AI VOID FIELD GUIDE

# Step-by-Step Tutorials

Focused, practical tutorials that teach you how to accomplish specific tasks step by step. Learn by doing with clear instructions and working code examples.

AI VOID

2026-05-20 20:31 IST

# Contents

CHAPTER 01

# Integrate Passkeys in Next.js on Vercel

**What you'll build:** You will integrate passkey authentication into a Next.js application and deploy it successfully on Vercel, understanding how to manage passkey flows in a serverless environment. **Time needed:** ~75 minutes **Prerequisites:** Basic understanding of Next.js and React, Node.js (LTS) and npm/yarn/pnpm installed, A Vercel account, A Git provider account (e.g., GitHub), Familiarity with environment variables **Version used:** unknown

---

## Introduction: Understanding Passkeys on Vercel

Welcome! In this tutorial, we're going to dive into one of the most exciting advancements in web authentication: **Passkeys**. Passkeys offer a significant leap forward in security and user experience, replacing traditional passwords with cryptographic credentials tied to your devices. Imagine logging in with just your fingerprint, face scan, or device PIN, without ever typing a password. That's the power of passkeys.

### Why Passkeys Matter

Passkeys solve many of the chronic problems associated with passwords:

- **Phishing Resistance**: Passkeys are cryptographically bound to the website they were created for, making them immune to phishing attacks.
- **Improved Security**: They eliminate weak, reused, or stolen passwords, drastically reducing the risk of account takeovers.
- **Enhanced User Experience**: No more remembering complex passwords, password resets, or multi-factor authentication codes. Just a simple, secure biometric or PIN verification.

### Next.js and Vercel: A Perfect Match for Passkeys

Next.js, the React framework for the web, combined with Vercel's serverless platform, provides an ideal environment for implementing passkeys.

- **Serverless Flexibility**: Vercel's Edge Functions and Serverless Functions allow you to handle the server-side components of passkey authentication (like generating cryptographic challenges) without managing a dedicated server.
- **Seamless Deployment**: Vercel's Git integration makes deploying your Next.js app, including passkey functionality, incredibly straightforward.
- **Scalability**: Both Next.js and Vercel are designed for scale, ensuring your authentication system can grow with your application.

At its core, passkey technology is built on the **WebAuthn API**. This browser API allows web applications to interact with platform authenticators (like Touch ID, Face ID, Windows Hello) or roaming authenticators (like YubiKeys) to create and use cryptographic credentials. While WebAuthn is powerful, implementing it directly can be complex due to the cryptographic challenges and server-side state management involved. This is where third-party passkey providers often come in, simplifying the integration process.

By the end of this section, you understand the "why" behind passkeys and their synergy with Next.js and Vercel.

---

## Prerequisites and Initial Next.js Project Setup

Before we jump into code, let's ensure your development environment is ready. We'll start by creating a fresh Next.js project.

### Verifying Prerequisites

Make sure you have the following installed:

- **Node.js (LTS version)**: You can check your version by running `node -v`.
- **npm, yarn, or pnpm**: These package managers come with Node.js.
- **A Git provider account**: Such as GitHub, GitLab, or Bitbucket. You'll need this to deploy to Vercel.
- **A Vercel account**: Sign up at vercel.com.

## Creating Your Next.js Project

Let's create a new Next.js application. We'll use TypeScript as specified in our brief, and keep it simple.

1. **Open your terminal** and navigate to the directory where you want to create your project.
2. **Run the following command** to create a new Next.js application. We'll name it `nextjs-passkey-app`.

```
    npx create-next-app@latest nextjs-passkey-app --typescript --eslint --
tailwind --app --src-dir --use-pnpm
```

```
> ⚡ Note: We're using `pnpm` here for faster and more efficient package
management. If you prefer `npm` or `yarn`, replace `--use-pnpm` with `--use-
npm` or `--use-yarn` respectively. The `--app` flag ensures we use the App
Router, and `--src-dir` places our code in a `src` directory.
```

1. **Follow the prompts**:
    - `Would you like to use App Router? (recommended)`: `Yes`
    - `Would you like to customize the default import alias?`: `No`
2. **Navigate into your new project directory**:

```
    cd nextjs-passkey-app
```

## Running the Initial Project

Now, let's make sure everything is set up correctly by running the development server.

1. **Start the development server**:

```
    pnpm dev
```

```
or if you used npm/yarn:
```

```
    npm run dev
    # or
```

```
yarn dev
```

1. **Open your browser** and go to `<http://localhost:3000>`. You should see the default Next.js starter page.

> ⚡ Note: Keep this terminal window open. The development server will automatically reload when you make changes.

You've successfully set up your Next.js project. We now have a clean slate to begin integrating passkeys.

---

## Choosing and Integrating a Passkey Provider

Implementing WebAuthn directly involves managing complex cryptographic challenges, handling various credential types, and securely storing public keys on your server. This can be a significant undertaking. For a hands-on tutorial, leveraging a dedicated passkey provider simplifies this process immensely, allowing us to focus on the Next.js and Vercel integration.

Based on our research, **Corbado** offers a straightforward solution for integrating passkeys with Next.js and Vercel. We will use Corbado for this tutorial.

### Why Use a Passkey Provider?

- **Simplified WebAuthn**: Providers abstract away the complexities of the WebAuthn API.
- **Backend as a Service**: They handle the secure storage of public keys, challenge generation, and verification.
- **Cross-Platform Support**: Often provide SDKs that work seamlessly across different browsers and devices.
- **Compliance & Security**: They specialize in authentication security, ensuring best practices are followed.

## Setting Up Corbado

First, let's get a Corbado account and configure your project.

1. **Sign up for a free Corbado account**:
    - Go to corbado.com and sign up.
    - Once logged in, create a new project. You'll be given a **Project ID** and potentially an **API Secret**. Keep these handy; we'll need them for environment variables later.
2. **Configure your Corbado Project**:
    - In your Corbado dashboard, navigate to **Settings > Basic Information**.
    - Add your local development URL (`<http://localhost:3000>`) and your future Vercel deployment URL (e.g., `<https://your-app-name.vercel.app>`) to the **Origin URLs** list. This is crucial for WebAuthn to function correctly.
    - Make sure to enable Passkey-first authentication in your project settings if it's not already.

## Installing the Corbado SDK

Now, let's add the necessary Corbado SDK to our Next.js project.

1. **Stop your development server** (if it's still running) by pressing `Ctrl + C` in your terminal.
2. **Install the Corbado React SDK**:

```
pnpm add @corbado/react @corbado/web
```

```
> ⚡ Note: `@corbado/react` provides React components for easy integration,
while `@corbado/web` is the core WebAuthn client.
```

## Initializing Corbado in Your Next.js App

We need to initialize the Corbado client so it's available throughout our application. The best place for this in an App Router project is typically within a layout file or a client component.

1. **Create a `CorbadoProvider` component**: Inside your `src/app` directory, create a new file `src/app/corbado-provider.tsx`:

```
    // src/app/corbado-provider.tsx
    'use client';

    import { CorbadoProvider as CorbadoReactProvider } from '@corbado/react';
    import { ReactNode } from 'react';

    interface CorbadoProviderProps {
      children: ReactNode;
    }

    export default function CorbadoProvider({ children }:
CorbadoProviderProps) {
      const projectId = process.env.NEXT_PUBLIC_CORBADO_PROJECT_ID;

      if (!projectId) {
        console.error('NEXT_PUBLIC_CORBADO_PROJECT_ID is not defined.');
        return <div>Error: Corbado Project ID missing.</div>;
      }

      return (
        <CorbadoReactProvider projectId={projectId}>
          {children}
        </CorbadoReactProvider>
      );
    }
```

```
> ⚡ Note: We use `'use client'` because `@corbado/react` components require
client-side execution. `NEXT_PUBLIC_` prefix is essential for environment
variables to be exposed to the browser.
```

1. **Wrap your application with `CorbadoProvider`**: Modify your `src/app/layout.tsx` file to include the `CorbadoProvider`. This ensures Corbado is initialized for all pages.

```
    // src/app/layout.tsx
    import type { Metadata } from 'next';
    import { Inter } from 'next/font/google';
    import './globals.css';
    import CorbadoProvider from './corbado-provider'; // Import our new
provider

    const inter = Inter({ subsets: ['latin'] });

    export const metadata: Metadata = {
      title: 'Next.js Passkey App',
      description: 'Integrate Passkeys in Next.js on Vercel',
    };

    export default function RootLayout({
      children,
    }: Readonly<{
      children: React.ReactNode;
    }>) {
      return (
```

```
        <html lang="en">
          <body className={inter.className}>
            <CorbadoProvider> {/* Wrap children with CorbadoProvider */}
              {children}
            </CorbadoProvider>
          </body>
        </html>
      );
    }
```

1. **Create an environment variable file**: In the root of your project, create a new file named `.env.local`. Add your Corbado Project ID here.

```
# .env.local
NEXT_PUBLIC_CORBADO_PROJECT_ID="<YOUR_CORBADO_PROJECT_ID>"
```

```
**Replace `<YOUR_CORBADO_PROJECT_ID>` with the actual Project ID from your
Corbado dashboard.**
```

1. **Restart your development server** to load the new environment variable:

```
pnpm dev
```

You won't see any visual changes yet, but Corbado is now initialized and ready to be used in your application components. We've successfully integrated the Corbado SDK and prepared our application for passkey flows.

---

## Implementing Passkey Registration and Authentication Flows

Now that Corbado is set up, let's create the UI and logic for users to register and authenticate using passkeys. We'll build a simple page that displays the Corbado authentication component.

### Creating the Authentication Page

We'll modify the default `page.tsx` to include Corbado's pre-built authentication component. This component handles the entire passkey flow (registration, login, and even fallback to email OTP if passkeys aren't available).

1. **Modify `src/app/page.tsx`**: Replace the existing content in `src/app/page.tsx` with the following:

```
    // src/app/page.tsx
    'use client'; // This component will run on the client side

    import { Auth } from '@corbado/react';
    import { useRouter } from 'next/navigation';
    import { useCorbado } from '@corbado/react';
    import { useEffect, useState } from 'react';

    export default function HomePage() {
      const router = useRouter();
      const { user, isAuthenticated, loading } = useCorbado();
      const [showAuth, setShowAuth] = useState(false);

      useEffect(() => {
        if (!loading) {
          if (isAuthenticated) {
            // User is authenticated, redirect to a dashboard or profile page
            router.push('/dashboard');
          } else {
            // No user, show the auth component
            setShowAuth(true);
          }
        }
      }, [isAuthenticated, loading, router]);

      if (loading) {
        return (
          <div className="flex min-h-screen flex-col items-center justify-
center p-24">
            <p>Loading authentication state...</p>
          </div>
        );
      }

      return (
        <main className="flex min-h-screen flex-col items-center justify-
center p-24">
          <h1 className="text-4xl font-bold mb-8">Welcome to Passkey Demo!</
h1>
          {showAuth && (
            <div className="w-full max-w-md">
              <Auth
                onLoggedIn={() => {
                  console.log('User logged in!');
                  router.push('/dashboard'); // Redirect after successful
login/registration
                }}
                onSignedUp={() => {
                  console.log('User signed up!');
                  router.push('/dashboard'); // Redirect after successful
signup
                }}
                onLoginFailed={error => {
                  console.error('Login failed:', error);
                  alert('Login failed: ' + error.message);
                }}
                onSignupFailed={error => {
                  console.error('Signup failed:', error);
                  alert('Signup failed: ' + error.message);
                }}
```

```
            />
          </div>
        )}
      </main>
    );
  }
```

> ⚡ Note: The `Auth` component from `@corbado/react` is a powerful, pre-built UI that handles the entire passkey flow. It will automatically detect if a user has a passkey registered or prompt them to create one.

## Creating a Protected Dashboard Page

Let's create a simple dashboard page that only authenticated users can access.

1. **Create `src/app/dashboard/page.tsx`:**

```
  // src/app/dashboard/page.tsx
  'use client';

  import { useCorbado } from '@corbado/react';
  import { useRouter } from 'next/navigation';
  import { useEffect } from 'react';

  export default function DashboardPage() {
    const router = useRouter();
    const { user, isAuthenticated, loading, logout } = useCorbado();

    useEffect(() => {
      if (!loading && !isAuthenticated) {
        router.push('/'); // Redirect to home if not authenticated
      }
    }, [isAuthenticated, loading, router]);

    if (loading || !isAuthenticated) {
      return (
        <div className="flex min-h-screen flex-col items-center justify-
center p-24">
          <p>Loading dashboard...</p>
        </div>
      );
    }

    return (
      <main className="flex min-h-screen flex-col items-center justify-
center p-24">
        <h1 className="text-4xl font-bold mb-8">Welcome, {user?.name || 'Use
r'}!</h1>
        <p className="text-lg mb-4">You are successfully authenticated
using passkeys.</p>
        <p className="text-md mb-8">Your email: {user?.email}</p>
        <button
          onClick={async () => {
            await logout();
            router.push('/');
          }}
```

```
            className="px-6 py-3 bg-red-600 text-white rounded-lg hover:bg-
red-700 transition-colors"
          >
            Logout
          </button>
        </main>
      );
    }
```

### Testing Passkey Registration and Login Locally

Now, let's see our passkey integration in action!

1. **Ensure your development server is running**:

```
pnpm dev
```

1. **Open your browser to `<http://localhost:3000>`**.
    - You should see the Corbado `Auth` component.
    - **First-time user**: Enter an email address. Corbado will guide you through creating a passkey. This usually involves a browser prompt to use your device's biometric sensor (fingerprint, face ID) or PIN.
    - **Returning user**: If you've already registered with a passkey on this device, entering your email will prompt you to authenticate with your passkey directly.
2. **Upon successful registration or login**, you should be redirected to the `/dashboard` page.
3. **Test the logout functionality**. Clicking "Logout" should take you back to the home page, where you can log in again.

You've now implemented the core passkey registration and authentication flows using a third-party provider, and verified it works locally. This significantly simplifies the WebAuthn process.

---

## Configuring Environment Variables for Vercel

Environment variables are critical for managing sensitive information and configuration specific to different deployment environments (development, staging, production). For our Next.js app on Vercel, we need to ensure that the `NEXT_PUBLIC_CORBADO_PROJECT_ID` is available, and potentially other API keys if your application grows.

## Understanding Environment Variables in Next.js

- `.env.local`: Used for local development. Variables defined here are only available on your local machine.
- **`NEXT_PUBLIC_` prefix**: Any environment variable prefixed with `NEXT_PUBLIC_` will be exposed to the browser. This is necessary for client-side code that needs to interact with services like Corbado. Variables without this prefix are only available in Node.js environments (API routes, `getServerSideProps`, `getStaticProps`, etc.).
- **Vercel Dashboard**: For production deployments, you must configure environment variables directly in your Vercel project settings. Variables in `.env.local` are not automatically deployed to Vercel.

## Adding Environment Variables to Vercel

1. **Commit and push your code to a Git repository**: If you haven't already, initialize a Git repository and push your project to a service like GitHub.

```
    git init
    git add .
    git commit -m "Initial commit with Corbado integration"
    git branch -M main
    git remote add origin https://github.com/YOUR_USERNAME/nextjs-passkey-
app.git # Replace with your repo URL
    git push -u origin main
```

1. **Log in to your Vercel account** and go to your dashboard.
2. **Import your Git repository**:
    - Click "Add New..." -> "Project".
    - Select "Import Git Repository" and choose the repository you just pushed (e.g., `nextjs-passkey-app`).
    - Vercel will automatically detect that it's a Next.js project.

3. **Configure Environment Variables**:
    - During the import process, or after the project is created, navigate to your project's **Settings** tab.
    - Click on **Environment Variables** in the sidebar.
    - Add a new environment variable:
        - **Name**: `NEXT_PUBLIC_CORBADO_PROJECT_ID`
        - **Value**: Your actual Corbado Project ID (the same one you put in `.env.local`).
        - **Environments**: Ensure it's set for `Production`, `Preview`, and `Development` (or at least `Production` and `Preview`).

    > ⚠️ Common mistake: Forgetting to add `NEXT_PUBLIC_` prefixed variables to Vercel. If your client-side Passkey flow fails on Vercel, this is often the culprit. Vercel's build process needs these variables to be explicitly defined.

4. **Save your changes**.

By correctly setting up these environment variables, your Next.js application will have access to the necessary Corbado credentials when deployed on Vercel, allowing the passkey functionality to work seamlessly in production.

---

## Deploying Your Next.js App to Vercel

Now that our Next.js application is integrated with passkeys and configured with environment variables, it's time to deploy it to Vercel. Vercel is specifically designed for Next.js, making the deployment process incredibly smooth.

### Connecting Your Git Repository to Vercel

If you followed the previous step to add environment variables, you've likely already imported your Git repository. If not, here's how:

1. **Log in to your Vercel account** and go to your dashboard.
2. Click "Add New..." -> "Project".
3. Select "Import Git Repository" and choose the repository where you pushed your `nextjs-passkey-app` code.
4. Vercel will automatically detect that it's a Next.js project.

5. Click "Deploy".

> ⚡ Note: Vercel integrates directly with GitHub, GitLab, and Bitbucket. Once connected, every push to your main branch (or a configured branch) will trigger an automatic deployment.

### Understanding the Vercel Deployment Process

When you deploy a Next.js app to Vercel, several things happen automatically:

- **Build Step**: Vercel runs your project's build command (`next build` by default). This compiles your React components, optimizes images, generates static assets, and creates serverless functions for your API routes or server components.
- **Asset Upload**: All static assets (HTML, CSS, JS, images) are uploaded to Vercel's CDN (Content Delivery Network).
- **Serverless Function Deployment**: Your API routes and any server-side logic are deployed as serverless functions (either Edge Functions or Lambda functions, depending on your configuration and Next.js version).
- **Preview URLs**: For every pull request, Vercel can generate a unique preview URL, allowing you to test changes before merging to `main`.
- **Production Deployment**: Once merged to your main branch, Vercel deploys to your production domain.

### Initiating Your First Deployment

If you've already imported your project, Vercel will likely have started the deployment automatically. You can monitor its progress:

1. In your Vercel dashboard, click on your `nextjs-passkey-app` project.
2. Go to the "Deployments" tab.
3. You'll see the status of your latest deployment (Building, Deploying, Ready).
4. Once the deployment status changes to "Ready", you'll see a "Visit" button or a domain link (e.g., `<https://nextjs-passkey-app-xxxxxxxx.vercel.app>`). Click this to open your live application.

> ⚠️ Common mistake: If the deployment fails, check the build logs in the Vercel dashboard. Common issues include missing environment variables, syntax errors in your code, or incorrect build commands.

You have successfully deployed your Next.js application with passkey integration to Vercel. The next step is to verify that the passkey functionality works correctly in the live environment.

---

## Testing Passkey Functionality on Vercel

With your application now live on Vercel, it's crucial to test the passkey registration and authentication flows in the deployed environment. This confirms that all configurations, especially environment variables and origin settings, are correct.

### Accessing Your Deployed Application

1. **Open your browser** and navigate to the unique Vercel URL for your project (e.g., `<https://your-app-name.vercel.app>`). You can find this URL in your Vercel dashboard on the project overview page or the "Deployments" tab.

### Performing Passkey Registration (New User)

1. **Simulate a new user**: If you've already registered with a passkey on your device for this app, you might want to try with a different email address or on a different device to simulate a fresh registration.
2. **Enter an email address** into the Corbado `Auth` component on the home page.
3. **Follow the browser prompts** to create a new passkey. This will involve your device's biometric authentication (fingerprint, face scan) or PIN.
    - The browser will confirm that you are creating a passkey for the specific Vercel domain.
    - Ensure your Corbado project settings include your Vercel domain in the "Origin URLs." If this is incorrect, the passkey creation will fail with an error like "The operation is not allowed on this origin."

### Performing Passkey Authentication (Returning User)

1. **Log out** from the dashboard if you were redirected there after registration.
2. **Return to the home page** (`<https://your-app-name.vercel.app>`).
3. **Enter the email address** you used for registration.

4. **Follow the browser prompts** to authenticate with your existing passkey. Again, this will involve your device's biometric authentication or PIN.
    - The browser will confirm you are using a passkey for the Vercel domain.

### Verifying Functionality

- **Successful Login/Registration**: After either flow, you should be redirected to the `/dashboard` page, confirming that authentication was successful.
- **Logout**: Test the logout button on the dashboard to ensure it clears the session and returns you to the home page.
- **Error Handling**: Intentionally try to use an invalid email or cancel a passkey prompt to see how the application handles errors (e.g., the `onLoginFailed` or `onSignupFailed` callbacks).

By successfully completing these tests, you've confirmed that your Next.js application, deployed on Vercel, can effectively register and authenticate users using passkeys. This is a significant milestone in enhancing your application's security and user experience.

---

## Troubleshooting Common Deployment and Passkey Issues

Even with careful setup, you might encounter issues during deployment or when testing passkey functionality. Here's a guide to common problems and how to troubleshoot them.

## 1. Vercel Deployment Failures

- **Missing Environment Variables**:
    - **Problem**: Your build fails, or the app throws errors like "Corbado Project ID missing" on Vercel, even though it works locally.
    - **Cause**: You forgot to add `NEXT_PUBLIC_CORBADO_PROJECT_ID` (or other necessary `NEXT_PUBLIC_` variables) to your Vercel project's Environment Variables settings.
    - **Solution**: Double-check Vercel project settings (`Settings > Environment Variables`). Ensure the variable name and value are correct for all relevant environments (Production, Preview, Development).
    - `> ⚠️ Common mistake:` Developers often forget that `.env.local` is only for local development and variables must be explicitly set in Vercel.
- **Build Command Errors**:
    - **Problem**: Vercel logs show errors during the `next build` step.
    - **Cause**: Syntax errors in your code, missing dependencies, or incompatible Node.js versions.
    - **Solution**:
        - Inspect the Vercel build logs carefully for specific error messages.
        - Run `pnpm install` (or `npm install`/`yarn install`) and `pnpm build` (or `npm run build`/`yarn build`) locally to ensure your project builds without errors.
        - Check your `package.json` for any missing dependencies or incorrect scripts.
        - Ensure your Vercel project's Node.js version (configured in `Settings > General > Node.js Version`) matches your local development environment.
- **Incorrect `pnpm` Configuration**:
    - **Problem**: If you used `pnpm`, Vercel might default to `npm` or `yarn` and fail to install dependencies.
    - **Solution**: In Vercel project settings (`Settings > General > Build & Development Settings`), ensure the "Install Command" is `pnpm install` and "Build Command" is `pnpm build`.

## 2. Passkey Functionality Issues

- **"The operation is not allowed on this origin" Error**:
    - **Problem**: Passkey registration or authentication fails with this specific error message, especially on the deployed Vercel app.
    - **Cause**: The domain where your app is running (e.g., `<https://your-app-name.vercel.app>`) is not registered as an allowed origin in your Corbado project settings.
    - **Solution**: Go to your Corbado dashboard (`Settings > Basic Information`) and add your Vercel production domain (and any preview domains if you want to test them) to the "Origin URLs" list. This is a critical security measure for WebAuthn.
- **Passkey Prompts Not Appearing**:
    - **Problem**: The Corbado `Auth` component loads, but when you enter an email, no passkey prompt appears from the browser.
    - **Cause**:
        - **Browser/Device Support**: The browser or device you're using might not fully support passkeys or WebAuthn.
        - **Corbado Configuration**: Passkey-first authentication might not be enabled in your Corbado project settings.
        - **Network/Ad Blocker**: Sometimes, network issues or aggressive ad blockers can interfere with WebAuthn API calls.
    - **Solution**:
        - Try a different, modern browser (Chrome, Safari, Edge) on a device with biometric capabilities (macOS with Touch ID, Windows with Hello, iOS, Android).
        - Verify Corbado project settings.
        - Check your browser's developer console for any network errors or security policy violations.

- **User Not Redirected After Authentication**:
    - **Problem**: A passkey flow completes, but the user isn't redirected to `/dashboard`.
    - **Cause**: Issues with the `onLoggedIn` or `onSignedUp` callbacks in your `Auth` component, or an issue with the `useRouter` hook.
    - **Solution**:
        - Add `console.log` statements inside your `onLoggedIn` and `onSignedUp` callbacks to see if they are being triggered.
        - Ensure your `router.push('/dashboard')` call is correctly executed.
        - Check the browser console for any JavaScript errors.

### General Debugging Tips

- **Vercel Logs**: Always check the "Logs" tab for your deployment in the Vercel dashboard. This is your primary source for server-side errors.
- **Browser Developer Console**: For client-side issues, open your browser's developer console (F12 or Cmd+Option+I). Look for errors in the "Console" tab and network requests in the "Network" tab.
- **Corbado Dashboard Logs**: Corbado typically provides logs for authentication attempts, which can be invaluable for debugging issues related to their service.

By systematically going through these troubleshooting steps, you can resolve most common issues encountered when integrating and deploying passkeys with Next.js on Vercel.

## What to Build Next

Congratulations! You've successfully integrated passkey authentication into a Next.js application and deployed it to Vercel. This is a solid foundation, but there's always more to explore and build. Here are three concrete ideas to extend your project and deepen your understanding:

1. **User Profile Management**:
    - **Challenge**: Extend the `/dashboard` page to allow authenticated users to view and update their profile information (e.g., name, display picture).
    - **Implementation**: Create a Next.js API route (`src/app/api/user/route.ts`) that uses the Corbado SDK (or a similar backend library) to fetch or update user details. Secure this API route to ensure only authenticated users can access it. You'll need to understand how to verify the user's session on the server-side.
    - **Learning**: This will teach you about securing API routes, server-side session validation, and interacting with your authentication provider's backend APIs.
2. **Add Multi-Factor Authentication (MFA) or Social Logins**:
    - **Challenge**: While passkeys are strong, some users might still prefer or need additional authentication methods. Integrate a secondary factor like a Time-based One-Time Password (TOTP) or allow users to log in with social providers (Google, GitHub).
    - **Implementation**: Many passkey providers like Corbado offer these features out-of-the-box or with minimal configuration. Explore their documentation for integrating these additional methods into your existing `Auth` component or by adding separate components for these flows.
    - **Learning**: You'll gain insight into flexible authentication strategies, managing multiple credential types, and enhancing user choice while maintaining security.

3. **Implement Webhooks for User Events**:
    - **Challenge**: React to user events (e.g., new user registration, passwordless login, account deletion) in real-time within your application or other services.
    - **Implementation**: Configure webhooks in your Corbado dashboard to send notifications to a Vercel Serverless Function (an API route in Next.js). This function would then process the event, perhaps by sending a welcome email, updating an internal CRM, or logging the activity.
    - **Learning**: This will introduce you to event-driven architectures, securing webhooks, and integrating your authentication system with other parts of your application ecosystem.