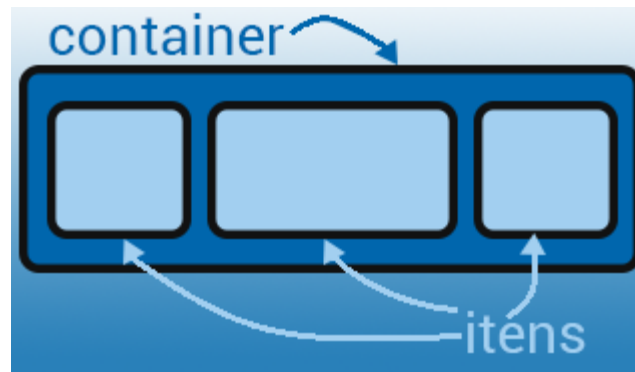


# O Que É CSS Grid?

O CSS Grid é um sistema de estruturação de layout que o CSS nos fornece. Diferente do Flexbox, que apenas nos permite trabalhar em uma única dimensão, o CSS Grid nos permite configurar layouts em duas dimensões (linhas e colunas). A junção de linhas e colunas formam uma grade, o que dá o nome a esse sistema (Grid).

O CSS Grid é bem poderoso e completo, permitindo que certas coisas sejam feitas de mais de uma maneira, o que pode confundir muitos iniciantes. Portanto, aqui irei mostrar as principais partes da forma mais simples para que você já termine de ler este post sabendo trabalhar com o CSS Grid.

Primeiro de tudo temos que saber que teremos propriedades CSS para trabalhar com o elemento que possui nossos itens (grid container ou elemento pai) e propriedades para os nossos itens da grid (elementos filhos).



## Grid Container

Neste primeiro post veremos as propriedades que ficam no elemento pai dos nossos itens, o grid container. Teremos basicamente um container com alguns itens dentro. Dependendo do exemplo a quantidade de itens poderá ser alterada...

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

## Display

Primeiro precisamos definir que o nosso container é do tipo “grid”; Fazemos isso com a propriedade “display”.

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

Isso irá criar uma grid do tipo `block`, significando que o `.container` irá ocupar a linha inteira. Caso você queira containers `inline`, utilize `display: inline-grid`;

## grid-template-columns e grid-template-rows

Como estamos trabalhando com uma grade, precisamos indicar quantas linhas e quantas colunas nossa grade terá. Para colunas usamos o `grid-template-columns` e para linhas usamos o `grid-template-rows`.

Se queremos 3 colunas, não indicamos com o número três. O que fazemos na verdade é indicar qual o tamanho de cada coluna que queremos. A medida pode ser em qualquer unidade (px, em, %, cm, etc). Cada valor deve ser separado por um espaço...

Result [Edit in JSFiddle](#)

```
display: grid;
grid-template-columns: 120px
```

1	
2	
3	
4	
5	

Ou...

Result [Edit in JSFiddle](#)

```
.container {
display: grid;
grid-template-columns: 50% 80px
```

1	2	
3	4	
5	6	

Ou então...

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: 20% 50px 1fr;  
}
```

1	2	3
4	5	6

Exatamente da mesma maneira nós declaramos as linhas utilizando a propriedade `grid-template-rows`.

Result [Edit in JSFiddle](#)

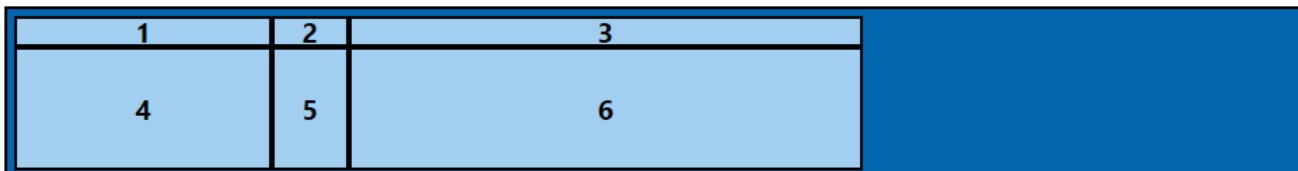
```
.container {  
  display: grid;  
  grid-template-columns: 20% 50px 40%;  
  grid-template-rows: 60px 30px;  
}
```

1	2	3	
4	5	6	

Ou...

```
Result Edit in JSFiddle

.container {
  display: grid;
  grid-template-columns: 20% 50px 40%;
  grid-template-rows: 20px 80px
}
```



## grid-template-areas

Até agora só declaramos o tamanho das nossas colunas e linhas. Você deve ter notado que nossos itens vão se organizando na grid de acordo com a ordem em que estão no HTML.

Porém, o CSS Grid nos permite ir além disso: podemos dar nomes para cada área da nossa grade e depois indicar onde cada elemento deve ir. Dessa maneira fica bem simples de entender a estrutura da nossa grid. Fazemos isso com a propriedade `grid-template-areas`. Vamos alterar um pouquinho o nosso HTML. Iremos colocar as classes `header`, `sidenav`, `content` e `footer` nos itens da grid para podermos indicar a posição de cada um deles depois.

```
<div class="container" >
  <div class="item header" ></div>
  <div class="item sidenav" ></div>
  <div class="item content" ></div>
  <div class="item footer" ></div>
</div>
```

Para cada item, vamos declarar a propriedade `grid-area`, que serve para indicar o nome da área em que cada elemento deverá ocupar na grid. Para facilitar eu dei o mesmo nome das classes, mas você pode dar o nome que quiser.

```
.header{
  grid-area: header;
}
.sidenav{
  grid-area: sidenav;
}
.content{
  grid-area: content;
}
.footer{
  grid-area: footer;
}
```

Agora vamos ao que interessa: indicar o nome das áreas da nossa grid. Para dar nomes às áreas de uma mesma linha, escrevemos dentro de aspas `""`. Ao abrir novas aspas estaremos indicando que estamos indo para a linha seguinte.

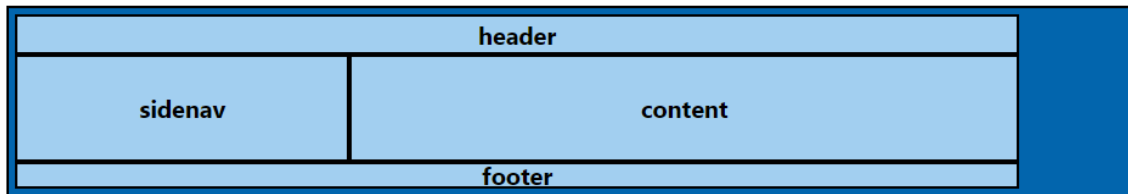
No exemplo abaixo nós escrevemos o nome das áreas em uma única linha. Mas para ficar mais simples de entender nós poderíamos ter escrito em linhas separadas. Fica visualmente mais legível:

```
.container{
  grid-template-areas: "header header"
                      "sidenav content"
                      "footer footer";
}
```

Repetimos `header` e `footer` duas vezes porque declaramos duas colunas e queremos que eles ocupem as duas...

```
Result Edit in JSFiddle

.container {
  display: grid;
  grid-template-columns: 30% 60%;
  grid-template-rows: 30px 80px 20px;
  grid-template-areas: "header header" "sidenav content" "footer footer"
}
```



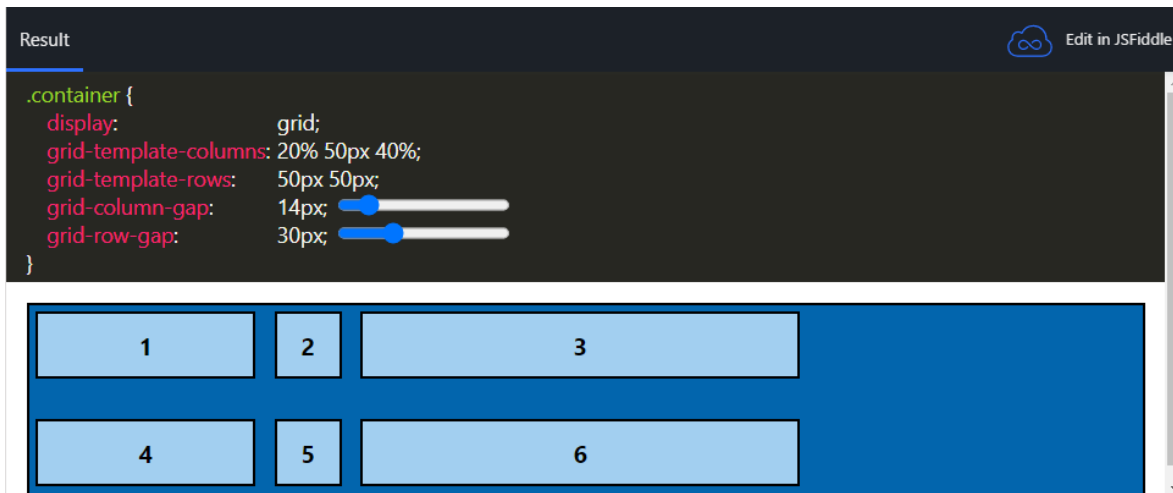
Experimente mudar os nomes das áreas de lugar e veja que nossos elementos irão mudar de posição também. Algumas sugestões:

- “sidenav header” “sidenav content” “sidenav footer”
- “header header” “content sidenav” “footer footer”
- “header header” “content sidenav” “footer sidenav”

Se quiser criar uma área sem nome, coloque `.`. Experimente: “header header” “content sidenav” “. footer”

## grid-column-gap, grid-row-gap e grid-gap

Nossos itens estão todos colados. Podemos indicar qual o espaço entre linhas e colunas. Para isso usamos as propriedades `grid-column-gap` e `grid-column-row-gap`...



O `grid-gap` é um atalho para declarar as duas propriedades em um só lugar. Se você passar um único valor, os dois serão aplicados para o espaço para linhas e colunas. Se passar dois valores, o primeiro será para linhas e o segundo para colunas.

```
.container{
  grid-gap: 20px;
  grid-gap: 20px 50px;
}
```

- O prefixo `grid-` destas propriedades está sendo removido e já é suportado nos navegadores modernos. Portanto, também pode-se utilizar as propriedades `column-gap`, `row-gap` e `gap`.

## justify-items

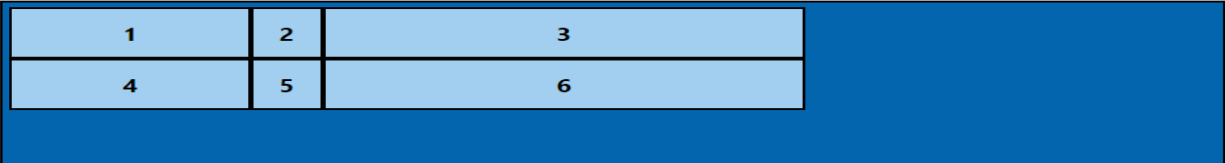
Alinha todos os itens da grid horizontalmente dentro de sua própria célula (área).

- **stretch (padrão):** estica os itens horizontalmente para eles preencherem sua célula...



Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: 20% 50px 40%;  
  grid-template-rows: 50px 50px;  
  justify-items: stretch  
}
```




A visual representation of a CSS Grid layout. The grid has two rows and three columns defined by the template: 20%, 50px, and 40%. The first row contains three items: '1' (20%), '2' (50px), and '3' (40%). The second row contains three items: '4' (20%), '5' (50px), and '6' (40%). The 'justify-items' property is set to 'stretch', which makes all items expand to fill the width of their respective columns. The background of the grid container is blue.

- **start:** alinha os itens no início de suas células

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: 20% 50px 40%;  
  grid-template-rows: 50px 50px;  
  justify-items: start  
}
```




A visual representation of a CSS Grid layout. The grid has two rows and three columns defined by the template: 20%, 50px, and 40%. The first row contains three items: '1' (20%), '2' (50px), and '3' (40%). The second row contains three items: '4' (20%), '5' (50px), and '6' (40%). The 'justify-items' property is set to 'start', which aligns all items to the left edge of their respective columns. The background of the grid container is blue.

- **end:** alinha os itens no final de suas células

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: 20% 50px 40%;  
  grid-template-rows: 50px 50px;  
  justify-items: end  
}
```



A visual representation of a CSS Grid layout. The grid has two rows and three columns defined by the template: 20%, 50px, and 40%. The first row contains three items: '1' (20%), '2' (50px), and '3' (40%). The second row contains three items: '4' (20%), '5' (50px), and '6' (40%). The 'justify-items' property is set to 'end', which aligns all items to the right edge of their respective columns. The background of the grid container is blue.

- **center:** alinha os itens no centro de suas células

Esse mesmo princípio irá acontecer para align-items, place-items, com pequenas diferenças...

## align-items

Alinha todos os itens da grid verticalmente dentro de sua própria célula (área).

- **stretch (padrão):** estica os itens verticalmente para eles preencherem sua célula
- **start:** alinha os itens na parte de cima de suas células
- **end:** alinha os itens na parte de baixo de suas células
- **center:** alinha os itens no centro de suas células

## place-items

Atalho para `align-items` e `justify-items` em uma única declaração.

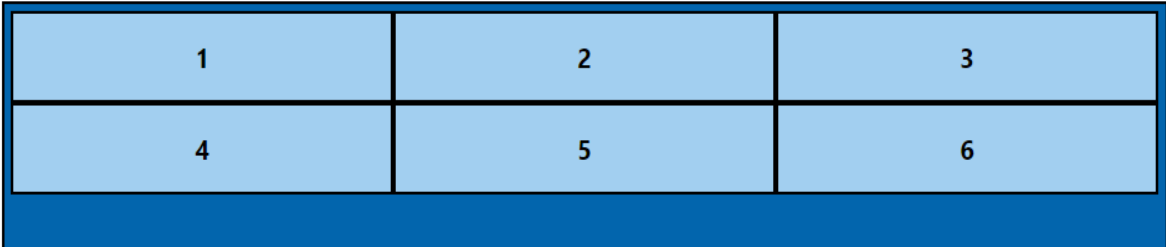
## justify-content

O `justify-item` alinha cada item dentro de sua própria área. Já o `justify-content` alinha horizontalmente as áreas em relação ao container.

- **stretch:** redimensiona os itens da grid para preencherem a largura do container. Apenas funciona se você não tiver declarado uma largura para as colunas...

Result [Edit in JSFiddle](#)


```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, auto);  
  grid-template-rows: 60px 60px;  
  justify-content: stretch;  
}
```



- **start:** alinha as áreas no início da grid

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, auto);  
  grid-template-rows: 60px 60px;  
  justify-content: start;  
}
```



- **end:** alinha as áreas no final da grid

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, auto);  
  grid-template-rows: 60px 60px;  
  justify-content: end;  
}
```

- **center:** alinha as áreas no centro da grid

Result [Edit in JSFiddle](#)

```
.container {  
  display: grid;  
  grid-template-columns: repeat(3, auto);  
  grid-template-rows: 60px 60px;  
  justify-content: center;  
}
```

- **space-around:** distribui as áreas com o espaço a sua volta
- **space-between:** distribui as áreas com o espaço entre elas
- **space-evenly:** distribui o espaço igualmente entre as áreas

## align-content

O `align-item` alinha cada item dentro de sua própria área. Já o `align-content` alinha verticalmente as áreas em relação ao container.

- **stretch:** redimensiona os itens da grid para preencherem a altura do container. Apenas funciona se você não tiver declarado uma altura para as linhas.
- **start:** alinha as áreas no topo da grid
- **end:** alinha as áreas na base da grid
- **center:** alinha as áreas no centro da grid
- **space-around:** distribui as áreas com o espaço a sua volta
- **space-between:** distribui as áreas com o espaço entre elas
- **space-evenly:** distribui o espaço igualmente entre as áreas

## place-content

Atalho para `align-content` e `justify-content` em uma única declaração.