**Week4\_REST - Get country based on country code**

#### 1. Project Generation (Spring Initializr)

A base project was created at start.spring.io with the following metadata:

* Group: com.cognizant
* Artifact: spring-learn
* Dependencies: Spring Boot DevTools, Spring Web

#### 2. Command-Line Build

The project was built for the first time using the command line to validate the setup and download dependencies through the corporate proxy.

Command Used:

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

### Part 2: Spring Core - XML Bean Configuration

Objective: To understand Spring's core Inversion of Control (IoC) container by managing a Java object (SimpleDateFormat) through an XML configuration file.

#### 1. XML Configuration (date-format.xml)

A configuration file was created in src/main/resources to define the bean.

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

<bean id="dateFormat" class="java.text.SimpleDateFormat">

<constructor-arg value="dd/MM/yyyy" />

</bean></beans>

#### 2. Application Logic (SpringLearnApplication.java)

The main application class was modified to load the XML context and retrieve the bean, demonstrating dependency retrieval.

public static void displayDate() {

// Load the Spring container from the XML file

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

// Retrieve the pre-configured bean

SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);

Date date = format.parse("31/12/2018");

System.out.println(date);

}

### Part 3: Building RESTful API Endpoints

Objective: To create a series of REST endpoints, starting with a simple "Hello World" and progressing to returning structured JSON data from Java objects.

#### 3.1. "Hello World" Service (/hello)

* Configuration (application.properties): The server port was set.

server.port=8083

* Controller (HelloController.java): A simple controller was created to return a hardcoded string.

@RestControllerpublic class HelloController {

@GetMapping("/hello")

public String sayHello() {

return "Hello World!!";

}

}

#### 3.2. Get Single Country Service (/country)

* Data Model (Country.java): A POJO was created to hold country data (code, name).
* XML Modification (date-format.xml): The file was modified to add a new bean definition for India using property injection.

<!-- ... existing dateFormat bean ... --><bean id="in" class="com.cognizant.spring\_learn.Country">

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

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

* Controller (CountryController.java): A new controller was created to load the context and return the Country object, which Spring Boot automatically converted to JSON.

#### 3.3. Get All Countries Service (/countries)

* New XML Configuration (country.xml): A dedicated XML file was created to define a list of countries.

<beans ...>

<!-- Individual country beans for IN, US, DE, JP -->

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

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

<!-- ... -->

<!-- A bean that is a List of other beans -->

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

<constructor-arg>

<list>

<ref bean="in" />

<ref bean="us" />

<ref bean="de" />

<ref bean="jp" />

</list>

</constructor-arg>

</bean></beans>

* Controller Modification (`CountryController.java`):A new method was added to load `country.xml` and return the `List<Country>`, which Spring Boot automatically converted to a JSON array.

### Part 4: Refactoring to a Professional Layered Architecture

Objective: To refactor the application by introducing a service layer for business logic and implementing robust exception handling for invalid API requests.

#### 4.1. Introducing the Service Layer

* Service Class (CountryService.java): A new class was created and annotated with @Service. Business logic (loading country data from XML) was moved here from the controller.
* Controller Refactoring (CountryController.java): The controller was modified to delegate calls to the service. Dependency Injection was used to provide the service instance to the controller.

@RestControllerpublic class CountryController {

// Spring injects the service bean automatically

@Autowired

private CountryService countryService;

@GetMapping("/countries")

public List<Country> getAllCountries() {

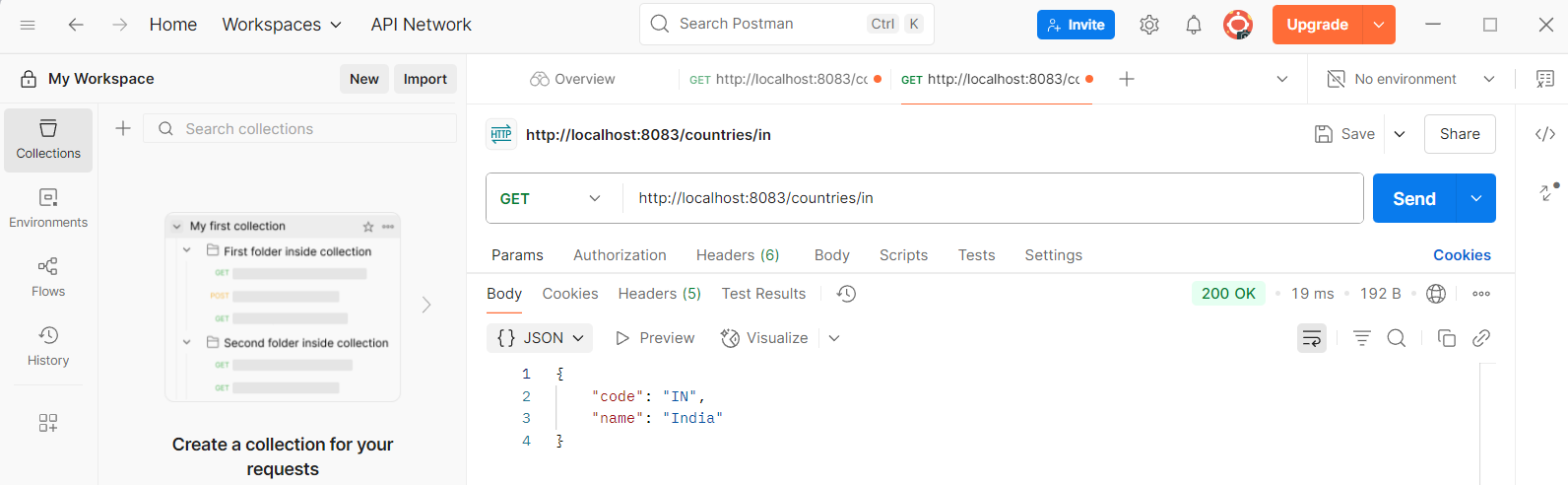
// Controller's only job is to delegate

return countryService.getAllCountries();

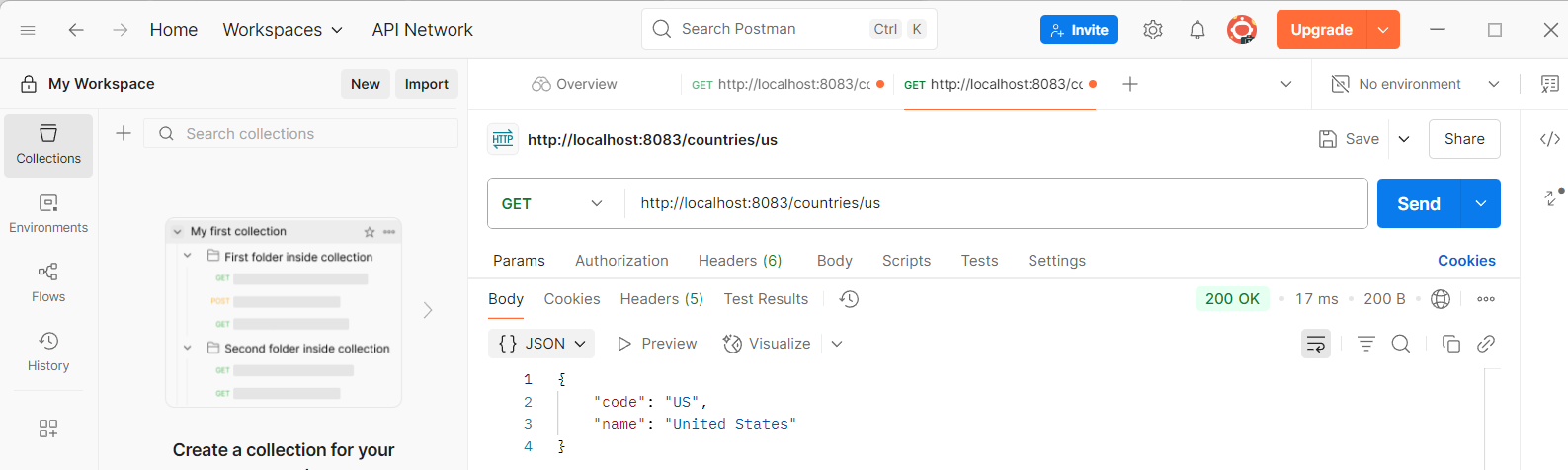
}

}

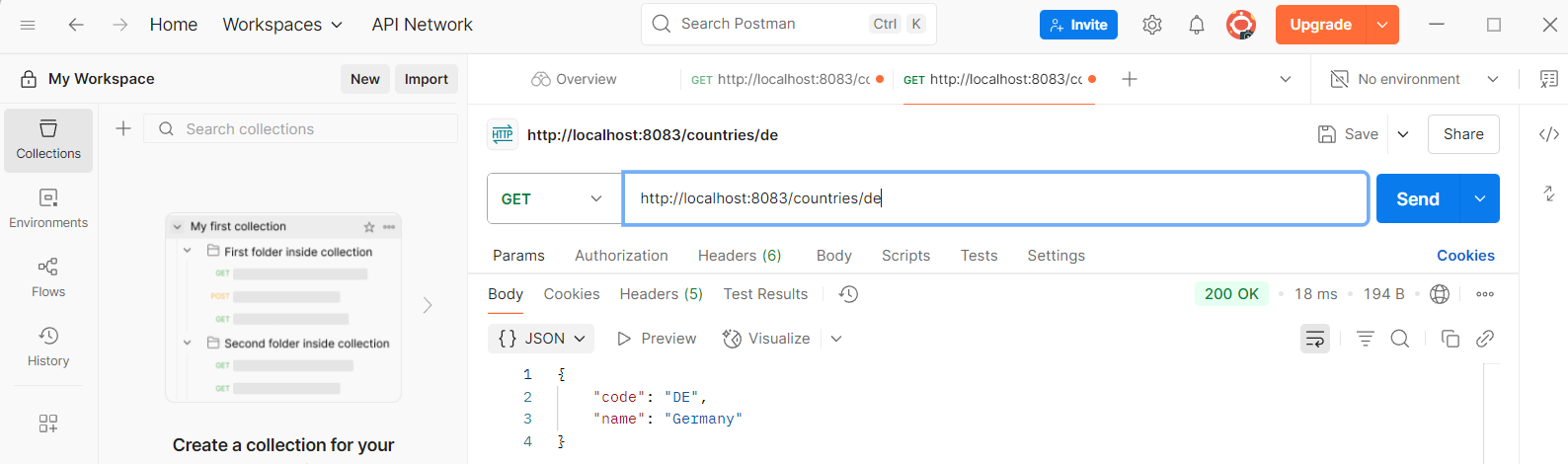
**Output(in) - http://localhost:8083/countries/us**



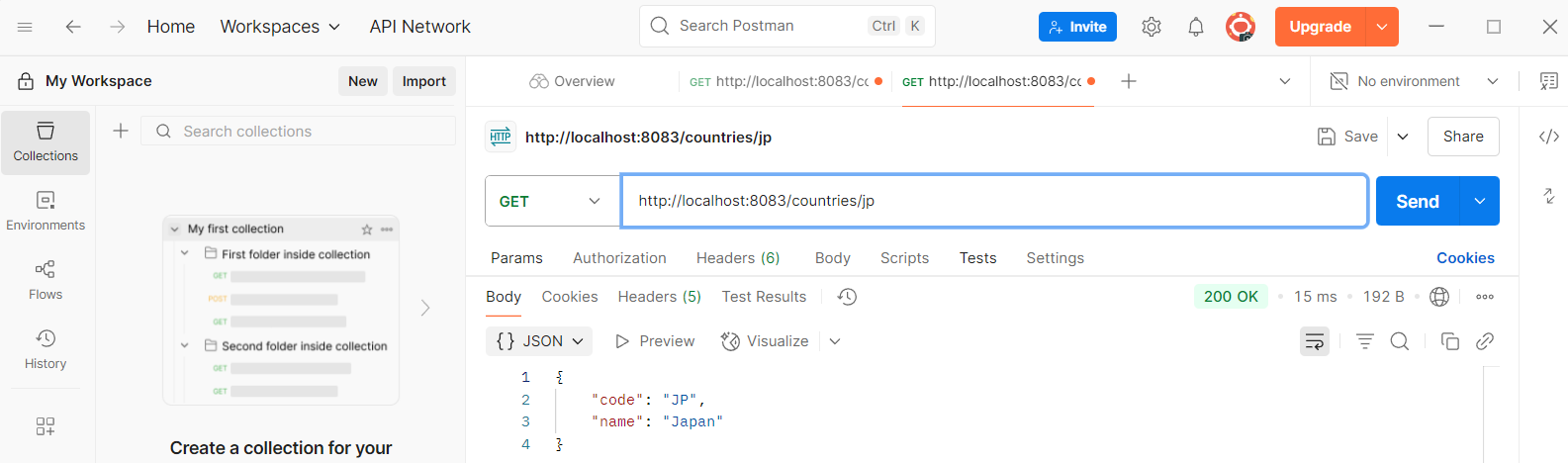
**Output(us) - http://localhost:8083/countries/us**



**Output(us) - http://localhost:8083/countries/de**



**Output(us) - http://localhost:8083/countries/jp**



#### 4.2. Get Country by Code with Exception Handling

* New Controller Endpoint: A method was added to handle dynamic path variables.

// In CountryController.java@GetMapping("/countries/{code}")public Country getCountry(@PathVariable String code) throws CountryNotFoundException {

return countryService.getCountry(code);

}

* Custom Exception (CountryNotFoundException.java): A custom exception was created. The @ResponseStatus annotation tells Spring to automatically return an HTTP 404 error when this exception is thrown.

@ResponseStatus(value = HttpStatus.NOT\_FOUND, reason = "Country not found")public class CountryNotFoundException extends RuntimeException { }

* Service Logic Modification (CountryService.java): The logic to find a country was updated. Instead of returning null, it now throws the custom exception if a country is not found.

// In CountryService.javapublic Country getCountry(String code) {

return getAllCountries().stream()

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

.findFirst()

// Throws the exception if the stream is empty

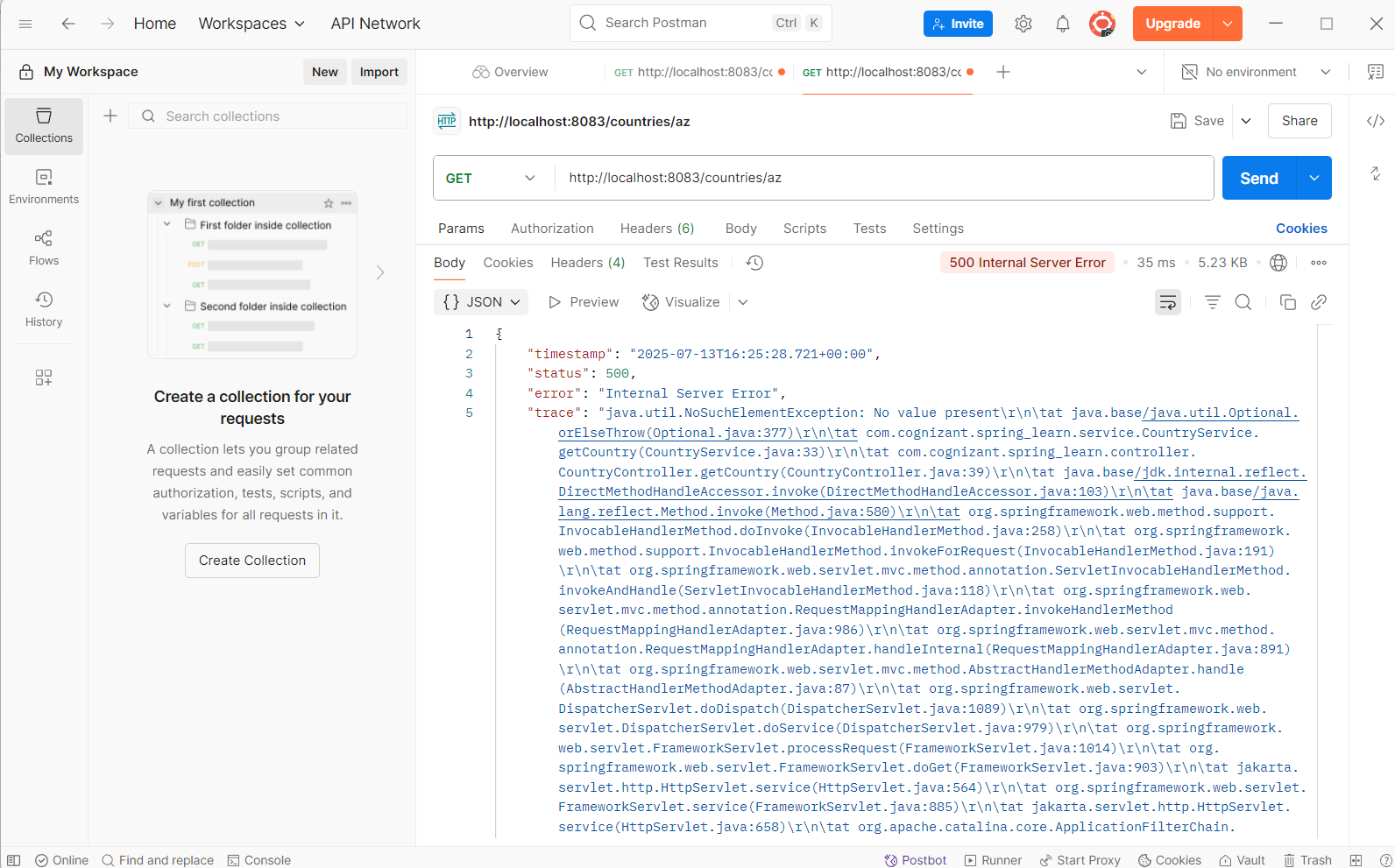
.orElseThrow(() -> new CountryNotFoundException());

}

#### 4.3. Final Testing and Verification

* The new endpoint and its exception handling were tested using Postman and the curl command line utility.

**Output(Postman)**



Test Command for Failure Case:

curl -i http://localhost:8083/countries/az

* Result: The command returned an HTTP/1.1 404 status code and a JSON error body, confirming the exception handling was working correctly.

**Output(Gitbash)**

