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/

Page Break

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.

Page Break

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.

Page Break

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

Page Break

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

Page Break

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(count
ry);
        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.to
String());
```

```
        }
```

- 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.

Page Break

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.ResponseEntityExceptionHa
ndler;
```

- 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().m
ap(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.getFieldNam
e() + "'");
            }
        }

        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

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