# TypeScript

▶ **Version History**

Next.js provides an integrated TypeScript experience, including zero-configuration set up and built-in types for Pages, APIs, and more.

- Clone and deploy the TypeScript starter
- View an example application

## `create-next-app` support

You can create a TypeScript project with `create-next-app` using the `--ts, --typescript` flag like so:

```
npx create-next-app@latest --ts
# or
yarn create next-app --typescript
# or
pnpm create next-app -- --ts
```

## Existing projects

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

```
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.

You can also provide a relative path to a tsconfig.json file by setting `typescript.tsconfigPath` prop inside your `next.config.js` file.

Starting in `v12.0.0`, Next.js uses SWC by default to compile TypeScript and TSX for faster builds.

> *Next.js will use Babel to handle TypeScript if `.babelrc` is present. This 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 or edit it** as it can change at any time.*

> *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`.*

> *Instead of editing `next-env.d.ts`, you can include additional types by adding a new file e.g. `additional.d.ts` and then referencing it in the `include` array 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 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.

It is highly recommended to be on at least `v4.3.2` of TypeScript to experience the best performance when leveraging this feature.

## 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:

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