MDX

Markdown is a lightweight markup language used to format text. It allows you to write using plain text syntax and convert it to structurally valid HTML. It's commonly used for writing content on websites and blogs.

You write...

```
I **love** using [Next.js](https://nextjs.org/)
```

Output:

```
<p>I <strong>love</strong> using <a href="https://nextjs.org/">Next.js</a></p>
```

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 React components within your content.

Next.js can support both local MDX content inside your application, as well as remote MDX files fetched dynamically on the server. The Next.js plugin handles tranforming Markdown and React components into HTML, including support for usage in Server Components (default in `app`).

`@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` or `/app` directory.

Getting Started

Install the required packages:

Terminal

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

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:

```js
// next.config.js

const withMDX = require('@next/mdx')({
  extension: /\.mdx?$/,
  options: {
    // If you use remark-gfm, you'll need to use next.config.mjs
    // as the package is ESM only
    // https://github.com/remarkjs/remark-gfm#install
    remarkPlugins: [],
    rehypePlugins: [],
    // If you use `MDXProvider`, uncomment the following line.
    // providerImportSource: "@mdx-js/react",
  },
});

/** @type {import('next').NextConfig} */
const nextConfig = {
  // Configure pageExtensions to include md and mdx
  pageExtensions: ['ts', 'tsx', 'js', 'jsx', 'md', 'mdx'],
  // Optionally, add any other Next.js config below
  reactStrictMode: true,
};

// Merge MDX config with Next.js config
module.exports = withMDX(nextConfig);
```

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

```
  your-project
  ├── pages
  |    └── my-mdx-page.mdx
  └── package.json
```

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 />
```

Remote MDX

If your Markdown or MDX files do *not* live inside your application, you can fetch them dynamically on the server. This is useful for fetching content from a CMS or other data source.

There are two popular community packages for fetching MDX content: `next-mdx-remote` ↗ and `contentlayer` ↗. For example, the following example uses `next-mdx-remote`:

Note: Please proceed with caution. MDX compiles to JavaScript and is executed on the server. You should only fetch MDX content from a trusted source, otherwise this can lead to remote code execution (RCE).

app/page.tsx

```tsx
import { MDXRemote } from 'next-mdx-remote/rsc';

export default async function Home() {
  const res = await fetch('https://...');
  const markdown = await res.text();
  return <MDXRemote source={markdown} />;
}
```

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:

pages/index.jsx

```jsx
import { MyLayoutComponent } from 'my-components';
import HelloWorld from './hello.mdx';

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

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

Remark and Rehype Plugins

You can optionally provide `remark` and `rehype` plugins to transform the MDX content. For example, you can use `remark-gfm` to support GitHub Flavored Markdown.

Since the `remark` and `rehype` ecosystem is ESM only, you'll need to use `next.config.mjs` as the configuration file.

next.config.js

```js
/** @type {import('next').NextConfig} */
const nextConfig = {
  experimental: {
    appDir: true,
  },
};

const withMDX = require('@next/mdx')({
  options: {
    remarkPlugins: [],
    rehypePlugins: [],
    // If you use `MDXProvider`, uncomment the following line.
    // providerImportSource: "@mdx-js/react",
  },
});
module.exports = withMDX(nextConfig);
```

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
```

Custom Elements

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

```
This is a list in markdown:

- One
- Two
- Three
```

The above generates the following `HTML`:

```
<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

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

Then setup the provider in your page

pages/index.js

```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} sizes="100vw" style={ width: '100%', height: 'auto' }  {...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.

Deep Dive: How do you transform markdown into HTML?

React does not natively understand Markdown. The markdown plaintext needs to first be transformed into HTML. This can be accomplished with `remark` and `rehype`.

`remark` is an ecosystem of tools around markdown. `rehype` is the same, but for HTML. For example, the following code snippet transforms markdown into HTML:

```
import { unified } from 'unified';
import remarkParse from 'remark-parse';
import remarkRehype from 'remark-rehype';
import rehypeSanitize from 'rehype-sanitize';
import rehypeStringify from 'rehype-stringify';

main();

async function main() {
  const file = await unified()
    .use(remarkParse) // Convert into markdown AST
    .use(remarkRehype) // Transform to HTML AST
    .use(rehypeSanitize) // Sanitize HTML input
    .use(rehypeStringify) // Convert AST into serialized HTML
    .process('Hello, Next.js!');

  console.log(String(file)); // <p>>Hello, Next.js!</p>
}
```

The `remark` and `rehype` ecosystem contains plugins for syntax highlighting↗, linking headings↗, generating a table of contents↗, and more.

When using `@next/mdx` as shown below, you **do not** need to use `remark` or `rehype` directly, as it is handled for you.

Using the Rust-based MDX compiler (Experimental)

Next.js supports a new MDX compiler written in Rust. This compiler is still experimental and is not recommended for production use. To use the new compiler, you need to configure `next.config.js` when you pass it to `withMDX`:

next.config.js

```
module.exports = withMDX({
  experimental: {
    mdxRs: true,
  },
});
```

Helpful Links

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