**2.SPRING-REST-HANDSON**

**HTTP Request Response**   
  
To get a granular level of details about HTTP Request and Response, follow the steps below:

* Open the link https://tools.ietf.org/html/rfc7230 in browser. This document contains the standard definition for HTTP request response.
* Refer sample HTTP request and response in page number 7. This is the actual bytes of data that is transferred between the browser and server.
* Specific details about the request and response:
  + Request

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

Accept-Language: en, mi

* Line 1 contains:
  + Method type - GET
  + Resource - /hello.txt
  + HTTP Version - HTTP/1.1
* Line 2 contains the details about the client
* Line 3 contains the server that will respond to this request
* The URL given in the browser is broken into Resource and Host in the HTTP Request

* Response

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

Hello World! My payload includes a trailing CRLF.

* Line 1
  + HTTP Version - HTTP/1.1
  + Response Status - 200 (this is means the request is responded successfully)
  + Response Message - Contains the response message
* Line 2 - Date of request
* Line 9 - Type of content returned. There is a list of predefined Content-Types. Based on Content-Type browser decides how the conent has to be visually displayed. Few examples below:
  + text/plain - Text content
  + text/html - HTML Document
  + application/json - JSON content
  + image/png - Image content of type PNG
* Last line contains the content of the resource.
  + ​​​​​​​In case of text/html, this will contain the HTML tags
  + In case of application/json, this will contain the JSON response
  + In case of image/png, this will contain the bytes to render the image

* To view the request and response details in browser, follow the steps below:
  + Open Chrome Browser
  + Press F12 to open the Developer Tools
  + Go to 'Network' table in Developer Tools
  + Open google search website in this browser window
  + Click on the first link available in the 'Network' tab
  + A new window will open in the right hands side. Observe the following details:
    - It will contain 3 sections. The data displayed will be similar to the HTTP request, response given above.
      * General
      * Response Headers
      * Request Headers

**HTTP Request:**

1. **Line 1:**
   * **Method Type:** GET — A request method used to retrieve data from the specified resource.
   * **Resource:** /hello.txt — The file or resource being requested.
   * **HTTP Version:** HTTP/1.1 — The version of the HTTP protocol used for the request.
2. **Line 2:**
   * **Client Details:**
     + User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3 — Information about the client making the request. In this case, it’s the curl tool with details about the libraries used.
3. **Line 3:**
   * **Server:** Host: www.example.com — The server to which the request is directed.
4. **Line 4:**
   * **Accept-Language:** en, mi — The languages the client prefers to receive the response in (English and Maori).

**HTTP Response:**

1. **Line 1:**
   * **HTTP Version:** HTTP/1.1 — The version of the HTTP protocol used for the response.
   * **Response Status Code:** 200 OK — Indicates that the request was successfully processed.
   * **Response Message:** OK — A standard message for successful HTTP requests.
2. **Line 2:**
   * **Date:** Mon, 27 Jul 2009 12:28:53 GMT — The date and time the response was generated.
3. **Line 3:**
   * **Server:** Apache — The server software handling the request.
4. **Line 4:**
   * **Last-Modified:** Wed, 22 Jul 2009 19:15:56 GMT — The last modification date of the resource.
5. **Line 5:**
   * **ETag:** "34aa387-d-1568eb00" — A unique identifier for the resource, which helps in cache validation.
6. **Line 6:**
   * **Accept-Ranges:** bytes — Indicates that the server accepts requests to transfer a portion of the resource.
7. **Line 7:**
   * **Content-Length:** 51 — The length of the response body in bytes.
8. **Line 8:**
   * **Vary:** Accept-Encoding — The response varies based on the encoding used in the request (e.g., compression).
9. **Line 9:**
   * **Content-Type:** text/plain — Specifies the format of the content. In this case, it’s plain text.
10. **Last Line:**

* **Content:** Hello World! My payload includes a trailing CRLF. — The actual data being returned in the response, typically the content of the requested resource.

**How to View HTTP Request/Response in Browser (Using Chrome):**

1. **Open Chrome Browser.**
2. **Press F12** to open Developer Tools.
3. **Go to the 'Network' tab** in Developer Tools.
4. **Visit a website,** such as Google Search.
5. **Click on a request** in the 'Network' tab, typically the first one after the page loads.
6. In the right pane:  
   * **General Section** — Contains basic details about the request/response like status, method, and URL.
   * **Response Headers Section** — Lists headers sent by the server, including Content-Type, Date, and more.
   * **Request Headers Section** — Lists headers sent by the client, including User-Agent, Host, Accept-Language, etc.

**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

:

### **1. Add necessary dependencies in pom.xml (if not already present):**

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

### **2. Create the Controller (HelloController.java):**

package com.cognizant.springlearn.controller;

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

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

@RestController

public class HelloController {

@GetMapping("/hello")

public String sayHello() {

// Logging the start of the method

System.out.println("start: sayHello() method");

// Hard-coded response message

String response = "Hello World!!";

// Logging the end of the method

System.out.println("end: sayHello() method");

return response;

}

}

### **3. Run the Spring Boot Application:**

Ensure your main Spring Boot application is set up to run the application:

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

### **4. Test the Web Service:**

* **URL:** http://localhost:8083/hello
* **Method:** GET

### **5. Observing HTTP Headers in the Browser (Chrome Developer Tools):**

1. Open **Chrome** and press **F12** to open Developer Tools.
2. Go to the **Network** tab.
3. Visit the URL http://localhost:8083/hello.
4. In the **Network** tab, find the request with the URL /hello. Click on it to open the request details.
5. Look under the **Headers** section to observe the HTTP headers sent and received. Example headers might include:  
   * **Request Headers**:  
     + Host: localhost:8083
     + User-Agent: Mozilla/5.0 ...
     + Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,...
   * **Response Headers**:  
     + HTTP/1.1 200 OK
     + Content-Type: text/plain;charset=UTF-8
     + Date: [timestamp]
     + Content-Length: 14

### **6. Observing HTTP Headers in Postman:**

1. Open **Postman** and send a **GET** request to http://localhost:8083/hello.
2. After receiving the response, click on the **Headers** tab in Postman to view the HTTP header details.
3. Example headers might include:  
   * **Request Headers**:  
     + Host: localhost:8083
     + User-Agent: PostmanRuntime/...
     + Accept: \*/\*
   * **Response Headers**:  
     + HTTP/1.1 200 OK
     + Content-Type: text/plain; charset=UTF-8
     + Content-Length: 14
     + Date: [timestamp]

### **Expected Response:**

When you visit the URL in either Chrome or Postman, you will see the following response:

nginx

Hello World!!

This will be displayed as the text content returned by the REST API.

### **1. Create the Country Bean (India)**

First, create a Java class to represent the country details.

package com.cognizant.springlearn.model;

public class Country {

private String code;

private String name;

// Getters and Setters

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;

}

}

### **2. Spring XML Configuration to Define the Bean**

Next, configure the **Country** bean in the Spring XML file (applicationContext.xml):

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

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

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

</bean>

This XML configuration defines a Country bean with properties code as "IN" and name as "India".

### **3. Create the Controller (CountryController.java)**

The controller will handle the /country endpoint and return the India details as a JSON response.

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country;

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

import org.springframework.context.ApplicationContext;

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

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

@RestController

public class CountryController {

@Autowired

private ApplicationContext context;

@RequestMapping("/country")

public Country getCountryIndia() {

// Load the India bean from the Spring application context

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

}

}

### **4. Spring Boot Configuration (SpringLearnApplication.java)**

Make sure your Spring Boot application class is set up to run the Spring Boot application.

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

### **5. Test the REST API**

* **URL:** http://localhost:8083/country
* **Method:** GET

### **6. Expected JSON Response**

When you visit the URL http://localhost:8083/country, the response will be:

{

"code": "IN",

"name": "India"

}

### **SME Explanation:**

1. **What happens in the controller method?**
   * The controller method getCountryIndia() is invoked when the user makes a GET request to /country.
   * The Spring ApplicationContext is used to load the Country bean configured in the Spring XML file.
   * The Country bean with code "IN" and name "India" is returned by the controller method.
   * The Spring Framework automatically converts the Java object into JSON using Jackson (which is included by default in Spring Boot).
2. **How the bean is converted into JSON response?**
   * The @RestController annotation is a specialized version of @Controller that combines @ResponseBody, indicating that the return value of the methods should be written directly to the HTTP response body.
   * Spring Boot automatically uses Jackson to serialize the Country Java object to a JSON response. Jackson converts the fields of the Country class (code and name) into corresponding JSON properties.
   * As a result, when the Country object is returned, it is automatically converted to JSON, and the Content-Type of the response is set to application/json.
3. **In the Network tab of Developer Tools, show the HTTP header details received:**
   * **Open Chrome**, press **F12** to open Developer Tools, and go to the **Network** tab.
   * Visit http://localhost:8083/country.
   * Click on the network request related to /country to see the headers.
   * **Request Headers**:  
     + Host: localhost:8083
     + User-Agent: Mozilla/5.0 ...
     + Accept: application/json
   * **Response Headers**:  
     + HTTP/1.1 200 OK
     + Content-Type: application/json;charset=UTF-8
     + Date: [timestamp]
     + Content-Length: [response length]
4. **In Postman, click on "Headers" tab to view the HTTP header details received:**
   * Open **Postman**, send a **GET** request to http://localhost:8083/country.
   * After receiving the response, click on the **Headers** tab.
   * **Response Headers**:  
     + HTTP/1.1 200 OK
     + Content-Type: application/json; charset=UTF-8
     + Content-Length: [response length]
     + Date: [timestamp]
   * The **Request Headers** in Postman might look like:  
     + Host: localhost:8083
     + User-Agent: PostmanRuntime/...
     + Accept: application/json

### **1. Create Country Bean:**

Create a Country model to represent the country details.

package com.cognizant.springlearn.model;

public class Country {

private String code;

private String name;

// Getters and Setters

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;

}

}

### **2. Spring XML Configuration for Country Beans:**

Create an XML file (country.xml) to define multiple country beans.

<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">

<!-- Defining Country Beans -->

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

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

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

</bean>

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

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

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

</bean>

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

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

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

</bean>

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

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

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

</bean>

</beans>

### **3. CountryService Class:**

Create the CountryService class that loads the list of countries and provides the method to get a country by code.

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

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

import org.springframework.context.ApplicationContext;

import org.springframework.stereotype.Service;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private ApplicationContext context;

public List<Country> getAllCountries() {

// Load all country beans from Spring context

return Arrays.asList(

(Country) context.getBean("countryIndia"),

(Country) context.getBean("countryUS"),

(Country) context.getBean("countryJapan"),

(Country) context.getBean("countryGermany")

);

}

public Country getCountry(String code) {

// Get all countries

List<Country> countries = getAllCountries();

// Case insensitive matching of country code

Optional<Country> country = countries.stream()

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

.findFirst();

return country.orElse(null);

}

}

### **4. CountryController Class:**

Create the CountryController class to handle the RESTful endpoints.

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

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

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

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

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

import java.util.List;

@RestController

public class CountryController {

@Autowired

private CountryService countryService;

// Endpoint to get all countries

@GetMapping("/countries")

public List<Country> getAllCountries() {

return countryService.getAllCountries();

}

// Endpoint to get country by country code

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

public Country getCountry(@PathVariable String code) {

return countryService.getCountry(code);

}

}

### **5. Spring Boot Application Configuration:**

Ensure the main Spring Boot application class is set up to run the Spring Boot application.

package com.cognizant.springlearn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringLearnApplication {

public static void main(String[] args) {

SpringApplication.run(SpringLearnApplication.class, args);

}

}

### **6. Test the REST API**

#### **a. Get All Countries:**

* **URL:** http://localhost:8083/countries
* **Method:** GET

**Sample Response**:

[

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

{ "code": "US", "name": "United States" },

{ "code": "JP", "name": "Japan" },

{ "code": "DE", "name": "Germany" }

]

#### **b. Get Country by Code:**

* **URL:** http://localhost:8083/countries/in
* **Method:** GET

**Sample Response**:

{

"code": "IN",

"name": "India"

}

### **SME Explanation:**

1. **Controller Method (getAllCountries() and getCountry()):**
   * The @GetMapping("/countries") endpoint invokes the getAllCountries() method in the controller, which retrieves the list of countries by calling the CountryService.getAllCountries() method. The list is returned as a JSON array.
   * The @GetMapping("/countries/{code}") endpoint invokes the getCountry() method. It receives the country code as a @PathVariable, and it calls the CountryService.getCountry(code) method to find the country by its code (case insensitive). The result is returned as a JSON object.
2. **How the Bean is Converted into JSON Response:**
   * Spring Boot uses **Jackson** (by default) to automatically serialize Java objects (like Country) into JSON format.
   * The @RestController annotation ensures that the return value of the methods is automatically written as the response body in JSON format, with the appropriate Content-Type: application/json header.
3. **Network Tab in Developer Tools (Chrome):**
   * Open **Developer Tools** (press F12) in **Chrome** and go to the **Network** tab.
   * Visit http://localhost:8083/countries and check the HTTP request and response headers.
   * **Request Headers** might include:  
     + Host: localhost:8083
     + User-Agent: Mozilla/5.0 ...
     + Accept: application/json
   * **Response Headers**:  
     + HTTP/1.1 200 OK
     + Content-Type: application/json;charset=UTF-8
     + Date: [timestamp]
     + Content-Length: [response length]
4. **In Postman:**
   * Send a **GET** request to http://localhost:8083/countries.
   * After receiving the response, click on the **Headers** tab to view the HTTP response headers.
   * **Response Headers** will include:  
     + HTTP/1.1 200 OK
     + Content-Type: application/json; charset=UTF-8
     + Content-Length: [response length]
     + Date: [timestamp]

### **Steps to Handle the Exceptional Scenario for Country Not Found:**

To handle the case when the country code provided in the URL does not exist, you will create a custom exception CountryNotFoundException and modify the CountryService and CountryController to throw and handle this exception.

### **1. Create the CountryNotFoundException Exception Class:**

The CountryNotFoundException class will be responsible for handling the error scenario when a country is not found by its country code.

package com.cognizant.springlearn.service.exception;

import org.springframework.http.HttpStatus;

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

@ResponseStatus(value = HttpStatus.NOT\_FOUND, reason = "Country not found")

public class CountryNotFoundException extends RuntimeException {

public CountryNotFoundException(String message) {

super(message);

}

}

* **@ResponseStatus Annotation:** This annotation ensures that the response status is set to 404 Not Found and the reason "Country not found" is sent in the response.
* The CountryNotFoundException extends RuntimeException to make it an unchecked exception.

### **2. Update CountryService to Throw CountryNotFoundException:**

The CountryService class needs to check if the country exists, and if not, it will throw the CountryNotFoundException.

package com.cognizant.springlearn.service;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

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

import org.springframework.context.ApplicationContext;

import org.springframework.stereotype.Service;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private ApplicationContext context;

public List<Country> getAllCountries() {

return Arrays.asList(

(Country) context.getBean("countryIndia"),

(Country) context.getBean("countryUS"),

(Country) context.getBean("countryJapan"),

(Country) context.getBean("countryGermany")

);

}

public Country getCountry(String code) {

List<Country> countries = getAllCountries();

// Case-insensitive country matching

Optional<Country> country = countries.stream()

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

.findFirst();

if (!country.isPresent()) {

// If country code not found, throw CountryNotFoundException

throw new CountryNotFoundException("Country with code " + code + " not found");

}

return country.get();

}

}

* If the country code is not found, the getCountry method throws the CountryNotFoundException, which will be handled by Spring's exception handling mechanism.

### **3. Update CountryController to Handle CountryNotFoundException:**

Modify the CountryController class to declare that the getCountry method throws the CountryNotFoundException.

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

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

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

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

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

import java.util.List;

@RestController

public class CountryController {

@Autowired

private CountryService countryService;

@GetMapping("/countries")

public List<Country> getAllCountries() {

return countryService.getAllCountries();

}

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

public Country getCountry(@PathVariable String code) throws CountryNotFoundException {

return countryService.getCountry(code);

}

}

* The throws CountryNotFoundException clause in the getCountry method signature indicates that this exception might be thrown and should be handled by Spring automatically.

### **4. Test the Service using Postman and Curl:**

#### **Postman Test:**

1. Open **Postman**.
2. Make a GET request to http://localhost:8083/country/az.
3. **Response:**
   * **Status:** 404 Not Found
   * **Body:**

{

"timestamp": "2019-10-02T03:27:54.521+0000",

"status": 404,

"error": "Not Found",

"message": "Country not found",

"path": "/country/az"

}

#### **Curl Command Test:**

1. Open **Git Bash** or your terminal.
2. Execute the following curl command to test the service:

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

**Expected Response:** HTTP/1.1 404 Not Found

Content-Type: application/json;charset=UTF-8

Date: Wed, 02 Oct 2019 03:27:54 GMT

Content-Length: 123

{

"timestamp": "2019-10-02T03:27:54.521+0000",

"status": 404,

"error": "Not Found",

"message": "Country not found",

"path": "/country/az"

}