Email verification

Verify user email addresses during sign-up or account creation

BETA

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

Email verification ensures users own the email addresses they register with. Neon Auth supports two verification methods:

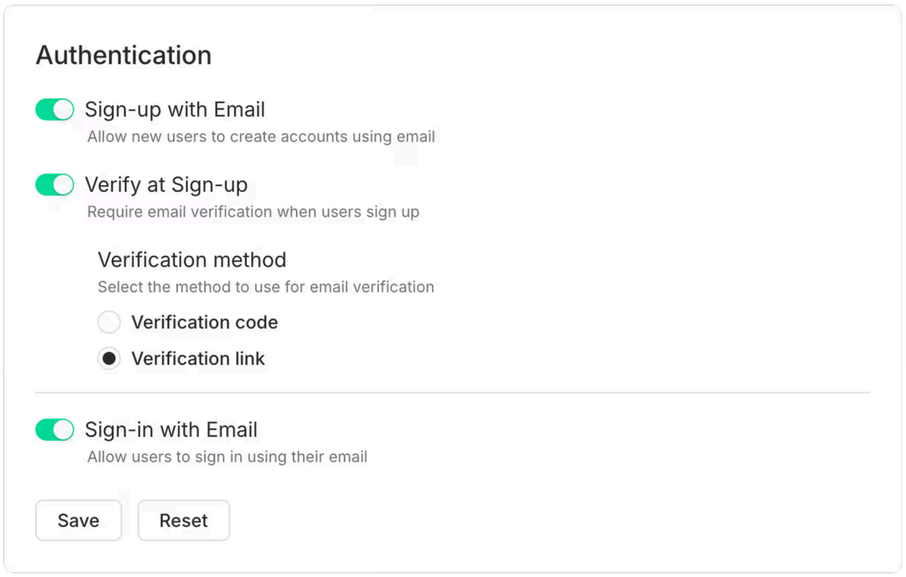
- **Verification codes** (users enter a numeric code from their email) - works with shared or custom email providers
- **Verification links** (users click a link in their email) - requires a custom email provider

NOTE

Verification links require a custom email provider. If you're using the shared email provider, use verification codes instead.

Enable email verification

In your project's **Settings** → **Auth** page, enable **Sign-up with Email** and **Verify at Sign-up**. Choose your verification method.



Authentication

Sign-up with Email: on
Allow new users to create accounts using email

Verify at Sign-up: on
Require email verification when users sign up

Verification method
Select the method to use for email verification

○ Verification code
● Verification link

Sign-in with Email: on
Allow users to sign in using their email

Verification links require a custom email provider. See Email provider configuration to set this up.

When a user clicks the verification link in their email, the Neon Auth server handles verification and redirects them back to your application. Your app checks for the new session and shows the appropriate UI.

1. Check session on mount

Add a session check when your component mounts to detect when a user returns from clicking the verification link:

src/App.jsx

```jsx
import { useEffect, useState } from 'react';
import { authClient } from './auth';

export default function App() {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    authClient.getSession().then(({ data }) => {
      if (data?.session) {
        setUser(data.session.user);
      }
      setLoading(false);
    });
  }, []);
}
```

2. Handle sign-up with verification

After calling `signUp.email()` , check if verification is required and show a message:

src/App.jsx

```jsx
const handleSignUp = async (e) => {
  e.preventDefault();
  setMessage('');

  try {
    const { data, error } = await authClient.signUp.email({
      email,
      password,
      name: name || email.split('@')[0] || 'User',
    });

    if (error) throw error;

    // Check if email verification is required
    if (data?.user && !data.user.emailVerified) {
      setMessage('Check your email for a verification link!');
    } else {
      setMessage('Account created! Please sign in.');
    }
  } catch (error) {
    setMessage(error?.message || 'An error occurred');
  }
};
```

3. Check verification status

Access the `emailVerified` field from the user object:

src/App.jsx

```
const { data } = await authClient.getSession();

if (data?.session?.user && !data.session.user.emailVerified) {
  // Show verification prompt or restrict features
  console.log('Please verify your email to continue');
}
```

Verification codes

If you prefer verification codes, users receive a numeric code via email and enter it in your application. Your app switches between the auth form and a verification form.

1. Add verification state

Add state to track which form to show:

src/App.jsx

```
const [step, setStep] = useState('auth'); // 'auth' or 'verify'
const [code, setCode] = useState('');
```

2. Handle code verification

Create a handler for code verification:

src/App.jsx

```
const handleVerify = async (e) => {
  e.preventDefault();
  setMessage('');

  try {
    const { data, error } = await authClient.emailOtp.verifyEmail({
      email,
      otp: code,
    });

    if (error) throw error;

    // Check if auto-sign-in is enabled (default behavior)
    if (data?.session) {
      setUser(data.session.user);
      setStep('auth');
    } else {
      setMessage('Email verified! You can now sign in.');
      setStep('auth');
      setIsSignUp(false);
      setCode('');
    }
  } catch (error) {
    setMessage(error?.message || 'An error occurred');
  }
};
```

3. Show verification form

When `step` is `'verify'`, show the verification form:

src/App.jsx

```
if (step === 'verify') {
  return (
    <div>
      <h1>Verify Your Email</h1>
      <p>Enter the code sent to {email}</p>
      <form onSubmit={handleVerify}>
        <input
          type="text"
          placeholder="Verification code"
          value={code}
          onChange={(e) => setCode(e.target.value)}
          required
        />
        <button type="submit">Verify</button>
      </form>
      {message && <p>{message}</p>}
    </div>
  );
}
```

4. Switch to verification after sign-up

After calling `signUp.email()`, switch to the verification step:

src/App.jsx

```
if (data?.user && !data.user.emailVerified) {
  setMessage('Check your email for a verification code');
  setStep('verify'); // Switch to verification form
}
```

Complete example: App.jsx with verification codes

Resending verification emails

Both verification links and verification codes expire after **15 minutes**. Allow users to request a new one:

src/App.jsx

```
const handleResend = async () => {
  try {
    const { error } = await authClient.sendVerificationEmail({
      email,
      callbackURL: window.location.origin + '/',
    });

    if (error) throw error;
    setMessage('Verification email sent! Check your inbox.');
  } catch (error) {
    setMessage(error?.message || 'An error occurred');
  }
};
```

The server sends whichever type (verification link or verification code) you configured in the Console.

Required vs optional verification

When email verification is **required** in your Console settings, users cannot sign in until they verify. When verification is **optional**, users can sign in immediately but their `emailVerified` field remains `false` until verified.

Last updated on January 7, 2026 Was this page helpful? Yes No

NEON

A Databricks Company

All systems operational