getServerSideProps

If you export a function called `getServerSideProps` (Server-Side Rendering) from a page, Next.js will pre-render this page on each request using the data returned by `getServerSideProps`.

pages/index.tsx

```tsx
import type { InferGetServerSidePropsType, GetServerSideProps } from 'next';

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

export const getServerSideProps: GetServerSideProps<{
  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,
}: InferGetServerSidePropsType<typeof getServerSideProps>) {
  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 does getServerSideProps run

`getServerSideProps` only runs on server-side and never runs on the browser. If a page uses `getServerSideProps`, then:

- When you request this page directly, `getServerSideProps` runs at request time, and this page will be pre-rendered with the returned props
- When you request this page on client-side page transitions through `next/link` or `next/router`, Next.js sends an API request to the server, which runs `getServerSideProps`

`getServerSideProps` returns JSON which will be used to render the page. All this work will be handled automatically by Next.js, so you don't need to do anything extra as long as you have `getServerSideProps` defined.

You can use the next-code-elimination tool ↗ to verify what Next.js eliminates from the client-side bundle.

`getServerSideProps` can only be exported from a **page**. You can't export it from non-page files.

Note that you must export `getServerSideProps` as a standalone function — it will **not** work if you add `getServerSideProps` as a property of the page component.

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

When should I use getServerSideProps

You should use `getServerSideProps` only if you need to render a page whose data must be fetched at request time. This could be due to the nature of the data or properties of the request (such as `authorization` headers or geo location). Pages using `getServerSideProps` will be server side rendered at request time and only be cached if cache-control headers are configured.

If you do not need to render the data during the request, then you should consider fetching data on the client side or `getStaticProps`.

getServerSideProps or API Routes

It can be tempting to reach for an API Route when you want to fetch data from the server, then call that API route from `getServerSideProps`. This is an unnecessary and inefficient approach, as it will cause an extra request to be made due to both `getServerSideProps` and API Routes running on the server.

Take the following example. An API route is used to fetch some data from a CMS. That API route is then called directly from `getServerSideProps`. This produces an additional call, reducing performance. Instead,

directly import the logic used inside your API Route into `getServerSideProps`. This could mean calling a CMS, database, or other API directly from inside `getServerSideProps`.

getServerSideProps with Edge API Routes

`getServerSideProps` can be used with both Serverless and Edge Runtimes, and you can set props in both. However, currently in the Edge Runtime, you do not have access to the response object. This means that you cannot — for example — add cookies in `getServerSideProps`. To have access to the response object, you should **continue to use the Node.js runtime**, which is the default runtime.

You can explicitly set the runtime on a per-page basis by modifying the `config`, for example:

pages/index.tsx

```tsx
export const config = {
  runtime: 'nodejs', // or "edge"
};

export const getServerSideProps = async () => {};
```

Fetching data on the client side

If your page contains frequently updating data, and you don't need to pre-render the data, you can fetch the data on the client side. An example of this is user-specific data:

- First, immediately show the page without data. Parts of the page can be pre-rendered using Static Generation. You can show loading states for missing data
- Then, fetch the data on the client side and display it when ready

This approach works well for user dashboard pages, for example. Because a dashboard is a private, user-specific page, SEO is not relevant and the page doesn't need to be pre-rendered. The data is frequently updated, which requires request-time data fetching.

Using getServerSideProps to fetch data at request time

The following example shows how to fetch data at request time and pre-render the result.

```jsx
function Page({ data }) {
  // Render data...
}

// This gets called on every request
export async function getServerSideProps() {
  // Fetch data from external API
  const res = await fetch(`https://.../data`);
  const data = await res.json();

  // Pass data to the page via props
  return { props: { data } };
}

export default Page;
```

Caching with Server-Side Rendering (SSR)

You can use caching headers (`Cache-Control`) inside `getServerSideProps` to cache dynamic responses. For example, using `stale-while-revalidate` ↗.

```jsx
// This value is considered fresh for ten seconds (s-maxage=10).
// If a request is repeated within the next 10 seconds, the previously
// cached value will still be fresh. If the request is repeated before 59 seconds,
// the cached value will be stale but still render (stale-while-revalidate=59).
//
// In the background, a revalidation request will be made to populate the cache
// with a fresh value. If you refresh the page, you will see the new value.
export async function getServerSideProps({ req, res }) {
  res.setHeader(
    'Cache-Control',
    'public, s-maxage=10, stale-while-revalidate=59',
  );

  return {
    props: {},
  };
}
```

Learn more about caching.

Does getServerSideProps render an error page

If an error is thrown inside `getServerSideProps`, it will show the `pages/500.js` file. Check out the documentation for 500 page to learn more on how to create it. During development this file will not be used and the dev overlay will be shown instead.