Cloud Incognito Actions

IDKit is required in your app's frontend for Incognito Actions, and thezero-knowledge proof received from the user will be verified via theDeveloper Portal API in your backend.

Creating actions

Create an action for your app in the Developer Portal. You must provide the following values:

- Action Name
- : The stringified action to be taken by the user.
- Description
- : This is shown to your user in the World app as they sign with their World ID. Make sure to fully describe the exact action the user is taking.
- Max Verifications
- : The number of times a user can take this action. A value of0
- indicates that unlimited verifications can take place.

An action scopes uniqueness for users, which means users will always generate the same ID (nullifier hash) when performing the same action.Cloud actions natively handle sybil-resistance with a limit set in the Developer Portal.

Installing IDKit

The JS package can be included in your project either as a module (which supports tree shaking to reduce bundle size) or you can add the script directly to your website.

Install IDKit

npm yarn pnpm npm install

@worldcoin/idkit Copy Copied!

Usage

Import and render IDKit. You'll want to do this on the screen where the user executes the protected action (e.g. before they click "Claim airdrop" or "Vote on proposal").

import { IDKitWidget , VerificationLevel } from

'@worldcoin/idkit'

< IDKitWidget app_id = "app_GBkZ1KlVUdFTjeMXKlVUdFT"

// obtained from the Developer Portal action = "vote_1"

// this is your action id from the Developer Portal onSuccess = {onSuccess} // callback when the modal is closed handleVerify = {handleVerify} // optional callback when the proof is received verification_level = { VerificationLevel .Device}

{({ open }) => < button

onClick

{open}>Verify with World ID</ button

} </ IDKitWidget

Copy Copied!

More configuration options can be found in theIDKit reference .

When a user clicks the button, the IDKit modal will open and prompt them to scan a QR code and verify with World ID. Once this proof is received, the optionalhandleVerify callback is called immediately and theonSuccess callback will be called when the modal is closed. One of these callbacks should begin the process of verifying the proof.

IDKit with Dynamic Actions

To accommodate dynamic content, actions can also be created at the time a user completes a World ID verification. Simply pass the desiredaction andaction_description values inIDKit's parameters . A new action will automatically be created and tracked, and will appear the next time you log into the Developer Portal.

As an example, using IDKit with Dynamic Actions may look like this:

const

getUserChoice

= userId => { const

choice

= userChoices['userId'] return choice }

return (< IDKitWidget { /... / } action = { getUserChoice (userId)} action_description = "verify for an action" { /... / }

    </ IDKitWidget

) Copy Copied!

Response

Upon successful completion of the World ID flow, you will receive a response object. This response object of typeISuccessResult has the following attributes.Normally, you will forward these parameters to your backend for verification.

ISuccessResult

{ "merkle_root" :

"0x1f38b57f3bdf96f05ea62fa68814871bf0ca8ce4dbe073d8497d5a6b0a53e5e0" , "nullifier_hash" :

"0x0339861e70a9bdb6b01a88c7534a3332db915d3d06511b79a5724221a6958fbe" , "proof" :

"0x063942fd7ea1616f17787d2e3374c1826ebcd2d41d2394..." , "verification_level" :

"orb" } Copy Copied!

- Name
- merkle_root
- Type
- string
- Description
- This is the hash pointer to the root of the Merkle tree that proves membership of the user's identity in the
- list of identities verified by the Orb.
- Name
- nullifier_hash
- Type
- string
- Description
- The unique identifier for this combination of user, app, and action.
- Name
- proof
- Type
- string
- Description
- The Zero-knowledge proof of the verification.
- Name
- verification_level
- Type
- "orb" | "device"
- Description
- Eitherorb
- ordevice
- . Returns the verification_level used to generate the proof.

Verifying Proofs

This section describes how to verify proofs via theDeveloper Portal API .

You should pass the proof to your backend when verifying proofs via the API. Users can manipulate information in the frontend, so the proof must be verified in a trusted environment.

Your backend should receive theproof ,merkle_root ,nullifier_hash , andverification_level returned by IDKit, as well as thesignal that was input into IDKit, and send it to theDeveloper Portal API for verification. Theaction ID should be accessible in your backend as an environment variable unless using Dynamic Actions, in which case you should pass theaction ID to the backend with the proof and signal. TheDeveloper Portal API will return a200 response if the proof is valid, and a400 response with extra error detail if the proof is invalid. After performing your own backend actions based on this result, you then pass the success or error messages back to your frontend.

pages/api/verify.ts

```
export

type

VerifyReply

= { code :

string detail ?:

string }

export

default

function

handler (req :

NextApiRequest , res :

NextApiResponse < VerifyReply

    ) { const

reqBody

= { merkle_root :

req . body .merkle_root , nullifier_hash :

req . body .nullifier_hash , proof :

req . body .proof , verification_level :

req . body .verification_level , action :

process . env . NEXT_PUBLIC_WLD_ACTION_ID , signal :

req . body .signal ??

'' ,

// if we don't have a signal, use the empty string } fetch (https://developer.worldcoin.org/api/v1/verify/ { process . env .
NEXT_PUBLIC_WLD_APP_ID } , { method :

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

'application/json' , } , body :

JSON .stringify (reqBody) , }) .then (verifyRes => { verifyRes .json () .then (wldResponse => { if ( verifyRes .status ==

200 ) { // this is where you should perform backend actions based on the verified credential // i.e. setting a user as "verified"
in a database res .status ( verifyRes .status) .send ({ code :

'success' }) } else { // return the error code and detail from the World ID /verify endpoint to our frontend res .status (
verifyRes .status) .send ({ code :
```

wldResponse .code , detail :

wldResponse .detail , }) } }) }) } Copy Copied!

Post-Verification

IfhandleVerify does not throw an error, the user will see a success state and theonSuccess callback will be called when the modal is closed. TheonSuccess callback should redirect a user to a success page, or perform any other actions you want to take after a user has been verified.

pages/index.tsx

const

onSuccess

= (result :

ISuccessResult) => { // This is where you should perform frontend actions once a user has been verified window .alert (
Successfully verified with World ID! Your nullifier hash is:

+

result .nullifier_hash) } Copy Copied!

For more information on configuration, see theIDKit andCloud API reference pages.