VALIDACIONES

Mediante anotaciones de validación vamos a poder tener mayor control sobre los datos que manejamos desde la vista, ya que podremos personalizar el alta y modificación de datos y persistir de forma correcta. Spring Boot Starter Validation nos brinda una fuente que integra validadores personalizados, el cual se encuentra en el paquete jakarta.validation.constraints.

Lo primero que haremos será añadir en nuestro archivo pom.xml la dependencia de spring boot starter validation, para esto nos dirigimos a la página mvnrepository.com y copiamos la dependencia.

Para este ejemplo se utilizó la siguiente:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation -->
<dependency>

	<groupId>org.springframework.boot</groupId>

	<artifactId>spring-boot-starter-validation</artifactId>

</dependency>
```

ANOTACIONES DE VALIDACIÓN

Las anotaciones de validación pueden ser manejados tanto en los métodos de la capa controller como en atributos de clases. Aunque por lo general al validar todos los atributos de una clase lo apropiado es atribuir las anotaciones en la clase y utilizar la anotación *@Valid* para validar los datos.

Otra característica de las anotaciones, es que podemos agregar mensajes personalizados, que se mostrarán en el caso de que no se cumpla las condiciones de las mismas. Si no añadimos los mensajes personalizados se mostrará el mensaje de error predefinido por cada anotación de validación.

Algunas de las anotaciones son:

@Size: Determina la longitud de una cadena con las propiedades min y max.

@Min: Anotación para campo numérico que solo permite un valor mínimo especificado. Ejemplo: *@Min(4)* solo permite ingresar como valor mínimo 4. Si desea personalizar el mensaje sería de la siguiente forma *@Min(value=4, message= "Debe ingresar un número mayor o igual a 4 ")*.

@Max: Anotación para campo numérico que permite el ingreso hasta un valor máximo especificado. Ejemplo: *@Max(4)* solo permite ingresar como valor máximo el número 4.

@NotBlank: Anotación para campo tipo cadena que no permite un valor vacío, es decir que debe tener al menos un caracter. La diferencia con la anotación *@NotEmpty* es que *@NotBlank* se utiliza para campos tipo String.

@Email: Verifica que el valor sea un correo electrónico valido.

@NotEmpty: Al igual que *@NotBlank* verifica que el valor del atributo de la clase no sea vacío ni nulo. Esta anotación no es válida para atributos tipo fecha.

@NotNull: Anotación que no permite un campo con valor nulo.

@Past: Determina que el valor fecha ingresado sea una fecha pasada a la fecha actual.

@Future: Determina que el valor fecha ingresado sea una fecha futura a la fecha actual.

@Pattern: Determina que el valor cumpla con una expresión especificada. Ejemplo: *@Pattern(regexp="[a-z]*+/+[1-3]").*Esta expresión solicita que la cadena ingresada deba comenzar con un caracter o caracteres en minúscula del alfabeto seguido de la barra diagonal "/" y un número del 1 al 3. Si ingreso la cadena "ab2c/3", no lo tomará como valido ya que al comienzo contiene la expresión [a-z] y solo admite letras al comienzo y no valores numéricos.

Mas anotaciones:

javax.validation.constraints (API de especificación de Java(TM) EE 7) (oracle.com)

ANOTACIONES EN LAS CLASES

Ahora pasamos a modificar las clases con las anotaciones que consideremos necesarias para validar los atributos de las clases LIBRO y AUTOR.

```java
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
@Component
public class Autor {

	private int id;

	@NotBlank(message="Debe ingresar nombre del autor")
	@Size(min=3, max=20, message="El nombre debe contener como mínimo 3 caracteres y como máximo 20 caracteres")
	@Pattern(regexp= "[a-z A-Z]*", message="Debe ingresar únicamente letras")
	private String nombre;

	@NotBlank(message="Debe ingresar apellido del autor")
	@Size(min=2, max=40,message="El apellido no puede llevar menos de 2 caracteres y más de 30 caracteres")
	@Pattern(regexp= "[a-z A-Z]*", message="Debe ingresar únicamente letras")
	private String apellido;

	@NotBlank(message="Debe ingresar la nacionalidad")
	@Pattern(regexp= "[a-z A-Z]*", message="Debe ingresar únicamente letras")
	private String nacionalidad;
```

```java
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Past;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
@Component
public class Libro {

	private int id;

	@NotBlank(message="Debe ingresar el ISBN")
	@Size(min=13,max=16, message="El código ISBN debe contener como minimo 10 dígitos y como maximo 13 dígitos")
	@Pattern(regexp="[0-9]+-+[0-9]*+-+[0-9]*+-+[0-9]",message="Código ISBN incorrecto.  (Ej: 0-7645-2641-3)")
	private String isbn;

	@NotBlank(message="Debe ingresar título del libro")
	private String titulo;

	@NotBlank(message="Debe ingresar nombre de la editorial")
	@Pattern(regexp= "[a-z A-Z]*", message="Debe ingresar únicamente letras")
	private String editorial;

	@Past (message="La fecha de publicación debe ser anterior a la fecha actual")
	@NotNull(message="Debe ingresar la fecha de publicación")
	@DateTimeFormat(pattern = "yyyy-MM-dd")
	private LocalDate anioPublicacion;

	private String imagen;

	@Autowired
	@NotNull(message="Debe seleccionar un Autor")
	private Autor autor;
```

VALIDACIONES EN LA CAPA CONTROLLER

Luego de haber definido las validaciones necesarias en las clases, tenemos que validar en el controlador los datos que son enviados desde la vista del formulario. Es por esto que únicamente vamos a realizar modificaciones en el método que recibe la petición de guardar los datos de dicho formulario. Y suponiendo que los datos ingresados por el usuario son inválidos, debemos re direccionar nuevamente al formulario renderizado.

Controlador ***"LibroController"***

```
@PostMapping("/guardar")
	public ModelAndView postPageSaveLibro(@Valid @ModelAttribute("unLibro")
		Libro libro, BindingResult result, @RequestParam("file")
		MultipartFile image) throws Exception {
		 ModelAndView mav;
		if (result.hasErrors()) {
			mav= new ModelAndView("nuevoLibro");
			mav.addObject("autores", ListaAutores.getListaAutores());
		}else {
			unAutor=ListaAutores.findAutorById(libro.getAutor().getId());

			libro.setAutor(unAutor);
			if(libro.getId()>0) {
				unLibro= ListaLibros.findLibroById(libro.getId());
				if (!image.isEmpty()) {
					if(unLibro.getImagen() != null &&
					   unLibro.getImagen().length() > 0)
						uploadFile.delete(unLibro.getImagen());
					String uniqueFileName = uploadFile.copy(image);
					libro.setImagen(uniqueFileName);
				}else {
					if(unLibro.getImagen() != null)
						libro.setImagen(unLibro.getImagen());
				}
				ListaLibros.libros.set(libro.getId()-1, libro);
			}else {
				if (!image.isEmpty()) {
					String uniqueFileName = uploadFile.copy(image);
					libro.setImagen(uniqueFileName);
				}
				ListaLibros.addLibro(libro);
			}
		    mav= new ModelAndView("listaLibros");
		    mav.addObject("libros", ListaLibros.getListLibros());

		}
		return mav;
	}
```

Controlador **“*AutorController*”**

```
@PostMapping("/guardar")
	public ModelAndView postPageSaveAutor (@Valid  @ModelAttribute("unAutor")
Autor autor, BindingResult result) {
		ModelAndView mav;
		if(result.hasErrors()) {
			mav= new ModelAndView("nuevoAutor");

		}else {
			if(autor.getId()>0)
				ListaAutores.autores.set(autor.getId()-1, autor);
			else
			    ListaAutores.addAutor(autor);
		    mav= new ModelAndView("listaAutores");
		    mav.addObject("autores", ListaAutores.getListaAutores());
		}


		return mav;
	}
```

```
39    @PostMapping("/guardar")
40    public ModelAndView postPageSaveAutor (@Valid  @ModelAttribute("unAutor") Autor autor,
41          BindingResult result) {
42        ModelAndView mav;
43        if(result.hasErrors()) {
44            mav= new ModelAndView("nuevoAutor");
45        }else {
46            if(autor.getId()>0)
47                ListaAutores.autores.set(autor.getId()-1, autor);
48            else
49                ListaAutores.addAutor(autor);
50            mav= new ModelAndView("listaAutores");
51            mav.addObject("autores", ListaAutores.getListaAutores());
52        }
53
54        return mav;
55    }
56
```

Para controlar si los datos enviados desde el formulario son válidos se utiliza la anotación ***@Valid*** y se lo coloca en el parámetro del objeto a validar. Como su nombre lo indica, valida el argumento recibido. Además, se agrega otro parámetro de tipo ***BindingResult***.
BindingResult nos permite recuperar el resultado de las validaciones vinculadas en los campos de nuestro formulario.
Por ultimo agregamos una condición que controle si los datos enviados contienen errores. Como se puede ver en la línea 43, si los datos contienen errores se re direccionará nuevamente a la página del formulario “***nuevoAutor***”, de lo contrario guardará los datos.

VALIDACIONES EN EL FORMULARIO

Por último, nos queda agregar, en cada campo del formulario, las etiquetas y propiedades necesarias para mostrar los mensajes de errores de cada atributo de la clase, en el caso de que los datos no sean válidos.

Página ***"nuevoAutor.html"***

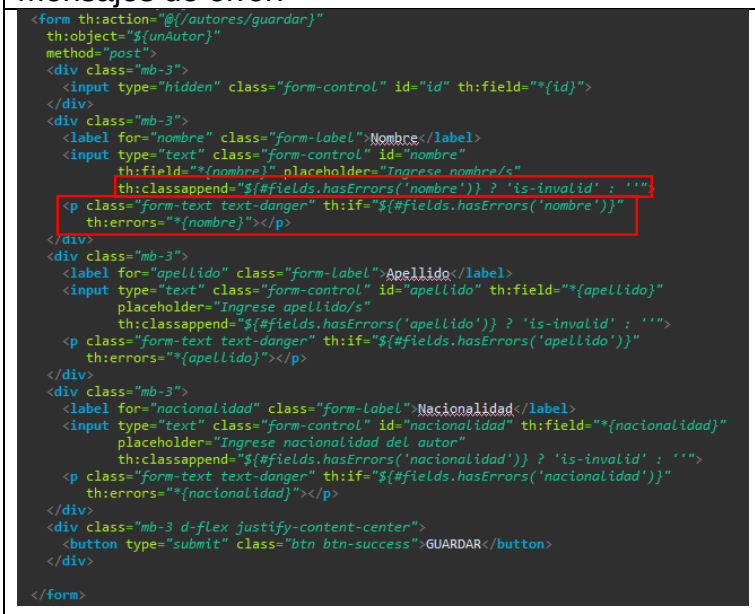
```
<form th:action="@{/autores/guardar}"
      th:object="${unAutor}"
      method="post">
      <div class="mb-3">
            <input type="hidden" class="form-control" id="id" th:field="*{id}">
            </div>
            <div class="mb-3">
                <label for="nombre" class="form-label">Nombre</label>
                <input type="text" class="form-control" id="nombre"
                   th:field="*{nombre}" placeholder="Ingrese nombre/s"
      th:classappend="${#fields.hasErrors('nombre')} ? 'is-invalid' : ''">
                <p class="form-text text-danger"
th:if="${#fields.hasErrors('nombre')}"  th:errors="*{nombre}"></p>
            </div>
            <div class="mb-3">
            <label for="apellido" class="form-label">Apellido</label>
            <input type="text" class="form-control" id="apellido"
th:field="*{apellido}" placeholder="Ingrese apellido/s"
th:classappend="${#fields.hasErrors('apellido')} ? 'is-invalid' : ''">
            <p class="form-text text-danger"
th:if="${#fields.hasErrors('apellido')}" th:errors="*{apellido}"></p>
            </div>
            <div class="mb-3">
            <label for="nacionalidad" class="form-label">Nacionalidad</label>
            <input type="text" class="form-control" id="nacionalidad"
th:field="*{nacionalidad}" placeholder="Ingrese nacionalidad del autor"

th:classappend="${#fields.hasErrors('nacionalidad')} ? 'is-invalid' : ''">
            <p class="form-text text-danger"
th:if="${#fields.hasErrors('nacionalidad')}"  th:errors="*{nacionalidad}"></p>
            </div>
           <div class="mb-3 d-flex justify-content-center">
            <button type="submit" class="btn btn-success">GUARDAR</button>
            </div>

</form>
```

Página ***"nuevoLibro.html"***

```
<form th:action="@{/libros/guardar}"
      th:object="${unLibro}"
      method="post"
      enctype="multipart/form-data">
     <div class="mb-3"><input type="hidden" class="form-control" id="id"
         th:field="*{id}" ></div>
      <div class="mb-3">
        <label for="titulo" class="form-label">Titulo</label>
        <input type="text" class="form-control" id="titulo"
              th:field="*{titulo}" placeholder="Titulo del libro"
       th:classappend="${#fields.hasErrors('titulo')} ? 'is-invalid' : ''" >
       <p class="form-text text-danger"
          th:if="${#fields.hasErrors('titulo')}" th:errors="*{titulo}"></p>
       </div>
```

```
        <div class="mb-3">
            <label for="isbn" class="form-label">ISBN</label>
            <input type="text" class="form-control" id="isbn" th:field="*{isbn}"
                   placeholder="Ingrese código ISBN del libro" maxlength="16"
oninput="if(this.value.length > this.maxLength) this.value = this.value.slice(0,
this.maxLength);"
th:classappend="${#fields.hasErrors('isbn')} ? 'is-invalid' : ''">
            <p class="form-text text-danger" th:if="${#fields.hasErrors('isbn')}"
               th:errors="*{isbn}"></p>
        </div>
        <div class="mb-3">
            <label for="autor" class="form-label">Autor</label>
            <select class="form-select" id="autor" th:field="*{autor.id}"
                    name="autor"
          th:classappend="${#fields.hasErrors('autor')}? 'is-invalid' : ''">
              <option value="0" disabled selected>SELECCIONAR</option>
              <option th:each="a: ${autores}"th:value="${a.id}"
              th:text="${a.nombre}+'-'+${a.apellido}"></option></select>
              <p class="form-text text-danger"
             th:if="${#fields.hasErrors('autor')}" th:errors="*{autor}"></p>
        </div>
        <div class="mb-3">
            <label for="editorial" class="form-label">Editorial</label>
            <input type="text" class="form-control" id="editorial"
                   th:field="*{editorial}" placeholder="Nombre de la Editorial"
          th:classappend="${#fields.hasErrors('editorial')} ? 'is-invalid' : ''">
            <p class="form-text text-danger"
          th:if="${#fields.hasErrors('editorial')}" th:errors="*{editorial}"></p>
        </div>
        <div class="mb-3">
            <label for="anioPublicacion" class="form-label">Año de
                   Publicación</label>
            <input type="date" class="form-control" id="anioPublicacion"
              th:field="*{anioPublicacion}"
              th:classappend="${#fields.hasErrors('anioPublicacion')} ? 'is-
              invalid' : ''">
              <p class="form-text text-danger"
               th:if="${#fields.hasErrors('anioPublicacion')}"
               th:errors="*{anioPublicacion}"></p>
        </div>
        <div class="mb-3">
             <label for="file" class=form-label>Portada del libro</label>
             <input type="file" name="file" id="file" class="form-control"
              th:text="${unLibro.imagen}"/> <br>
        </div>
        <div class="mb-3 d-flex justify-content-center">
              <button type="submit" class="btn btn-success">GUARDAR</button>
        </div>
</form>
```

Cambios agregados en el formulario de la página “nuevoAutor.html” para los mensajes de error.

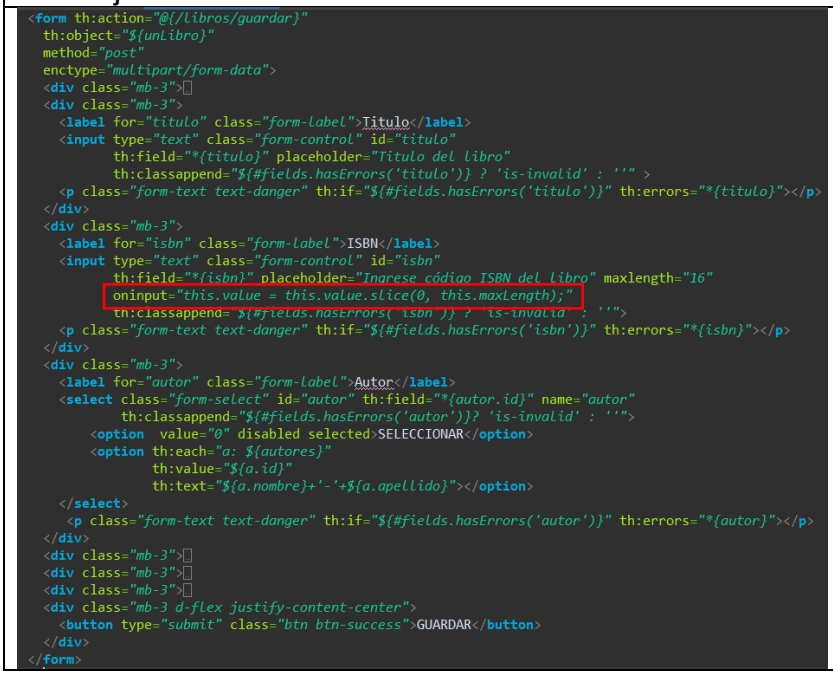


```
<form th:action="@{/autores/guardar}"
  th:object="${unAutor}"
  method="post">
  <div class="mb-3">
    <input type="hidden" class="form-control" id="id" th:field="*{id}">
  </div>
  <div class="mb-3">
    <label for="nombre" class="form-label">Nombre</label>
    <input type="text" class="form-control" id="nombre"
           th:field="*{nombre}" placeholder="Ingrese nombre/s"
           th:classappend="${#fields.hasErrors('nombre')} ? 'is-invalid' : ''">
    <p class="form-text text-danger" th:if="${#fields.hasErrors('nombre')}"
       th:errors="*{nombre}"></p>
  </div>
  <div class="mb-3">
    <label for="apellido" class="form-label">Apellido</label>
    <input type="text" class="form-control" id="apellido" th:field="*{apellido}"
           placeholder="Ingrese apellido/s"
           th:classappend="${#fields.hasErrors('apellido')} ? 'is-invalid' : ''">
    <p class="form-text text-danger" th:if="${#fields.hasErrors('apellido')}"
       th:errors="*{apellido}"></p>
  </div>
  <div class="mb-3">
    <label for="nacionalidad" class="form-label">Nacionalidad</label>
    <input type="text" class="form-control" id="nacionalidad" th:field="*{nacionalidad}"
           placeholder="Ingrese nacionalidad del autor"
           th:classappend="${#fields.hasErrors('nacionalidad')} ? 'is-invalid' : ''">
    <p class="form-text text-danger" th:if="${#fields.hasErrors('nacionalidad')}"
       th:errors="*{nacionalidad}"></p>
  </div>
  <div class="mb-3 d-flex justify-content-center">
    <button type="submit" class="btn btn-success">GUARDAR</button>
  </div>

</form>
```

th:classappend: este atributo de thymeleaf permite agregar una clase a la propiedad *class* ya determinado en un elemento. De acuerdo a una expresión o condición podemos agregar una clase a la clase ya existente, es por esto que mediante la expresión ***#fields.hasErrors(‘titulo’)?*** consultamos si el usuario introdujo datos inválidos, si es verdadero agrega la clase dinámica ***‘is-invalid’***(esta clase es un estilo para campos invalido establecido por Bootstrap).

Luego agregamos una etiqueta para mostrar los errores del campo. Para el ejemplo utilizamos la etiqueta de párrafo**<p>,** que aparecerá únicamente cuando los datos ingresados en el atributo sean inválidos. Esto sucede debido a la condición ***th:if*** que permitirá mostrar los errores de dicho campo del formulario.

Cambios agregados en el formulario de la página “nuevoLibro.html” para los mensajes de error.



```
<form th:action="@{/libros/guardar}"
  th:object="${unLibro}"
  method="post"
  enctype="multipart/form-data">
  <div class="mb-3">
  <div class="mb-3">
    <label for="titulo" class="form-label">Titulo</label>
    <input type="text" class="form-control" id="titulo"
           th:field="*{titulo}" placeholder="Titulo del libro"
           th:classappend="${#fields.hasErrors('titulo')} ? 'is-invalid' : ''" >
    <p class="form-text text-danger" th:if="${#fields.hasErrors('titulo')}" th:errors="*{titulo}"></p>
  </div>
  <div class="mb-3">
    <label for="isbn" class="form-label">ISBN</label>
    <input type="text" class="form-control" id="isbn"
           th:field="*{isbn}" placeholder="Ingrese código ISBN del libro" maxlength="16"
           oninput="this.value = this.value.slice(0, this.maxLength);"
           th:classappend="${#fields.hasErrors('isbn')} ? 'is-invalid' : ''">
    <p class="form-text text-danger" th:if="${#fields.hasErrors('isbn')}" th:errors="*{isbn}"></p>
  </div>
  <div class="mb-3">
    <label for="autor" class="form-label">Autor</label>
    <select class="form-select" id="autor" th:field="*{autor.id}" name="autor"
            th:classappend="${#fields.hasErrors('autor')}? 'is-invalid' : ''">
        <option  value="0" disabled selected>SELECCIONAR</option>
        <option th:each="a: ${autores}"
                th:value="${a.id}"
                th:text="${a.nombre}+'-'+${a.apellido}"></option>
    </select>
     <p class="form-text text-danger" th:if="${#fields.hasErrors('autor')}" th:errors="*{autor}"></p>
  </div>
  <div class="mb-3">
  <div class="mb-3">
  <div class="mb-3">
  <div class="mb-3 d-flex justify-content-center">
    <button type="submit" class="btn btn-success">GUARDAR</button>
  </div>
</form>
```

Para el formulario de libro agregamos un control para determinar el tamaño de la cadena de caracteres que se pueden ingresar, esto lo hacemos con el evento ***oninput***. Este evento se activa cuando el valor de input cambia, es decir cuando recibe la entrada del usuario. Y este evento permite que el valor que se ingrese en el campo tenga como máximo 16 caracteres.

Ejemplo de cómo quedaría los formularios con las validaciones de Spring Boot.