[Spring RESTful Web Service - GET](javascript:void(0);)[Scores](javascript:void(0))

**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

[](https://cognizant.e-box.co.in/projectComponent/show/15714) [](https://cognizant.e-box.co.in/quiz/show/15721)

[Consuming Spring RESTful Web Service in Angular](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate integration of RESTful Web Service of type GET with Angular front end
  + REST Web Service architecture with Controller, Service and Dao, service methods, invoking service using http client in angular service, cross origin
    - Cross Origin reference - https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html

NOTE: There is no Quiz for this session

[](https://cognizant.e-box.co.in/projectComponent/show/15722)

[1 Spring RESTful Web Service - POST, PUT](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate implementation of RESTful Web Service using POST/PUT/DELETE method with input validation
  + HTTP method types (GET, POST, PUT, DELETE), REST service URL naming guidelines, @RequestMapping, @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, setting POST request payload and invoking the REST service in Postman and curl, JSON to bean mapping, @RequestBody, validating input request using javax.validation and hibernate validators, @Size, @NotNull, @NotBlank, @Min, @Max, @JsonFormat, @Valid, global exception handling, handle number formatting errors
    - HTTP Request Methods - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
    - RESTful API naming guide - https://restfulapi.net/resource-naming/
    - Request Mapping - https://docs.spring.io/spring/docs/5.2.0.RELEASE/spring-framework-reference/web.html#mvc-ann-requestmapping
    - Validation - https://www.mkyong.com/spring-boot/spring-rest-validation-example/

[](https://cognizant.e-box.co.in/projectComponent/show/15725) [](https://cognizant.e-box.co.in/quiz/show/15727)

[Consume PUT service in Angular and RESTful DELETE Service](javascript:void(0);)[Scores](javascript:void(0))

**Objectives defined in the previous session is applicable for this session as well.**  
  
NOTE: There is no Quiz for this session

[](https://cognizant.e-box.co.in/projectComponent/show/15728)

[Performance Testing RESTful Web Service](javascript:void(0);)[Scores](javascript:void(0))

**Objectives**

* Demonstrate executing performance testing of a restful web service using JMeter
  + Test Plan, Thread Groups, HTTP Request, Configuring Listeners, configuring thread groups with loading of user
    - JMeter User Manual - https://jmeter.apache.org/usermanual/index.html

**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

**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

**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

**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 Implemetation**: 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
* **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"}
* ]

**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 Implemetation**:

* 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"

}

**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"

}

**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"));

**Significance of HTTP Method Types in RESTful Web Services**  
  
SME to explain the importance of HTTP Method Types for RESTful Web Services.

|  |  |
| --- | --- |
| **HTTP Method** | **Usage Scenario** |
| GET | Used to get data about a resource |
| POST | Used to create a resource |
| PUT | Used to update a resource |
| DELETE | Used to delete a resource |

The method type is just a classification and does not actually have the persistence implemented. The respective application is expected to take responsibility in implementing the persistence.

**RESTful Web Service resource naming guidelines**  
  
Find below the guidelines to define a RESTful Web Service URL:

* Each resource should have a unique and specific URL
* To get all resources provide the resource name in plural
* To get a specific resource provide resource name in plural followed with slash and parameter
* To create a resource the URL should be the resource name in plural and the data to create the resource should be defined in the payload
* To update a resource the URL should be the resource name in plural with data in payload
* To delete a resource the URL should be the resource name in plural followed by slash and the specific resource to delete
* Resource name with multiple words should be separated by hyphen and not with underscore. For example, if the resource it menu item implement the URL as "menu-item".

Refer table below with example for resource as country.

|  |  |  |  |
| --- | --- | --- | --- |
| **Method Type** | **URL** | **Description** | **Annotation** |
| GET | http://sample.api.com/app-name/countries | Get all countries | @GetMapping |
| GET | http://sample.api.com/app-name/countries/{code} | Get a specific country | @GetMapping("/{id}") |
| POST | http://sample.api.com/app-name/countries | Create country based on data in post | @PostMapping |
| PUT | http://sample.api.com/app-name/countries | Update country based on data in post | @PutMapping |
| DELETE | http://sample.api.com/app-name/countries/{code} | Delete a specific country | @DeleteMapping("/{id}") |

For a particular resource, the URL should be the same for all the methods. Hence in CountryController, the URL can be defined at the class level:

@RequestMapping("/countries")

Find below the method specific annotation definitions:  
  
Get All

@GetMapping

Get specific resource

@GetMapping("/{id}")

Create resource

@PostMapping

NOTE: Payload data should be sent in the body of the request  
  
Update resource

@PutMapping

NOTE: Payload data should be sent in the body of the request  
  
Delete resource

@DeleteMapping("/{id}")

Going forward ensure that this convention is followed when defining a new service.  
  
Modify CountryController to adhere to the above mentioned standards.

**Create  RESTful Web Service to handle POST request of Country**  
  
A new RESTful Web Service method to handle POST request of Country. Follow steps below to incorporate the same:

* Create new method in CountryController based on the following details:
  + Annotation - @PostMapping()
  + Method Signature - public void addCountry()
* Within this method  include "Start" logger.
* Start the web application
* Open Git Bash
* Execute the following curl command, to invoke the web service:
  + -i to display the headers
  + -X to define the HTTP method type
  + -s silent mode, so that performance details are not displayed

curl -i -X POST -s http://localhost:8090/countries

* Check if "Start" is displayed in the console output
* Following is the expected output:

HTTP/1.1 200

Content-Length: 0

Date: Tue, 01 Oct 2019 06:41:49 GMT

* The invocation of web service can also be done using Postman.
* Check the logger if "Start" is logged

**Read country data as a bean in RESTful Web Service**  
  
The country data should be included in the request payload, which should be read by the controller method.  
  
Follow steps below to incorporate the same:

* Include country as parameter to addCountry() method with @RequestBody annotation and country as parameter. Refer method signature below.

public Country addCountry(@RequestBody Country country)

* Include log to display country details
* Return the country. This is to check if country details are populated correctly
* Invoke the service using the following curl command. This can also be tried for execution from Postman.
  + -H denotes inclusion of header. This denotes that we are sending content type in the request header and it mentions that the request payload is of type JSON
  + -d denotes the data payload sent in the request. This represents the country to be added

curl -i -H 'Content-Type: application/json' -X POST -s -d '{"code":"IN","name":"India"}' http://localhost:8090/countries

* Refer the expected HTTP response below:

HTTP/1.1 200

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

Transfer-Encoding: chunked

Date: Tue, 01 Oct 2019 17:23:47 GMT

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

* Try running the request with minor change and let us see the response. Sample response below. The attribute name is intentionally provided with a spelling mistake.

curl -i -H 'Content-Type: application/json' -X POST -s -d '{"code":"IN","nae":"India"}' http://localhost:8090/countries

* Refer the expected HTTP response below:

HTTP/1.1 200

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

Transfer-Encoding: chunked

Date: Tue, 01 Oct 2019 17:23:47 GMT

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

**SME to provide explanation about the following aspects:**

* Explain how spring framework takes care of converting the request payload into country bean
* Spring parses the JSON request payload data using Jackson parser
* For each attribute in JSON, respective method name is constructed by applying initcaps and get prefix. For example, the name attribute is changed with initcaps as Name, then get is prefixed to it which results in getName, based on this the respective method is invoked using Reflection API.
* Spring creates country object and invokes the respective setter method based on JSON data.
* The it invokes the controller method passing the country object created
* Provide explanation regarding bean naming conventions

**Validating country code**  
  
As the POST request is a plain text, there are good possibilities to key in incorrect data. Moreover, hackers might try to pass inconsistent data which might affect the integrity of the application. Hence it becomes important that necessary check are in place for all the fields. In this hands on we will take a simple validation criteria and will see how it can be implemented.  
  
The country code needs to be validated and ensured that it does not exceed more than 2 characters. Refer the steps below to incorporate the same:

* Open Country.java and include below annotations for the code property. @NotNull ensure that code is not null. @Size ensure that the width is exactly 2 characters.

    @NotNull

    @Size(min=2, max=2, message="Country code should be 2 characters")

    private String code;

* In CountryController.addCountry() method add below lines after the logger. This uses the javax.validation specification to check if the bean has errors based on the annotations defined in the earlier step. All new class references in this code snippet needs to be imported from javax.validation.

// Create validator factory

       ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

        Validator validator = factory.getValidator();

  // Validation is done against the annotations defined in country bean

        Set<ConstraintViolation<Country>> violations = validator.validate(country);

        List<String> errors = new ArrayList<String>();

  // Accumulate all errors in an ArrayList of type String

        for (ConstraintViolation<Country> violation : violations) {

            errors.add(violation.getMessage());

        }

  // Throw exception so that the user of this web service receives appropriate error message

        if (violations.size() > 0) {

            throw new ResponseStatusException(HttpStatus.BAD\_REQUEST, errors.toString());

        }

* Invoke the service using curl and check the response. Refer sample response below:

HTTP/1.1 400

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

Transfer-Encoding: chunked

Date: Wed, 02 Oct 2019 10:28:56 GMT

Connection: close

{"timestamp":"2019-10-02T10:28:56.506+0000","status":400,"error":"Bad Request","message":"[Country code should be 2 characters]","path":"/countries"}

**Question for all Learners -**What needs to be done if there is another controller EmployeeController and similar validation needs to be done for Employee payload data?  
  
SME to explain the disadvantage of the above solution.  
  
This disadvantage will be overcome in the next hands on.

**Include global exception handler for validation errors**  
  
Following steps create a global validation error handler. This will validate all errors that may happen in any controller.  
  
**Create global exception handler**

* Create class com.cognizant.springlearn.GlobalExceptionHandler that extends ResponseEntityExceptionHandler with annotation @ControllerAdvice
* Include method handler for handling the validation error and include a start logger within the method implementation.

    @Override

    protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,

            HttpHeaders headers, HttpStatus status, WebRequest request) {

        LOGGER.("Start");

    }

* Refer imports below:

import org.springframework.http.HttpHeaders;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.MethodArgumentNotValidException;

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

import org.springframework.web.context.request.WebRequest;

import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

* Include @Valid annotation in the addCountry() method. This initimates spring framework to validate the country bean based on the validation annotations added in the Country class. Refer code below:

public Country addCountry(@RequestBody @Valid Country country)

* Remove all the validation code included in the previous hands on.
* Run the application and invoke the curl request with single character for country code.

curl -i -H 'Content-Type: application/json' -X POST -s -d '{"code":"I","name":"India"}' http://localhost:8090/countries

* Check the logs and see if the start logger is present. Also notice that the logs of CountryController is not present, which means that the global exception handler method is called if there are validation errors and the controller method is not invoked.

**Response with bad request in global exception handler**

* Include the below code in the handleMethodArgumentNotValid() method:

        // Map that contains the error details

        Map<String, Object> body = new LinkedHashMap<>();

        body.put("timestamp", new Date());

        body.put("status", status.value());

        // Get all validation errors

        List<String> errors = ex.getBindingResult().getFieldErrors().stream().map(x -> x.getDefaultMessage())

                .collect(Collectors.toList());

// Add errors to the response map

        body.put("errors", errors);

  LOGGER.info("End");

        return new ResponseEntity<>(body, headers, status);

* Execute the updated web application and execute the curl command with single character for country code
* See expected response below.

HTTP/1.1 400

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

Transfer-Encoding: chunked

Date: Thu, 03 Oct 2019 04:10:17 GMT

Connection: close

{"timestamp":"2019-10-03T04:10:17.277+0000","status":400,"errors":["Country code should be 2 characters"]}

**Implement REST service for updating an employee**  
  
Based on the learning done with REST service for country, implement a service to update employee details.  
  
Follow steps below to incorporate the same:

* Include below validations in Employee, Department and Skill beans
  + Employee
    - id - should not be null, should be a number
    - name - should not be null, should not be blank, minimum 1 character and maximum 30 characters
    - salary - should not be null, should zero or above
    - permanent - should not be null
    - dateOfBirth - should match the date pattern. Use below annotation

@JsonFormat(shape=JsonFormat.Shape.STRING, pattern="dd/MM/yyyy")

* Department
  + id - should not be null, should be a number
  + name - should not be null, should not be blank, minimum 1 character and maximum 30 characters
* Skill
  + id - should not be null, should be a number
  + name - should not be null, should not be blank, minimum 1 character and maximum 30 characters

* Implement the Employee service with below aspects incorporated:
  + Define EmployeeNotFoundException with HttpStatus annotation
  + Include updateEmployee() method in EmployeeDao that modifies employee list. If the employee is not found throw EmployeeNotFoundException.
  + Include updateEmployee() method in EmployeeService that invokes the dao update employee method
  + Include updateEmployee() method in EmployeeController with below signature with @PutMapping annotation. Refer method signature below:
  + public void updateEmployee(@RequestBody @Valid Employee employee) throws EmployeeNotFoundException
  + Follow necessary URL guidelines for the above method signature.
  + If string value is included in a numeric field (for example: id), the failure happens even before validation, include a new method in global exception handler which handles this scenario. Refer code below:

 protected ResponseEntity<Object> handleHttpMessageNotReadable(

            HttpMessageNotReadableException ex, HttpHeaders headers, HttpStatus status,

            WebRequest request) {

        Map<String, Object> body = new LinkedHashMap<>();

        body.put("timestamp", new Date());

        body.put("status", status.value());

  body.put("error", "Bad Request");

        List<String> errors = new ArrayList<String>();

        if (ex.getCause() instanceof InvalidFormatException) {

            final Throwable cause = ex.getCause() == null ? ex : ex.getCause();

            for (InvalidFormatException.Reference reference : ((InvalidFormatException) cause).getPath()) {

                body.put("message", "Incorrect format for field '" + reference.getFieldName() + "'");

            }

        }

        return new ResponseEntity<>(body, headers, status);

    }

* Test the service using Postman passing the employee data as JSON, which should include department and skills.
* Using Postman invoke get all employees service to verify if the update is reflected

* Include MockMvc test for the exceptional scenario

**Consume PUT Employee service in Angular component**  
  
Perform steps below to implement consumption of service to update employee details.  
  
**employee.ts**

* Modify the dateOfBirth data type as string, as we have included @JsonFormat which accepts date of birth as string.
* Update the changes in all the places where dateOfBirth value is defined in 'dd/mm/yyyy' format.

**employee.service.ts**

* Include constant for setting the content type of the request as JSON type. Include below code above @Injectable decorator defintion.

const httpOptions = {

  headers: new HttpHeaders({

    'Content-Type': 'application/json',

  })

};

* Include a new method updateEmployee which makes the PUT method call:

  updateEmployee(employee: Employee): Observable<void> {

    return this.httpClient.put<void>(this.employeeApiUrl, employee, httpOptions);

  }

**edit-employee.template.component.css**

* Include a new style for displaying success message

.success {

  color:green;

}

**edit-employee.template.component.ts**

* Include a new property error of type string for displaying the error message if the form submission returns error.
* Incorporate the below code in the onSubmit() event that is triggered when save button is clicked. The error section gets the message from the error response and sets the error property value.

  onSubmit() {

    this.employeeService.updateEmployee(this.employee).subscribe(

      data => {

        console.log('Employee update successful.');

        this.error = '';

      },

      error => {

        console.log(error);

        this.error = error.error.message;

        if (error.error.errors != null) {

          this.error = error.error.errors[0];

        }

      });

  }

**edit-employee.template.component.html**

* Include the below div above the Save button to display the error message.

    <div \*ngIf="error == ''" class="success">Employee details saved successfully.</div>

    <div \*ngIf="error != ''" class="text-error">{{error}}</div>

Test various scenarios as specified below by changing test date:

* Define dateOfBIrth value in incorrect date format
* Negative value for salary
* Success scenario where all field values are correct

**Consume PUT Employee service in reactive form component**  
  
Incorporate edit employee form in the edit employee reactive component.  
  
Consider below aspects when implementing:

* Reuse the updateEmployee() service method earlier
* On submission of Save button, get the employee form data from FormGroup object
* Refer previous hands on for rest of the implementation.

**Implement REST DELETE Service**  
  
Implement steps below to complete this:

* Implement a new delete service for employee by incorporating relevant code in EmployeeController, EmployeeService and EmployeeDao.
* The EmployeeDao should have the code to remove the respective item from the list or throw EmployeeNotFoundException if the id not found.
* Test the service and check if deletion happens correctly
* In employee-list component include a delete icon for each employee in the list
* On click of this icon include necessary coding in the component to invoke a new service that calls the new delete service created in the earlier steps.

**Setup and Run JMeter**  
  
Follow steps below to execute JMeter:

* Extract the apache-jmeter-5.1.1zip file to D:
* Go to D:\apache-jmeter-5.1.1\bin in windows file explorer and execute the file jmeter.bat
* Check if this opens JMeter application

NOTE: If OneIT team did not install JMeter, SME can download apache-jmeter-5.1.1 from SharePoint

**Test your favorite internet web site using JMeter**  
  
Follow steps below to configure testing of your favorite website:  
  
**Configuring the HTTP Request**

* JMeter launch opens up new test plan creation option
* Key in the test plan name as 'favorite-site-test'
* Right click on the 'Test Plan' in the left hands side and choose the below option:
  + Add > Thread (User) > Thread Group
* Leave the 'Thread Group' with default values
* Right click on 'Thread Group and select below option:
  + Add > Sampler > HttpRequest
* Set protocol as http or https based on the prefix of your favorite website url (for example, if the URL is https://www.google.com, then enter https)
* Click Advanced tab and set the proxy details:
  + Server Name: proxy.cognizant.com
  + Port Number: 6050
  + User Name: Employee ID
  + Password: Network Password
* Enter host name (Example: www.google.com or www.cognizant.com)
* Right click on 'Thread Group' and select the below specified options separately:
  + Add > Listener > View Results Tree
  + Add > Listener > View Results in Table
* Now all configuration is set. Save the test plan, using the menu 'File > Save' option

**Executing the test and analysing the results**

* To execute the test using the green play icon or using menu 'Run > Start'
* Click on View Results Tree and View Result in Table to see the results
* View Results Tree
  + Click on 'HTTP Request' in the left hand side
  + Click below tabs and view the details
    - Sampler Result
    - Request > Request Body
    - Request > Request Header
    - Response Data
* View Results in Table
  + Sample Time (ms) - denotes the total time taken to receive the response. This data is represented in milliseconds (1 Second = 1000 milliseconds)
  + Status - HTTP Response Status
  + Bytes - Number of bytes received in response
  + Sent Bytes - Number of bytes sent as request
  + Connect Time (ms) - Number of milliseconds it took to connect to the server

**Executing with multiple requests**

* Clear the existing results by right clicking on "View Results Tree' and 'View Results in Table', then selecting the 'Clear' option in the menu.
* Click on 'Thread Group'
* Change 'Number of Threads' to 2
* Run the test using play button.
* Now there will be two requests present in 'View Results in Table'

**Execute Performance Testing of RESTful Web Service with JMeter**  
  
The RESTful Service for getting all countries needs to performance tested. Identify the maximum threshold of your service.  
  
Follow steps below to complete the performance testing activity:

* Create a new Test Plan using menu File > New
* Name the test plan as 'get-countries-perf-test'
* Follow all configuration done in the previous hands on, but following configuration values needs to be changed:
  + Thread Group > Number of Threads = 10
  + HTTP Request > Basic
    - Protocol: http
    - Server Name: localhost
    - Port Number: 8090
    - Method: GET
    - Path: /countries
* Execute the test and see the results
* Save the test plan
* Change the Number of Threads gradually by increasing to 20, 30, etc. and observe the gradual increase in response time.
* In Thread Group modify the Loop Count value to 2 and the result. This will execute the threads in groups one after the other rather than in a single execution.

**Solving a business problem**  
  
The REST Service that returns all countries has to be deployed in production.  
  
It is expected that at any given point of time maximum 30 requests can be received for this service when it is in production.  
  
Also the business expects that the service should respond within 3 seconds.  
  
Given the above scenario verify if the service will be able to accomplish the above requirements or not.  
  
**NOTE:** This problem needs to be done by the learners independently with guidance from the SME.