Tabs

Docusaurus provides the `<Tabs>` component that you can use in Markdown thanks to MDX:

```
import Tabs from '@theme/Tabs';
import TabItem from '@theme/TabItem';

<Tabs>
  <TabItem value="apple" label="Apple" default>
    This is an apple
  </TabItem>
  <TabItem value="orange" label="Orange">
    This is an orange
  </TabItem>
  <TabItem value="banana" label="Banana">
    This is a banana
  </TabItem>
</Tabs>
```

http://localhost:3000

Apple
Orange
Banana

This is an apple

It is also possible to provide `values` and `defaultValue` props to `Tabs`:

```
<Tabs
  defaultValue="apple"
  values={[
    {label: 'Apple', value: 'apple'},
    {label: 'Orange', value: 'orange'},
    {label: 'Banana', value: 'banana'},
  ]}>
  <TabItem value="apple">This is an apple    </TabItem>
  <TabItem value="orange">This is an orange    </TabItem>
  <TabItem value="banana">This is a banana    </TabItem>
</Tabs>
```

http://localhost:3000

Apple
Orange
Banana

This is an apple

- `Tabs` props take precedence over the `TabItem` props:

TIP

By default, all tabs are rendered eagerly during the build process, and search engines can index hidden tabs.

It is possible to only render the default tab with `<Tabs lazy />`.

Displaying a default tab

The first tab is displayed by default, and to override this behavior, you can specify a default tab by adding `default` to one of the tab items. You can also set the `defaultValue` prop of the `Tabs` component to the label value of your choice. For example, in the example above, either setting `default` for the `value="apple"` tab or setting `defaultValue="apple"` for the tabs forces the "Apple" tab to be open by default.

Docusaurus will throw an error if a `defaultValue` is provided for the `Tabs` but it refers to a non-existing value. If you want none of the tabs to be shown by default, use `defaultValue={null}`.

Syncing tab choices

You may want choices of the same kind of tabs to sync with each other. For example, you might want to provide different instructions for users on Windows vs users on macOS, and you want to change all OS-specific instructions tabs in one click. To achieve that, you can give all related tabs the same `groupId` prop. Note that doing this will persist the choice in `localStorage` and all `<Tab>` instances with the same `groupId` will update automatically when the value of one of them is changed. Note that group IDs are globally namespaced.

```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Use Ctrl + C to copy.</TabItem>
  <TabItem value="mac" label="macOS">Use Command + C to copy.</TabItem>
</Tabs>

<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Use Ctrl + V to paste.</TabItem>
  <TabItem value="mac" label="macOS">Use Command + V to paste.</TabItem>
</Tabs>
```

http://localhost:3000

Windows
macOS

Use Ctrl + C to copy.

Windows
macOS

Use Ctrl + V to paste.

For all tab groups that have the same `groupId`, the possible values do not need to be the same. If one tab group is chosen a value that does not exist in another tab group with the same `groupId`, the tab group with the missing value won't change its tab. You can see that from the following example. Try to select Linux, and the above tab groups don't change.

```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">
    I am Windows.
  </TabItem>
  <TabItem value="mac" label="macOS">
    I am macOS.
  </TabItem>
  <TabItem value="linux" label="Linux">
    I am Linux.
  </TabItem>
</Tabs>
```

http://localhost:3000

Windows
macOS
Linux

I am Windows.

Tab choices with different group IDs will not interfere with each other:

```
<Tabs groupId="operating-systems">
  <TabItem value="win" label="Windows">Windows in windows.</TabItem>
  <TabItem value="mac" label="macOS">macOS is macOS.</TabItem>
</Tabs>

<Tabs groupId="non-mac-operating-systems">
  <TabItem value="win" label="Windows">Windows is windows.</TabItem>
  <TabItem value="unix" label="Unix">Unix is unix.</TabItem>
</Tabs>
```

● ● ●

http://localhost:3000

Windows
macOS

Windows in windows.

Windows
Unix

Windows is windows.

Customizing tabs

You might want to customize the appearance of a certain set of tabs. You can pass the string in `className` prop, and the specified CSS class will be added to the `Tabs` component:

```
<Tabs className="unique-tabs">
  <TabItem value="Apple">This is an apple   </TabItem>
  <TabItem value="Orange">This is an orange   </TabItem>
  <TabItem value="Banana">This is a banana   </TabItem>
</Tabs>
```

● ● ●

http://localhost:3000

Orange
Banana

This is an apple

Customizing tab headings

You can also customize each tab heading independently by using the `attributes` field. The extra props can be passed to the headings either through the `values` prop in `Tabs`, or props of each `TabItem`—in the same way as you declare `label`.

some-doc.mdx

```jsx
import styles from './styles.module.css';

<Tabs>
  <TabItem value="apple" label="Apple" attributes={{className: styles.red}}>
    This is an apple
  </TabItem>
  <TabItem value="orange" label="Orange" attributes={{className: styles.orange}}>
    This is an orange
  </TabItem>
  <TabItem value="banana" label="Banana" attributes={{className: styles.yellow}}>
    This is a banana
  </TabItem>
</Tabs>
```

styles.module.css

```css
.red {
  color: red;
}
.red[aria-selected='true'] {
  border-bottom-color: red;
}

.orange {
  color: orange;
}
.orange[aria-selected='true'] {
  border-bottom-color: orange;
}

.yellow {
  color: yellow;
}
.yellow[aria-selected='true'] {
  border-bottom-color: yellow;
}
```

● ● ●

http://localhost:3000

Apple

Orange

Banana

This is an apple

TIP

`className` would be merged with other default class names. You may also use a custom `data-value` field (`{'data-value': 'apple'}`) paired with CSS attribute selectors:

styles.module.css

```css
li[role='tab'][data-value='apple'] {
  color: red;
}
```

Query string

It is possible to persist the selected tab into the url search parameters. This enables deep linking: the ability to share or bookmark a link to a specific tab, that will be pre-selected when the page loads.

Use the `queryString` prop to enable this feature and define the search param name to use.

```jsx
<Tabs queryString="current-os">
  <TabItem value="android" label="Android">
    Android
  </TabItem>
  <TabItem value="ios" label="iOS">
    iOS
  </TabItem>
</Tabs>
```

● ● ●

http://localhost:3000

Android
iOS

Android

As soon as a tab is clicked, a search parameter is added at the end of the url: `?current-os=android` or `?current-os=ios`.

TIP

`queryString` can be used together with `groupId`.

For convenience, when the `queryString` prop is `true`, the `groupId` value will be used as a fallback.

```
<Tabs groupId="current-os" queryString>
  <TabItem value="android" label="Android">
    Android
  </TabItem>
  <TabItem value="ios" label="iOS">
    iOS
  </TabItem>
</Tabs>
```

http://localhost:3000

Android
iOS

Android

When the page loads, the tab query string choice will be restored in priority over the `groupId` choice (using `localStorage`).

[Edit this page]

*Last updated on **Mar 23, 2023** by **Sébastien Lorber***

Previous
« MDX and React

Next
Code blocks »

Learn

Introduction

Installation

Migration from v1 to v2

Community

Stack Overflow

Feature Requests

Discord

Help

More

Blog