Tailwind CSS

https://tailwindcss.com/
https://github.com/tailwindlabs

Resumen de todas las clases:
https://www.creative-tim.com/twcomponents/cheatsheet/

Colores y degradados:
https://ui.shadcn.com/colors
https://tailscan.com/colors
https://tailscan.com/gradients

Herramienta para diseñar grid:
https://tailscan.com/resources/grids

Para facilitar clases responsive:
https://fluid.tw/

Instalación

Tailwind se puede instalar como un plugin de Vite o de PostCSS, entre otras maneras.

Vite

```
npm i tailwindcss @tailwindcss/vite
```

Luego hay que configurar el plugin:

vite.config.ts

```
import { defineConfig } from 'vite'
import tailwindcss from '@tailwindcss/vite'

export default defineConfig({
  plugins: [
    tailwindcss(),
  ],
})
```

PostCSS

Para usarlo en **Next.js** hay que instalar Tailwind como plugin para PostCSS:

```
npm i tailwindcss @tailwindcss/postcss postcss
```

Y lo añadimos en la configuración de PostCSS:

postcss.config.mjs

```
export default {
  plugins: {
    "@tailwindcss/postcss": {},
  }
}
```

Importar

En ambos casos importamos `tailwindcss` en el archivo de estilos `.css` (`globals.css` en Next.js):

```
@import "tailwindcss";
```

Cuando se importa `tailwindcss` estamos importando lo siguiente:

tailwind/index.css

```
@layer theme, base, components, utilities;

@import './theme.css' layer(theme);
@import './preflight.css' layer(base);
@import './utilities.css' layer(utilities);
```

En `theme.css` se definen las variables que configuran las clases de Tailwind (colores, breakpoints, fuentes, tamaños, aceleración de animaciones, etc.).
En `preflight.css` se hace un reseteo inicial.
`utilities.css` está vacío. (https://tailwindcss.com/docs/adding-custom-styles#adding-custom-utilities)

@theme

Tailwind: Theme variables

A partir de la versión 4 no hay un archivo de configuración, sino que usamos la directiva `@theme` en el archivo de estilos `.css` tanto para extender como para sobreescribir la configuración inicial.

Extender configuración

Para añadir nuevos colores creamos una variable CSS con el prefijo `--color`. Esto generará automáticamente clases con ese color. Por ejemplo, si creamos la variable `--color-mint-500` podremos usar clases `bg-mint-500`, etc.
Creamos tipografías con el prefijo `--font`, por ejemplo, `--font-poppins`.
Tamaños responsive, con el prefijo `--breakpoint`.

styles.css

```
@import "tailwindcss";

@theme {
  --color-mint-500: --color-mint-500: oklch(0.72 0.11 178);
  --font-poppins: Poppins, sans-serif;
  --breakpoint-3xl: 120rem;
}
```

Los prefijos que generan clases nuevas son los siguientes:
`--color-*`, `--font-*`, `--text-*`, `--font-weight-*`, `--tracking-*`, `--leading-*`, `--breakpoint-*`, `--container-*`, `--spacing-*`, `--radius-*`, `--shadow-*`, `--inset-shadow-*`, `--drop-shadow-*`, `--blur-*`, `--perspective-*`, `--aspect-*`, `--ease-*`, `--animate-*`.

Sobreescribir configuración

Para cancelar la configuración inicial asignamos el valor `initial` a las variables CSS. Por ejemplo:

```
@theme {
  --color-*: initial;
}
```

El ejemplo anterior elimina los colores por defecto de Tailwind. Para borrar TODOS los ajustes por defecto usamos `--*: initial;`.

Configuración inicial

Podemos comprobar el reset inicial (preflight) de Tailwind en
https://github.com/tailwindlabs/tailwindcss/blob/main/packages/tailwindcss/preflight.css

Y la configuración inicial del tema en
https://github.com/tailwindlabs/tailwindcss/blob/main/packages/tailwindcss/theme.css

Nombres de las clases

Tailwind: Detecting classes in source files

Los nombres de las clases deben ser strings completos. No se puede interpolar entre medias de un string, porque entonces Tailwind no lo reconocerá como el nombre de una clase.

```
class="text-{{ error ? 'red' : 'green' }}-600"              // Así no

class="{{ error ? 'text-red-600' : 'text-green-600' }}"     // Esto sí
```

Esto quiere decir que tampoco se pueden usar props de un componente de React para crear el nombre de las clases:

```
// Así no:

function Button({ color, children }) {
  return (
    <button className={`bg-${color}-600 hover:bg-${color}-500 ...`}>
      {children}
    </button>
  )
}


// Esto sí:

function Button({ color, children }) {

  const colorVariants = {
    blue: 'bg-blue-600 hover:bg-blue-500',
    red: 'bg-red-600 hover:bg-red-500',
  }

  return (
    <button className={`${colorVariants[color]} ...`}>
      {children}
    </button>
  )
}
```

Clases dinámicas

Si hacen falta clases dinámicas se puede recurrir a variables CSS. Por ejemplo:

```
<div
  className="bg-[var(--bg-color)]"
  style={{ '--bg-color': `rgb(${color.r} ${color.g} ${color.b})`}}
/>
```

Un ejemplo con React:

```
const [isCollapsed, setIsCollapsed] = useState(false);

<div
  style={{ "--width": isCollapsed ? "8rem" : "16rem" }}
  className="w-[--width]"
/>
```

También se puede conseguir lo mismo con atributos `data` y selectores de atributo:

```
<div
  data-collapsed={isCollapsed}
  className="w-32 data-[collapsed=false]:w-64"
/>
```

Selectores anidados

Un elemento puede hacer referencia a la clase de un elemento hijo con `&`.
Por ejemplo:

```
const [isCollapsed, setIsCollapsed] = useState(false);

<div
  data-collapsed={isCollapsed}
  className="[&[data-collapsed=true]_svg]:rotate-180"
>
  <div>
    <ChevronUp size={24} />                                  // Renderiza un svg
  </div>
</div>
```

También se puede hacer de otra manera:

```
<div data-collapsed={isCollapsed}>
  <div>
    <ChevronUp size={24} className="[[data-collapsed=true]_&]:rotate-180" />
  </div>
</div>
```

Más trucos avanzados: https://www.youtube.com/watch?v=9z2Ifq-OPEI

Colores

Valores especiales (5)

`white`, `black`, `transparent`, `current` e `inherit`.

Colores neutros (5)

`slate`, `gray`, `zinc`, `neutral` y `stone`.

Colores normales (17)

`red`, `orange`, `amber`, `yellow`, `lime`, `green`, `emerald`, `teal`, `cyan`, `sky`, `blue`, `indigo`, `violet`, `purple`, `fuchsia`, `pink` y `rose`.

Los 5 colores neutros y los 17 normales tienen 11 variantes en una escala desde 50 hasta 950. Por ejemplo: `red-50`, `red-100`, `red-200`, `red-300`, `red-400`, `red-500`, `red-600`, `red-700`, `red-800`, `red-900`, `red-950`.

Todos los colores se pueden usar en las siguientes clases:

```
bg-*              background-color
text-*            color
decoration-*      Sets the text decoration color of an element
border-*          border-color
outline-*         Sets the outline color of an element
shadow-*          Sets the color of box shadows
inset-shadow-*    Sets the color of inset box shadows
ring-*            Sets the color of ring shadows
inset-ring-*      Sets the color of inset ring shadows
accent-*          Sets the accent color of form controls
caret-*           Sets the caret color in form controls
fill-*            Sets the fill color of SVG elements
stroke-*          Sets the stroke color of SVG elements
```

Por ejemplo, `bg-stone-200`, `accent-amber-700`, etc.

Opacidad

Para ajustar la opacidad añadimos una barra `/` y un número entre `0` y `100`.
Por ejemplo, `bg-red-400/75` para 75% de opacidad.

Notas sobre algunas clases de Tailwind

size

Para aplicar `width` y `height` se usan las clases `w-*` y `h-*`. Pero si las dos medidas van a ser iguales se puede usar `size-*`.

divide

> Tailwind: divide-width | divide-color | divide-style

Las clases `divide-x` y `divide-y` añaden un borde de 1px horizontal o vertical entre los elementos.
La clase se aplica al contenedor. Además, hay que especificar un color, por ejemplo: `divide-violet-600`.
Se pueden usar bordes de otros tamaños, por ejemplo: `divide-x-2`, `divide-y-4`, etc.
Por defecto el estilo del borde es `solid`, pero se puede usar `divide-dashed`, `-dotted`, `-double` o `-none`.

space

> Tailwind: space

Las clases `space-x-*` y `space-y-*` añaden un margen horizontal o vertical entre los elementos, igual que `gap`. La clase se aplica al contenedor y se usa con alguna medida, por ejemplo, `space-y-4`.

ring

Las clases `ring-*` simulan bordes con `box-shadow`. Es distinto a las propiedades `border` y `outline`.
Se usan para añadir bordes adicionales cuando no se puede hacer con `border` o `outline`.

border

La clase `border` aplica `border-width: 1px`. Para otras medidas se usan `border-2`, `border-4`, etc.
El color por defecto es `currentColor`. Para colorear el borde se añade una clase `border-*` seguida de un color: `border-black`, `border-slate-200`, etc.
¿Por defecto las clases `border` aplican `border-style: solid`, así que no hace falta especificarlo?

rounded (border-radius)

Con las clases `rounded-*` se especifica el `border-radius`.

```
rounded-none      border-radius: 0px;
rounded-xs        border-radius: 0.125rem;
rounded-sm        border-radius: 0.25rem;
rounded-md        border-radius: 0.375rem;
rounded-lg        border-radius: 0.5rem;
rounded-xl        border-radius: 0.75rem;
rounded-2xl       border-radius: 1rem;
rounded-3xl       border-radius: 1.5rem;
rounded-full      border-radius: 9999px;
```

Se pueden añadir modificadores para especificar el lado del borde o la esquina concreta: `-s` (start), `-e` (end), `-t` (top), `-b` (bottom), `-r` (right), `-l` (left). Por ejemplo, `rounded-bl-*` para la propiedad `border-bottom-left-radius`.

shadow

Con las clases `shadow-*` se añade una sombra negra de baja opacidad y distintos tamaños:

```
shadow-2xs box-shadow:   0 1px rgb(0 0 0 / 0.05);

shadow-xs  box-shadow:   0 1px 2px 0 rgb(0 0 0 / 0.05);

shadow-sm  box-shadow:   0 1px 3px 0 rgb(0 0 0 / 0.1),
                         0 1px 2px -1px rgb(0 0 0 / 0.1);

shadow-md  box-shadow:   0 4px 6px -1px rgb(0 0 0 / 0.1),
                         0 2px 4px -2px rgb(0 0 0 / 0.1);

shadow-lg  box-shadow:   0 10px 15px -3px rgb(0 0 0 / 0.1),
                         0 4px 6px -4px rgb(0 0 0 / 0.1);

shadow-xl  box-shadow:   0 20px 25px -5px rgb(0 0 0 / 0.1),
                         0 8px 10px -6px rgb(0 0 0 / 0.1);

shadow-2xl box-shadow:   0 25px 50px -12px rgb(0 0 0 / 0.25);

shadow-inner             box-shadow: inset 0 2px 4px 0 rgb(0 0 0 / 0.05);

shadow-none              box-shadow: 0 0 #0000;
```

Para modificar el color hay que añadir, además, una clase `shadow-*` con el color deseado.
Por ejemplo: `shadow shadow-slate-400`, `shadow-lg shadow-gray-500`, `shadow-sm shadow-current`, etc.

Escala de tamaños

Tailwind tiene una escala de espacios por defecto, desde `0` hasta `24rem`. 1 espacio equivale a `4px`.
Las clases `m`, `p`, `w`, `min-w`, `max-w`, `h`, `min-h`, `max-h`, `gap`, `inset`, `space`, `translate-x`, `translate-y`, `scroll-m` y `scroll-p` usan esta escala de tamaños por defecto.
Por ejemplo, `m-2` añade un margen de `0.5rem`, `w-96` establece una anchura de `384px`.

Name	Size	Pixels
0	0px	0px
px	1px	1px
0.5	0.125rem	2px
1	0.25rem	4px
1.5	0.375rem	6px
2	0.5rem	8px
2.5	0.625rem	10px
3	0.75rem	12px
3.5	0.875rem	14px
4	1rem	16px
5	1.25rem	20px
6	1.5rem	24px
7	1.75rem	28px
8	2rem	32px
9	2.25rem	36px
10	2.5rem	40px
11	2.75rem	44px
12	3rem	48px
14	3.5rem	56px
16	4rem	64px
20	5rem	80px
24	6rem	96px
28	7rem	112px
32	8rem	128px
36	9rem	144px
40	10rem	160px
44	11rem	176px
48	12rem	192px
52	13rem	208px
56	14rem	224px
60	15rem	240px
64	16rem	256px
72	18rem	288px
80	20rem	320px
96	24rem	384px

Algunas clases, como `max-w`, permiten usar nombres adicionales para tamaños grandes:

Name	Size	Pixels
xs	20rem	320px
sm	24rem	384px
md	28rem	448px
lg	32rem	512px
xl	36rem	576px
screen-sm	40rem	640px
2xl	42rem	672px
3xl / screen-md	48rem	768px
4xl	56rem	896px
5xl / screen-lg	64rem	1024px
6xl	72rem	1152px
7xl / screen-xl	80rem	1280px
screen-2xl	96rem	1536px

Breakpoints y media queries

Tailwind usa los siguientes breakpoints por defecto:

```
sm        640px
md        768px
lg        1024px
xl        1280px
2xl       1536px
```

Estos breakpoints se usan para aplicar media queries de tipo `min-width`.
Por ejemplo, `sm:text-red-400` aplica un texto rojo si la pantalla mide al menos 640px.
Para aplicar clases a pantallas pequeñas no ponemos ningún prefijo.

```
text-red-400            // Todas las pantallas desde 0 hasta 640px
sm:text-red-400         // Pantalla mayor que 640px
md:text-red-400         // Pantalla mayor que 768px
lg:text-red-400         // Pantalla mayor que 1024px
xl:text-red-400         // Pantalla mayor que 1280px
2xl:text-red-400        // Pantalla mayor que 1536px
```

container

Tailwind: container

La clase `container` aplica un `width` o un `max-width` según el tamaño de pantalla.
Es decir, ajusta el `max-width` de un elemento al `min-width` del breakpoint que esté activo.
Como no se centra por defecto, habrá que aplicarle un `mx-auto`.

Para modificar la clase `container` usaremos la directiva `@utility`:

```
@utility container {
  margin-inline: auto;
  padding-inline: 2rem;
}
```

Pseudoclases

Tailwind: Pseudo-class reference

Las pseudoclases se aplican con prefijos separados por dos puntos. Por ejemplo, `hover:`, `active:`, `focus:`, `first:`, `last:`, `odd:`, `even:`, `disabled:`, `required:`, `invalid:`, etc.

Un ejemplo con `hover:`:

```
<button class="bg-sky-500 hover:bg-sky-700 …"> … </button>

// equivale a

.btn-primary {
  background-color: #0ea5e9;
}

.btn-primary:hover {
  background-color: #0369a1;
}
```

group

Si el estilo de un elemento depende del estado de un elemento ancestro, hay que marcar el ancestro con `group` y usar el prefijo `group-` en los hijos: `group-hover:`, `group-focus:`, etc. Por ejemplo:

```
<div class="group …">
  <h1 class="group-hover:text-white"> … </h1>
  <p class="group-hover:text-black"> … </p>
</div>
```

Si hay varios grupos el nombre se puede personalizar, por ejemplo, `group/edit`, `group/item`, etc.

peer

Si el estilo depende del estado de un elemento hermano anterior (hermano mayor), en lugar de `group` se usa `peer`.

Estilos de texto

Las clases `text-*` se utilizan para especificar las propiedades de CSS `font-size`, `color`, `text-align`, `text-overflow` y `text-wrap`.
Las clases `font-*` especifican las propiedades de CSS `font-family` y `font-weight`.

text-* (font-size)

El tamaño de fuente se especifica con las clases `text-*`, que además aplican un `line-height` determinado. El `line-height` se puede personalizar con un sufijo: `text-xl/8`, `text-base/6`, etc.
Además, el `line-height` se puede personalizar con las clases `leading`.

```
text-xs           0.75rem       12px
text-sm           0.875rem      14px
text-base         1rem          16px
text-lg           1.125rem      18px
text-xl           1.25rem       20px
text-2xl          1.5rem        24px
text-3xl          1.875rem      30px
text-4xl          2.25rem       36px
text-5xl          3rem          48px
text-6xl          3.75rem       60px
text-7xl          4.5rem        72px
text-8xl          6rem          96px
text-9xl          8rem          128px
```

text-* (color)

El color del texto se determina con las clases `text-*`, que admite todos los colores válidos de Tailwind, además de `inherit`, `current` y `transparent`.

```
text-black              color: rgb(0 0 0);
text-white              color: rgb(255 255 255);
text-blue-800           color: rgb(30 64 175);
text-inherit            color: inherit;
text-current            color: currentColor;
text-transparent        color: transparent;
```

text-* (text-align)

```
text-left               text-align: left;
text-center             text-align: center;
text-right              text-align: right;
text-justify            text-align: justify;
text-start              text-align: start;
text-end                text-align: end;
```

font-* (font-weight)

```
font-thin               font-weight: 100;
font-extralight         font-weight: 200;
font-light              font-weight: 300;
font-normal             font-weight: 400;
font-medium             font-weight: 500;
font-semibold           font-weight: 600;
font-bold               font-weight: 700;
```

```
font-extrabold          font-weight: 800;
font-black              font-weight: 900;
```

font-* (font-style)

Con las clases `italic` y `non-italic` se controla el texto en cursiva.

```
italic                  font-style: italic;
non-italic              font-style: normal;
```

tailwind-merge, clsx, cn

```
npm i tailwind-merge

npm i clsx
```

tailwind-merge

GitHub: tailwind-merge

tailwind-merge permite fusionar clases. Tiene dos funciones: `twJoin` y `twMerge`. `twMerge` resuelve conflictos cuando se usan clases que se pisan, pero `twJoin` no. Un ejemplo de uso con `twMerge`:

```
import { twMerge } from 'tailwind-merge'

<div
  className={twMerge(
    TYPOGRAPHY_STYLES_LABEL_SMALL,
    'grid w-max gap-2',
    forceHover ? 'bg-gray-200' : ['bg-white', !disabled && 'hover:bg-gray-200'],
    isMuted && 'text-gray-600',
  )}
/>
```

clsx

GitHub: clsx

clsx ayuda a construir clases condicionalmente. Permite usar una sintaxis de objeto, algo que tailwind-merge no puede hacer.
Por ejemplo, podemos resolver un condicional con sintaxis de objeto:

```
className={cn(
  "bg-teal-400",
  { "opacity-50 cursor-not-allowed": props.disabled },
  "px-2 py-1"
)}
```

cn

cn es una utilidad que permite usar clsx dentro de tailwind-merge. No se instala, simplemente se exporta esta función y listo:

lib/utils.ts

```
import { twMerge } from 'tailwind-merge'
import { clsx, ClassValue } from 'clsx'

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs))
}
```

Class Variance Authority

https://cva.style/docs

Transiciones

Las clases `transition` y `transition-all` aplican transiciones a múltiples propiedades.
Tienen una duración de `150ms` y una curva de aceleración de tipo `ease-in-out`.

```
transition-all          transition-property: all;
                        transition-timing-function: cubic-bezier(0.4, 0, 0.2, 1);
                        transition-duration: 150ms;

transition              transition-property: color, background-color, border-color,
                          text-decoration-color, fill, stroke, opacity, box-shadow,
                          transform, filter, backdrop-filter;
                        transition-timing-function: cubic-bezier(0.4, 0, 0.2, 1);
                        transition-duration: 150ms;
```

Si queremos ser más específicos podemos usar las clases `transition-colors`, `transition-opacity`, `transition-transform` y `transition-shadow`. También tienen la misma duración y curva de aceleración:

```
transition-colors       transition-property: color, background-color, border-color,
                          text-decoration-color, fill, stroke;

transition-opacity      transition-property: opacity;

transition-transform    transition-property: transform;

transition-shadow       transition-property: box-shadow;
```

Por defecto la duración de las transiciones es de `150ms` pero se puede modificar con la clase `duration-*`:

```
duration-0              transition-duration: 0s;
duration-75             transition-duration: 75ms;
duration-100            transition-duration: 100ms;
duration-150            transition-duration: 150ms;
duration-200            transition-duration: 200ms;
duration-300            transition-duration: 300ms;
duration-500            transition-duration: 500ms;
duration-700            transition-duration: 700ms;
duration-1000           transition-duration: 1000ms;
```

Con la clase `delay-*` se puede controlar el retraso. Admite las mismas duraciones en milisegundos entre `0` y `1000` que `duration-*`. Por ejemplo: `delay-0`, `delay-200`, `delay-1000`, etc.

La curva de aceleración también se puede modificar:

```
ease-linear             transition-timing-function: linear;
ease-in                 transition-timing-function: cubic-bezier(0.4, 0, 1, 1);
ease-out                transition-timing-function: cubic-bezier(0, 0, 0.2, 1);
ease-in-out             transition-timing-function: cubic-bezier(0.4, 0, 0.2, 1);
```

Animaciones

Animaciones predefinidas

`animate-spin`, `animate-ping`, `animate-pulse`, `animate-bounce`.
Sin animaciones: `animate-none`.

Animaciones personalizadas

Para añadir una nueva animación la declaramos en `extend.animation`. Podemos crear nuevas animaciones a partir de las animaciones predefinidas. Por ejemplo, aquí estamos creando `spin-slow` a partir de `spin` pero con otra duración:

tailwind.config.js

```
extend: {
  animation: {
    "spin-slow": "spin 10s linear infinite",
  }
}
```

Otra manera es hacerlo con valores arbitrarios entre corchetes:

```
<div className="animate-[spin_10s_linear_infinite]" />
```

También podemos hacer animaciones desde cero declarando los keyframes:

tailwind.config.js

```
extend: {
  keyframes: {
    my-custom-keyframes: {
      "0%, 100%": { transform: "rotate(-10deg)" },
      "50%": { transform: "rotate(10deg)" },
    }
  },
  animation: {
    "my-custom-animation": "my-custom-keyframes 1s linear infinite",
  }
}
```

Para un mayor control se recomienda usar algún plugin:
https://github.com/jamiebuilds/tailwindcss-animate
https://github.com/new-data-services/tailwindcss-animated

Plugins

Tailwind Labs: Typography | Forms

typography

El plugin oficial Typography ayuda a formatear texto sin necesidad de indicar clases a cada elemento. Muy útil para formatear markdown o contenido de un CMS.

forms

Este otro plugin resetea los estilos de los elementos de los formularios.

Ordenar clases

Prettier plugin | ESLint plugin

Para que se ordenen las clases hay dos plugins. Elegir uno de los dos:

prettier-plugin-tailwindcss

Para instalarlo:

```
npm install -D prettier-plugin-tailwindcss
```

Y luego se añade al archivo de configuración de Prettier:

.prettierrc

```
{
  "plugins": ["prettier-plugin-tailwindcss"]
}
```

eslint-plugin-tailwindcss

El plugin de ESLint se puede configurar. Por ejemplo:

.eslintrc.json

```
{
  "extends": ["next/core-web-vitals", "plugin:tailwindcss/recommended"],
  "root": true,
  "plugins": ["tailwindcss"],
  "rules": {
    "tailwindcss/no-custom-classname": "off",
    "tailwindcss/classnames-order": "error",
  },
  "settings": {
    "tailwindcss": {
      "callees": ["cn"],
      "config": "tailwind.config.js",
    }
  }
}
```

Ejemplo anterior sacado de https://www.jamesshopland.com/blog/tailwind-css-best-practices