TAILWIND

¿QUÉ ES?

Tailwind CSS es un framework CSS que permite un desarrollo ágil, basado en clases de utilidad que se pueden aplicar con facilidad en el código HTML y unos flujos de desarrollo que permiten optimizar mucho el peso del código CSS.

Es decir, es un framework que tiene clases para, por ejemplo, el fondo de un elemento, el color del texto o simplemente el margen por la parte de arriba. Pero **no aporta muchos componentes en las que hay muchos estilos preconcebidos**, a diferencia de otros frameworks como Bootstrap. De hecho ofrece muy pocos componentes . En cambio lo que propone es entregar una enorme cantidad de clases de utilidad, atajos a propiedades css, que combinadas en distintas variantes ofrecen prácticamente un número ilimitado de variantes de diseño, que permite una personalización del aspecto realmente única para cada proyecto.

Además Tailwind es una herramienta que se apoya en **PostCSS** para todo lo que es la generación del código CSS. Gracias a PostCSS se alcanza un flujo de desarrollo muy

avanzado, **personalizable**, **ágil** y sobre todo, extremadamente **optimizado**

Sin embargo, hay también desventajas en mi opinión:

- Debes tener un conocimiento alto de css para utilizarlo, ya que en realidad utilizar atajos a propiedades css.
- No trae componentes rápidos de utilizar.
- Utilización de gran conjunto de clases que al final parece que creas estilos en linea. Por lo que puede parecer que los estilos y el html se mezclan.

> **PostCSS** es una herramienta moderna para la gestión del código CSS que permite aumentar la productividad a la vez que libera al desarrollador de trabajos adicionales así como conocimientos técnicos del estado actual del lenguaje y el soporte en los navegadores.

INSTALAR TALWIND

En su página oficial está todo el contenido y tutoriales. Voy a basarme en ella para explicaros como iniciarse en este framework.

La instalación se va a realizar mediante `npm`. Pero previamente tienes que tener instalada nodejs. !!!Instala la versión lts!!!.

Lo primero es crear una carpeta o directorio donde vamos a crear nuestro proyecto. Abrimos una consola en ese directorio e iniciamos el proyecto con el comando `npm init`. Nos va hacer unas cuantas preguntas pero no es necesario que respondamos. Esto nos va a crear en nuestro directorio un fichero `package.json`. Donde podremos configurar **nuestro proyecto e incluir scripts de automatización.**

Ya que, TailWind depende de PostCSS. Este lo vamos a instalar como un plugin de PostCSS. Deberemos instalar los siguientes paquetes de la siguiente forma.

```
npm install -D tailwindcss postcss autoprefixer --save-dev
// -D para que nuestro proyecto dependa de estos módulos en entorno de desarrollo
// --save-dev para que solo los incluya en el directorio en el que se estemos trabajando y no de forma global.
```

Como podemos ver en la siguiente imagen podemos ver las dependencias de nuestro proyecto en la sección de desarrollo. También podemos las versiones de estas.

```
{
  "name": "web-tailwind",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Depurar
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Julio",
  "license": "ISC",
  "devDependencies": {
    "autoprefixer": "^10.4.13",
    "postcss": "^8.4.21",
    "tailwindcss": "^3.2.4"
  }
}
```

Ahora podemos introducir el siguiente comando `npx tailwindcss --help` para ver la versión y la ayuda. Deberemos ejecutarlo con `npx` comprobará si `<comando> o <paquete>` existe en `$PATH,` o en los binarios del proyecto local, y si es así lo ejecutará.

Una de los pasos más importante de en la instalación es introducir las clases en nuestro proyecto. Para ello, vamos a crear una carpeta `src` dentro de la carpeta principal donde guardamos los estilos sin compilar. Y otra carpeta `css` donde se incluirán todos nuestros estilos compilados.

Dentro de `src` vamos a crear un fichero que se llama `styles.css` (el nombre da igual). en el que importamos tailwind. Una modificación este archivo implica una nueva compilación para que se modifique.

Para compilar este archivo

```
npx tailwindcss -i ./src/styles.css -o ./css/styles.css
```

```
// Una opción importante es --watch . Entra en modo watching y la
modificación de ese archivo hace que se compile de nuevo de forma
automática.
```

Si observamos la carpeta `"./css/styles.css"` Vemos que ha creado ese archivo con un montón de clases. Unas 500 clases que pertenecen a `@tailwind base;` y no se han incluido las utilidades ni componentes. Esto sucede porque solo las clases que incluyas en el fichero html o js se incluirán en el fichero css compilado. Para ello, deberemos indicarle cual es contenido que tiene que escanear. Por lo que deberemos generar el fichero de configuración de TailWind (`tailwind.config.js`) con el siguiente comando `npx tailwindcss init` . En la sección content del archivo de configuración nos indica que archivos tiene que escanear y cuál es la dirección.

> **!!OJO!! Configura tu dirección**

Configurar script

Dentro del fichero de configuración `package.json` disponemos de una sección `scripts` . Dentro de ella podemos establecer un conjunto de comandos que nos evitan teclear de nuevo, por ejemplo el comando anterior. De tal forma que crearemos un comando build que nos realizará la compilación, como muestra la siguiente imagen.

```
{
  "name": "web-tailwind",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Depurar
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "build": "tailwindcss -i ./src/styles.css -o ./css/styles.css --watch"
  },
  "author": "Julio",
  "license": "ISC",
  "devDependencies": {
    "autoprefixer": "^10.4.13",
    "postcss": "^8.4.21",
    "tailwindcss": "^3.2.4"
  }
}
```

Como vemos, **no** hemos puesto el comando `npx` y hemos incluido la opción `--watch`. Para ejecutar el nuevo script/comando deberemos abrir una consola en el directorio (mediante vscode) y ejecutar `npm run build`

PRIMEROS PASOS CON TAILWINDS

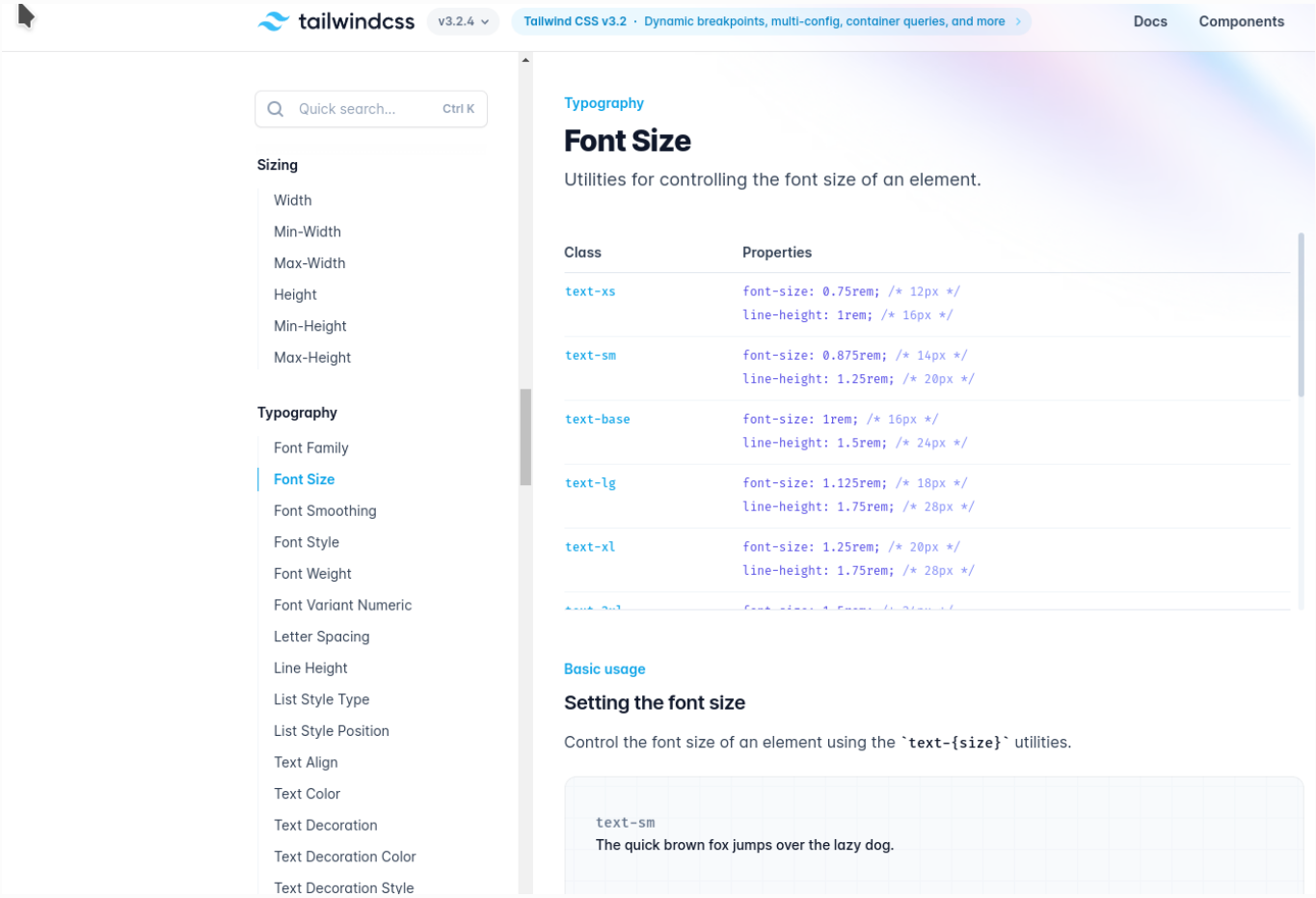
Tenemos instalado tailwind y ahora vamos a utilizarlo en una web ejemplo que vamos a crear. Crea un `index.html` y enlazalo con el fichero `./css/styles.css`.
Si disponemos del siguiente código html.

```
<body>
  <h1>Hola Mundo</h1>
  <ul>
    <li>Manzana</li>
    <li>Pera</li>
    <li>Limón</li>
  </ul>
</body>
```

El resultado lo podemos ver en la siguiente imagen. Por lo que podemos observar Tailwind tiene sus propios estilos reseteados. Para saber que propiedades le incluye, las podemos observar en el inspector del navegador.

A continuación vamos a utilizar las clases que nos proporciona TailWind. Por ejemplo, podemos poner `text-2xl` esta clase la he buscado en la documentación del framework

en la web. Como podemos ver en la siguiente imagen.



Además de la anterior clase, podemos probar a introducir al `H1` algunas de las clases como:

- `bg-black` : Fondo negro.
- `text-white` : Color de texto blanco
- `m-5` : que corresponde con `margin: 1.25rem; /* 20px */`.
- `p-5` : que corresponde con `p-5 padding: 1.25rem; /* 20px */`
- `border-8` : Que corresponde con `border-width: 8px;`
- `text-center` : Centramos horizontalmente el texto.

Pero también podemos dar colores a ese borde. Utilizando la siguiente clase `border-yellow-900`. En el que se le indica en el nombre de la clase el color y la intensidad. Si vamos a la sección de colores de la documentación, disponemos de la paleta de colores que dispone y su intensidad expresada en número.

Además podemos controlar su opacidad mediante `border-indigo-500/50`. El nombre de la clase le indica que es el 50% de opacidad. También podemos indicar que lados queremos bordes y así con un montón de utilidades. Todos estos ejemplos están sacados de la documentación. Revisala para aplicar la clase que te convenga.

En la siguiente imagen podemos ver el resultado de la inclusión de las clases anteriormente comentadas.

TRABAJANDO CON TAILWIND

Ahora vamos a trabajar con la web ejemplo del **tema de responsive**. Esta web la modificaremos y utilizaremos Tailwind para su creación.

Lo primero que vamos a incluir es una imagen de fondo que disponemos en la carpeta `img` . Para añadir una imagen a un elemento **no podemos usar una clase** , ya que no existe una clase para este fin, por lo que necesitamos añadir reglas CSS personalizadas, lo más sencillo es simplemente añadir el CSS personalizado a tu hoja de estilos:

```
@tailwind base;
@tailwind components;
@tailwind utilities;

.my-custom-style {
  background-image: url("../img/pattern.png");
}
/* Esta clase la ponemos en el body. */
```

> Considerar que cada vez que se haga una modificación en este archivo o en el html se compilará de nuevo.

También puedes utilizar la directiva `@layer` para añadir estilos a la `base, de components y de utilities` .

```
@layer base {
  body {
    background-image: url("../img/pattern.png");
  }
}
```

De esta forma hemos añadido nuestros estilos, pero podemos añadir tus propias imágenes de fondo editando la sección t`heme.backgroundImage` del archivo `tailwind.config.js`:

```
module.exports = {
  theme: {
    extend: {
      backgroundImage: {
        "body-pattern": "url('../img/pattern.png')",
      },
    },
  },
};
```

Ahora hemos modificado o más bien hemos añadido una clase a clase `background(bg)`. Por lo que tenemos que poner la clase `bg-body-pattern`.

Modificando la Cabecera.

Ahora vamos a modificar la cabecera que tenemos y añadir un menu en lugar de la dirección que tenemos para ver como funcionan algunas de las clases que dispone este framework. Lo primero que tenemos que hacer es crear el html. Que es el siguiente:

```
<body class="bg-body-pattern">
  <header class="bg-black text-white bebas">
    <div class="container w-11/12 mx-auto flex flex-row">
      <div class="flex-1" id="logo">
        <div
          class="w-72 pt-0.5 pb-0.5 text-center tracking-wider transition
duration-100 rounded-sm"
        >
```

```
          <span class="symbol">S</span>
          <h3>INFORMÁTICA</h3>
        </div>
      </div>
      <nav class="flex-1">
        <ul>
          <li><a href="#">INICIO</a></li>
          <li><a href="#">CONTACTO</a></li>
          <li><a href="#">PERFIL</a></li>
        </ul>
      </nav>
    </div>
  </header>
</body>
```

Ahora vamos a explicar algunos de las clases que se han establecido en el anterior ejemplo, pero como siempre deberéis acudir a la documentación del propio framework.

- `container` : Un componente para fijar la anchura de un elemento. No está centrado por lo que deberemos poner la clase `mx-auto` . Pero, por ejemplo para que ocupe el 91% utilizaremos la clase `w-11/12` (división de 11 entre 12)
- `flex` : Dispone los elementos a flex tomando como dirección la fila con la clase `flex-row` . Además la clase `flex-1` permite que un elemento flexible crezca y se encoja según sea necesario, ignorando su tamaño inicial.
- `tracking-wider` :Utilidad para controlar el espaciado entre letras.
- `transition` es una utilidad para controlar la propiedad `transition` . Además con esta clase `duration-100` controlamos la duración de la transición durante 100ms.
- `rounded-sm` : Utilidad para aplicar redondeo a los bordes.

Crear un color.

Ahora quiero crear un color nuevo que no se encuentra determinado en nuestro framework y aplicarselo al background del logo. Tendremos hacerlo como antes con el background del body. Nos vamos a la configuración y extendemos el tema.

```
module.exports = {
  content: ["./*.{html,js}"],
  theme: {
    extend: {
      backgroundImage: {
        "body-pattern": "url('../img/pattern.png')",
      },
      colors: {
        "azul-claro": "#84b6f4",
```

```
          "azul-oscuro": "#005187",
        },
      },
    },
    plugins: [],
  };
```

Ahora, como antes, la manera de aplicar la clase es mediante `bg-azul-claro`. Pero, también podemos aplicar el color al texto, por ejemplo text-azul-claro. En definitiva a cualquier clase que podamos tener color.

Incrustar fuentes desde archivos.

Podemos ver que en el código html, que hemos utilizado, la clase symbol que no se encuentra dentro de las utilidades. Lo que hemos hecho es incrustar ficheros de fuentes para utilizarlo. Para lo cual, podemos realizarlo modificando el fichero ./src/styles con la directiva `@layer`.

1. Crear un directorio fonts en la carpeta css donde tenemos todos nuestros ficheros de la fuentes correspondientes.
2. Crear las clases correspondientes con `@layer base{}`
3. Ya podemos utilizar las clases. Por ejemplo , `symbol` para la S que es la tuerca o bebas para el `header`. Busca WebSymbolsRegular si quieres cambiarlo.

```
@layer base {
  /*FUENTES*/
  @font-face {
    font-family: "TrebuchetMS";
    src: url("fonts/TrebuchetMS.ttf");
    font-weight: normal;
    font-style: normal;
  }

  @font-face {
    font-family: "BebasNeue";
    src: url("fonts/BebasNeue.otf");
    font-weight: normal;
    font-style: normal;
  }

  @font-face {
    font-family: "WebSymbolsRegular";
    src: url("fonts/websymbols-regular-webfont.eot");
    src: url("fonts/websymbols-regular-webfont.eot?#iefix")
```

```
  format("embedded-opentype"),
        url("fonts/websymbols-regular-webfont.woff") format("woff"),
        url("fonts/websymbols-regular-webfont.ttf") format("truetype"),
  url("fonts/websymbols-regular-webfont.svg#WebSymbolsRegular")
          format("svg");
      font-weight: normal;
      font-style: normal;
    }

    .symbol {
      font-family: "WebSymbolsRegular";
    }
    .bebas {
      font-family: "BebasNeue";
    }
    .trebuchet {
      font-family: "TrebuchetMS";
    }
  }
```

Ahora vamos a aprender como utilizar las clases hover. Nuestro objetivo es que una vez que pasemos el ratón por encima cambie de color el texto y el fondo blanco. Para ello, deberemos utilizar la clase `hover:text-black hover:bg-white`.

También disponemos de utilidades como float, las vamos a utilizar en el logo de la siguiente manera:

```
<span class="symbol block float-left text-3xl mt-1 ml-11">S</span>
<h3 class="block float-right text-4xl mt-1.5 mr-14">INFORMÁTICA</h3>
```

Ten en cuenta que si utilizamos el float salen del flujo normal y si no ponemos la clase `overflow-hidden` en el contenedor saldrán del contenedor. Otra forma de arreglarlo es mediante la creación de un elemento clearfix.

Crear componentes

Ahora vamos a aprender a crear componentes. En nuestro fichero ./src/styles.scss vamos a a añadir clases a components. mediante el siguiente código.

```
@layer components {
  .menu-item {
    @apply mr-5 ml-5 tracking-wider;
  }
  .menu-item-a {
    @apply block transition duration-200 hover:text-azul-oscuro transform
hover:scale-125;
  }
}
```

Hemos creado dos clases que se van a reutilizar mucho y con la regla `@apply` le aplicamos las utilidades que necesitemos.

```
<nav class="flex-1">
  <ul
    class="flex flex-row h-20 items-center justify-end text-2xl text-
center mr-6"
  >
    <li class="menu-item">
      <a class="menu-item-a" href="#">INICIO</a>
    </li>
    <li class="menu-item">
      <a class="menu-item-a" href="#">CONTACTO</a>
    </li>
    <li class="menu-item">
      <a class="menu-item-a" href="#">PERFIL</a>
    </li>
  </ul>
</nav>
```

Además podemos observar que hemos utilizado algunas utilidades nuevas de flex como `items-center` que alinea verticalmente.

Crear animaciones.

Si vamos a la documentación tenemos animaciones básicas. Vamos a utilizar `animate-spin` para hacer que la tuerca de vueltas y modificaremos su comportamiento.

```
<span class=" animate-spin symbol block float-left text-3xl mt-1 ml-
11">S</span>
```

Como podemos observa la rueda va muy rápido. Podemos pensar que con la clase duration podemos modificar el comportamiento pero en la documentación tenemos que extender el tema para modificar una animación que ya está creada , crear una nueva o incluir nuevos keyframes. Vamos a modificar la animación spin.

```
module.exports = {
  content: ["./*.{html,js}"],
  theme: {
    extend: {
      backgroundImage: {
        "body-pattern": "url('../img/pattern.png')",
      },
      colors: {
        "azul-claro": "#84b6f4",
        "azul-oscuro": "#005187",
      },
      animation: {
        "spin-slow": "spin 3s linear infinite",
        miWiggle: "wiggle 2s linear infinite",
      },
      keyframes: {
        wiggle: {
          "0%, 100%": { transform: "rotate(-3deg)" },
          "50%": { transform: "rotate(3deg)" },
        },
      },
    },
  },
  plugins: [],
};
```

Ahora lo que vamos hacer es que cuando hagamos un `hover` sobre sobre cualquier parte del logo (padre), la animación de la tuerca se pare. En tailwind, tenemos introducir la clase `group` al padre y y utilizar modificadores `group-*` como group-hover para dar estilo al elemento. Si vemos la documentación con solo poner `group-hover:animate-none` sobre la tuerca debería pararse.

```
 <div class="group overflow-hidden w-72 pt-0.5 pb-0.5 text-center
tracking-wider transition duration-100 rounded-sm bg-azul-claro
```

```
hover:text-black hover: hover:bg-white"
          >
            <span
              class="symbol animate-spin-slow group-hover:animate-none
block float-left text-3xl mt-1 ml-11"
              >S</span
            >
            <div class="invisible group-hover:visible">I'm here!</div>

            <h3
              class="block group-hover:text-lg group-hover:animate-
miWiggle float-right text-4xl mt-1.5 mr-14"
            >
              INFORMÁTICA
            </h3>
          </div>
        </div>
```

Como hemos visto podemos modificar las animations que ya están predefinidas. Pero también podemos realizar nuestras animaciones. En este caso vamos hacer una animación sobre el Informática en la que hemos generado un keyframe diferente. Observa el archivo de configuración.

Resposive design

Ahora vamos a adaptar esa cabecera a los diferentes breakpoints que tiene establecido y veremos como trabajar con responsive. Que podemos ver aquí. Como podemos ver en ejemplo hemos puesto un borde para cada uno de los breakpoints.

```
<header
  class="text-white bebas bg-black border-8 border-red-600 sm:border-
yellow-400 md:border-green-600 lg:border-blue-600 xl:border-indigo-900
2xl:border-b-gray-700"
>
  ```
</header>
```

Además podemos personalizar nuestros breakpoints en el archivo de configuración. Como podemos ver el archivo de configuración. Podemos utilizar la función `min y max` . **Por defecto, si no ponemos ni `min` ni `max` es `min` .**
```

```
screens: {
        xs: { max: "768px" },
        // => @media (max-width: 768px) { ... }
         xs: {'min': '640px', 'max': '767px'},
      // => @media (min-width: 640px and max-width: 767px) { ... }
      },
```

Vamos a diseñar nuestro menú de tal forma que el `nav` en cuando llegue a `xs` se disponga debajo. Para lo cual, como hemos visto tenemos que establecer las clases correspondientes con el breakpoint asociado.

```
<div
  class="container xs:flex-col w-11/12 mx-auto flex flex-row items-center"
></div>
```

Eso lo hemos conseguido con la clase `xs:flex-col`, que como sabemos, cambia la dirección a columnas. También vamos a procurar que los elementos del nav ocupen todos el mismo espacio, para ello tenemos que añadir `xs:flex-1` pero como lo tenemos como una componente lo debemos incluir en el apply.

```
@layer components {
  .menu-item {
    @apply mr-5
      ml-5
      tracking-wider
      xs:flex-1;
  }
```

Creación de Main

Ahora vamos a crear el main con los artículos y el aside. El HTML lo vamos a cambiar con el objetivo de utilizar más utilidades.
Ya que, toda la teoría esta comentada con los ejemplos anteriores.

Comenzaremos con el siguiente código html después del `header` en donde hemos estado practicando.

```
   <main class="container w-11/12 mx-auto mt-5 flex flex-row">
      <aside id="lateral" class="order-2 basis-1/5">
        <h3 class="aside-header">Buscar</h3>
        <div
          id="search"
          class="w-11/12 h-7 m-2.5 mx-auto bg-white border border-gray-200
rounded shadow-inner"
        >
          <form action="">
            <input
              class="w-10/12 h-6 border-none pl-1.5 rounded-md bg-
transparent text-gray-500 transition duration-300 outline-none
focus:outline-none"
              type="text"
              name=""
              id=""
            /><input
              type="button"
              value="L"
              class="symbol h-6 cursor-pointer text-base bg-transparent
border-none text-gray-500 pl-2"
            />
          </form>
        </div>
        <h3 class="aside-header">LOGIN</h3>
        <div id="login">
          <form action="">
            <label class="login-icon" for="">U</label
            ><input class="login-input" type="email" name="" id="" />
            <label class="login-icon mt-7" for="password">w</label>
            <input class="login-input" type="password" name="password"
id="" />
            <input class="login-button" type="submit" value="Entrar" />
            <input class="login-button ml-1" type="reset"
value="Reiniciar" />
            <a class="login-link" href="#"> Registrate aquí</a>
            <a class="login-link" href="#"> ¿Has olvidado la Constraseña?
</a>
          </form>
        </div>
      </aside>
      <!-- FIN DE LA BARRA lateral -->

      <section class="basis-4/5 mr-10">
        <h2 class="articles-header">Últimos artículos</h2>
        <article class="article-item">
          <div class="article-data">
```

```
            <span class="m-2.5">Fecha:10</span
            ><span class="m-2.5">Categorías</span>
          </div>
          <h4 class="article-title">
            <a href="#">Título del artículo</a>
          </h4>
          <figure class="article-figure">
            <img
              class=""
              src="./img/senthil-kumar-moorthy-bn5Upp6c5L4-unsplash
(1).jpg "
              alt="Tigre"
            />
          </figure>
          <p class="article-description">
            Lorem ipsum dolor sit amet consectetur adipisicing elit. Est,
            obcaecati accusamus facilis libero eveniet nemo architecto
            recusandae excepturi optio explicabo dolore voluptate quasi
numquam
            dolorem repudiandae expedita similique saepe deserunt?
          </p>
        </article>

        <article class="article-item">
          <div class="article-data">
            <span class="m-2.5">Fecha:10</span
            ><span class="m-2.5">Categorías</span>
          </div>
          <h4 class="article-title"><a href="#">Título del artículo</a>
</h4>
          <figure class="article-figure">
            <img
              class=""
              src="./img/birger-strahl-bmjp_dwpoU8-unsplash.jpg"
              alt="Tigre"
            />
          </figure>
          <p class="article-description">
            Lorem ipsum dolor sit amet consectetur adipisicing elit. Est,
            obcaecati accusamus facilis libero eveniet nemo architecto
            recusandae excepturi optio explicabo dolore voluptate quasi
numquam
            dolorem repudiandae expedita similique saepe deserunt?
          </p>
        </article>

        <article class="article-item">
          <div class="article-data">
            <span class="m-2.5">Fecha:10</span
            ><span class="m-2.5">Categorías</span>
          </div>
          <h4 class="article-title"><a href="#">Título del artículo</a>
</h4>
          <p>
```

```
            Lorem ipsum dolor sit amet consectetur adipisicing elit. Est,
            obcaecati accusamus facilis libero eveniet nemo architecto
            recusandae excepturi optio explicabo dolore voluptate quasi
numquam
            dolorem repudiandae expedita similique saepe deserunt?
          </p>
        </article>
      </section>
    </main>
    <footer
      id="footer""
    >
      <div id="container >
        <div id="info" class="footer-box">
          <h5 class="footer-header">Desarrollado con</h5>
          <p class="mb-6">
            <img
              class="block mx-auto"
              src="img/html5-badge-h-css3-graphics-multimedia-performance-
semantics.png"
            />
          </p>
        </div>
      </div>
    </footer>
```

Lo pimero que vamos hacer es que nuestro main (contededor) tenga las siguientes clases. con el objetivo de tener un contenedor con 91.6666667% del tamaño y centrado. Además pondremos que son elementos flex para posicionar los elementos hijos (aside y section).

```
<main class="container w-11/12 flex flex-row mx-auto"></main>
```

Además vemos que el aside tiene que estar en el otro lado por lo que le asignamos la siguiente clase (`order-2`).Ahora lo que queremos es establecer un ancho en el aside del 25% por lo que añadimos el siguiente tamaño con la clase `basis-1/5`.

```
<aside id="lateral" class="order-2 basis-1/5 mt-5"></aside>
```

Para identificar a la section con los articulos le he puesto un borde como veis y le indicado con `basis-4/5` que sea un 75%.

```
<section class="border-4 border-blue-600 basis-4/5 mr-10"></section>
```

Como podéis ver he utilizado la propiedad `flex-basis` de flex. También podríamos haber utilizado el `width` con porcentajes con con rems pero último supone que la página no sea fluida.

Vamos a tocar,ahora, el título de los artículos. Creamos una clase en components que sea `articles-header` con sombra. De la siguiente forma.
En el archivo de configuración podemos establecer la sombra y en el archivo styles.scss podemos crear la componente y aplicar el estilo que hemos creado.

```
<h2 class="articles-header">Últimos artículos</h2>
```

```
// Fichero de configuración

boxShadow: {
        header3D:
          "0px 1px #393d3f,1px 2px 0px #393d3f,2px 3px 0px #393d3f,3px 4px
0px #393d3f",
      },
```

```
/* styles.css */
.articles-header {
  font-family: "BebasNeue";
  @apply w-full h-16 text-center text-5xl tracking-widest pt-4 bg-slate-50
shadow-header3D font-medium;
}
```

Para continuar con el tutorial y para hacer la actividad os voy a dejar la sombra que se le aplicado al login.

```
boxShadow: {
        box: "0px 0px 1px rgba(0,0,0,0.3),0px 3px 7px rgba(0,0,0,0.3),0px
1px white inset,0px -3px 1px rgba(0,0,0,0.3) inset",
      }
```