**Spring REST using Spring Boot 3**

**Question : 01**

**Create a Spring Web Project using Maven**

**1. Create the Project from Spring Initializer**

Go to: <https://start.spring.io>

* **Project Type**: Maven
* **Language**: Java
* **Spring Boot Version**: (Use the default stable version)
* **Group**: com.cognizant
* **Artifact**: spring-learn
* **Name**: spring-learn
* **Package name**: com.cognizant.springlearn
* **Packaging**: Jar
* **Java Version**: 8 or above (Choose based on your JDK)
* **Dependencies**:
  + Spring Boot DevTools
  + Spring Web

Click on **"Generate"**, and a ZIP file will be downloaded.

**2. Extract & Move to Eclipse Workspace**

* Unzip the downloaded file.
* Move the extracted folder (spring-learn) into your Eclipse workspace directory.

**3. Build the Project from Command Line**

Open **Command Prompt/Terminal** and run:

cd path\to\spring-learn

mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

This command cleans previous builds and creates a fresh package with proxy setup (for corporate networks).

**4. Import the Project into Eclipse**

In **Eclipse**:

File > Import > Maven > Existing Maven Projects  
Click **Browse**, select the spring-learn folder > Click **Finish**

**5. Verify Execution of main() Method**

Navigate to:  
 SpringLearnApplication.java inside com.cognizant.springlearn

Add a simple log or System.out.println() in the main() method to verify startup:

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

System.out.println("SpringLearnApplication has started!");

}

**Run the application:**

Right-click on SpringLearnApplication.java > Run As > Java Application

**SME Walkthrough Topics**

**1. src/main/java**

* Contains your actual Java source code.
* We will place your controllers, services, entities, etc. here.
* E.g., com.cognizant.springlearn.SpringLearnApplication.java

**2. src/main/resources**

* Contains configurations like application.properties or application.yml
* You can define properties like:

server.port=8081

logging.level.org.springframework=INFO

**3. src/test/java**

* Used for unit and integration testing.
* Default test class is usually created (e.g., SpringLearnApplicationTests)

**4. SpringLearnApplication.java**

Main class:

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

**5. Purpose of @SpringBootApplication**

This annotation is a combination of:

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

So it bootstraps the whole application.

**6. pom.xml**

This file defines:

* Project info (name, version, etc.)
* Dependencies
* Plugin configurations
* Java version

Example snippet:

xml

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

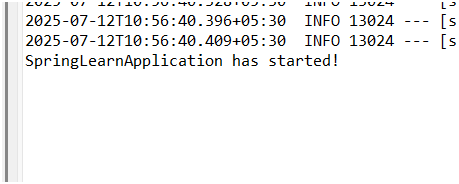
<artifactId>spring-boot-devtools</artifactId>

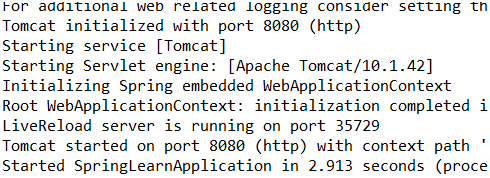
<scope>runtime</scope>

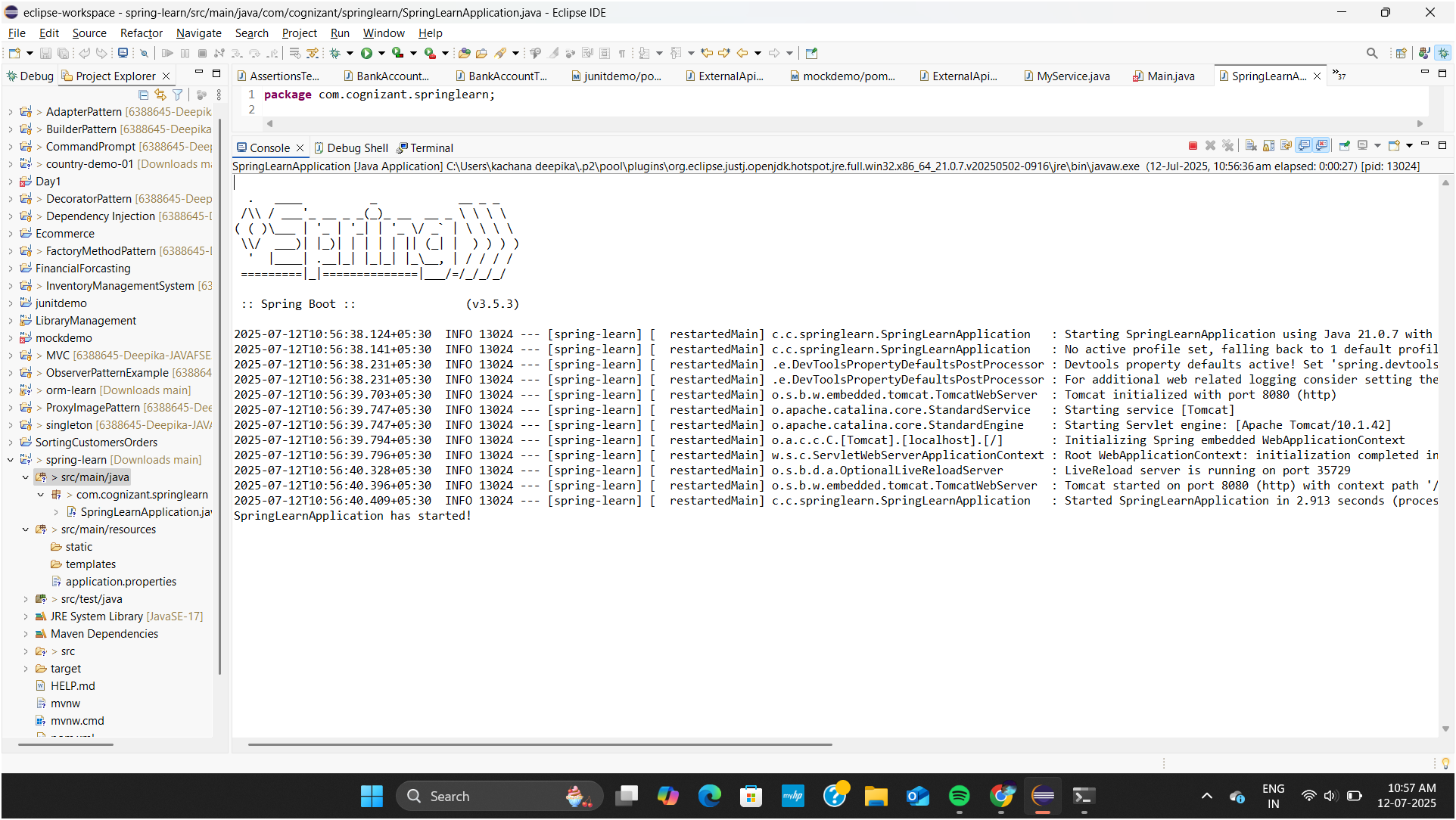
</dependency>

</dependencies>

**Output:**







**Question : 2**

**Hands on 4**

**Spring Core – Load Country from Spring Configuration XML**   
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country. 

|  |  |
| --- | --- |
| Code | Name |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.

**Project Structure :**

spring-learn

├── src

│ ├── main

│ │ ├── java

│ │ │ └── com.cognizant.springlearn

│ │ │ ├── Country.java

│ │ │ └── SpringLearnApplication.java

│ │ └── resources

│ │ └── country.xml

**Step 1: Create Country class**

**Path:** src/main/java/com/cognizant/springlearn/Country.java

package com.cognizant.springlearn;

public class Country {

private String code;

private String name;

public Country() {

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

}

public String getCode() {

System.out.println("Getting code");

return code;

}

public void setCode(String code) {

System.out.println("Setting code");

this.code = code;

}

public String getName() {

System.out.println("Getting name");

return name;

}

public void setName(String name) {

System.out.println("Setting name");

this.name = name;

}

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**Step 2: Create country.xml**

**Path:** src/main/resources/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">

<!-- Individual Country beans -->

<bean id="country1" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="country2" class="com.cognizant.springlearn.Country">

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

<property name="name" value="United States"/>

</bean>

<bean id="country3" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="country4" class="com.cognizant.springlearn.Country">

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

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

</bean>

<!-- List of countries -->

<bean id="countryList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="country1"/>

<ref bean="country2"/>

<ref bean="country3"/>

<ref bean="country4"/>

</list>

</constructor-arg>

</bean>

</beans>

**Step 3: Create SpringLearnApplication class**

package com.cognizant.springlearn;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class SpringLearnApplication {

public static void main(String[] args) {

System.out.println("Main method started...");

displayCountries();

}

public static void displayCountries() {

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

@SuppressWarnings("unchecked")

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

for (Country c : countryList) {

System.out.println(c);

}

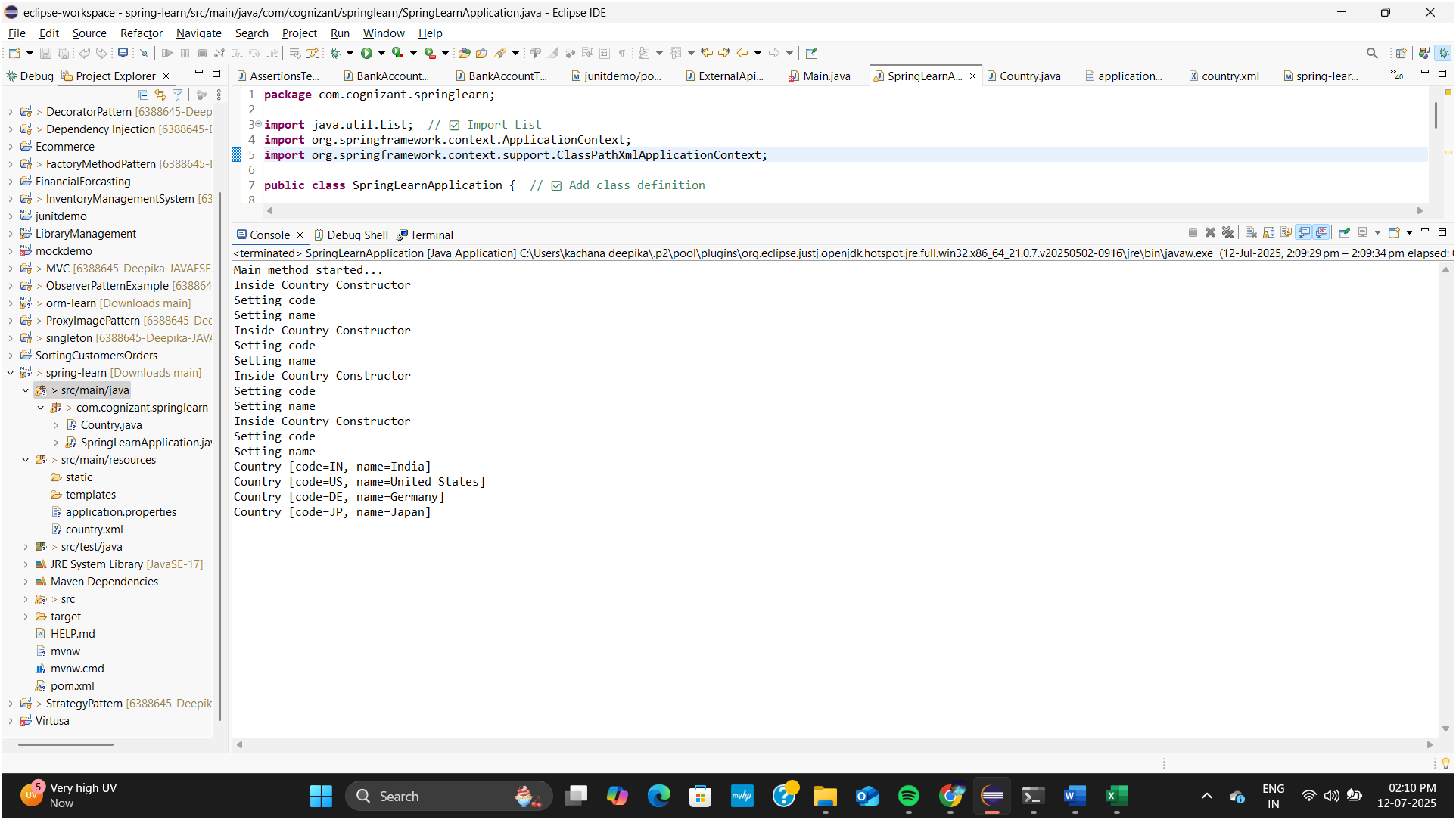
}

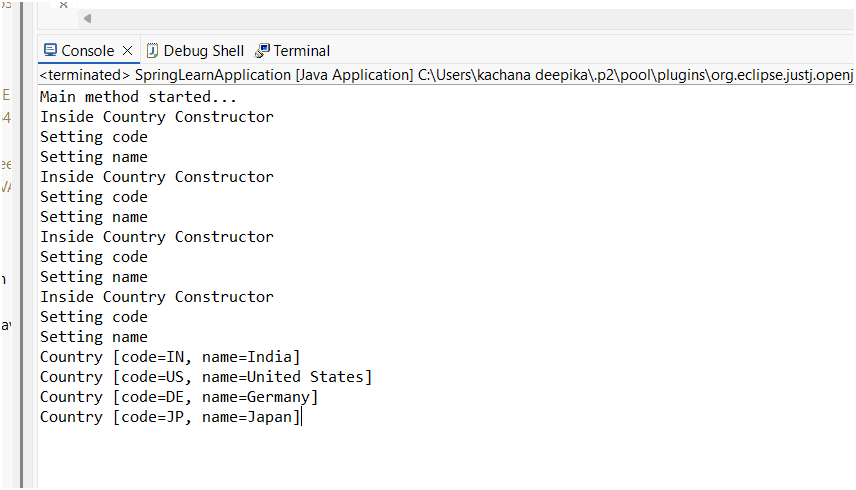
}

**Step 4: Run the Application**

* Right-click SpringLearnApplication.java → Run As → Java Application

**OutPut:**





**Question : 03**

**Hello World RESTful Web Service - Spring Boot**

**Project Overview**

* **Project Name:** spring-learn
* **Module:** Hello World RESTful API
* **Technology Stack:** Java, Spring Boot, Spring Web, Maven
* **IDE Used:** Eclipse
* **Port Used:** 8083

**Objective**

To create a simple RESTful service using Spring Boot that returns a hardcoded message "Hello World!!" at the /hello endpoint via HTTP GET request.

**REST Endpoint Details**

* **Method:** GET
* **URL:** /hello
* **Response:** Hello World!!
* **Controller Class:** com.cognizant.springlearn.controller.HelloController
* **Method Signature:** public String sayHello()

**Implementation Steps**

**Step 1: Add Spring Web Dependency in pom.xml**

<dependency>

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

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

</dependency>

**Step 2: Create 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.info("START: sayHello()");

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

return "Hello World!!";

}

}

**Step 3: Run the Application**

* Right-click SpringLearnApplication.java
* Run As → Java Application
* Ensure the application starts on port 8083

**Testing the API**

**A. Using Browser**

* URL: http://localhost:8083/hello
* Expected Output: Hello World!!

**B. Using Postman**

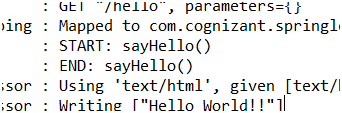
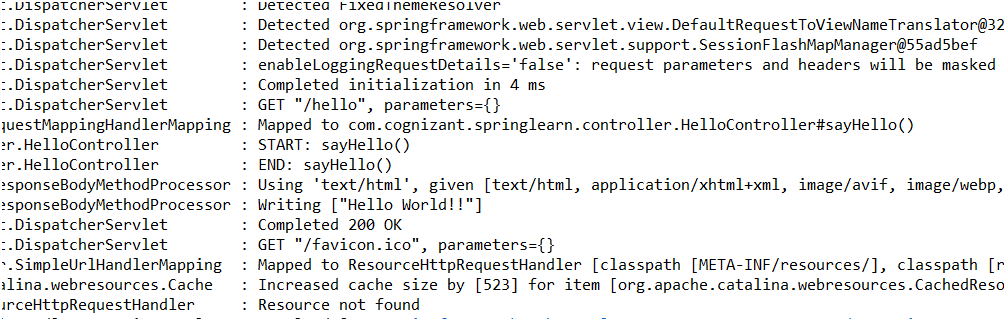
* Method: GET
* URL: http://localhost:8083/hello
* Output: Hello World!!
* Inspect headers tab to verify Content-Type: text/plain

**Sample Console Logs (Eclipse)**

INFO --- HelloController : START: sayHello()

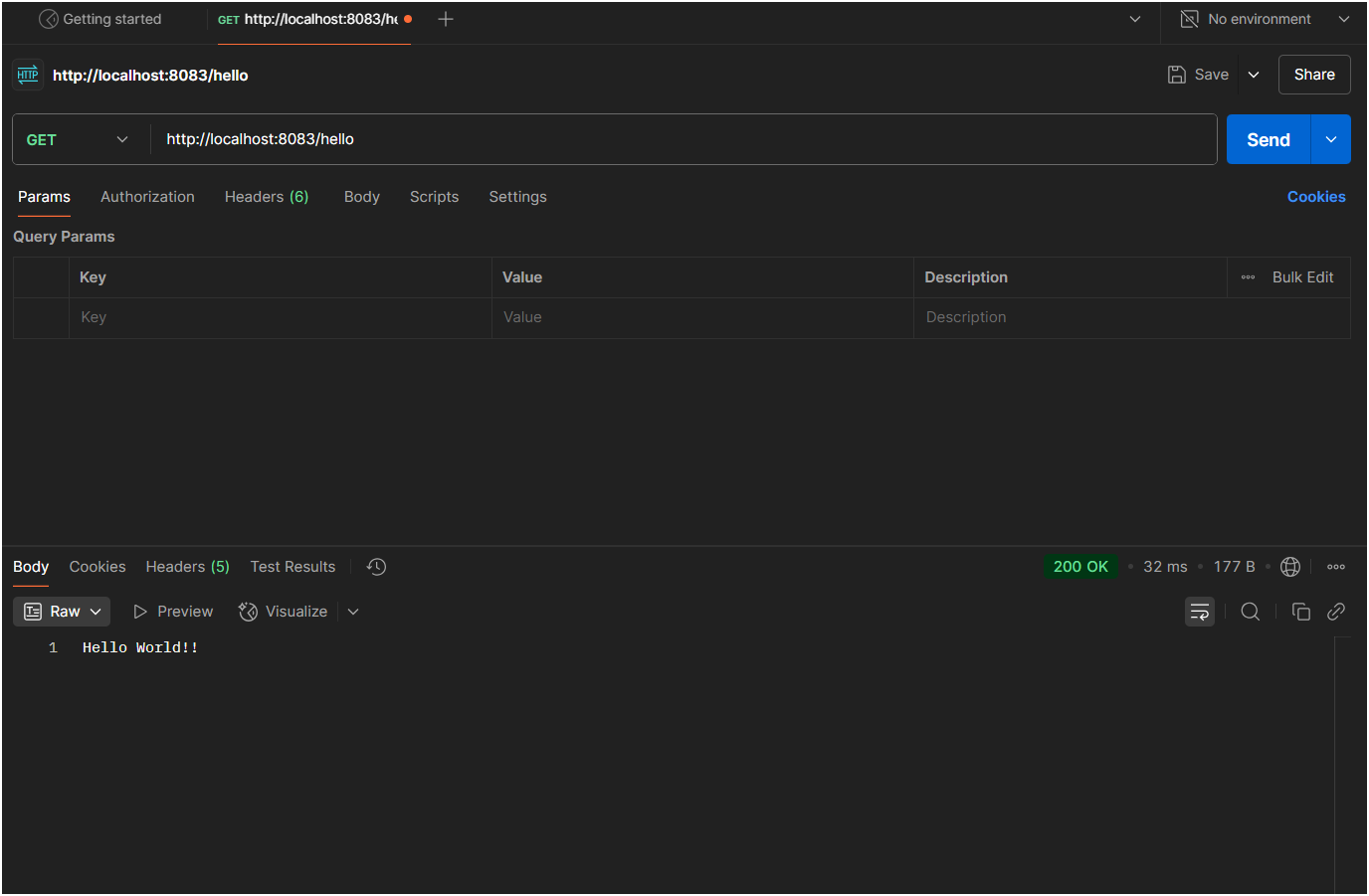
INFO --- HelloController : END: sayHello()

**Output:**

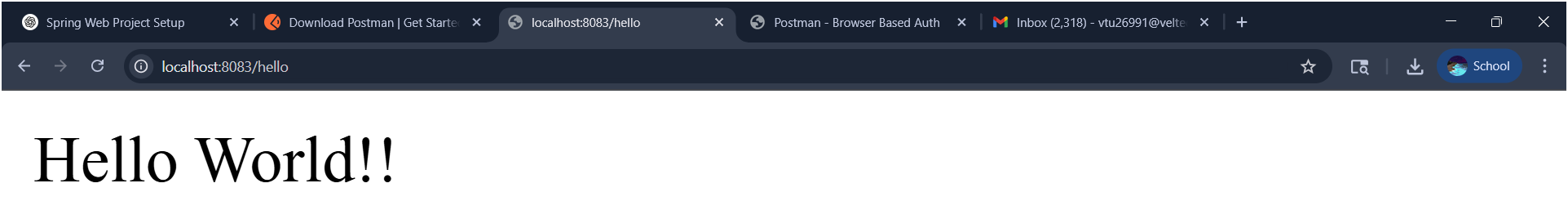


**Testing :**

**Method: Using Postman:**



**Method: Using Browser**



**Question : 04**

**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.

**Project Overview**

* Project Name: spring-learn
* Modules: Hello World & Country RESTful APIs
* Technology Stack: Java, Spring Boot, Spring Web, Maven
* IDE Used: Eclipse
* **Port Used: 8083**

**Objective**

To create simple RESTful services using Spring Boot that return:

* A hardcoded message "Hello World!!" at the /hello endpoint
* Country details (India) at the /country endpoint, using bean loaded from spring.xml
* A complete list of multiple countries at the /countries endpoint

**Implementation Steps**

**Step 1: Add Spring Web Dependency in pom.xml**

<dependency>

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

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

</dependency>

**Step 2: Create 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.info("START: sayHello()");

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

return "Hello World!!";

}

}

**Step 3: Create CountryController.java**

package com.cognizant.springlearn.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.springlearn.Country;

import java.util.List;

@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("country1", Country.class);

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

return country;

}

@RequestMapping("/countries")

public List<Country> getAllCountries() {

LOGGER.info("START: getAllCountries()");

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

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

LOGGER.info("END: getAllCountries()");

return countryList;

}

}

**Step 4: Create Bean Configuration in 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="country1" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="country2" class="com.cognizant.springlearn.Country">

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

<property name="name" value="United States"/>

</bean>

<bean id="country3" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="country4" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="countryList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="country1"/>

<ref bean="country2"/>

<ref bean="country3"/>

<ref bean="country4"/>

</list>

</constructor-arg>

</bean>

</beans>

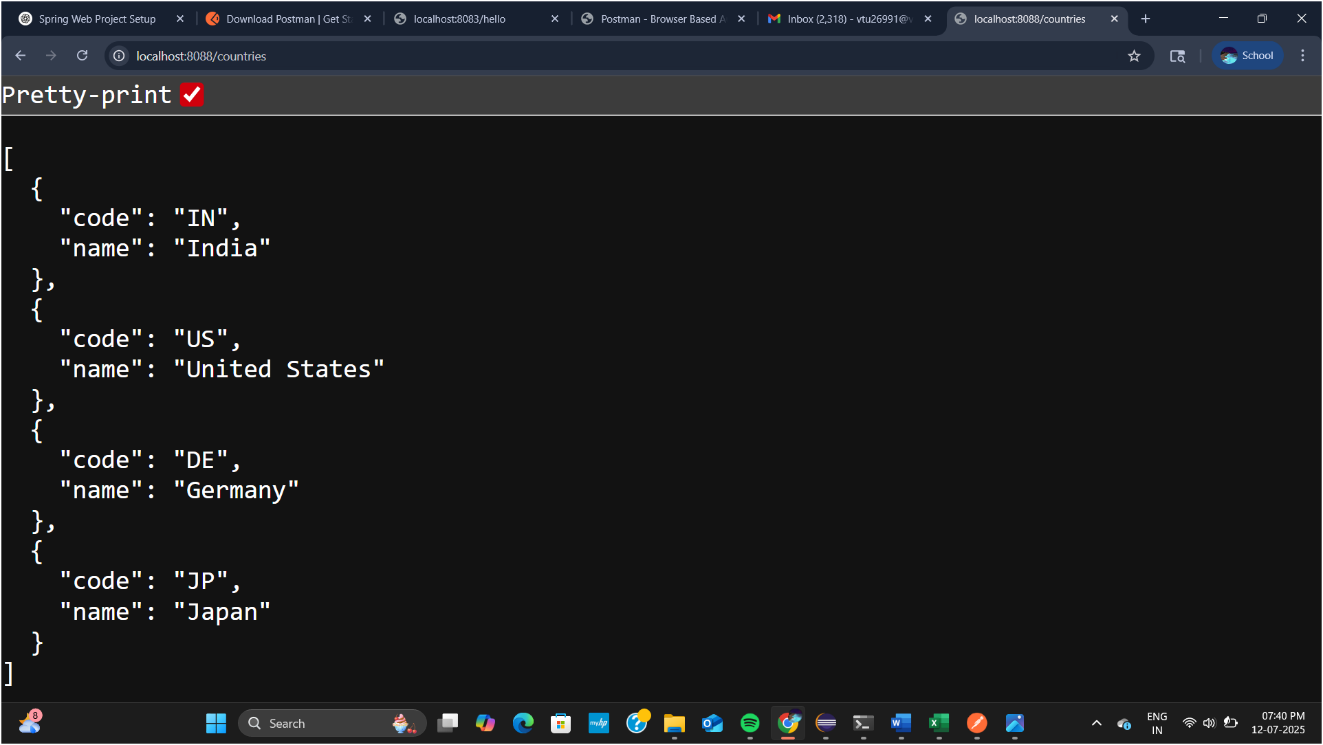
**Step 5: Run the Application**

* Right-click SpringLearnApplication.java
* Run As → Java Application
* Confirm that the application runs on port 8088

**Testing the API**

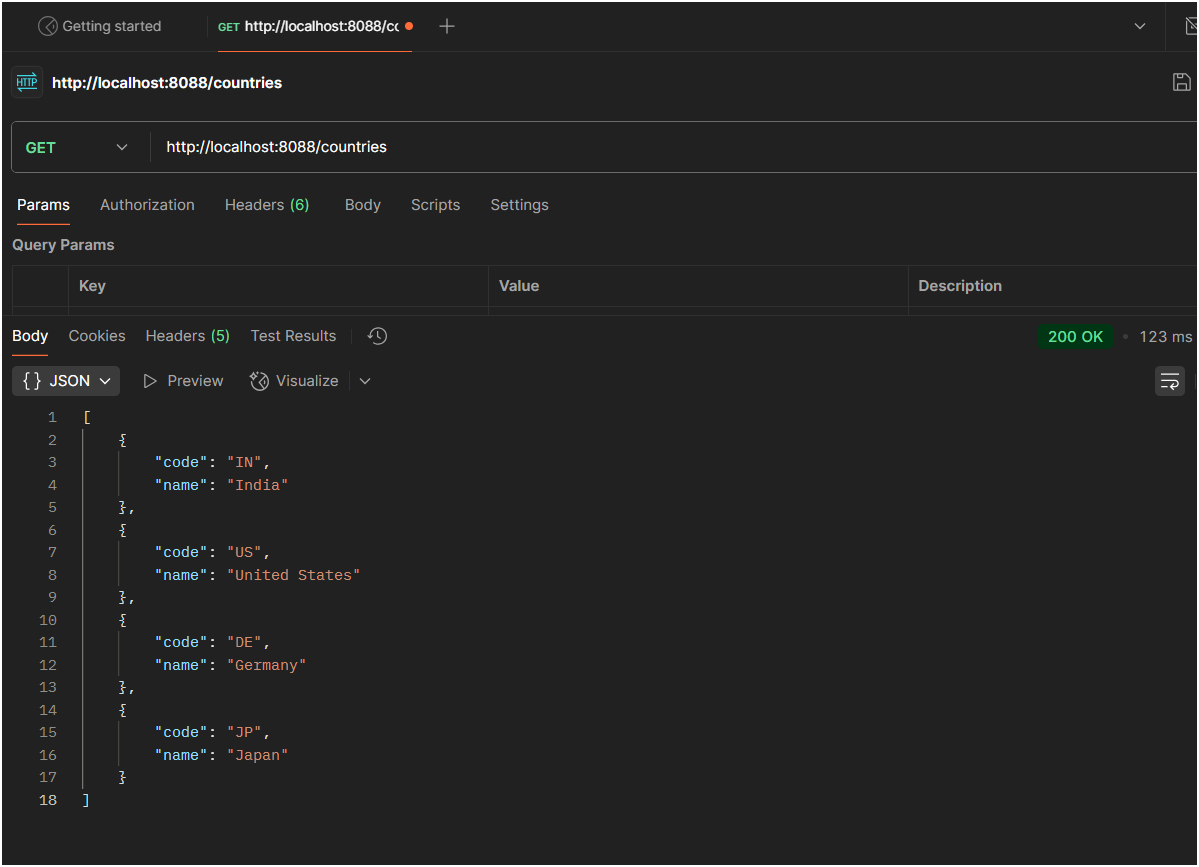
**/countries - Browser**

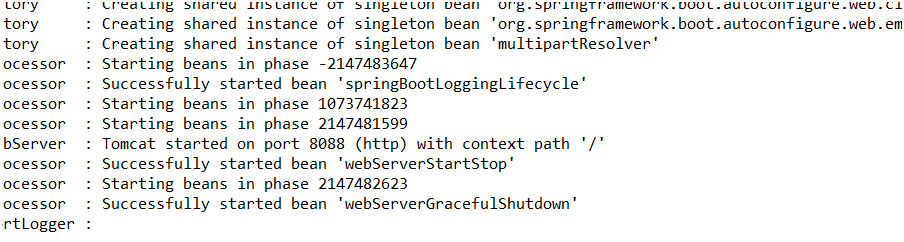
* URL: http://localhost:8088/countries
* Response:



**/countries - Postman**

* URL: http://localhost:8088/countries
* Response:







**Question : 05**

**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"

}

**Step-by-Step Implementation**

**Country.java:**

package com.cognizant.springlearn;

public class Country {

private String code;

private String name;

public Country() {}

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code.toUpperCase(); // Optional: normalize code

}

public String getName() {

return name;

}

public void setName(String name) {

this.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="country1" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="country2" class="com.cognizant.springlearn.Country">

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

<property name="name" value="United States"/>

</bean>

<bean id="country3" class="com.cognizant.springlearn.Country">

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

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

</bean>

<bean id="countryList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="country1"/>

<ref bean="country2"/>

<ref bean="country3"/>

</list>

</constructor-arg>

</bean>

</beans>

CountryService.java

package com.cognizant.springlearn.service;

import java.util.List;

import com.cognizant.springlearn.Country;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Qualifier;

import org.springframework.stereotype.Service;

@Service

public class CountryService {

@Autowired

@Qualifier("countryList") // load from XML

private List<Country> countryList;

public Country getCountry(String code) {

return countryList.stream()

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

.findFirst()

.orElse(null);

}

}

CountryController.java

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.Country;

import com.cognizant.springlearn.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class CountryController {

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

private final CountryService countryService;

public CountryController(CountryService countryService) {

this.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;

}

}

Enable XML loading in SpringLearnApplication.java:

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ImportResource;

@SpringBootApplication

@ImportResource("classpath:country.xml") // Load XML config

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

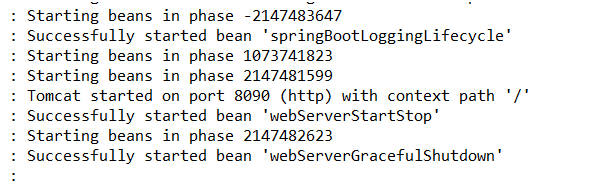
}

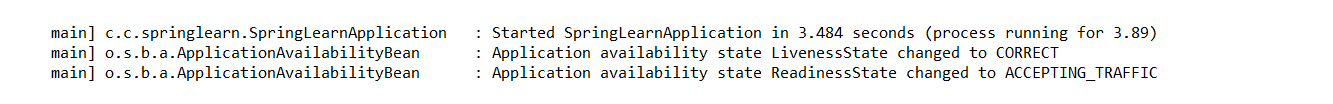
}

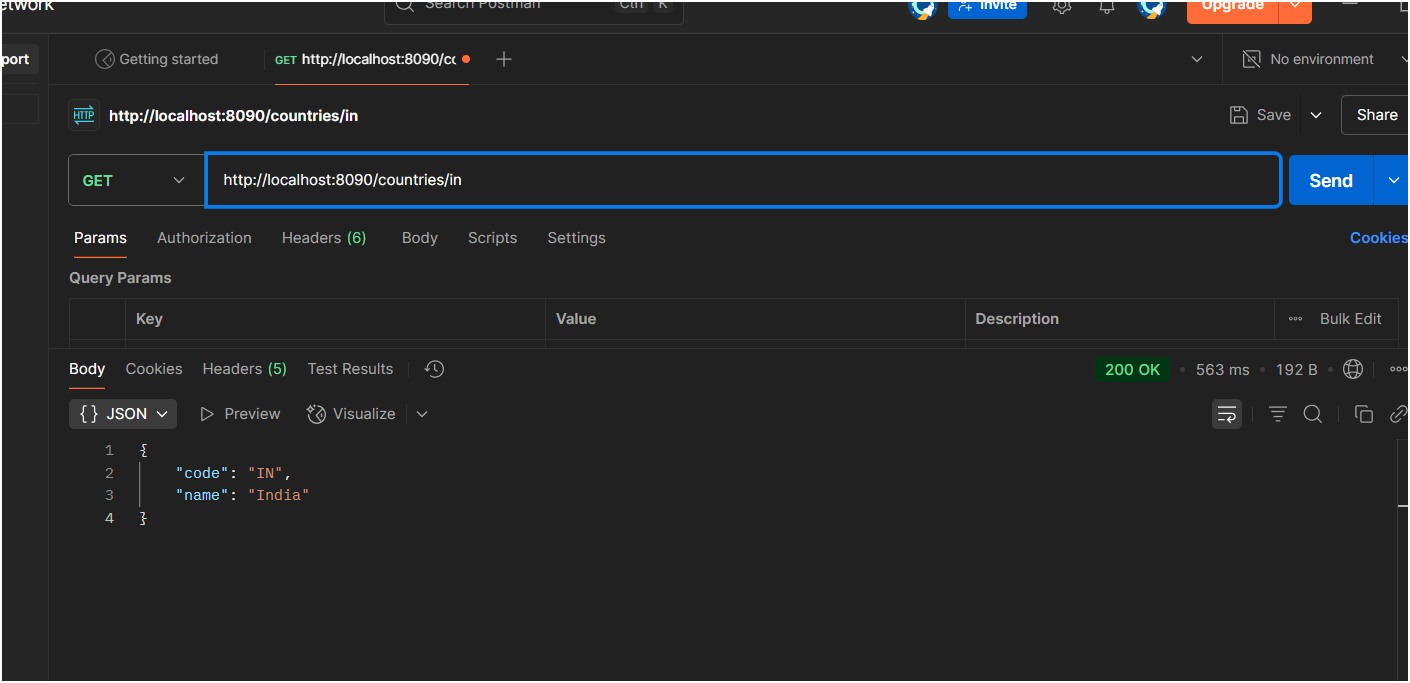
**Newly Added or Modified from Previous Applications:**

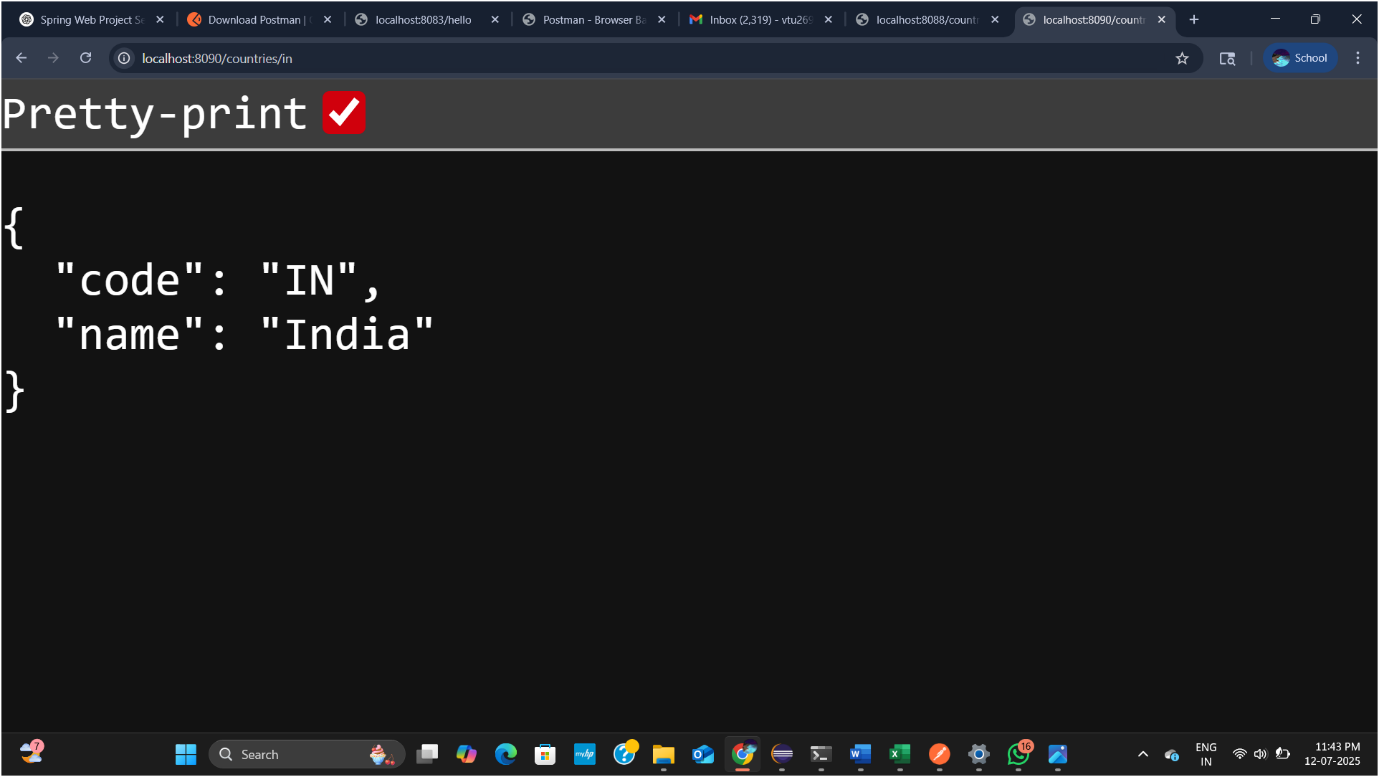
1. **New REST Endpoint:**
   * @GetMapping("/countries/{code}") added in CountryController.
2. **Service Class Added:**
   * CountryService.java created to handle logic for fetching country by code.
3. **Autowired country list:**
   * countryList is injected in CountryService using @Qualifier("countryList") from country.xml.
4. **Updated XML Configuration:**
   * country.xml defines multiple country beans and a countryList bean containing them.
5. **Enabled XML in Main Class:**
   * @ImportResource("classpath:country.xml") added to SpringLearnApplication.

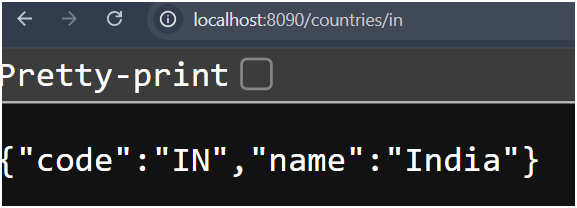
**Output:**











**Question 6 :**

**Create authentication service that returns JWT**   
  
As part of first step of JWT process, the user credentials needs to be sent to authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This service acts as an authentication provider. When a client sends a request to the /authenticate endpoint with valid Basic Authentication credentials (username and password), the service verifies these credentials and, if successful, generates and returns a JWT. This JWT can then be used by the client to access other protected resources in a stateless manner.

**pom.xml (Maven Dependencies)**

XML

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> </parent>

<groupId>com.example</groupId>

<artifactId>jwt-auth</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>jwt-auth</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<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>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**application.properties**

spring.application.name=jwt-auth

server.port=8067

spring.security.user.name=

spring.security.user.password=

**SpringLearnJwtApplication.java (Main Application Class)**

package com.cognizant.springlearnjwt;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnJwtApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnJwtApplication.class, args);

}

}

**AuthResponse.java (Response Model)**

package com.cognizant.springlearn.model;

public class AuthResponse {

private String token;

public AuthResponse(String token) {

this.token = token;

}

public String getToken() {

return token;

}

public void setToken(String token) {

this.token = token;

}

}

**JwtUtil.java (JWT Utility Service)**

package com.cognizant.springlearn.service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Service;

import java.util.Date;

@Service

public class JwtUtil {

private final String secret = "mySecretKey"; // TODO: Use a stronger key and load from environment variable in production!

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

.setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60)) // 1 hour expiration

.signWith(SignatureAlgorithm.HS256, secret)

.compact();

}

}

**SecurityConfig.java (Spring Security Configuration)**

package com.cognizant.springlearn.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.Customizer;

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

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

public class SecurityConfig {

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http.csrf(csrf -> csrf.disable()) // Disable CSRF for API

.authorizeHttpRequests(auth -> auth

.requestMatchers("/authenticate").permitAll() // Allow /authenticate without authentication

.anyRequest().authenticated() // All other requests require authentication

)

.httpBasic(Customizer.withDefaults()) // Enables curl -u and Postman Basic Auth

.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS)); // Crucial for JWT: No sessions

return http.build();

}

@Bean

public InMemoryUserDetailsManager userDetailsService() {

UserDetails user = User.withUsername("user")

.password("{noop}pwd") // {noop} means no password encoding

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

}

**AuthController.java (Authentication Endpoint)**

package com.cognizant.springlearn.controller;

import java.util.Base64;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpHeaders;

import org.springframework.http.ResponseEntity;

import org.springframework.util.StringUtils;

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

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

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

import com.cognizant.springlearn.model.AuthResponse;

import com.cognizant.springlearn.service.JwtUtil;

@RestController

public class AuthController {

@Autowired

private JwtUtil jwtUtil;

@GetMapping("/authenticate")

public ResponseEntity<AuthResponse> authenticate(@RequestHeader(HttpHeaders.AUTHORIZATION) String authHeader) {

if (StringUtils.hasText(authHeader) && authHeader.startsWith("Basic ")) {

String base64Credentials = authHeader.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];

// In a real application, you'd integrate with Spring Security's AuthenticationManager

// to authenticate against your UserDetailsService and PasswordEncoder.

// For this demonstration, we're using a simple hardcoded check.

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

String token = jwtUtil.generateToken(username);

return ResponseEntity.ok(new AuthResponse(token));

}

}

return ResponseEntity.status(401).build(); // Unauthorized

}

}

**Output:**

