TypeScript

▶ **Examples**

Next.js provides an integrated TypeScript experience out of the box, similar to an IDE.

To get started, create an empty `tsconfig.json` file in the root of your project:

```
touch tsconfig.json
```

Next.js will automatically configure this file with default values. Providing your own `tsconfig.json` with custom compiler options is also supported.

> Next.js uses Babel to handle TypeScript, which has some caveats, and some compiler options are handled differently.

Then, run `next` (normally `npm run dev` or `yarn dev`) and Next.js will guide you through the installation of the required packages to finish the setup:

```
npm run dev

# You'll see instructions like these:
#
# Please install typescript, @types/react, and @types/node by running:
#
```

```
#         yarn add --dev typescript @types/react @types/node
#
# ...
```

You're now ready to start converting files from `.js` to `.tsx` and leveraging the benefits of TypeScript!.

> A file named `next-env.d.ts` will be created in the root of your project. This file ensures Next.js types are picked up by the TypeScript compiler. **You cannot remove it**, however, you can edit it (but you don't need to).

> TypeScript `strict` mode is turned off by default. When you feel comfortable with TypeScript, it's recommended to turn it on in your `tsconfig.json`.

By default, Next.js will do type checking as part of `next build`. We recommend using code editor type checking during development.

If you want to silence the error reports, refer to the documentation for Ignoring TypeScript errors.

Static Generation and Server-side Rendering

For `getStaticProps`, `getStaticPaths`, and `getServerSideProps`, you can use the `GetStaticProps`, `GetStaticPaths`, and `GetServerSideProps` types respectively:

```
import { GetStaticProps, GetStaticPaths, GetServerSideProps } from 'next'

export const getStaticProps: GetStaticProps = async (context) => {
  // ...
}

export const getStaticPaths: GetStaticPaths = async () => {
  // ...
}

export const getServerSideProps: GetServerSideProps = async (context) => {
  // ...
}
```

If you're using `getInitialProps`, you can follow the directions on this page.

API Routes

The following is an example of how to use the built-in types for API routes:

```
import type { NextApiRequest, NextApiResponse } from 'next'

export default (req: NextApiRequest, res: NextApiResponse) => {
  res.status(200).json({ name: 'John Doe' })
}
```

You can also type the response data:

```
import type { NextApiRequest, NextApiResponse } from 'next'

type Data = {
  name: string
}

export default (req: NextApiRequest, res: NextApiResponse<Data>) => {
  res.status(200).json({ name: 'John Doe' })
}
```

Custom `App`

If you have a custom `App`, you can use the built-in type `AppProps` and change file name to `./pages/_app.tsx` like so:

```
// import App from "next/app";
import type { AppProps /*, AppContext */ } from 'next/app'

function MyApp({ Component, pageProps }: AppProps) {
  return <Component {...pageProps} />
}

// Only uncomment this method if you have blocking data requirements for
// every single page in your application. This disables the ability to
// perform automatic static optimization, causing every page in your app to
// be server-side rendered.
//
// MyApp.getInitialProps = async (appContext: AppContext) => {
//   // calls page's `getInitialProps` and fills `appProps.pageProps`
//   const appProps = await App.getInitialProps(appContext);

//   return { ...appProps }
// }

export default MyApp
```

Path aliases and baseUrl

Next.js automatically supports the `` `tsconfig.json` `` `` `"paths"` `` and `` `"baseUrl"` `` options.

You can learn more about this feature on the Module Path aliases documentation.

‹ Fast Refresh

Environment Variables ›

Was this helpful?

😭 😕 😀 🤩

Edit this page on GitHub