Tabs

Tabs make it easy to explore and switch between different views.

Tabs organize and allow navigation between groups of content that are related and at the same level of hierarchy.

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

Basic tabs

A basic example with tab panels.

{{"demo": "BasicTabs.js"}}

Experimental API

`@mui/lab` offers utility components that inject props to implement accessible tabs following WAI-ARIA authoring practices.

{{"demo": "LabTabs.js"}}

Wrapped labels

Long labels will automatically wrap on tabs. If the label is too long for the tab, it will overflow, and the text will not be visible.

{{"demo": "TabsWrappedLabel.js"}}

Colored tab

{{"demo": "ColorTabs.js"}}

Disabled tab

A tab can be disabled by setting the `disabled` prop.

{{"demo": "DisabledTabs.js"}}

Fixed tabs

Fixed tabs should be used with a limited number of tabs, and when a consistent placement will aid muscle memory.

Full width

The `variant="fullWidth"` prop should be used for smaller views. This demo also uses react-swipeable-views to animate the Tab transition, and allowing tabs to be swiped on touch devices.

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

Centered

The `centered` prop should be used for larger views.

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

Scrollable tabs

Automatic scroll buttons

By default, left and right scroll buttons are automatically presented on desktop and hidden on mobile. (based on viewport width)

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

Forced scroll buttons

Left and right scroll buttons be presented (reserve space) regardless of the viewport width with `scrollButtons={true}` `allowScrollButtonsMobile`:

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

If you want to make sure the buttons are always visible, you should customize the opacity.

```
.MuiTabs-scrollButtons.Mui-disabled {
  opacity: 0.3;
}
```

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

Prevent scroll buttons

Left and right scroll buttons are never be presented with `scrollButtons={false}`. All scrolling must be initiated through user agent scrolling mechanisms (e.g. left/right swipe, shift mouse wheel, etc.)

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

Customization

Here is an example of customizing the component. You can learn more about this in the overrides documentation page.

{{"demo": "CustomizedTabs.js"}}

🎨 If you are looking for inspiration, you can check MUI Treasury's customization examples.

Vertical tabs

To make vertical tabs instead of default horizontal ones, there is `orientation="vertical"`:

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

Note that you can restore the scrollbar with `visibleScrollbar`.

Nav tabs

By default, tabs use a `button` element, but you can provide your custom tag or component. Here's an example of implementing tabbed navigation:

{{"demo": "NavTabs.js"}}

Icon tabs

Tab labels may be either all icons or all text.

{{"demo": "IconTabs.js"}}

{{"demo": "IconLabelTabs.js"}}

Icon position

By default, the icon is positioned at the `top` of a tab. Other supported positions are `start`, `end`, `bottom`.

{{"demo": "IconPositionTabs.js"}}

Third-party routing library

One frequent use case is to perform navigation on the client only, without an HTTP round-trip to the server. The `Tab` component provides the `component` prop to handle this use case. Here is a more detailed guide.

Accessibility

(WAI-ARIA: https://www.w3.org/TR/wai-aria-practices/#tabpanel)

The following steps are needed in order to provide necessary information for assistive technologies:

1. Label `Tabs` via `aria-label` or `aria-labelledby`.
2. `Tab`s need to be connected to their corresponding `[role="tabpanel"]` by setting the correct `id`, `aria-controls` and `aria-labelledby`.

An example for the current implementation can be found in the demos on this page. We've also published an experimental API in `@mui/lab` that does not require extra work.

Keyboard navigation

The components implement keyboard navigation using the "manual activation" behavior. If you want to switch to the "selection automatically follows focus" behavior you have pass `selectionFollowsFocus` to the `Tabs` component. The WAI-ARIA authoring practices have a detailed guide on how to decide when to make selection automatically follow focus.

Demo

The following two demos only differ in their keyboard navigation behavior. Focus a tab and navigate with arrow keys to notice the difference, e.g. `Arrow Left`.

```
/* Tabs where selection follows focus */
<Tabs selectionFollowsFocus />
```

{{"demo": "AccessibleTabs1.js", "defaultCodeOpen": false}}

```
/* Tabs where each tab needs to be selected manually */
<Tabs />
```

{{"demo": "AccessibleTabs2.js", "defaultCodeOpen": false}}