**Objectives**

* Explain in detail about HTTP Request and Response
  + HTTP Request and Response, HTTP Request Format, HTTP Response Format, Request URL, Request Method, Content-Type, User-Agent
    - Ref - https://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol

* Explain the need and benefits of RESTful Web Services
  + REST stands for REpresentational State Transfer, lightweight, maintainable, scalable, underlying protocol is HTTP; composed of resources, verbs, header, body, response status code, client-server technology
    - Ref - https://www.chakray.com/advantages-of-rest-api/

* Demonstrate implementation of RESTful Web Service using GET method
  + @RestController, @GetMapping, invoking get request from browser, invoking get request from postman, bean transformation to JSON, get method with parameter, return array, @PathVariable
    - Dispatcher Servlet - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/web.html#mvc-servlet
    - Spring REST (Getting Started) - https://spring.io/guides/gs/rest-service/
    - Request Mapping - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/web.html#mvc-ann-requestmapping

* Demonstrate implementation of end to end testing of RESTful Web Service using MockMVC
  + @AutoConfigureMockMvc, MockMvc, @Test, get(), perform(), andExpect(), status().isOk(), jsonPath().exists, jsonPath().value(), status().isBadRequest(), status().reason test execution in Eclipse, test execution in command line using maven
    - Server Side Testing - https://docs.spring.io/spring/docs/5.1.9.RELEASE/spring-framework-reference/testing.html#spring-mvc-test-server

**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

**Solutions:**

**Request Example:**

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3

Host: www.example.com

Accept-Language: en

* Method: GET
* Resource: /hello.txt
* Protocol: HTTP/1.1

**Response Example:**

HTTP/1.1 200 OK

Date: <timestamp>

Server: Apache

Content-Type: text/plain

Content-Length: 51

Hello World! My payload includes a trailing CRLF.

**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

Solutions:

**HelloController.java**

@RestController

public class HelloController {

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

@GetMapping("/hello")

public String sayHello() {

LOGGER.info("Start");

LOGGER.info("End");

return "Hello World!!";

}

}

**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

Solutions:

**country.xml**

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

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

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

</bean>

**CountryController.java**

@RestController

@RequestMapping("/country")

public class CountryController {

@GetMapping

public Country getCountryIndia() {

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

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

}

}

**REST - Get all countries**   
  
Write a REST service that returns all the countries.  
  
**Controller**: com.cognizant.spring-learn.controller.CountryController  
**Method Annotation**: @GetMapping("/countries")  
**Method Name**: getAllCountries()  
**Method Implementation**: Load country list from country.xml and return  
  
**Sample Request**: http://localhost:8083/countries  
**Sample Response**:

[

  { "code": "IN", "name": "India"},

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

  { "code": "JP", "name": "Japan"},

  { "code": "DE", "name": "Germany"}

]

Solution:

**country.xml**

<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>

**CountryController.java**

@GetMapping("/countries")

public List<Country> getAllCountries() {

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

return context.getBean("countryList", List.class);

}

**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:

**CountryService.java**

public Country getCountry(String code) throws CountryNotFoundException {

List<Country> countries = getAllCountries();

return countries.stream()

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

.findFirst()

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

}

**CountryController.java**

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

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

return countryService.getCountry(code);

}

**Exception Class:**

@ResponseStatus(HttpStatus.NOT\_FOUND)

public class CountryNotFoundException extends Exception {

}

**REST - Get country exceptional scenario**   
  
In the previous hands on where we implemented getting country based on country code, what happens if the country code provided in the URL is not present.  
  
**Refer steps below to implement**

* Create a new exception class com.cognizant.springlearn.service.exception.CountryNotFoundException
* Include below specified annotation at the class level in CountryNotFoundException class

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

* In CountryService.getCountry() method include the logic to throw CountryNotFoundException if the country code does not exists in the list.
* In CountryController.getCountry() method include throws clause in method signature. This will respond to the caller of the web service with appropriate error message in JSON format.
* Test the service in postman and using curl command. Refer below for executing curl command.

**Steps to invoke RESTful Web Service using curl command**

* Open Git Bash
* Execute the below command

curl -i http://localhost:8090/country/az

**Sample Request**: http://localhost:8083/country/az  
  
**Sample Response**:

{

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

"status": 404,

"error": "Not Found",

"message": "Country not found",

"path": "/country/az"

}

Solution:

**Test Load Controller**

@Autowired

private CountryController countryController;

@Test

public void contextLoads() {

assertNotNull(countryController);

}

**Test Get Country Service**

@Autowired

private MockMvc mvc;

@Test

public void testGetCountry() throws Exception {

ResultActions actions = mvc.perform(get("/country"));

actions.andExpect(status().isOk())

.andExpect(jsonPath("$.code").value("IN"))

.andExpect(jsonPath("$.name").value("India"));

}

**Test Not Found Scenario**

@Test

public void testGetCountryException() throws Exception {

mvc.perform(get("/country/az"))

.andExpect(status().isNotFound());

}

**MockMVC - Test get country service**   
  
Using MockMVC test the get country service.  
  
Create a test cases to test the following aspects:

* Test is the CountryController is loaded
* Invoke the service to get country and check in the response if it contains code as "IN" and name as "India"

Refer steps below to implement

* **Test loading CountryController**
  + Include CountryController instance variable in SpringLearnApplicationTests.java and autowire the instance variable using annotation.

   @Autowired

    private CountryController countryController;

* Include assertion in contextLoads() method to check if controller is loaded.

   @Test

    public void contextLoads() {

        assertNotNull(countryController);

    }

* Run the JUnit testing by right clicking on SpringLearnApplicationTests.java > Run As > JUnit Test
* This test can also be executed in command line using the following maven command in the root folder of the project. (Note: don't forget to include proxy details in the below command)

mvn clean test

* Check if the log in the constructor of CountryController is called.

* **Test service to get the country**
  + Include below imports

import static org.junit.Assert.assertNotNull;

import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.ResultActions;

* Include @AutoConfigureMockMvc annotation for SpringLearnApplicationTests.java
* Autowire mock mvc in SpringLearnApplicationTests.java

  @Autowired

    private MockMvc mvc;

* Include a new test method in SpringLearnApplicationTests.java

   @Test

    public void testGetCountry() throws Exception {

}

* Include the following line in the new method that calls the service method. Execute the JUnit test and check if the test case is successful.

   @Test

    public void testGetCountry() throws Exception {

        ResultActions actions = mvc.perform(get("/country"));

}

* Include the following line to check if the HTTP Status is 200, which means the call is successful. Execute JUnit test and check if the test case is successful.

   @Test

    public void testGetCountry() throws Exception {

        ResultActions actions = mvc.perform(get("/country"));

        actions.andExpect(status().isOk());

}

* Include the following line to check if the code is available in the reponse

   @Test

    public void getCountry() throws Exception {

        ResultActions actions = mvc.perform(get("/country"));

        actions.andExpect(status().isOk());

        actions.andExpect(jsonPath("$.code").exists());

}

* Include the following line to check if the value of code is "IN"

   @Test

    public void getCountry() throws Exception {

        ResultActions actions = mvc.perform(get("/country"));

        actions.andExpect(status().isOk());

        actions.andExpect(jsonPath("$.code").exists());

        actions.andExpect(jsonPath("$.code").value("IN"));

}

* Using above two steps include checks for "name" attribute and check if it's value is "India"

**MockMVC - Test get country service for exceptional scenario**   
  
Include MockMVC test that checks if correct response is received when there is an error.  
  
Refer steps below to implement

* Include a new test method testGetCountryException() in SpringLearnApplicationTests.java
* Validate the error response using status(). Refer code below.

       actions.andExpect(status().isBadRequest());

        actions.andExpect(status().reason("Country Not found"));