Routing libraries

By default, the navigation is performed with a native `<a>` element. You can customize it to use your own router. For instance, using Next.js's Link or react-router.

Navigation components

There are two main components available to perform navigations. The most common one is the `Link` as its name might suggest. It renders a native `<a>` element and applies the `href` as an attribute.

{{"demo": "LinkDemo.js"}}

You can also make a button perform navigation actions. If your component is extending `ButtonBase`, providing a `href` prop enables the link mode. For instance, with a `Button` component:

{{"demo": "ButtonDemo.js"}}

Global theme Link

In real-life applications, using a native `<a>` element is rarely enough. You can improve the user experience by using an enhanced Link component systematically. The theme of MUI allows configuring this component once. For instance, with react-router:

```
const LinkBehavior = React.forwardRef<
  any,
  Omit<RouterLinkProps, 'to'> & { href: RouterLinkProps['to'] }
>((props, ref) => {
  const { href, ...other } = props;
  // Map href (MUI) -> to (react-router)
  return <RouterLink ref={ref} to={href} {...other} />;
});

const theme = createTheme({
  components: {
    MuiLink: {
      defaultProps: {
        component: LinkBehavior,
      },
    },
    MuiButtonBase: {
      defaultProps: {
        LinkComponent: LinkBehavior,
      },
    },
  },
});
```

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

> ⚠️ *This approach has limitations with TypeScript. The `href` prop only accepts a string. In the event you need to provide a richer structure, see the next section.*

`component` prop

You can achieve the integration with third-party routing libraries with the `component` prop. You can learn more about this prop in the **composition guide**.

Link

Here are a few demos with react-router. You can apply the same strategy with all the components: BottomNavigation, Card, etc.

{{"demo": "LinkRouter.js"}}

Button

{{"demo": "ButtonRouter.js"}}

Note: The button base component adds the `role="button"` attribute when it identifies the intent to render a button without a native `<button>` element. This can create issues when rendering a link. If you are not using one of the `href`, `to`, or `component="a"` props, you need to override the `role` attribute. The above demo achieves this by setting `role={undefined}` **after** the spread props.

```
const LinkBehavior = React.forwardRef((props, ref) => (
  <RouterLink ref={ref} to="/" {...props} role={undefined} />
));
```

Tabs

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

List

{{"demo": "ListRouter.js"}}

More examples

Next.js

Next.js has a custom Link component. The example folder provides adapters for usage with MUI.

- The first version of the adapter is the `NextLinkComposed` component. This component is unstyled and only responsible for handling the navigation. The prop `href` was renamed `to` to avoid a naming conflict. This is similar to react-router's Link component.

  ```
  import Button from '@mui/material/Button';
  import { NextLinkComposed } from '../src/Link';

  export default function Index() {
    return (
      <Button
        component={NextLinkComposed}
        to={{
          pathname: '/about',
  ```

```
        query: { name: 'test' },
      }}
    >
      Button link
    </Button>
  );
}
```

- The second version of the adapter is the `Link` component. This component is styled. It leverages the link component of MUI with `NextLinkComposed`.

```
import Link from '../src/Link';

export default function Index() {
  return (
    <Link
      href={{
        pathname: '/about',
        query: { name: 'test' },
      }}
    >
      Link
    </Link>
  );
}
```

Gatsby

The Link component of Gatsby is built on `@reach/router`. You can use the same previous documentation for react-router. Unlike Next.js, it doesn't require you to create an adapter.