Authenticating

Authentication verifies who a user is, while authorization controls what a user can access. Next.js supports multiple authentication patterns, each designed for different use cases. This page will go through each case so that you can choose based on your constraints.

Authentication Patterns

The first step to identifying which authentication pattern you need is understanding the data-fetching strategy you want. We can then determine which authentication providers support this strategy. There are two main patterns:

- Use static generation to server-render a loading state, followed by fetching user data client-side.
- Fetch user data server-side to eliminate a flash of unauthenticated content.

Authenticating Statically Generated Pages

Next.js automatically determines that a page is static if there are no blocking data requirements. This means the absence of `getServerSideProps` and `getInitialProps` in the page. Instead, your page can render a loading state from the server, followed by fetching the user client-side.

One advantage of this pattern is it allows pages to be served from a global CDN and preloaded using `next/link`. In practice, this results in a faster TTI (Time to Interactive ↗).

Let's look at an example for a profile page. This will initially render a loading skeleton. Once the request for a user has finished, it will show the user's name:

pages/profile.js

```
import useUser from '../lib/useUser';
import Layout from '../components/Layout';
```

```

const Profile = () => {
  // Fetch the user client-side
  const { user } = useUser({ redirectTo: '/login' });

  // Server-render loading state
  if (!user || user.isLoggedIn === false) {
    return <Layout>Loading...</Layout>;
  }

  // Once the user request finishes, show the user
  return (
    <Layout>
      <h1>Your Profile</h1>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </Layout>
  );
};

export default Profile;
```

You can view this example in action ↗. Check out the `with-iron-session` ↗ example to see how it works.

Authenticating Server-Rendered Pages

If you export an `async` function called `getServerSideProps` from a page, Next.js will pre-render this page on each request using the data returned by `getServerSideProps`.

```
export async function getServerSideProps(context) {
  return {
    props: {}, // Will be passed to the page component as props
  };
}
```

Let's transform the profile example to use server-side rendering. If there's a session, return `user` as a prop to the `Profile` component in the page. Notice there is not a loading skeleton in this example ↗.

pages/profile.js

```
import withSession from '../lib/session';
import Layout from '../components/Layout';

export const getServerSideProps = withSession(async function ({ req, res }) {
  const { user } = req.session;

  if (!user) {
    return {
      redirect: {
        destination: '/login',
        permanent: false,
      },
    };
  }

  return {
    props: { user },
  };
});

const Profile = ({ user }) => {
  // Show the user. No loading state is required
  return (
    <Layout>
      <h1>Your Profile</h1>
      <pre>{JSON.stringify(user, null, 2)}</pre>
    </Layout>
  );
};

export default Profile;
```

An advantage of this pattern is preventing a flash of unauthenticated content before redirecting. It's important to note fetching user data in `getServerSideProps` will block rendering until the request to your authentication provider resolves. To prevent creating a bottleneck and increasing your TTFB (Time to First Byte ↗), you should ensure your authentication lookup is fast. Otherwise, consider static generation.

Authentication Providers

Now that we've discussed authentication patterns, let's look at specific providers and explore how they're used with Next.js.

Bring Your Own Database

▼ **Examples**

- with-iron-session ↗
- next-auth-example ↗

If you have an existing database with user data, you'll likely want to utilize an open-source solution that's provider agnostic.

- If you want a low-level, encrypted, and stateless session utility use `iron-session` ↗.

- If you want a full-featured authentication system with built-in providers (Google, Facebook, GitHub…), JWT, JWE, email/password, magic links and more… use `next-auth`↗.

Both of these libraries support either authentication pattern. If you're interested in Passport↗, we also have examples for it using secure and encrypted cookies:

- with-passport↗
- with-passport-and-next-connect↗

Other Providers

To see examples with other authentication providers, check out the examples folder↗.

▼ **Examples**

- Auth0↗
- Clerk↗
- Firebase↗
- Magic↗
- Nhost↗
- Ory↗
- Supabase↗
- Supertokens↗
- Userbase↗