TypeScript

Next.js provides a TypeScript-first development experience for building your React application.

It comes with built-in TypeScript support for automatically installing the necessary packages and configuring the proper settings.

New Projects

`create-next-app` now ships with TypeScript by default.

Terminal

```
npx create-next-app@latest
```

Existing Projects

Add TypeScript to your project by renaming a file to `.ts` / `.tsx`. Run `next dev` and `next build` to automatically install the necessary dependencies and add a `tsconfig.json` file with the recommended config options.

Minimum TypeScript Version

It is highly recommended to be on at least `v4.5.2` of TypeScript to get syntax features such as type modifiers on import names ↗ and performance improvements ↗.

Static Generation and Server-side Rendering

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

pages/blog/[slug].tsx

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

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

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

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

API Routes

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

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

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

You can also type the response data:

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

type Data = {
  name: string;
};

export default function handler(
  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 type { AppProps } from 'next/app';

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

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.

Type checking next.config.js

The `next.config.js` file must be a JavaScript file as it does not get parsed by Babel or TypeScript, however you can add some type checking in your IDE using JSDoc as below:

```
// @ts-check

/**
 * @type {import('next').NextConfig}
 **/
const nextConfig = {
  /* config options here */
};

module.exports = nextConfig;
```

Incremental type checking

Since `v10.2.1` Next.js supports incremental type checking ↗ when enabled in your `tsconfig.json`, this can help speed up type checking in larger applications.

Ignoring TypeScript Errors

Next.js fails your **production build** (`next build`) when TypeScript errors are present in your project.

If you'd like Next.js to dangerously produce production code even when your application has errors, you can disable the built-in type checking step.

If disabled, be sure you are running type checks as part of your build or deploy process, otherwise this can be very dangerous.

Open `next.config.js` and enable the `ignoreBuildErrors` option in the `typescript` config:

next.config.js

```
module.exports = {
  typescript: {
    // !! WARN !!
    // Dangerously allow production builds to successfully complete even if
    // your project has type errors.
    // !! WARN !!
    ignoreBuildErrors: true,
  },
};
```

Version Changes

Version	Changes
`v13.2.0`	Statically typed links are available in beta
`v12.0.0`	SWC is now used by default to compile TypeScript and TSX for faster builds.
`v10.2.1`	Incremental type checking ↗ support added when enabled in your `tsconfig.json`.