

# Sistema Material-UI

CSS utilities for rapidly laying out custom designs.

Material-UI comes with dozens of **ready-to-use** components in the core. Esses componentes são um ponto de partida incrível, mas quando se trata de fazer seu site se destacar com um design customizado, pode ser mais simples começar de um estado sem estilos. Apresentando o sistema:

O **sistema** permite que você crie rapidamente componentes de UI customizados utilizando os valores definidos no seu tema.

## Demonstração

*(Redimensione a janela para ver os pontos de quebra responsivos)*

```
{{"demo": "Demo.js", "bg": true, "defaultCodeOpen": true}}
```

## Instalação

```
// with npm
npm install @material-ui/system@next @emotion/react @emotion/styled

// with yarn
yarn add @material-ui/system@next @emotion/react @emotion/styled
```

Or if you want to use `styled-components` as a styling engine:

```
// with npm
npm install @material-ui/system@next @material-ui/styled-engine-sc@next styled-components

// with yarn
yarn add @material-ui/system@next @material-ui/styled-engine-sc@next styled-components
```

Dê uma olhada no [guia do Motor de Estilização](#) para mais informações sobre como configurar `componentes estilizados` como seu motor de estilização.

## Por que usar o sistema?

Compare como o mesmo componente de estatística pode ser construído com duas APIs diferentes.

```
{{"demo": "Why.js", "bg": true, "defaultCodeOpen": false}}
```

1. ❌ usando a API do styled-components:

```
const StatWrapper = styled('div') (
  ({ theme }) => `
    background-color: ${theme.palette.background.paper};
    box-shadow: ${theme.shadows[1]};
```

```

    border-radius: ${theme.shape.borderRadius}px;
    padding: ${theme.spacing(2)};
    min-width: 300px;
  },
);

const StatHeader = styled('div')(
  ({ theme }) => `
    color: ${theme.palette.text.secondary};
  `,
);

const StyledTrend = styled(TrendingUpIcon)(
  ({ theme }) => `
    color: ${theme.palette.success.dark};
    font-size: 16px;
    vertical-align: sub;
  `,
);

const StatValue = styled('div')(
  ({ theme }) => `
    color: ${theme.palette.text.primary};
    font-size: 34px;
    font-weight: ${theme.typography.fontWeightMedium};
  `,
);

const StatDiff = styled('div')(
  ({ theme }) => `
    color: ${theme.palette.success.dark};
    display: inline;
    font-weight: ${theme.typography.fontWeightMedium};
    margin-left: ${theme.spacing(0.5)};
    margin-right: ${theme.spacing(0.5)};
  `,
);

const StatPrevious = styled('div')(
  ({ theme }) => `
    color: ${theme.palette.text.secondary};
    display: inline;
    font-size: 12px;
  `,
);

return (
  <StatWrapper>
    <StatHeader>Sessões</StatHeader>
    <StatValue>98.3 K</StatValue>
    <StyledTrend />
    <StatDiff>18.77%</StatDiff>
  </StatWrapper>
);

```

```
    <StatPrevious>em relação ultima semana</StatPrevious>
  </StatWrapper>
);
```

## 2. usando o sistema:

```
<Box
  sx={{
    bgcolor: 'background.paper',
    boxShadow: 1,
    borderRadius: 1,
    p: 2,
    minWidth: 300,
  }}
>
<Box sx={{ color: 'text.secondary' }}>Sessions</Box>
<Box sx={{ color: 'text.primary', fontSize: 34, fontWeight: 'medium' }}>
  98.3 K
</Box>
<Box
  component={TrendingUpIcon}
  sx={{ color: 'success.dark', fontSize: 16, verticalAlign: 'sub' }}
/>
<Box
  sx={{
    color: 'success.dark',
    display: 'inline',
    fontWeight: 'medium',
    mx: 0.5,
  }}
>
  18.77%
</Box>
<Box sx={{ color: 'text.secondary', display: 'inline', fontSize: 12 }}>
  vs. last week
</Box>
</Box>
```

## Problema resolvido

O sistema foca na resolução de 3 principais problemas:

### 1. Mudar de contexto desperdiça tempo.

Não há necessidade de saltar constantemente entre o uso dos componentes customizados e onde eles são definidos. Com o sistema, essas descrições estão corretas onde é necessário estar.

### 2. Nomear as coisas é difícil.

Você já se encontrou com dificuldades para encontrar um bom nome para um componente customizado? O sistema mapeia os estilos diretamente para o elemento. Tudo o que você precisa fazer é se preocupar com as propriedades de estilo atuais.

### 3. Manter consistência nas UI é difícil.

Isso é especialmente verdadeiro quando mais de uma pessoa está construindo a aplicação, já que tem que haver alguma coordenação entre os membros da equipe sobre a escolha dos tokens de design e como eles são usados, quais partes da estrutura do tema devem ser usadas com quais propriedades CSS e assim por diante.

O sistema oferece acesso direto ao valor no tema. Fica mais fácil de lidar com restrições.

## A propriedade `sx`

A propriedade `sx`, como a parte principal do sistema, resolve esses problemas, fornecendo uma maneira rápida & simples de aplicar os tokens de design corretos para propriedades CSS específicas diretamente a um elemento React. A [demonstração acima](#) mostra como ela pode ser usada para criar um design único.

This prop provides a superset of CSS (contains all CSS properties/selectors in addition to custom ones) that maps values directly from the theme, depending on the CSS property used. Além disso, permite uma maneira simples de definir valores responsivos que correspondem aos pontos de quebra definidos no tema. Para mais detalhes, visite a [página da propriedade `sx`](#).



### Quando usar ela?

- **styled-components:** a API é excelente para construir componentes que precisam suportar uma ampla variedade de contextos. Estes componentes são usados em diversos locais da aplicação e suportam diferentes combinações de propriedades.
- **propriedade `sx`:** a API é excelente para aplicar estilos pontuais. É chamado de "utilitário" por esse motivo.

### Desempenho

O sistema depende do CSS-in-JS. Funciona com ambos, emotion e styled-components.

Prós:

-  Permite uma grande flexibilidade na API. A propriedade `sx` suporta um super conjunto de CSS. Não há **nenhuma necessidade de aprender CSS duas vezes**. Uma vez que você aprendeu a sintaxe padronizada do CSS, é seguro pois, não mudou durante uma década. Então, você pode **opcionalmente** aprender os atalhos, se você valoriza a economia de tempo que eles trazem.
-  Auto-purge. Somente o CSS usado na página é enviado para o cliente. O custo inicial do tamanho do pacote é **fixo**. Ele não aumenta com o número de propriedades CSS usadas. You pay the cost of [@emotion/react](#) and [@mui/system](#). Você paga o custo de [@emotion/react](#) e [@material-ui/system](#). Custa cerca de ~15 kB gzipped.

Contras:

- O desempenho em tempo de execução é impactado.

Benchmark	Fragmento de código	Tempo normalizado
c. Renderizar 1,000 styled components	<code>&lt;div className="..."&gt;</code>	100ms
b. Renderizar 1.000 componentes	<code>&lt;Div&gt;</code>	120ms
c. Render 1,000 styled components	<code>&lt;StyledDiv&gt;</code>	160ms
d. a. Renderizar 1.000 Box	<code>&lt;Box sx={...}&gt;</code>	370ms

Head to the [benchmark folder](#) for a reproduction of these metrics.

Nós acreditamos que para a maioria das situações é **rápido o suficiente**, mas há soluções alternativas simples onde a performance se torna crítica. Por exemplo, ao renderizar uma lista com muitos itens, você pode usar um seletor filho CSS para ter um único ponto de "injeção de estilo" (usando `d.` para o wrapper e `a.` para cada item).

## API tradeoff

Having the system under one prop ( `sx` ) helps to differentiate props defined for the sole purpose of CSS utilities, vs. those for component business logic. It's important for the **separation of concerns**. For instance, a `color` prop on a button impacts multiple states (hover, focus, etc.), not to be confused with the color CSS property.

Only the `Box` , `Stack` , `Typography` , and `Grid` components accept the system properties as *props* for the above reason. These components are designed to solve CSS problems, they are CSS component utilities.

## Uso

### Tokens de design no tema

Você pode explorar a página de [Propriedades do sistema](#) para descobrir como as diferentes propriedades do CSS (e customizadas) são mapeadas para as chaves do tema.

### Abreviações

Existem muitas abreviações disponíveis para as propriedades do CSS. Estas são documentadas nas próximas páginas, por exemplo, [o espaçamento](#). Aqui está um exemplo demonstrando-as:

```
<Box
  sx={{
    boxShadow: 1, // theme.shadows[1]
    color: 'primary.main', // theme.palette.primary.main
    m: 1, // margin: theme.spacing(1)
    p: {
      xs: 1, // [theme.breakpoints.up('xs')]: { padding: theme.spacing(1) }
    },
    zIndex: 'tooltip', // theme.zIndex.tooltip
  }}
>
```

Estas abreviações são **opcionais**, elas são ótimas para economizar tempo quando escrevemos estilos, mas pode ser frustrante aprender novas APIs customizadas. Talvez você queira pular essa parte e apostar em CSS, ele está padronizado há décadas, vá para a [próxima seção](#).

## Super conjunto de CSS

Como parte da propriedade, você pode usar qualquer CSS normalmente: seletores filhos ou pseudo seletores, consultas de mídia, valores CSS brutos, etc. Aqui estão alguns exemplos: Here are a few examples:

- Usando pseudo seletores:

```
<Box
  sx={{
    // some styles
    ":hover": {
      boxShadow: 6,
    },
  }}
>
```

- Usando consultas de mídia:

```
<Box
  sx={{
    // some styles
    '@media print': {
      width: 300,
    },
  }}
>
```

- Usando seletor aninhado:

```
<Box
  sx={{
    // some styles
    '& . ChildSelector': {
      bgcolor: 'primary.main',
    },
  }}
>
```

## Valores responsivos

Se você quiser ter valores responsivos para uma propriedade CSS, você pode usar a sintaxe abreviada de pontos de quebra. Há duas maneiras de definir os pontos de quebra:

### 1. Pontos de quebra como um objeto


A primeira opção para definir pontos de quebra é defini-los como um objeto, usando os pontos de quebra como chaves. Note that each breakpoint property matches the breakpoint and every larger breakpoint. For example, `width: { lg: 100 }` is equivalent to `theme.breakpoints.up('lg')`. Aqui está o exemplo anterior novamente, usando a sintaxe do objeto.

```
{{"demo": "BreakpointsAsObject.js"}}
```

### 2. Pontos de quebra como um array

A segunda opção é definir seus pontos de quebra como um array, do menor ao maior ponto de quebra.

```
{{"demo": "BreakpointsAsArray.js"}}
```

 Esta opção só é recomendada quando o tema tem um número limitado de pontos de quebra, p. ex. 3. Prefira a API de objeto se você tiver mais pontos de quebra. Por exemplo, o tema padrão do Material-UI tem 5.

Você pode ignorar pontos de quebra usando o valor como `null` :

```
<Box sx={{ width: [null, null, 300] }}>Este box tem uma largura responsiva.</Box>
```

## Pontos de quebra customizados

Você também pode especificar seus próprios pontos de quebras customizados, e usá-los como chaves ao definir o objeto de pontos de quebra. Aqui está um exemplo de como o fazer.

```
import * as React from 'react';
import Box from '@material-ui/core/Box';
import { createTheme, ThemeProvider } from '@material-ui/core/styles';

const theme = createTheme({
  breakpoints: {
    values: {
      mobile: 0,
      tablet: 640,
      laptop: 1024,
      desktop: 1280,
    },
  },
});

export default function CustomBreakpoints() {
  return (
    <ThemeProvider theme={theme}>
      <Box
        sx={{
          width: {
            mobile: 100,
            laptop: 300,
          },
        }}
      >
        This box has a responsive width
      </Box>
    </ThemeProvider>
  );
}
```

Se você estiver usando TypeScript, você também deverá usar a [extensão de módulos](#) para que o tema aceite os valores acima.

```
declare module '@material-ui/core/styles/createBreakpoints' {
  interface BreakpointOverrides {
    xs: false; // removes the `xs` breakpoint
    sm: false;
```

```
md: false;
lg: false;
xl: false;
tablet: true; // adds the `tablet` breakpoint
laptop: true;
desktop: true;
}
}
```

## Recuperando o tema

Se você deseja usar o tema para uma propriedade CSS que não é suportada nativamente pelo sistema, você pode usar uma função como valor, no qual você pode acessar o objeto do tema.

```
{{"demo": "ValueAsFunction.js"}}
```

## Implementações

A propriedade `sx` pode ser usada em quatro locais diferentes:

### 1. Componentes do core

Todos os componentes Material-UI do core suportarão a propriedade `sx`.

### 2. Box

[Box](#) é um componente leve que dá acesso a propriedade `sx`, e pode ser usado como um componente utilitário, e como um encapsulador para outros componentes. Ele renderiza um elemento `<div>` por padrão.

### 3. Componentes customizados

In addition to Material-UI components, you can add the `sx` prop to your custom components too, by using the `styled` utility from `@material-ui/core/styles`.

```
import { styled } from '@material-ui/core/styles';

const Div = styled('div')``;
```

### 4. Qualquer elemento com o plugin babel

A fazer [#23220](#).