Using PnP Web Modal SDK

Once you've installed and successfully initialized Web3Auth, you can use it to authenticate your users. Further, you can use the native provider given by Web3Auth to connect the users to the respective blockchain network.

Natively, the instance ofWeb3Auth (referred to asweb3auth in our examples) returns the following functions:

- connect()
  - Showing the Modal and Logging in the User
- 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

connect()

â

You can show the modal in the browser by callingconnect() function onweb3auth instance. Your user can choose their preferred login method and access your application.

await web3auth . connect ( ) ;

Returnsâ

connect ( ) :

Promise < IProvider

|

null

; On successful login, theconnect() 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.

tip Read more about connecting your users with the selected blockchain in the Providers SDK Reference .

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"

}) ;

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