

Guía de ejercicios 2 - Grid



¡Hola! Te damos la bienvenida a esta nueva guía de ejercicios.

¿En qué consiste esta guía?

Anteriormente, definimos como tarea el terminar de construir los estilos base para todos los dispositivos móviles.

Ahora, conoceremos otro componente importante a la hora de crear un diseño responsive llamado grilla. Estas, no suponen ninguna novedad para nosotros, ya que la hemos usado muchas veces mientras construimos sitios web con Bootstrap, pero en realidad, las grillas son mucho más que solo un elemento valioso de este framework CSS.

¡Vamos con todo!



Tabla de contenidos

Grid-container y grid-item	3
Actividad 1	4
Gap	4
row-gap y col-gap	5
Actividad 2	5
Alineando el contenido	5
Actividad 3	6
Construyendo una galería de imágenes (versión 1)	7
Especificando el ancho de los ítems	8
Especificando el ancho en fracciones	12
Mezclando fracciones y otras unidades de medida	13
repeat	13
Creando un layout con CSS Grid	15
Cambiando el orden del layout	17
Grid-template-area	18
Actividad 4	19
Actividad 5	20
Creando una galería de imágenes con grid-template-areas	21



¡Comencemos!

Grid-container y grid-item

Grid es una herramienta que, al igual que Flexbox, nos permite alinear contenido. Un caso básico de uso sería un HTML con la estructura base, y dentro del body lo siguiente:

```
<div  
  class="grid-container">  
  <div class="grid-item">1</div>  
  <div class="grid-item">2</div>  
  <div class="grid-item">3</div>  
</div>
```

Y en el CSS (dentro de style en el head o archivo externo):

```
.grid-container{  
  display: grid;  
  grid-template-columns: auto auto auto;  
}
```

Lo que nos daría como resultado:

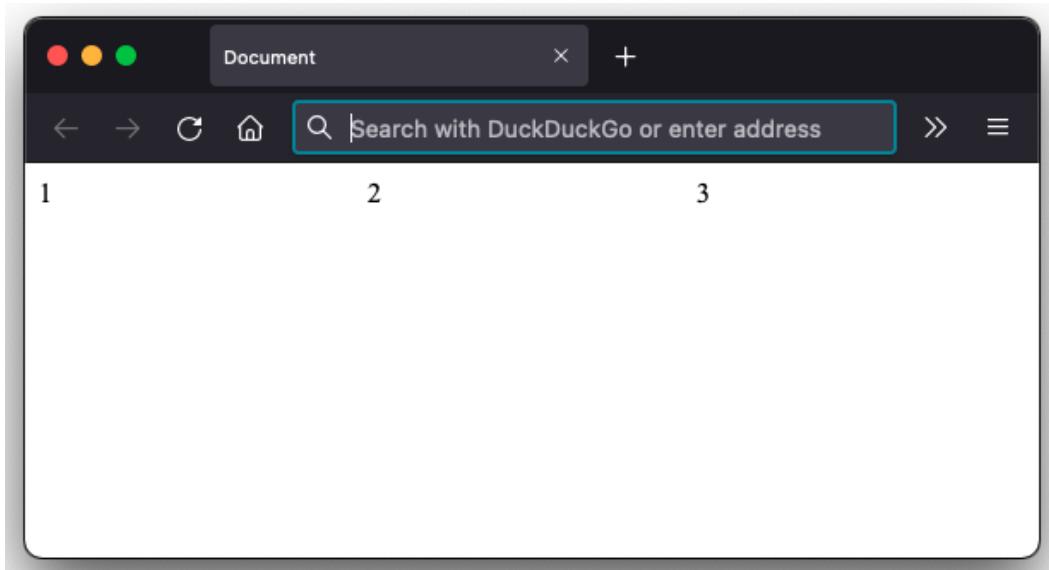


Imagen 1. Screenshot de ejemplo básico de grid.

Fuente: Desafío Latam.

O sea, basta aplicar la propiedad display:grid y definir una cantidad de columnas con grid-templates-columns para poder alinear los contenidos uno al lado del otro.



Actividad 1

- Repite desde cero el ejercicio anterior.
- Utilizando el inspector de elementos, cambia el valor de grid-template-columns a auto auto (solo 2), esto modificará la grilla a que funcione únicamente con 2 columnas.
- El tercer elemento debería quedar abajo.
- De esta forma se pueden construir grillas de 2 o tantas como necesitemos.

Gap

El gap (en español se traduciría como espacio o brecha) nos permite controlar la separación entre filas y columnas.

Para probar el gap, sumaremos más ítems a nuestro HTML, agregaremos un gap de 10 píxeles y un color de fondo para poder observar efectivamente la separación.

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>
```

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
}
.grid-item {
  background-color: aqua;
}
```

Al guardar y abrir la página en el navegador, deberíamos ver nuestras 3 columnas y cada ítem separado por 10px.

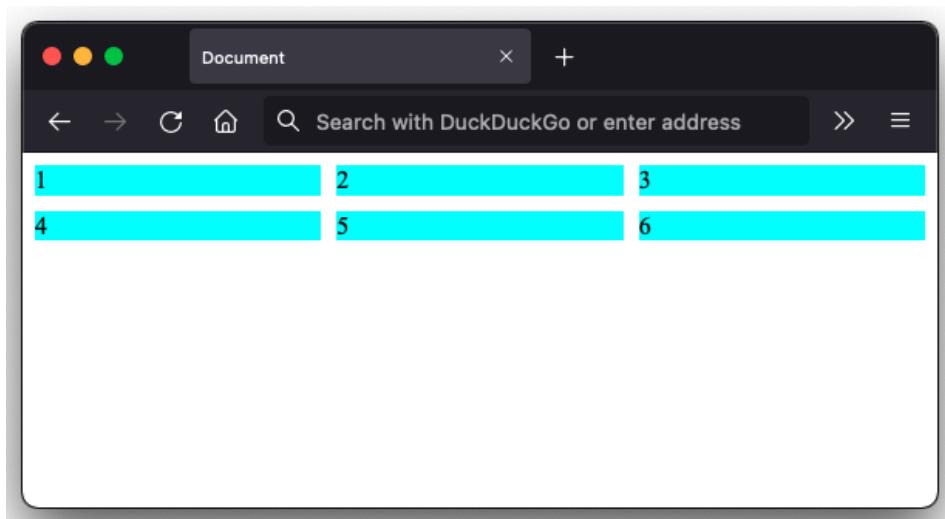


Imagen 2. Screenshot de ejemplo básico de grid con color de fondo.

Fuente: Desafío Latam.

row-gap y col-gap

La propiedad gap está compuesta de dos propiedades, row-gap y col-gap, de esta forma podemos dar espacios distintos a nuestras columnas y filas



Actividad 2

- Utilizando como base el ejercicio anterior, agrega row-gap y col-gap, para modificarlo y dar 50 píxeles de espacio a las filas (row) y solo 20 de espacio a las columnas.



Cuidado

También existen las propiedades grid-row-gap y grid-col-gap, pero están obsoletas. De todas formas, si intentas utilizarlas en el editor de visual studio code se te avisará de esto en las sugerencias.

Alineando el contenido

Usando justify-items y align-items podemos alinear los contenidos de los ítems dentro de un contenedor-flex.

```
<div class="grid-container">
  <div class="grid-item">1</div>
  <div class="grid-item">2</div>
  <div class="grid-item">3</div>
  <div class="grid-item">4</div>
  <div class="grid-item">5</div>
  <div class="grid-item">6</div>
</div>
```

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: center;
  align-items: center;
  height: 500px;
}
.grid-item {
  background-color: aqua;
}
```

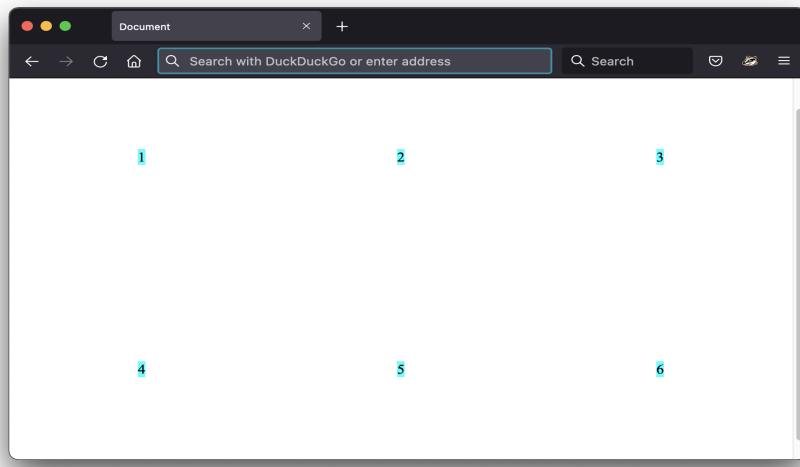


Imagen 3. Screenshot de alineación de contenido en una grid.

Fuente: Desafío Latam.



Actividad 3

- Utilizando como base el código anterior, ocupa el inspector de elementos para cambiar la propiedad `justify-items` y `align-items` y observa los resultados.
- Luego, intenta cambiar el alto y prueba también asignándole el valor 0.



Para que la alineación funcione, es necesario que el contenedor tenga un alto o que al menos un elemento dentro tenga un alto distinto al resto, de otra forma, se verán todos al mismo nivel.

Construyendo una galería de imágenes (versión 1)

Vamos a repasar lo aprendido hasta ahora, construyendo una galería de imágenes donde en su mayoría tienen tamaños distintos.

Como base utilizaremos el siguiente HTML:

```
<div class="grid-container">
  <div class="grid-item">
    
  </div>
  <div class="grid-item">
    
  </div>
</div>
```

Y el mismo CSS que hemos utilizado hasta ahora:

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  justify-items: center;
  align-items: center;
  height: 500px;
}
```

```
.grid-item {  
    background-color: aqua;  
}
```

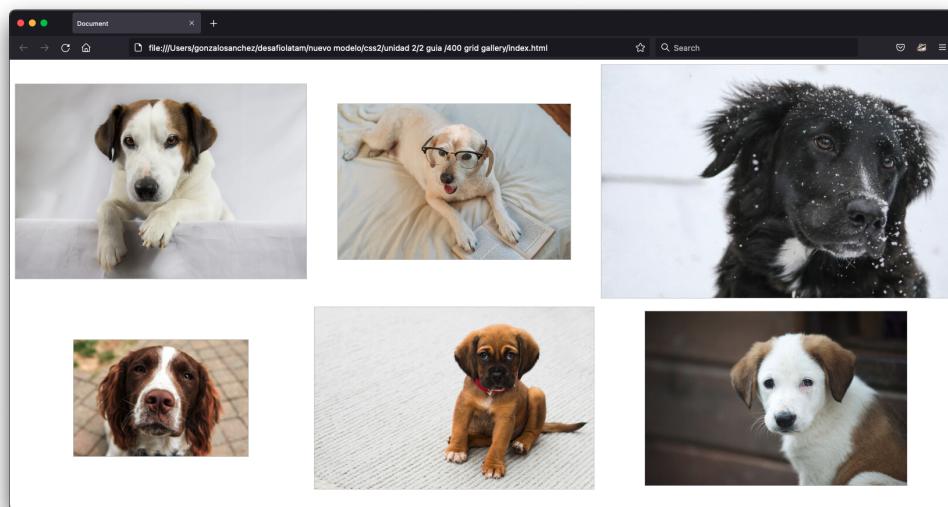


Imagen 4. Creando una galería de imágenes con grid y la propiedad gap.

Fuente: Desafío Latam.

Especificando el ancho de los ítems

La propiedad `grid-template-columns` permite especificar el ancho de cada elemento.

Para trabajar con estas propiedades, volveremos temporalmente a nuestro archivo HTML anterior sin imágenes. También eliminaremos las propiedades `justify-items`, `align-items` y `grid-template-columns`, donde especificaremos un ancho fijo para una columna.

```
<div class="grid-container">  
    <div class="grid-item">1</div>  
    <div class="grid-item">2</div>  
    <div class="grid-item">3</div>  
    <div class="grid-item">4</div>  
    <div class="grid-item">5</div>  
    <div class="grid-item">6</div>  
</div>
```

```
.grid-container {  
    display: grid;  
    grid-template-columns: auto 80% auto;  
    gap: 10px;
```

```
}
```

```
.grid-item {
```

```
    background-color: aqua;
```

```
}
```

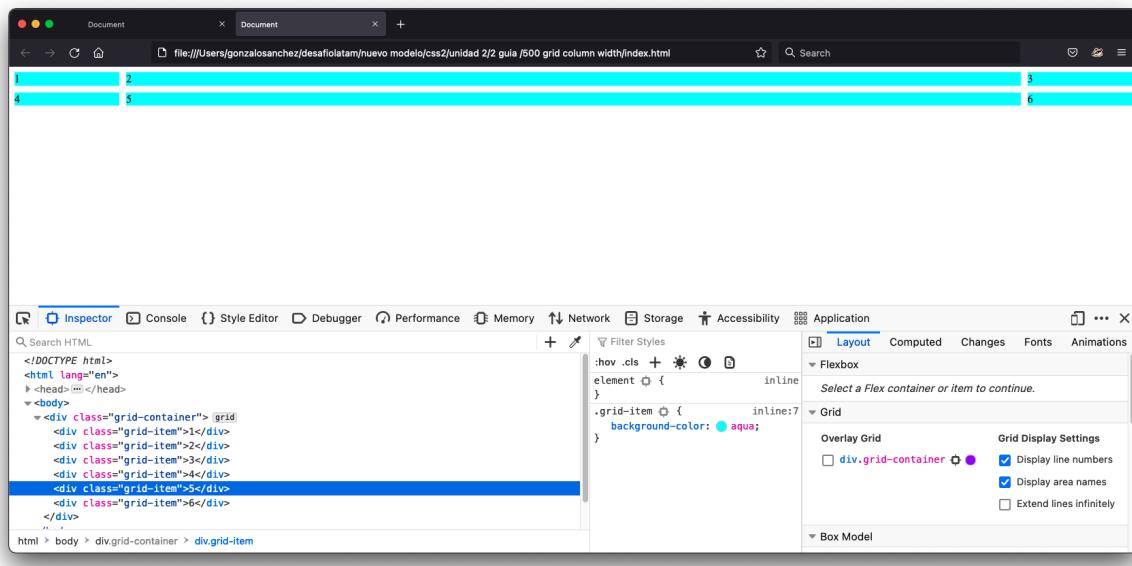


Imagen 5. Especificando el ancho con grid-template-columns.

Fuente: Desafío Latam.

Pero si ahora decidimos centrar los ítems agregando en .grid-container justify-items: center observaremos algo curioso.

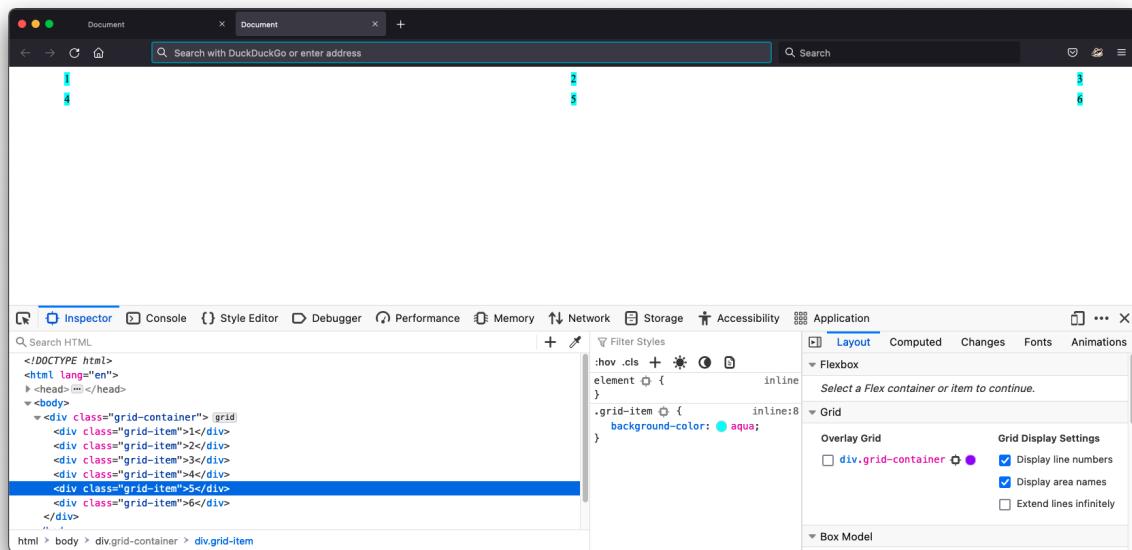


Imagen 6. Screenshot demostrando la pérdida de los anchos al utilizar justify-items.

Fuente: Desafío Latam.

Aquí los ítems están centrados, sin embargo, perdemos el tamaño de las columnas dado que cuando utilizamos el centrado, el tamaño ahora se calcula en función del contenido de las cajas. Esto lo podemos demostrar fácilmente cambiando los números por un texto lorem ipsum.

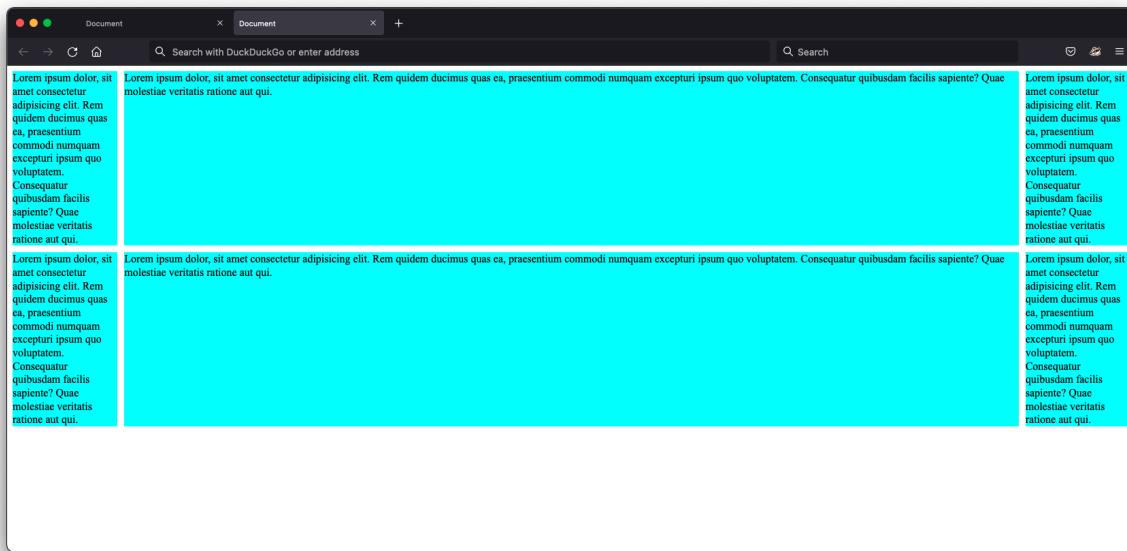


Imagen 7. Screenshot demostrando que el tamaño se calcula en función del contenido.

Fuente: Desafío Latam.



Cuando se alinea el contenido ya sea con justify-items o align-items el tamaño de la caja pasa a ser automáticamente del menor tamaño posible y su tamaño se calcula en función del contenido.

Los ítems que no ocupan todo el espacio se alinean con la propiedad especificada, en nuestro ejemplo sería center.

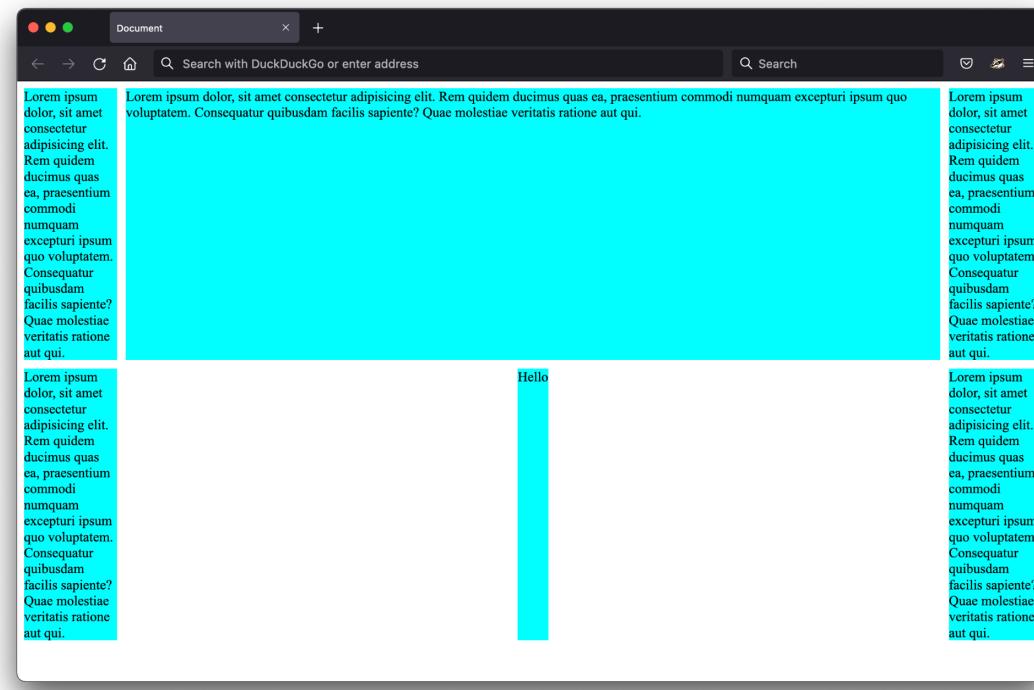


Imagen 8. Screenshot especificando grid-template-column: auto 80% auto con justify-items: center.

Fuente: Desafío Latam.

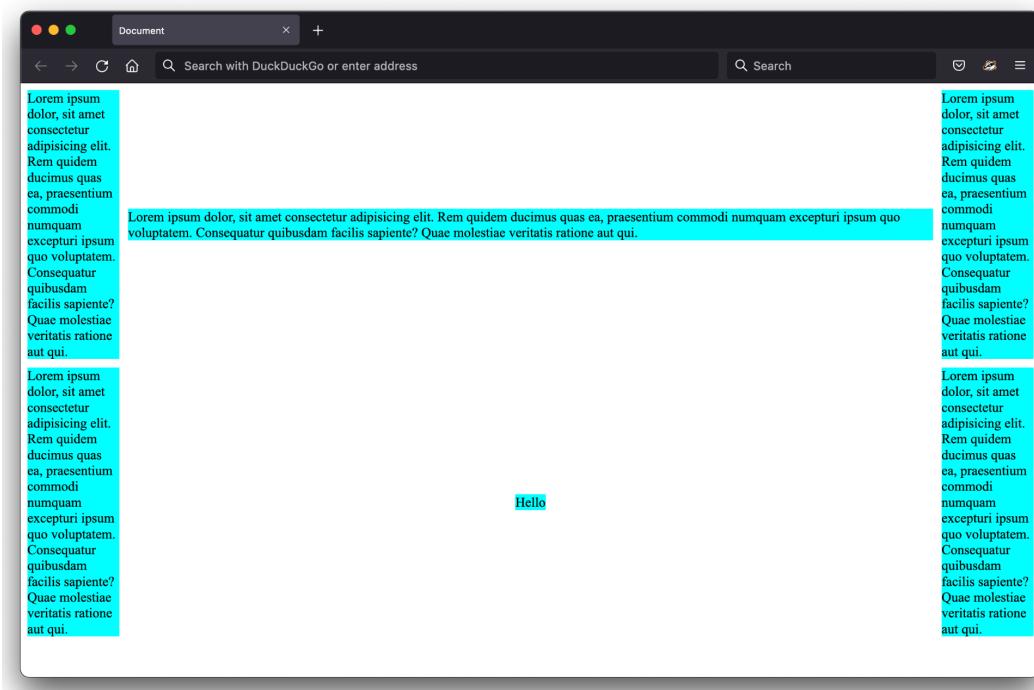


Imagen 9. Screenshot especificando grid-template-column: auto 80% auto con justify-items: center y align-items: center.

Fuente: Desafío Latam.

Especificando el ancho en fracciones

En grid se recomienda utilizar la unidad de medida de fracción. Por ejemplo, grid-template-columns: 1fr 1fr; dividiría la página automáticamente en dos columnas en partes iguales.

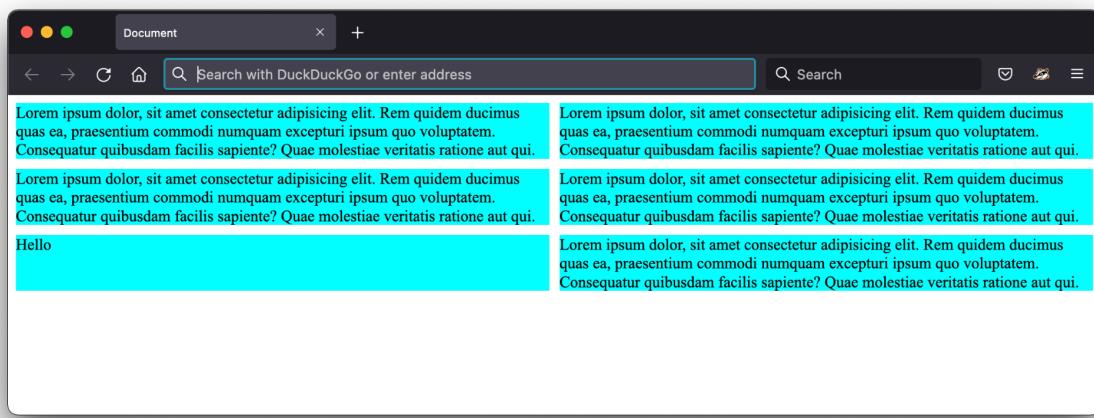


Imagen 10. Screenshot especificando grid-template-column con fracciones.

Fuente: Desafío Latam.

Mientras que si utilizamos grid-template-columns: 2fr 1fr; significa que de 3 partes, la columna izquierda ocuparía 2 partes del espacio disponible y la columna derecha solo 1.

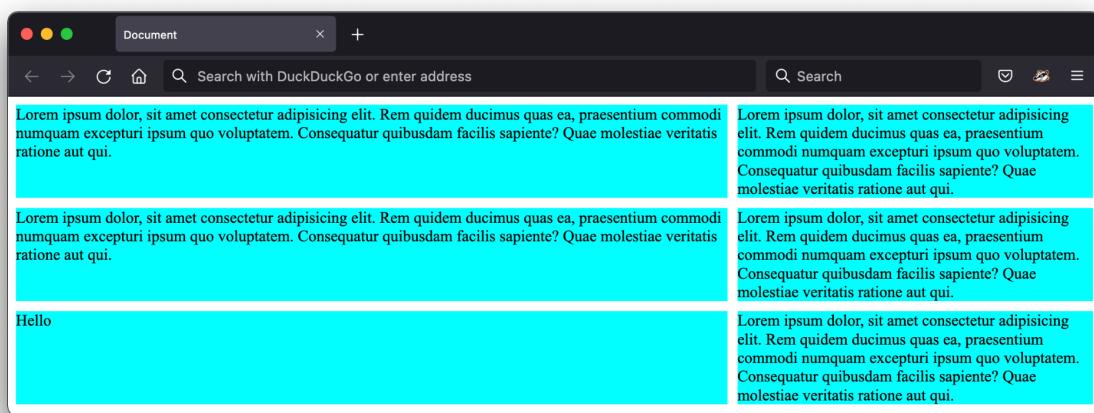


Imagen 11. Screenshot especificando grid-template-column con fracciones de 2fr y 1fr respectivamente.

Fuente: Desafío Latam.

Podríamos probar también crear 3 columnas, una de 1fr, otra de 2fr y una tercera de 1fr. Esto significa que de 5 fracciones ($1 + 2 + 1$) la primera ocuparía $\frac{1}{4}$ (25%) del espacio disponible, la segunda $\frac{2}{4}$ (50%) y la última el $\frac{1}{4}$ (25%) restante.

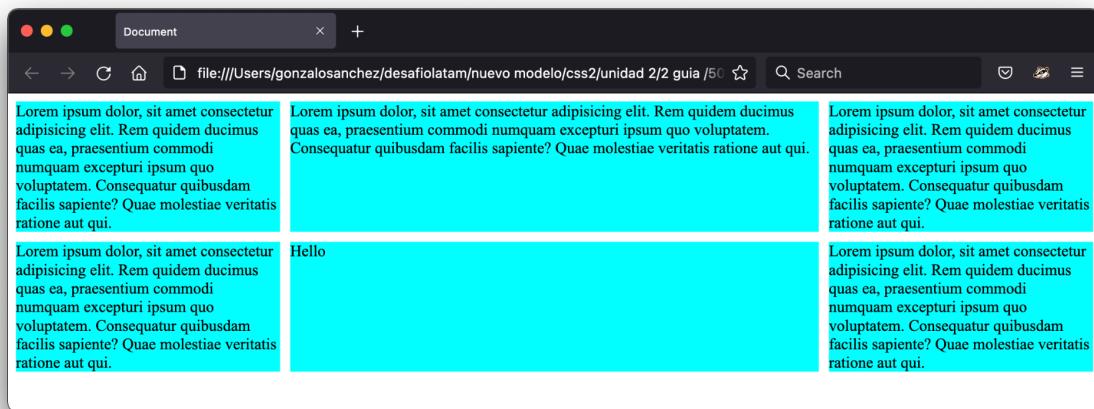


Imagen 12. Screenshot especificando grid-template-column con fracciones de 1fr, 2fr y 1 fr.

Fuente: Desafío Latam.

Mezclando fracciones y otras unidades de medida

Las fracciones se calculan sobre el espacio disponible libre. Por lo que grid-template-columns:50% 1fr 1fr; significa 3 columnas, la primera ocupa el 50% del espacio, y el 50% restante se divide en 2 fracciones en partes iguales, o sea 25% cada una.

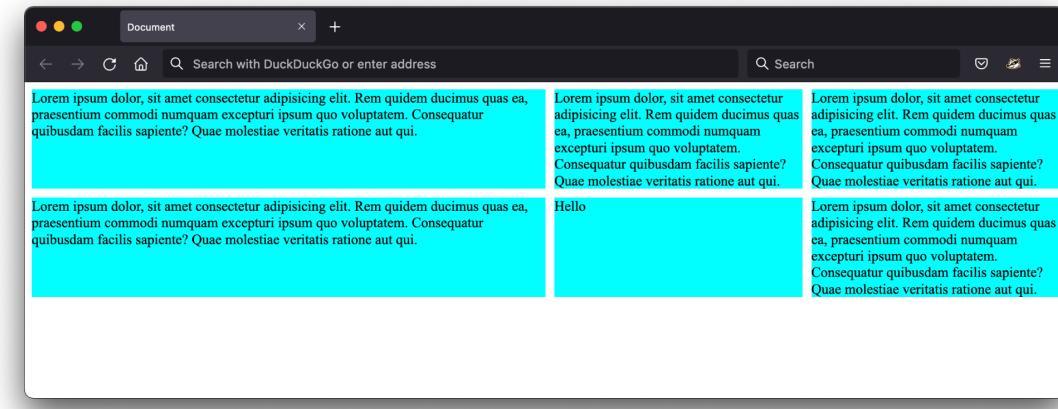


Imagen 13. Screenshot especificando grid-template-column con unidades de medida mixtas.

Fuente: Desafío Latam.

repeat

En algunas situaciones vamos a necesitar crear múltiples columnas, para este tipo de casos será más cómodo utilizar la instrucción repeat.

Utilizando el mismo ejemplo anterior, podemos cambiar la propiedad grid-template-column por grid-template-columns: repeat(4, 1fr) y de esta forma generar 4 columnas de 1 fracción.

También podemos utilizar otras unidades de medida como repeat(5, 20%) para obtener 5 columnas del 20%.

Otra opción de uso es que solo algunos elementos se repitan, supongamos que queremos una columna especial al principio y el resto distintas, en ese caso, podríamos hacer lo siguiente: grid-template-columns: 3fr repeat(3, 1fr). Así, tendríamos una columna de 3 partes de un total de 6, o sea un 50% y luego el otro 50% dividido en 3 partes iguales.

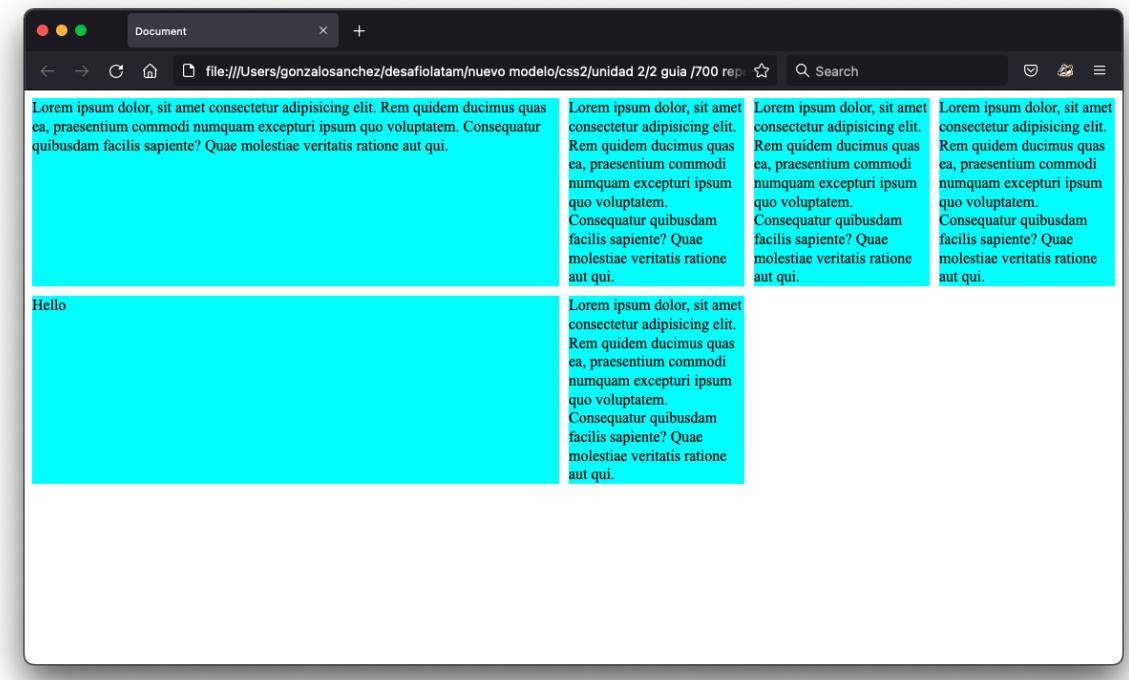


Imagen 14. Screenshot especificando grid-template-column con repeat.

Fuente: Desafío Latam.

Creando un layout con CSS Grid

Partiremos creando un layout sencillo. Una página con header y footer, un menú lateral y una sección de contenido principal:

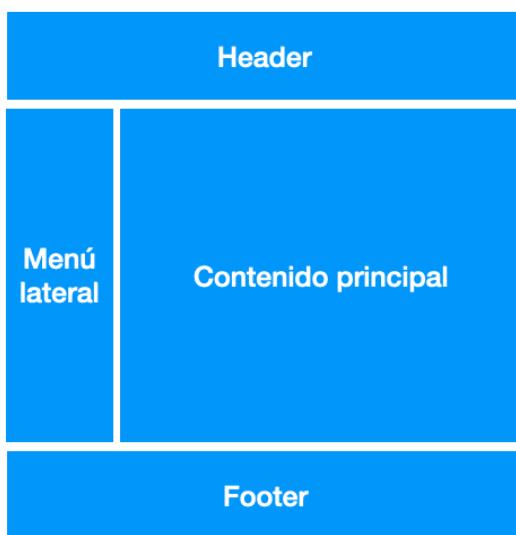


Imagen 15. Layout genérico de 2 columnas.

Fuente: Desafío Latam.

Este layout puede ser útil tanto como para una página informativa, un blog o incluso un sitio e-commerce. La separación entre las partes es solo para aclarar en esta guía donde empieza uno y termina otro.

Para construir el layout haremos uso de dos propiedades nuevas, grid-area y grid-template-areas. Con grid-area le daremos un nombre a cada una de las áreas y con grid-template-areas especificaremos a qué parte del layout corresponde cada nombre.

Primero crearemos los elementos básicos del HTML:

```
<div class="grid-container">
  <header> Header </header>
  <menu> Menú lateral </menu>
  <main> Contenido principal </main>
  <footer> Footer </footer>
</div>
```

Luego, con la propiedad grid en CSS, especificamos un nombre a cada área de nuestro layout. De paso, agregamos un color de fondo para poder visualizar de forma sencilla cada una de estas partes.

```
header{  
    grid-area: header;  
    background-color: #66f;  
}  
menu{  
    grid-area: menu;  
    background-color: #66f;  
}  
main{  
    grid-area: main;  
    background-color: #66f;  
}  
footer{  
    grid-area: footer;  
    background-color: #66f;  
}
```

A continuación del mismo archivo CSS, agregamos el contenedor grid. Aquí tenemos que añadir la propiedad display:grid y grid-template-area especificando el layout, de paso agregamos un gap para separar un poco las secciones y que sea más sencillo diferenciarlas visualmente.

```
.grid-container{  
    display: grid;  
    grid-template-areas:  
        "header header"  
        "menu main"  
        "footer footer";  
    gap: 10px;  
}
```

La propiedad grid-template-area la explicaremos un poco más adelante, por ahora observemos el efecto. Al cargar la página veremos lo siguiente:

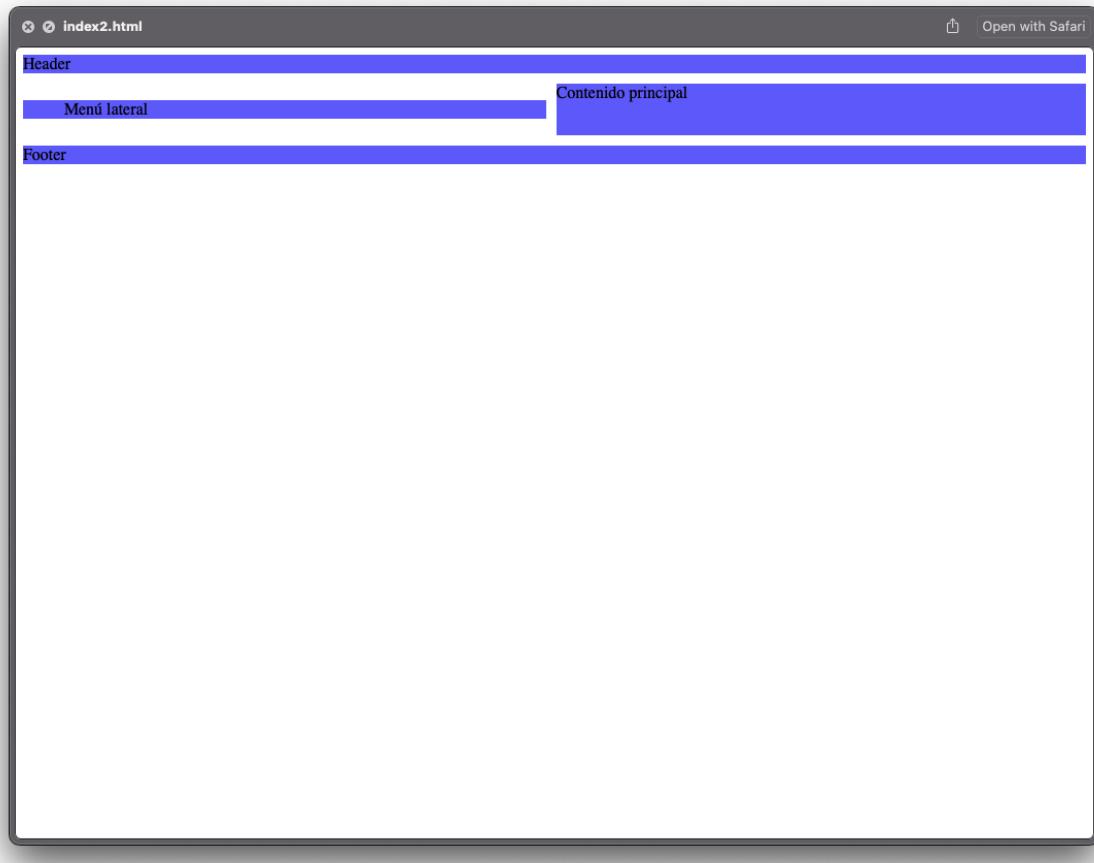


Imagen 16. Screenshot del layout creado de dos columnas con CSS Grid.

Fuente: Desafío Latam.

Con el inspector de elementos podemos observar que el texto Menú lateral tiene un poco de padding, dado que es agregado por la etiqueta <menu>, pero lo podemos eliminar fácilmente utilizando CSS, por ahora ignorémoslo.

Cambiando el orden del layout

Supongamos ahora que cambiamos de opinión, y ya no queremos el menú lateral a la izquierda, sino que a la derecha, para lograrlo podemos rápidamente ordenar el layout solo cambiando el CSS.

```
.grid-container{  
  display: grid;  
  grid-template-areas:  
  "header header"  
  "main menu"  
  "footer footer";  
  gap: 10px;
```

}



Imagen 17. Screenshot del layout creado de dos columnas con CSS Grid con orden invertido.

Fuente: Desafío Latam.

Grid-template-area

Con la propiedad `grid-template-area` podemos especificar donde queremos que vaya cada sección de nuestro sitio.

En la propiedad, cada fila va dentro de una comilla, la primera columna está representada por la primera palabra de cada fila:

```
"header header"  
"main menu"
```

Esto significa que el sitio tiene 2 columnas y dos filas, el header ocupa una fila y dos columnas, y luego en la segunda fila se divide entre main y menú.

```
.grid-container{  
    display: grid;
```

```
grid-template-areas:  
  "header header"  
  "main menu"  
  "footer footer";  
  gap: 10px;  
}
```

Probemos otro ejemplo similar:

```
.grid-container{  
  display: grid;  
  grid-template-areas:  
    "header header header"  
    "main main menu";  
  gap: 10px;  
}
```

En este último ejemplo, el sitio tiene 3 columnas, donde el header utiliza todas las de la primera fila, y la segunda fila se divide entre main y menú. Luego, si queremos cambiar el orden del menú y la sección principal, simplemente cambiamos "main main menu" por "menu main main"

Es muy importante que cada fila tenga la misma cantidad de elementos, por ejemplo el siguiente código resultará mal:

```
.grid-container{  
  display: grid;  
  grid-template-areas:  
    "header header"  
    "main main menu";  
  gap: 10px;  
}
```



Actividad 4

- Ya no queremos un menú lateral, sino uno justo debajo del header.
- Para esto, crea un grid-template-area con una sola palabra por fila.

El resultado debería ser siguiente:



Imagen 18. Resultado esperado de la actividad.

Fuente: Desafío Latam.



Actividad 5

- Crear el siguiente layout:

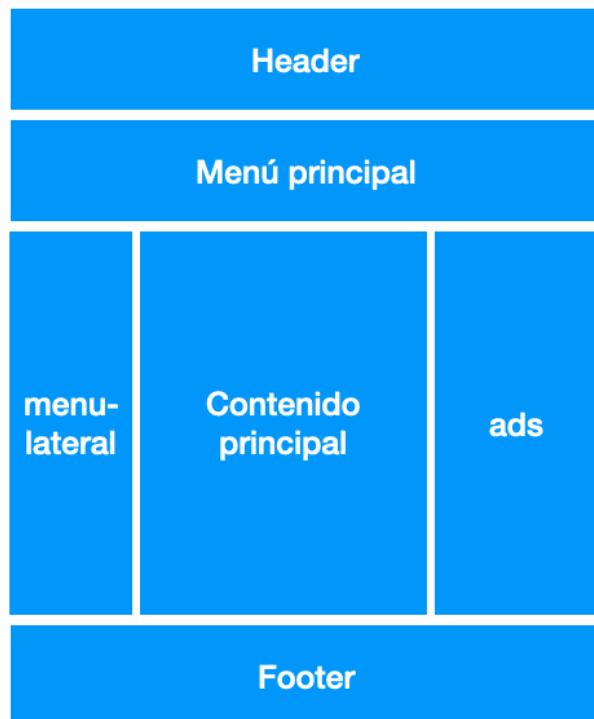


Imagen 19. Resultado esperado de la actividad.

Fuente: Desafío Latam.

Para lograrlo:

1. Crea un archivo nuevo con la base de HTML.
2. Dentro del body, agrega un contenedor grid.
3. Dentro del contenedor grid, agrega cada una de las partes del layout (todas serán etiquetas hermanas), puedes usar divs, etiquetas semánticas a la combinación que estimes conveniente.
4. Agrega la propiedad grid al contenedor grid.
5. Agrega la propiedad grid-area:"nombre de parte del layout" a cada una de las partes del layout.
6. Agrega la propiedad grid-template-areas: al contenedor grid, y dentro de las comillas agrega cada una de las partes.
 - a. Pista importante ¿Cuántas columnas hay en el layout?
7. Para visualizar mejor los resultados:
 - a. Agrega un color de fondo a cada una de las partes.
 - b. Agrega gap al contenedor grid.

Creando una galería de imágenes con grid-template-areas

Utilizando la propiedad grid-template-areas podemos crear una galería de imágenes bastante flexible como la siguiente:

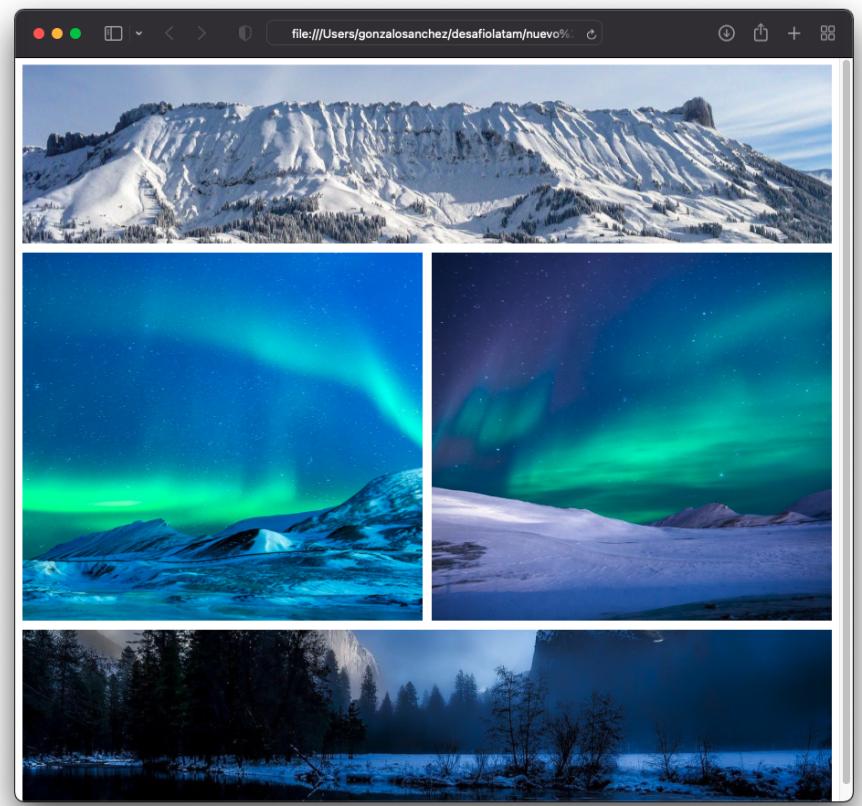


Imagen 20. Galería de imágenes con CSS Grid.

Fuente: Desafío Latam.

El truco será utilizar imágenes de fondo con la propiedad background-image: url(""), en lugar de agregarlas con la etiqueta img.

Para lograrlo, crearemos un nuevo HTML con la base y dentro del body:

```
<section class="gallery">
  <div class="imagen1 bg"></div>
  <div class="imagen2 bg"></div>
  <div class="imagen3 bg"></div>
  <div class="imagen4 bg"></div>
</section>

.bg {
  background-position: center;
  background-size: cover;
}

.imagen1 {
  grid-area: a;
  background-image: url("1.jpg");
}

.imagen2 {
  grid-area: b;
  background-image: url("2.jpg");
}

.imagen3 {
  grid-area: c;
  background-image: url("3.jpg");
}

.imagen4 {
  grid-area: d;
  background-image: url("4.jpg");
}

.gallery {
  height: 100vh;
  display: grid;
  gap: 10px;
  grid-template-areas:
    "d d d d"
    "a a b b"
    "a a b b"
    "c c c c";
}
```

Adicionalmente, podemos utilizar la propiedad background-position y background-size para mostrar lo mejor de nuestras imágenes, usualmente la combinación de valores background-position: center y background-size: cover es la mejor, pero en algunos casos podría ser conveniente cambiarla para todas o para alguna de las imágenes en específico, prueba con imágenes en tu computador y con el inspector de elementos.

Otro detalle es que en este caso le dimos un alto a la galería de 100vh, o sea el total del alto del navegador, pero también pudimos haber establecido un tamaño distinto.