Using MDX with Next.js

MDX is a superset of markdown that lets you write JSX directly in your markdown files. It is a powerful way to add dynamic interactivity, and embed components within your content, helping you to bring your pages to life.

Next.js supports MDX through a number of different means, this page will outline some of the ways you can begin integrating MDX into your Next.js project.

Why use MDX?

Authoring in markdown is an intuitive way to write content, its terse syntax, once adopted, can enable you to write content that is both readable and maintainable. Because you can use `HTML` elements in your markdown, you can also get creative when styling your markdown pages.

However, because markdown is essentially static content, you can't create *dynamic* content based on user interactivity. Where MDX shines is in its ability to let you create and use your React components directly in the markup. This opens up a wide range of possibilities when composing your sites pages with interactivity in mind.

MDX Plugins

Internally MDX uses remark and rehype. Remark is a markdown processor powered by a plugins ecosystem. This plugin ecosystem lets you parse code, transform `HTML` elements, change syntax, extract frontmatter, and more.

Rehype is an `HTML` processor, also powered by a plugin ecosystem. Similar to remark, these plugins let you manipulate, sanitize, compile and configure all types of data, elements and content.

To use a plugin from either remark or rehype, you will need to add it to the MDX packages config.

`@next/mdx`

The `@next/mdx` package is configured in the `next.config.js` file at your projects root. **It sources data from local files**, allowing you to create pages with a `.mdx` extension, directly in your `/pages` directory.

Setup `@next/mdx` in Next.js

The following steps outline how to setup `@next/mdx` in your Next.js project:

1. Install the required packages:

   ```
   npm install @next/mdx @mdx-js/loader
   ```

2. Require the package and configure to support top level `.mdx` pages. The following adds the `options` object key allowing you to pass in any plugins:

   ```
   // next.config.js

   const withMDX = require('@next/mdx')({
     extension: /\.mdx?$/,
     options: {
       remarkPlugins: [],
       rehypePlugins: [],
   ```

```
    // If you use `MDXProvider`, uncomment the following line.
    // providerImportSource: "@mdx-js/react",
  },
})
module.exports = withMDX({
  // Append the default value with md extensions
  pageExtensions: ['ts', 'tsx', 'js', 'jsx', 'md', 'mdx'],
})
```

3. Create a new MDX page within the `/pages` directory:

```
  - /pages
    - my-mdx-page.mdx
  - package.json
```

Using Components, Layouts and Custom Elements

You can now import a React component directly inside your MDX page:

```
import { MyComponent } from 'my-components'

# My MDX page

This is a list in markdown:

- One
- Two
- Three

Checkout my React component:

<MyComponent/>
```

Frontmatter

Frontmatter is a YAML like key/value pairing that can be used to store data about a page. `@next/mdx` does **not** support frontmatter by default, though there are many solutions for adding frontmatter to your MDX content, such as gray-matter.

To access page metadata with `@next/mdx`, you can export a meta object from within the `.mdx` file:

```
export const meta = {
author: 'Rich Haines'
}

# My MDX page
```

Layouts

To add a layout to your MDX page, create a new component and import it into the MDX page. Then you can wrap the MDX page with your layout component:

```
import { MyComponent, MyLayoutComponent } from 'my-components'

export const meta = {
author: 'Rich Haines'
}

# My MDX Page with a Layout

This is a list in markdown:

- One
- Two
- Three

Checkout my React component:

<MyComponent/>

export default ({ children }) => <MyLayoutComponent meta={meta}>{children}
</MyLayoutComponent>
```

Custom Elements

One of the pleasant aspects of using markdown, is that it maps to native `HTML` elements, making writing fast, and intuitive:

```
# H1 heading

## H2 heading

This is a list in markdown:

- One
- Two
- Three
```

The above generates the following `HTML` :

```
<h1>H1 heading</h1>

<h2>H2 heading</h2>

<p>This is a list in markdown:</p>

<ul>
  <li>One</li>
  <li>Two</li>
  <li>Three</li>
</ul>
```

When you want to style your own elements to give a custom feel to your website or application, you can pass in shortcodes. These are your own custom components that map to `HTML` elements. To do this you use the `MDXProvider` and pass a components object as a prop. Each object key in the components object maps to a `HTML` element name.

To enable you need to specify `providerImportSource: "@mdx-js/react"` in `next.config.js`.

```
// next.config.js

const withMDX = require('@next/mdx')({
  // ...
  options: {
    providerImportSource: '@mdx-js/react',
  },
})
```

Then setup the provider in your page

```
// pages/index.js

import { MDXProvider } from '@mdx-js/react'
import Image from 'next/image'
import { Heading, InlineCode, Pre, Table, Text } from 'my-components'

const ResponsiveImage = (props) => (
  <Image alt={props.alt} layout="responsive" {...props} />
)

const components = {
  img: ResponsiveImage,
  h1: Heading.H1,
  h2: Heading.H2,
  p: Text,
  pre: Pre,
  code: InlineCode,
}

export default function Post(props) {
  return (
    <MDXProvider components={components}>
      <main {...props} />
    </MDXProvider>
  )
}
```

If you use it across the site you may want to add the provider to `_app.js` so all MDX pages pick up the custom element config.

Helpful Links

- MDX
- `@next/mdx`
- remark
- rehype