AJUSTES FUNCIONALES Y DE PRESENTACION EN SPRING BOOT.

Para agregar una mejor funcionalidad y presentación a nuestras páginas html vamos a realizar tres acciones, la primera nos permite validar los datos capturados en el formulario, la segunda es usar una plantilla para facilitar la construcción de las páginas del proyecto reutilizando código escrito en un fragmento de página y por último usaremos agregaremos la librería de Bootstrap a nuestro proyecto.

1. Validaciones en el formulario.

   Gracias a spring boot se puede hacer una validación de los datos escritos sobre un formulario, para poder realizarlo se debe agregar la dependencia de Maven que gestiona la validación de JPA, se ajusta la clase del modelo de datos con las anotaciones que definen la validación.

2. Uso de Plantilla con Thymeleaf.

   Por medio de una plantilla podemos ahorrar tiempo en la escritura de nuestro programa creando un fragmento de código con thymeleaf que se puede reutilizar en las páginas que lo requieren.

3. Agregar Bootstrap a nuestro proyecto.

   Bootstrap nos permite mejorar la presentación gracias a la estructura de css y javascript definidos por la librería, podemos hacerlo creando nuestros propios estilos o haciendo uso de las librerías ya disponibles en el mercado como WebJars.

1. Validaciones en el formulario.

Para hacer validaciones en el formulario debemos hacer los siguientes pasos:

- Agregar la dependencia al Maven que gestione las validaciones con Hibernate.

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>5.1.3.Final</version>
</dependency>
```

- Agregar a la clase las anotaciones dentro de cada campo en el modelo de datos.

```
@Id
@NotNull
private int codigo;

@NotEmpty
@Size(min = 3, max = 50)
private String nombre;

@Min(value = 1)
private int nitproveedor;

private double precio_compra;

@Min(value = 0)
@Max(value = 35)
private double iva;
private double precio_venta;
```

Se deben agregar las librerías necesarias para las validaciones como lo que se aprecia en la figura:

```
8   import javax.validation.constraints.Max;
9   import javax.validation.constraints.Min;
    import javax.validation.constraints.NotBlank;
11  import javax.validation.constraints.NotEmpty;
12  import javax.validation.constraints.NotNull;
13  import javax.validation.constraints.Size;
```

Se muestran algunas restricciones definidas por la API de validación de Java Bean:
@NotEmpty: especifica que el campo anotado no debe ser nulo o vacío.
@NotBlank: el campo anotado no debe ser nulo y debe contener al menos un carácter que no sea un espacio en blanco.
@NotNull: el campo anotado no debe ser nulo.
@Email: la cadena debe ser una dirección de correo electrónico bien formada.
@Min: el campo debe ser un número cuyo valor debe ser mayor o igual al valor mínimo especificado.
@Max: el campo debe ser un número cuyo valor debe ser menor o igual al valor máximo especificado.
@Size: el tamaño del campo debe estar dentro del rango especificado.

Aplicado para cadenas, colecciones y matrices.

Las anotaciones provienen del paquete javax.validation.constraints .
Y gracias a Spring Boot, que aplica la configuración automática, no tenemos que configurar nada para usar la API de validación de Java Bean. Se incluye automáticamente en la dependencia spring-boot-starter-web, pero debimos agregar la dependencia de validación de hibernate.

- Modificar en el formulario cada campo que permita validar el resultado.

```
<div>
    <label>Nombre de Producto:</label>
    <input type="text" th:field="*{nombre}" >
    <span th:if="${#fields.hasErrors('nombre')}" th:errors="*{nombre}">Error de Nombre</span>
</div>
<div>
    <label>Nit Proveedor:</label>
    <input type="text" th:field="*{nitproveedor}" >
    <span th:if="${#fields.hasErrors('nitproveedor')}" th:errors="*{nitproveedor}">Error de Nit Proveedor</span>
</div>
<div>
    <label>Precio de Compra:</label>
    <input type="text" th:field="*{precio_compra}" >
</div>
<div>
    <label>Iva:</label>
    <input type="text" th:field="*{iva}" >
    <span th:if="${#fields.hasErrors('iva')}" th:errors="*{iva}">Error de Iva</span>
</div>
```

- Modificar el controlador.

Ajustamos el método que permite crear un producto, cambiando el tipo de validación con @Valid y agregamos la gestión de errores con la clase "Errors", agregamos una validación de la presencia de errores que nos permita dirigir la salida.

```
@PostMapping("/save")
public String saveProducto(@Valid @ModelAttribute("producto") Productos producto, Errors errores, Model model) {
    if(errores.hasErrors()){
        return "nuevo_producto";
    }else{
        servicio.save(producto);
        return "redirect:/";
    }
}
```

2. Uso de Plantilla con Thymeleaf.

Para poder aprovechar al máximo el beneficio de las plantillas se recomienda llevar a cabo el siguiente procedimiento:

1. Creación de carpeta de plantillas.
2. Creación de plantilla.
3. Agregar a la página deseada.

1. Creación de carpeta de plantillas.

   En la carpeta templates creamos un folder llamado layout.

2. Creación de plantilla.

   Sobre el folder creamos una página html llamada plantilla con el siguiente contenido:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
    <head>
        <title>Plantilla</title>
        <meta charset="UTF-8"/>
    </head>
    <body>
        <header th:fragment="header">
            <h1>Encabezado de Páginas</h1>
        </header>

        <footer th:fragment="footer">
            <p>Derechos Reservados <a href="https://www.unbosque.edu.co/" target="_blank">Universidad El Bosque</a></p>
        </footer>

    </body>
</html>
```

Se agrega el xmlns que permite usar la plantilla de thymeleaf y definimos con th:fragment los pedazos de código que se desean reutilizar en cada página, en este caso se define un encabezado (header) y un pie de página (footer).

3. Agregar a la página deseada.

   En cada página que se necesite se agregan las secciones de las plantillas necesarias usando la etiqueta "replace", en este caso lo haremos sobre la página de inicio (index) y la de creación de producto (nuevo_producto) agregando en el lugar que lo desea un código como el que se muestra en la figura.

```
<body>
        <header th:replace="layout/plantilla :: header"></header>

    <footer th:replace="layout/plantilla :: footer"></footer>
```

Al ejecutar el proyecto se puede visualizar la información del encabezado y el pie creada en la plantilla como se aprecia en la figura:

Encabezado de Páginas

Listado de Productos

Crear un nuevo Producto

Código	Nombre	Nit Proveedor	Precio Compra	Iva	Precio Venta	Acciones
1	Melocotones	1	25505.0	19.0	30351.0	Editar Eliminar
2	Manzanas	3	18108.0	19.0	21549.0	Editar Eliminar
3	Plátanos	4	29681.0	19.0	36000.0	Editar Eliminar
4	Lechuga	3	29788.0	19.0	35448.0	Editar Eliminar
5	Tomates	1	12739.0	19.0	15500.0	Editar Eliminar
6	Calabaza	1	21315.0	19.0	25365.0	Editar Eliminar
7	Apio	2	19249.0	19.0	22906.0	Editar Eliminar
8	Pepino	2	10958.0	19.0	13040.0	Editar Eliminar
9	Champiñones	2	11046.0	19.0	13145.0	Editar Eliminar
10	Leche	5	21150.0	19.0	25169.0	Editar Eliminar
11	Queso	5	26571.0	19.0	31619.0	Editar Eliminar
12	Huevos	2	12445.0	19.0	14810.0	Editar Eliminar
13	Requesón	1	14329.0	19.0	17052.0	Editar Eliminar
14	Crema agria	1	14856.0	19.0	17679.0	Editar Eliminar
15	Yogur	5	14941.0	19.0	17780.0	Editar Eliminar
16	Ternera	5	29335.0	19.0	34909.0	Editar Eliminar
17	Salmón salvaje	5	11878.0	19.0	14135.0	Editar Eliminar
18	Patas de cangrejo	1	29951.0	19.0	35642.0	Editar Eliminar
21	Pepinillo	2	12000.0	15.0	25000.0	Editar Eliminar

Derechos Reservados Universidad El Bosque

3. Agregar Bootstrap a nuestro proyecto.

En pocas palabras, los WebJars son dependencias del lado del cliente empaquetadas en archivos JAR. Funcionan con la mayoría de los contenedores JVM y los marcos web.

¿Por qué utilizar WebJars?

Agregar y administrar manualmente las dependencias del lado del cliente a menudo resulta en dificultades para mantener las bases de código.

El principal problema que resuelve WebJars es hacer que las dependencias del lado del cliente estén disponibles en Maven Central y se puedan utilizar en cualquier proyecto estándar de Maven.

Aquí hay algunas ventajas interesantes de WebJars:

- Podemos gestionar de forma explícita y sencilla las dependencias del lado del cliente en aplicaciones web basadas en JVM.
- Podemos usarlos con cualquier herramienta de construcción de uso común, por ejemplo: Maven, Gradle, etc.
- Los WebJars se comportan como cualquier otra dependencia de Maven, lo que significa que también obtenemos dependencias transitivas

Aquí hay algunos WebJars populares: Twitter Bootstrap , jQuery , Angular JS , Chart.js , etc. una lista completa está disponible en el sitio web oficial.

- Bootstrap: es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web.
- Font Awesome: es un conjunto de herramientas de fuentes e íconos basado en CSS y Less. A partir de 2020, Font Awesome fue utilizado por el 38% de los sitios que utilizan scripts de fuentes de terceros, colocando a Font Awesome en el segundo lugar después de Google Fonts
- Locator: Permite direccionan los archivos que usan las librerías.

Los ajustes sobre el proyecto se dividen en los siguientes pasos:

1. Agregar dependencias Maven.
2. Agregar la librería a las plantillas.
3. Ajuste sobre cada página.

1. Agregar dependencias Maven.

   Agregar las dependencias de Maven para gestionar las librerías en el pom.xml.

   Se debe primero agregar al pom.xml las siguientes líneas que permiten añadir las dependencias:

```
<!-- Bootstrap -->
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>bootstrap</artifactId>
    <version>4.4.1-1</version>
</dependency>
<!-- font-awesome -->
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>font-awesome</artifactId>
    <version>5.12.0</version>
</dependency>
<!-- Localizador -->
<dependency>
    <groupId>org.webjars</groupId>
    <artifactId>webjars-locator</artifactId>
    <version>0.38</version>
</dependency>
```

En algún momento si se requiere en el siguiente link se pueden encontrar las dependencias para maven:

https://mvnrepository.com/artifact/org.springframework.boot

2. Agregar la librería a las plantillas.

Ajustar las plantillas que permitan habilitar el uso de los estilos de los íconos, tipos de letra y css de bootstrap, el javascript y el jquery si se requiere de tal forma que se pueda agregar su uso en forma general.

```
<head th:fragment="head">
    <title>Plantilla</title>
    <meta charset="UTF-8"/>
    <link rel="stylesheet" th:href="@{/webjars/bootstrap/css/bootstrap.min.css}"/>
    <link rel="stylesheet" th:href="@{/webjars/font-awesome/css/all.css}"/>
    <script th:src="@{/webjars/jquery/jquery.min.js}"></script>
    <script th:src="@{/webjars/popper.js/umd/popper.min.js}"></script>
    <script th:src="@{/webjars/bootstrap/js/bootstrap.min.js}"></script>
</head>
```

3. Ajuste sobre cada página.

Se ajustan las clases a utilizar en cada página si es necesario, además podemos reutilizar las páginas que tienen la misma función como el de editar y crear.

```
<head th:replace="layout/plantilla :: head">
        <meta charset="UTF-8"/>
        <title>Gestión de Productos</title>
</head>
```

La nueva presentación del proyecto se puede apreciar en la siguiente figura:

Encabezado de Páginas

Listado de Productos

Crear un nuevo Producto

Código	Nombre	Nit Proveedor	Precio Compra	Iva	Precio Venta	Acciones
1	Melocotones	1	25505.0	19.0	30351.0	Editar Eliminar
2	Manzanas	3	18108.0	19.0	21549.0	Editar Eliminar
3	Plátanos	4	29681.0	19.0	36000.0	Editar Eliminar
4	Lechuga	3	29788.0	19.0	35448.0	Editar Eliminar
5	Tomates	1	12739.0	19.0	15500.0	Editar Eliminar
6	Calabaza	1	21315.0	19.0	25365.0	Editar Eliminar
7	Apio	2	19249.0	19.0	22906.0	Editar Eliminar

Taller.

- **Agrega al proyecto el menú de gestión del proyecto del ciclo IV.**
- **Ajusta el encabezado y el pie de página aplicando las características de estilo.**
- **Ajusta cada una de las páginas creadas en el proyecto usando los controles de Bootstrap que mejoren la presebtación.**