getStaticProps

If you export a function called `getStaticProps` (Static Site Generation) from a page, Next.js will pre-render this page at build time using the props returned by `getStaticProps`.

pages/index.tsx

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

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

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 } };
};

export default function Page({
  repo,
}: InferGetStaticPropsType<typeof getStaticProps>) {
  return repo.stargazers_count;
}
```

Note that irrespective of rendering type, any `props` will be passed to the page component and can be viewed on the client-side in the initial HTML. This is to allow the page to be hydrated ↗ correctly. Make sure that you don't pass any sensitive information that shouldn't be available on the client in `props`.

When should I use getStaticProps?

You should use `getStaticProps` if:

- The data required to render the page is available at build time ahead of a user's request
- The data comes from a headless CMS
- 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
- The data can be publicly cached (not user-specific). This condition can be bypassed in certain specific situation by using a Middleware to rewrite the path.

When does getStaticProps run

`getStaticProps` always runs on the server and never on the client. You can validate code written inside `getStaticProps` is removed from the client-side bundle with this tool↗.

- `getStaticProps` always runs during `next build`
- `getStaticProps` runs in the background when using `fallback: true`
- `getStaticProps` is called before initial render when using `fallback: blocking`
- `getStaticProps` runs in the background when using `revalidate`
- `getStaticProps` runs on-demand in the background when using `revalidate()`

When combined with Incremental Static Regeneration, `getStaticProps` will run in the background while the stale page is being revalidated, and the fresh page served to the browser.

`getStaticProps` does not have access to the incoming request (such as query parameters or HTTP headers) as it generates static HTML. If you need access to the request for your page, consider using Middleware in addition to `getStaticProps`.

Using getStaticProps to fetch data from a CMS

The following example shows how you can fetch a list of blog posts from a CMS.

pages/blog.tsx

```ts
// posts will be populated at build time by getStaticProps()
export default function Blog({ posts }) {
```

```
  return (
    <ul>
      {posts.map((post) => (
        <li>{post.title}</li>
      ))}
    </ul>
  );
}

// This function gets called at build time on server-side.
// It won't be called on client-side, so you can even do
// direct database queries.
export async function getStaticProps() {
  // Call an external API endpoint to get posts.
  // You can use any data fetching library
  const res = await fetch('https://.../posts');
  const posts = await res.json();

  // By returning { props: { posts } }, the Blog component
  // will receive `posts` as a prop at build time
  return {
    props: {
      posts,
    },
  };
}
```

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

Write server-side code directly

As `getStaticProps` runs only on the server-side, it will never run on the client-side. It won't even be included in the JS bundle for the browser, so you can write direct database queries without them being sent to browsers.

This means that instead of fetching an **API route** from `getStaticProps` (that itself fetches data from an external source), you can write the server-side code directly in `getStaticProps`.

Take the following example. An API route is used to fetch some data from a CMS. That API route is then called directly from `getStaticProps`. This produces an additional call, reducing performance. Instead, the logic for fetching the data from the CMS can be shared by using a `lib/` directory. Then it can be shared with `getStaticProps`.

lib/load-posts.js

```
// The following function is shared
// with getStaticProps and API routes
// from a `lib/` directory
export async function loadPosts() {
  // Call an external API endpoint to get posts
  const res = await fetch('https://.../posts/');
  const data = await res.json();

  return data;
}

// pages/blog.js
import { loadPosts } from '../lib/load-posts';

// This function runs only on the server side
export async function getStaticProps() {
  // Instead of fetching your `/api` route you can call the same
  // function directly in `getStaticProps`
  const posts = await loadPosts();

  // Props returned will be passed to the page component
  return { props: { posts } };
}
```

Alternatively, if you are **not** using API routes to fetch data, then the `fetch()` ↗ API *can* be used directly in `getStaticProps` to fetch data.

To verify what Next.js eliminates from the client-side bundle, you can use the next-code-elimination tool ↗.

Statically generates both HTML and JSON

When a page with `getStaticProps` is pre-rendered at build time, in addition to the page HTML file, Next.js generates a JSON file holding the result of running `getStaticProps`.

This JSON file will be used in client-side routing through `next/link` or `next/router`. When you navigate to a page that's pre-rendered using `getStaticProps`, Next.js fetches this JSON file (pre-computed at build time) and uses it as the props for the page component. This means that client-side page transitions will **not** call `getStaticProps` as only the exported JSON is used.

When using Incremental Static Generation, `getStaticProps` will be executed in the background to generate the JSON needed for client-side navigation. You may see this in the form of multiple requests being made for the same page, however, this is intended and has no impact on end-user performance.

Where can I use getStaticProps

`getStaticProps` can only be exported from a **page**. You **cannot** export it from non-page files, `_app`, `_document`, or `_error`.

One of the reasons for this restriction is that React needs to have all the required data before the page is rendered.

Also, you must use export `getStaticProps` as a standalone function — it will **not** work if you add `getStaticProps` as a property of the page component.

Note: if you have created a custom app, ensure you are passing the `pageProps` to the page component as shown in the linked document, otherwise the props will be empty.

Runs on every request in development

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

Preview Mode

You can temporarily bypass static generation and render the page at **request time** instead of build time using **Preview Mode**. For example, you might be using a headless CMS and want to preview drafts before they're published.