***Notas del Curso.***

**¿Qué es Tailwind CSS?**

**¿**[**Qué es TailWind CSS**](https://tailwindcss.com/)**?** 🧐  
Es un [framework](https://en.wikipedia.org/wiki/Software_framework) que permite la construcción de diseños altamente personalizados y de bajo nivel.

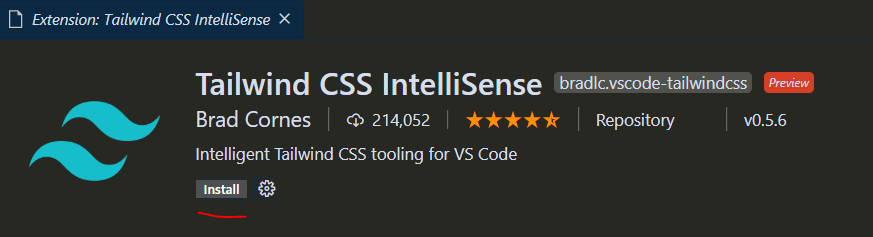
**¿**[**Por qué usar TailWind CSS**](https://tailwindcss.com/#what-is-tailwind)**?** 🤨  
A diferencia de otros frameworks css, como lo es [bootstrap](https://getbootstrap.com/), que otorgan componentes prediseñados, Tailwind provee clases a bajo nivel de css que nos permiten construir diseños completamente personalizados por nosotros y sin tener que pelear con el framework en sí.

Trabaja de manera amigable en el crecimiento de nuestro proyecto al proveernos herramientas para [extraer clases](https://tailwindcss.com/docs/extracting-components/).

Pero sobre todo, nos abre su código para [personalizarlo](https://tailwindcss.com/docs/configuration/) ya que está escrito en [PostCSS](https://postcss.org/) y configurado en JavaScript.

Tailwind is more than a CSS framework, it’s an engine for creating design systems.

**Extensión Recomendada.**



**Documentación:**

<https://tailwindcss.com/docs/installation>

# Instalación y ambiente de desarrollo.

# 

# Directivas de Tailwind.

Directiva es una instrucción que utiliza tailwind para insertar código en el archivo final de css que genera.

@tailwind base  
Esto inyecta los estilos base de Tailwind y cualquier estilo base registrado por plugins.

@tailwind components  
Esto inyecta las clases de componentes de Tailwind y cualquier clase de componente registrado por los plugins.

@tailwind utilities  
Esto inyecta las clases de utilidad de Tailwind y cualquier clase de utilidad registrada por los plugins.

@tailwind screens  
Esta directiva sirve para controlar donde Tailwind inyecta las variaciones responsivas de cada utilidad. Si se omite, Tailwind añadirá estas clases al final de tu css, por defecto.

# 

# Línea para crear archivo de configuración.

# 

# Personalización y configuración.

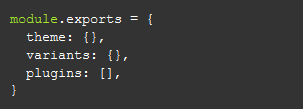
# Ya que Tailwind es un framework para construir UI a la medida, por default, se tiene un archivo opcional llamado *tailwind.config.js* en la raíz de la carpeta, donde está el package.json.

## **Creando un archivo de configuración**

Para generar un archivo de configuración para Tailwind, podemos usar el Tailwind CLI:



**Nota**. Podemos utilizar esta herramienta cuando instalamos la dependencia vía npm.  
Donde como resultado tendremos tailwind.config.js:



Cabe mencionar que cada sección, del archivo de configuración, es opcional.

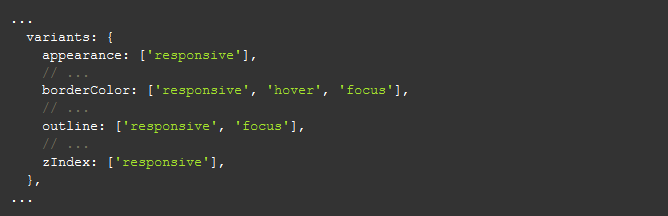
## **La sección Theme**

Esta sección es donde definimos los aspectos relacionados con el diseño visual de nuestro sitio.



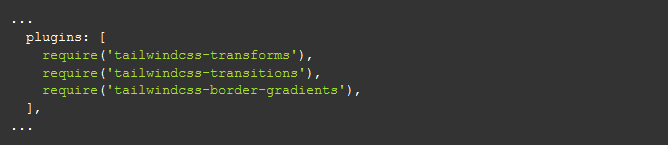
## **La sección Variants**

Esta sección nos permite controlar el comportamiento de las utilidades core, como responsive variants y pseudo-class variants.



## **La sección Plugins**

Esta sección nos permite registrar plugins de terceros con el objetivo de extender utilidades, componentes, estilos, etc.



<https://tailwindcss.com/docs/configuration>

# Responsive Design, Mobile First y Utility First

# 

# <https://tailwindcss.com/docs/utility-first>

# Colores.

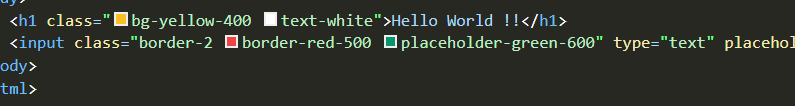
Los elementos que pueden ser afectados por los colores son:

* Fondo.
* Texto.
* Bordes.
* Placeholder.

Taildwind por default tiene valores ya predefinidos, para conocerlos tenemos que generar un archivo de configuración con todos los valores completos:

npx tailwindcss init tailwind.config.full.js --full

Se puede ver la paleta de colores por defecto aquí:  
<https://tailwindcss.com/docs/customizing-colors/#default-color-palette>



# Dimensiones y Espacios.

Aquí pueden ver los anchos  
<https://tailwindcss.com/docs/width>

Aquí pueden ver los padding  
<https://tailwindcss.com/docs/padding>

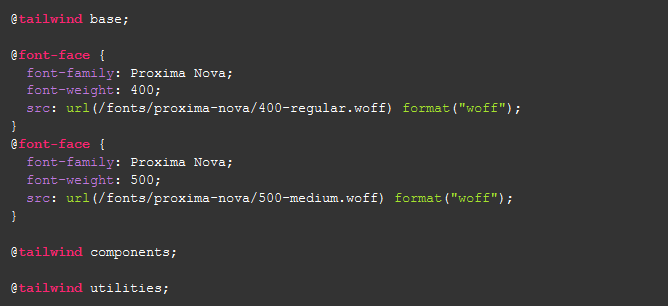
Aquí pueden ver los margin  
<https://tailwindcss.com/docs/margin>

# Cambiando las propiedades de la tipografía.

# Se puede consultar la documentación en: <https://tailwindcss.com/docs/font-weight/#app> <https://tailwindcss.com/docs/font-size/#app>

## **@font-faces**

Para aprovechar la regla @font-face para realizar cualquier personalización fuera de Tailwindcss modificamos el archivo CSS principal de la siguiente manera:



**Nota**. Para definir cualquier personalización hacia los estilos base, agregamos nuestros estilos después de @tailwind base y antes de @tailwind components para evitar problemas de compilación.

# Ajustando el espaciado entre letras y líneas

**Tracking** espaciamiento entre letras.

**Leading** espaciamiento entre líneas.

**text-left** Alineado a la Izquierda

**text-right** Alineado a la Derecha

**text-center** Alineación al centro

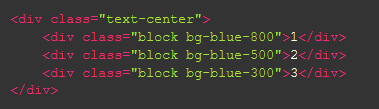
**text-justify** Texto justificado

Transformando el texto:  
**uppercase** text-transform: uppercase;  
**lowercase** text-transform: lowercase;  
**capitalize** text-transform: capitalize;  
**normal-case** text-transform: none;

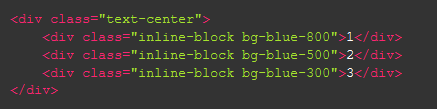
# Display.

Las propiedades de display son útiles para poder controlar las dimensiones o espacios. Las propiedades básicas de display son:

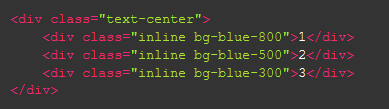
* **Block** → Con esta configuración los bloques abarcan toda la pantalla.



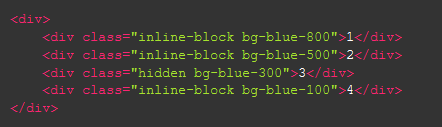
* **Inline-block** → Sólo ocupan el espacio necesario para mostrar lo que hay en su interior.



* **Inline** → Sólo ocupan el espacio necesario para mostrar lo que hay en su interior y la altura del elemento es indiferente.



* **Hidden** → El elemento en cuestión no se muestra.

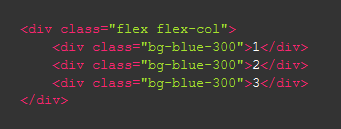


# Flexbox.

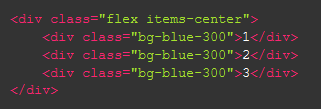
# <https://tailwindcss.com/docs/display#flex>

Flex es otra propiedad de display pero nos habilita otras propiedades para manipular nuestras cajas. Las propiedades propias de flexbox son:

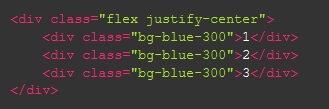
1. **Flex direction**.



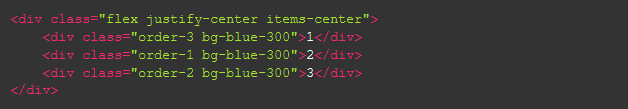
1. **Align ítems**.



1. **Justify content**



1. **Order**



# Cómo crear grids o columnas en Tailwind CSS

En esta clase vamos a aprender a crear un grid de columnas utilizando flex en Tailwind.

## **Grid Básico**

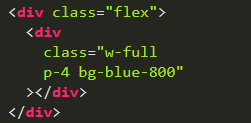
Verás que con lo que hemos aprendido hasta ahora es muy sencillo crear un grid.

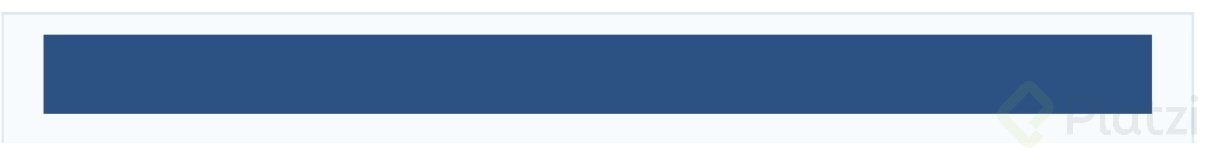
1. Crearemos un elemento que contendrá las columnas de nuestro Grid con la clase de flex.



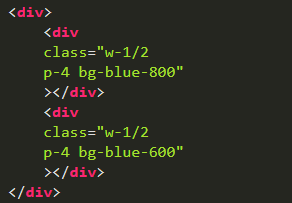
1. Teniendo de base el elemento del punto anterior insertaremos una etiqueta por cada columna que queramos tener de la siguiente manera.

**Una columna**

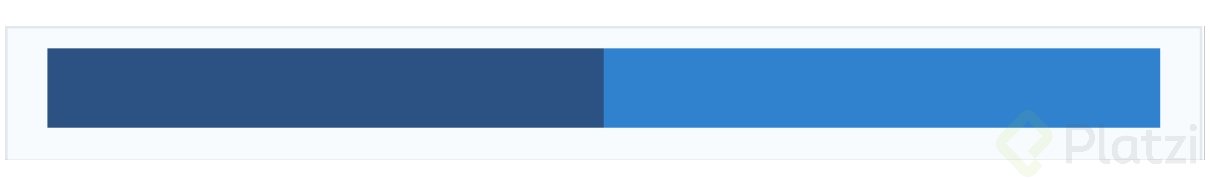


Resultado:  


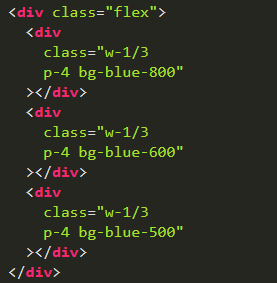
**Dos columnas**



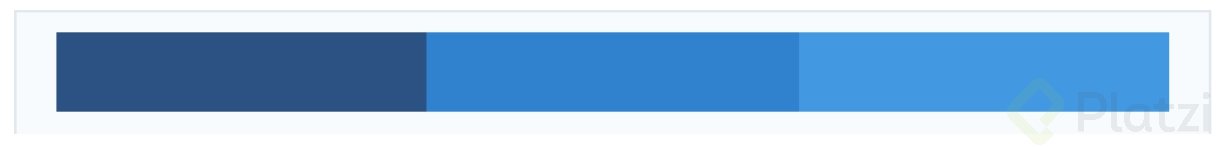
Resultado



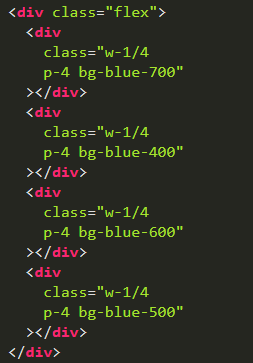
**Tres Columnas**



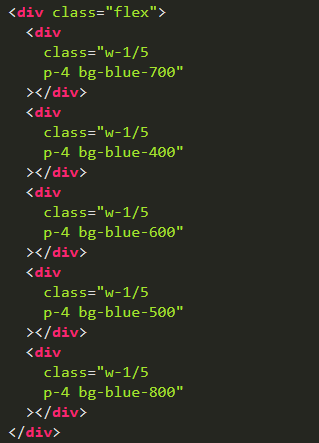
Resultado



**Cuatro Columnas**

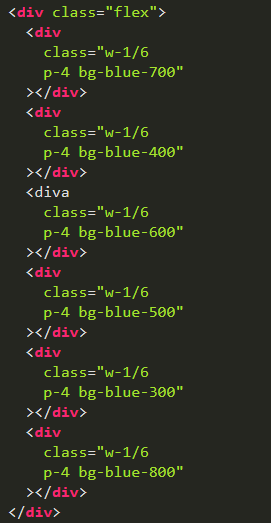


**Cinco columnas**



Resultado  


**Seis columnas**

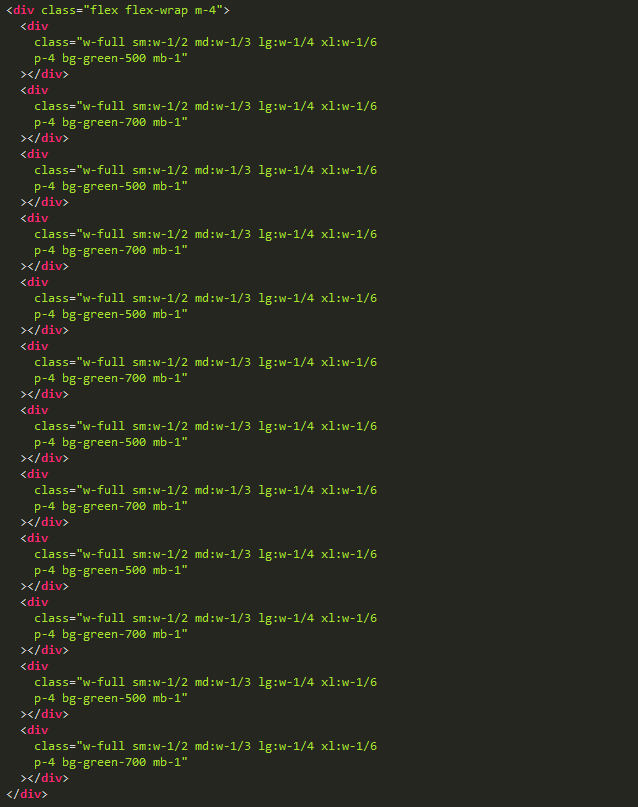


Resultado



## **Grid Responsivo con Columnas de Anchura Fija**

La diferencia para lograr que nuestro grid sea responsivo es que al elemento base donde tenemos la clase de flex le agregamos flex-wrap con esto logramos que cuando uno de los elemento exceda el tamaño, ese elemento pase al siguiente renglón. Y con el uso de las pseudo-clases responsivas de sm, md, lg y xl logramos el número de columnas deseada por tamaño de pantalla.

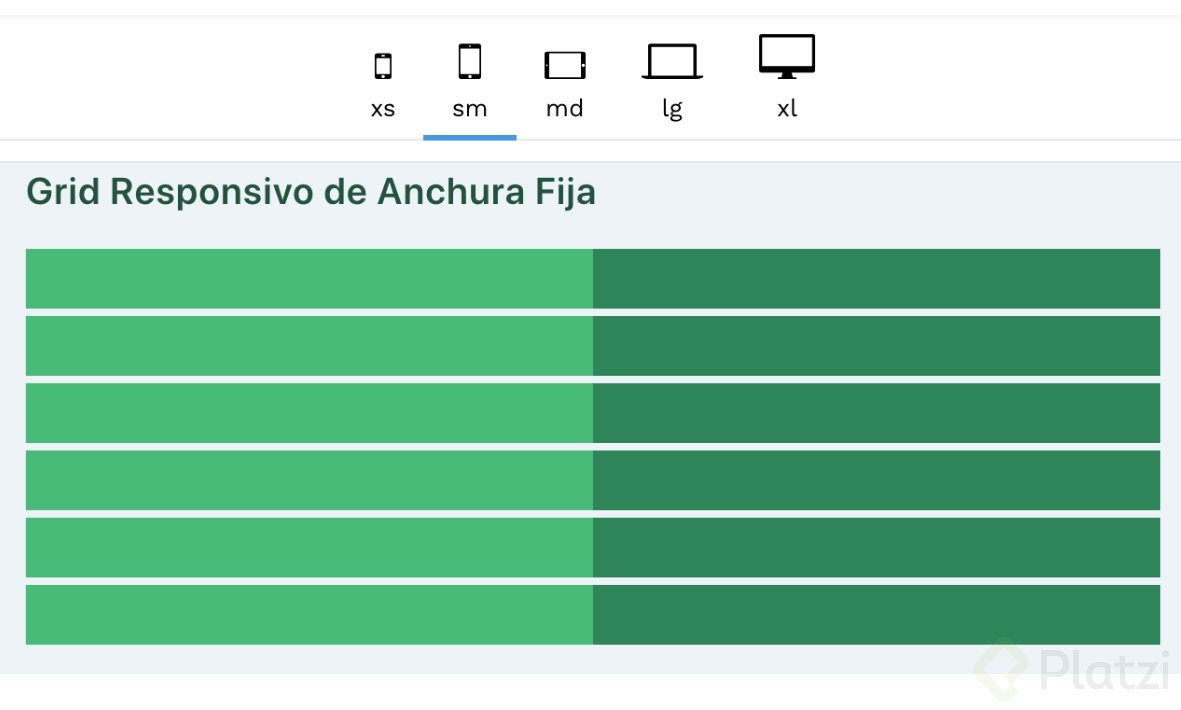


### *Resultados:*

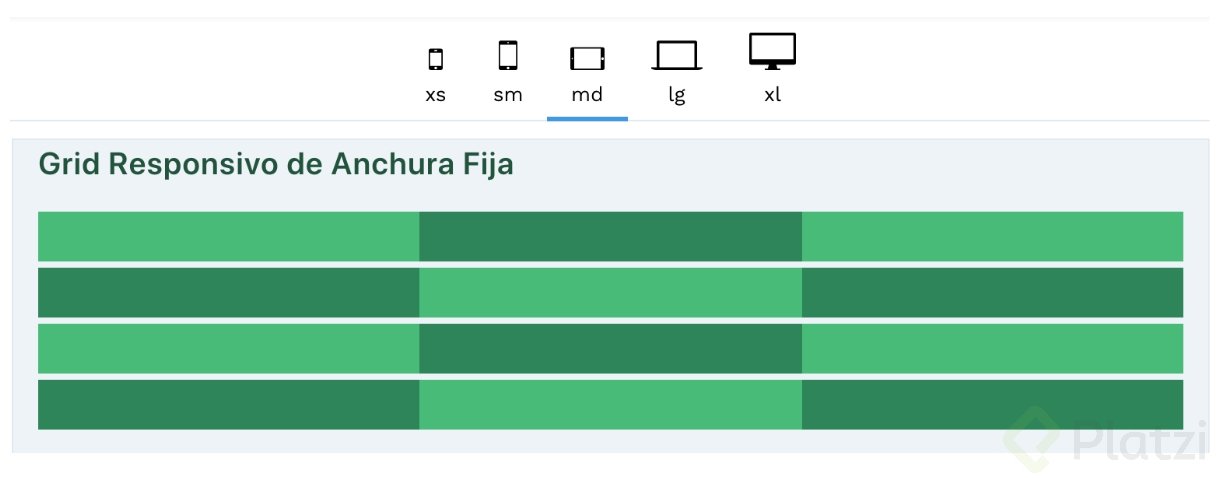
**default o xs**

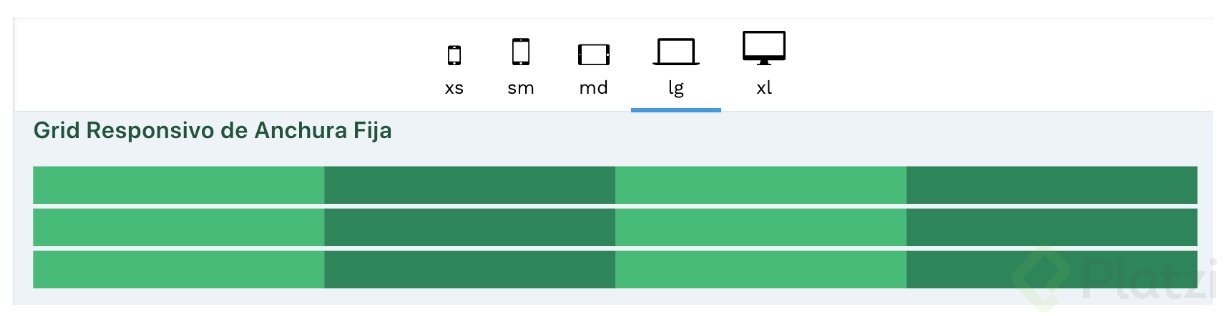


**sm**



**md**

  
**lg**

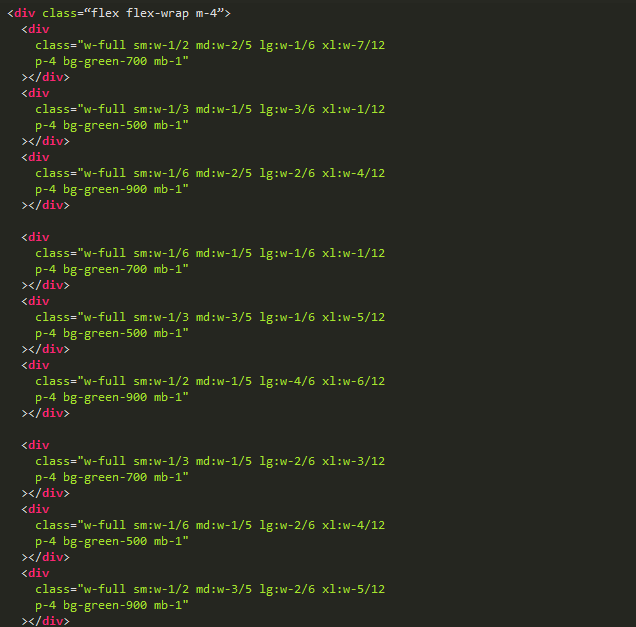


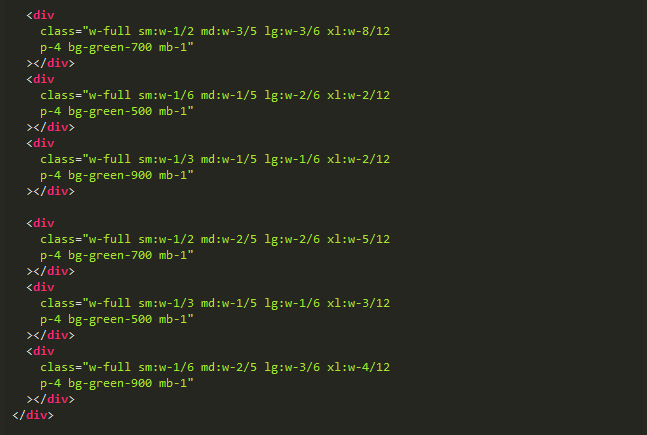
**xl**



## **Grid Responsivo con Columnas de Anchura Variable**

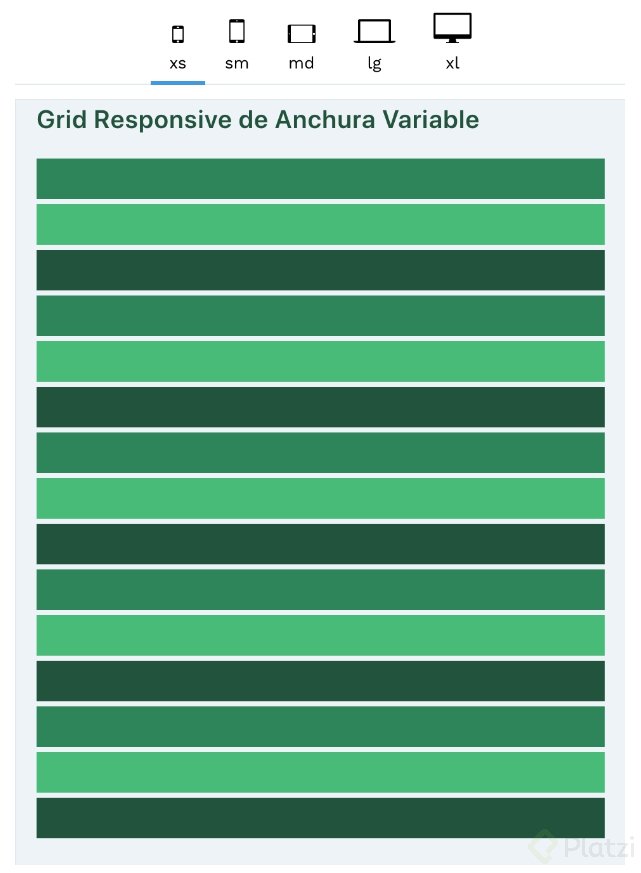
Al igual que con el grid responsivo anterior, agregamos en el elemento base de flex la clase de flex-wrap y de igual manera hacemos uso de las pseudo-classes responsivas de sm, md, lgy xl.



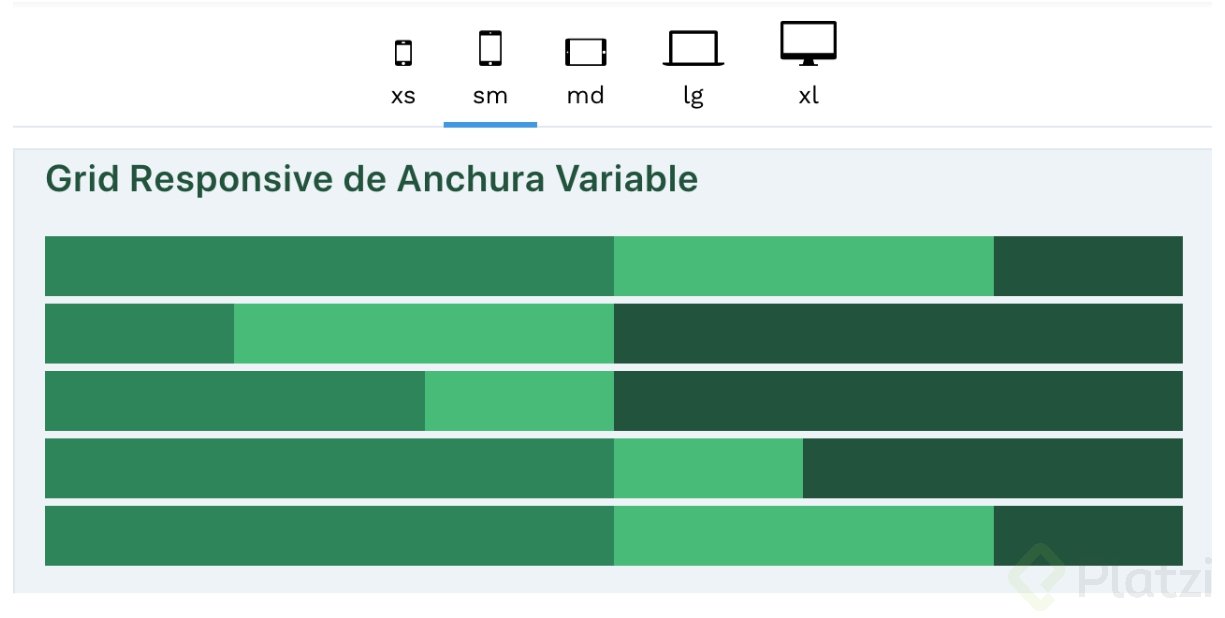


### *Resultados*

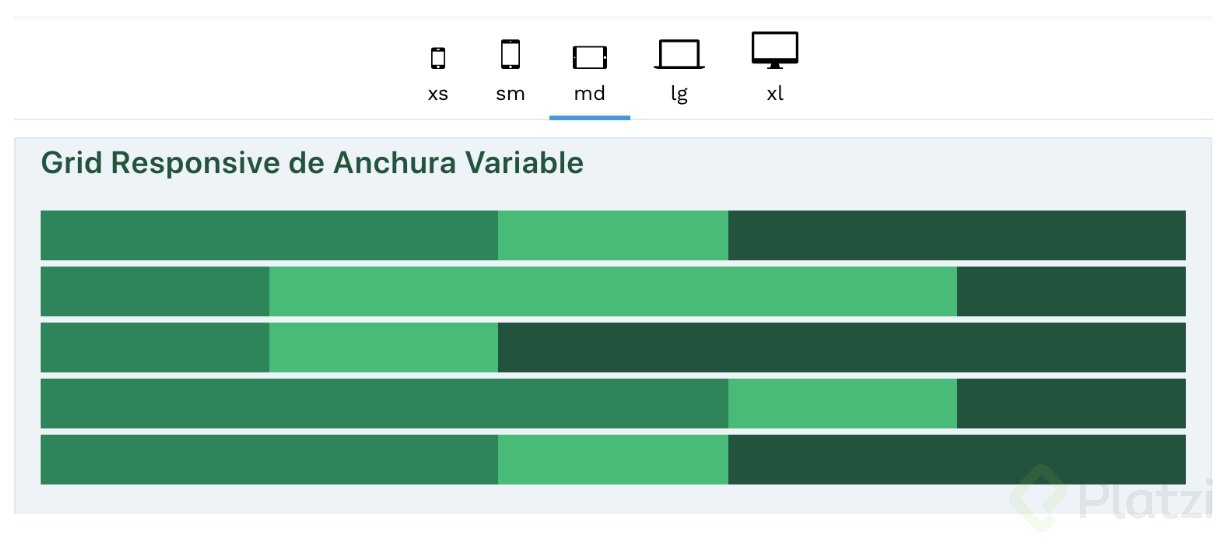
**default o xs**



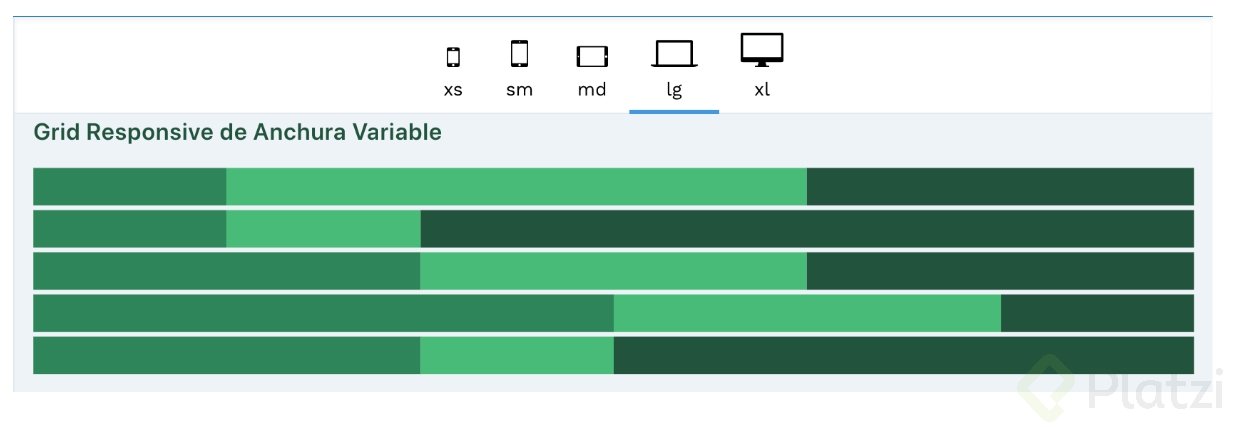
**sm**

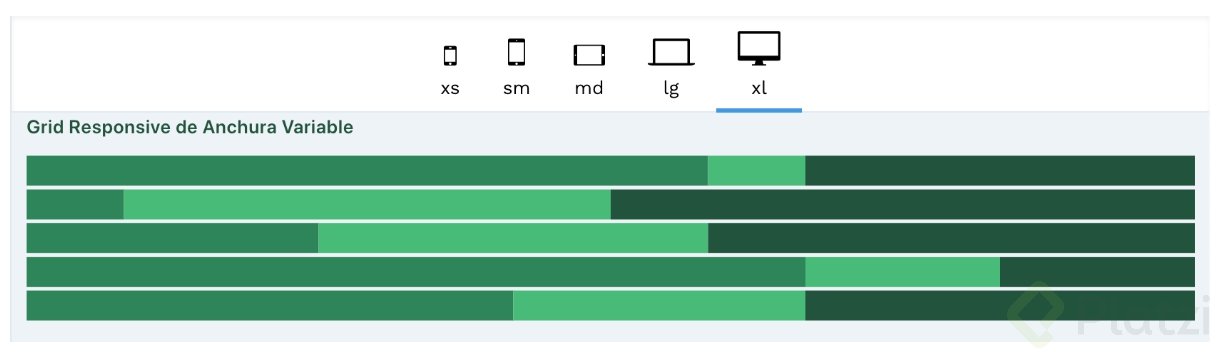


**md**



**lg**



**xl**  


## **Conclusión.**

Crear grids en Tailwind usando flex es muy sencillo.  
En flex podemos hacer uso de las clases de dimensiones porcentuales, algunos ejemplos son:  
w-1/2, w-2/3, w-3/4, w-5/6, w-7/12,.  
Estas mediciones nos facilita poder crear columnas de 2, 3, 4, 5, 6 y 12 columnas y sus combinaciones.

# Crea tus propias utilerías.

# Para agregar variantes a nuestro código CSS como un *hover,* debemosrealizar lo siguiente:

# 

# Para llamarla:

# 

# Entendiendo las variantes y las pseudo-clases.

# 

# Aquí está el código que agrega de la documentación:

# 

# El código que se agrega a el *tailwind.config.js:*

# 

# Se tiene que agregar el atributo disabled a la etiqueta para que funcione.

# 

# Directivas de Apply.

# Por defecto algunas etiquetas HTML no tienen estilos predeterminados como por ejemplo un h1. Para asignar estilos a estas etiquetas usamos @apply.

# 

# Un uso bastante frecuente es el de almacenar clases de Tailwind en una única clase para evitar las declaraciones de *muchas clases en línea* dentro de nuestro HTML*.*

# 

# 

# ¡Es hora de practicar!

### **Colores y Dimensiones**

Tip: Una buena práctica al combinar colores, como en botones por ejemplo, es no poner la letra color blanco o negro. En su lugar utiliza las tonalidades del mismo color para hacerla resaltar.

● Crea un botón color azul siguiendo estos consejos, recuerda que las tonalidades predefinidas van del 100 al 900.  
● Crea un botón que se adapte según el tamaño de dispositivo.  
● Cuando sea de tamaño pequeño, este botón debe abarcar el ancho de la pantalla y cuando sea un poco más grande debe tener un tamaño predeterminado.

Si no estás seguro sobre cuáles colores escoger, te recomiendo que uses Paletton, ahí podrás escoger un color y el programa te sugerirá colores que puedan combinar. (<https://paletton.com/>)

### **Display y Flexbox**

Tip: Una buena práctica es utilizar el margen de cada elemento para posicionarlo con respecto a otros. Esto se logra con margin left (ml-x), margin-top (mt-x). Si usamos margin right y margin bottom lo que hacemos es mover otros elementos respecto del elemento con el que estamos trabajando.

● Crea lo que se conoce como group button de 3 buttons o más. Esto lo logras utilizando lo aprendido en display o flexbox, así como margin y padding.

### **Pseudo—Class Variants**

Utilizando variantes como hover y active, logra crear efectos cambiando el color de los botones creados anteriormente.

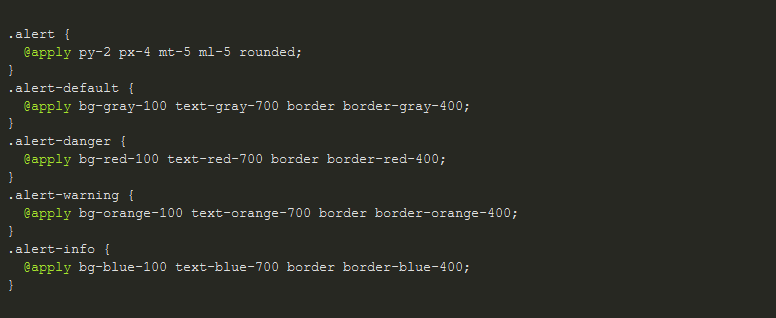
● Cuando pases el cursor encima de uno de los botones deberá cambiar el fondo a un color más oscuro, de tal manera que resalte con el texto. Y al estar en estado normal, el fondo deberá ser claro y el texto, color oscuro.

### **Extraer componentes usando @apply**

Utilizando la directiva @apply, crea un componente de “alert” con sus variaciones para que se comporte distinto según la clase que le acompañe.

Hay que crear:  
● alert (default) - Utiliza un color neutro  
● alert (danger) - Utiliza un color rojo  
● alert (warning) - Utiliza un color amarillo o naranja  
● alert (info) - Utiliza un color azul

Para desarrollar este reto tendrás que hacer uso de:  
● margin  
● padding  
● border  
● background  
● text color



# Alpine JS.

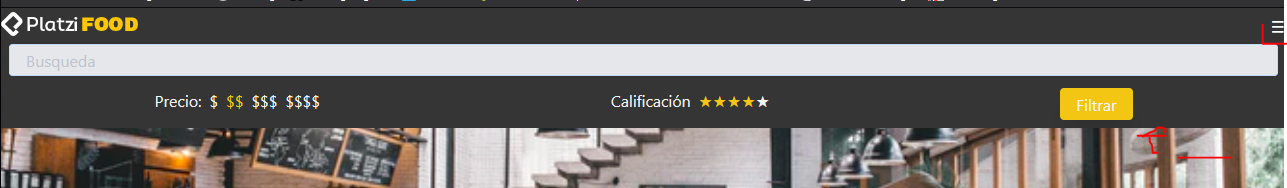
# Con alpine.js vamos a poder agregar la funcionalidad de click sobre algún botón (*Menu Hamburguesa*) y que despliegue algún otro elemento (*menú*)

# <https://github.com/alpinejs/alpine/blob/master/README.es.md>

# 

# 





Con las siguientes etiquetas pueden animar la transición del toggle del navbar.  
Estas etiquetas van en donde tienen la propiedad **x-show=‘open’** .

x-transition:enter="transition ease-linear duration-300"

x-transition:enter-start="opacity-0 transform -translate-y-4"

x-transition:enter-end="opacity-100 transform translate-y-0"

x-transition:leave="transition ease-linear duration-300"

x-transition:leave-start="opacity-100 transform translate-y-0"

x-transition:leave-end="opacity-0 transform -translate-y-4"

En la etiqueta **x-transition:enter** van todas las propiedades de la duración de la animación de entrada  
En la etiqueta **x-transition:enter-star** van todas las propiedades iniciales de estilos de la etiqueta, algo así como como el “frame 0”% de la entrada  
En la etiqueta **x-transition:enter-end** van todas las propiedades finales de estilos de la etiqueta, algo así como como el “frame 100%” de la entrada  
.  
.  
En la etiqueta **x-transition:leave** van todas las propiedades de la duración de la animación de salida  
En la etiqueta **x-transition:leave-start** van todas las propiedades iniciales de estilos de la etiqueta, algo asi como como el “frame 0”% de la salida  
En la etiqueta **x-transition:leave-end** van todas las propiedades finales de estilos de la etiqueta, algo así como como el “frame 100%” de la salida

# Optimiza tu archivo:

# PurgeCSS y NanoCSS.

# <https://www.npmjs.com/package/purgecss>

# <https://www.npmjs.com/package/nano-css>

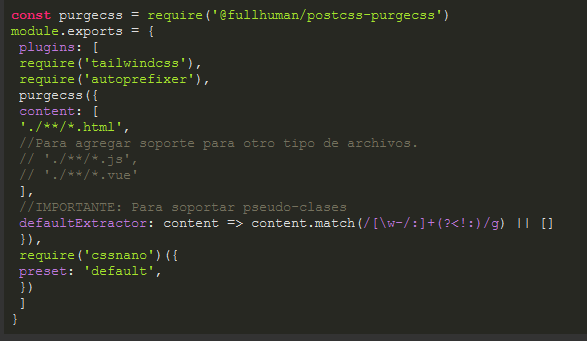
# Instalación PurgeCSS:

# 

# Instalación Nano:

# 

Este es el contenido del archivo de configuración postcss.config.js completo.  
Es Importante agregar la línea de:  
defaultExtractor: content => content.match(/[\w-/:]+(?<!:)/g) || []  
Para el correcto funcionamiento de Pseudo-classes.



**Codigo:**

const purgecss = require('@fullhuman/postcss-purgecss')

module.exports = {

plugins: [

require('tailwindcss'),

require('autoprefixer'),

purgecss({

content: [

'./\*\*/\*.html',

//Para agregar soporte para otro tipo de archivos.

// './\*\*/\*.js',

// './\*\*/\*.vue'

],

//IMPORTANTE: Para soportar pseudo-clases

defaultExtractor: content => content.match(/[\w-/:]+(?<!:)/g) || []

}),

require('cssnano')({

preset: 'default',

})

]

}

***👇 Cheat Sheet de Tailwind CSS***  
[***https://nerdcave.com/tailwind-cheat-sheet***](https://nerdcave.com/tailwind-cheat-sheet)

***👇 Curso del creador de Tailwind:***

[***https://www.youtube.com/watch?list=PL7CcGwsqRpSM3w9BT\_21tUU8JN2SnyckR&v=21HuwjmuS7A&index=1***](https://www.youtube.com/watch?list=PL7CcGwsqRpSM3w9BT_21tUU8JN2SnyckR&v=21HuwjmuS7A&index=1)

***Important Questions and Answers.***

