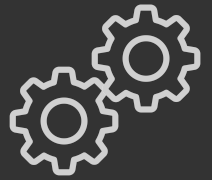


Testing

// Validaciones

IT BOARDING

BOOTCAMP

Índice

IT BOARDING

BOOTCAMP

Validaciones

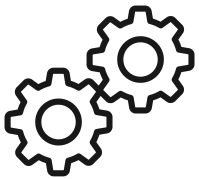
IT BOARDING

BOOTCAMP

Validaciones

La validación de los inputs del usuario es un requerimiento bastante común en la mayoría de las aplicaciones. Puede realizarse manualmente con una concatenación infinita de bloques if o podemos aprovechar las facilidades que nos ofrecen las diferentes librerías y frameworks disponibles para el lenguaje Java.

El framework Java Bean Validation se ha convertido en uno de los estándares más utilizados para implementar esta lógica.

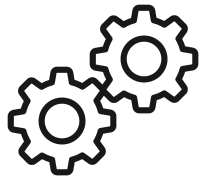


Validaciones

Para validar un bean en Java podemos utilizar el framework standard JSR 380, también conocido como **Bean Validation 2.0**. JSR 380, es una especificación de la API Java para validación de beans, es parte de Jakarta EE y JavaSE. Nos asegura que las propiedades de un bean cumplan con criterios específicos, usando diferentes anotaciones. Esta versión requiere Java 8 en adelante.

Para hacer uso de validation-api debemos agregar la siguiente dependencia en nuestro proyecto:

```
<dependency>
     <groupId>javax.validation</groupId>
     <artifactId>validation-api</artifactId>
     <version>2.0.1.Final</version>
</dependency>
```

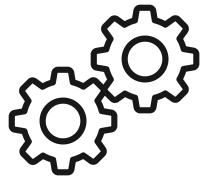


Validaciones

Hibernate Validator es la referencia de implementación de validation API. Para utilizarla debemos agregar la siguiente dependencia:

```
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
```

hibernate-validator se encuentra totalmente separado de los aspectos de persistencia de hibernate. Por lo que, al agregar esta dependencia NO estamos agregando esos aspectos de persistencia a nuestro proyecto.

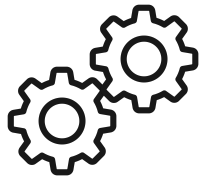


Validaciones

JSR 380 permite expresiones dentro de mensajes donde ocurre un incumplimiento de validaciones.

Para parsear estas expresiones, se agrega la dependencia javax.el de GlassFish, esta contiene una implementación de la especificación “Expression Language”:

```
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.el</artifactId>
    <version>3.0.0</version>
</dependency>
```



Validaciones

En resumen... para activar estas validaciones se utilizarán las siguientes dependencias, las cuales se agregan al pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-validation</artifactId>
</dependency>
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.0.13.Final</version>
</dependency>
<dependency>
    <groupId>org.glassfish</groupId>
    <artifactId>javax.el</artifactId>
    <version>3.0.0</version>
</dependency>
```

TESTING

Anotaciones para Validaciones

IT BOARDING

BOOTCAMP

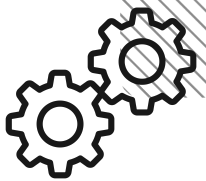
Validar Beans (DTO)

```java
@RestController
public class ControladorUsuario {

    @PostMapping("/usuario")
    ResponseEntity<String> agregarUsuario(@Valid @RequestBody Usuario usuario) {
        // persistiendo el usuario
        return ResponseEntity.ok("El usuario es válido");
    }

}
```

```java
public class Usuario {
    //...

    @Valid
    private DireccionUsuario direccionUsuario;

    //...
}
```

- Para indicarle a spring que valide un elemento, se utilizará la anotación **@Valid**, la que ejecutará la implementación predeterminada de JSR 380 (Hibernate Validator).
- Se podrá emplear tanto desde los parámetros de un método como en un atributo, para indicar la validación de otro bean (DTO) anidado.
- Esta anotación activa la validación **TOTAL** del elemento, es decir, validará en cascada todos los atributos del objeto señalado.

Validar Beans (DTO)



- Para validar los tipos de datos nativos dentro de un Bean (DTO) se utilizarán una serie de anotaciones, que se podrán colocar sobre cada atributo a validar, así como tambíen en los parámetros recibidos o de retorno de un método.

- Algunas anotaciones aceptan diferentes atributos, pero el atributo **message** es común a todas. Representa el mensaje que se va mostrar normalmente cuando la respectiva validación falle.

Validar Beans (DTO)

// Personalizar mensajes de validaciones

Capturando las excepciones **MethodArgumentNotValidException** y **HttpMessageNotReadableException** desde el controlador base para el manejo de excepciones (@ControllerAdvice, @ExceptionHandler) se podrá manejar y personalizar los mensajes devueltos por la falla de las validaciones de los beans (DTOs).

```
@ControllerAdvice
public class ExceptionController extends ResponseEntityExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    protected ResponseEntity<CustomException> handleValidationExceptions(MethodArgumentNotValidException ex) {
        //define custom exception and return correspondant HttpStatus
        return new ResponseEntity<>(customException, HttpStatus...);
    }

    @ExceptionHandler(HttpMessageNotReadableException.class)
    protected ResponseEntity<CustomException> handleValidationExceptions(HttpMessageNotReadableException ex) {
        CustomException customException = new CustomException("Custom message", ex.getMessage());
        return new ResponseEntity<>(customException, HttpStatus.BAD_REQUEST);
    }

}
```

// Anotaciones de validaciones

- ***@NotNull*** valida que el valor de la propiedad anotada no sea *null*.
- ***@AssertTrue*** valida que el valor de la la propiedad anotada sea *true*.
- ***@Size*** valida que el valor de la propiedad anotada tenga un tamaño entre los atributos *min* y *max*. Puede ser aplicado a *String*, *Collection*, *Map*, y propiedades de *Array*.
- ***@Min*** valida que el valor de la propiedad anotada tenga un valor mayor que el atributo *value*.
- ***@Max*** valida que el valor de la propiedad anotada tengan un valor menor que el atributo *value*.
- ***@Email*** valida que el valor de la propiedad anotada sea una dirección de email válida.

Anotaciones adicionales que se pueden encontrar en JSR:

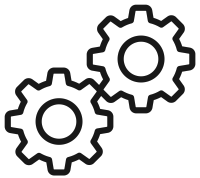
- ***@NotEmpty*** valida que la propiedad no sea nula o vacía; se puede aplicar a *String*, *Collection*, *Map* o valores de *Array*.
- ***@NotBlank*** se puede aplicar solamente a valores de texto y valida que la propiedad no sea nula o tenga espacios vacíos (whitespace).
- ***@Positive*** and ***@PositiveOrZero*** se aplica a valores numéricos y valida que sean estrictamente positivos, o positivos incluído 0.
- ***@Negative*** and ***@NegativeOrZero*** se aplica a valores numéricos y valida que sean estrictamente negativos, o negativos incluído 0.
- ***@Past*** and ***@PastOrPresent*** valida que el valor de fecha esté en el pasado o en el pasado incluyendo el presente. Se aplica a tipos de fecha (Date) incluyendo aquellos en Java 8.
- ***@Future*** *and* ***@FutureOrPresent*** valida que el valor de la fecha esté en el futuro, o en el futuro incluyendo el presente.

Las anotaciones de validaciones también pueden ser aplicadas a elementos de una collection:

List<**@NotBlank** String> preferences;

Anotaciones para validaciones

Aquí se agregan validaciones simples al bean Usuario

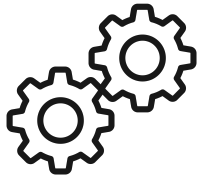


```java
public class Usuario {

    @NotNull(message = "Nombre no puede ser nulo")
    private String nombre;

    @AssertTrue
    private boolean trabaja;

    @Size(min = 10, max = 200, message
            = "Acerca de Mi debe tener entre 10 y 200 caracteres")
    private String acercaDeMi;

    @Min(value = 18, message = "Edad no debe ser menor que 18")
    @Max(value = 150, message = "Edad no debe ser mayor que 150")
    private int edad;

    @Email(message = "Email debe ser válido")
    private String email;

    // getters y setters
}
```

Jackson para manejo de JSON

fasterxml.jackson puede sernos muy útil para la serialización de objetos java en JSON y viceversa.

- **@JsonPropertyOrder** → Se utiliza para definir el orden que seguirá al serializar las propiedades del objeto.
- **@JsonSerialize** → Se utiliza para especificar un serializador personalizado para ordenar el objeto JSON.
- **@JsonIgnoreProperties** → Se utiliza a nivel de clase para marcar una propiedad o lista de propiedades que se ignorarán.
- **@JsonIgnore** → Se utiliza a nivel de atributo para marcar una propiedad o lista de propiedades que se ignorarán.
- **@JsonInclude** → Permite excluir propiedades con valores vacíos, null, o por defecto.
- **@JsonFormat** → Se utiliza para especificar el formato durante la serialización o deserialización. Se usa principalmente con campos de tipo Date.



Gracias

IT BOARDING

BOOTCAMP