# SPRING REST USING SPRING BOOT

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

CountryController.java

package com.cognizant.spring\_learn.controller;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class CountryController {

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

// Existing methods...

@PostMapping("/countries") public void addCountry() {

LOGGER.info("Start");

// For now, no implementation since only logging is requested.

}

}

# output

HTTP/1.1 200

Content-Length: 0 Date: ...

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

Country.java

package com.cognizant.spring\_learn.model;

public class Country { private String code; private String name;

public Country() {

}

public Country(String code, String name) { this.code = code;

this.name = name;

}

// Getter and Setter methods public String getCode() {

return code;

}

public void setCode(String code) { this.code = code;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

}

# CountryController.java

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.model.Country; import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class CountryController {

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

// Existing GET methods (not shown here)

/\*\*

\* POST method to accept Country JSON, log it, and return it.

\*/

@PostMapping("/countries")

public Country addCountry(@RequestBody Country country) {

LOGGER.info("Received country: code = {}, name = {}", country.getCode(), country.getName());

// Return the same country object to confirm JSON parsing was successful return country;

}

}

# output

HTTP/1.1 200

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

...

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

Validating country code

Country.java

package com.cognizant.spring\_learn.model;

import jakarta.validation.constraints.NotNull; import jakarta.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;

public Country() {

}

public Country(String code, String name) { this.code = code;

this.name = name;

}

// getters and setters

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

}

# CountryController.java

import jakarta.validation.ConstraintViolation; import jakarta.validation.Validation;

import jakarta.validation.Validator;

import jakarta.validation.ValidatorFactory;

import org.springframework.http.HttpStatus;

import org.springframework.web.server.ResponseStatusException;

import java.util.ArrayList; import java.util.List; import java.util.Set;

@PostMapping("/countries")

public Country addCountry(@RequestBody Country country) {

LOGGER.info("Received country: code = {}, name = {}", country.getCode(), country.getName());

// Create validator factory and validator instance

ValidatorFactory factory = Validation.buildDefaultValidatorFactory(); Validator validator = factory.getValidator();

// Validate country object based on annotations

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 exception with all validation errors collected

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

}

return country;

}

Include global exception handler for validation errors

Country.java

package com.cognizant.springlearn.model;

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;

public Country() {}

public Country(String code, String name) {

this.code = code; this.name = name;

}

public String getCode() { return code;

}

public void setCode(String code) { this.code = code;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

}

# CountryController.java

package com.cognizant.springlearn.controller;

import com.cognizant.springlearn.model.Country; import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import javax.validation.Valid;

@RestController

public class CountryController {

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

@PostMapping("/countries")

public Country addCountry(@RequestBody @Valid Country country) { LOGGER.info("Inside CountryController.addCountry()");

return country;

}

}

# GlobalExceptionHandler.java

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;

@ControllerAdvice

public 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 - GlobalExceptionHandler");

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 - GlobalExceptionHandler");

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

}

}

# output

HTTP/1.1 400

Content-Type: application/json Transfer-Encoding: chunked

Date: Sat, 12 Jul 2025 09:46:00 GMT

Connection: close

{

"timestamp": "2025-07-12T09:46:00.123+0000",

"status": 400,

"errors": [

"Country code should be 2 characters"

]

}

Implement REST service for updating an employee

Employee.java

import javax.validation.constraints.\*;

import com.fasterxml.jackson.annotation.JsonFormat;

public class Employee {

@NotNull(message = "Employee ID must not be null") private Integer id;

@NotBlank(message = "Name must not be blank")

@Size(min = 1, max = 30, message = "Name must be between 1 and 30 characters") private String name;

@NotNull(message = "Salary must not be null") @Min(value = 0, message = "Salary must be 0 or above") private Double salary;

@NotNull(message = "Permanent status must not be null") private Boolean permanent;

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

@Valid

private Department department;

@Valid

private List<Skill> skills;

}

# Department.java

public class Department {

@NotNull(message = "Department ID must not be null") private Integer id;

@NotBlank(message = "Department name must not be blank")

@Size(min = 1, max = 30, message = "Department name must be between 1 and 30 characters")

private String name;

}

# Skill.java

public class Skill {

@NotNull(message = "Skill ID must not be null") private Integer id;

@NotBlank(message = "Skill name must not be blank")

@Size(min = 1, max = 30, message = "Skill name must be between 1 and 30 characters") private String name;

}

# EmployeeDao.java

public class EmployeeDao {

private static List<Employee> employeeList = new ArrayList<>();

public void updateEmployee(Employee updatedEmployee) throws EmployeeNotFoundException {

for (int i = 0; i < employeeList.size(); i++) {

if (employeeList.get(i).getId().equals(updatedEmployee.getId())) { employeeList.set(i, updatedEmployee);

return;

}

}

throw new EmployeeNotFoundException("Employee with ID " + updatedEmployee.getId()

+ " not found");

}

public List<Employee> getAllEmployees() { return employeeList;

}

}

# EmployeeService.java

@Service

public class EmployeeService { @Autowired

private EmployeeDao employeeDao;

public void updateEmployee(Employee employee) throws EmployeeNotFoundException { employeeDao.updateEmployee(employee);

}

public List<Employee> getAllEmployees() { return employeeDao.getAllEmployees();

}

}

# EmployeeController.java

@RestController

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

@Autowired

private EmployeeService employeeService;

@PutMapping

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

employeeService.updateEmployee(employee);

}

@GetMapping

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

}

# MockMvc Test

@WebMvcTest(EmployeeController.class) public class EmployeeControllerTest {

@Autowired

private MockMvc mockMvc;

@Test

public void testInvalidIdShouldReturnBadRequest() throws Exception {

String invalidJson = "{ \"id\": \"abc\", \"name\": \"Test\", \"salary\": 10000, \"permanent\": true, \"dateOfBirth\": \"01/01/2000\", \"department\": {\"id\": 1, \"name\": \"HR\"}, \"skills\": []

}";

mockMvc.perform(put("/employees")

.contentType(MediaType.APPLICATION\_JSON)

.content(invalidJson))

.andExpect(status().isBadRequest())

.andExpect(jsonPath("$.message").value("Incorrect format for field 'id'"));

}

}

# output

{

"timestamp": "2025-07-12T12:50:00.456+00:00",

"status": 400,

"error": "Bad Request",

"message": "Incorrect format for field 'id'"

}

## Implement REST DELETE Service

EmployeeDao.java

@Repository

public class EmployeeDao {

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

// Add this method

public void deleteEmployee(int id) throws EmployeeNotFoundException { boolean removed = EMPLOYEE\_LIST.removeIf(emp -> emp.getId() == id); if (!removed) {

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

}

}

// Other DAO methods like getAllEmployees, updateEmployee, etc.

}

EmployeeService.java

@Service

public class EmployeeService {

@Autowired

private EmployeeDao employeeDao;

// Add this method

public void deleteEmployee(int id) throws EmployeeNotFoundException { employeeDao.deleteEmployee(id);

}

// Other service methods

}

# EmployeeController.java

@RestController

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

@Autowired

private EmployeeService employeeService;

// DELETE endpoint @DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable int id) throws

EmployeeNotFoundException { employeeService.deleteEmployee(id);

return ResponseEntity.noContent().build(); // HTTP 204 No Content

}

// Other endpoints like GET, POST, PUT

}

# output

HTTP/1.1 404 Not Found

Content-Type: application/json

{

"timestamp": "2025-07-12T14:32:15.782+0000",

"status": 404,

"error": "Not Found",

"message": "Employee with ID 999 not found", "path": "/employees/999"

}