getStaticPaths

If a page has Dynamic Routes and uses `getStaticProps`, it needs to define a list of paths to be statically generated.

When you export a function called `getStaticPaths` (Static Site Generation) from a page that uses dynamic routes, Next.js will statically pre-render all the paths specified by `getStaticPaths`.

pages/repo/[name].tsx

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

type Repo = {
  name: string;
  stargazers_count: number;
};

export const getStaticPaths: GetStaticPaths = async () => {
  return {
    paths: [
      {
        params: {
          name: 'next.js',
        },
      }, // See the "paths" section below
    ],
    fallback: true, // false or "blocking"
  };
};

export const getStaticProps: GetStaticProps<{
  repo: Repo;
}> = async () => {
  const res = await fetch('https://api.github.com/repos/vercel/next.js');
  const repo = await res.json();
  return { props: { repo } };
};
```

```
32
33 export default function Page({
34   repo,
35 }: InferGetStaticPropsType<typeof getStaticProps>) {
36   return repo.stargazers_count;
37 }
```

The `getStaticPaths` API reference covers all parameters and props that can be used with `getStaticPaths`.

When should I use getStaticPaths?

You should use `getStaticPaths` if you're statically pre-rendering pages that use dynamic routes and:

- The data comes from a headless CMS
- The data comes from a database
- The data comes from the filesystem
- The data can be publicly cached (not user-specific)
- The page must be pre-rendered (for SEO) and be very fast — `getStaticProps` generates `HTML` and `JSON` files, both of which can be cached by a CDN for performance

---

When does getStaticPaths run

`getStaticPaths` will only run during build in production, it will not be called during runtime. You can validate code written inside `getStaticPaths` is removed from the client-side bundle with this tool↗.

How does getStaticProps run with regards to getStaticPaths

- `getStaticProps` runs during `next build` for any `paths` returned during build
- `getStaticProps` runs in the background when using `fallback: true`
- `getStaticProps` is called before initial render when using `fallback: blocking`

Where can I use getStaticPaths

- `getStaticPaths` **must** be used with `getStaticProps`
- You **cannot** use `getStaticPaths` with `getServerSideProps`
- You can export `getStaticPaths` from a Dynamic Route that also uses `getStaticProps`
- You **cannot** export `getStaticPaths` from non-page file (e.g. your `components` folder)
- You must export `getStaticPaths` as a standalone function, and not a property of the page component

---

Runs on every request in development

In development (`next dev`), `getStaticPaths` will be called on every request.

Generating paths on-demand

`getStaticPaths` allows you to control which pages are generated during the build instead of on-demand with `fallback`. Generating more pages during a build will cause slower builds.

You can defer generating all pages on-demand by returning an empty array for `paths`. This can be especially helpful when deploying your Next.js application to multiple environments. For example, you can have faster builds by generating all pages on-demand for previews (but not production builds). This is helpful for sites with hundreds/thousands of static pages.

pages/posts/[id].js

```
export async function getStaticPaths() {
  // When this is true (in preview environments) don't
  // prerender any static pages
  // (faster builds, but slower initial page load)
  if (process.env.SKIP_BUILD_STATIC_GENERATION) {
    return {
      paths: [],
      fallback: 'blocking',
    };
  }

  // Call an external API endpoint to get posts
  const res = await fetch('https://.../posts');
  const posts = await res.json();

  // Get the paths we want to prerender based on posts
  // In production environments, prerender all pages
```

```
  // (slower builds, but faster initial page load)
  const paths = posts.map((post) => ({
    params: { id: post.id },
  }));

  // { fallback: false } means other routes should 404
  return { paths, fallback: false };
}
```