**WEEK-04**

**HANDS-ON SOLUTIONS**

**2. SPRING-REST-HANDSON**

**EXERCISE-01:**

**HELLO WORLD RESTFUL WEB SERVICE**

Write a REST service in the spring learn application created earlier, that returns the text "Hello World!!" using Spring Web Framework. Refer details below:

Method:

GET URL: /hello Controller: com.cognizant.spring-learn.controller.HelloController

Method Signature: public String sayHello()

Method Implementation: return hard coded string "Hello World!!"

Sample Request: <http://localhost:8083/hello>

Sample Response: Hello World!!

IMPORTANT NOTE: Don't forget to include start and end log in the sayHello() method.

Try the URL http://localhost:8083/hello in both chrome browser and postman.

SME to explain the following aspects:

· In network tab of developer tools show the HTTP header details received.

· In postman click on "Headers" tab to view the HTTP header details received.

**SOLUTION:**

As part of Exercise 01 of the Spring REST hands-on,

I developed a simple RESTful web service using Spring Boot that returns the message "Hello World!!".

I began by creating a new controller class named `HelloController` under the package `com.cognizant.spring\_learn.controller`.

Inside this class, I implemented a `@RestController` with a `@GetMapping("/hello")` method named `sayHello()` that returns the hardcoded string "Hello World!!".

Logging statements were also added at the beginning and end of the method using SLF4J to track execution flow.

I ensured the application was configured to run on port 8085 by updating the `application.properties` file with the entry `server.port=8085`.

After running the `SpringLearnApplication` class, the embedded Tomcat server started on the specified port, and the REST endpoint was successfully exposed.

I verified the functionality by sending a GET request to `http://localhost:8085/hello` using both Chrome browser and Postman.

In both cases, the response returned was as expected: "Hello World!!".

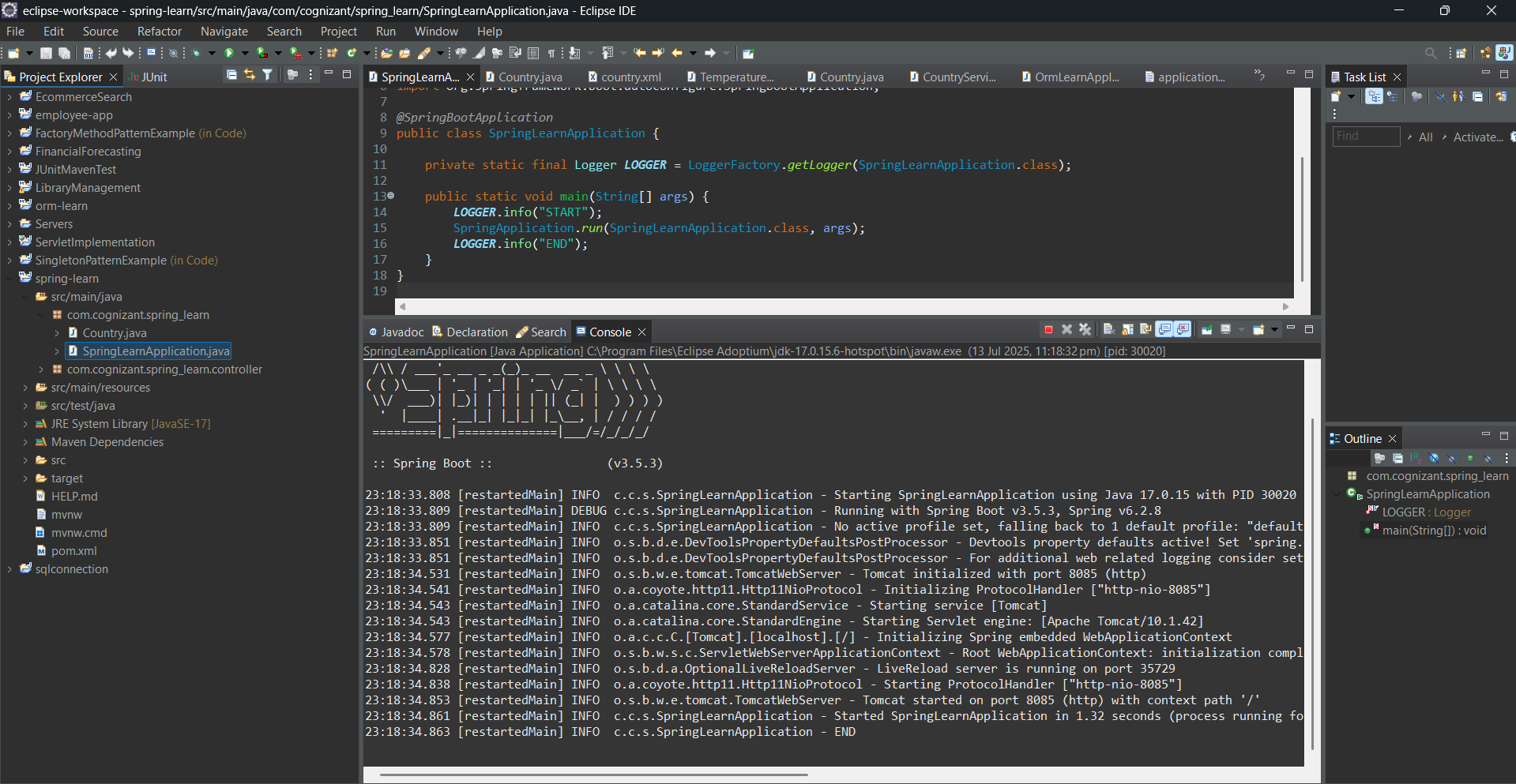
In Chrome, I used the Developer Tools → Network tab to inspect the HTTP headers and confirmed that the request and response were correctly formed.

In Postman, the “Headers” tab also displayed relevant HTTP details, reaffirming the service behavior.

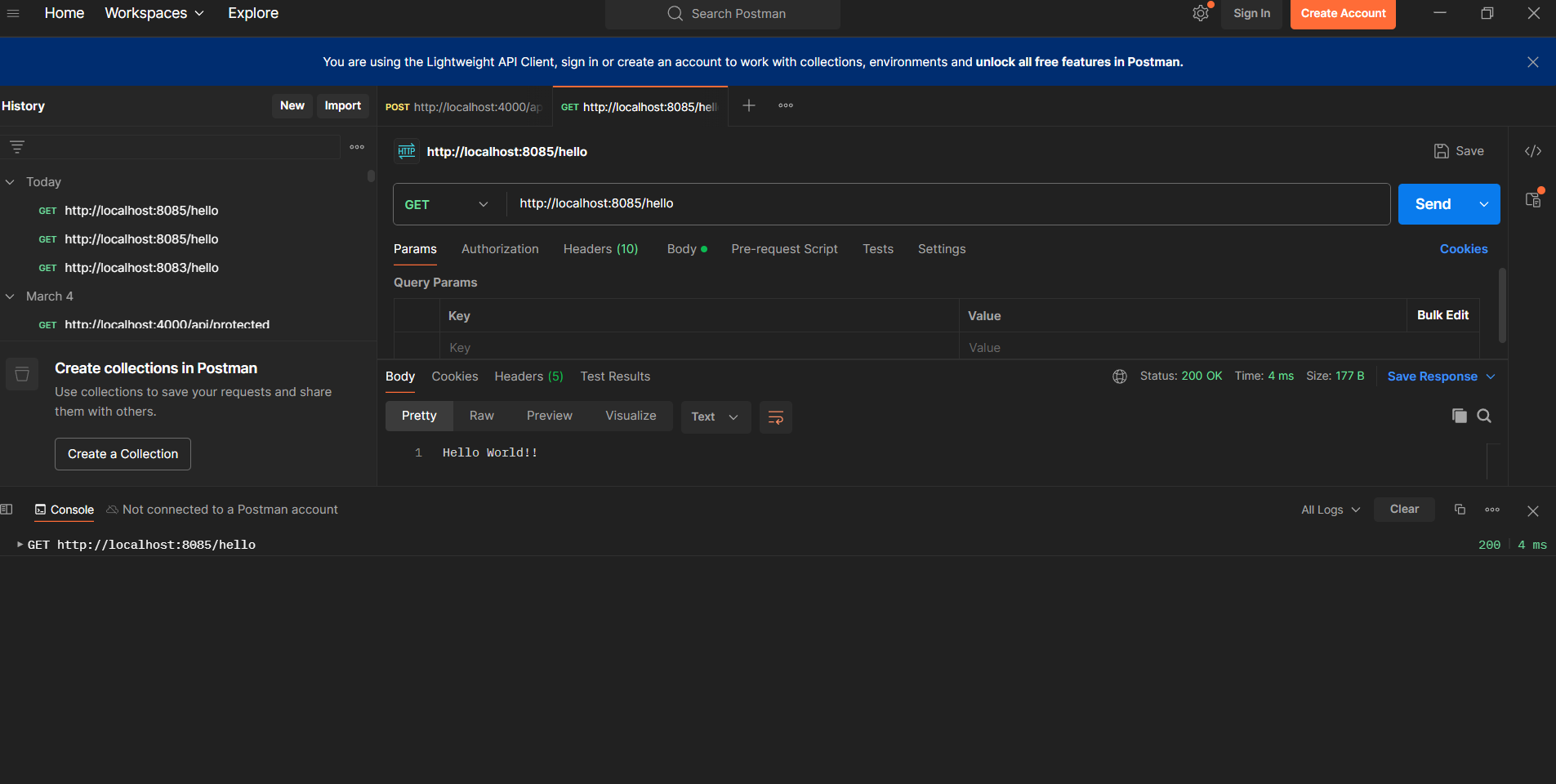
This exercise helped reinforce key concepts of building a RESTful endpoint using Spring Boot, understanding HTTP methods, request mapping, and logging best practices.

The successful implementation validated that the application setup was complete and that the REST service worked as intended.

**OUTPUT:**

****

****

****

**EXERCISE-02:**

**REST - Country Web Service**

Write a REST service that returns India country details in the earlier created spring learn application.

URL: /country

Controller: com.cognizant.spring-learn.controller.

CountryController Method

Annotation: @RequestMapping

Method Name: getCountryIndia()

Method Implementation: Load India bean from spring xml configuration and return

Sample Request: <http://localhost:8083/country>

Sample Response:

{

"code": "IN",

"name": "India"

}

SME to explain the following aspects:

· What happens in the controller method?

· How the bean is converted into JSON reponse?

· In network tab of developer tools show the HTTP header details received.

· In postman click on "Headers" tab to view the HTTP header details received.

**SOLUTION:**

As part of Exercise 2, I implemented a RESTful web service in the existing Spring Boot application to return the country details for India.

The goal of the task was to expose a `/country` endpoint that fetches a `Country` object from an XML configuration and returns it as a JSON response.

To achieve this, I reused the `country.xml` configuration file already present in the `src/main/resources` directory.

This file defines a bean named `country` with the code `IN` and name `India`.

The bean is of type `com.cognizant.spring\_learn.Country`, which contains fields for code and name, along with their getters, setters, and a `toString()` method.

I then created a new controller class named `CountryController` under the `com.cognizant.spring\_learn.controller` package.

This class was annotated with `@RestController` to indicate that it handles RESTful HTTP requests.

Inside the controller, I defined the method `getCountryIndia()`, which is mapped to the `/country` URL using the `@RequestMapping` annotation.

The method uses `ClassPathXmlApplicationContext` to load the Spring configuration file and retrieve the `country` bean.

After logging the bean’s details for debugging, the method returns the `Country` object.

The main application class, `SpringLearnApplication`, was updated to use `SpringApplication.run()` to launch the Spring Boot web server.

I ran the application and accessed the `/country` endpoint via both browser and Postman using the URL `http://localhost:8085/country`.

The application returned the expected JSON response:

{ "code": "IN", "name": "India" }

**What happens in the controller method?**

When the `/country` endpoint is hit, the controller method `getCountryIndia()` is executed.

It loads the XML configuration, retrieves the `country` bean from the Spring context, and returns it as the response.

**How is the bean converted to JSON?**

Spring Boot uses the Jackson library (included by default) to automatically convert Java objects to JSON.

Since `@RestController` is used, the returned `Country` object is serialized into JSON and sent as the HTTP response.

**Network Tab / Postman Headers:** The network tab in Chrome Developer Tools and Postman’s “Headers” tab display the HTTP headers of the response.

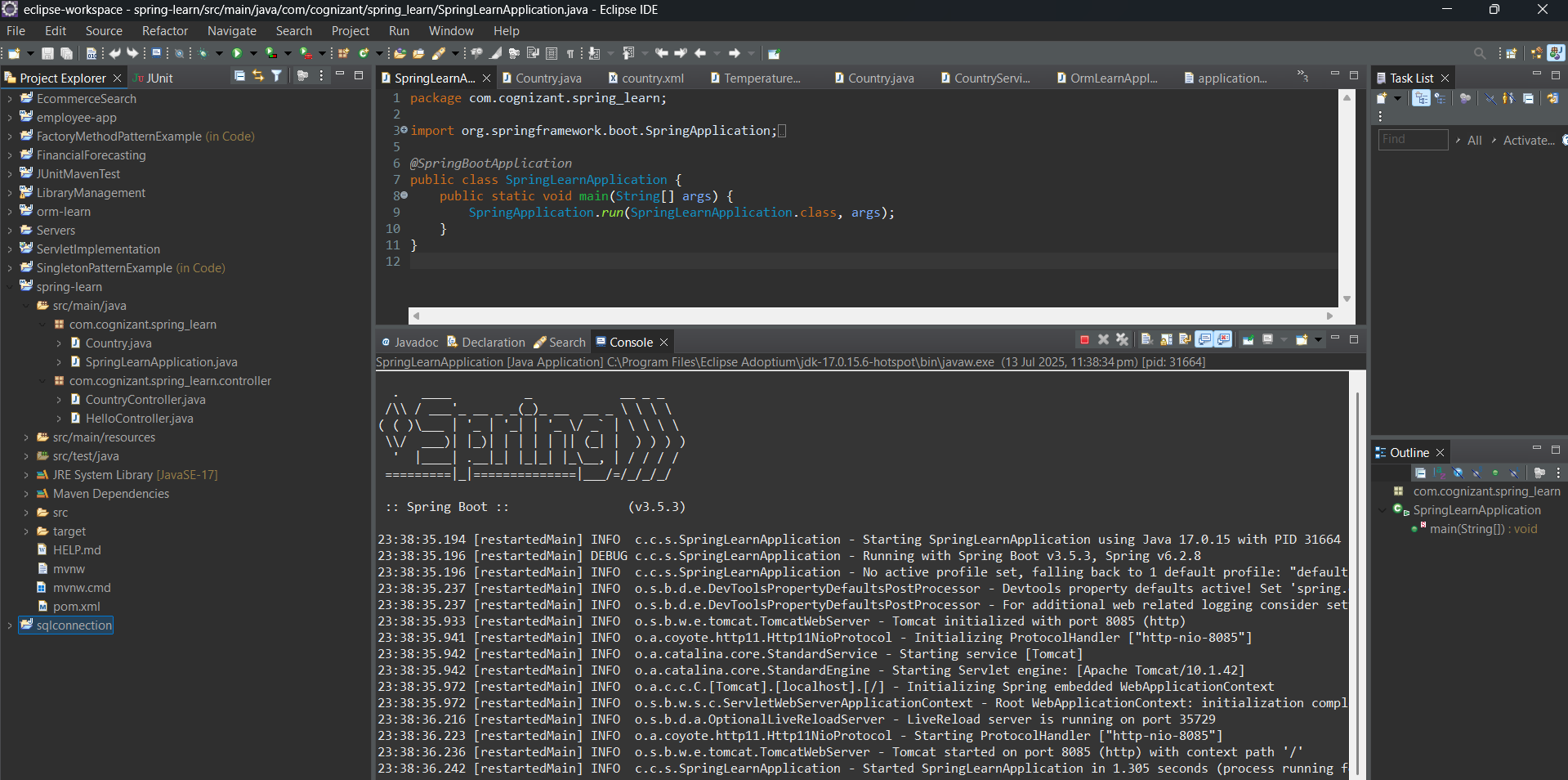
These include: **`Content-Type: application/json`**

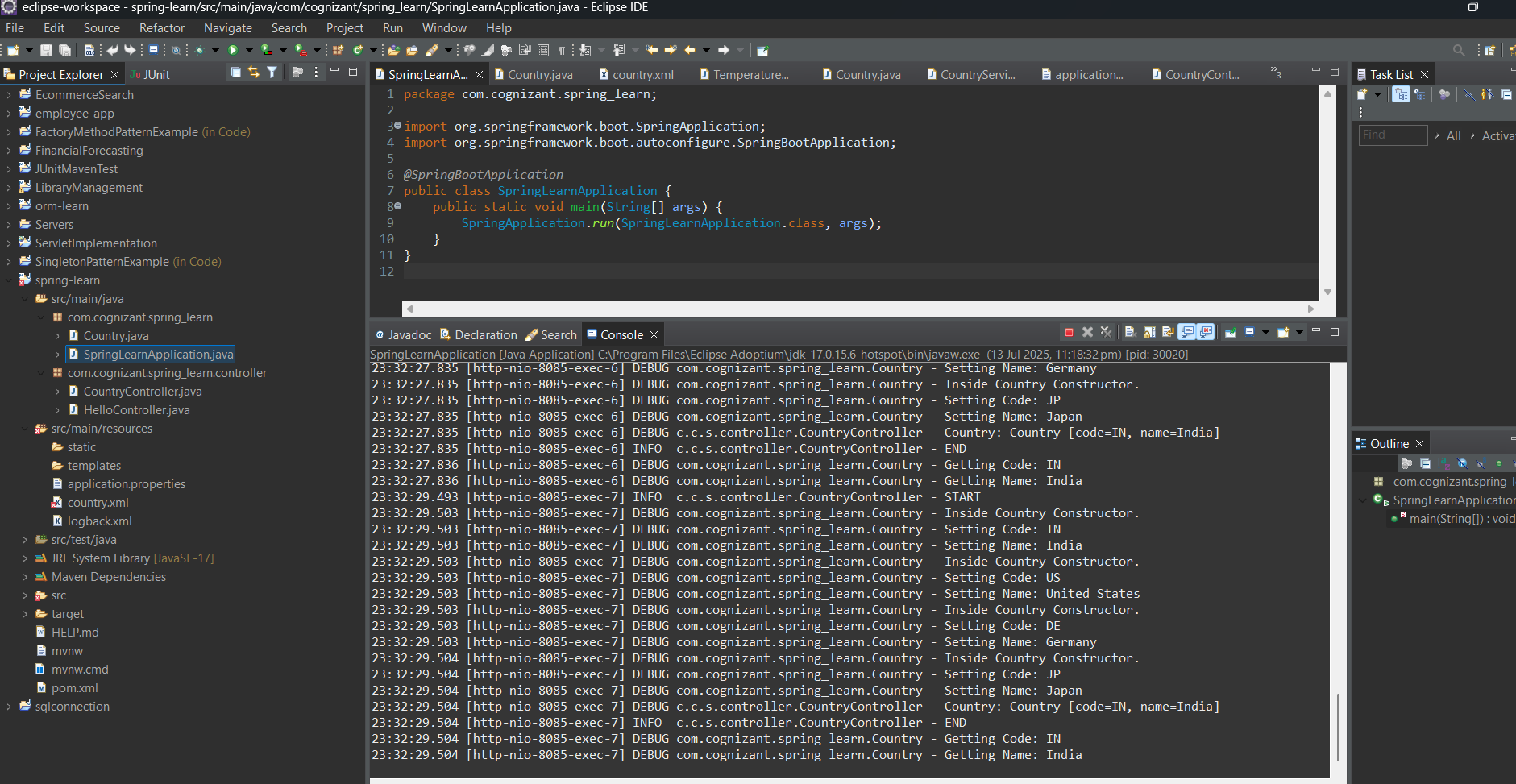
**`Status: 200 OK`**

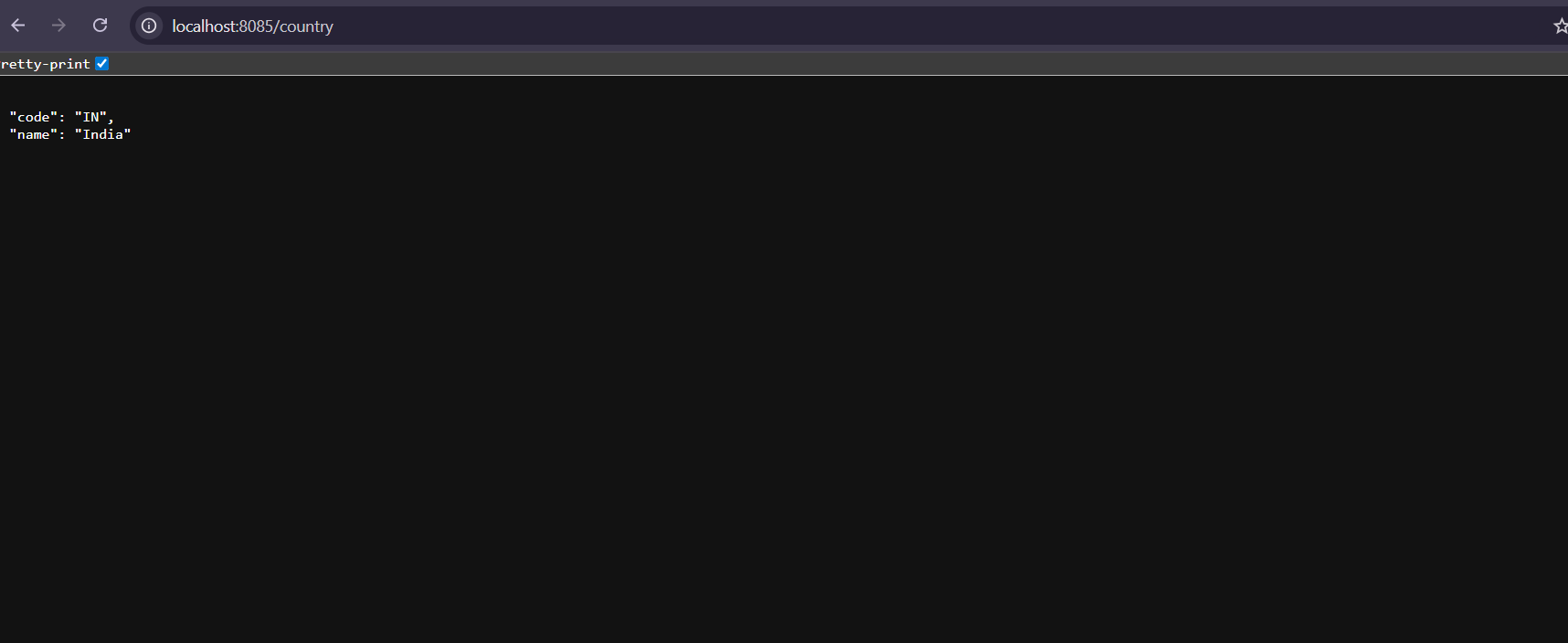
Additional metadata like **`Date`, `Content-Length`,** and server info.

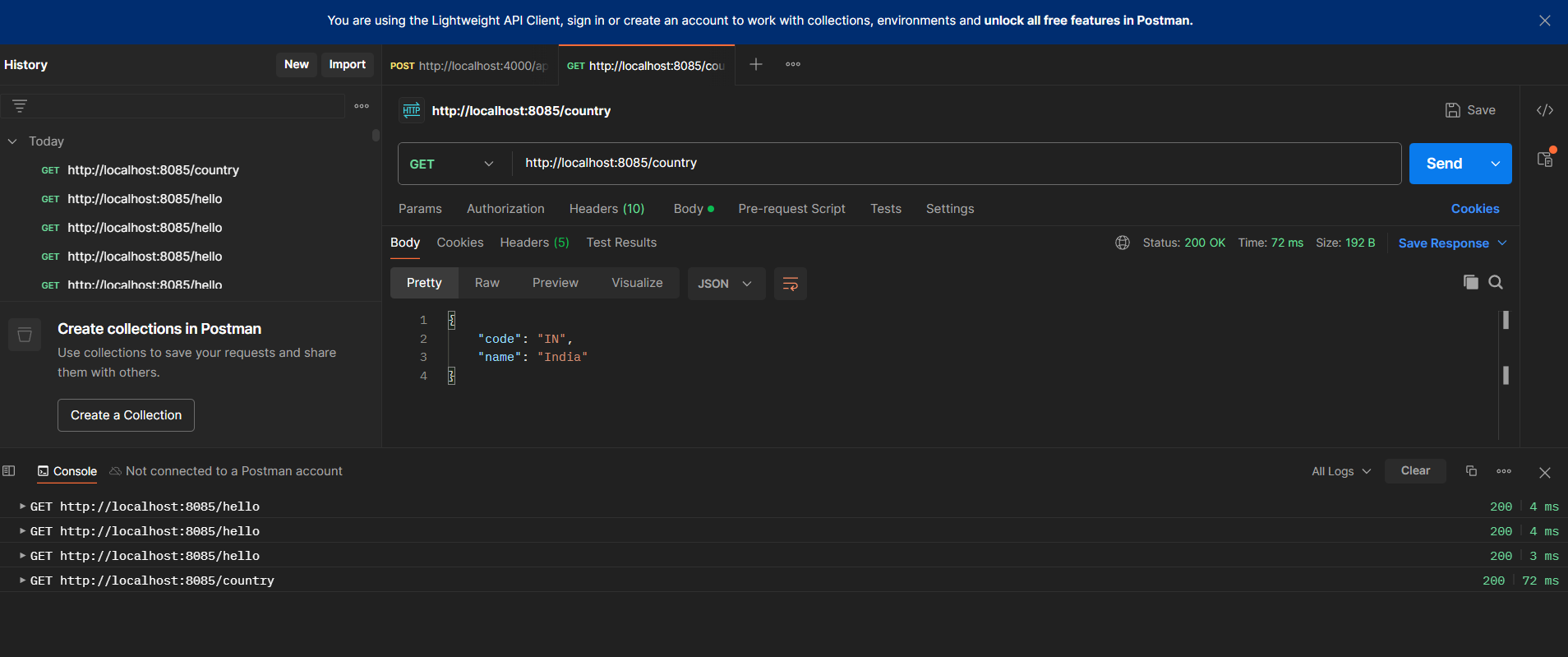
The output matched expectations and the endpoint performed as intended, completing the exercise successfully.

**OUTPUT:**

****

****

****

****

**EXERCISE-03**

**REST - GET COUNTRY BASED ON COUNTRY CODE**

Write a REST service that returns a specific country based on country code.

The country code should be case insensitive.

Controller: com.cognizant.spring-learn.controller.CountryController

Method Annotation: @GetMapping("/countries/{code}")

Method Name: getCountry(String code)

Method Implemetation: Invoke countryService.getCountry(code)

Service Method: com.cognizant.spring-learn.service.

CountryService.getCountry(String code)

Service Method Implementation:

· Get the country code using @PathVariable

· Get country list from country.xml

· Iterate through the country list

· Make a case insensitive matching of country code and return the country.

· Lambda expression can also be used instead of iterating the country list

Sample Request: <http://localhost:8083/country/in>

Sample Response:

{

"code": "IN",

"name": "India"

}

**SOLUTION:**

As part of Exercise 03, I implemented a RESTful web service in the Spring Boot project to fetch country details based on a dynamic country code.

The key requirement was to accept country codes in a case-insensitive manner and return the corresponding country's information from the Spring configuration.

To begin with, I updated the country.xml configuration file to include multiple country beans:

India (IN), United States (US), Germany (DE), and Japan (JP), each with a unique ID and respective ISO code and name.

Next, I created a new service class CountryService under the package com.cognizant.spring\_learn.service.

Within this class, I defined a method getCountry(String code) that reads all beans of type Country using ApplicationContext, retrieves them as a list, and iterates (or uses lambda expressions) to match the requested code with the stored codes in a case-insensitive manner.

If a match is found, the corresponding Country object is returned.

In the controller class CountryController, I added the method getCountry(@PathVariable String code) with the annotation @GetMapping("/countries/{code}").

This method invokes the countryService.getCountry(code) and returns the country object as the HTTP response.

Since the controller is annotated with @RestController, Spring Boot automatically converts the returned Country object into a JSON response using Jackson under the hood.

Upon running the application and testing the endpoint using both a browser and Postman (e.g., http://localhost:8085/countries/in), the correct JSON response was received.

This exercise highlights key concepts of Spring RESTful services, such as:

@GetMapping and @PathVariable: Used to define dynamic URL-based REST endpoints and extract path parameters.

Dependency Injection via ApplicationContext: Used to load multiple beans and manage them effectively.

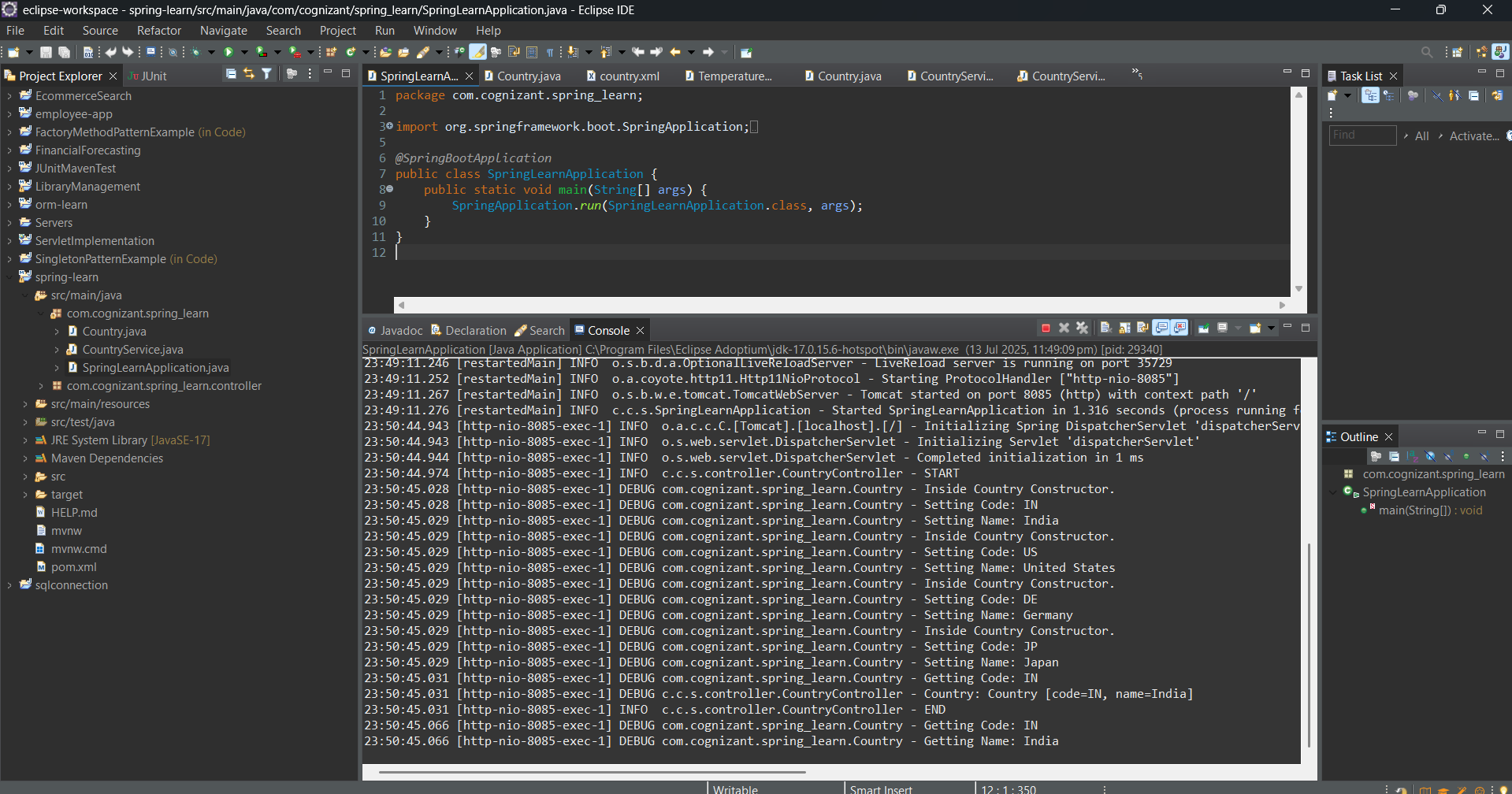
JSON Serialization: Spring Boot automatically uses Jackson to serialize POJOs into JSON format for REST responses.

Case-insensitive matching: Ensures flexibility in user input when querying for country details.

The service worked as intended, correctly matched codes regardless of letter case, and returned the expected structured response.

This marks the successful and complete implementation of Exercise 03.

**OUTPUT:**

****