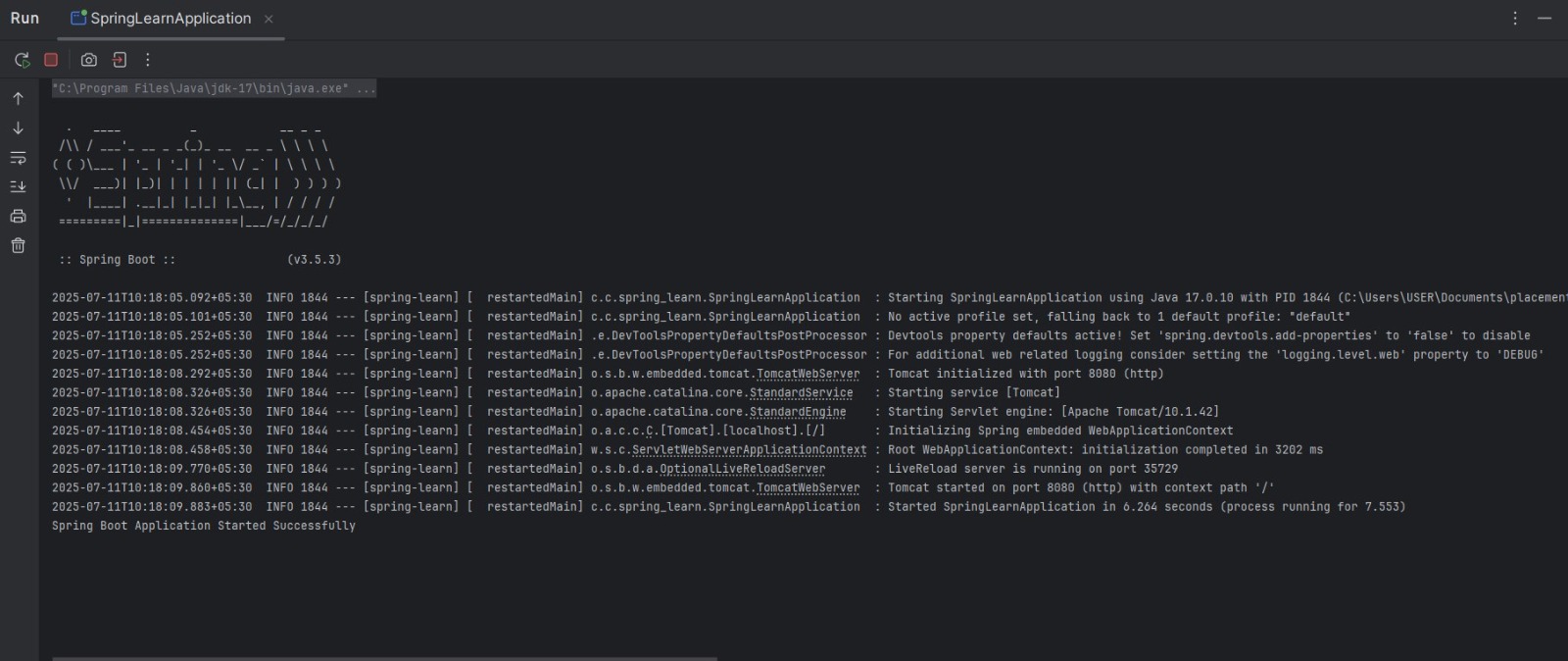
Step 3: Ran the Maven build command

Step 4: Imported Maven project into INTELLIJ



Step 5: Added logs in main() method:  
  
@SpringBootApplication  
public class SpringLearnApplication {  
 public static void main(String[] args) {  
 System.out.println("Application started...");  
 SpringApplication.run(SpringLearnApplication.class, args);  
 }  
}

Step 6: Ran the application and verified log output



# Project Structure & Explanation

1. src/main/java- Contains Java code including SpringLearnApplication.java

2. src/main/resources - Holds application.properties and configuration files

3. src/test/java- Contains test files for unit testing (auto-generated)

4. SpringLearnApplication.java - The entry point of the Spring Boot application

# Annotations & Purpose

@SpringBootApplication

A combination of:

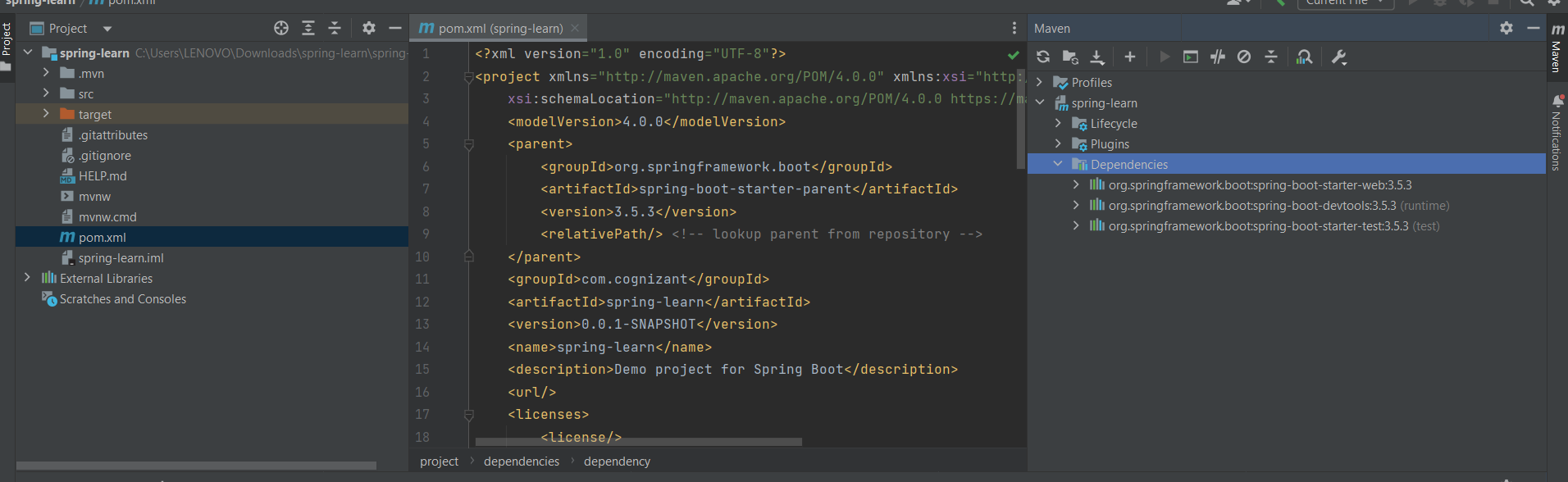
- @Configuration  
- @EnableAutoConfiguration  
- @ComponentScan

It enables Spring Boot auto-configuration and component scanning.

# Maven pom.xml

Key Dependencies:

<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>  
 </dependency>  
 <!-- Additional dependencies auto-added -->  
</dependencies>



# Dependency Hierarchy :

# 

# Final Output :

Application successfully built and ran.

Console printed: Application started...

# 

# Conclusion:

This hands-on helped understand the basics of:  
- Spring Boot project setup using Maven  
- INTELLIJ project import and build  
- Maven dependencies and logs  
- Spring Boot application structure

**EXERCISE 2 :**

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

SOLUTION :

# Spring Configuration: country.xml

<bean id="country" class="com.cognizant.springlearn.Country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
</bean>

Country.java

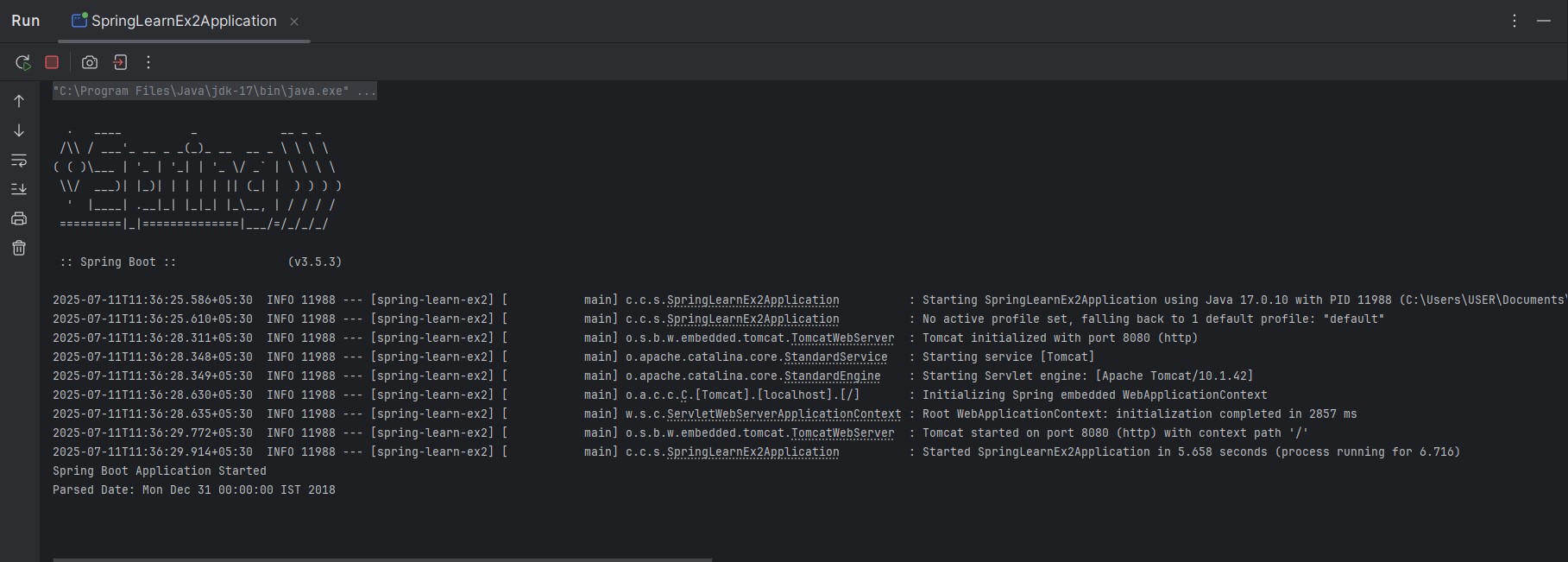
package com.cognizant.springlearn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class Country {  
 private String code;  
 private String name;  
  
 private static final Logger LOGGER = LoggerFactory.getLogger(Country.class);  
  
 public Country() {  
 LOGGER.debug("Inside 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 + ''' +  
 '}';  
 }  
}

# SpringLearnApplication.java

package com.cognizant.springlearn;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class SpringLearnApplication {  
 private static final Logger LOGGER = LoggerFactory.getLogger(SpringLearnApplication.class);  
  
 public static void main(String[] args) {  
 LOGGER.debug("START");  
 displayCountry();  
 LOGGER.debug("END");  
 }  
  
 public static void displayCountry() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 Country country = context.getBean("country", Country.class);  
 LOGGER.debug("Country : {}", country.toString());  
 }  
}

.

**OUTPUT :**



**EXERCISE 3 :**

**Hello World RESTful Web Service**

**SOLUTION :**

HelloController.java

package com.cognizant.springlearn.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 LOGGER = LoggerFactory.getLogger(HelloController.class);  
  
 @GetMapping("/hello")  
 public String sayHello() {  
 LOGGER.debug("Start sayHello()");  
 String message = "Hello World!!";  
 LOGGER.debug("End sayHello()");  
 return message;  
 }  
}

SpringLearnApplication.java

package com.cognizant.springlearn;  
  
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.debug("Application Started");  
 SpringApplication.run(SpringLearnApplication.class, args);  
 LOGGER.debug("Application Ended");  
 }  
}

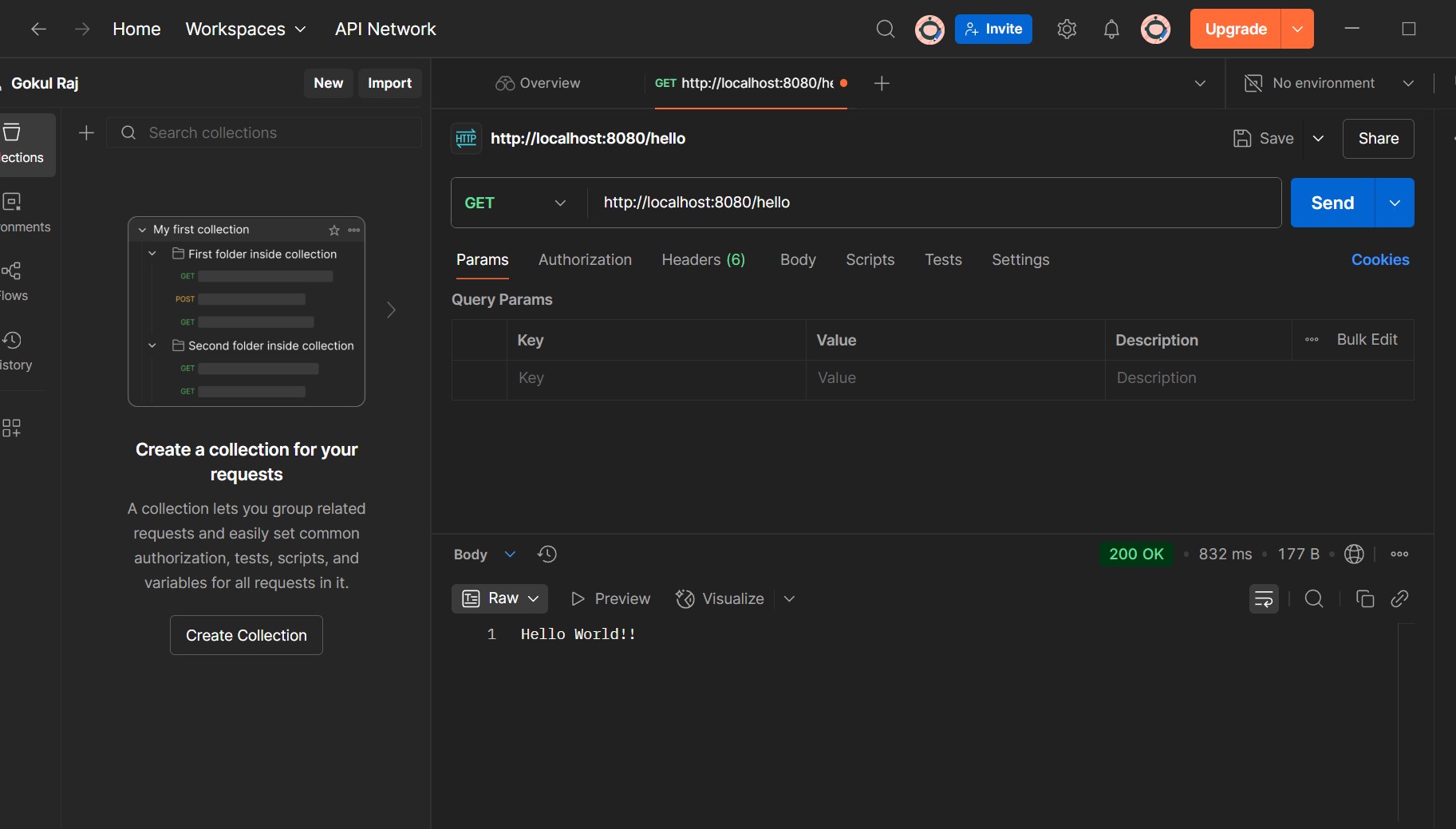
application.properties (Optional)

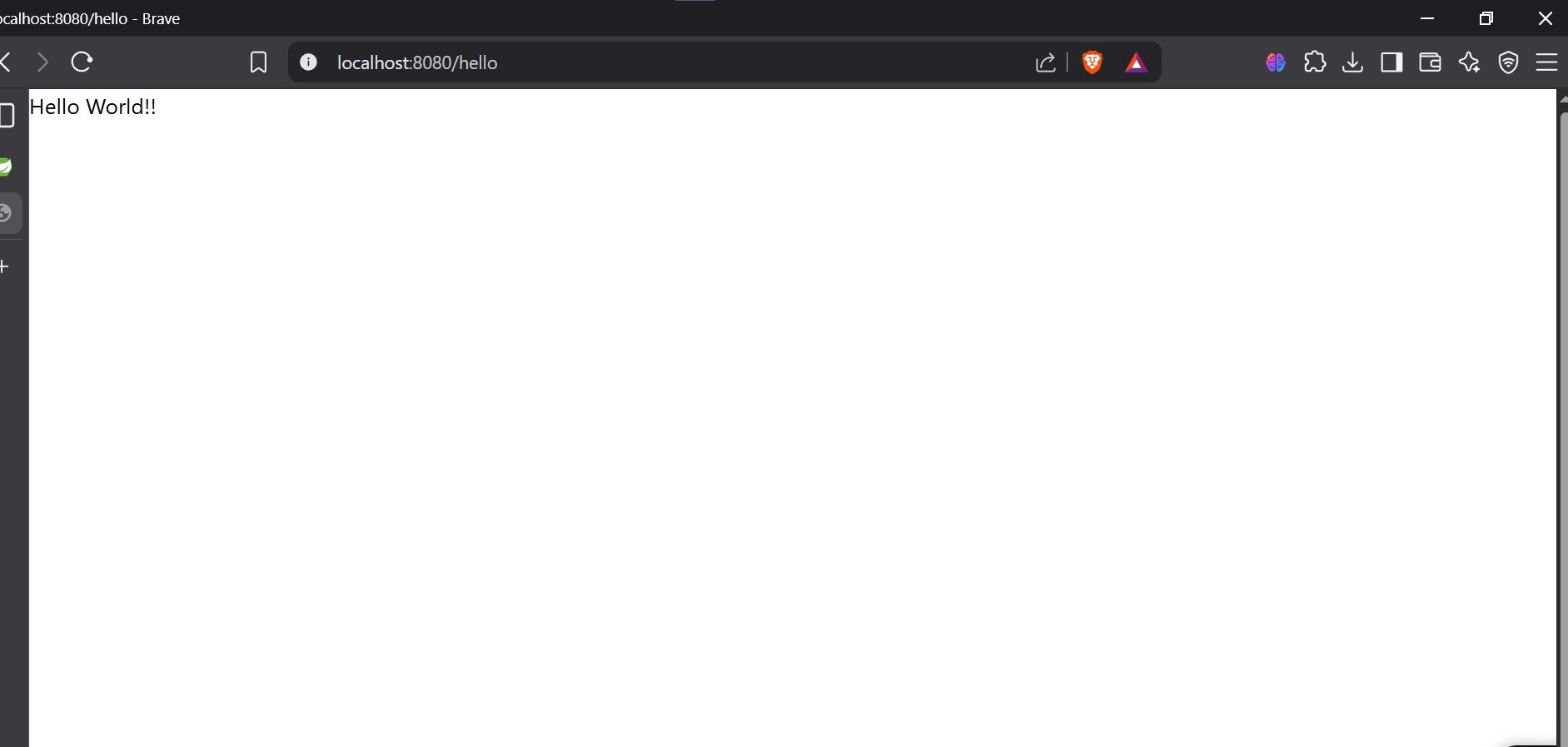
server.port=8083

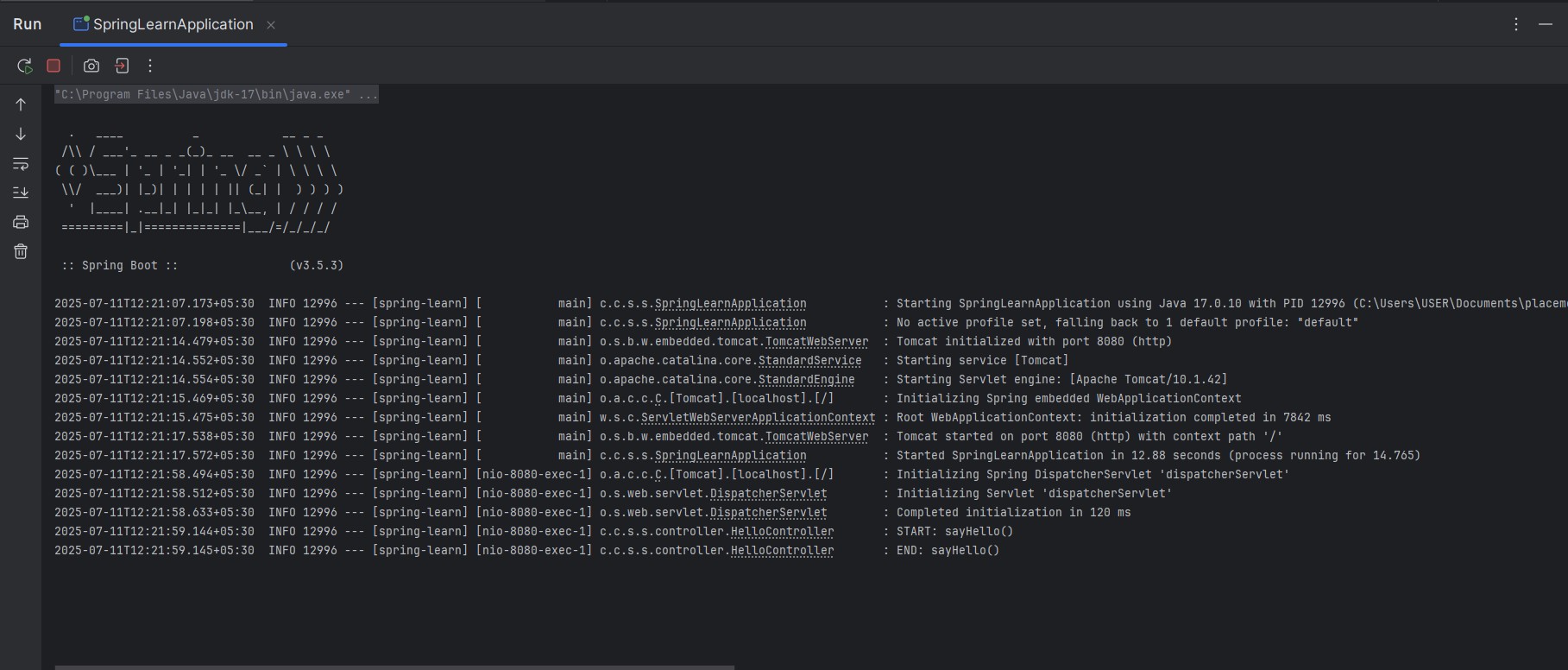
Spring Web Dependency (pom.xml)

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>

**OUTPUT :**

****

****

****

**EXERCISE 4 :**

**REST - Country Web Service**

**SOLUTION :**

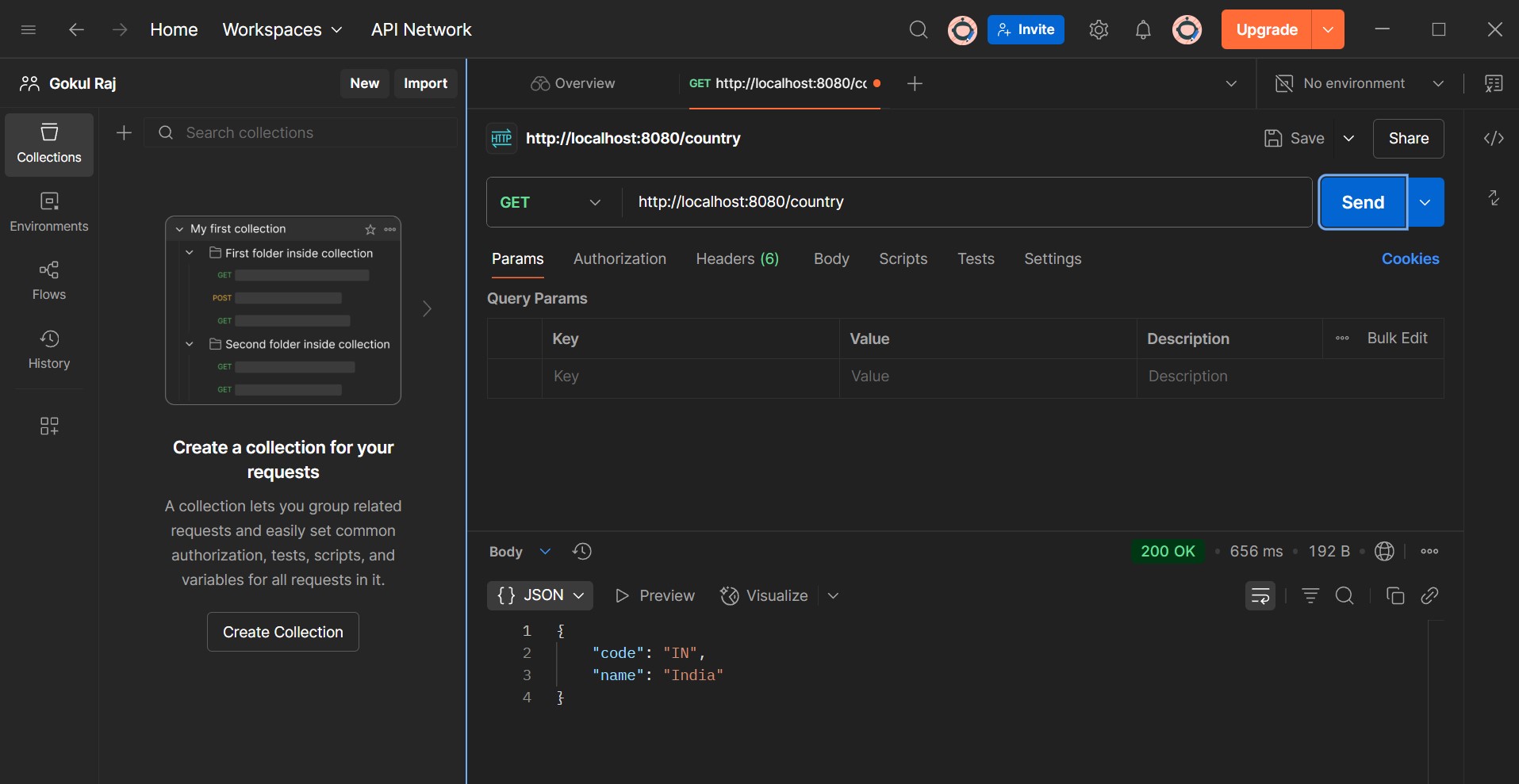
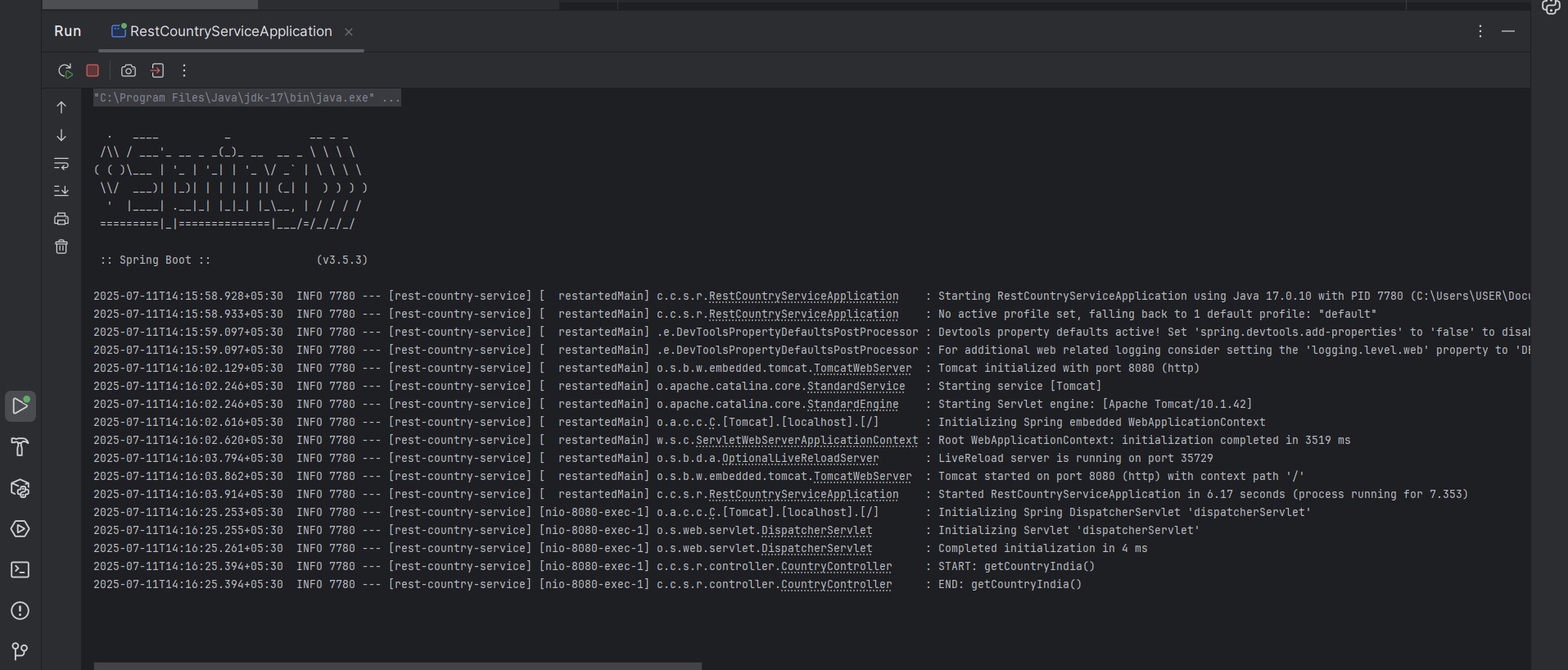
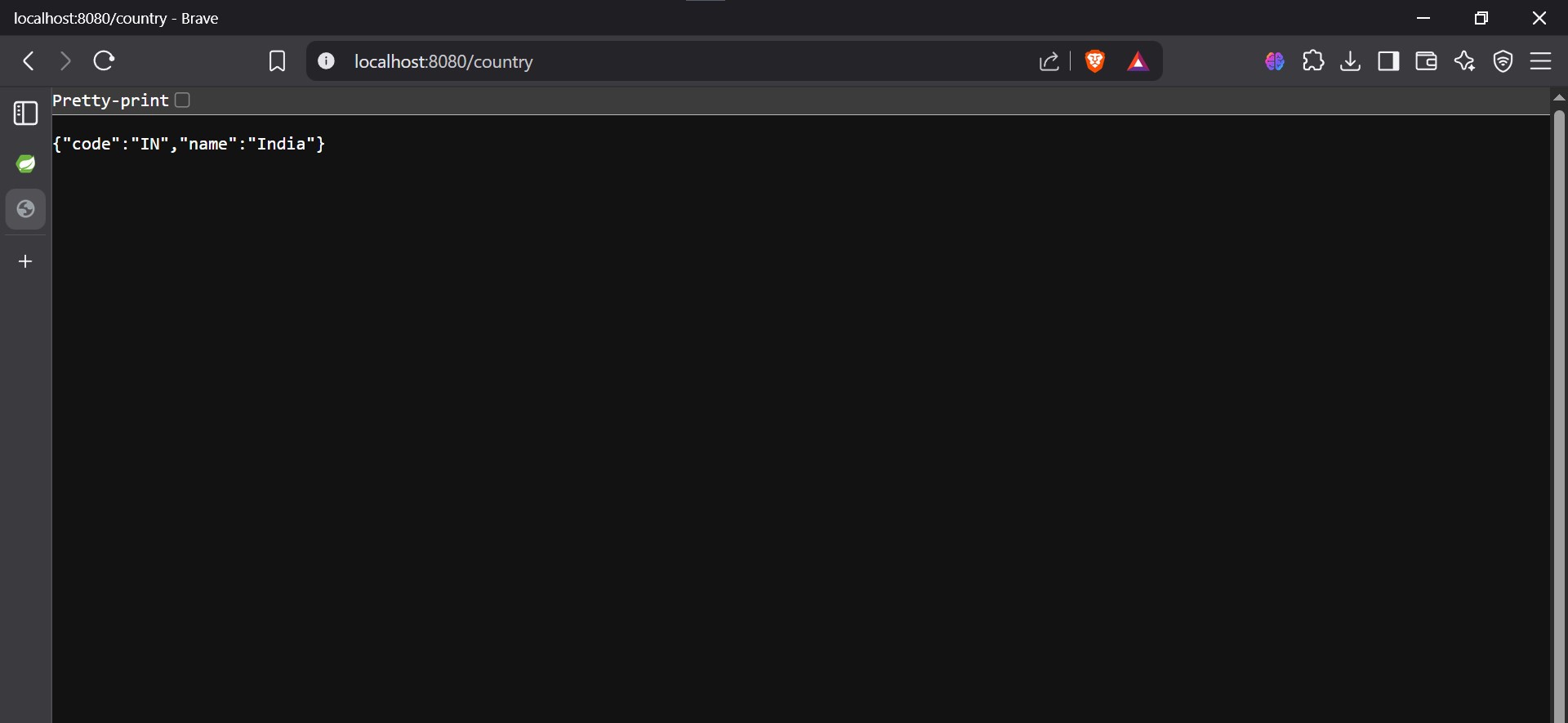
country.xml :

<bean id="country" class="com.cognizant.springlearn.Country">  
 <property name="code" value="IN" />  
 <property name="name" value="India" />  
</bean>

CountryController.java :

package com.cognizant.springlearn.controller;  
  
import com.cognizant.springlearn.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.debug("Start getCountryIndia()");  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 Country country = context.getBean("country", Country.class);  
 LOGGER.debug("End getCountryIndia()");  
 return country;  
 }  
}

**OUTPUT :**

  
}

**EXERCISE 5 :**

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

**SOLUTION :**

. Country Model

package com.cognizant.spring\_learn.model;  
  
public class Country {  
 private String code;  
 private String name;  
  
 public Country() {}  
  
 public Country(String code, String name) {  
 this.code = code;  
 this.name = name;  
 }  
  
 public String getCode() {  
 return code;  
 }  
  
 public void setCode(String code) {  
 this.code = code.toUpperCase();  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

2. CountryService

package com.cognizant.spring\_learn.service;  
  
import com.cognizant.spring\_learn.model.Country;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.oxm.jaxb.Jaxb2Marshaller;  
import org.springframework.stereotype.Service;  
  
import javax.xml.transform.stream.StreamSource;  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 @Autowired  
 private Jaxb2Marshaller marshaller;  
  
 public Country getCountry(String code) throws Exception {  
 StreamSource source = new StreamSource(new ClassPathResource("country.xml").getInputStream());  
 Countries countries = (Countries) marshaller.unmarshal(source);  
  
 return countries.getCountryList().stream()  
 .filter(c -> c.getCode().equalsIgnoreCase(code))  
 .findFirst()  
 .orElseThrow(() -> new Exception("Country not found with code: " + code));  
 }  
}

3. CountryController

package com.cognizant.spring\_learn.controller;  
  
import com.cognizant.spring\_learn.model.Country;  
import com.cognizant.spring\_learn.service.CountryService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
@RestController  
public class CountryController {  
  
 @Autowired  
 private CountryService countryService;  
  
 @GetMapping("/countries/{code}")  
 public Country getCountry(@PathVariable String code) throws Exception {  
 return countryService.getCountry(code);  
 }  
}

4. Countries Wrapper Class

package com.cognizant.spring\_learn.model;  
  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlRootElement;  
import java.util.List;  
  
@XmlRootElement(name = "countries")  
public class Countries {  
  
 private List<Country> countryList;  
  
 @XmlElement(name = "country")  
 public List<Country> getCountryList() {  
 return countryList;  
 }  
  
 public void setCountryList(List<Country> countryList) {  
 this.countryList = countryList;  
 }  
}

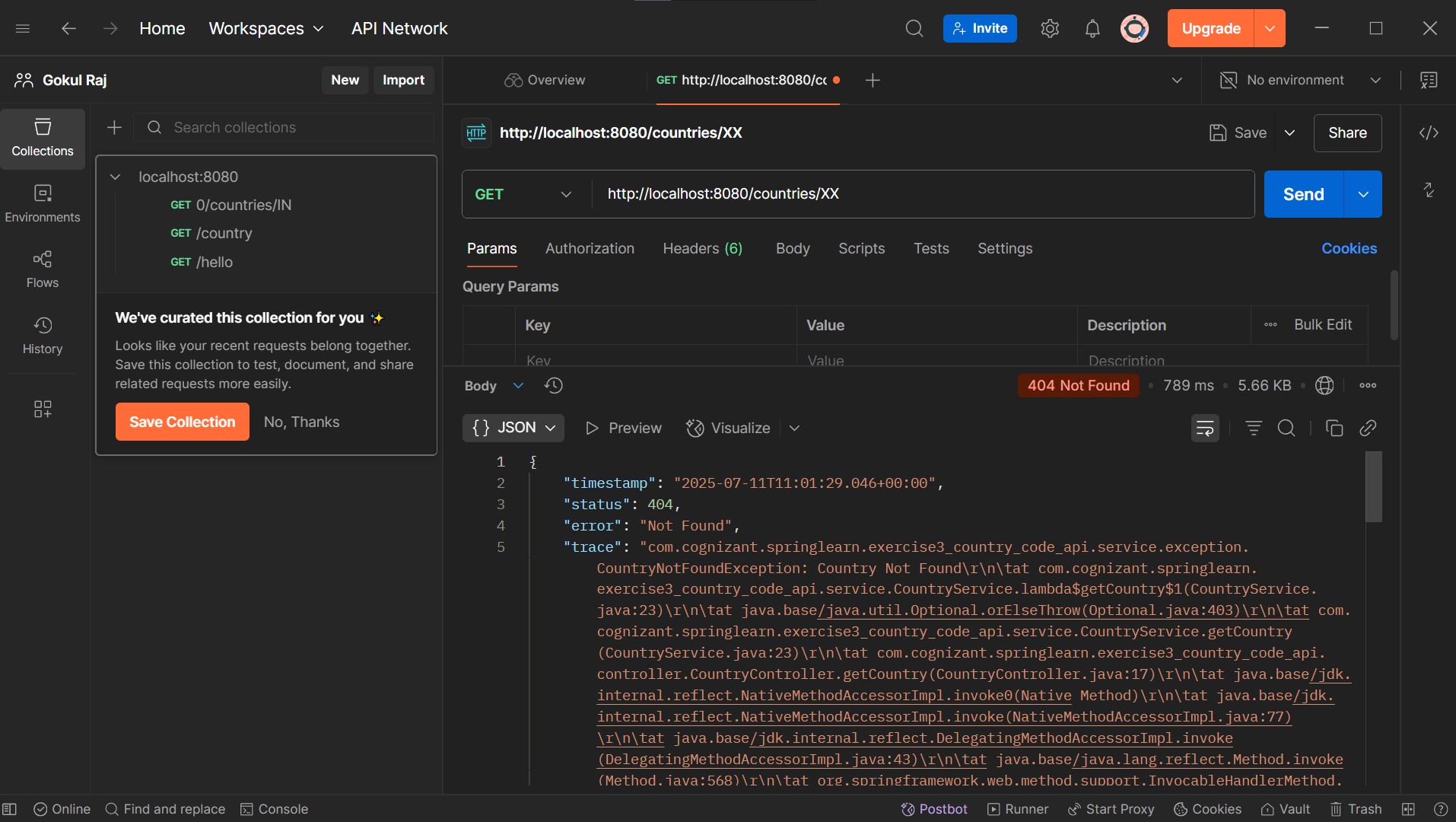
5. country.xml (Example)

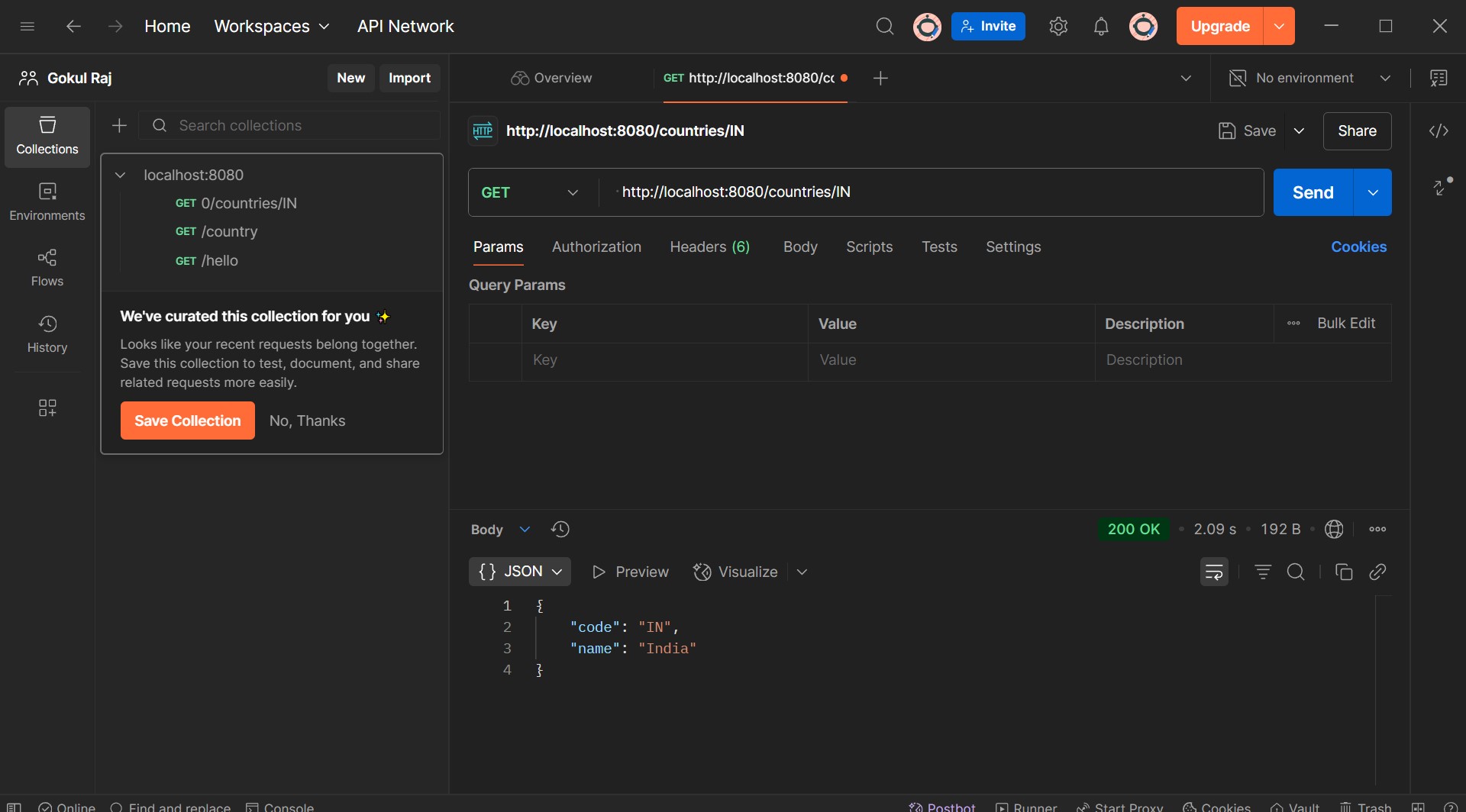
<countries>  
 <country>  
 <code>IN</code>  
 <name>India</name>  
 </country>  
 <country>  
 <code>US</code>  
 <name>United States</name>  
 </country>  
</countries>

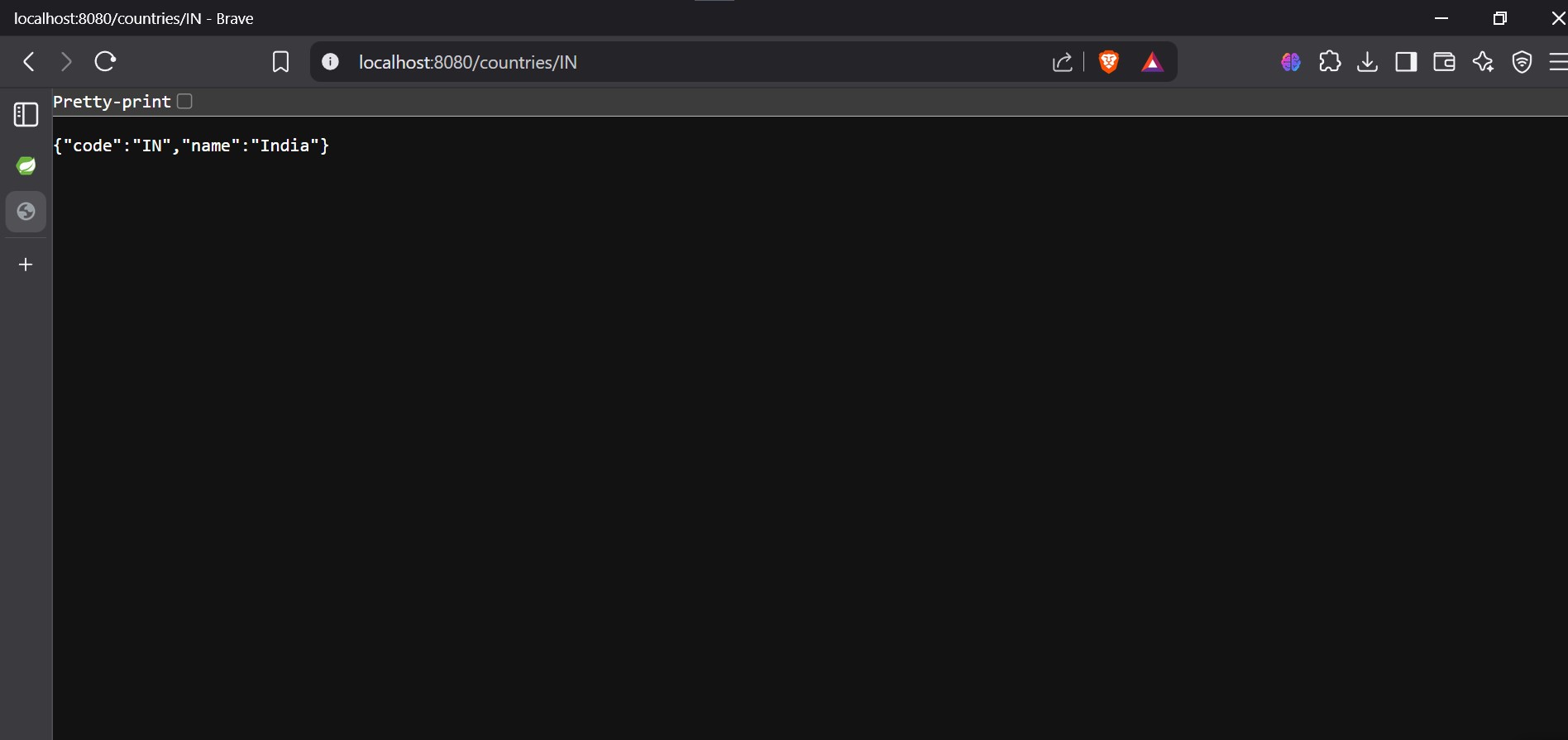
6. JAXB Configuration

package com.cognizant.spring\_learn.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.oxm.jaxb.Jaxb2Marshaller;  
  
@Configuration  
public class WebConfig {  
  
 @Bean  
 public Jaxb2Marshaller jaxb2Marshaller() {  
 Jaxb2Marshaller marshaller = new Jaxb2Marshaller();  
 marshaller.setPackagesToScan("com.cognizant.spring\_learn.model");  
 return marshaller;  
 }  
}

**OUTPUT :**







**EXERCISE 6 :**

**Create authentication service that returns JWT** 

SOLUTION :

Step 1: Create Authentication Controller

@RestController  
public class AuthenticationController {  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @PostMapping("/authenticate")  
 public ResponseEntity<?> generateToken(@RequestHeader("Authorization") String authHeader) {  
 String[] credentials = decodeBasicAuth(authHeader);  
 String username = credentials[0];  
 String password = credentials[1];  
  
 // Ideally validate against DB or in-memory store  
 if ("user".equals(username) && "pwd".equals(password)) {  
 String token = jwtUtil.generateToken(username);  
 return ResponseEntity.ok(Collections.singletonMap("token", token));  
 } else {  
 return ResponseEntity.status(HttpStatus.UNAUTHORIZED).build();  
 }  
 }  
  
 private String[] decodeBasicAuth(String authHeader) {  
 String base64Credentials = authHeader.substring("Basic".length()).trim();  
 byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);  
 String credentials = new String(credDecoded, StandardCharsets.UTF\_8);  
 return credentials.split(":", 2);  
 }  
}

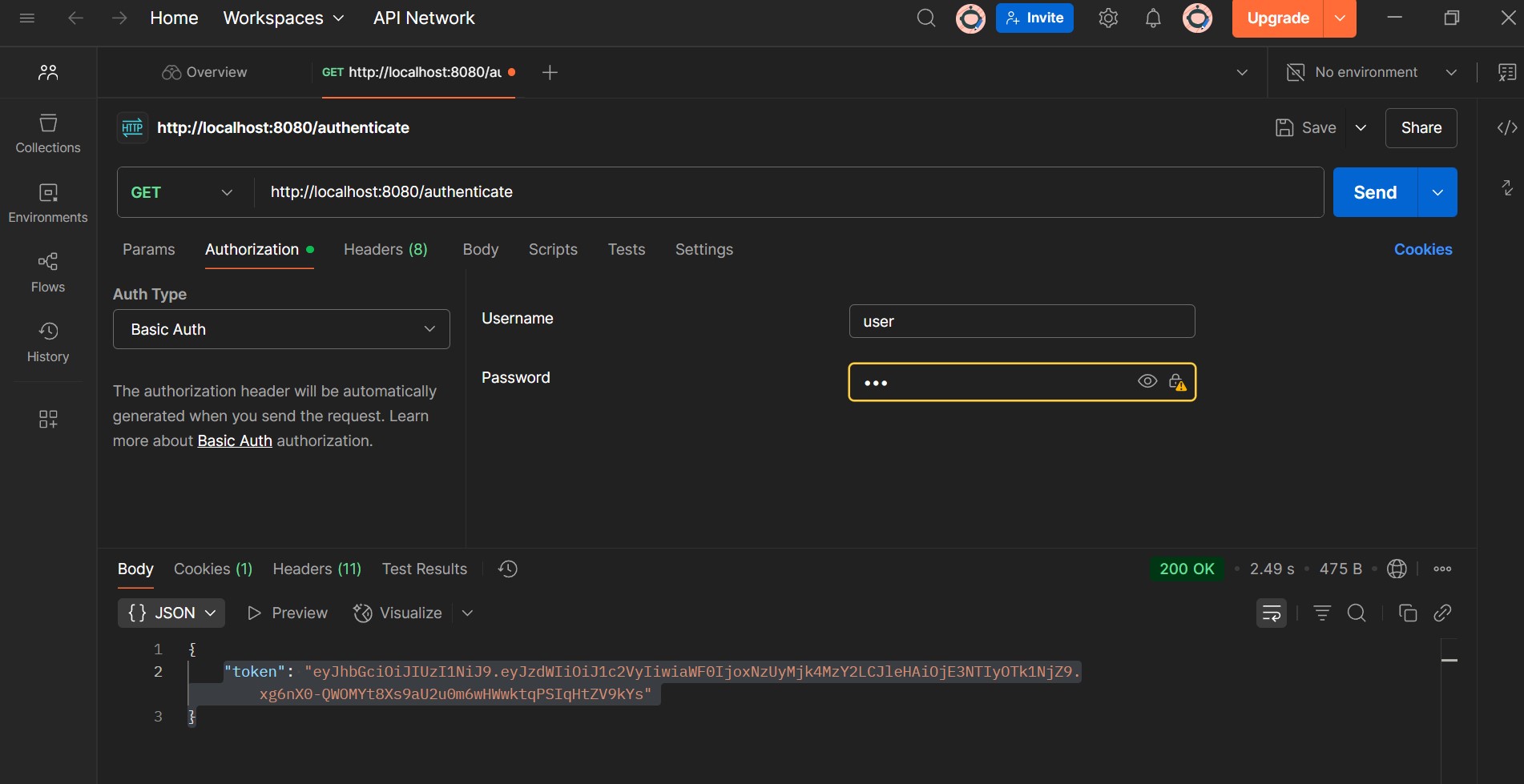
Step 2: Configure Security in SecurityConfig

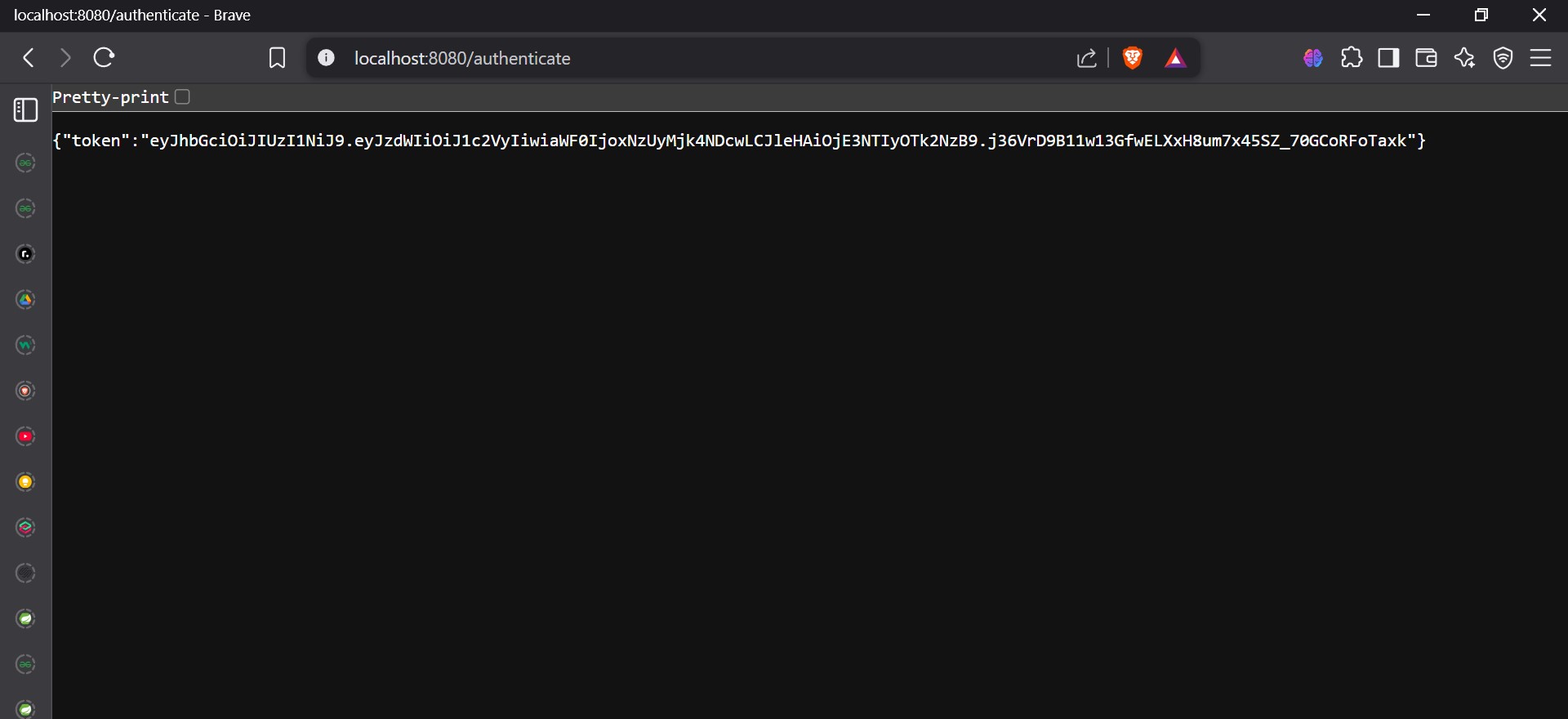
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeRequests()  
 .antMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated();  
 }  
}

Step 3: JWT Utility Class

@Component  
public class JwtUtil {  
  
 private String SECRET\_KEY = "secret";  
  
 public String generateToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.currentTimeMillis()))  
 .setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))  
 .signWith(SignatureAlgorithm.HS256, SECRET\_KEY)  
 .compact();  
 }  
}

**OUTPUT :**

****

****