**Disadvantages of the Current Validation Approach**

While the current implementation using Java Bean Validation (JSR-380) ensures data integrity, it has several drawbacks when applied in scalable, enterprise-grade systems.

**1. Manual Validation in Controller Reduces Maintainability**

Embedding validation logic directly within the controller method violates the Separation of Concerns principle. The controller should delegate validation responsibilities to dedicated layers or use Spring’s declarative validation mechanism. This manual approach increases complexity and makes the codebase harder to maintain.

**Better approach**: Use Spring's @Valid annotation with @RequestBody to automatically trigger validation.

**2. Code Duplication Across Controllers**

If multiple controllers (e.g., CountryController, EmployeeController) require similar validations, the manual approach leads to code duplication. This redundancy increases the risk of inconsistencies and makes changes more error-prone.

**Better approach**: Reuse validation logic via DTOs and shared constraint annotations.

**3. Lack of Integration with Spring’s Validation and Exception Handling**

Spring Boot offers built-in support for validation via @Valid and global error handling through @ControllerAdvice and @ExceptionHandler. Ignoring these mechanisms means:

* Error responses are not standardized
* More boilerplate code is required to manually handle and format validation messages

**Better approach**: Let Spring handle validation errors and respond with consistent error messages using centralized exception handling.

**4. Inefficient Validator Initialization**

Creating a new ValidatorFactory for every request (Validation.buildDefaultValidatorFactory()) introduces unnecessary performance overhead. This object is meant to be reused and should not be created per request.

**Better approach**: Define the validator as a Spring-managed bean to initialize it once and inject wherever needed.

**5. Limited Support for Complex Validation Scenarios**

The current setup only supports simple field-level validation. It does not easily support:

* Cross-field validations (e.g., date range consistency)
* Conditional logic
* Reusable custom validation annotations

**Better approach**: Implement custom validators and annotations for complex business rules and leverage Spring’s extensibility features.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |