**Mostrar el listado de post**

**\*\*Antes de continuar se recomienda instalar tailwind css intellesense para VSC**

1. **Crear controlador para los Post**

**Una captura de pantalla de un celular con texto e imagen

Descripción generada automáticamente con confianza media**

1. **Luego codificamos dentro del controlador. Se deben llamar al modelo Post.**

**Texto

Descripción generada automáticamente**

1. **Retornamos dentro del método, la vista index**

**Texto

Descripción generada automáticamente**

1. **La ruta posts.index aún no existe. La creamos dentro de resources/views**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

1. **En esta vista indicamos que queremos que se extienda nuestra plantilla principal**

**Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja**

1. **Creamos la ruta respectiva**

**\*\*\* corrección: en el name de la ruta sería posts.index no post.index**

**Pantalla de computadora con letras

Descripción generada automáticamente con confianza media**

1. **Actualizamos la página y solo veremos la plantilla.**
2. **Ahora implementamos un div containter dentro de este llamado del componente. Para ello, creamos un div de color (el color que se desee) así:**

**Pantalla negra con letras blancas

Descripción generada automáticamente**

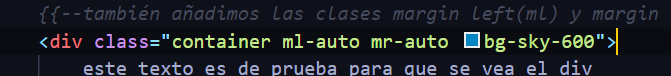
**Probamos en el navegador y veremos un div de color.**

**\*\*\*\*\*\*Si las clases de tailwind no funcionan, es necesario correr nuevamente npm run dev. En algunos casos, adicional a ello, es necesario borrar las cookies del navegador.**

**Texto

Descripción generada automáticamente**

1. **Si quisiéramos simplemente centrar sin pensar en el ancho específico del div, digitamos lo siguiente:**

****

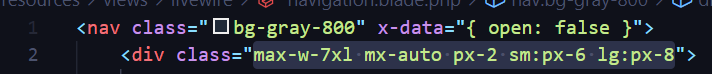
**Pero si recargamos el navegador web, veremos que no luce centrado adecuadamente en relación al menú de la parte superior. Para solucionar esto, hacemos lo siguiente:**

1. **Vamos a navigation**

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media**

**Copiamos los atributos de la segunda línea**

****

1. **Y los pegamos dentro de nuestro div container en el index**

**Una captura de pantalla de un celular

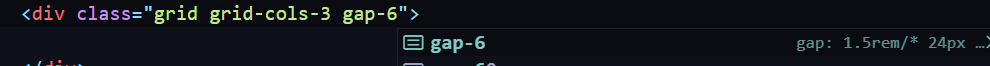
Descripción generada automáticamente con confianza media**

1. **Ahora dentro del div vamos a crear un grid (El grid es una cuadrícula sobre la que se distribuyen los distintos elementos que componen la web). Podemos escribir .grid para obtenerlo de manera automática con su respectiva clase.**

**Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente**

1. **Indicamos el número de columnas con col y además la separación (gap) que queremos que haya entre cada columna. Con gap-6 estaríamos indicando una separación de 1.5 rem**

****

1. **Creamos un foreach para poder iterar los elementos del array $posts del controlador y allí mostraremos las imágenes. Para poder mostrar dichas imágenes que están en la tabla images, es necesario recurrir a la relación anteriormente creada que está en el modelo Post**

**Texto

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**\*\*Sin embargo, aquí hay un error, porque la clase/modelo images no existe. Realmente, el modelo invocado se llama image. De hecho al ser una clase, debió llamarse Image.**

**La relación necesaria se llama image**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

**Texto

Descripción generada automáticamente**

**Al recargar la página web, observaremos una gran cantidad de información de los posts. Pero solamente necesitamos la url de las imágenes.**

**\*\*\*\*en caso de que no se visualicen los datos en 3 columnas, será necesario correr una vez más el comando npm run dev (solo aplica para Laravel 8 y versiones anteriores).**

1. **Para cargar solo las imágenes de los posts lo indicamos así:**

**Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente**

***Importante:*** Los siguientes pasos solo aplican si fue posible descargar directamente las imágenes en nuestro proyecto, con lo que tendríamos una copia de cada archivo de imagen en nuestros directorios. Pero si estamos viendo las imágenes de prueba online, es decir, vía url, estos pasos no aplican. Se debe saltar al paso 16.

**Pantalla de computadora con letras

Descripción generada automáticamente con confianza media**

* 1. **Ahora si colocamos una ruta específica de estas que estamos viendo en pantalla, al lado de la ruta principal del proyecto, no se visualizará ninguna imagen:**

**Texto

Descripción generada automáticamente**

**Pero si le agregáramos la ruta storage, sí visualizaríamos la imagen:**

**Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente**

* 1. **Lo que debemos hacer como en otros proyectos es usar el método Storage::url(), y dentro del paréntesis colocar lo que habíamos digitado:**

**Interfaz de usuario gráfica, Texto

Descripción generada automáticamente**

**En ese caso sí veremos la url completa de la imagen:**

**Captura de pantalla de computadora

Descripción generada automáticamente**

* 1. **Luego, para ver las imágenes, las incluimos en un img de la siguiente manera:**

**Texto

Descripción generada automáticamente**

* 1. **No obstante, pudiera ser que si no tenemos personalizado un dominio para el proyecto, las imágenes no se muestren. En ese caso, es necesario ir al archivo .env y configurar la APP\_URL exactamente como la estamos usando actualmente, con :8000, así:**

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja**

**Captura de pantalla de un celular

Descripción generada automáticamente**

1. **Si estamos trabajando con imágenes de prueba vía url.** En este caso no es necesario usar Storage::url. Más bien lo haríamos de la siguiente manera directamente:

****

1. **Quitamos el color de fondo.**

**Para escribir encima de la imagen, debemos hacer un cambio**

**Debido a que necesitamos escribir encima de la imagen, vamos a cambiar el método para mostrar las imágenes. En este caso sería usando un estilo css incrustado en el article, así:**

**Texto

Descripción generada automáticamente**

**La captura anterior está recortada. A continuación, el faltante de la derecha:**

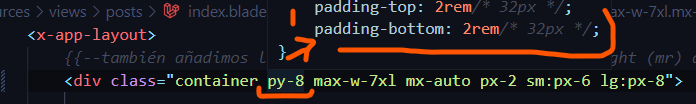
****

**\*\*\*en caso de que las imágenes no se vean, será necesario correr una vez más npm run dev (investigar más a fondo sobre este comando).**

**Puliendo la manera en que se muestran las imágenes del post**

Como puede notarse, las imágenes quedan pegadas a los otros elementos, tanto por arriba de la pantalla como por debajo.

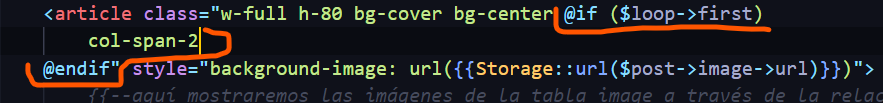
1. Vamos a añadir un padding. Para lograrlo solo escribimos py-8 en el container. La p significa padding y la y indica que el cambio se hace en vertical, lo que afecta el top y el botton. Si pasamos el cursor por encima, Visual Studio Code nos mostrará más detalles sobre py-8, como lo indica la flecha en la imagen:



Donde podemos apreciar que se agregarán 2 rem arriba y 2 rem abajo. Para mayor información, puede investigar sobre rem.

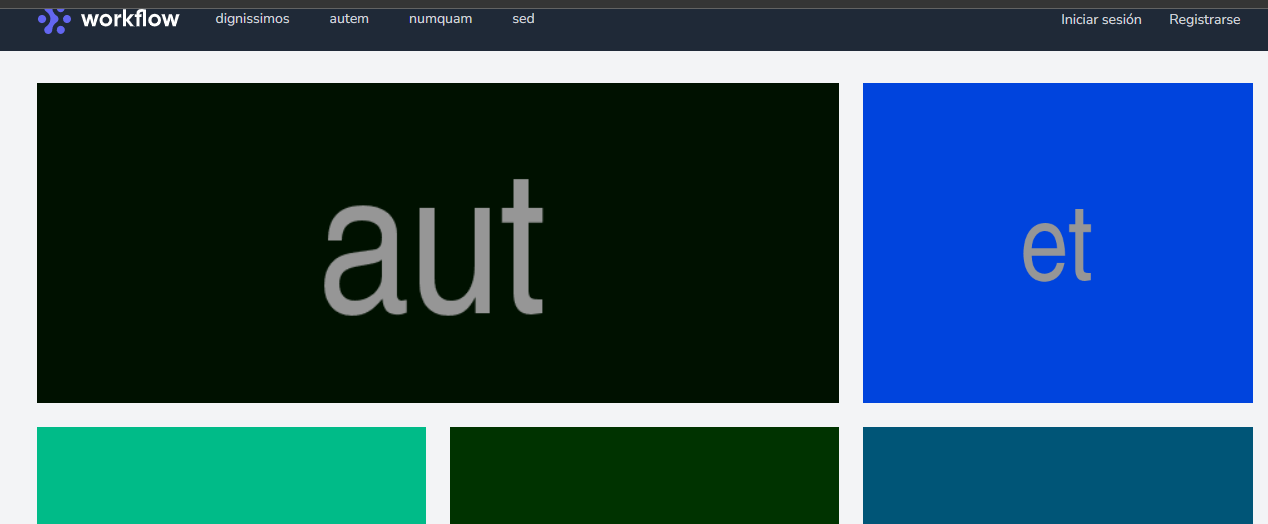
**Nota**: En Laravel 9, gracias a Vite, no es necesario estar ejecutando npm run dev cada vez. Eso sí, se deben tener en ejecución tanto php artisan serve como php run dev en dos terminales diferentes. Bastará con guardar cada cambio para que se refleje en el navegador automáticamente.

1. Ahora queremos que únicamente la primera imagen (que está dentro de un elemento article) sea más ancha que todas las demás. Lo lograremos introduciendo una condicional así:

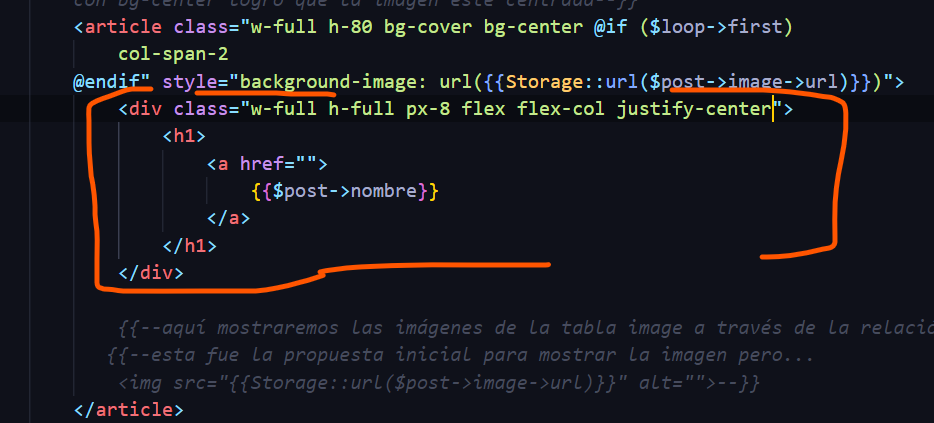


Fíjese bien que estas instrucciones se colocaron aún dentro de las comillas del class. Aunque se dieron espacios con enter, esto no afecta en nada negativo al proyecto. Lo que se hizo fue preguntar si era la primera iteración de este foreach, y si se cumple la condición, que ese article tenga el estilo de col-span-2.

1. Verificamos que luzca así:



1. Implementamos el siguiente código en el article:



Con w-full y h-full (clases de tailwind) indico que el div ocupe todo el ancho y alto posibles. Con px-8 se dio un padding en el eje equis de 8. También se usa la flex para indicar que se hará uso de una caja flexible (o flex-box) y flex-col para posicionar los elementos verticalmente. Para entender mejor esto de los flex box tenga en cuenta los siguientes puntos:

* *Todos los elementos en un html son cajas. <article> es una caja, <p> es una caja, <div> es una caja, etc.*
* *Lo normal es que cada una de estas cajas ocupe todo el ancho de la pantalla, cada una en su propia fila.*
* *Hasta una simple etiqueta <p> ocupa todo el ancho, su propia fila.*
* *Pero ¿Cómo lograr que los elementos que están dentro de una etiqueta, por ejemplo, los elementos dentro de un <div> estén uno al lado del otro y no cada uno en su propia fila? O, ¿Cómo organizarlos a nuestro gusto?*
* *De lo anterior nació el concepto y uso de las cajas flexibles o flex-box*

Para más información, visite <https://tailwindcss.com/docs/flex-direction>.

En nuestro caso, flex flex-col también ayudarán a mantener los elementos centrados ya que justify-center no tendrá efecto si estos.

Finalmente, con justify-center se centran los elementos debidamente.

1. En el h1, añadimos los siguientes estilos:

Texto

Descripción generada automáticamente

* 4xl (cuatro equis ele) hará que el texto tenga una tamaño mucho mayor
* leading se usa para el interlineado
* Y font-bold hará que la letra sea más gruesa

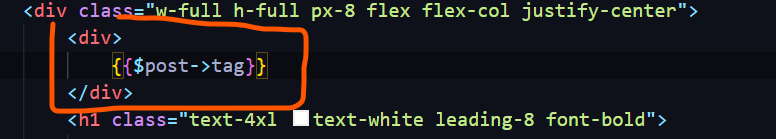
Puede haber casos donde sea necesario ejecutar de nuevo mpm run dev en la terminal, para visualizar cambios. O en caso de estar usando Laravel 8, se puede ejecutar npm run watch, comando que estará “a la escucha” de nuevos cambios.

El resultado será:

Texto

Descripción generada automáticamente

1. Encima de ese título o nombre del post, queremos mostrar las etiquetas (tags).
   1. Así que creamos un nuevo div y dentro llamamos a la relación tag() que previamente se especificó mediante una función en el modelo Post:

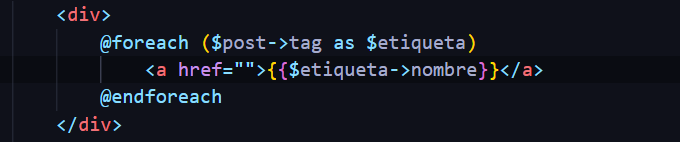


Esto llamará a todos los tags que están conectados a cada post en específico. Guardamos y vemos el resultado en el navegador:

Imagen que contiene Texto

Descripción generada automáticamente

* 1. Vamos a organizar mejor la información con un foreach, dentro del cual insertaremos el $post->tag del punto 6.1.



En el navegador veremos solo los nombres de los tags.

* 1. Añadimos estilos de tailwind:



* Con px-3 añadimos padding a la derecha y a la izquierda
* h-6 da una altura específica
* con rounde-full se da un estilo redondeado.

El resultado será:

Interfaz de usuario gráfica

Descripción generada automáticamente

* 1. Vamos a añadir unos colores al azar para los fondos de estas etiquetas. Para lograrlo, vamos a hacer lo siguiente:
     1. En el archivo de migración de tags, se añade el campo ‘color’:

Texto

Descripción generada automáticamente

* + 1. Luego en la ruta database/factories, indicamos que el campo color se rellene con unos valores aleatorios así:

Captura de pantalla de un celular

Descripción generada automáticamente

* + 1. Se ejecuta la migración, sin olvidar que se debe hacer un fresh y el sembrado de datos de prueba con –seed
    2. Se revisa la tabla tags en la BD para ver si aparecen los colores.



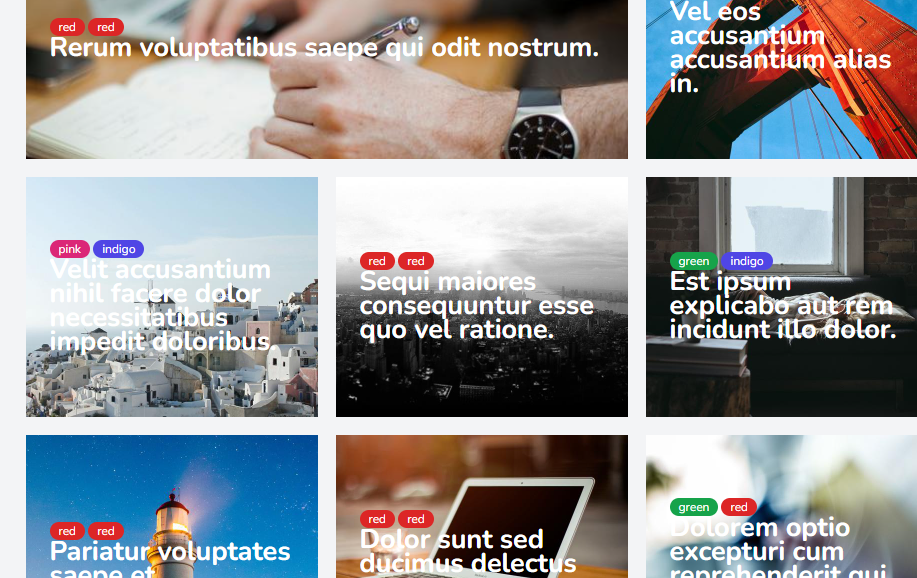
* + 1. En la ruta resources / views / posts, en el archivo index, borramos el color de fondo estático que está allí para los tags y llamamos de manera dinámica al nombre de color que ahora tenemos en la tabla de tags, de la siguiente manera:

Texto

Descripción generada automáticamente

* + 1. Se ejecuta npm run dev de nuevo (\*solo aplica para Laravel 8. En Laravel 9, con tan solo guardar, se ejecuta la compilación).
    2. Se supone que con la compilación, todos los estilos necesarios se reflejarán en el archivo app.css que se encuentra en public/css.

De hecho, al ejecutar php artisan serve e iniciar la aplicación, se debería ver así:



* + 1. No obstante, en algunos casos se ha observado que algunos colores no aparecen en la página. Por favor, revise el archivo app.css en la ruta public/css, aproximadamente en la línea 1379. Verá algo como:



Pero es probable que no vea bg-pink-600, bg-green-600, bg-blue-600, bg-yellow-600, bg-purple-600. En ese caso, debe añadirlos manualmente como se observa a continuación:

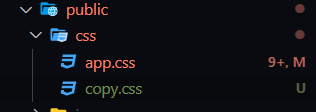
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

* + 1. Guardamos y recargamos la página web.
    2. Creamos un archivo como copia de seguridad del fragmento que acabamos de digitar ya que cuando en otra ocasión necesitemos hacer npm run dev es probable que estos cambios se eliminen. Por lo cual, creamos el archivo copy.css allí en el mismo directorio:



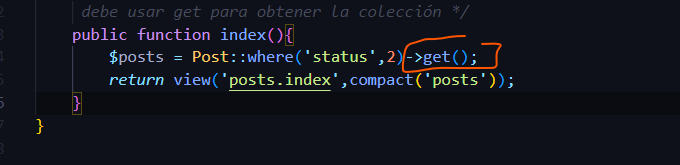
Allí guardamos copia del código que acabamos de digitar. Por lo pronto, esto no afectará en nada al proyecto ya que este archivo copy.css no se llama desde ninguna parte.

Hasta este punto hemos trabajado especialmente sobre las etiquetas (tags).

**Publicar o mostrar una cantidad limitada de Post en pantalla**

En vez de mostrar todos los posts, se podría hacer limitado de la siguiente manera:

1. Vamos a PostController y ubicamos donde está get:

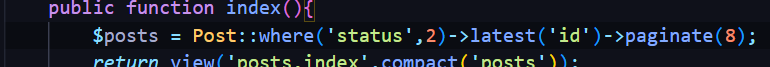


1. Cambiamos get por paginate e indicamos entre paréntesis 8:

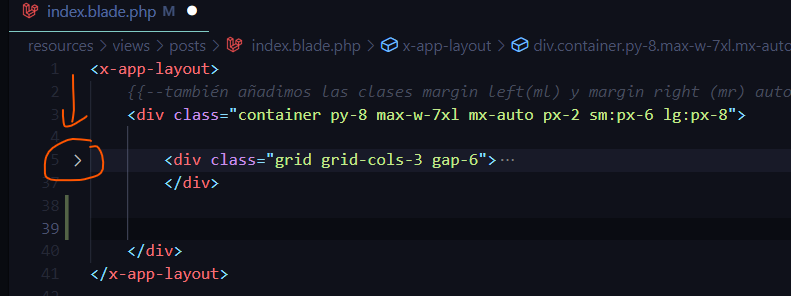
Texto

Descripción generada automáticamente

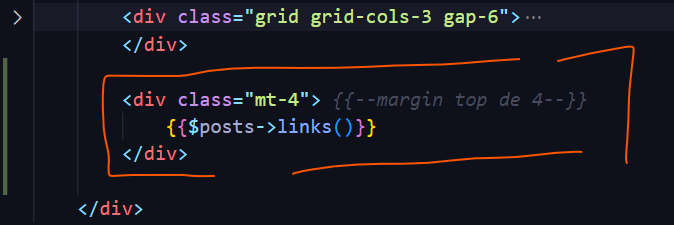
1. Igualmente, queremos que se muestre la información en forma descendente, así se mostrará de primero el último post publicado. Para ello usamos latest e indicamos como parámetro que haga este ordenamiento por ‘id’:



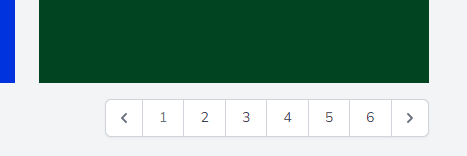
1. Verificamos su efecto desde el navegador web, recargando la página (en Laravel 9 no es necesario recargar la página).
2. Vamos al index donde veníamos trabajando (en resources/views/posts) y contraemos el código como se muestra en la captura:



1. Debajo del cierre del div indicado, hacemos lo siguiente:

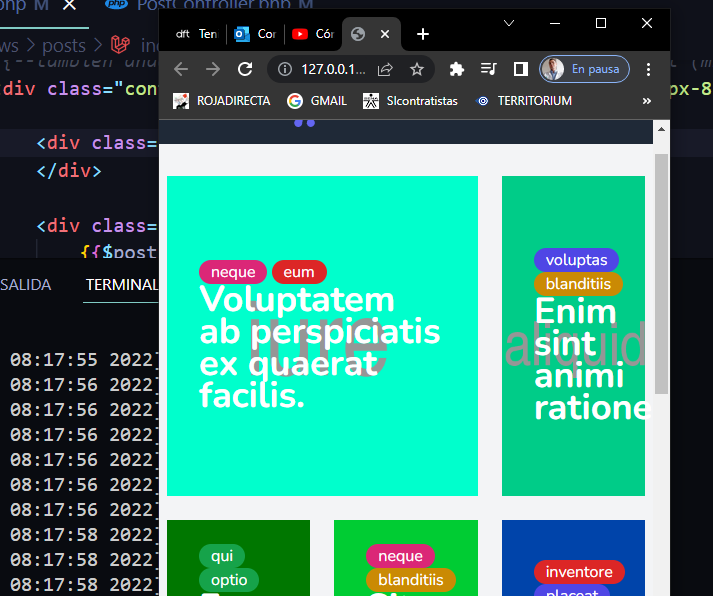


Con el método links, podremos establecer links de navegación entre las diferentes secciones según la paginación empleada en el controlador. La página web luciría así en su parte inferior:



**Hacer responsive la página**

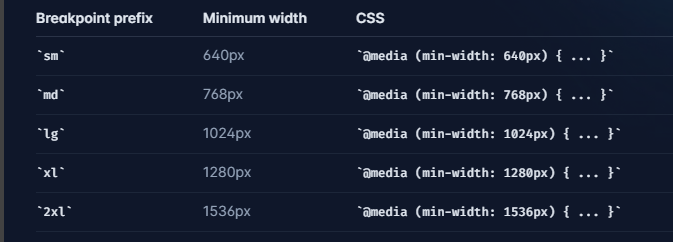
En este momento, si hacemos más pequeña la ventana del navegador, nos percatamos que la página web no es responsive.



¿Pero cómo hacemos la página web responsive con Tailwind? Para hacerlo, empezamos siempre desde lo más pequeño a lo más grande, es decir, primero determinamos cómo deseamos que se vea la página web en un dispositivo de pantalla pequeña, luego en un dispositivo de pantalla mediana, y finalmente en un dispositivo de pantalla grande. Claro, estos no son los únicos tamaños de pantalla posibles, pero sí los más comunes.

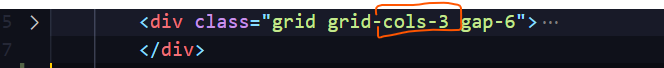
Para mayor información, visite <https://tailwindcss.com/docs/responsive-design>

Consultando la fuente anterior, vemos la siguiente tabla sobre tamaños de pantalla:

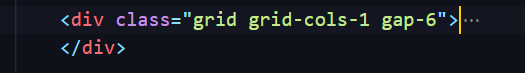


En nuestro proyecto, haremos los siguientes cambios:

1. En el index que venimos trabajando, ubicamos la parte referente a columnas, como se ve a continuación:



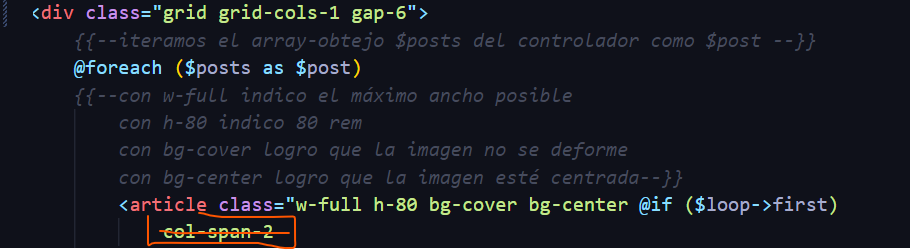
Y cambiamos cols-3 por cols-1.



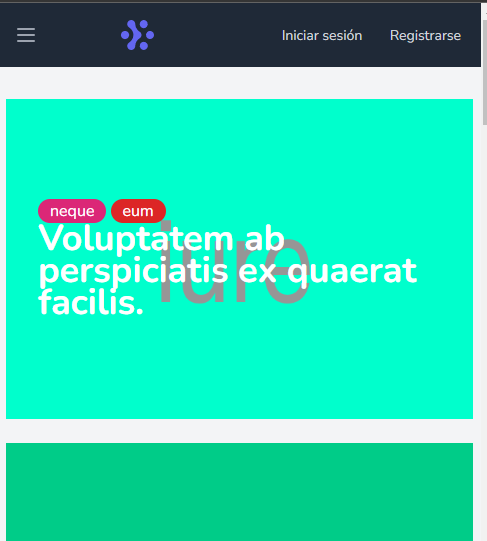
Con ello, estamos indicando que mientras la pantalla esté en el tamaño mínimo, cada grid constará de 1 sola columna, y no 3 como estaba planteado originalmente. Sin embargo, si agrandamos la ventana del navegador, no se verá 1 columna sino incluso 2.

Por otra parte, el primer elemento del listado se ve un poco más grande que los otros. Estos errores los vamos a corregir.

1. En el article que está dentro del foreach, quitamos el col-span-2:



Guardamos y actualizamos. Debería verse así:

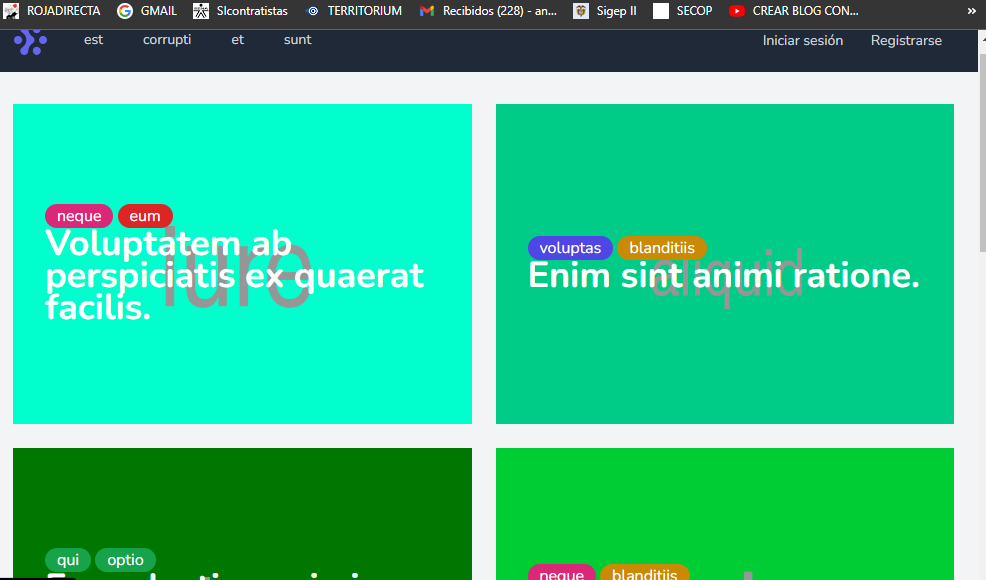


**Ajustando para una pantalla mediana**

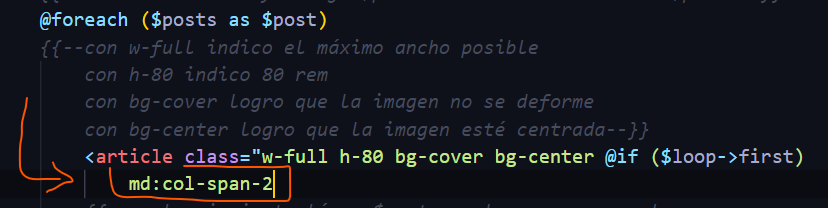
1. Agregue lo siguiente en el div del grid principal:



Con md: estamos indicando un estilo para pantalla mediana.

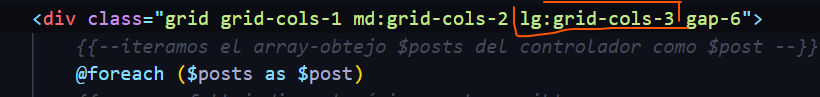


1. A partir de la pantalla mediana, queremos que el primer elemento ocupe 2 columnas. Lo indicamos así:



**Ajustando para una pantalla grande**

1. Indicamos lo siguiente en el div principal:



La propiedad lg: está indicada para pantallas grandes (aunque no extra grandes). Hemos indicado, además, que contendrá 3 columnas.

Es posible que tengamos que hacer nuevamente npm run dev (aplica para Laravel 8 y versiones anteriores. En Laravel 9 y superior se supone que con tan solo guardar debería ocurrir la compilación).

\*Nota: Si se usó npm run dev, es posible que se deba usar el código de respaldo que tenemos en copy.css y volverlo a colocar en app.css (carpeta public), ya que se ha reportado que en algunas versiones de Laravel esto se borra automáticamente al compilar, puesto que fueron líneas de estilo agregadas manualmente y no por compilación.