**Solution for Validating Multiple Controllers (e.g., EmployeeController)**

**Problem:**

**Repeating the same manual validation code in every controller (like CountryController and EmployeeController) leads to:**

* Code duplication
* Maintenance
* Inconsistent error handling

**What You Would Need to Do:**

java

// EmployeeController.java (BAD APPROACH - DUPLICATED CODE)

public Employee addEmployee(@RequestBody Employee employee) {

// 1. Create validator

ValidatorFactory factory = Validation.buildDefaultValidatorFactory();

Validator validator = factory.getValidator();

// 2. Validate manually (same as CountryController)

Set<ConstraintViolation<Employee>> violations = validator.validate(employee);

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

for (ConstraintViolation<Employee> violation : violations) {

errors.add(violation.getMessage());

}

// 3. Throw exception (same as CountryController)

if (!violations.isEmpty()) {

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

}

return employee; }

**Disadvantages of This Approach**

* Violates DRY Principle
* Identical validation code copied across controllers
* Changes require updates in every controller (e.g., modifying error formatting)
* High Maintenance Overhead
* Adding a new controller?Copy-paste validation boilerplate
* Changing validation logic? Update every controller manually
* Error Response Inconsistency

**Different controllers might format errors differently:**

java

// CountryController

throw new ResponseStatusException(..., errors.toString()); // [message1, message2]

// EmployeeController

throw new ResponseStatusException(..., "Errors: " + errors); // "Errors: [message1]"

**Testing Complexity**

* Each controller's validation must be tested separately
* Tests become repetitive and fragile
* Cluttered Controllers

**Business logic buried under validation boilerplate:**

java

@PostMapping("/employees")

public Employee addEmployee(...) {

// 15 lines of validation code

// 1 line of business logic

return employeeService.save(employee);

}

Real-World Consequences

Scenario: Change error format from List<String> to JSON object

json

// New required format:

{ "errors": ["Country code should be 2 characters"] }

**Impact:**

* Must update every controller manually
* Risk of missing some controllers
* Inconsistent responses during transition period

Before Change (CountryController):

HTTP 400: [message1, message2]

After Change (EmployeeController not updated):

HTTP 400: Errors: message1 // Inconsistent!