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", "design": false}}

Basic tabs

Tabs are implemented using a handful of components, each with a specific purpose:

- TabUnstyled - an actual tab. Clicking on it displays the associated content.
- TabsListUnstyled - container for tabs. Responsible for handling focus and keyboard navigation between tabs.
- TabPanelUnstyled - hosts the content associated with a tab.
- TabsUnstyled - top-level component wrapping TabsListUnstyled and TabPanelUnstyled and letting them communicate.

```
import TabsUnstyled from '@mui/base/TabsUnstyled';
import TabsListUnstyled from '@mui/base/TabUnstyled';
import TabUnstyled from '@mui/base/TabUnstyled';
import TabPanelUnstyled from '@mui/base/TabPanelUnstyled';
```

You can associate a tab with tab panel using the `value` prop. If a tab with a given `value` is selected, a tab panel with a matching `value` will be shown.

Note that setting a `value` on a `TabUnstyled` is not mandatory. If you omit this prop, it will default to order of a tab within its parent TabsListUnstyled. This order is 0-based (the first tab has its `value` set to `0`, the next one `1`, etc.).

{{"demo": "UnstyledTabsBasic.js"}}

Customizing the root element

By default, the `TabUnstyled` renders a native `button` element. You are free to override this by setting the `component` or `components.Root` prop. If a non-interactive element (such as a span) is provided this way, the `TabUnstyled` will take care of adding accessibility attributes.

The `TabPanelUnstyled`, on the other hand, renders a native `div` element by default. You are free to override this as well by setting the `component` or `components.Root` prop on the `TabPanelUnstyled`.

{{"demo": "UnstyledTabsCustomized.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 `TabsUnstyled` via `aria-label` or `aria-labelledby`.
2. `TabUnstyled`s need to be connected to their corresponding tab panels 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.

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 Right`.

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

{{"demo": "KeyboardNavigation.js"}}