

Grid

O leiaute responsivo da grade do Material Design se adapta ao tamanho e orientação da tela, garantindo a consistência entre leiautes.

Uma [grade](#) cria consistência visual entre leiautes, enquanto permite flexibilidade em uma ampla variedade de projetos. A UI responsiva do Material Design é baseada em uma grade de 12 colunas.

```
{{"component": "modules/components/ComponentLinkHeader.js"}}
```

O componente `Grid` não deve ser confundido com um data grid; ele está mais próximo de um layout grid. Para um cabeçalho do data grid para [o componente `DataGrid`](#).

Como funciona

O sistema de grade é implementado com o componente `Grid` :

- Ela usa [o módulo CSS de box flexível](#) para alta flexibilidade.
- Existem dois tipos de leiautes: *contêineres* e *itens*.
- Larguras de itens são definidas em porcentagens, desse modo são sempre fluidas e dimensionadas de acordo os seus elementos pai.
- Itens têm preenchimento para criar o espaçamento entre itens individuais.
- Existem cinco pontos de quebra (breakpoints) na grade: xs, sm, md, lg e xl.
- Integer values can be given to each breakpoint, indicating how many of the 12 available columns are occupied by the component when the viewport width satisfies the [breakpoint constraints](#).

Se você é **novoo ou não está familiarizado com o flexbox**, nós recomendamos você a ler este [guia do Flexbox CSS-Tricks](#).

Grades fluídas

As grades fluídas usam colunas que dimensionam e redimensionam o conteúdo. Uma disposição fluida de malha pode usar pontos de ruptura para determinar se precisa mudar drasticamente.

Grade básica

As larguras das colunas é representada por um numero inteiro que varia de 1 a 12. Podemos aplica-las a qualquer ponto de ruptura para indicar quantas colunas são ocupadas pelo componente.

Um valor aplicado a um ponto de ruptura se aplica a todos os outros pontos maiores, a menos que seja alterado, como será visto posteriormente nesta página. Por exemplo nesse trecho `xs={12}` definimos que o componente ocupará toda a largura da tela, independente do tamanho da tela.

```
{{"demo": "BasicGrid.js", "bg": true}}
```

Grade com pontos de quebra

Podemos definir várias larguras para os componentes, resultando em um leiaute que muda de acordo com o ponto de ruptura. Os valores de largura dados para pontos de quebra maiores, substituem aqueles dados a pontos de quebra menores.

Aqui por exemplo `xs={12} sm={6}` define que o componente ocupará metade da largura da tela (6 colunas) em um dispositivo com resolução de 600px pra cima. Já para dispositivos menores, o componente vai tomar todas as 12 colunas.

```
{{"demo": "FullWidthGrid.js", "bg": true}}
```

Espaçamento

To control space between children, use the `spacing` prop. The spacing value can be any positive number, including decimals and any string. The prop is converted into a CSS property using the [theme.spacing\(\)](#) helper.

```
{{"demo": "SpacingGrid.js", "bg": true}}
```

Row & column spacing

As props `rowSpacing` e `columnSpacing` nos permite especificar os espaços entre linhas e entre colunas de forma independente. É similar as propriedades `row-gap` e `column-gap` da [Malha CSS](#).

```
{{"demo": "RowAndColumnSpacing.js", "bg": true}}
```


Valores responsivos

You can switch the props' value based on the active breakpoint. Podemos ativar as props de acordo com ponto de ruptura ativo Podemos ativar as props de acordo com ponto de ruptura ativo Por exemplo, podemos implementar o leiaute responsivo do Material Design ["recommended"](#)

```
{{"demo": "ResponsiveGrid.js", "bg": true}}
```

Valores responsivos são suportados por:

- `colunas`
- `espaçoDeColuna`
- `direção`
- `espaçoDeLinha`
- `spacing`
- todas as outras propriedades do sistema [other props](#)

 Quando Usamos a prop de código responsivo `colunas`, cada item da manha precisa de seu correspondente ponto de ruptura. For instance, this is not working. The grid item misses the value for `md`:

```
<Grid container columns={{ xs: 4, md: 12 }}>
  <Grid item xs={2} />> >{ ' ' }
> </Grid>
```

Interativo

Abaixo está uma demonstração interativa que permite explorar os resultados visuais das diferentes configurações:

```
{{"demo": "InteractiveGrid.js", "hideToolbar": true, "bg": true}}
```

Leiaute Automático

O leiaute automático faz com que o espaço disponível seja compartilhado de forma proporcional *items* O leiaute automático faz com que o espaço disponível seja compartilhado de forma proporcional *items* Isso também quer dizer que podemos definir que a largura de um *item* se ajustará automaticamente ao redor dele That also means you can set the width of one *item* and the others will automatically resize around it.

```
{{"demo": "AutoGrid.js", "bg": true}}
```

Margem negativa

A demo a seguir não segue as normas do Material Design mas ilustra bem como a malha pode ser usada para criar layouts complexos

```
{{"demo": "VariableWidthGrid.js", "bg": true}}
```

Grade Complexa

As props `container` e `item` são booleanas independentes; Podem ser combinados para permitir que um componente Grid seja tanto um contêiner flex como um filho (item do contêiner)

```
{{"demo": "ComplexGrid.js", "bg": true}}
```

Grade Aninhada

As props `container` e `item` são booleanas independentes; Podem ser combinados para permitir que um componente Grid seja tanto um contêiner flex como um filho (item do contêiner)

Um **container flex** é a caixa gerada por um elemento com uma exibição definida por `flex` ou `inline-flex`. Os filhos em um fluxo de um container flex são chamados de flex **items** e são dispostos usando o modelo de layout flex (flex layout).

<https://www.w3.org/TR/css-flexbox-1/#box-model>

```
{{"demo": "NestedGrid.js", "bg": true}}
```

⚠ Definindo uma largura explícita para um elemento de grade que é um contêiner flexível, item flexível e tem espaçamento ao mesmo tempo, tem um comportamento inesperado, evite fazendo isto:

```
<Grid spacing={1} container item xs={12}>
```

Se você precisar fazer isso, remova uma das propriedades.

Colunas

You can change the default number of columns (12) with the `columns` prop.

```
{{"demo": "ColumnsGrid.js", "bg": true}}
```

Limitações

Margem negativa

Há uma limitação com a margem negativa que usamos para implementar o espaçamento entre itens. Isto pode levar a comportamentos inesperados. Por exemplo, para aplicar uma cor de fundo, você precisa aplicar `display: flex;` no elemento pai.

white-space: nowrap;

A configuração inicial em itens flexíveis é `min-width: auto` . Isto causa um conflito de posicionamento quando os elementos filhos estão usando `white-space: nowrap` . Você pode enfrentar o problema com:

```
<Grid item xs>
  <Typography noWrap>
```

In order for the item to stay within the container you need to set `min-width: 0` . In practice, you can set the `zeroMinWidth` prop:

```
<Grid item xs zeroMinWidth>
  <Typography noWrap>
```

```
{{"demo": "AutoGridNoWrap.js", "bg": true}}
```

direction: column | column-reverse

As propriedades `xs` , `sm` , `md` , `lg` , e `xl` **não são suportadas** com containers `direction="column"` e `direction="column-reverse"` .

Elas definem o número de grades que o componente usará para um determinado ponto de quebra. Elas destinam-se a controlar a **largura** usando `flex-basis` em contêineres `row` , mas elas irão impactar a altura em contêineres `column` . Se usadas, essas propriedades podem ter efeitos indesejáveis na altura dos elementos do item `Grid` .

Leiaute de Grade CSS

O componente `Grid` está usando o CSS flexbox internamente. Mas como visto abaixo, você pode facilmente usar [o sistema](#) e CSS Grid para a personalização de suas páginas.

```
{{"demo": "CSSGrid.js", "bg": true}}
```

System props

Como um componente CSS utilitário, o `Grid` suporta todas as propriedades [do sistema](#) . You can use them as props directly on the component. Por exemplo, um preenchimento:

```
<Grid item p={2}>
```