**Objectives**

## ****1. Overview of HTTP Methods****

| **Method** | **Purpose** |
| --- | --- |
| GET | Retrieve resource |
| POST | Create resource |
| PUT | Update resource |
| DELETE | Remove resource |

Reference: [MDN HTTP Methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods" \t "_new)

**2. REST URL Naming Guidelines**

Use **nouns** (not verbs): /users, /departments

Use plural names: /products not /product

Nest resources logically: /departments/{id}/employees

Use query parameters for filtering/sorting: /employees?dept=HR&sort=name

Reference: [RESTful API Naming](https://restfulapi.net/resource-naming/" \t "_new)

**3. Spring Annotations**

| **Annotation** | **Description** |
| --- | --- |
| @RestController | Declares REST controller |
| @RequestMapping | Base path mapping |
| @GetMapping | Maps GET requests |
| @PostMapping | Maps POST requests |
| @PutMapping | Maps PUT requests |
| @DeleteMapping | Maps DELETE requests |
| @RequestBody | Binds JSON to Java object |
| @Valid | Enables validation on request body |

Reference: [Spring @RequestMapping](https://docs.spring.io/spring/docs/5.2.0.RELEASE/spring-framework-reference/web.html" \l "mvc-ann-requestmapping" \t "_new)

**4. Example Project Structure**

src/

├── controller/

│ └── UserController.java

├── model/

│ └── User.java

├── service/

│ └── UserService.java

├── exception/

│ └── GlobalExceptionHandler.java

**5. Example: User Entity with Validation**

import javax.validation.constraints.\*;

public class User {

private int id;

@NotBlank(message = "Name is mandatory")

private String name;

@Min(value = 18, message = "Age should be at least 18")

@Max(value = 100, message = "Age should be less than or equal to 100")

private int age;

@Email(message = "Invalid email format")

private String email;

// Getters and setters

}

**6. Controller with POST/PUT/DELETE**

@RestController@RequestMapping("/users")public class UserController {

@PostMapping

public ResponseEntity<String> createUser(@Valid @RequestBody User user) {

// save logic

return new ResponseEntity<>("User created", HttpStatus.CREATED);

}

@PutMapping("/{id}")

public ResponseEntity<String> updateUser(@PathVariable int id, @Valid @RequestBody User user) {

// update logic

return ResponseEntity.ok("User updated");

}

@DeleteMapping("/{id}")

public ResponseEntity<String> deleteUser(@PathVariable int id) {

// delete logic

return ResponseEntity.ok("User deleted");

}

}

**7. Global Exception Handler**

@ControllerAdvicepublic class GlobalExceptionHandler {

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<Map<String, String>> handleValidationErrors(MethodArgumentNotValidException ex) {

Map<String, String> errors = new HashMap<>();

ex.getBindingResult().getFieldErrors().forEach(error ->

errors.put(error.getField(), error.getDefaultMessage()));

return new ResponseEntity<>(errors, HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(NumberFormatException.class)

public ResponseEntity<String> handleNumberFormat(NumberFormatException ex) {

return new ResponseEntity<>("Invalid number format", HttpStatus.BAD\_REQUEST);

}

}

**8. Testing with Postman / curl**

### POST (Create):

POST /users

Content-Type: application/json{

"name": "Alice",

"age": 25,

"email": "alice@example.com"}

### curl:

curl -X POST http://localhost:8080/users \

-H "Content-Type: application/json" \

-d '{"name":"Alice","age":25,"email":"alice@example.com"}'

**Significance of HTTP Method Types in RESTful Web Services**

RESTful Web Services are built on top of **HTTP**, and they use **standard HTTP methods** to perform **CRUD operations** (Create, Read, Update, Delete) on resources. Each method has a **specific semantic meaning**, which brings **clarity**, **consistency**, and **best practices** to API design.

**1. GET** – Read a resource

**Usage**: Retrieve a resource or a list of resources.

**Example**: GET /users/101 → Returns user with ID 101.

**Key Point**: Should not modify the state of the server (idempotent, safe).

### ****2. POST**** – Create a new resource

**Usage**: Add a new resource to the server.

**Example**: POST /users with body { "name": "Alice" }

**Key Point**: Not idempotent (each request creates a new resource).

### ****3. PUT**** – Update an existing resource

**Usage**: Replace the entire resource.

**Example**: PUT /users/101 with updated user details.

**Key Point**: Idempotent (multiple identical requests yield the same result).

### ****4. DELETE**** – Delete a resource

**Usage**: Remove a resource from the server.

**Example**: DELETE /users/101

**Key Point**: Idempotent (deleting same resource twice has no additional effect).

Important Clarification:

“The method type is just a classification and does not actually have the persistence implemented.”

This means that **HTTP methods only define the** intent **of the operation**, but **don’t perform any database actions on their own**.  
It’s the **application logic** behind these methods that:

Talks to the database

Stores, updates, or deletes data

Returns appropriate responses

For example:

POST /users alone won’t save anything unless you write logic in your controller and service layer to persist the data.

**RESTful Web Service resource naming guidelines**   
To ensure consistency, readability, and adherence to RESTful principles, follow the below guidelines when defining URLs for Web Services:

**Each resource should have a unique and specific URL**  
Example: /countries, /departments

**To get all resources**, use the **plural form** of the resource name  
Example: GET /countries

**To get a specific resource**, use the plural form followed by the identifier  
Example: GET /countries/{code}

**To create a resource**, use POST with the plural form in the URL. Data should be sent in the **request body**  
Example: POST /countries

**To update a resource**, use PUT with the plural form. Updated data should be in the **request body**  
Example: PUT /countries

**To delete a resource**, use DELETE with the plural form followed by the identifier  
Example: DELETE /countries/{code}

**For multi-word resources**, use hyphens -, not underscores \_  
Example: menu-item , menu\_item

Example Resource: 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 specific country | @GetMapping("/{code}") |
| POST | http://sample.api.com/app-name/countries | Create a new country | @PostMapping |
| PUT | http://sample.api.com/app-name/countries | Update a country | @PutMapping |
| DELETE | http://sample.api.com/app-name/countries/{code} | Delete specific country | @DeleteMapping("/{code}") |

**Note**: For a particular resource, the base URL should remain the same across all operations. In the controller, this can be defined at the class level using:  
@RequestMapping("/countries")

Method-Specific Annotation Mapping:

**Get all countries**  
@GetMapping

**Get a specific country**  
@GetMapping("/{code}")

**Create a new country**  
@PostMapping  
(Data should be sent in the body)

**Update a country**  
@PutMapping  
(Data should be sent in the body)

**Delete a country**  
@DeleteMapping("/{code}")

Modified CountryController (as per standard)

package com.cognizant.springrest.controller;

import com.cognizant.springrest.model.Country;import com.cognizant.springrest.service.CountryService;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.http.ResponseEntity;import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;import java.util.List;

@RestController@RequestMapping("/countries") // Base URL for the resourcepublic class CountryController {

@Autowired

private CountryService countryService;

// Get all countries

@GetMapping

public List<Country> getAllCountries() {

return countryService.getAllCountries();

}

// Get a specific country by code

@GetMapping("/{code}")

public Country getCountry(@PathVariable String code) {

return countryService.getCountry(code);

}

// Create a new country

@PostMapping

public ResponseEntity<String> addCountry(@Valid @RequestBody Country country) {

countryService.addCountry(country);

return ResponseEntity.ok("Country created successfully");

}

// Update an existing country

@PutMapping

public ResponseEntity<String> updateCountry(@Valid @RequestBody Country country) {

countryService.updateCountry(country);

return ResponseEntity.ok("Country updated successfully");

}

// Delete a country by code

@DeleteMapping("/{code}")

public ResponseEntity<String> deleteCountry(@PathVariable String code) {

countryService.deleteCountry(code);

return ResponseEntity.ok("Country deleted successfully");

}

}

**Create RESTful Web Service to handle POST request of Country**

A new RESTful Web Service method is created in the CountryController class to handle a POST request. The method uses the @PostMapping annotation with the signature public void addCountry(). Inside the method, a logger is used to log the message "Start" to indicate the invocation of the service.

The controller looks as follows:

package com.cognizant.springrest.controller;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;

import org.springframework.web.bind.annotation.\*;

@RestController@RequestMapping("/countries")public class CountryController {

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

@PostMapping

public void addCountry() {

LOGGER.info("Start");

}

}

After starting the Spring Boot application on port 8090, the endpoint can be invoked using the following curl command in Git Bash:

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

This command sends a silent POST request and displays the HTTP response headers. Upon successful execution, the expected output is:

HTTP/1.1 200Content-Length: 0Date: Tue, 01 Oct 2019 06:41:49 GMT

The log output in the Spring Boot console should display:

INFO [CountryController] - Start

Alternatively, the same POST request can be tested using Postman. In both cases, the appearance of "Start" in the console confirms that the POST method has been successfully invoked.

**Read country data as a bean in RESTful Web Service**

A RESTful Web Service method is created in the CountryController to read **country details from the request payload** using the @RequestBody annotation. The incoming JSON data is automatically converted into a Country object by Spring using Jackson.

The updated controller method is:

package com.cognizant.springrest.controller;

import com.cognizant.springrest.model.Country;import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.web.bind.annotation.\*;

@RestController@RequestMapping("/countries")public class CountryController {

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

@PostMapping

public Country addCountry(@RequestBody Country country) {

LOGGER.info("Start");

LOGGER.info("Country: {}", country);

return country;

}

}

Testing with curl

The REST endpoint can be tested using the following command:

curl -i -H 'Content-Type: application/json' -X POST -s \

-d '{"code":"IN","name":"India"}' http://localhost:8090/countries

### Expected Output:

HTTP/1.1 200

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

Transfer-Encoding: chunkedDate: Tue, 01 Oct 2019 17:23:47 GMT

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

Testing with Incorrect Attribute Name

If you intentionally provide a wrong attribute name, for example:

curl -i -H 'Content-Type: application/json' -X POST -s \

-d '{"code":"IN","nae":"India"}' http://localhost:8090/countries

### Expected Output:

HTTP/1.1 200

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

Transfer-Encoding: chunkedDate: Tue, 01 Oct 2019 17:23:47 GMT

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

In this case, name is not set because the attribute nae does not match any field in the bean.

SME Explanation: How Spring Converts JSON Payload to Bean

### ****Automatic Conversion using Jackson****

Spring Boot uses the **Jackson JSON parser** under the hood to convert incoming JSON into Java objects.

### ****How Mapping Works****

For each attribute in the JSON payload:

Spring constructs a method name using **InitCaps** (capitalizing the first letter).

It prefixes it with set to match the setter method of the bean.

Example:

JSON attribute: "name"

Becomes method: setName(String name)

Spring then uses **Java Reflection API** to find this method in the Country class and invoke it to set the value.

### 3. ****Object Creation****

Spring creates an instance of the Country bean using the default constructor.

It sets all the matching fields using their respective setter methods.

4. **Calling the Controller**

Once the bean is populated, Spring **passes it as an argument** to the controller method:

public Country addCountry(@RequestBody Country country)

Bean Naming Conventions

To enable proper mapping between JSON fields and Java fields, follow these conventions:

Field names should be in **camelCase** (e.g., countryName, countryCode)

Each field must have a corresponding **getter** and **setter**:

private String name;

public String getName() and public void setName(String name)

Correct conventions are important because Jackson relies on these method names to set and retrieve values.

**Validating country code**

To ensure data integrity and prevent malicious or invalid input, it's essential to apply field-level validation for all request payloads. In this scenario, the code property of the Country object must be validated to ensure it is **not null** and exactly **2 characters** long.

This is achieved using validation annotations in the Country model class. The @NotNull annotation ensures that the code is not missing, and @Size(min=2, max=2) ensures the code has exactly two characters. A custom message can be provided to clearly indicate validation failure.

The controller method addCountry() is responsible for handling POST requests. After logging the request start, it performs validation using the Validator API from javax.validation. If any constraint violations are detected, they are collected and returned to the client as a 400 Bad Request using ResponseStatusException.

### Country.java

import javax.validation.constraints.NotNull;import javax.validation.constraints.Size;

public class Country {

@NotNull

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

private String code;

private String name;

// Getters and Setters

}

### CountryController.java

import com.cognizant.springrest.model.Country;import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.http.HttpStatus;import org.springframework.web.bind.annotation.\*;import org.springframework.web.server.ResponseStatusException;

import javax.validation.\*;import java.util.\*;

@RestController@RequestMapping("/countries")public class CountryController {

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

@PostMapping

public Country addCountry(@RequestBody Country country) {

LOGGER.info("Start");

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

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

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

for (ConstraintViolation<Country> violation : violations) {

errors.add(violation.getMessage());

}

if (!violations.isEmpty()) {

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

}

LOGGER.info("Country: {}", country);

return country;

}

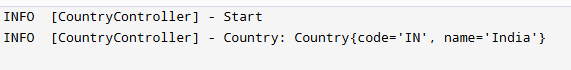
}

### Sample cURL Invocation

curl -i -H 'Content-Type: application/json' -X POST -s \

-d '{"code":"IND","name":"India"}' <http://localhost:8090/countries>

**OUTPUT:**



**Include global exception handler for validation errors**

In a RESTful web service, input validation is essential to ensure data integrity and protect against malformed or malicious requests. Rather than manually validating input in every controller, Spring Boot allows centralized validation using a **global exception handler**.

Spring provides the @ControllerAdvice annotation, which can be used to intercept and handle validation exceptions across all controllers. The validation is triggered automatically using the @Valid annotation in the controller method parameter.

The CountryController method uses the @Valid annotation to trigger validation based on the constraints defined in the Country bean:

@PostMappingpublic Country addCountry(@RequestBody @Valid Country country) {

LOGGER.info("Start");

LOGGER.info("Country: {}", country);

return country;

}

A global exception handler class is created to handle any validation errors that occur during the binding of the request payload to the Java object. This class extends ResponseEntityExceptionHandler and overrides the handleMethodArgumentNotValid() method. When a validation error occurs, this method is automatically called, bypassing the controller method itself.

package com.cognizant.springlearn;

import org.slf4j.Logger;import org.slf4j.LoggerFactory;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;

import java.util.\*;import java.util.stream.Collectors;

@ControllerAdvicepublic class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

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

@Override

protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,

HttpHeaders headers,

HttpStatus status,

WebRequest request) {

LOGGER.info("Start");

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

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

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

List<String> errors = ex.getBindingResult()

.getFieldErrors()

.stream()

.map(x -> x.getDefaultMessage())

.collect(Collectors.toList());

body.put("errors", errors);

LOGGER.info("End");

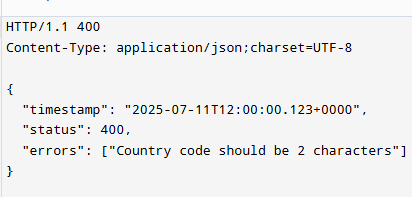
return new ResponseEntity<>(body, headers, status);

}

}

When a request with an invalid country code is sent (e.g., one character instead of two), the validation fails before reaching the controller. The global exception handler intercepts the error, logs the start and end of the handling process, and returns a structured response.

**OUTPUT:**



**Implement REST service for updating an employee**

### 1. Employee, Department, and Skill Validation Annotations

#### Employee.java

import javax.validation.constraints.\*;import com.fasterxml.jackson.annotation.JsonFormat;

public class Employee {

@NotNull

private Integer id;

@NotBlank

@Size(min = 1, max = 30)

private String name;

@NotNull

@Min(0)

private Double salary;

@NotNull

private Boolean permanent;

@NotNull

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

private Date dateOfBirth;

@NotNull

private Department department;

@NotNull

private List<Skill> skillList;

// Getters and Setters

}

#### Department.java

public class Department {

@NotNull

private Integer id;

@NotBlank

@Size(min = 1, max = 30)

private String name;

// Getters and Setters

}

#### Skill.java

public class Skill {

@NotNull

private Integer id;

@NotBlank

@Size(min = 1, max = 30)

private String name;

// Getters and Setters

}

2. EmployeeNotFoundException.java

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

@ResponseStatus(HttpStatus.NOT\_FOUND)public class EmployeeNotFoundException extends Exception {

public EmployeeNotFoundException(String message) {

super(message);

}

}

3. EmployeeDao.java

import java.util.\*;

@Repositorypublic class EmployeeDao {

private static List<Employee> EMPLOYEE\_LIST = new ArrayList<>();

public void updateEmployee(Employee updatedEmployee) throws EmployeeNotFoundException {

boolean found = false;

for (int i = 0; i < EMPLOYEE\_LIST.size(); i++) {

if (EMPLOYEE\_LIST.get(i).getId().equals(updatedEmployee.getId())) {

EMPLOYEE\_LIST.set(i, updatedEmployee);

found = true;

break;

}

}

if (!found) {

throw new EmployeeNotFoundException("Employee not found with id: " + updatedEmployee.getId());

}

}

public List<Employee> getAllEmployees() {

return EMPLOYEE\_LIST;

}

}

4. EmployeeService.java

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

@Servicepublic class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

public void updateEmployee(Employee employee) throws EmployeeNotFoundException {

employeeDao.updateEmployee(employee);

}

}

5. EmployeeController.java

import org.slf4j.Logger;import org.slf4j.LoggerFactory;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.web.bind.annotation.\*;

import javax.validation.Valid;import java.util.List;

@RestController@RequestMapping("/employees")public class EmployeeController {

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

@Autowired

private EmployeeService employeeService;

@PutMapping

public void updateEmployee(@RequestBody @Valid Employee employee) throws EmployeeNotFoundException {

LOGGER.info("Start");

employeeService.updateEmployee(employee);

LOGGER.info("End");

}

@GetMapping

public List<Employee> getAllEmployees() {

return employeeService.employeeDao.getAllEmployees();

}

}

6. GlobalExceptionHandler.java (additional method for parsing errors)

import com.fasterxml.jackson.databind.exc.InvalidFormatException;import org.springframework.http.\*;import org.springframework.web.bind.annotation.ControllerAdvice;import org.springframework.web.context.request.WebRequest;import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;

@ControllerAdvicepublic class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

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

@Override

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");

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);

}

}

**OUTPUT:**



**Implement REST DELETE Service**

### ****1. EmployeeNotFoundException.java****

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

@ResponseStatus(HttpStatus.NOT\_FOUND)public class EmployeeNotFoundException extends RuntimeException {

public EmployeeNotFoundException(String message) {

super(message);

}

}

**2. EmployeeDao.java**

import java.util.\*;import org.springframework.stereotype.Repository;

@Repositorypublic class EmployeeDao {

private static List<Employee> EMPLOYEE\_LIST = new ArrayList<>();

// DELETE method

public void deleteEmployee(int id) {

boolean removed = EMPLOYEE\_LIST.removeIf(emp -> emp.getId() == id);

if (!removed) {

throw new EmployeeNotFoundException("Employee not found with id: " + id);

}

}

public List<Employee> getAllEmployees() {

return EMPLOYEE\_LIST;

}

// For testing: Add method to insert employees

public void addEmployee(Employee employee) {

EMPLOYEE\_LIST.add(employee);

}

}

**3. EmployeeService.java**

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

@Servicepublic class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

public void deleteEmployee(int id) {

employeeDao.deleteEmployee(id);

}

}

**4. EmployeeController.java**

import org.springframework.beans.factory.annotation.Autowired;import org.springframework.web.bind.annotation.\*;

@RestController@RequestMapping("/employees")public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// DELETE endpoint

@DeleteMapping("/{id}")

public void deleteEmployee(@PathVariable int id) {

employeeService.deleteEmployee(id);

}

}

**OUTPUT:**

#### ****Case 1: Employee Found and Deleted****

**Request:**

DELETE http://localhost:8080/employees/101

**Response:**

Status: 200 OKBody: (empty)

Or (if customized in controller to return a message):

Status: 200 OK

Body:"Employee deleted successfully"

**Case 2: Employee Not Found**

**Request:**

DELETE http://localhost:8080/employees/999

**Response:**

Status: 404 Not Found

Body:{

"timestamp": "2025-07-11T15:35:00.456+00:00",

"status": 404,

"error": "Not Found",

"message": "Employee not found with id: 999",

"path": "/employees/999"}