Grid

The Material Design responsive layout grid adapts to screen size and orientation, ensuring consistency across layouts.

The grid creates visual consistency between layouts while allowing flexibility across a wide variety of designs. Material Design's responsive UI is based on a 12-column grid layout.

{{"component": "modules/components/ComponentLinkHeader.js", "design": false}}

> ⚠️ *The `Grid` component shouldn't be confused with a data grid; it is closer to a layout grid. For a data grid head to the `DataGrid` component.*

How it works

The grid system is implemented with the `Grid` component:

- It uses CSS's Flexible Box module for high flexibility.
- There are two types of layout: *containers* and *items*.
- Item widths are set in percentages, so they're always fluid and sized relative to their parent element.
- Items have padding to create the spacing between individual items.
- There are five grid breakpoints: xs, sm, md, lg, and 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.

If you are **new to or unfamiliar with flexbox**, we encourage you to read this CSS-Tricks flexbox guide.

Fluid grids

Fluid grids use columns that scale and resize content. A fluid grid's layout can use breakpoints to determine if the layout needs to change dramatically.

Basic grid

Column widths are integer values between 1 and 12; they apply at any breakpoint and indicate how many columns are occupied by the component.

A value given to a breakpoint applies to all the other breakpoints wider than it (unless overridden, as you can read later in this page). For example, `xs={12}` sizes a component to occupy the whole viewport width regardless of its size.

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

Grid with multiple breakpoints

Components may have multiple widths defined, causing the layout to change at the defined breakpoint. Width values given to larger breakpoints override those given to smaller breakpoints.

For example, `xs={12} sm={6}` sizes a component to occupy half of the viewport width (6 columns) when viewport width is 600 or more pixels. For smaller viewports, the component fills all 12 available columns.

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

Spacing

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

The `rowSpacing` and `columnSpacing` props allow for specifying the row and column gaps independently. It's similar to the `row-gap` and `column-gap` properties of CSS Grid.

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

Responsive values

You can switch the props' value based on the active breakpoint. For instance, we can implement the "recommended" responsive layout grid of Material Design.

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

Responsive values is supported by:

- `columns`
- `columnSpacing`
- `direction`
- `rowSpacing`
- `spacing`
- all the other props of the system

> ⚠️ *When using a responsive `columns` prop, each grid item needs its corresponding breakpoint. 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>
> ```

Interactive

Below is an interactive demo that lets you explore the visual results of the different settings:

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

Auto-layout

The Auto-layout makes the *items* equitably share the available space. That also means you can set the width of one *item* and the others will automatically resize around it.

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

Variable width content

Set one of the size breakpoint props to `"auto"` instead of `true` / a `number` to size a column based on the natural width of its content.

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

Complex Grid

The following demo doesn't follow the Material Design guidelines, but illustrates how the grid can be used to build complex layouts.

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

Nested Grid

The `container` and `item` props are two independent booleans; they can be combined to allow a Grid component to be both a flex container and child.

> *A flex **container** is the box generated by an element with a computed display of `flex` or `inline-flex`. In-flow children of a flex container are called flex **items** and are laid out using the flex layout model.*

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

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

⚠️ Defining an explicit width to a Grid element that is flex container, flex item, and has spacing at the same time lead to unexpected behavior, avoid doing it:

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

If you need to do such, remove one of the props.

Columns

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

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

Limitations

Negative margin

The spacing between items is implemented with a negative margin. This might lead to unexpected behaviors. For instance, to apply a background color, you need to apply `display: flex;` to the parent.

white-space: nowrap

The initial setting on flex items is `min-width: auto`. It's causing a positioning conflict when the children is using `white-space: nowrap;`. You can experience the issue with:

```
<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

The `xs`, `sm`, `md`, `lg`, and `xl` props are **not supported** within `direction="column"` and `direction="column-reverse"` containers.

They define the number of grids the component will use for a given breakpoint. They are intended to control **width** using `flex-basis` in `row` containers but they will impact height in `column` containers. If used, these props may have undesirable effects on the height of the `Grid` item elements.

CSS Grid Layout

The `Grid` component is using CSS flexbox internally. But as seen below, you can easily use the system and CSS Grid to layout your pages.

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

System props

As a CSS utility component, the `Grid` supports all `system` properties. You can use them as props directly on the component. For instance, a padding:

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