Using PnP Web No Modal SDK

Once you've installed and successfully initializedWeb3AuthNoModal , you can use it to authenticate your users. Further, you can use the native provider given byWeb3AuthNoModal to connect the users to the respective blockchain network.

Natively, the instance ofWeb3AuthNoModal (referred to asweb3auth in our examples) returns the following functions:

- connectTo()
-
    - Logging in the User with the given Wallet Adapter and respective Login Parameters
- provider()
-
    - Returns the native provider that can be used to make different blockchain transactions.
- connected()
-
    - Returnstrue
- orfalse
- depending on whether the web3auth instance is available or not.
- getUserInfo()
-
    - Getting the User's Information
- authenticateUser()
-
    - Getting the idToken from Web3Auth
- addChain()
-
    - Add chain config details to the connected adapter.
- switchChain()
-
    - Switch chain as per chainId already added to chain config.
- getAdapter()
-
    - Get the connected adapter instance.
- configureAdapter()
-
    - Configure the adapter instance.
- clearCache()
-
    - Clear the cache.
- addPlugin()
-
    - Add a plugin to Web3Auth.
- logout()
-
    - Logging out the User

Logging in the Userâ

connectTo()

â

To log a user in the Web3Auth Plug and Play No Modal SDK, you need to call theconnectTo() function. This function helps you customize the login process according to your own needs, by taking the following parameters:

- Table
- Function Definition

Variable Description walletName Wallet Adapter you want to use for logging in your user. It acceptsWALLET_ADAPTER_TYPE . loginParams? Login Parameters specific to your wallet adapter. connectTo < T

(walletName :

WALLET_ADAPTER_TYPE , loginParams ? :

T) :

Promise < IProvider

|

null

; export

type

WALLET_ADAPTER_TYPE

=

(typeof

WALLET_ADAPTERS) [keyof

typeof

WALLET_ADAPTERS] ; export

declare

const

WALLET_ADAPTERS :

{ OPENLOGIN :

string ; WALLET_CONNECT_V2 :

string ; TORUS_SOLANA :

string ; PHANTOM :

string ; SOLFLARE :

string ; SLOPE :

string ; TORUS_EVM :

string ; METAMASK :

string ; COINBASE :

string ; } ;

Returnsâ

connectTo < T

(walletName :

WALLET_ADAPTER_TYPE , loginParams ? :

T) :

Promise < IProvider

|

null

; On successful login, theconnectTo() function returns aIProvider instance. This instance contains the respective provider corresponding to your selected blockchain. You can use this provider to connect your user to the blockchain and make transactions.

On unsuccessful login, this function will return anull value.

Exampleâ

tip You can use theconnectTo() function to connect to the custom authentication verifiers you might have deployed on the Web3Auth Dashboard, as well as the default ones that Web3Auth provides. For the default verifiers, you don't need to provide any additional parameters, just pass on the login provider type. * Google * Auth0 * JWT * Facebook * Discord *

Email Passwordless * SMS Passwordless * Twitter * Reddit * Twitch * Apple * GitHub * LinkedIn

import

{

WALLET_ADAPTERS

}

from

"@web3auth/base" ; // inside your async function with on click handler const web3authProvider =

await web3auth . connectTo (WALLET_ADAPTERS . OPENLOGIN ,

{ loginProvider :

"google" , }) ; const web3authProvider =

await web3auth . connectTo (WALLET_ADAPTERS . OPENLOGIN ,

{ loginProvider :

"jwt" , extraLoginOptions :

{ verifierIdField :

"sub" ,

// same as your JWT Verifier ID domain :

"https://YOUR-APPLICATION-DOMAIN" ,

// your service provider domain, e.g. Auth0 } , }) ; // Login using JWT, either obtained from Firebase, Okta, Auth0 or bring your own JWT. const web3authProvider =

await web3auth . connectTo (WALLET_ADAPTERS . OPENLOGIN ,

{ loginProvider :

"jwt" , extraLoginOptions :

{ id_token :

"idToken" ,

// in JWT Format verifierIdField :

"sub" ,

// same as your JWT Verifier ID } , }) ; import

{

WALLET_ADAPTERS

}

from

"@web3auth/base" ;

const web3authProvider =

await web3auth . connectTo (WALLET_ADAPTERS . OPENLOGIN ,

{ loginProvider :

"facebook" , }) ; import

{

WALLET_ADAPTERS

```
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"email_passwordless" , extraLoginOptions :
{ login_hint :
"hello@web3auth.io" ,
// email to send the OTP to } , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"sms_passwordless" , extraLoginOptions :
{ login_hint :
"+65-XXXXXXX" , } , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"discord" , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
```

```
{ loginProvider :
"twitter" , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"reddit" , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"twitch" , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
"apple" , } ) ; import
{
WALLET_ADAPTERS
}
from
"@web3auth/base" ;
const web3authProvider =
await web3auth . connectTo ( WALLET_ADAPTERS . OPENLOGIN ,
{ loginProvider :
```

"github" , }) ; import

{

WALLET_ADAPTERS

}

from

"@web3auth/base" ;

const web3authProvider =

await web3auth . connectTo (WALLET_ADAPTERS . OPENLOGIN ,

{ loginProvider :

"linkedin" , }) ; tip Read more about connecting your users with the selected blockchain in the Providers SDK Reference .

walletName

â

- Table
- Type Declaration

Variable Value Description WALLET_ADAPTERS.OPENLOGIN "openlogin" Helps you connect to the Openlogin Adapter WALLET_ADAPTERS.WALLET_CONNECT_V1 "wallet-connect-v1" (deprecated) Helps you connect to the Wallet Connect V1 Adapter. Currently deprecated. WALLET_ADAPTERS.WALLET_CONNECT_V2 "wallet-connect-v2" Helps you connect to the Wallet Connect V2 Adapter WALLET_ADAPTERS.TORUS_SOLANA "torus-solana" Helps you connect to the Torus Solana Wallet Adapter WALLET_ADAPTERS.PHANTOM "phantom" Helps you connect to the Phantom Wallet Adapter WALLET_ADAPTERS.SOLFLARE "solflare" Helps you connect to the Solflare Wallet Adapter WALLET_ADAPTERS.SLOPE "slope" Helps you connect to the Slope Wallet Adapter WALLET_ADAPTERS.TORUS_EVM "torus-evm" Helps you connect to the Torus EVM Wallet Adapter WALLET_ADAPTERS.METAMASK "metamask" Helps you connect to the Metamask Wallet Adapter WALLET_ADAPTERS.COINBASE "coinbase" Helps you connect to the Coinbase Wallet Adapter export

declare

type

WALLET_ADAPTER_TYPE

=

typeof

WALLET_ADAPTERS [keyof

typeof

WALLET_ADAPTERS] ;

export

declare

const

WALLET_ADAPTERS :

{ readonly

OPENLOGIN :

"openlogin" ; readonly

WALLET_CONNECT_V1 :

"wallet-connect-v1" ; readonly

WALLET_CONNECT_V2 :

"wallet-connect-v2" ; readonly

TORUS_SOLANA :

"torus-solana" ; readonly

PHANTOM :

"phantom" ; readonly

SOLFLARE :

"solflare" ; readonly

SLOPE :

"slope" ; readonly

TORUS_EVM :

"torus-evm" ; readonly

METAMASK :

"metamask" ; readonly

COINBASE :

"coinbase" ; } ;

loginParams

â

TheloginParams are specific for each and every function. Please refer to theAdapters SDK Reference to know more about the login parameters specific to each wallet adapter.

Foropenlogin-adapter which enables social logins, you can refer to the followingloginParams .

- Table
- Type Declaration

LoginSettings

â

Variable Description loginProvider Sets the OAuth login method to be used. You can use any of the valid loginProvider from the supported list.

The list of available options:google ,facebook ,reddit ,discord ,twitch ,apple ,line ,github ,kakao ,linkedin ,twitter ,weibo ,wechat ,email_passwordless ,sms_passwordless ,jwt mfaLevel? You can set themfaLevel to customize when mfa screen should be shown to user. It currently accepts 4 values: *'default' * Setting mfa level todefault * will present mfa screen to user on every third login. * 'optional' * Setting mfa level todefault * will present mfa screen to user on every login but user can skip it. * 'mandatory' * Setting mfa level tomandatory * will make it mandatory for user to setup mfa after login. * 'none' * Setting mfa level tonone * will make the user skip the mfa setup screen extraLoginOptions? This can be used to pass standard oauth login options to loginProvider. For ex: you will have to passlogin_hint as user's email anddomain as your app domain inextraLoginOptions while usingemail_passwordless loginProvider. It acceptsExtraLoginOptions as a value. dappShare? Custom logins can get a dApp share returned to them post successful login. This is useful if the dapps want to use this share to allow users to login seamlessly.dappShare is a 24 word seed phrase. It acceptsstring as a value. curve? This curve will be used to determine the public key encoded in the jwt token which returned ingetUserInfo function after user login. You can use that public key from jwt token as a unique user identifier in your backend.* 'secp256k1' * : secp256k1 based pub key is added as a wallet public key in jwt token to use. * 'ed25519' * : ed25519 based pub key is added as a wallet public key in jwt token to use.

Note: This parameter won't change format of private key returned by openlogin. Private key returned by openlogin is alwayssecp256k1 . It acceptsSUPPORTED_KEY_CURVES_TYPE as a value. appState? Any custom state you wish to pass along. This will be returned to you post redirect. Use this to store data that you want to be available to the dApp after login. It acceptsstring as a value. redirectUrl? App's URL where user will be redirected after login. Register this URL in thedeveloper dashboard , else initialization will give error. It acceptsstring as a value. export

type

LoginSettings

=

Partial < LoginParams

&

Partial < BaseRedirectParams

;

export

type

LoginParams

=

BaseRedirectParams

&

{ /* * loginProvider sets the oauth login method to be used. * You can use any of the valid loginProvider from the supported list. */ loginProvider :

LOGIN_PROVIDER_TYPE

|

CUSTOM_LOGIN_PROVIDER_TYPE ; /* * You can set the mfaLevel to customize when mfa screen should be shown to user. * It currently accepts 4 values:- * - 'default': Setting mfa level to default will present mfa screen to user on every third login. * - 'optional': Setting mfa level to default will present mfa screen to user on every login but user can skip it. * - 'mandatory': Setting mfa level to mandatory will make it mandatory for user to setup mfa after login. * - 'none': Setting mfa level to none will make the user skip the mfa setup screen * * Defaults to default * @defaultValue default */ mfaLevel ? :

MfaLevelType ; /* * extraLoginOptions can be used to pass standard oauth login options to * loginProvider. * * For ex: you will have to pass login_hint as user's email and domain * as your app domain in extraLoginOptions while using email_passwordless * loginProvider */ extraLoginOptions ? :

ExtraLoginOptions ; /* * Custom Logins can get a dapp share returned to them post successful login. * This is useful if the dapps want to use this share to allow users to login seamlessly * dappShare is a 24 word seed phrase */ dappShare ? :

string ; /* * This curve will be used to determine the public key encoded in the jwt token which returned in getUserInfo function after user login. * You can use that public key from jwt token as a unique user identifier in your backend. * * - 'secp256k1': secp256k1 based pub key is added as a wallet public key in jwt token to use. * - 'ed25519': ed25519 based pub key is added as a wallet public key in jwt token to use. * * Note: This parameter won't change format of private key returned by openlogin. Private key returned * by openlogin is always secp256k1. As of now you have to convert it to 'ed25519' if you want. * You can use @toruslabs/openlogin-ed25519 npm package for this purpose. * * * @defaultValue secp256k1 */ curve ? :

SUPPORTED_KEY_CURVES_TYPE ; } ;

export

type

BaseRedirectParams

=

{ /* * redirectUrl is the dapp's url where user will be redirected after login. * * @remarks * Register this url at {@link "https://dashboard.web3auth.io"| developer dashboard} * else initialization will give error. */ redirectUrl ? :

string ; /* * Any custom state you wish to pass along. This will be returned to you post redirect. * Use this to store data that you want to be available to the dapp after login. */ appState ? :

string ; } ;

/* * {@label loginProviderType}/ export

type

LOGIN_PROVIDER_TYPE

=

(typeof

LOGIN_PROVIDER) [keyof

typeof

LOGIN_PROVIDER] ; export

declare

const

LOGIN_PROVIDER :

{ readonly

GOOGLE :

"google" ; readonly

FACEBOOK :

"facebook" ; readonly

REDDIT :

"reddit" ; readonly

DISCORD :

"discord" ; readonly

TWITCH :

"twitch" ; readonly

APPLE :

"apple" ; readonly

LINE :

"line" ; readonly

GITHUB :

"github" ; readonly

KAKAO :

"kakao" ; readonly

LINKEDIN :

"linkedin" ; readonly

TWITTER :

"twitter" ; readonly

WEIBO :

"weibo" ; readonly

WECHAT :

"wechat" ; readonly

EMAIL_PASSWORDLESS :

"email_passwordless" ; readonly

SMS_PASSWORDLESS :

"sms_passwordless" ; readonly

WEBAUTHN :

"webauthn" ; readonly

JWT :

"jwt" ; } ;

export

type

CUSTOM_LOGIN_PROVIDER_TYPE

=

string

&

{ toString ? :

(radix ? :

number)

=>

string ; } ;

/* * *{@label MfaLevelType}* /

export

type

MfaLevelType

=

(typeof

MFA_LEVELS) [keyof

typeof

MFA_LEVELS] ; export

declare

const

MFA_LEVELS :

{ readonly

DEFAULT :

"default" ; readonly

OPTIONAL :

"optional" ; readonly

MANDATORY :

"mandatory" ; readonly

NONE :

"none" ; } ;

/* * {@label SUPPORTED_KEY_CURVES_TYPE}/

export

type

SUPPORTED_KEY_CURVES_TYPE

=

(typeof

SUPPORTED_KEY_CURVES) [keyof

typeof

SUPPORTED_KEY_CURVES] ; export

declare

const

SUPPORTED_KEY_CURVES :

{ readonly

SECP256K1 :

"secp256k1" ; readonly

ED25519 :

"ed25519" ; } ; Further, for Custom Authentication,ExtraLoginOptions can be used to pass the standard OAuth login options.

- Table
- Interface

ExtraLoginOptions

â

Parameter Description additionalParams? Additional params in[key: string] format for OAuth login, use id_token(JWT) to authenticate with web3auth. domain? Your custom authentication domain instring format. For example, if you are using Auth0, it can be example.au.auth0.com. client_id? Client id instring format, provided by your login provider used for custom verifier. leeway? The value used to account for clock skew in JWT expirations. The value is in the seconds, and ideally should no more than 60 seconds or 120 seconds at max. It takesstring as a value. verifierIdField? The field in JWT token which maps to verifier id. Please make sure you selected correct JWT verifier id in the developer dashboard. It takesstring as a value. isVerifierIdCaseSensitive? boolean to confirm Whether the verifier id field is case sensitive or not. display? Allows developers the configure the display of UI. It takesstring as a value. prompt? Prompt shown to the user during authentication process. It takesstring as a value. max_age? Max time allowed without reauthentication. If the last time user authenticated is greater than this value, then user must reauthenticate. It takesstring as a value. ui_locales? The space separated list of language tags, ordered by preference. For instancefr-CA fr en . id_token_hint? It denotes the previously issued ID token. It takesstring as a value. id_token? JWT (ID Token) to be passed for login. login_hint? It is used to send the user's email address during Email Passwordless login. It takesstring as a value. acr_values? acc_values scope? The default scope to be used on authentication requests. The defaultScope defined in the Auth0Client is included along with this scope. It takesstring as a value. audience? The audience, presented as the aud claim in the access token, defines the intended consumer of the token. It takesstring as a value. connection? The name of the connection configured for your application. If null, it will redirect to the Auth0 Login Page and show the Login Widget. It takesstring as a value. redirect_uri? It can be used to specify the default url, where your custom jwt verifier can redirect your browser to with the result. If you are using Auth0, it must be whitelisted in the Allowed Callback URLs in your Auth0's application. export

interface

ExtraLoginOptions

extends

BaseLoginOptions

{ /* * *Your Auth0 account domain such as* *'example.auth0.com', * 'example.eu.auth0.com'* *or , 'example.mycompany.com' * (when using* *custom domains)* / domain ? :

string ; /* * *The Client ID found on your Application settings page*/ client_id ? :

string ; /* * *The default URL where Auth0 will redirect your browser to with * the authentication result. It must be whitelisted in * the "Allowed Callback URLs" field in your Auth0 Application's * settings. If not provided here, it should be provided in the other * methods that provide authentication.* / redirect_uri ? :

string ; /* * *The value in seconds used to account for clock skew in JWT expirations. * Typically, this value is no more than a minute or two at maximum. * Defaults to 60s.* / leeway ? :

number ; /* * *The field in jwt token which maps to verifier id*/ verifierIdField ? :

string ; /* * *Whether the verifier id field is case sensitive * @defaultValue true*/ isVerifierIdCaseSensitive ? :

boolean ; } export

interface

BaseLoginOptions

{ /* * *If you need to send custom parameters to the Authorization Server, * make sure to use the original parameter name.*/ [key :

string] :

unknown ; /* * - *'page': displays the UI with a full page view * -'popup': displays the UI with a popup window * -'touch': displays the UI in a way that leverages a touch interface * - 'wap': displays the UI with a "feature phone" type interface*/ display ? :

"page"

|

"popup"

|

"touch"

|

"wap"

|

string ; /* * - *'none': do not prompt user for login or consent on re-authentication * -'login': prompt user for re-authentication * - 'consent': prompt user for consent before processing request * -'select_account': prompt user to select an account*/ prompt ? :

"none"

|

"login"

|

"consent"

|

"select_account"

|

string ; /* * *Maximum allowable elapsed time (in seconds) since authentication. * If the last time the user authenticated is greater than this value, * the user must be re-authenticated.* / max_age ? :

string

|

number ; /* * *The space-separated list of language tags, ordered by preference. * For example:'fr-CA fr en'.* / ui_locales ? :

string ; /* * *Previously issued ID Token.*/ id_token_hint ? :

string ; /* * *The user's email address or other identifier. When your app knows * which user is trying to authenticate, you can provide this parameter * to pre-fill the email box or select the right session for sign-in. * * This currently only affects the classic Lock experience.* / login_hint ? :

string ; id_token ? :

string ; acr_values ? :

string ; /* * *The default scope to be used on authentication requests. * The defaultScope defined in the Auth0Client is included * along with this scope* / scope ? :

string ; /* * *The default audience to be used for requesting API access.*/ audience ? :

string ; /* * *The name of the connection configured for your application. * If null, it will redirect to the Auth0 Login Page and show * the Login Widget.* / connection ? :

string ; }

Get a native providerâ

provider()

Returns the native provider that can be used to make different blockchain transactions.

Returnsâ

get

provider ( ) :

IProvider

|

null ;

Get connected statusâ

connected()

Returnstrue orfalse depending on whether the web3auth instance is available or not.

Returnsâ

get

connected ( ) : boolean ;

Get User's Informationâ

getUserInfo()

â

You can get the information about connected user by callinggetUserInfo() function.

note This function will only return information based on the connected adapter. These details are not stored anywhere and are fetched from the adapter during the connection and remain in the session. await web3auth . getUserInfo ( ) ;

Returnsâ

- Table
- Type Declarations

Variable Type Description email string Email of the connected user name string Name of the connected user profileImage string Profile image of the connected user aggregateVerifier string Details of the aggregate verifier - if present verifier string Details of the verifier (verifier type, ie.torus ,metamask ,openlogin etc.) verifierId string Verifier ID - if custom authentication is enabled, it will show the verifier identifier you set on dashboard typeOfLogin string Type of login done by the user (likegoogle ,facebook ,twitter ,github , etc.) dappShare string If you are using a Custom Verifier, you can get a dapp share after successful login. (dappShare is a 24 word seed phrase). This share can act as a replacement to your user's device share. idToken string JWT token (Note: This is issued by Web3Auth and is not bound to your OAuth provider) oAuthIdToken string JWT token (Note: This is issued by your OAuth provider and will be issued only for custom verifiers.) oAuthAccessToken string Access token (Note: This is issued by your OAuth provider and will be issued only for custom verifiers.) getUserInfo(): Promise>;

declare

type

UserInfo

=

OpenloginUserInfo ;

declare

type

OpenloginUserInfo

=

{ email :

string ; name :

string ; profileImage :

string ; aggregateVerifier :

string ; verifier :

string ; verifierId :

string ; typeOfLogin :

LOGIN_PROVIDER_TYPE

|

CUSTOM_LOGIN_PROVIDER_TYPE ; dappShare ? :

string ; idToken ? :

string ; oAuthIdToken ? :

string ; oAuthAccessToken ? :

string ; } ;

Sample Responseâ

{ "email": "john@gmail.com", "name": "John Dash", "profileImage": "https://lh3.googleusercontent.com/a/Ajjjsdsmdjmnm...", "aggregateVerifier": "tkey-google-lrc", "verifier": "torus", "verifierId": "john@gmail.com", "typeOfLogin": "google", "dappShare": "<24 words seed phrase>", // will be sent only incase of custom verifiers "idToken": "", "oAuthIdToken": "", // will be sent only incase of custom verifiers "oAuthAccessToken": "" // will be sent only incase of custom verifiers }

Get idTokenâ

authenticateUser()

â

You can get theidToken from Web3Auth by callingauthenticateUser() function.

note This function will only return information based on the connected adapter. These details are not stored anywhere and are fetched from the adapter during the connection and remain in the session. await web3auth . authenticateUser ( ) ;

Returnsâ

- Social Login
- External Wallet
- Table
- Type Declarations

Parameter Type Description iat number The "iat" (issued at) claim identifies the time at which the JWT was issued. aud string The "aud" (audience) claim identifies the recipients that the JWT is intended for. (Here, it's the dappclient_id) iss string The "iss" (issuer) claim identifies who issued the JWT. (Here, it's Web3Authhttps://api.openlogin.com/) email string The email address of the user (optional) name string The name of the user (optional) profileImage string The profile image of the user (optional) verifier string Web3auth's verifier used while user login verifierId string Unique user id given by OAuth login provider aggregateVerifier string Name of the verifier if you are using a single id verifier (aggregateVerifier) (optional) exp number The "exp" (expiration time) claim identifies the expiration time on or after which theJWT MUST NOT be accepted for processing . wallets array list of wallets for which this token is issued:* curve * "secp256k1" (default) or "ed25519" * You can specify which curve you want use for the encoded public key in the login parameters * type * "web3auth_key" incase of social logins * public_key * compressed public key derived based on the specified curve authenticateUser(): Promise

export

type

UserAuthInfo

=

{ idToken :

string

} ;

Sample Responseâ

{ "iat": 1655835494, "aud": "BCtbnOamqh0cJFEUYA0NB5YkvBECZ3HLZsKfvSRBvew2EiiKW3UxpyQASSR0artjQkiUOCHeZ_ZeygXpYpxZjOs", "iss": "https://api.openlogin.com/", "email": "xyz@xyz.com", "name": "John Doe", "profileImage": "https://lh3.googleusercontent.com/a/AATXAJx3lnGmHiM4K97uLo9Rb0AxOceH-dQCBSRqGbck=s96-c", "verifier": "torus", "verifierId": "xyz@xyz.com", "aggregateVerifier": "tkey-google-lrc", "exp": 1655921894, "wallets": [{ "public_key": "035143318b83eb5d31611f8c03582ab1200494f66f5e11a67c34f5581f48c1b70b", "type": "web3auth_key", "curve": "secp256k1" }] } * Table * Type Declarations

Parameter Type Description iat number The "iat" (issued at) claim identifies the time at which the JWT was issued. aud string The "audience" claim identifies the recipients that the JWT is intended for. (Here, it'swebsite's url) iss string The "issuer" claim identifies who issued the JWT. Here, issuer could betorus-evm ,torus-solana ,metamask ,phantom ,walletconnect-v1 ,coinbase ,slope ,solflare wallets array list of wallets for which this token is issued:* address * : Wallet public address * type * Network Type such as "ethereum", "solana" or "starkware" incase of external wallets authenticateUser(): Promise

export

type

UserAuthInfo

=

{ idToken :

string

} ;

Sample Responseâ

{ "iat": 1661158877, "issuer": "", "audience": "https://requesting.website", "wallets": [{ "address": "0x809d4310d578649d8539e718030ee11e603ee8f3", "type": "ethereum" }], "exp": 1661245277 }

Add Chainâ

addChain()

â

To add a chain config to a connected adapter, you need to calladdChain() function. This function helps you add the chain config by taking the following parameter:

- Table
- Function Definition

Variable Description chainConfig CustomChainConfig for the chain you want to add. async

addChain (chainConfig :

CustomChainConfig) :

Promise < void

{ if

(this . status

!==

ADAPTER_STATUS . CONNECTED

||

! this . connectedAdapterName)

throw

WalletLoginError . notConnectedError (No wallet is connected) ; return

this . walletAdapters [this . connectedAdapterName] . addChain (chainConfig) ; }

export

type

CustomChainConfig

=

{ chainNamespace :

ChainNamespaceType ; /* * *The chain id of the chain*/ chainId :

string ; /* * *RPC target Url for the chain*/ rpcTarget :

string ; /* * *Display Name for the chain*/ displayName :

string ; /* * *Url of the block explorer*/ blockExplorerUrl :

string ; /* * *Default currency ticker of the network (e.g: ETH)*/ ticker :

string ; /* * *Name for currency ticker (e.g:Ethereum)* / tickerName :

string ; /* * *Number of decimals for the currency ticker (e.g: 18)*/ decimals ? :

number ; /* * *Logo for the chain*/ logo :

string ; /* * *Whether the network is testnet or not*/ isTestnet ? :

boolean ; } ; await web3auth . addChain ({ chainId :

"0xaa36a7" , displayName :

"Ethereum Sepolia" , chainNamespace :

CHAIN_NAMESPACES . EIP155 , tickerName :

"Sepolia" , ticker :

"ETH" , decimals :

18 , rpcTarget :

"https://rpc.ankr.com/eth_sepolia" , blockExplorerUrl :

"https://sepolia.etherscan.io" , logo :

"https://images.toruswallet.io/eth.svg" , isTestnet :

true , }) ;

Switch Chainâ

switchChain()

â

To switch the Chain to the added chain config, you need to callswitchChain() function. This function takes the following parameter:

- Table
- Function Definition

Variable Description { chainId: "0xaa36a7" } chainId of the added chain config, e.g{ chainId: "0xaa36a7" } async

switchChain (params :

{ chainId :

string

}) :

Promise < void

{ if

(this . status

!==

ADAPTER_STATUS . CONNECTED

||

! this . connectedAdapterName)

throw

WalletLoginError . notConnectedError (No wallet is connected) ; return

this . walletAdapters [this . connectedAdapterName] . switchChain (params) ; } await web3auth . switchChain ({

chainId :

"0xaa36a7"

}) ;

Get Adapterâ

Get the adapter connected by passing the name of the adapter.

export

declare

const

WALLET_ADAPTERS :

{ OPENLOGIN :

string ; WALLET_CONNECT_V2 :

string ; TORUS_SOLANA :

string ; PHANTOM :

string ; SOLFLARE :

string ; SLOPE :

string ; TORUS_EVM :

string ; METAMASK :

string ; COINBASE :

string ; } ; getAdapter (adapterName :

WALLET_ADAPTER_TYPE) :

IAdapter < unknown

|

null ;

Configure Adapterâ

Configure an adapter instance to Web3Auth.

configureAdapter (adapter :

IAdapter < unknown

) :

Web3AuthNoModal ;

Clear Cacheâ

Clear the cache.

clearCache ( ) :

void ;

Logging out the Userâ

web3auth.logout()

â

You can disconnect from connected wallet usinglogout function.

await web3auth . logout ( ) ;

Type Referenceâ

logout (options ? :

{ cleanup :

boolean ;

}) :

Promise < void

;

Connecting to a Blockchainâ

For Connecting to Blockchain and making RPC calls within your dApp, you can use theIProvider instance returned while logging in the user. This instance gives you the respective provider for your selection of blockchain. This provider can be used to interact with the connected chain using exposed functions within the provider.

Web3Auth is chain agnostic, ie. be it any chain you're connecting to, you can use Web3Auth to connect to it. You can use theEVM orSolana provider, that contain the native functions to connect to the blockchain or use the private key directly to connecting to the respective blockchain.

Currently web3auth supports providers for both EVM and Solana chains. For other chains, one can easily get the private key from the base provider from Web3Auth.

tip Checkout theProviders SDK Reference to learn more.

- For EVM based Chains@web3auth/ethereum-provider
- For Solana Blockchain@web3auth/solana-provider
- Default Provider for all other Blockchains

Fetching the Connected Adapterâ

You can configure a lot ofadapters in the Web3Auth instance. If you have installed one of the adapters, you can fetch theconnectedAdapterName variable to get the currently used adapter.

// Initialized Web3Auth instance

// User is logged in using one of the adapters (metamask, torus-evm, etc.)

const adapterName = web3auth . connectedAdapterName ; // contains the currently used adapter name (metamask, torus-evm, etc.) Edit this page Previous Initialize Next Whitelabel