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

**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">

<!-- Single Country (India) -->

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

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

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

</bean>

<!-- List of Countries -->

<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="DE" />

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

</bean>

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

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

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

</bean>

</list>

</constructor-arg>

</bean>

</beans>

**CountryController.java**

package com.cognizant.spring\_learn.controller;

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.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

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

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

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);

@Autowired

private CountryService countryService;

@RequestMapping("/country")

public Country getCountryIndia() {

LOGGER.info("START");

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

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

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

LOGGER.info("END");

return country;

}

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

public Country getCountry(@PathVariable String code) {

LOGGER.info("START - Get country with code: {}", code);

Country country = countryService.getCountry(code);

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

LOGGER.info("END");

return country;

}

}

**CountryService.java**

package com.cognizant.spring\_learn.service;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.stereotype.Service;

import com.cognizant.spring\_learn.Country;

*@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))

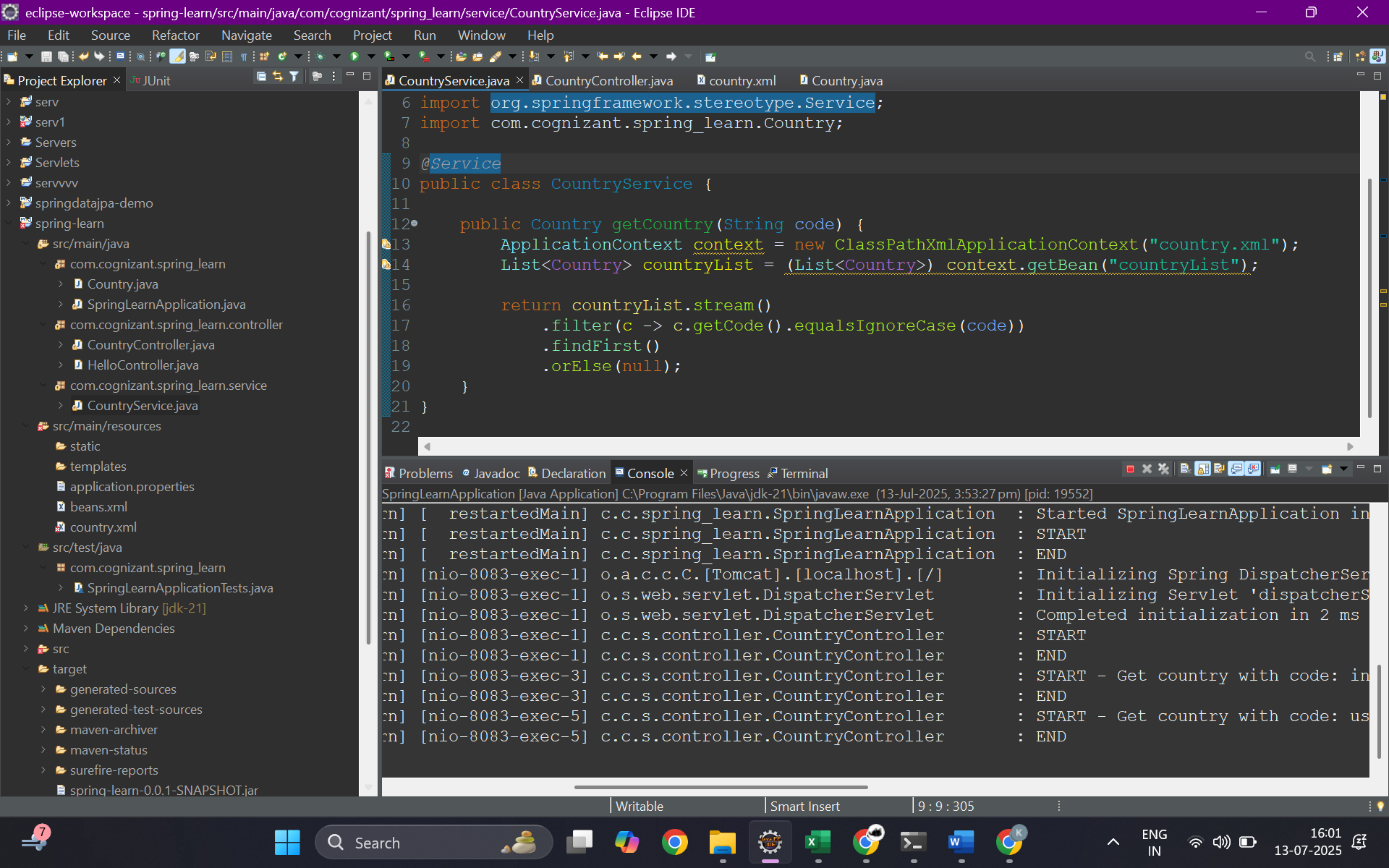
.findFirst()

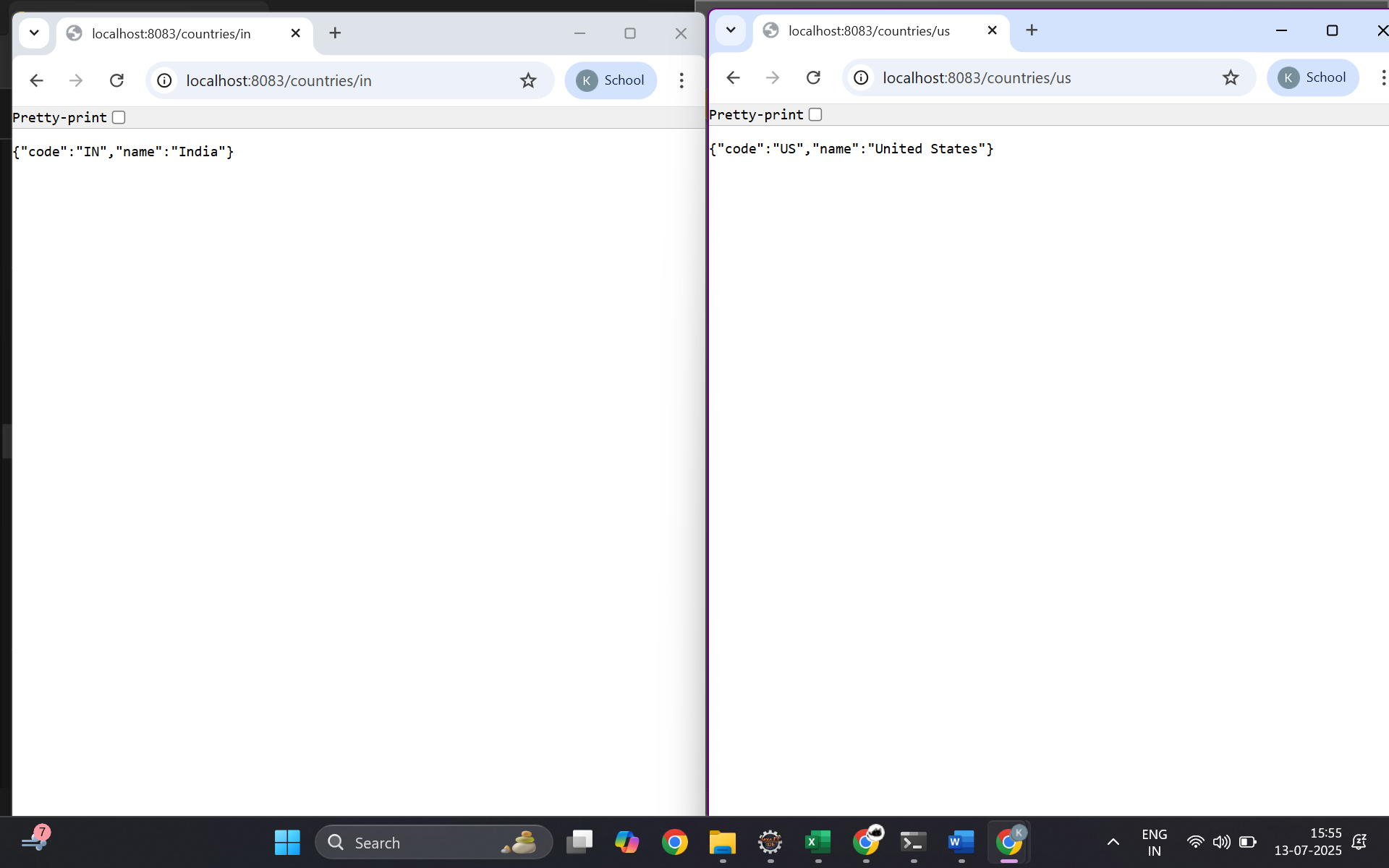
.orElse(null);

}

}

**Output:**





**SME**:

In this hands-on, we created a RESTful web service that fetches country details based on a country code passed dynamically through the URL. The controller method is mapped using @GetMapping("/countries/{code}"), and the {code} is extracted using @PathVariable. The method then calls CountryService.getCountry(code), which loads a list of countries from the country.xml Spring configuration file and searches for the country with the matching code in a case-insensitive manner. The matching Country bean is returned from the service and sent as a JSON response to the client. The automatic conversion of the Country Java object to JSON format is handled by Spring Boot’s default Jackson library, which is enabled through the use of @RestController. When the object is returned from the controller, Jackson serializes the fields (code and name) into JSON, which becomes the response body. In a web browser, the request and response headers can be viewed through the Developer Tools’ Network tab, showing details like request URL, method (GET), and content type (application/json). Similarly, in Postman, the Headers tab shows the HTTP response metadata like status (200 OK), content type, and other response headers. This hands-on demonstrates how Spring Boot simplifies the creation of REST endpoints and automates object-to-JSON conversion while supporting dynamic path-based routing.