Configuring IDKit

IDKit is a single React component, and we'll be writing two callback functions:handleVerify andonSuccess .

IDKitWidget

First, add theIDKitWidget component to your site. The values for theapp_id andaction props were obtained from the Developer Portal inInstallation . We'll define thehandleVerify andonSuccess callbacks next.

/verify.tsx

'use client'

// for Next.js app router import { IDKitWidget , VerificationLevel , ISuccessResult } from

'@worldcoin/idkit'

// ...

< IDKitWidget app_id = "your app id"

// obtained from the Developer Portal action = "your action id"

// obtained from the Developer Portal onSuccess = {onSuccess} // callback when the modal is closed handleVerify = {handleVerify} // callback when the proof is received verification_level = { VerificationLevel .Orb}

{({ open }) => // This is the button that will open the IDKit modal < button

onClick

{open}>Verify with World ID</ button

} </ IDKitWidget

Copy Copied!

handleVerify

ThehandleVerify callback is called when the user's proof is received. This will send the proof to your backend for verification:

/verify.tsx

const

handleVerify

=

async (proof :

ISuccessResult) => { const

res

=

await

fetch ("/api/verify" , { // route to your backend will depend on implementation method :

"POST" , headers : { "Content-Type" :

"application/json" , } , body :

JSON .stringify (proof) , }) if (! res .ok) { throw

new

Error ("Verification failed."); // IDKit will display the error message to the user in the modal } }; Copy Copied!

Backend Verification

ThehandleVerify callback sends the proof to your backend for verification. The backend handler should verify the proof with the Developer Portal API:

The call to the Developer Portal API must be made from your backend, not from the frontend.

/api/verify.ts

const

verifyRes

=

await

fetch (https://developer.worldcoin.org/api/v1/verify/ { your_app_id } , { method :

"POST" , headers : { "Content-Type" :

"application/json" , } , body :

JSON .stringify ({ ... proof , action :

"your action id" }) , }) Copy Copied!

If the verification request to the Developer Portal succeeds, the backend should perform any necessary actions, such as updating a user's status in a database. If the verification request fails, the backend should handle the error and return an appropriate response to the frontend.

/api/verify.ts

if (verifyRes .ok) { // This is where you should perform backend actions if the verification succeeds // Such as, setting a user as "verified" in a database res .status (verifyRes .status) .send ({ code :

"success" , detail :

"This action verified correctly!" , }); } else { // This is where you should handle errors from the World ID /verify endpoint. // Usually these errors are due to a user having already verified. res .status (verifyRes .status) .send ({ code :

wldResponse .code , detail :

wldResponse .detail }); } Copy Copied!

onSuccess

TheonSuccess callback is called when the user closes the modal. This is where you can perform any necessary actions, such as redirecting the user to a new page.

/verify.tsx

const

onSuccess

= () => { // This is where you should perform any actions after the modal is closed // Such as redirecting the user to a new page window . location .href =

"/success" ; }; Copy Copied!