Use Neon Auth with Next.js (UI Components)

Set up authentication in Next.js using pre-built UI components

BETA

The **Neon Auth with Better Auth** is in Beta. Share your feedback on Discord or via the Neon Console.

NOTE

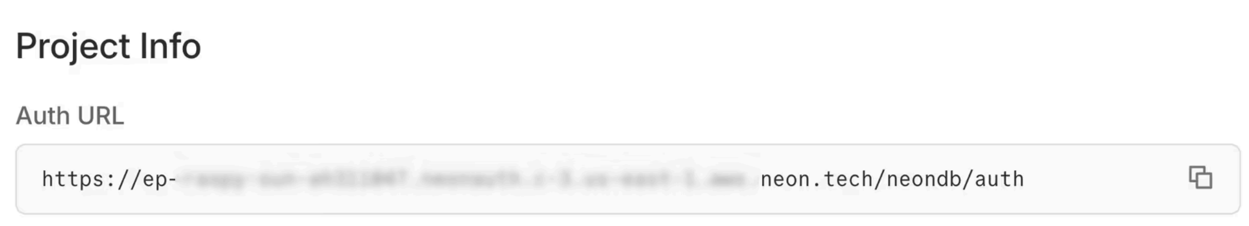
Upgrading from Neon Auth SDK v0.1? See the migration guide for step-by-step instructions.

1 Enable Auth in your Neon project

Enable Auth in your Neon project ↗ and copy your Auth URL from Configuration.

Console path: Project → Branch → Auth → Configuration

Console



Project Info

Auth URL

```
https://ep-[redacted]neon.tech/neondb/auth
```

2 Install the Neon Auth SDK

Install the Neon SDK into your Next.js app.

If you don't have a Next.js project

Terminal

```
npm install @neondatabase/auth
```

3 Set up environment variables

Create a `.env` file in your project root and add your Auth URL and a cookie secret:

> **NOTE**
>
> Replace the Auth URL with your actual Auth URL from the Neon Console. Generate a secure cookie secret with `openssl rand -base64 32` .

.env

```
NEON_AUTH_BASE_URL=https://ep-xxx.neonauth.us-east-1.aws.neon.tech/neondb/auth
NEON_AUTH_COOKIE_SECRET=your-secret-at-least-32-characters-long
```

4 Create auth server instance

Create a unified auth instance in `lib/auth/server.ts` . This single instance provides all server-side auth functionality:

- `.handler()` for API routes
- `.middleware()` for route protection
- `.getSession()` and all Better Auth server methods

See the Next.js Server SDK reference for complete API documentation.

lib/auth/server.ts

```
import { createNeonAuth } from '@neondatabase/auth/next/server';

export const auth = createNeonAuth({
  baseUrl: process.env.NEON_AUTH_BASE_URL!,
  cookies: {
    secret: process.env.NEON_AUTH_COOKIE_SECRET!,
  },
});
```

5 Set up auth API routes

Create an API route handler that proxies auth requests. All Neon Auth APIs will be routed through this handler. Create a route file inside `/api/auth/[...path]` directory:

app/api/auth/[...path]/route.ts

```
import { auth } from '@/lib/auth/server';

export const { GET, POST } = auth.handler();
```

6 Add authentication middleware

The middleware ensures users are authenticated before accessing protected routes. Create `proxy.ts` file in your project root:

proxy.ts

```
import { auth } from '@/lib/auth/server';

export default auth.middleware({
  // Redirects unauthenticated users to sign-in page
  loginUrl: '/auth/sign-in',
});

export const config = {
  matcher: [
    // Protected routes requiring authentication
    '/account/:path*',
  ],
};
```

NOTE

> Your Next.js project is now fully configured to use Neon Auth. Now, lets proceed with setting up the Auth UI Provider and wrap your layout with auth context.

7 Configure the auth client

The Auth UI components need access to auth APIs. Create the auth client in `lib/auth/client.ts` file, which you'll pass to `NeonAuthUIProvider` .

> **NOTE**
>
> The server-side `auth` instance was already created in a previous step. The client is separate and handles browser-side auth operations.

lib/auth/client.ts

```
'use client';

import { createAuthClient } from '@neondatabase/auth/next';

export const authClient = createAuthClient();
```

8 Wrap app layout with auth provider

The `NeonAuthUIProvider` component wraps your application with authentication context and provides essential hooks and auth methods required by auth components throughout your app. To make authentication globally accessible, wrap your entire app with `NeonAuthUIProvider` .

> **HYDRATION WARNING**
>
> Add `suppressHydrationWarning` to the `<html>` tag to prevent React hydration errors caused by `next-themes` client-side theme switching. This property only applies one level deep, so it won't block hydration warnings on other elements.

Copy and paste the following code into your `app/layout.tsx` file.

The `NeonAuthUIProvider` can be fully customized with settings you have configured in Neon Console. For example:

- Add social providers like Google, Github, and Vercel on sign-in page
- Allow your users to create and manage organizations in `/account/organizations`
- Localization support

Example: Adding optional props

app/layout.tsx

```
+ import { authClient } from '@/lib/auth/client';
+ import { NeonAuthUIProvider, UserButton } from '@neondatabase/auth/react';
  import type { Metadata } from "next";
  import { Geist, Geist_Mono } from "next/font/google";
  import "./globals.css";

  const geistSans = Geist({
    variable: "--font-geist-sans",
    subsets: ["latin"],
  });

  const geistMono = Geist_Mono({
    variable: "--font-geist-mono",
    subsets: ["latin"],
  });

  export const metadata: Metadata = {
    title: 'My Neon App',
    description: 'A Next.js application with Neon Auth',
  };

  export default function RootLayout({
    children,
  }: Readonly<{
    children: React.ReactNode;
  }>) {
    return (
+     <html lang="en" suppressHydrationWarning>
        <body
          className={`${geistSans.variable} ${geistMono.variable} antialiased`}
        >
+         <NeonAuthUIProvider
+           authClient={authClient}
+           redirectTo="/account/settings"
+           emailOTP
+         >
+           <header className='flex justify-end items-center p-4 gap-4 h-16'>
+             <UserButton size="icon" />
+           </header>
+           {children}
+         </NeonAuthUIProvider>
        </body>
      </html>
    );
  }
```

9

Add Neon Auth styles

Import the Neon Auth UI styles in your `app/globals.css` file. Add this line at the top of the file:

NOT USING TAILWIND?

See UI Component Styles for alternative setup options.

app/globals.css

```
  @import "tailwindcss";
+ @import "@neondatabase/auth/ui/tailwind";
```

NOTE

Now that the Auth provider and styles are set up, let's build the pages for signing up and signing in

10 Create the Auth & Account pages

Create a dynamic route segment for authentication and account views in `app/auth/[path]/page.tsx` and `app/account/[path]/page.tsx` respectively.

- `AuthView` - with dynamic route segment covers the following paths:
    - `/auth/sign-in` - Sign in with email/password and social providers
    - `/auth/sign-up` New account registration
    - `/auth/sign-out` Sign the user out of the applications
- `AccountView` - with dynamic route segment covers the following paths:
    - `/account/settings` - User can manage their profile details
    - `/account/security` - Change password and list active session

create app & account page

Auth Page Account Page

Create a new page in `app/auth/[path]/page.tsx` and copy-paste following code:

```
import { AuthView } from '@neondatabase/auth/react';

export const dynamicParams = false;

export default async function AuthPage({ params }: { params: Promise<{ path: string }> }) {
  const { path } = await params;

  return (
    <main className="container mx-auto flex grow flex-col items-center justify-center gap-3 self
      <AuthView path={path} />
    </main>
  );
}
```

11 Access user data on server and client

Server Components:

Use the `auth` instance from `lib/auth/server.ts` to access session data and call auth methods in server components and server actions.

Client Components:

Use the `authClient` from `lib/auth/client.ts` to access session data and call auth methods in client components.

Access user data

Server Component | Client Component | API Route

Create a new page at `app/server-rendered-page/page.tsx` and add the following code:

```
import { auth } from '@/lib/auth/server';

// Server components using auth methods must be rendered dynamically
export const dynamic = 'force-dynamic';

export default async function ServerRenderedPage() {
  const { data: session } = await auth.getSession();

  return (
    <div className="max-w-xl mx-auto p-6 space-y-4">
      <h1 className="text-2xl font-semibold">Server Rendered Page</h1>

      <p className="text-gray-400">
        Authenticated:{' '}
        <span className={session ? 'text-green-500' : 'text-red-500'}>
          {session ? 'Yes' : 'No'}
        </span>
      </p>

      {session?.user && <p className="text-gray-400">User ID: {session.user.id}</p>}

      <p className="font-medium text-gray-700 dark:text-gray-200">Session and User Data:</p>

      <pre className="bg-gray-100 dark:bg-gray-800 p-4 rounded text-sm overflow-x-auto text-gray
        {JSON.stringify({ session: session?.session, user: session?.user }, null, 2)}
      </pre>
    </div>
  );
}
```

12 Start your app

Start the development server, and then open http://localhost:3000/

- Visit `/auth/sign-in` to sign in or sign up
- Visit `/account/settings` to view account settings
- Visit `/server-rendered-page` to see user data on server

```
npm run dev
```

Last updated on January 30, 2026

Was this page helpful? Yes No