CURSO DE PROGRAMACIÓN FULL STACK

APÉNDICE E

PROGRAMACIÓN WEB EN JAVA

APLICACIONES WEB CON SPRING BOOT

EJEMPLO DE UN POM

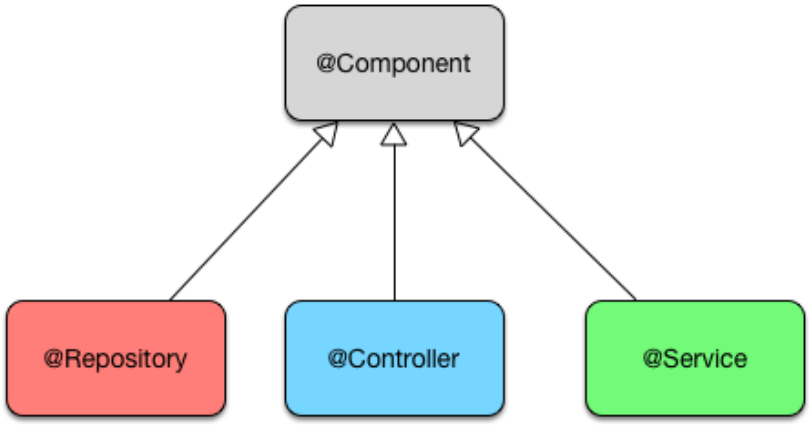
Ejemplo del POM en un proyecto maven. Para indicar que se quiere utilizar Spring Web con Spring Boot. Archivo pom.xml:

```
<parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.1.6.RELEASE</version>
            <relativePath/> <!-- lookup parent from repository -->
      </parent>
      <dependencies>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-data-jpa</artifactId>
            </dependency>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-thymeleaf</artifactId>
            </dependency>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-web</artifactId>
            </dependency>

            <dependency>
                  <groupId>mysql</groupId>
                  <artifactId>mysql-connector-java</artifactId>
                  <scope>runtime</scope>
            </dependency>
            <dependency>
                  <groupId>org.springframework.boot</groupId>
                  <artifactId>spring-boot-starter-test</artifactId>
                  <scope>test</scope>
            </dependency>
       <dependency>
```

De este modo, el proyecto creado extiende del proyecto padre spring-boot-starter-parent, e incluye las dependencias agrupadas en el starter spring-boot-starter-web. Un starter es un conjunto de dependencias que nos ayudan a cubrir las necesidades de un tipo de proyecto concreto.

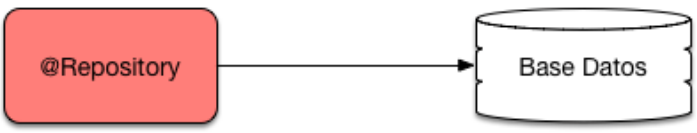
SPRING STEREOTYPES Y ANOTACIONES

¿Cuáles son los Spring Stereotypes?. Spring define un conjunto de anotaciones core que categorizan cada uno de los componentes asociándoles una responsabilidad concreta.

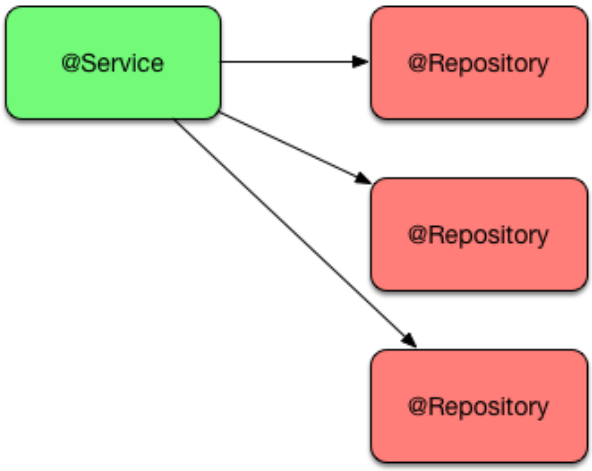


@Component: Es el estereotipo general y permite anotar un bean para que Spring lo considere uno de sus objetos. Un bean es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java. Sustituye la declaración del bean en el xml.

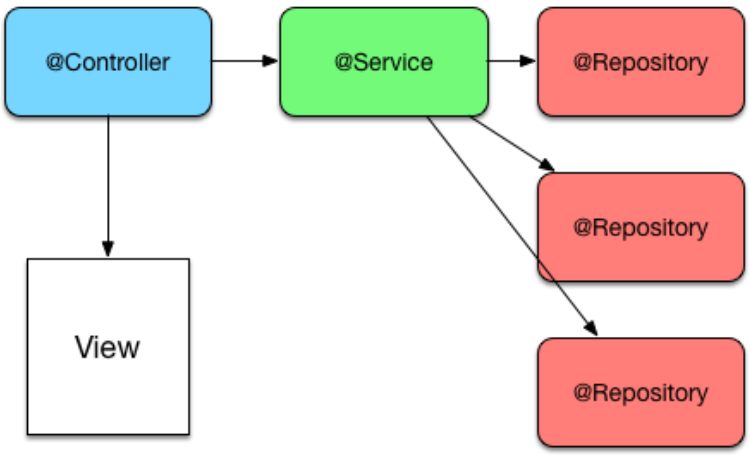
@Repository: Es el estereotipo que se encarga de dar de alta un bean para que implemente el patrón repositorio que es el encargado de almacenar datos en una base de datos o repositorio de información que se necesite. Al marcar el bean con esta anotación Spring aporta servicios transversales como conversión de tipos de excepciones.



@Service: Este estereotipo se encarga de gestionar las operaciones de negocio más importantes a nivel de la aplicación y aglutina llamadas a varios repositorios de forma simultánea. Su tarea fundamental es la de agregador.



@Controller: El último de los estereotipos que es el que realiza las tareas de controlador y gestión de la comunicación entre el usuario y la aplicación. Para ello se apoya habitualmente en algún motor de plantillas o librería de etiquetas que facilitan la creación de páginas. Donde se realiza la asignación de solicitudes desde la página de presentación, es decir, la capa de presentación (o Interface) no va a ningún otro archivo, va directamente a la clase @Controller y comprueba la ruta solicitada en la anotación *@RequestMapping* escrita antes de las llamadas al método si es necesario.



@Autowired Esta anotación le indica a Spring dónde debe ocurrir una inyección. Si se lo coloca en un método, por ejemplo: setMovie, entiende (por el prefijo que establece la anotación @Autowired) que se necesita inyectar un bean. Spring busca un bean de tipo Movie y, si lo encuentra, lo inyecta a este método. Sustituye la declaración de los atributos del bean en el xml. @Autowired se emplea para generar la inyección de dependenciaa de un tipo de Objeto que pertenece a una clase con la @Component.

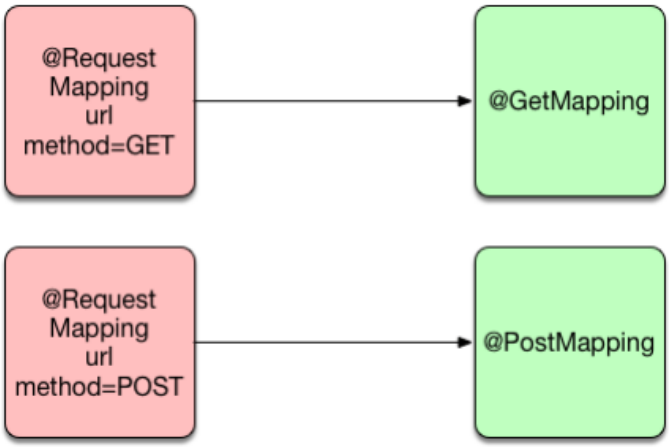
@Qualifier(«nombreBean»): es una de las anotaciones más prácticas de Spring cuando se quiere añadir versatilidad a como se realiza un @Autowired en los componentes. Sirve para indicar que clase es la que se debe inyectar. Con esta anotación podemos indicar el id del bean que se quiere inyectar, esta anotación se usa cuando el atributo que vamos a inyectar es una interfaz de la que hay varias implementaciones y entonces será mediante esta anotación con la que le diremos cual es la clase que queremos inyectar.

@Resource(«nombreBean»): Sustituye el uso de las anotaciones @Autowired y *@Qualifier(«nombreBean»)* de forma que es necesaria una sola anotación.

@Transactional: Esta anotación de Spring indica que el método en cuestión es transaccional. Lo que hará Spring es comprobar si ya existe una transacción abierta, si existe se unirá a ella, y si no existe, abrirá una nueva transacción (este comportamiento es configurable). De esta forma nos aseguramos que toda operación de la base de datos se realiza dentro de una transacción

@RequestMapping: Anotación que se encarga de relacionar un método con una petición http. La anotación @RequestMapping se puede utilizar a nivel de clase y a nivel de método. Si se define a nivel de clase, por ejemplo @RequestMapping(value="/home") las anotaciones de los métodos utilizarán el valor de la anotación de la clase como base o prefijo, es decir si se anota un método con @RequestMapping(value="/welcome") se estará mapeando /home/welcome.

@GetMapping es una anotación de Spring que permite simplificar el manejo de los diferentes métodos de Spring MVC y los @RequestMappings que a veces se hacen un poco pesados.



EJEMPLO LIBRERÍA

Servicios

```
@Servicio  ← Indica que esta clase es un controlador
public class LibreriaServicio{

        @PersistenceContext ← Esta anotación inicia la unidad de persistencia e instancia el EntityManager.
        private EntityManager em;

        @Transactional ← Esta anotación inicia y finaliza la transacción
        public void prestar(String idCliente, String idLibro){
                Prestamo prestamos = new Prestamo();

                Libro libro = em.findById(Libro.class, idLibro);
                Cliente cliente = em.findById(Cliente.class, idCliente);
                prestamo.setCliente(cliente);
                prestamo.setLibro(libro);
        em.persist(prestamo);
    }
```

¿Cómo instanciar Servicios?

```
@Controller  ← Indica que esta clase es un controlador
@RequestMapping("/libreria")  ← Indica que va a atender las URL que comienzan con /libreria
public class LibreriaController{

        @Autowired ← Esta anotación se encarga de que el servidor web instancie la variable.
        private LibreriaService libreriaService;

        @GetMapping("/prestamo") ← Se va a ejecutar cuando la URL sea  /libreria/prestamo
        public String prestar(ModelMap modelo){
                List<Libro> libros = libreriaService.buscarLibros();
        modelo.put("libros ", libros);
                return prestamo.html
    }

      @GetMapping("/devolucion")  ← Se va a ejecutar cuando la URL sea  /libreria/devolucion
        public String prestar(){
                return devolucion.html
    }
}
```

Controladores

```
@Controller  ← Indica que esta clase es un controlador
@RequestMapping("/libreria")  ← Indica que va a atender las URL que comienzan con /libreria
public class LibreriaController{

        @GetMapping("/prestamo") ← Se va a ejecutar cuando la URL sea /libreria/prestamo
        public String prestar(){
                return prestamo.html
    }

    @GetMapping("/devolucion")  ← Se va a ejecutar cuando la URL sea /libreria/devolucion
        public String prestar(){
                return devolucion.html
    }
}
```

Modelos

Para enviar información desde los controladores a las vistas, como indica el patrón MVC se debe utilizar los modelos. Spring trae algunas clases para manipular los modelos desde los controladores:

- *ModelMap*
- *ModelAndView*

La clase ModelMap funciona como un Mapa en donde utilizamos una llave y un valor.

```
@GetMapping("/prestamo") ← Se va a ejecutar cuando la URL sea /libreria/prestamo
public String prestar(ModelMap modelo){
        //Estas líneas utilizan un service para buscar los libros de la biblioteca y los envía a la vista.

        List<Libro> libros = libroService.buscarLibros();
        modelo.put("libros", libros);

        //Estas líneas utilizan un service para buscar el usuario que está logueado y manda sus datos a la vista.

        Usuario usuario = seguridadService.usuarioLogueado();
    modelo.put("usuario", usuario);

    return prestamo.html
}
```

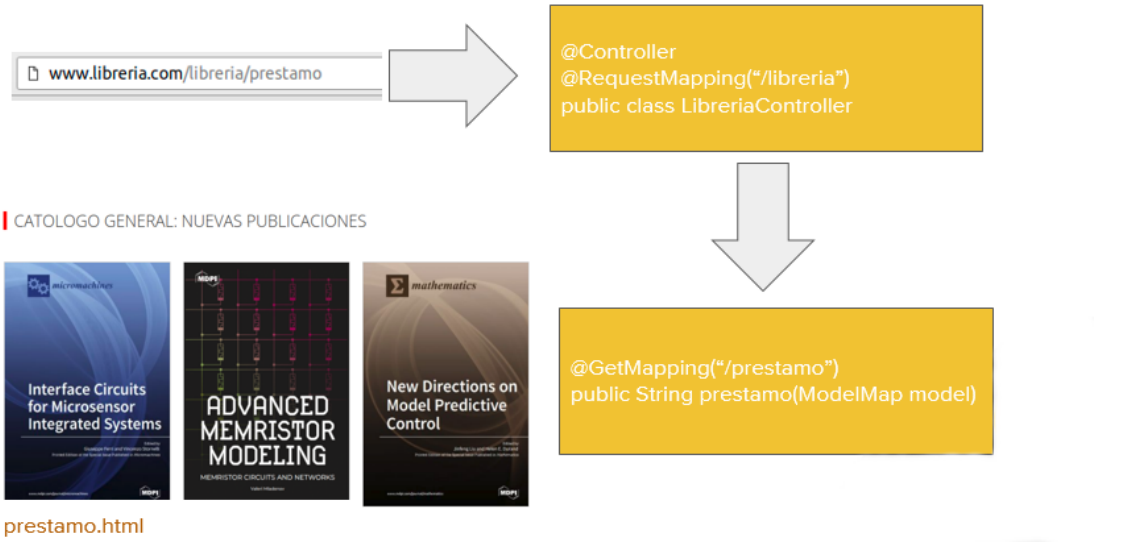
Parámetros

```
@GetMapping("/busqueda") ← Se va a ejecutar cuando la URL sea /libreria/busqueda
public String prestar(@RequestParam String titulo, ModelMap modelo){

//La anotación @RequestParam indica que la URL debe contener un parámetro llamado titulo.


//Estas líneas utilizan un service para buscar los libros por el titulo ingresado de la biblioteca y los envía a la vista.

        List<Libro> libros = libroService.buscarLibros("%" + titulo + "%");
        modelo.put("libros", libros);

//Estas líneas utilizan un service para buscar el usuario que está logueado y manda sus datos a la vista.

        Usuario usuario = seguridadService.usuarioLogueado();
    modelo.put("usuario", usuario);

    return prestamo.html
}
```

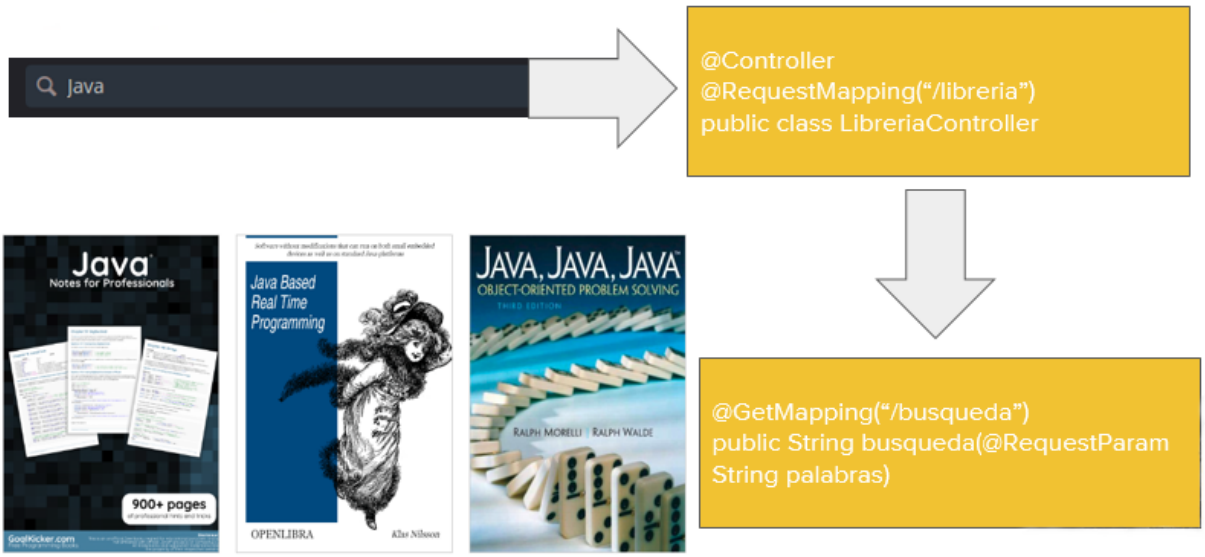
¿Cómo funciona?

www.libreria.com/libreria/prestamo → Va a llamar al método *prestamo* del controlador: *LibreriaController* va a llenar el *ModelMap* con la información solicitada y va a abrir la vista prestamo.html. Esta vista puede utilizar todos los objetos almacenados en el model.

www.libreria.com/libreria/busqueda?titulo=Mendoza → Va a llamar al método buscar del controlador: *LibreriaController* va a llenar el *ModelMap* con la información solicitada, en la variable titulo de este método se va a guardar el valor Mendoza. Va a buscar todos los libros que contengan la palabra Mendoza en su título y va a abrir la vista prestamo.html.



prestamo.html



prestamo.html

HTML5
CABECERAS

DOCTYPE

El estándar XHTML deriva de XML, por lo que comparte con él muchas de sus normas y sintaxis. Uno de los conceptos fundamentales de XML es la utilización del DTD o Document Type Definition ("Definición del Tipo de Documento"). El estándar XHTML define el DTD que deben seguir las páginas y documentos XHTML.
En este documento se definen las etiquetas que se pueden utilizar, los atributos de cada etiqueta y el tipo de valores que puede tener cada atributo.

```
<!DOCTYPE html>
```

HTML

En todo documento HTML, su elemento raíz o nodo superior siempre es la etiqueta html. Hasta ahora, este elemento raíz se define de la siguiente manera:

```
<html lang="es">
…
</html>
```

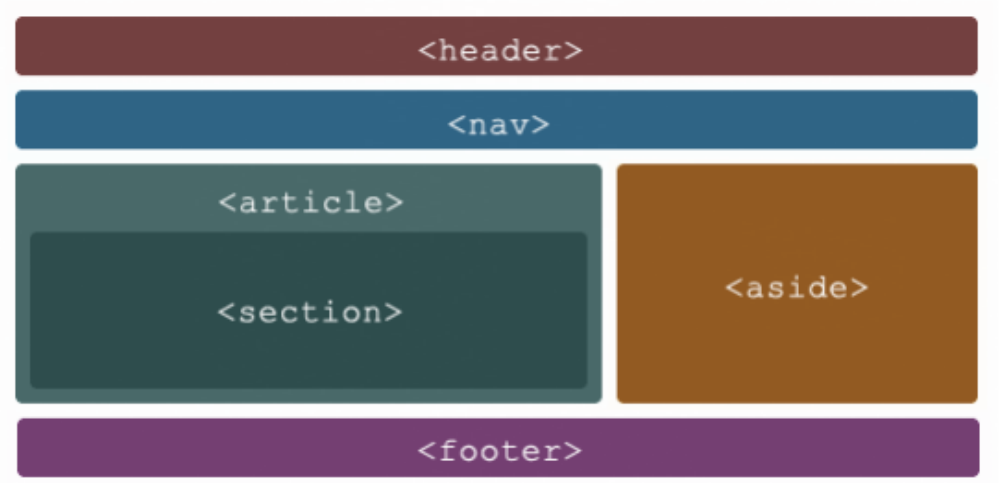
HEAD

El primer hijo del elemento raíz es generalmente el elemento head. El elemento head contiene los metadatos que aportan información extra sobre la página, como su título, descripción, autor, etc.

Además, puede incluir referencias externas a contenidos necesarios para que el documento se muestre y comporte de manera correcta (como hojas de estilos o scripts).

```
<head>
      <title>Mi Página</title>
<meta charset="utf-8" />

      <link rel="icon" href="/img/icono.ico" type="image/x-icon"/>
      <link rel="stylesheet" href="/css/bootstrap.min.css"/>
</head>
```

ESTRUCTURA BÁSICA DE UNA PÁGINA HTML5




```
<body>
    <header>
        <a href="/"><img src=logo.png alt="home"></a>
        <hgroup>
            <h1>Title</h1>
            <h2 class="tagline">
                A lot of effort went into making this effortless.
            </h2>
        </hgroup>
    </header>
    <nav>
        <ul>
            <li><a href="#">home</a></li>
            <li><a href="#">blog</a></li>
            <li><a href="#">gallery</a></li>
            <li><a href="#">about</a></li>
        </ul>
    </nav>
    <section class="articles">
        <article>
            <time datetime="2009-10-22">October 22, 2009</time>
            <h2>
                <a href="#" title="link to this post">Travel day</a>
            </h2>
            <div class="content">
                Content goes here...
            </div>
            <section class="comments">
                <p><a href="#">3 comments</a></p>
            </section>
        </article>
    </section>
    <aside>
        <div class="related"></div>
```

<section>: se utiliza para representar una sección "general" dentro de un documento o aplicación, como un capítulo de un libro. Puede contener subsecciones y si lo acompañamos de h1-h6 podemos estructurar mejor toda la página creando jerarquías del contenido, algo muy favorable para el buen posicionamiento web.

<article>: representa un componente de una página que consiste en una composición autónoma en un documento, página, aplicación, o sitio web con la intención de que pueda ser reutilizado y repetido.

<aside>: representa una sección de la página que abarca un contenido relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Este elemento puede utilizarse para efectos tipográficos, barras laterales, elementos publicitarios u otro contenido que se considere separado del contenido principal de la página.

<header>: representa un grupo de artículos introductorios o de navegación. Está destinado a contener por lo general la cabecera de la sección (un elemento h1-h6 o un elemento hgroup).

<nav>: representa una sección de una página que enlaza a otras páginas o a otras partes dentro de la página. No todos los grupos de enlaces en una página necesita estar en un elemento nav, sólo las secciones que constan de bloques de navegación principales son apropiadas para el elemento de navegación.

<footer>: representa el pie de una sección, con información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright o el año.

<hgroup>: representa el encabezado de una sección. El elemento se utiliza para agrupar un conjunto de elementos h1-h6 cuando el título tiene varios niveles, tales como subtítulos o títulos alternativos.

Elementos en bloque y en línea - https://www.w3schools.com/tags/default.asp
El lenguaje HTML clasifica a todos los elementos en dos grupos: elementos en línea o inline elements y elementos en bloque o block elements. La diferencia entre ambos viene dada por el modelo de contenido, por el formato y la dirección.
Los elementos en bloque siempre empiezan en una nueva línea y ocupan todo el espacio disponible hasta el final de la línea, mientras que los elementos en línea sólo ocupan el espacio necesario para mostrar sus contenidos.

Atributos

Una etiqueta por sí sola no contiene la suficiente información para estar completamente definida. Aparecen los **atributos**, para especificar el elemento. Los atributos se componen por *nombre* y su valor (contenido de las variables), este último separado por comillas dobles o simples. Se encuentran escritos en la etiqueta inicial del elemento HTML. Existen también atributos que afectan al elemento por su presencia (atributo *ismap* en etiqueta <img>)"

EJ: <a **href**="http:\\www.ejemplo.com" **target**="_blank" class="clase" > Muestra contenido

Donde:

<a> es la etiqueta inicial, y **** es la etiqueta de cierre;

href y **target** son los atributos;

Tipos de atributos

1) Básicos: Utilizadas en la mayoría de las etiquetas HTML y XML.

Ejemplos:

- **id**="texto": Establece un indicador único a cada elemento (#texto).
- **class**="texto": Establece la clase CSS que se aplica a los estilos del elemento (.texto).
- **style**="texto" Aplica de forma directa los estilos CSS de un elemento.

2) De Eventos: en etiquetas vinculadas a JavaScript para realizar acciones dinámicas sobre los elementos.

Ejemplos:

- **onclick I onmouseover** Utilizado por todos los elementos.
- **onload I onresize** Utilizado por <body>
- **onkeypress I onkeyup** Utilizado por elementos del formulario y <body>
- **onblur I onfocus** <button>, <input>, <label>, <select>, <textarea>, <body>
- **onsubmit I onreset** <form>

3) De Internalización: utilizado en aquellas páginas que muestran sus contenidos en varios idiomas y las que quieran indicar de forma explícita el idioma de sus contenidos.

- **lang="es":** Indica el idioma del elemento.

FORMULARIOS

TIPO EMAIL

El tipo email indica al navegador que no debe permitir que se envíe el formulario si el usuario no ha introducido una dirección de email válida, pero no comprueba si la dirección existe o no, sólo si el formato es válido. Como ocurre con el resto de campos de entrada, puede enviar este campo vacío a menos que se indique que es obligatorio.
El atributo multiple indica que el valor de este campo, puede ser una lista de emails válidos, separados por comas.

```
<input type="email" name="mail"/>
```

TIPO URL

El tipo url indica al navegador que no debe permitir que se envíe el formulario si el usuario no ha introducido una URL correcta.
Una URL no tiene que ser necesariamente una dirección web, sino que es posible utilizar cualquier formato de URI válido, como por ejemplo tel:555123456.

```
<input type="url" name="web"/>
```

TIPO DATE

En muchos de los sitios web es normal disponer de campos específicos de fecha, donde el usuario debe especificar fechas (para un concierto, vuelo, reserva de hotel, etc).

Al existir tantos formatos de fecha diferentes (DD-MM-YYYY o MM-DD-YYYY o YYYY-MM-DD), esto puede suponer un inconveniente para los desarrolladores o usuarios.

Este nuevo tipo de campo resuelve estos problemas, ya que es el navegador el que proporciona la interfaz de usuario para el calendario, e independientemente del formato en el que se muestre, los datos que se envían al servidor cumplen la norma ISO para el formato de fechas.

```
<input type="date" name="fechaNacimiento>
```

TIPO DATE, TIME, DATETIME, MONTH, WEEK

Estos tipos de datos utilizan el mismo principio que el tipo date solo que cambian la manera en la que se tomará la fecha.

TIME: Permite la selección de un horario del día.

DATETIME: Permite la selección de la fecha y la hora del día.

MONTH: Permite la selección de un mes del año.

WEEK: Permite la selección de una semana del año.

```
<input type="date" name="fechaNacimiento>
<input type="time" name="horaIngreso>
<input type="datetime" name="fechaVuelo>
<input type="month" name="cierreBalance>
```

TIPO NUMBER

Como es de esperar, el tipo valida la entrada de un tipo de dato numérico.
Este tipo de campo encaja perfectamente con los atributos min, max y step.

MIN: Establece el numero minimo a ingresar.

MAX: Establece el numero maximo a ingresar.

STEP: Establece el paso de variación del campo.

```
<input type="number" name="sueldo" min="0" max="1000" step="100">
```

TIPO RANGE

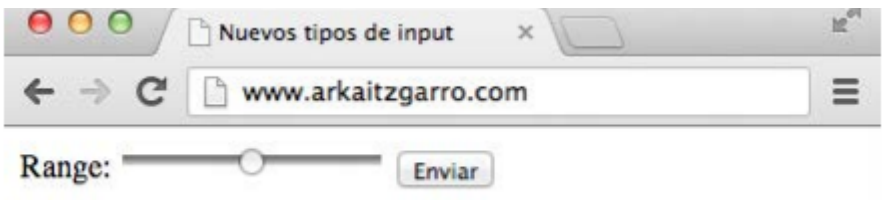
Muestra un control deslizante en el navegador. Para conseguir un elemento de este estilo, era necesario un gran esfuerzo para combinar imágenes, funcionalidad y accesibilidad, siendo ahora mucho más sencillo.

MIN: Establece el numero minimo a ingresar.

MAX: Establece el numero maximo a ingresar.

STEP: Establece el paso de variación del campo.

```
<input type="range" name="sueldo" min="0" max="1000" step="100">
```



ATRIBUTOS

AUTOFOCUS

El atributo booleano autofocus permite definir que control va a tener el foco cuando la página se haya cargado.
Hasta ahora, esto se conseguía a través de JavaScript, utilizando el método *.focus()* en un elemento concreto, al cargarse el documento.

Ahora es el navegador el encargado de esta tarea, y puede comportarse de manera más inteligente, como no cambiando el foco de un elemento si el usuario ya se encuentra escribiendo en otro campo (éste era un problema común con JavaScript). Únicamente debe existir un elemento con este atributo definido en el documento.

PLACEHOLDER

Una pequeña mejora en la usabilidad de los formularios, suele ser colocar un pequeño texto de ayuda en algunos campos, de manera discreta y que desaparece cuando el usuario introduce algún dato.
Como con el resto de elementos, hasta ahora era necesario utilizar JavaScript para realizar esta tarea, pero el atributo *placeholder* resuelve esta tarea.
Es importante recordar que este atributo no sustituye a la etiqueta.

```
<label>Nombre:</label>
<input type=”nombre” placeholder=”Introduzca el nombre del alumno”>
```

REQUIRED

Este atributo puede ser utilizado en un <textarea> y en la gran mayoría de los elementos <input> (excepto en los de tipo hidden, image o botones como submit).
Cuando este atributo está presente, el navegador no permite el envío del formulario si el campo en concreto está vacío.

```
<label>Documento:</label>
<input type=”documento” required>
```

MULTIPLE

Este atributo permite definir que un campo puede admitir varios valores, como URLs o emails.
Un uso muy interesante de este atributo es utilizarlo en conjunto con el campo, ya que de esta manera nos permite seleccionar varios ficheros que podemos enviar al servidor al mismo tiempo.

```
<label>Foto:</label>
<input type=”file” name=”foto” multiple>
```

AUTOCOMPLETE

Este atributo permite controlar el comportamiento del autocompletado en los campos de texto del formulario (que por defecto está activado).

```
<label>Tarjeta de Credito:</label>
<input type=”text” name=”numero” autocomplete=”false”>
```

PATTERN

Algunos de los tipos de input que hemos visto anteriormente (email, number, url…), son realmente expresiones regulares que el navegador evalúa cuando se introducen datos. El atributo pattern nos permite definir una expresión regular que el valor del campo debe cumplir.

Por ejemplo, si el usuario debe introducir un número seguido de tres letras mayúsculas, podríamos definir esta expresión regular:

```
<label>Código:</label>
<input type=”text” name=”codigo” pattern=”[0-9][A-Z]{3}”>
```

CSS

Antes de la adopción de CSS, los diseñadores de páginas web debían definir el aspecto de cada elemento dentro de las etiquetas HTML de la página. El siguiente ejemplo muestra una página HTML con estilos definidos sin utilizar CSS:

```
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de estilos sin CSS</title>
</head>

<body>

<h1><font color="red" face="Arial" size="5">Titular de la página</font></h1>

<p><font color="gray" face="Verdana" size="2">Un párrafo de texto no muy
largo.</font></p>

</body>
</html>
```

La solución que propone CSS es mucho mejor, como se puede ver en el siguiente ejemplo:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de estilos con CSS</title>
<style type="text/css">
  h1 { color: red;  font-family: Arial;   font-size: large;  }
  p  { color: gray; font-family: Verdana; font-size: medium; }
</style>
</head>

<body>
<h1>Titular de la página</h1>
<p>Un párrafo de texto no muy largo.</p>
</body>
</html>
```

¿Cómo incluir CSS en el HTML?

Dentro del archivo HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de estilos CSS en el propio documento</title>
<style type="text/css">
  p { color: black; font-family: Verdana; }
</style>
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

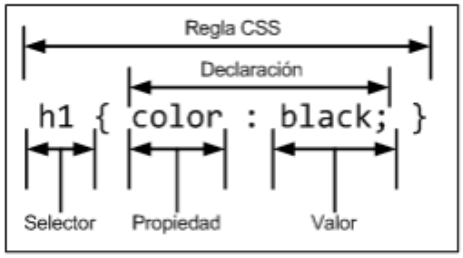
Fuera del archivo HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Ejemplo de estilos CSS en un archivo externo</title>
<link rel="stylesheet" type="text/css" href="/css/estilos.css" media="screen" />
</head>

<body>
<p>Un párrafo de texto.</p>
</body>
</html>
```

Normalmente, la etiqueta incluye cuatro atributos cuando se enlaza un archivo CSS:

- ***rel:*** *indica el tipo de relación que tiene el recurso enlazado y la página HTML. Para los archivos* ***CSS****, siempre se utiliza el valor* ***stylesheet****.*
- ***type:*** *indica el tipo de recurso enlazado. Sus valores están estandarizados y para los archivos* ***CSS*** *su valor siempre es* ***text/css***
- ***href:*** *indica la URL del archivo CSS que contiene los estilos. La URL indicada puede ser relativa o absoluta y puede apuntar a un recurso interno o externo al sitio web.*
- ***media:*** *indica el medio en el que se van a aplicar los estilos del archivo CSS. Más adelante se explican en detalle los medios CSS y su funcionamiento.*

CSS define una serie de términos que permiten describir cada una de las partes que componen los estilos CSS. El siguiente esquema muestra las partes que forman un estilo CSS muy básico:



Los diferentes términos se definen a continuación:

- *Regla: cada uno de los estilos que componen una hoja de estilos CSS.*
- *Selector: indica el elemento o elementos HTML a los que se aplica la regla CSS.*
- *Declaración: especifica los estilos que se aplican a los elementos. Está compuesta por una o más propiedades CSS.*
- *Propiedad: permite modificar el aspecto de una característica del elemento.*
- *Valor: indica el nuevo valor de la característica modificada en el elemento.*

MEDIOS CSS

Una de las características más importantes de las hojas de estilos CSS es que permiten definir diferentes estilos para diferentes medios o dispositivos: pantallas, impresoras, móviles, proyectores, etc.

Medio	Descripción
`all`	Todos los medios definidos
`braille`	Dispositivos táctiles que emplean el sistema braille
`embosed`	Impresoras braille
`handheld`	Dispositivos de mano: móviles, PDA, etc.
`print`	Impresoras y navegadores en el modo *"Vista Previa para Imprimir"*
`projection`	Proyectores y dispositivos para presentaciones
`screen`	Pantallas de ordenador
`speech`	Sintetizadores para navegadores de voz utilizados por personas discapacitadas
`tty`	Dispositivos textuales limitados como teletipos y terminales de texto
`tv`	Televisores y dispositivos con resolución baja

Ejemplo del uso de medios en CSS:

```
@media print {
        body { font-size: 10pt }
        p { color: black }
}

@media screen {
        body { font-size: 13px }
        p { color: red}
}

@media screen, print {
        body { line-height: 1.2 }
}
```

SELECTORES CCS

Selector Universal

Se utiliza para seleccionar todos los elementos de la página. El siguiente ejemplo elimina el margen y el relleno de todos los elementos HTML (por ahora no es importante fijarse en la parte de la declaración de la regla CSS):

```
* {
        margin: 0;
        padding: 0;
 }
```

Selector de Etiqueta

Selecciona todos los elementos de la página cuya etiqueta HTML coincide con el valor del selector. El siguiente ejemplo selecciona todos los párrafos de la página:

```
p {
        text-align: justify;
        font-family: Verdana;
 }
```

El siguiente ejemplo selecciona todas las tablas y div de la página:

```
table, div {
        border: 1px solid red;
 }
```

Selector Descendente

Selecciona los elementos que se encuentran dentro de otros elementos. Un elemento es descendiente de otro cuando se encuentra entre las etiquetas de apertura y de cierre del otro elemento.

El selector del siguiente ejemplo selecciona todos los elementos de la página que se encuentren dentro de un elemento <p>.

```
p span { color: red; }
```

Selector de Clase

¿Como hago para aplicarle estilos solo al primer párrafo?

```
<body>
  <p>Lorem ipsum dolor sit amet...</p>
  <p>Nunc sed lacus et est adipiscing accumsan...</p>
  <p>Class aptent taciti sociosqu ad litora...</p>
</body>
```

Una de las soluciones más sencillas para aplicar estilos a un solo elemento de la página consiste en utilizar el atributo class de HTML sobre ese elemento para indicar directamente la regla CSS que se le debe aplicar:

```
<body>
  <p class="destacado">Lorem ipsum dolor sit amet...</p>
  <p>Nunc sed lacus et est adipiscing accumsan...</p>
  <p>Class aptent taciti sociosqu ad litora...</p>
</body>
```

Ejemplos:

```
.error {
	color: red;
}

.destacado {
	font-size: 15px;
}

.especial {
	font-weight: bold;
}


#cabecera {
	color: black;
	brackground-color: red;
	font-weight: bolder;
}
```

Selector de Elemento

En ocasiones, es necesario aplicar estilos CSS a un único elemento de la página. Aunque puede utilizarse un selector de clase para aplicar estilos a un único elemento, existe otro selector más eficiente en este caso. El selector de ID permite seleccionar un elemento de la página a través del valor de su atributo id.

Este tipo de selectores sólo seleccionan un elemento de la página porque el valor del atributo id no se puede repetir en dos elementos diferentes de una misma página.

Prioridad en aplicación de estilos:

Herencia

Los hijos heredan los estilos de sus elementos padres, no es necesario declarar sus estilos si estos se mantienen igual.

```
body{
    color: yellow;
}
h2{
    color: yellow; /*No es necesario*/
}
```

Cascada

Todo estilo sobrescribe a uno anterior.

```
h2{
    color: yellow;
}
h2{
    color: red;
}
```

Especificidad

Cuando hay conflictos de estilos el navegador aplica sólo el de mayor especificidad.

```css
h2{
    color: red
}
h2.subtitle{
    color: purple;
}
```

PROPIEDADES CSS

Ancho

La propiedad CSS que controla la anchura de los elementos se denomina width.

width	Anchura
Valores	<medida> | <porcentaje> | auto | inherit
Se aplica a	Todos los elementos, salvo los elementos en línea que no sean imágenes, las filas de tabla y los grupos de filas de tabla
Valor inicial	auto
Descripción	Establece la anchura de un elemento

Alto

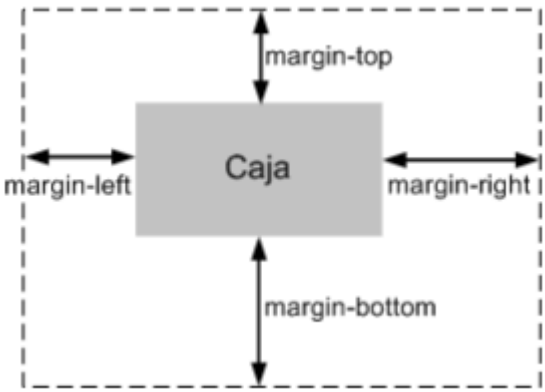
La propiedad CSS que controla la anchura de los elementos se denomina height.

height	Altura
Valores	<medida> | <porcentaje> | auto | inherit
Se aplica a	Todos los elementos, salvo los elementos en línea que no sean imágenes, las columnas de tabla y los grupos de columnas de tabla
Valor inicial	auto
Descripción	Establece la altura de un elemento

Margen

CSS define cuatro propiedades para controlar cada uno de los márgenes horizontales y verticales de un elemento.

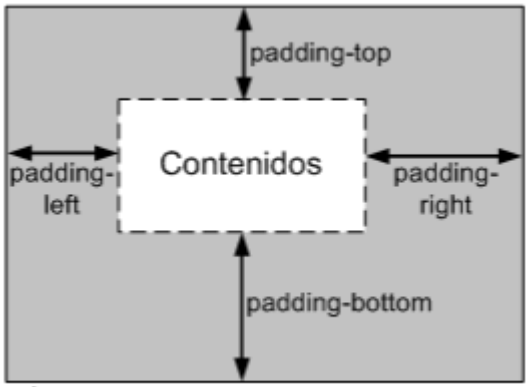
margin-top **margin-right** **margin-bottom** **margin-left**	Margen superior Margen derecho Margen inferior Margen izquierdo
Valores	`<medida> | <porcentaje> | auto | inherit`
Se aplica a	Todos los elementos, salvo margin-top y margin-bottom que sólo se aplican a los elementos de bloque y a las imágenes
Valor inicial	0
Descripción	Establece cada uno de los márgenes horizontales y verticales de un elemento



Relleno

CSS define cuatro propiedades para controlar cada uno de los espacios de relleno horizontales y verticales de un elemento.

padding-top **padding-right** **padding-bottom** **padding-left**	Relleno superior Relleno derecho Relleno inferior Relleno izquierdo
Valores	`<medida> | <porcentaje> | inherit`
Se aplica a	Todos los elementos excepto algunos elementos de tablas como grupos de cabeceras y grupos de pies de tabla
Valor inicial	0
Descripción	Establece cada uno de los rellenos horizontales y verticales de un elemento



caja

Bordes - Tamaño

CSS permite definir el aspecto de cada uno de los cuatro bordes horizontales y verticales de los elementos. Para cada borde se puede establecer su anchura, su color y su estilo.

border-top-width **border-right-width** **border-bottom-width** **border-left-width**	Anchura del borde superior Anchura del borde derecho Anchura del borde inferior Anchura del borde izquierdo
Valores	`( <medida> | thin | medium | thick ) | inherit`
Se aplica a	Todos los elementos
Valor inicial	Medium
Descripción	Establece la anchura de cada uno de los cuatro bordes de los elementos

Bordes - Color

El color de los bordes se controla con las cuatro propiedades siguientes:

border-top-color **border-right-color** **border-bottom-color** **border-left-color**	Color del borde superior Color del borde derecho Color del borde inferior Color del borde izquierdo
Valores	`<color> | transparent | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Establece el color de cada uno de los cuatro bordes de los elementos

Bordes - Estilo

El color de los bordes se controla con las cuatro propiedades siguientes:

`border-top-style`
`border-right-style`
`border-bottom-style`
`border-left-style`
	Estilo del borde superior
	Estilo del borde derecho
	Estilo del borde inferior
	Estilo del borde izquierdo

Valores	`none | hidden | dotted | dashed | solid | double | groove | ridge | inset | outset | inherit`
Se aplica a	Todos los elementos
Valor inicial	none
Descripción	Establece el estilo de cada uno de los cuatro bordes de los elementos

Bordes - Forma Resumida

Todos los estilos de los bordes se controlan con la siguiente siguientes:

`border`	Estilo completo de todos los bordes
Valores	`( <medida_borde> || <color_borde> || <estilo_borde> ) | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Establece el estilo completo de todos los bordes de los elementos

Fondo - Color

El color de fondo se establece con esta propiedad:

`background-color`	Color de fondo
Valores	`<color> | transparent | inherit`
Se aplica a	Todos los elementos
Valor inicial	transparent
Descripción	Establece un color de fondo para los elementos

La imagen de fondo se establece con esta propiedad:

`background-image`	Imagen de fondo
Valores	`<url> | none | inherit`
Se aplica a	Todos los elementos
Valor inicial	none
Descripción	Establece una imagen como fondo para los elementos

Fondo - Repetición

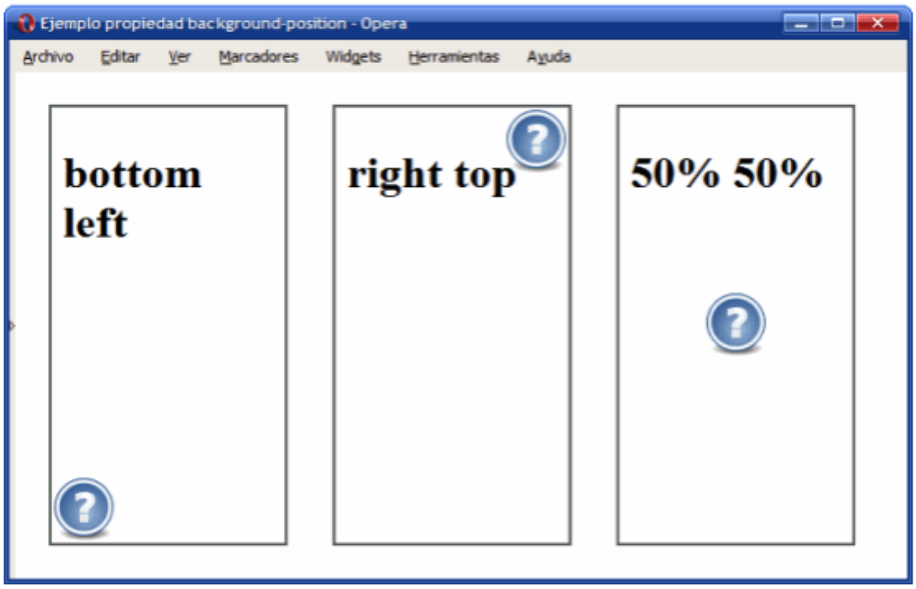
La repetición de la imagen de fondo se configura con esta propiedad:

background-repeat	Repetición de la imagen de fondo
Valores	repeat | repeat-x | repeat-y | no-repeat | inherit
Se aplica a	Todos los elementos
Valor inicial	repeat
Descripción	Controla la forma en la que se repiten las imágenes de fondo

Fondo - Posición

La posición de la imagen de fondo se configura con esta propiedad:

background-position	Posición de la imagen de fondo
Valores	((<porcentaje> | <medida> | left | center | right) (<porcentaje> | <medida> | top | center | bottom)?) | ((left | center | right) || (top | center | bottom)) | inherit
Se aplica a	Todos los elementos
Valor inicial	0% 0%
Descripción	Controla la posición en la que se muestra la imagen en el fondo del elemento



Fondo - Imagen de Fondo

Para controlar la manera de visualizar la imagen de fondo:

background-attachment	Comportamiento de la imagen de fondo
Valores	`scroll | fixed | inherit`
Se aplica a	Todos los elementos
Valor inicial	scroll
Descripción	Controla la forma en la que se visualiza la imagen de fondo: permanece fija cuando se hace scroll en la ventana del navegador o se desplaza junto con la ventana

Fondo - Resumida

Establecer todas las propiedades de fondo:

background	Fondo de un elemento
Valores	`( <background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position> ) | inherit`
Se aplica a	Todos los elementos
Valor inicial	-
Descripción	Establece todas las propiedades del fondo de un elemento

Tipografía - Resumida

CSS define numerosas propiedades para modificar la apariencia del texto. A pesar de que no dispone de tantas posibilidades como los lenguajes y programas específicos para crear documentos impresos, CSS permite aplicar estilos complejos y muy variados al texto de las páginas web. La propiedad básica que define CSS relacionada con la tipografía se denomina color y se utiliza para establecer el color de la letra.

color	Color del texto
Valores	`<color> | inherit`
Se aplica a	Todos los elementos
Valor inicial	Depende del navegador
Descripción	Establece el color de letra utilizado para el texto

Tipografía - Fuente

La otra propiedad básica que define CSS relacionada con la tipografía se denomina font-family y se utiliza para indicar el tipo de letra con el que se muestra el texto.

font-family	Tipo de letra
Valores	((<nombre_familia> | <familia_generica>) (,<nombre_familia> | <familia_generica>)*) | inherit
Se aplica a	Todos los elementos
Valor inicial	Depende del navegador
Descripción	Establece el tipo de letra utilizado para el texto

Tipografía – Tamaño

Una vez seleccionado el tipo de letra, se puede modificar su tamaño mediante la propiedad font-size.

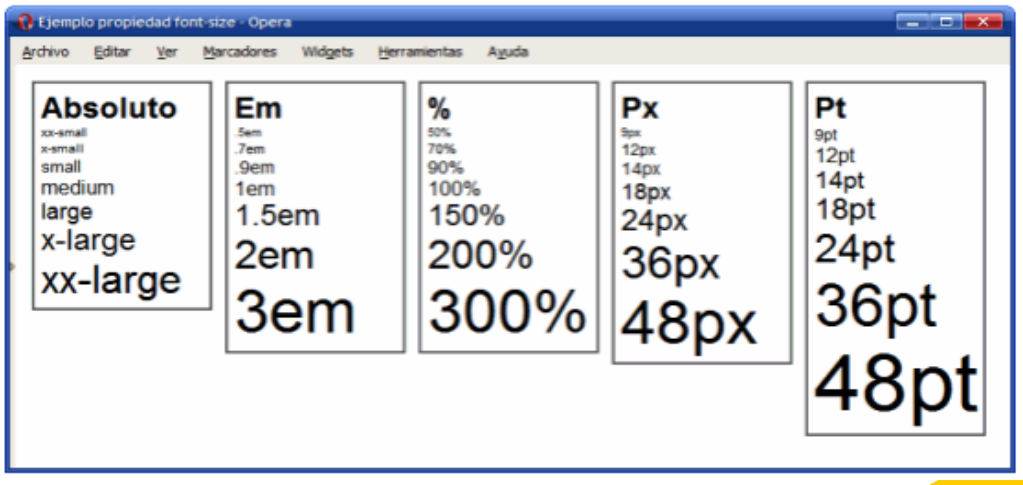
font-size	Tamaño de letra
Valores	<tamaño_absoluto> | <tamaño_relativo> | <medida> | <porcentaje> | inherit
Se aplica a	Todos los elementos
Valor inicial	medium
Descripción	Establece el tamaño de letra utilizado para el texto

Tipografía – Tamaño

Además de todas las unidades de medida relativas y absolutas y el uso de porcentajes, CSS permite utilizar una serie de palabras clave para indicar el tamaño de letra del texto:

- *Tamaño Absoluto: indica el tamaño de letra de forma absoluta mediante alguna de las siguientes palabras clave: xx-small, x-small, small, medium, large, x-large, xx-large.*

- *Tamaño Relativo: indica de forma relativa el tamaño de letra del texto mediante dos palabras clave (larger, smaller) que toman como referencia el tamaño de letra del elemento padre.*



Tipografía – Grosor

Una vez indicado el tipo y el tamaño de letra, es habitual modificar otras características como su grosor (texto en negrita) y su estilo (texto en cursiva). La propiedad que controla la anchura de la letra es font-weight.

font-weight	Anchura de la letra
Valores	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
Se aplica a	Todos los elementos
Valor inicial	normal
Descripción	Establece la anchura de la letra utilizada para el texto

Una vez indicado el tipo y el tamaño de letra, es habitual modificar otras características como su grosor (texto en negrita) y su estilo (texto en cursiva). La propiedad que controla la anchura de la letra es font-weight.

font-style	Estilo de la letra
Valores	normal | italic | oblique | inherit
Se aplica a	Todos los elementos
Valor inicial	normal
Descripción	Establece el estilo de la letra utilizada para el texto

Texto - Alineación

Para establecer la alineación del contenido del elemento. La propiedad text-align no sólo alinea el texto que contiene un elemento, sino que también alinea todos sus contenidos, como por ejemplo las imágenes.

text-align	Alineación del texto
Valores	left | right | center | justify | inherit
Se aplica a	Elementos de bloque y celdas de tabla
Valor inicial	left
Descripción	Establece la alineación del contenido del elemento

Texto - Interlineado

El interlineado de un texto se controla mediante la propiedad line-height, que permite controlar la altura ocupada por cada línea de texto:

line-height	Interlineado
Valores	normal | <numero> | <medida> | <porcentaje> | inherit
Se aplica a	Todos los elementos
Valor inicial	normal
Descripción	Permite establecer la altura de línea de los elementos

Texto - Decoración

El valor *underline* subraya el texto.

El valor *overline* añade una línea en la parte superior del texto.

El valor *line-through* muestra el texto tachado con una línea continua, por lo que su uso tampoco es muy habitual.

El valor *blink* muestra el texto parpadeante.

text-decoration	Decoración del texto
Valores	none | (underline || overline || line-through || blink) | inherit
Se aplica a	Todos los elementos
Valor inicial	none
Descripción	Establece la decoración del texto (subrayado, tachado, parpadeante, etc.)

Texto - Transformación

El valor *capitalize* transforma a mayúsculas la primera letra de las palabras del texto.

El valor *uppercase* transforma a mayúsculas todo el texto.

El valor *lowercase* transforma a minúsculas todo el texto.

text-transform	Transformación del texto
Valores	capitalize | uppercase | lowercase | none | inherit
Se aplica a	Todos los elementos
Valor inicial	none
Descripción	Transforma el texto original (lo transforma a mayúsculas, a minúsculas, etc.)

Pseudo-Clases

CSS también permite aplicar diferentes estilos a un mismo enlace en función de su estado. De esta forma, es posible cambiar el aspecto de un enlace cuando por ejemplo el usuario pasa el ratón por encima o cuando el usuario pincha sobre ese enlace.

Como con los atributos id o class no es posible aplicar diferentes estilos a un mismo elemento en función de su estado, CSS introduce un nuevo concepto llamado pseudo-clases. En concreto, CSS define las siguientes cuatro pseudo-clases:

:link, aplica estilos a los enlaces que apuntan a páginas o recursos que aún no han sido visitados por el usuario.

:visited, aplica estilos a los enlaces que apuntan a recursos que han sido visitados anteriormente por el usuario. El historial de enlaces visitados se borra automáticamente cada cierto tiempo y el usuario también puede borrarlo manualmente.

:hover, aplica estilos al enlace sobre el que el usuario ha posicionado el puntero del ratón.

:active, aplica estilos al enlace que está clickeado el usuario.

MEDIA QUERY

Las *media queries* son útiles cuando deseas modificar tu página web o aplicación en función del tipo de dispositivo (como una impresora o una pantalla) o de características y parámetros específicos (como la resolución de la pantalla o el ancho del navegador).

```
@media tv and (min-width: 700px) and (orientation: landscape) { ... }
```

Para aplicar una hoja de estilo a dispositivos con al menos 4 bits por componente de color:

```
@media all and (min-color: 4) { ... }
```

Para aplicar una hoja de estilo a un dispositivo portátil con una pantalla de 15 caracteres o más estrecha:

```
@media handheld and (grid) and (max-width: 15em) { ... }
```

Especificar una hoja de estilo para dispositivos portátiles o pantallas con un ancho de al menos 20em, usted puede usar esta query:

```
@media handheld and (min-width: 20em), screen and (min-width: 20em) { ... }
```

Las *media queries* son útiles cuando deseas modificar tu página web o aplicación en función del tipo de dispositivo (como una impresora o una pantalla) o de características y parámetros específicos (como la resolución de la pantalla o el ancho del navegador).

```
@media (max-width: 600px) {
  .menu_horizontal {
    display: none;
  }

  .botonera {
    display: block;
  }
}
```

Bootstrap - https://getbootstrap.com/docs/4.5/getting-started/introduction/

Bootstrap es un framework muy utilizado en desarrollo web, ya que proporciona diferentes herramientas de código abierto front-end que permite desarrollar interfaces responsivas y atractivas. Presenta variables Sass y mixins, sistema de cuadrícula sensible, componentes precompilados extensos y complementos potentes de JavaScript.
Presenta facilidades desde la estructuración del sitio (grillas y layout), estilos y acciones dinámicas.

Instalación y uso

Descarga desde el sitio oficial, vinculación a través de CDN u otros medios aplicados en desarrollo:

https://getbootstrap.com/docs/4.5/getting-started/introduction/#css

https://getbootstrap.com/docs/4.5/getting-started/introduction/#js

Plantillas

Dado que es uno de los framework más utilizados, podemos encontrar un amplio abanico de marcos de trabajo pensados y diseñados a partir de los componentes y estilos que presenta Bootstrap, de modo que existen variables y ejemplos listos para utilizar en proyectos específicos.

https://themes.getbootstrap.com/official-themes/

Thymeleaf

En las aplicaciones web desarrolladas con Java siempre se ha contado con múltiples opciones a la hora de presentar contenido mediante plantillas.

CARACTERÍSTICAS PRINCIPALES

Thymeleaf es un framework de templating que promete flexibilidad y adecuación a los nuevos estándares como HTML5.

ESTRUCTURA MÍNIMA

Para comenzar, podemos definir una primera plantilla simple.html con Thymeleaf que recoja el valor de una variable título y lo muestre mediante un `h1`:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
  <body>
    <h1 th:text="${titulo}">Título principal de la página</h1>
  </body>
</html>
```

Para poder ejecutarla y pasarle el valor de las variables que necesita, se deben completar tres simples pasos:

1) Encontrar plantillas

El primero de los pasos es conseguir un *TemplateResolver*. Este interfaz es la base para que Thymeleaf sea capaz de encontrar la plantilla que queremos usar, incluso si esta se encuentra almacenada en la caché.

Tenemos varias implementaciones de este interfaz Java. Las más comunes son *ServletContextTemplateResolver*, para trabajar en el contexto de una aplicación web o *ClassLoaderTemplateResolver* para el resto de casos generales.

```
TemplateResolver templateResolver = new ClassLoaderTemplateResolver();
templateResolver.setTemplateMode('HTML5');
templateResolver.setSuffix('.html');
templateResolver.setCacheable(true);
templateResolver.setCacheTTLMs(3600000L);
```

Como se puede observar en el ejemplo de código, en todas estas implementaciones se puede indicar:

- La ruta exacta al directorio que contiene nuestras plantillas (suffix).
- El modo de trabajo que deseamos: XML, VALIDXML, XHTML, VALIDXHTML, HTML5 o LEGACYHTML5.
- Si la caché está activa o no (recomendable poner esta opción a false durante las pruebas).
- El tiempo de cacheo de las plantillas en milisegundos.

2) Obtener el motor de transformación que realice el procesamiento

Para obtener el motor de transformación, sólo debemos instanciarlo y pasarle el *TemplateResolver* construido en el apartado anterior:

```
TemplateEngine templateEngine = new TemplateEngine();
templateEngine.setTemplateResolver(templateResolver);
```

3) Pasar los parámetros y lanzar la transformación

Ya sólo queda preparar los parámetros que va a necesitar la plantilla diseñada. En este caso, únicamente se necesita un parámetro llamado 'titulo', y ya se puede lanzar el procesamiento:

```
IContext context = new Context();
context.getVariables().put('titulo', 'Mi título en la plantilla');
String result = templateEngine.process('simple', context);
```

Nuevamente, se tienen varias implementaciones de la interface *IContext* en función de si el entorno es web (*WebContext*) o no (*Context*).

Si examinamos el ejemplo anterior, veremos que al método '*process*' del motor de procesamiento le pasamos el nombre del archivo que contiene la plantilla, pero sin la extensión '*.html*'. Es necesario pasarle sólo '*simple*', ya que en el primer apartado ya se estableció '*.html*' como *suffix*.

En definitiva, el resultado de la transformación debería de ser finalmente el esperado:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Mi título en la plantilla</h1>
  </body>
</html>
```

EXPRESIONES Y EL LENGUAJE OGNL

Thymeleaf soporta OGNL (Object-Graph Navigation Language) para la definición de expresiones y como forma de acceder a las variables que le proporcionamos a la plantilla como entrada. Este es un lenguaje que, en el contexto de Thymeleaf, simplifica y flexibiliza la creación de plantillas. A continuación, veremos algunos ejemplos.

Variables simples

Establecidas a partir de un valor textual o de una variable:

```
<input type="text" name="nombre" value="Pedro Perez" th:value="${nombre}" />
```

O también a partir de un bean que contenga un método getter con el nombre de la referencia (getNombre):

```
<input type="text" name="nombre" value="Pedro Perez" th:value="${user.nombre}" />
```

Variables simples con soporte de internacionalización

```
<h1 th:text="#{bienvenida}">Texto de bienvenida</h1>
```

En este ejemplo, al preceder a la variable bienvenida con el caracter # en lugar del $, le indicamos al motor que establezca su valor en función del Locale establecido.

Para el Locale por defecto, buscará el valor de la variable '*bienvenida*' en un fichero '*.properties*' que deberá llamarse igual que la plantilla (*simple.properties* si tomamos como referencia el ejemplo inicial). Así, podremos proporcionar distintas versiones en función del idioma: simple_es.properties, simple_en.properties, etc

Todos estos archivos de localización de cadenas, deben almacenarse en el mismo sitio que las plantillas. En el código de nuestra aplicación, nuestro código en servidor será el siguiente:

```
String result = templateEngine.process("simple_multilang", new Context(new Locale("en")));
```

De forma que si tenemos un fichero *simple_multilang_en.properties* con el siguiente contenido:

```
bienvenida=Welcome to Genbeta Dev web!!
```

El resultado final obtenido será:

```
<h1>Welcome to Genbeta Dev web!!</h1>
```

Bucles

Con Thymeleaf podemos iterar de forma muy sencilla sobre una colección que recibimos como parámetro. Por ejemplo, vamos a pasar un parámetro a la plantilla llamado productos. El valor de este parámetro será una colección definida por un ArrayList de instancias de tipo Producto. Producto es un bean y sus atributos principales son id y nombre. En este escenario, podemos iterar fácilmente sobre esta colección y mostrar todos sus valores

```
...
  <tr th:each="prod : ${productos}">
    <td th:text="${prod.id}">Código del producto</td>
    <td th:text="${prod.nobre}">Nombre del producto</td>
  </tr>
...
```

De forma que simplemente creando la colección en el código del servidor:

```
Producto producto = new Producto();
producto.setId(1);
producto.setNombre("Producto 1");
IContext context = new Context();
context.getVariables().put("productos", Collections.singletonList(producto));
String result = templateEngine.process("colecciones", context);
```

Obtendremos la tabla resultado:

```
<!DOCTYPE html>
<html>
  <body>
    <table>
      <tr>
        <td>1</td>
        <td>Producto 1</td>
      </tr>
    </table>
  </body>
</html>
```

EJEMPLO LIBRERÍA

¿Cómo usar las variables en las Vistas?

En el HTML debemos importar los tag de Thymeleaf para usar las variables del modelo:

```
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:th="http://www.thymeleaf.org" lang="es">
```

Luego en el lugar que queremos escribir el valor de alguna variable del modelo usamos:

```
<span th:text="${usuario.nombre}"></span>
<b th:text="${usuario.nombre}"></b>
<p th:text="${usuario.nombre}"></p>
<h1 th:text="${usuario.nombre}"></h1>
```

Variables Numéricas

Las variables numéricas se pueden utilizar de la misma manera:

```
<span  th:text="${cotizacion.maximo}"></span>
<span th:text="${cotizacion.minimo}"></span>
```

Comercialización

Mínimo	Máximo
547158.2963254593	555490.656167979

Imprime el numero tal como se imprimiría en la consola.

Las variables numéricas se pueden utilizar de la misma manera:

```
<span th:text="${#numbers.formatDecimal(cotizacion.maximo, 1,
2)}"></span>
<span th:text="${#numbers.formatDecimal(cotizacion.minimo,
1,2)}"></span>
```

Mínimo	Máximo
547158.29	555490.65

En este caso indicamos que el formato numérico debe contener como mínimo un numero entero y dos números decimales como máximo.

Variables de Fecha

Las variables de fecha se pueden utilizar de la misma manera:

```
<span th:text="${#dates.format(cotizacion.fecha, 'dd-MM-yyyy
HH:mm')}"></span>
<span th:text="${#dates.format(cotizacion.fecha, 'dd-MM-
yyyy')}"></span>
```

Listas

Luego en el lugar que queremos escribir el valor de alguna variable del modelo usamos:

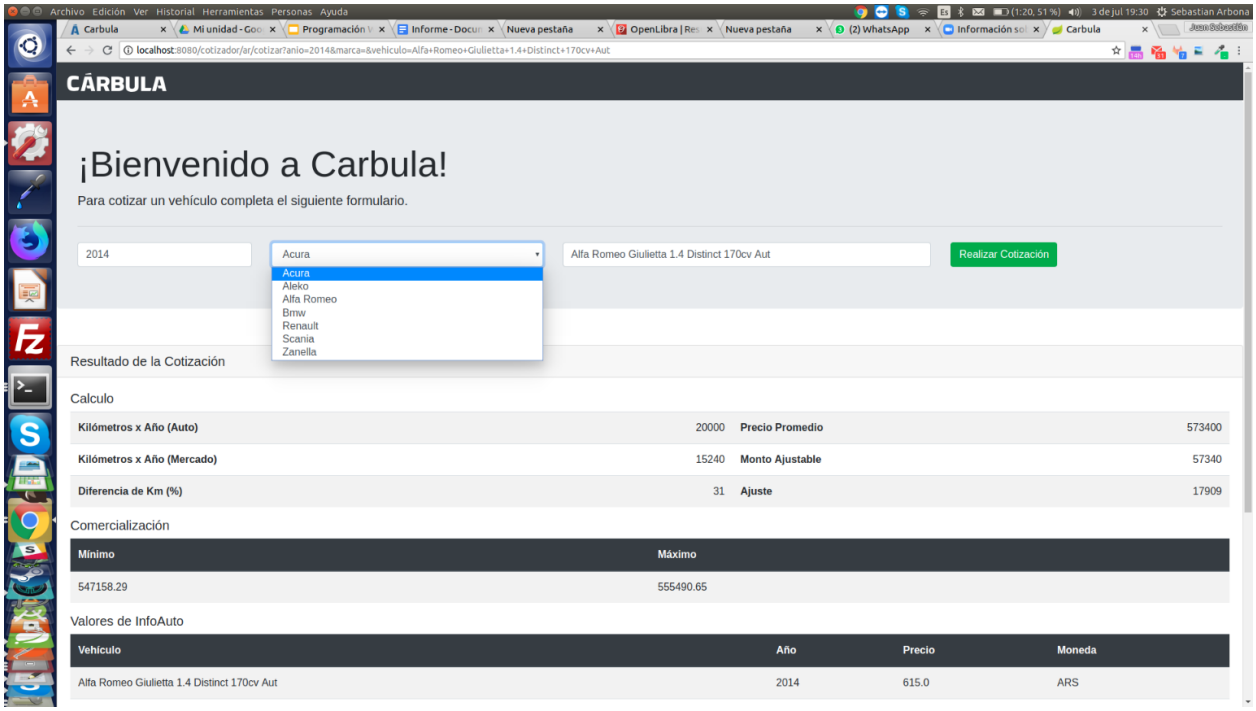
```
<table class="table table-striped">
        <thead class="thead-dark">
                <tr>
                        <th scope="col">ISBN</th>
                        <th scope="col">Titulo</th>
                        <th scope="col">Autor</th>
                        <th scope="col">Acciones</th>
                </tr>
         </thead>
        <tbody>
                <tr th:each="libro : ${libros}">
                        <td><span th:text="${libro.isbn}"></span></td>
                        <td><span th:text="${libro.titulo}"></span></td>
                        <td><span th:text="${libro.autor.nombre}"></span></td>
                                        <td><a href="#">Agregar</a></td>
                                </tr>
                </tbody>
</table>
```

Valores de InfoAuto

ISBN	Titulo	Autor	Acciones
1238442	Mendoza Tiembla	Martin Rumbo	Reservar
1241232	Mendoza Zombie Attack	Mario Japaz	Reservar

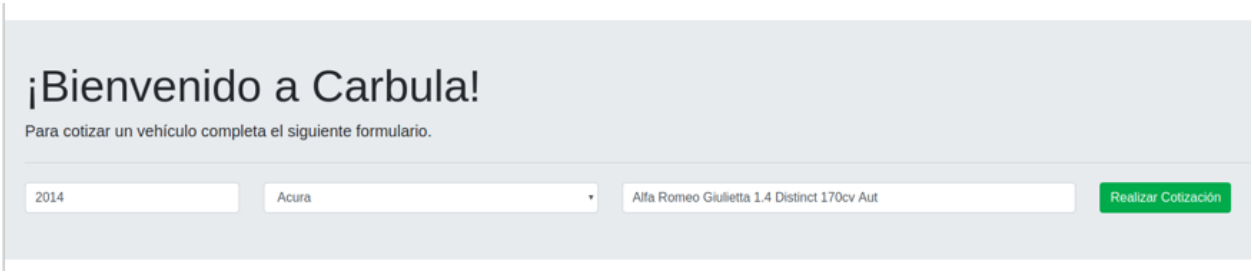
Luego en el lugar que queremos escribir el valor de alguna variable del modelo usamos:

```
<select type="text" class="form-control" id="marca" name="marca">
        <option th:each="marca : ${marcas}" th:text="${marca.nombre}" th:value="${marca.nombre}"></option>
</select>
```



¿Cómo enviar datos a los controladores?

```
<div class="jumbotron">
        <h1 class="display-4">¡Bienvenido a Carbula!</h1>
        <p class="lead">Para cotizar un vehículo completa el siguiente formulario.</p>
        <hr class="my-4">
        <form id="formulario" action="/cotizador/ar/cotizar" class="horizontal-form">
                <div class="row">
                        <div class="col-sm-2">
                                <input type="text" class="form-control" id="anio" name="anio" placeholder="Año del Vehículo" />
                        </div>
                        <div class="col-sm-3">
                                <select type="text" class="form-control" id="marca" name="marca">
                                        <option th:each="marca : ${marcas}" th:text="${marca.nombre}" th:value="${marca.nombre}"></option>
                                </select>
                        </div>
                        <div class="col-sm-4">
                                <input type="text" class="form-control" id="vehiculo" name="vehiculo" placeholder="Modelo y Versión">
                        </div>
                        <div class="col-sm-3">
                                <button type="submit" class="btn btn-success">Realizar Cotización</button>
                        </div>
                </div>
        </form>
</div>
```




```
@GetMapping("/{pais}/cotizar")
public String inicio(@PathVariable String pais, @RequestParam String vehiculo, @RequestParam Integer anio, ModelMap modelo) {

        Map<String,Object> cotizacion = cotizadorService.cotizar(pais, vehiculo, anio);
        modelo.putAll(cotizacion);

        modelo.put("marcas", cotizadorService.buscarMarcas(pais));
        modelo.put("vehiculo", vehiculo);
        modelo.put("anio", anio);

        return "inicio.html";
}
```

Ejemplos con Thymeleaf: usos en formularios y validación

http://acodigo.blogspot.com/2017/04/spring-mvc-thymeleaf-formularios.html

https://code-examples.net/es/q/17cd32a

https://programandoointentandolo.com/2018/12/spring-mvc-con-spring-boot-thymeleaf.html

MATERIAL EXTRA (VIDEOS)

¿Qué es Thymeleaf?

https://www.youtube.com/watch?v=u79dkQxuSv4

Primer Proyecto con Thymeleaf

https://www.youtube.com/watch?v=PJzEpzd4PtA