**Week4\_REST - Country Web Service**

### Implementation Procedure(Feature-1)

#### Step 1: Data Model Creation (POJO)

A simple Java class was created to serve as the data structure for a country.

* Class: Country.java
* Location: com.cognizant.spring\_learn package
* Contents: The class includes private fields for code and name, along with public getters and setters for property access.

**package** com.cognizant.spring\_learn;

**public** **class** Country {

**private** String code;

**private** String name;

**public** Country() {

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

}

**public** String getCode() {

**return** code;

}

**public** **void** setCode(String code) {

**this**.code = code;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

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

}

}

#### Step 2: XML Bean Configuration

The Spring container was configured to manage an instance of the Country class.

* File: date-format.xml (located in src/main/resources)
* Configuration Detail: A <bean> tag was added to define the "India" object. Property injection (using the <property> tag) was utilized to set the code and name values via the setter methods.

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

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

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

#### Step 3: REST Controller Development

A controller was implemented to expose the configured country data as a web service endpoint.

* Class: CountryController.java
* Location: com.cognizant.spring\_learn.controller package
* Endpoint Definition: The getCountryIndia() method was annotated with @RequestMapping("/country") to handle incoming requests to that URL.
* Core Logic: The method's implementation creates a ClassPathXmlApplicationContext to load the Spring configuration, retrieves the Country bean with the ID "in", and returns the object.

**package** com.cognizant.spring\_learn.controller;

**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\_learn.Country;

@RestController

**public** **class** CountryController {

@RequestMapping("/country")

**public** Country getCountryIndia() {

// 1. Create the Spring container by loading the XML

ApplicationContext context = **new** ClassPathXmlApplicationContext("date-format.xml");

// 2. Retrieve the 'in' bean of type Country

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

// 3. Return the Country object

**return** country;

}

}

### Testing and Verification

#### 1. Execution

The SpringLearnApplication was launched as a Spring Boot application. The integrated Tomcat server started successfully without errors on the configured port 8083.

**Output**

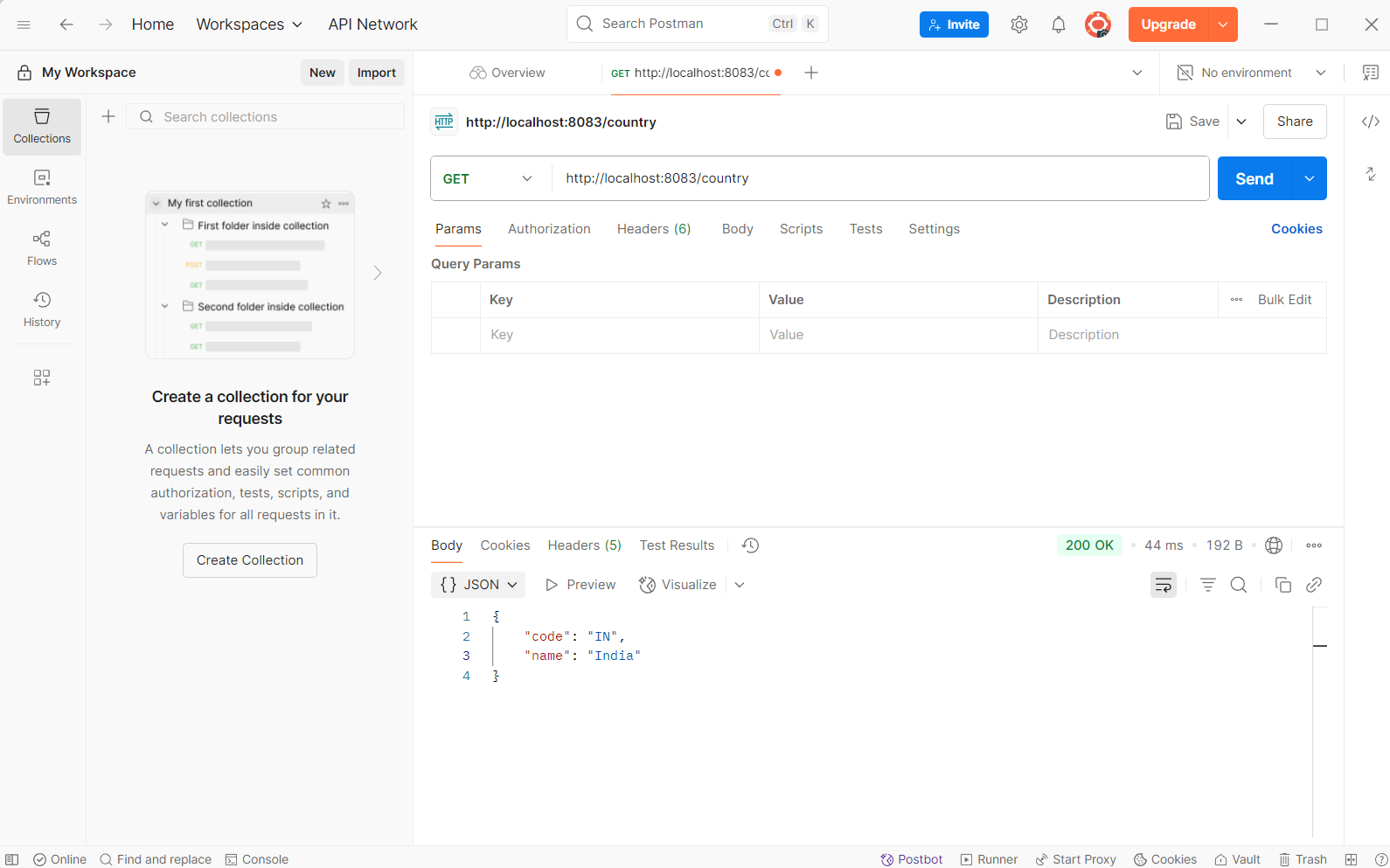


#### 2. API Test via Postman

The REST endpoint was tested to verify its functionality.

* Method: GET
* URL: http://localhost:8083/country
* Result: The service responded successfully with a 200 OK status and the expected JSON payload in the response body.

**Output**



#### Implementation Procedure(Feature-2)

1. XML Configuration (country.xml):

* A new, dedicated country.xml file was created.
* Individual beans were defined for India, USA, Germany, and Japan.
* A final bean (id="countryList") of type java.util.ArrayList was created. It was populated by adding references to the individual country beans using the <list> and <ref> tags.

1. Controller Logic (CountryController.java):

* A new method getAllCountries() was mapped to @GetMapping("/countries").
* This method loads the new country.xml context, retrieves the "countryList" bean, and returns the List<Country>.

### Testing and Verification

#### 1. Execution

The SpringLearnApplication was launched as a Spring Boot application. The integrated Tomcat server started successfully without errors on the configured port 8083.

**Output**



#### 2. API Test via Postman

The REST endpoint was tested to verify its functionality.

* Method: GET
* URL: http://localhost:8083/countries
* Response: A JSON array containing all four configured countries was received.

**Output**

