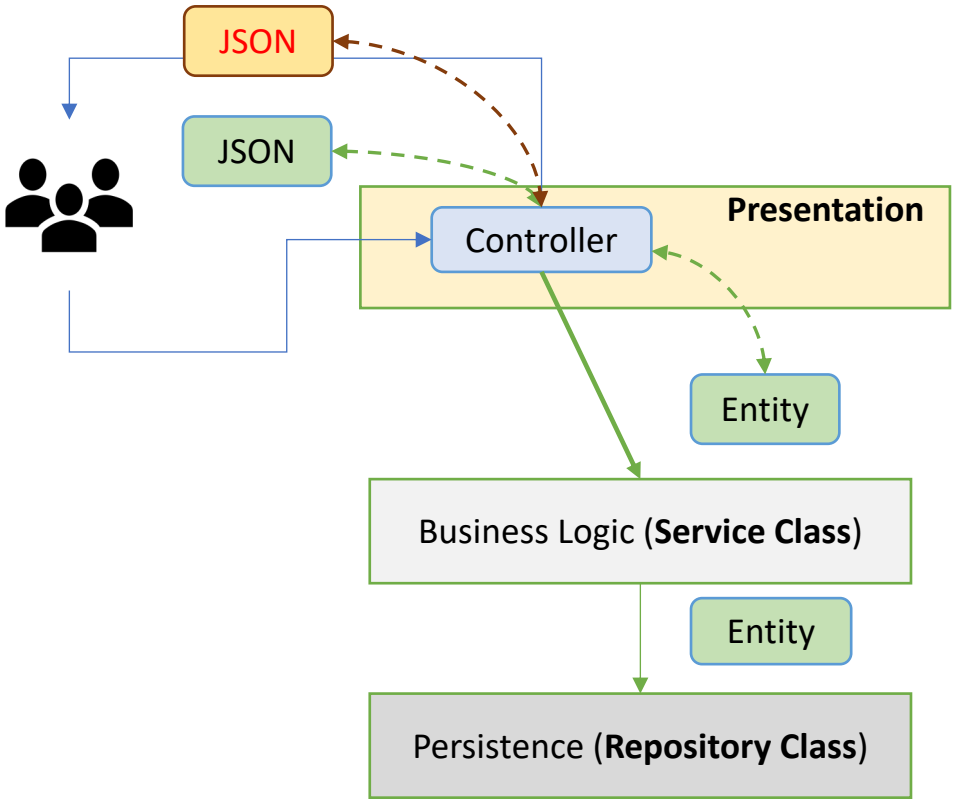
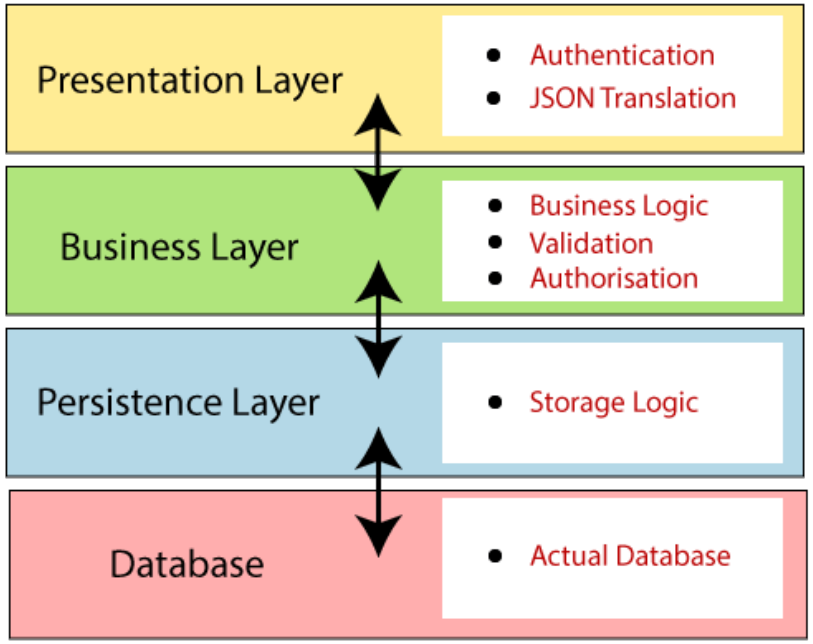
Spring RESTful API
Bean Validation Basics

By

Pichet Limvajiranan

Spring Boot Layer Architectures





Handling validation errors in the response.

```java
@Data
@RequiredArgsConstructor
@JsonInclude(JsonInclude.Include.NON_NULL)
public class ErrorResponse {
  private final int status;
  private final String message;
  private final String instance;
  private String stackTrace;
  private List<ValidationError> errors;

  public void addValidationError(String field, String message) {
    if (Objects.isNull(errors)) {
      errors = new ArrayList<>();
    }
    errors.add(new ValidationError(field, message));
  }
}
```

```java
@Data
@RequiredArgsConstructor
private static class ValidationError {
   private final String field;
   private final String message;
}
```

@RestControllerAdvice (1)

```
@RestControllerAdvice
public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {

    @ExceptionHandler(ItemNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ResponseEntity<ErrorResponse> handleItemNotFoundException(
            ItemNotFoundException exception, WebRequest request) {
        return buildErrorResponse(exception, HttpStatus.NOT_FOUND, request);
    }
```

@ControllerAdvice (2)

```
@ExceptionHandler(MethodArgumentNotValidException.class)
@ResponseStatus(HttpStatus.UNPROCESSABLE_ENTITY)
public ResponseEntity<ErrorResponse> handleMethodArgumentNotValid(
      MethodArgumentNotValidException ex,
      WebRequest request
) {
   ErrorResponse errorResponse = new ErrorResponse(
         HttpStatus.UNPROCESSABLE_ENTITY.value(),
         "Validation error. Check 'errors' field for details.“, request.getDescription(false)
   );

   for (FieldError fieldError : ex.getBindingResult().getFieldErrors()) {
      errorResponse.addValidationError(fieldError.getField(),
            fieldError.getDefaultMessage());
   }
   return ResponseEntity.unprocessableEntity().body(errorResponse);
}
```

Java Bean Validation

- Bean Validation provides a common way of validation through constraint declaration and metadata for Java applications.
- Using by annotate domain model properties with declarative validation constraints which are then enforced by the runtime.
- There are built-in constraints, and you can also define your own custom constraints.
- Example:

```
@Table(name = "offices")
public class Office {
    @Id
    @Column(name = "officeCode", nullable = false, length = 10)
    @NotBlank
    private String officeCode;

    @Column(name = "city", nullable = false, length = 15)
    @Size(min=5,max = 15)
    @NotBlank
    private String city;
```

```
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Size;
```

Common Validation Annotations

Some of the most common validation annotations are:

- **@NotNull** - validates that the annotated field is not null
- **@NotEmpty** - validates that the annotated field is not null or empty
- **@Min** - validates that the annotated field has a value no smaller than the value attribute
- **@Max** - validates that the annotated field has a value no greater than the value attribute
- **@Email** - validates that the annotated field is a valid email address
- **@Past** - validates that the date field is in the past
- **@PastOrPresent** - validates that the date field is in the past
- **@Pattern** - validates that the string field is matches a certain regular expression
- **@Email -** validates that the string field must be a valid email address.

Validator

- To validate if an object is valid, we pass it into a Validator which checks if the constraints are satisfied:

```
@Service
 public class OfficeService {
   @Autowired
   private Validator validator;

   public Office createNewOffice(Office office) {
      Errors errors = validator.validateObject(office);
      if (errors.hasErrors()) {
         throw new RuntimeException(errors.toString());
      }
      return repository.save(office);
   }
```

@Validation & @Valid

- We can use @Validation and @Valid annotations to let Spring know that we want to have a certain object validated.
- The @Validated annotation is a **class-level** annotation that we can use to tell Spring to validate parameters that are passed into a method of the annotated class.
- We can put the @Valid annotation **on method parameters and fields** to tell Spring that we want a method parameter or field to be validated.

Validating Input to a Spring REST Controller

- There are three things we can validate for any incoming HTTP request:
  - The request body
    - In POST and PUT requests, it’s common to pass a JSON payload within the request body. Spring automatically maps the incoming JSON to a Java object. Now, we want to check if the incoming Java object meets our requirements.
  - Variables within the path (e.g. id in /foos/{id}) and Query parameters
    - We’re not validating complex Java objects in this case, since path variables and request parameters are primitive types like int or their counterpart objects like Integer or String.

Validating a Request Body

```
• @Data
  public class NewCustomerDto {
      @NotNull
      @Min(900)
      private Integer id;
      @NotEmpty
      @Size(min=5, max = 50)
      private String customerName;
      @Size(min=3, max = 50)
      private String contactFirstName;
      @Size(min=3, max = 50)
      private String contactLastName;
      @Pattern(regexp = "^\\s*(?:\\+?(\\d{1,3}))?[-. (]*(\\d{3})[-. )]*(\\d{3})[-.
  ]*(\\d{4})(?: *x(\\d+))?\\s*$")
      private String phone;
```

^\s*(?:\+?(\d{1,3}))?[-. (]*(\d{3})[-.)]*(\d{3})[-.]*(\d{4})(?: *x(\d+))?\s*$

Expression	Description
^\s*	#Line start, match any whitespaces at the beginning if any.
(?:\+?(\d{1,3}))?	#GROUP 1: The country code. Optional. [-. (]* #Allow certain non numeric characters that may appear between the Country Code and the Area Code.
(\d{3})	#GROUP 2: The Area Code. Required. [-.)]* #Allow certain non numeric characters that may appear between the Area Code and the Exchange number.
(\d{3})	#GROUP 3: The Exchange number. Required. [-.]* #Allow certain non numeric characters that may appear between the Exchange number and the Subscriber number.
(\d{4})	#Group 4: The Subscriber Number. Required.
(?: *x(\d+))?	#Group 5: The Extension number. Optional. \s*$ #Match any ending whitespaces if any and the end of string.

Group1: Country Code (ex: 1 or 86)
Group2: Area Code (ex: 800)
Group3: Exchange (ex: 555)
Group4: Subscriber Number (ex: 1234)
Group5: Extension (ex: 5678)

18005551234 1
800 555 1234
+1 800 555-1234
+86 800 555 1234

1-800-555-1234 1
(800) 555-1234
(800)555-1234
(800) 555-1234

Validating a Request Body

- To validate the request body of an incoming HTTP request, we annotate the request body with the @Valid annotation in a REST controller:

```
@RestController
@RequestMapping("/customers")
public class CustomerControlller {
    :
    :
    @PostMapping("")
    public NewCustomerDto createCustomer(@Valid @RequestBody NewCustomerDto newCustomer) {
        return service.createCustomer(newCustomer);
    }
}
```

- If the validation fails, it will trigger a MethodArgumentNotValidException.
- By default, Spring will translate this exception to a HTTP status 400 (Bad Request).

Exercise: Validate Customer

1. Add Dependency to project.

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

2. Create Customer Entity (Customer.java), Customer DTO (NewCustomerDto.java & Employee DTO (SimpleEmployeeDto.java)
3. Create Customer service
4. Create Customer controller

** You can gets 2-4 from MS-Teams **

Customer Service

```
• @Service
  public class CustomerService {
      :
      :
     public NewCustomerDto createCustomer(NewCustomerDto newCustomer) {
        if(repository.existsById(newCustomer.getId())){
           throw new ResponseStatusException(HttpStatus.CONFLICT, "Duplicate customer for id "+
                 newCustomer.getId());
        }
        Customer customer = mapper.map(newCustomer, Customer.class);
        return mapper.map(repository.saveAndFlush(customer), NewCustomerDto.class);
     }
     public List<NewCustomerDto> getAllCustomers() {
        return listMapper.mapList(repository.findAll(), NewCustomerDto.class, mapper);
     }
  }
```

NewCustomer & SimpleEmployee DTO

```java
@Data
public class SimpleEmployeeDto {
   private Integer employeeNumber;
   @JsonIgnore
   private String firstName;
   @JsonIgnore
   private String lastName;
   public String getName() {
      return firstName + ' '+ lastName;
   }
}
```

```java
@Data
public class NewCustomerDto {
   @NotNull
   @Min(900)
   private Integer id;
   @NotEmpty
   @Size(min=5, max = 50)
   private String customerName;
   @Size(min=3, max = 50)
   private String contactFirstName;
   @Size(min=3, max = 50)
   :
   @NotNull
   private SimpleEmployeeDto sales;
   @Min(0)   @Max(10000)
   @NotNull(message = "Credit Limit Must be >=0 and <=10,000")
   private Double creditLimit;
}
```

Customer Controller

```
• @RestController
  @RequestMapping("/customers")
  public class CustomerControlller {
      @Autowired
      CustomerService service;

      @GetMapping
      public List<NewCustomerDto> getCustomers() {
          return service.getAllCustomers();
      }
      @PostMapping("")
      public NewCustomerDto createCustomer(
        @Valid @RequestBody NewCustomerDto newCustomer) {
•
          return service.createCustomer(newCustomer);
      }
  }
```

Test Data

```
{
"id": null,
"customerName": "SI",
"contactFirstName": "Khaitong",
"contactLastName": "Lim",
"phone": "0861681110",
"addressLine1": "54, rue Royale",
"addressLine2": null,
"city": "Nantes",
"state": null,
"postalCode": "44000",
"country": "France",
"sales" : {
    "employeeNumber": 1370
},
"creditLimit": 9000
}
```

```
{
"id": 201,
"customerName": "SI",
"contactFirstName": "Khaitong",
"contactLastName": "Lim",
"phone": "0861681110",
"addressLine1": "54, rue Royale",
"addressLine2": null,
"city": "Nantes",
"state": null,
"postalCode": "44000",
"country": "France",
"sales" : {
    "employeeNumber": 1370
},
"creditLimit": 9000
}
```

```
{
"id": 901,
"customerName": "SIT Vintage Shop",
"contactFirstName": "Khaitong",
"contactLastName": "Lim",
"phone": "0861681110",
"addressLine1": "54, rue Royale",
"addressLine2": null,
"city": "Nantes",
"state": null,
"postalCode": "44000",
"country": "France",
"sales" : {
    "employeeNumber": 1370
},
"creditLimit": 9000
}
```

Validating Input to a Spring Service Method

- Instead of (or additionally to) validating input on the controller level, we can also validate the input to any Spring components. In order to to this, we use a combination of the @Validated and @Valid annotations:

```
@Service
@Validated
class ValidatingService{

    void validateCustomer(@Valid Customer customer){
        // do something
    }
}
```

Validating Path Variables and Request Parameters

- Instead of annotating a class field like above, we're adding a constraint annotation (in this case @Min) directly to the method parameter in the Spring controller
- In contrast to request body validation a failed validation will trigger a HandlerMethodValidationException instead of a MethodArgumentNotValidException.

```
@GetMapping("")
  public ResponseEntity<Object> getAllProducts(
        @RequestParam(defaultValue = "") String partOfProductName,
        @RequestParam(defaultValue = "0") Double lower,
        @RequestParam(defaultValue = "0") Double upper,
        @RequestParam(defaultValue = "") String sortBy,
        @RequestParam(defaultValue = "ASC") String sortDirection,
        @RequestParam(defaultValue = "0") @Min(0) int pageNo,
        @RequestParam(defaultValue = "10") @Min(10) int pageSize
  ) {
            // your existing code
}
```

Add Handler method to Controller Advice

```
@ExceptionHandler(HandlerMethodValidationException.class)
  public ResponseEntity<ErrorResponse> handleHandlerMethodValidationException (
      HandlerMethodValidationException exception, WebRequest request) {

    ErrorResponse errorResponse = new ErrorResponse("Invalid parameter(s)",
            HttpStatus.UNPROCESSABLE_ENTITY.value(),
            "Validation error. Check 'errors' field for details.", request.getDescription(false));

     List<ParameterValidationResult> paramNames = exception.getAllValidationResults();
     for (ParameterValidationResult param : paramNames) {
        errorResponse.addValidationError(
                param.getMethodParameter().getParameterName(),
                param.getResolvableErrors().get(0).getDefaultMessage() + " ("+
                param.getArgument().toString()+ ")");
     }
     return ResponseEntity.unprocessableEntity().body(errorResponse);
  }
```