**Spring REST using Spring Boot 3**

## Exercise 1: Create a Spring Web Project using Maven Step 1: Open Spring Initializr

* Open your web browser.
* Visit: <https://start.spring.io>
  + Fill the fields as follows:
    - **Project**: Maven
    - **Language**: Java
    - **Spring Boot Version**: Default
    - **Group**: com.cognizant
    - **Artifact**: spring-learn
    - **Name**: spring-learn
    - **Description**: Spring Web Project
    - **Packaging**: Jar
  + Click on **"Add Dependencies"**, and select:
    - **Spring Web**
    - **Spring Boot DevTools**
* **Download Folder and Extract**

**Step 2: Import Project in Eclipse**

* Open **Eclipse IDE**.
* Go to File > Import.
* Choose Maven > Existing Maven Projects > Click Next.
* Click Browse and select the folder where you extracted the project (spring-learn).
* Click Finish.

**Code :**

**SpringLearnApplication.java**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@SpringBootApplication

public class SpringLearnApplication {

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

public static void main(String[] args) {

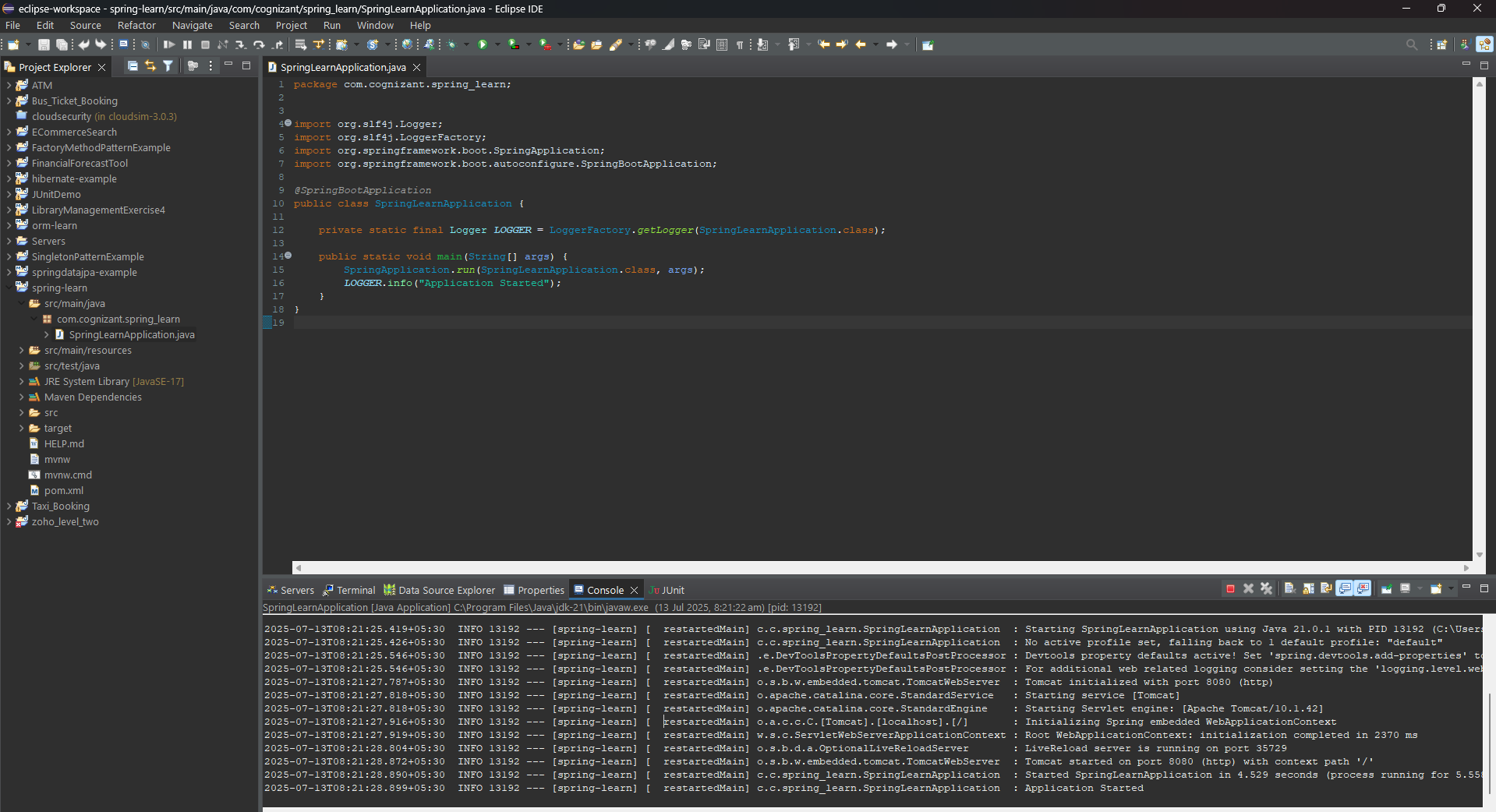
SpringApplication.run(SpringLearnApplication.class, args);

LOGGER.info("Application Started");

}

}

**Output :**

****

## SME Walkthrough

**1. src/main/java**

Contains your **application logic**, controller classes, models, services, etc.

**2. src/main/resources**

Holds **configuration files** like:

* application.properties
* static files (HTML, CSS, JS)

**3. src/test/java**

Contains **unit and integration tests** using JUnit or Mockito.

**4. SpringLearnApplication.java**

Main class with:

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

**5. @SpringBootApplication Annotation**

It combines:

* @Configuration: Marks class as source of bean definitions.
* @EnableAutoConfiguration: Enables Spring Boot’s auto-configuration.
* @ComponentScan: Scans the current package and sub-packages for components.

It tells Spring Boot to **bootstrap the application**.

**6. pom.xml - Maven Project Object Model**

Open pom.xml and review:

* Project coordinates (groupId, artifactId, version)
* Dependencies like spring-boot-starter-web, spring-boot-devtools
* Build plugins

Example:

<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>

</dependencies>

**7. View Dependency Tree**

* Open pom.xml
* Switch to the **"Dependency Hierarchy"** tab at the bottom
* This shows how dependencies are nested and what libraries are included.

**Exercise 2: Spring Core – Load Country from Spring Configuration XML**

**Step 1: Create a New Maven Project in Eclipse**

* Open **Eclipse**.
* Go to File > New > Maven Project.
* In the wizard:
  + Uncheck Use default Workspace location if needed.
  + Click Next.
* In Archetype Selection, choose:
  + maven-archetype-quickstart and click Next.
* Fill these:
  + **Group Id**: com.cognizant
  + **Artifact Id**: spring-core-demo
  + Click Finish.

**Step 2: Add Spring Core Dependency to pom.xml**

Open pom.xml, add:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

**Step 3: Necessary Files**

**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.springcore.Country">

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

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

</bean>

</beans>

**Country.java Class**

package com.cognizant.springcore;

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 + "]";

}

}

**SpringLearnApplication.java**

package com.cognizant.springcore;

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) {

displayCountry();

}

public static void displayCountry() {

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

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

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

}

}

**pom.xml**

**Add dependencies**

<!-- SLF4J Logger -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<!-- SLF4J Simple Implementation -->

<dependency>

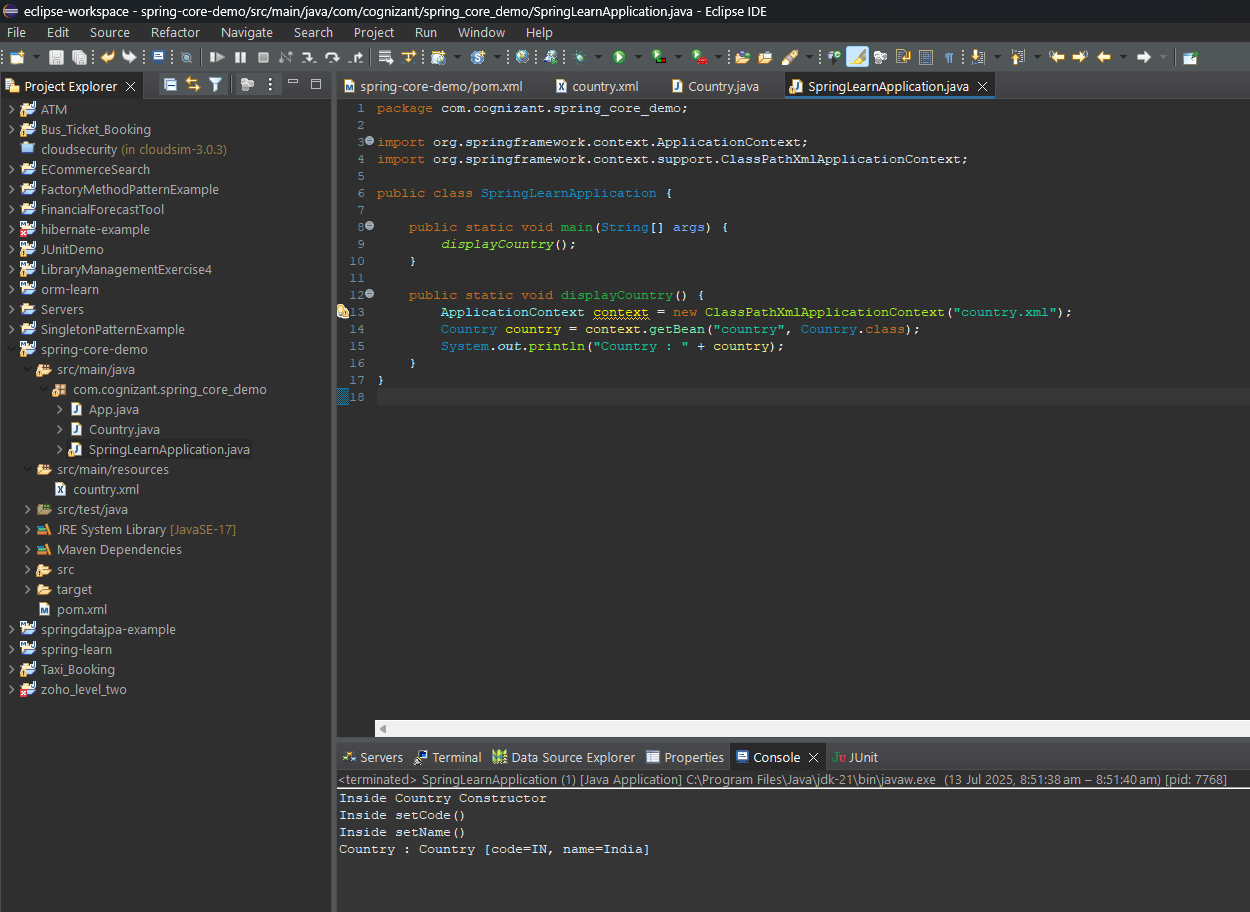
<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

**Output :**

****

## SME EXPLANATION

* **bean Tag in XML**

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

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

</bean>

* bean: Defines a Spring-managed Java object
* id: Unique name to refer the bean
* class: Fully-qualified Java class name
* property: Sets value for fields via setters
* name: Field name
* value: Value to assign
* **ApplicationContext vs ClassPathXmlApplicationContext**
* **ApplicationContext**: Central Spring container interface
* **ClassPathXmlApplicationContext**: Loads bean definitions from an XML file on the classpath

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

* **What Happens When context.getBean()**
* Spring loads XML
* Finds <bean> with id="country"
* Creates instance of Country class
* Sets properties using setters (setCode(), setName())
* Returns the fully-initialized object

**Exercise 3: Hello World RESTful Web Service**

**Step 1. Open your existing project**

**Step 2. Add Spring Web Dependency**

**pom.xml**

<dependency>

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

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

</dependency>

**Step 3. Change Port to 8083**

Create a file **application.properties** inside: src/main/resources

Add:

server.port=8083

**Step 4. Create Necessary File**

**HelloController.java**

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 LOGGER = LoggerFactory.getLogger(HelloController.class);

@GetMapping("/hello")

public String sayHello() {

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

String message = "Hello World!!";

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

return message;

}

}

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

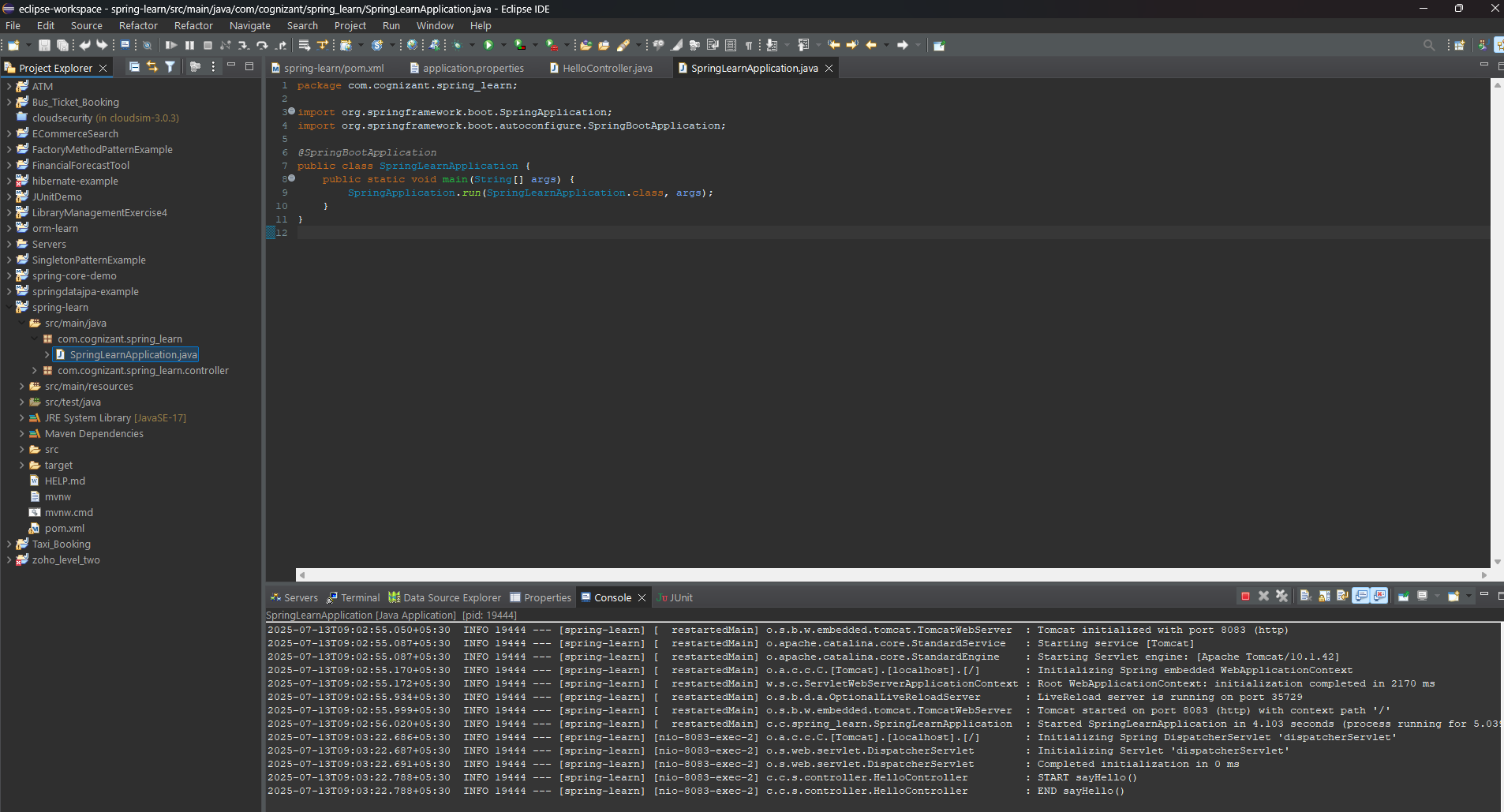
public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

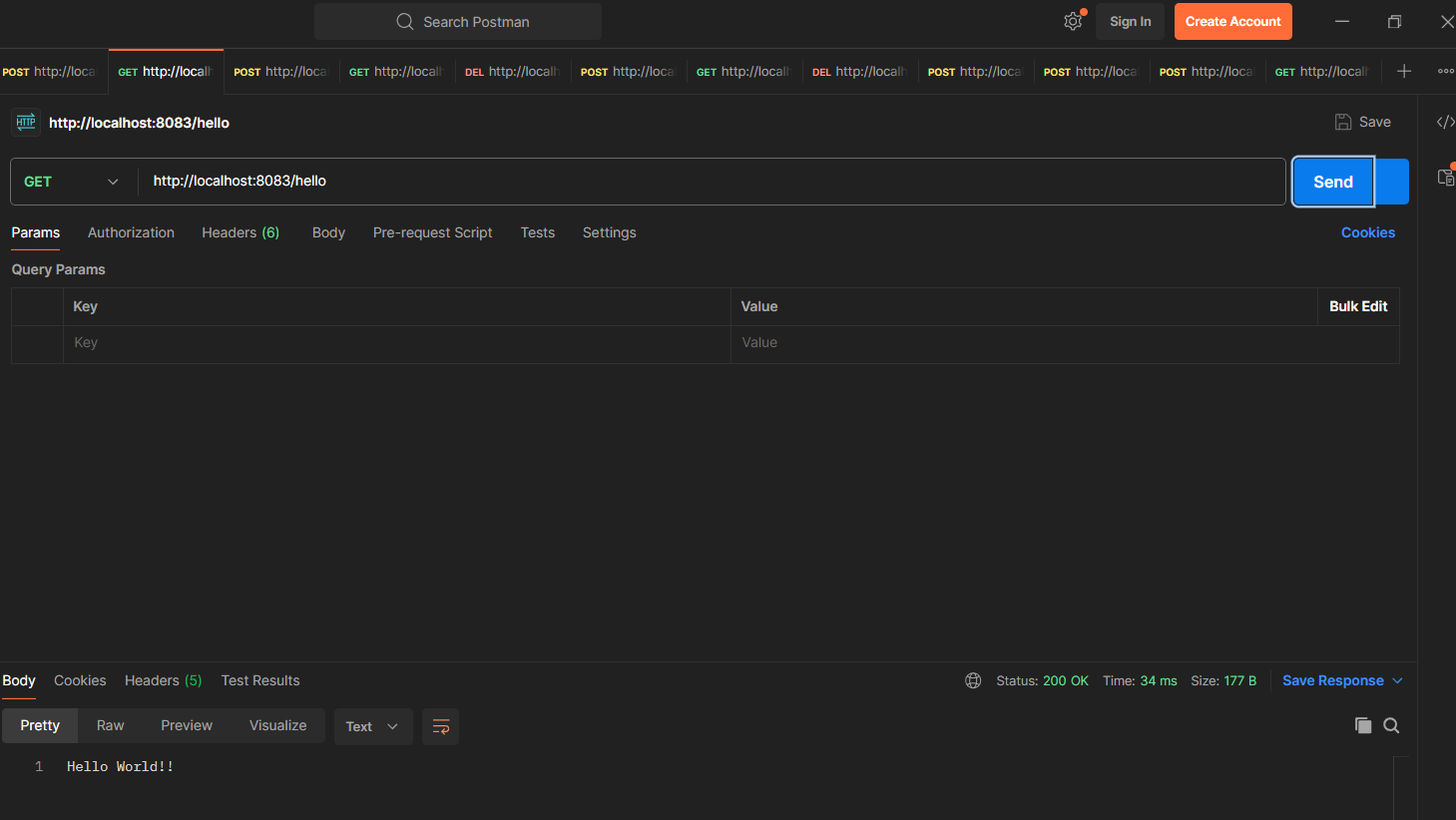
**Output :**

****

* **Test in Browser**

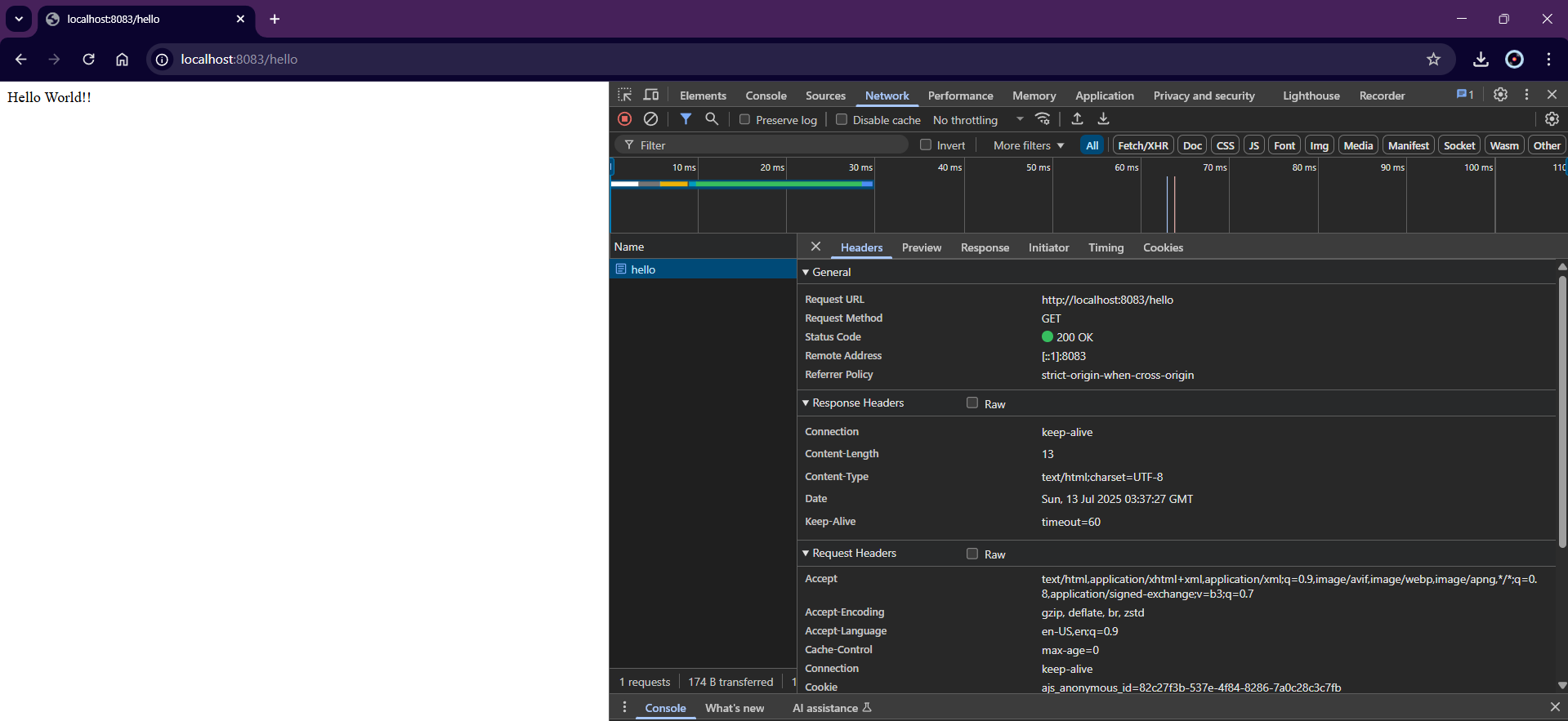


* **Test in Postman**

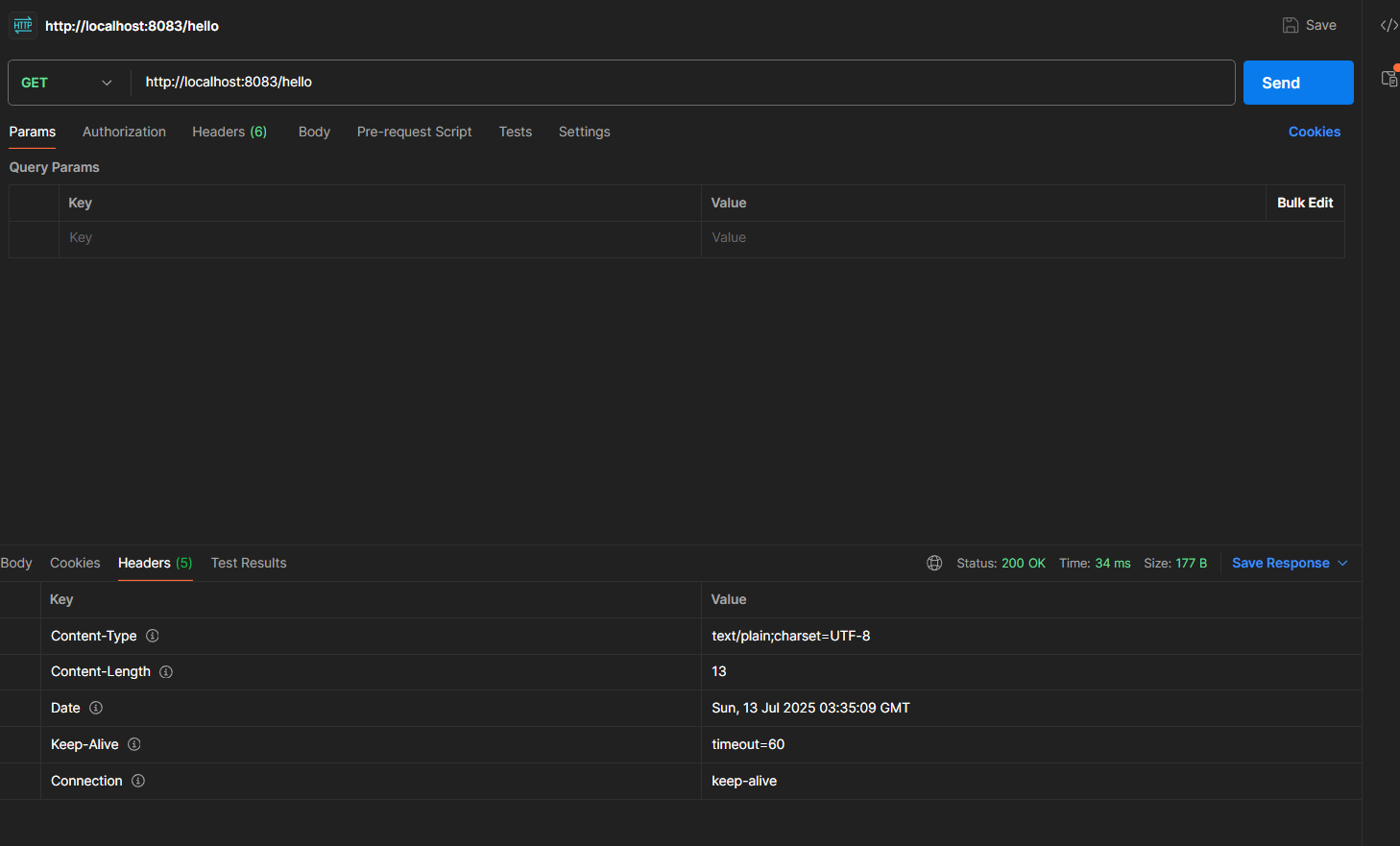


**SME Discussion**

* **In Browser (Chrome DevTools)**

****

* **In Postman**

****

**Exercise 4: REST - Country Web Service**

**Step 1:Continue project of Exercise 2**

**Step 2. Verify (or) create files**

**country.xml**

<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\_core\_demo.Country">

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

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

</bean>

</beans>

**Country.java**

package com.cognizant.spring\_core\_demo;

public class Country {

private String code;

private String name;

public Country() {

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

}

public String getCode() {

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

return code;

}

public void setCode(String code) {

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

this.code = code;

}

public String getName() {

System.out.println("Inside 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 + "]";

}

}

**CountryController.java**

package com.cognizant.spring\_learn.controller;

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;

import com.cognizant.spring\_core\_demo.Country;

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

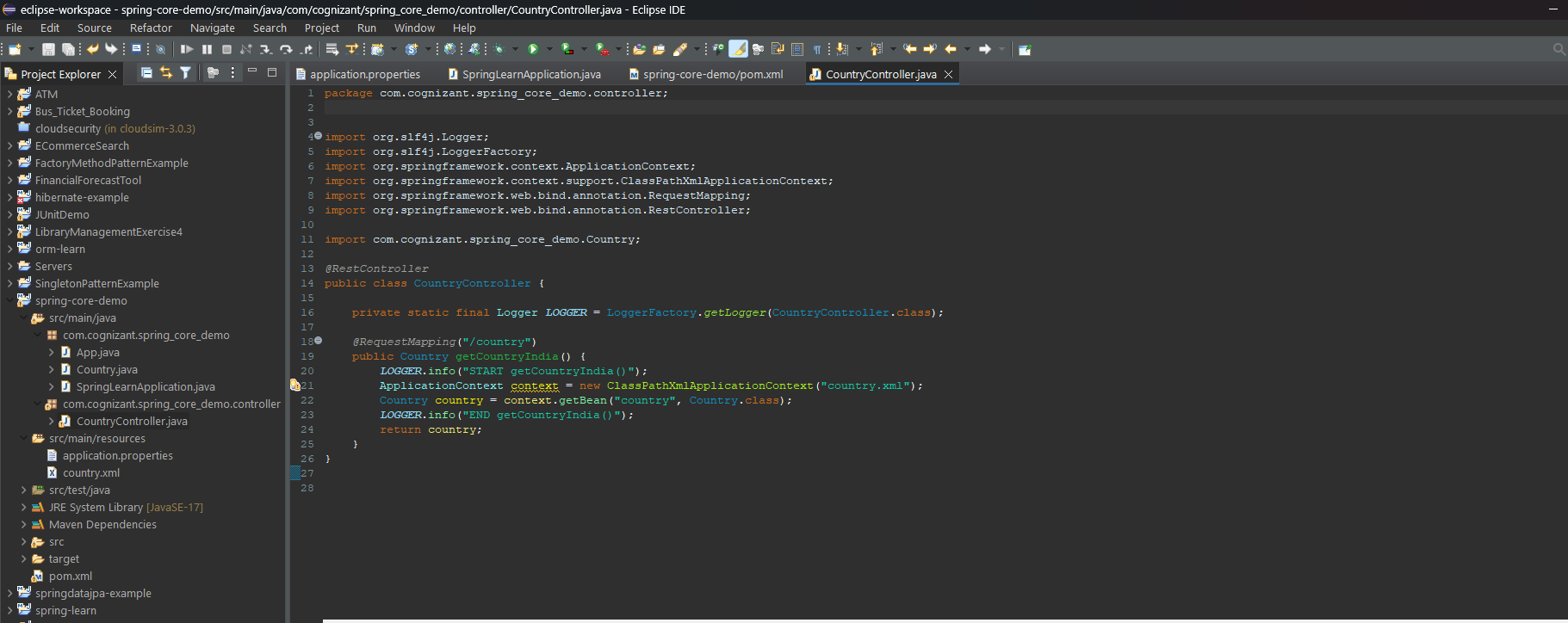
LOGGER.info("END getCountryIndia()");

return country;

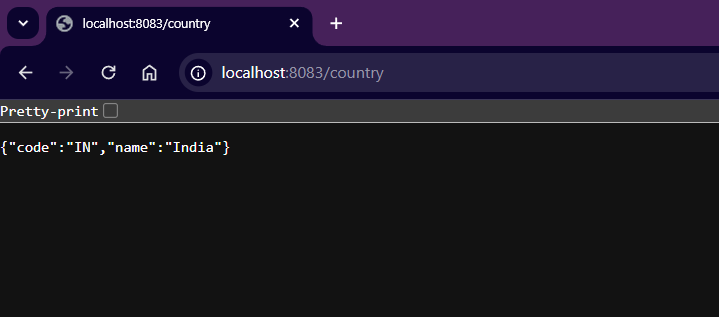
}

}

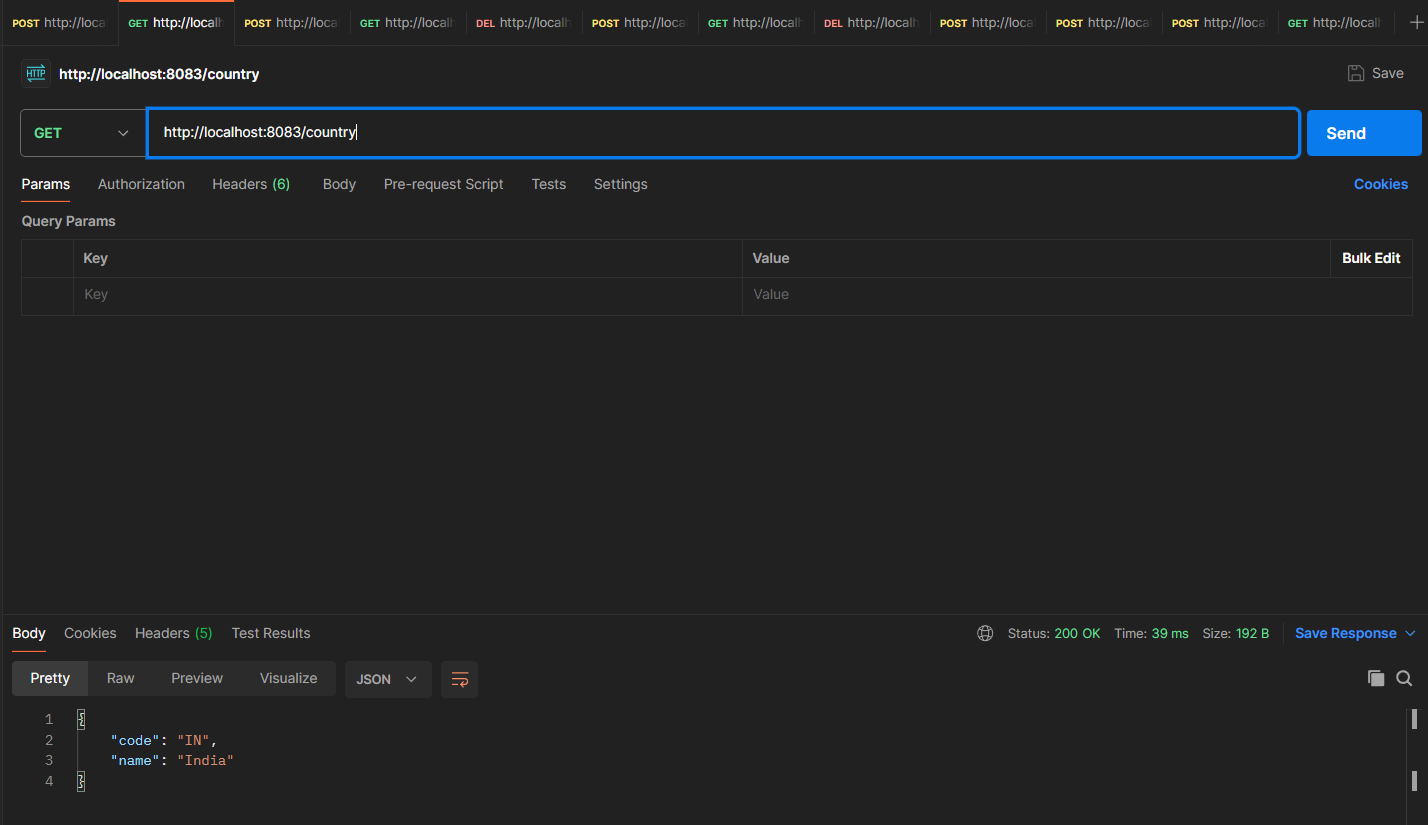
**Output :**

****

* **Test in Browser**

****

* **Postman**



## SME Explanation

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

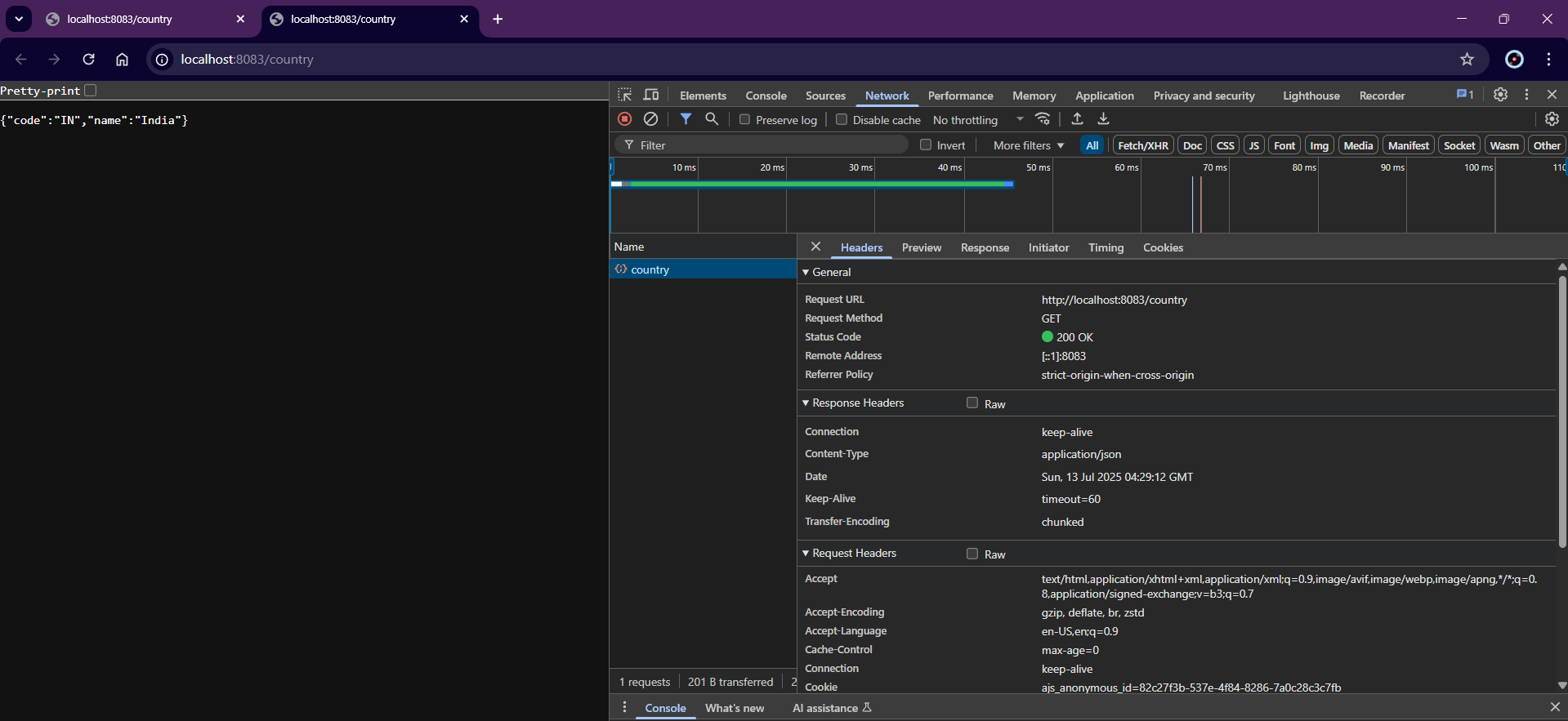
* When you call /country, Spring finds the method with @RequestMapping("/country").
* It creates a Spring context from country.xml.
* It gets the bean with ID country (i.e., India).
* The Country object is returned to the caller.

**2. How is the Country object converted to JSON?**

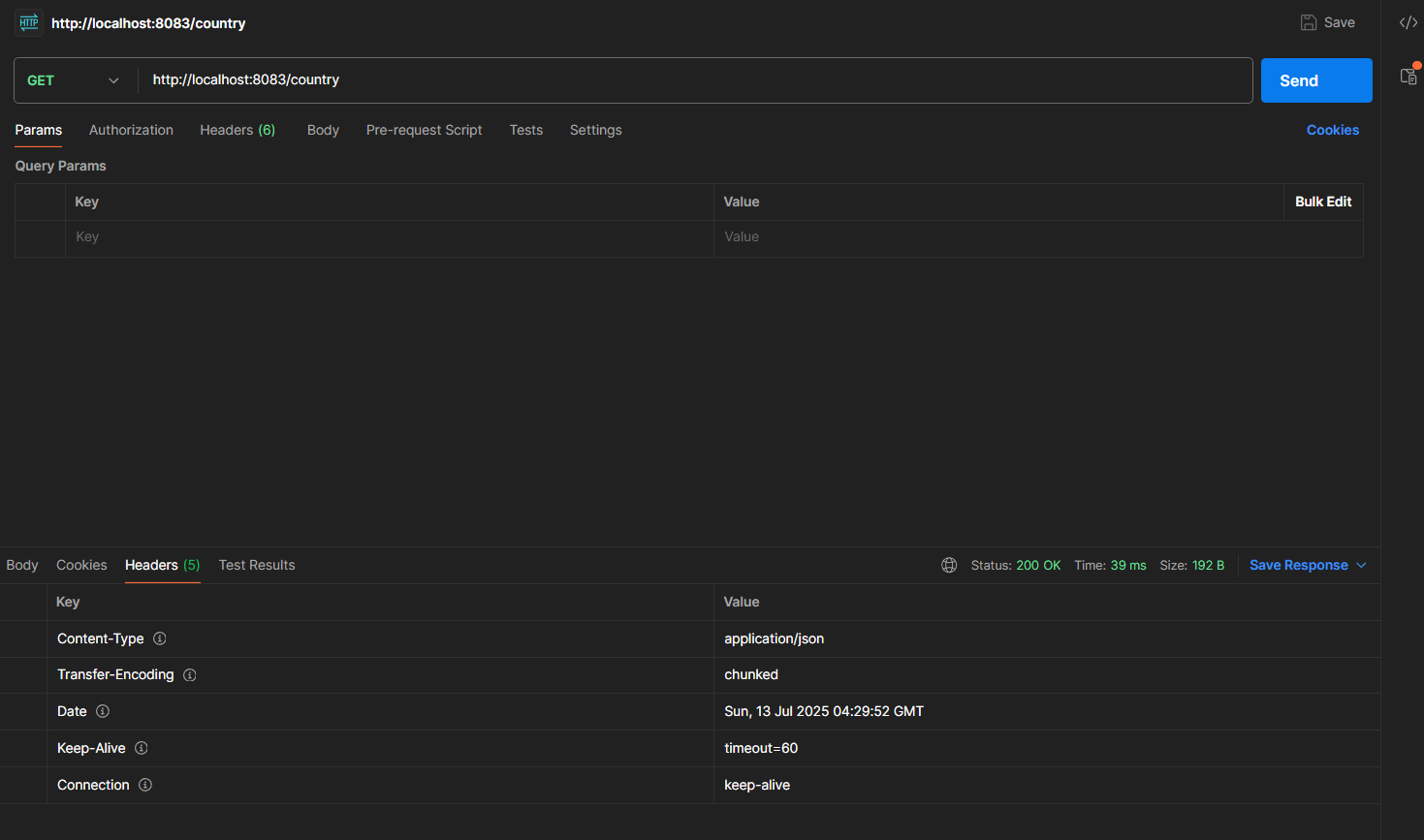
Spring Boot uses **Jackson (JSON converter)** under the hood:

* It checks the return type (Country)
* Uses **Jackson's ObjectMapper** to convert the POJO into JSON
* Automatically sets Content-Type: application/json in response

**3. In Chrome Dev Tools (Network Tab)**

****

**4. In Postman**



**Exercise 5: REST - Get country based on country code**

(Updates made on the previous project)

**Step 1. Update country.xml to include multiple countries**

<?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">

<!-- Country list -->

<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="DE"/>

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

</bean>

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

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

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

</bean>

</list>

</constructor-arg>

</bean>

</beans>

**Step 2. Create Country .java**

package com.cognizant.spring\_learn.model;

public class Country {

private String code;

private String name;

public Country() {

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

}

public String getCode() {

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

return code;

}

public void setCode(String code) {

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

this.code = code;

}

public String getName() {

System.out.println("Inside 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 + "]";

}

}

**Step 3. Create CountryService.java**

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 java.util.List;

@Service

public class CountryService {

public Country getCountry(String code) {

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

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

return countryList.stream()

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

.findFirst()

.orElse(null); // You can also throw a custom exception here if not found

}

}

**Step 4. Create the REST Controller**

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.model.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;

}

}

**Step 5. Verify application properties**

server.port=8083

**Step 6. SpringLearnApplication.java**

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

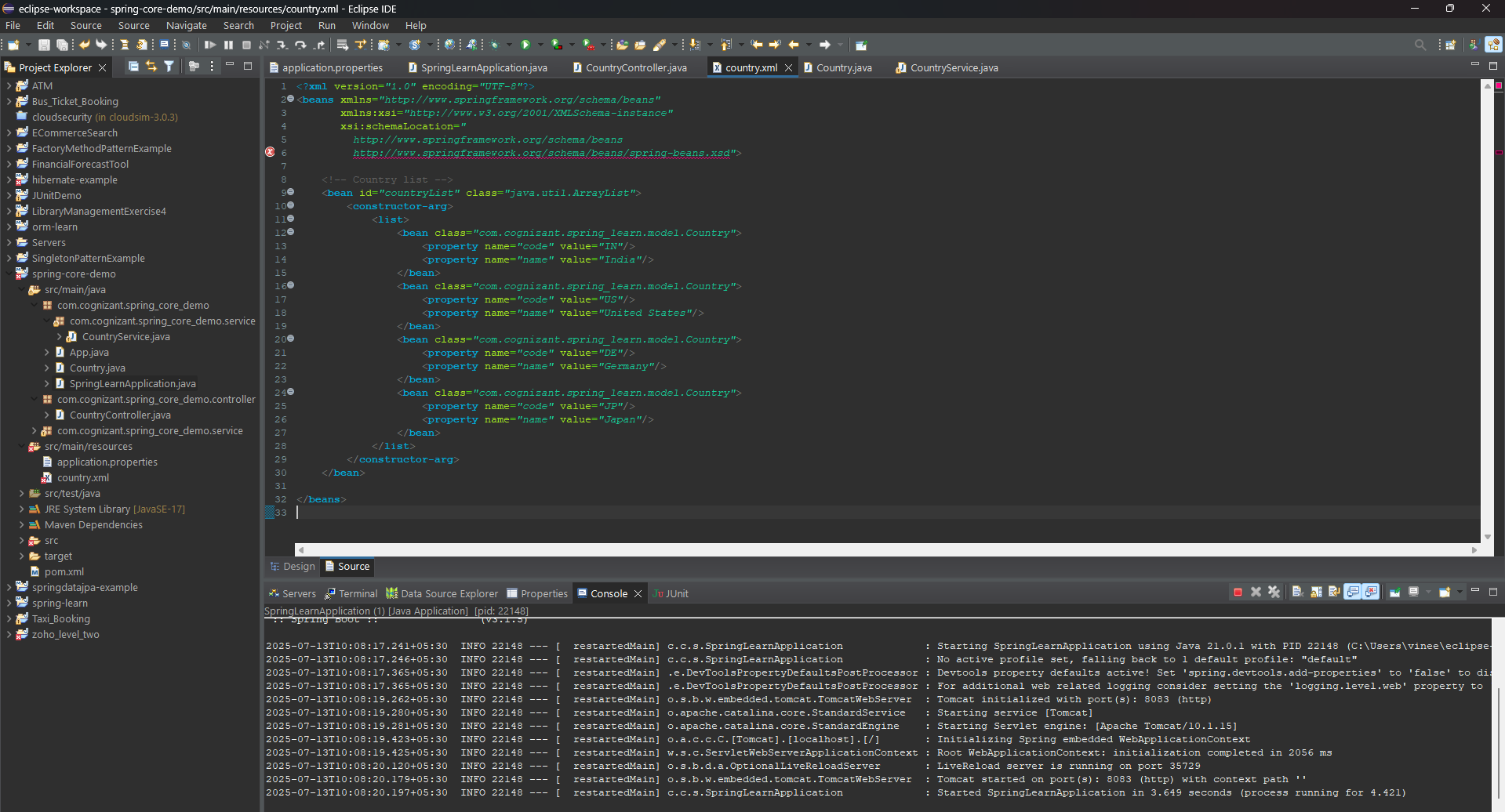
public static void main(String[] args) {

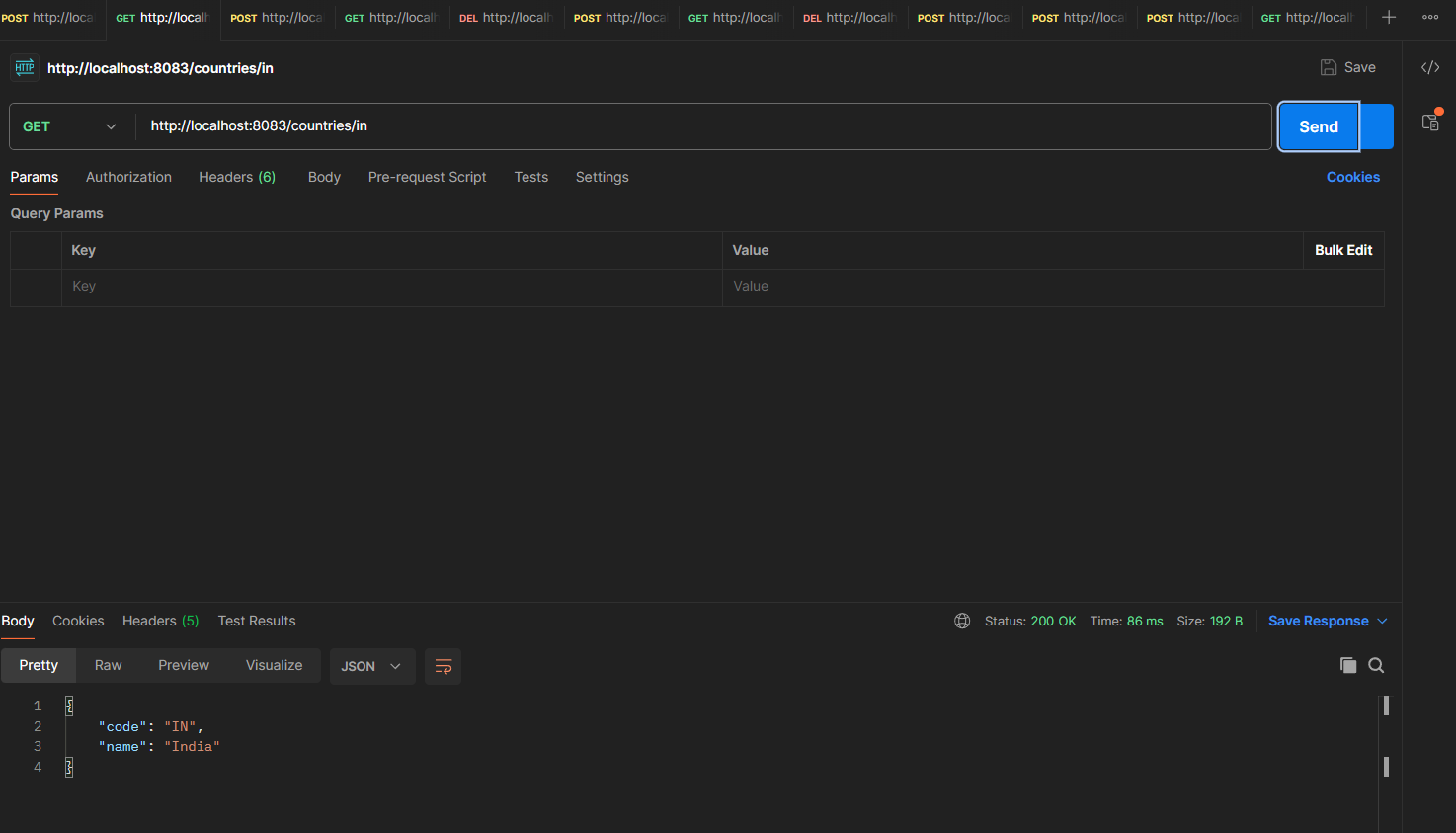
SpringApplication.run(SpringLearnApplication.class, args);

}

}

**Output :**



**Postman Output :**

**Exercise 5: REST - Get country based on country code**

## Step 1: Open Spring Initializr

* Open your web browser.
* Visit: <https://start.spring.io>
  + Fill the fields as follows:
    - **Project**: Maven
    - **Language**: Java
    - **Spring Boot Version**: Default
    - **Group**: com.example
    - **Artifact**: jwt-auth
    - **Name**: spring-learn
    - **Description**: Spring Web Project
    - **Packaging**: Jar
  + Click on **"Add Dependencies"**, and select:
    - **Spring Web**
    - **Spring Boot DevTools**
    - **Spring Security**
* **Download Folder and Extract**

**Step 2: Import Project in Eclipse**

* Open **Eclipse IDE**.
* Go to File > Import.
* Choose Maven > Existing Maven Projects > Click Next.
* Click Browse and select the folder where you extracted the project (spring-learn).
* Click Finish.

**Step 4: Add JWT Dependency**

**pom.xml**

<!-- JWT Library -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**Step 4: Create Necessary Files**

**SecurityConfig.java**

package com.example.jwtauth.config;

import org.springframework.context.annotation.Bean;

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

import org.springframework.security.web.SecurityFilterChain;

import static org.springframework.security.config.Customizer.withDefaults;

import org.springframework.context.annotation.Configuration;

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf(csrf -> csrf.disable())

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll()

.anyRequest().authenticated()

)

.httpBasic(withDefaults()); // enable HTTP Basic Auth

return http.build();

}

}

**JwtUtil.java**

package com.example.jwtauth.util;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import java.util.Date;

public class JwtUtil {

private static final String SECRET\_KEY = "mysecret";

public static String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 10)) // 10 mins

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

}

**AuthenticationController.java**

package com.example.jwtauth.controller;

import com.example.jwtauth.util.JwtUtil;

import org.springframework.http.ResponseEntity;

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

import java.util.Base64;

import jakarta.servlet.http.HttpServletRequest;

@RestController

public class AuthenticationController {

@GetMapping("/authenticate")

public ResponseEntity<?> authenticate(HttpServletRequest request) {

// Read Authorization Header

String header = request.getHeader("Authorization");

if (header == null || !header.startsWith("Basic ")) {

return ResponseEntity.status(401).body("Missing Authorization Header");

}

// Decode base64 encoded username:password

String base64Credentials = header.substring("Basic ".length());

byte[] credDecoded = Base64.getDecoder().decode(base64Credentials);

String credentials = new String(credDecoded);

String[] values = credentials.split(":", 2);

String username = values[0];

String password = values[1];

// Hardcoded credentials for testing

if ("user".equals(username) && "pwd".equals(password)) {

String token = JwtUtil.generateToken(username);

return ResponseEntity.ok().body("{\"token\":\"" + token + "\"}");

} else {

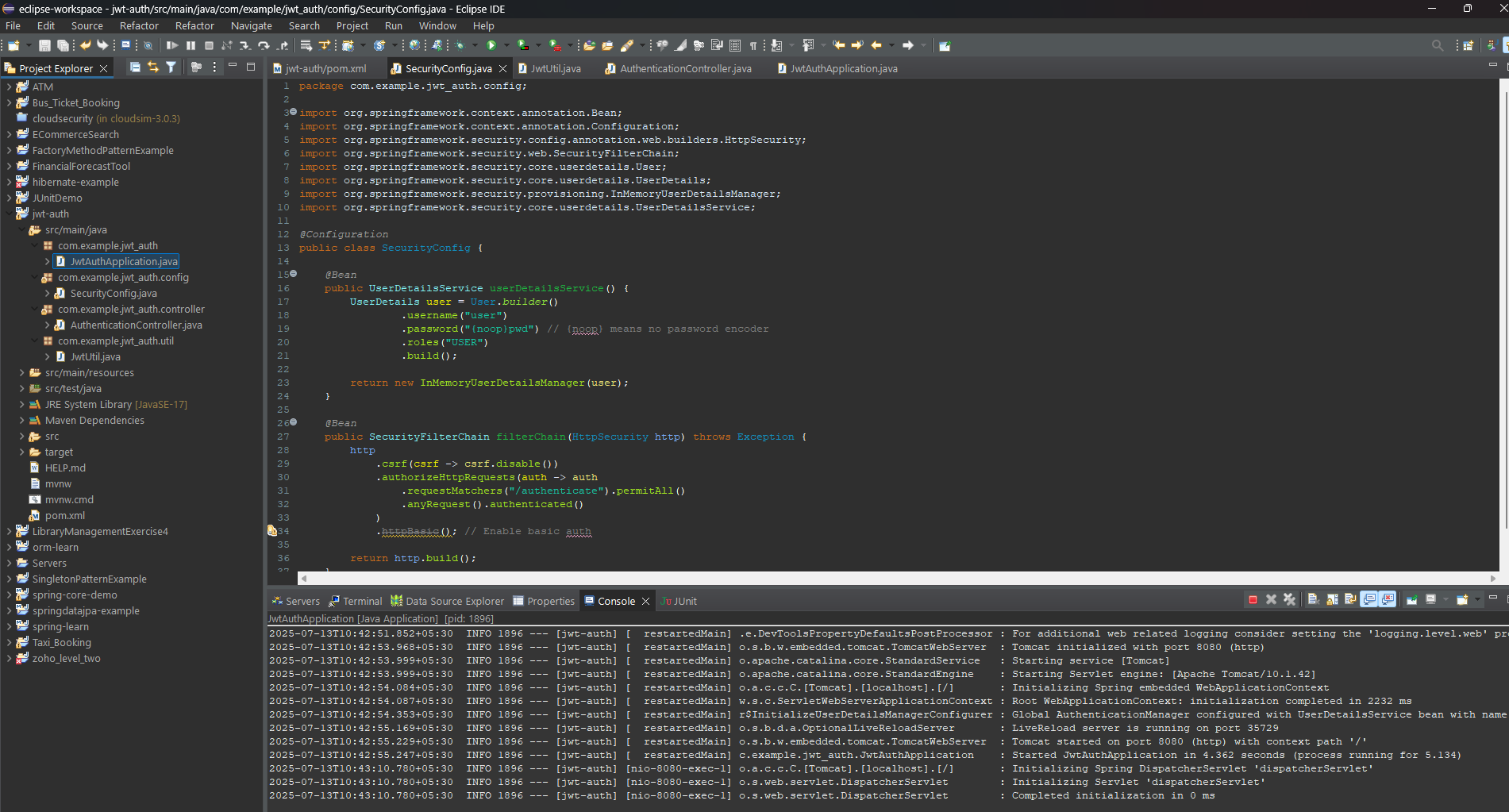
return ResponseEntity.status(401).body("Invalid Credentials");

}

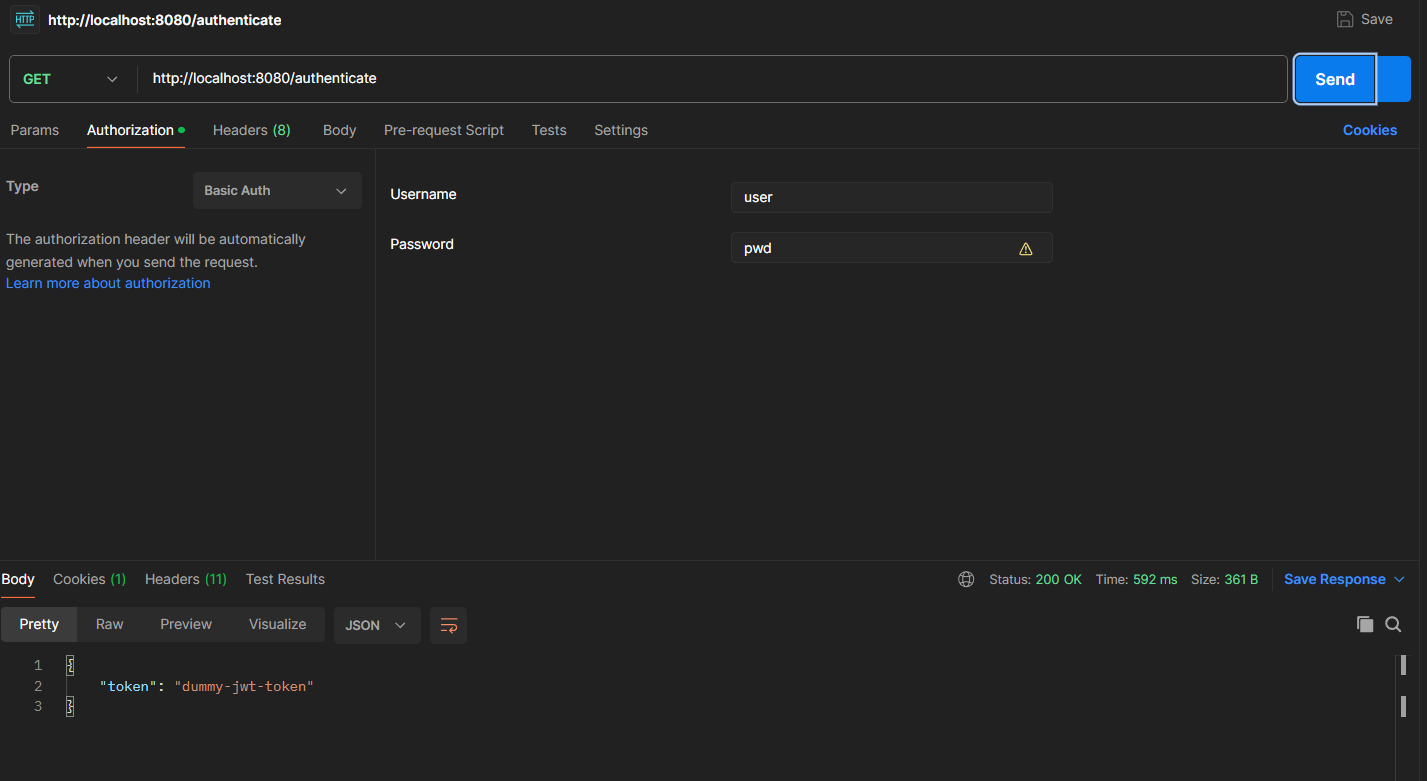
}

}

**Output :**

****

* **Postman Test**

****

**SME Explaination**

| **Concept** | **Explanation** |
| --- | --- |
| Authorization: Basic ... | This is a base64 encoded username:password sent in the request header |
| SecurityConfig | Disables CSRF and sets up HTTP Basic Auth |
| JwtUtil | Generates a signed JWT using HS256 algorithm |
| authenticate() | Reads and decodes auth header, validates, and returns token |
| Response Headers | Contains Content-Type: application/json and other HTTP headers |
| Postman Headers Tab | Shows headers sent by server like Content-Type, Content-Length, etc |