Custom Authentication with PnP Web No Modal SDK

Custom Authentication is a way to authenticate users with your custom authentication service. For example, while authenticating with Google, you can use your own Google Client ID and Dashboard to authenticate users directly or use aggregate services like Auth0, Firebase, AWS Cognito etc. Additionally, you can make your own JWT token authentication system and pass over the ID Token to Web3Auth to generate a private key for them.

For enabling this, you needCreate a Verifier from theCustom Auth section of the

Web3Auth Developer Dashboard with your desired configuration.

note This is a paid feature and the minimumpricing plan to use this SDK in a production environment is theGrowth Plan . You can use this feature in the development environment for free. tip If you want to know more about setting up a verifier and how to use it, please refer to theCustom Authentication Documentation .

Installing Openlogin Adapterâ

Social logins in Web3Auth are enabled by theopenlogin-adapter . Natively, it is already present and preconfigured within the Plug and Play SDK, but for custom configurations, you need to install the adapter package.

- npm
- Yarn
- pnpm

npm install --save @web3auth/openlogin-adapter yarn add @web3auth/openlogin-adapter pnpm add @web3auth/openlogin-adapter

Configuring Openlogin Adapterâ

While instantiating the Openlogin Adapter, you can pass some configuration objects to the constructor. One of these configurations is theadapterSettings configuration which enables you to make changes in the adapter, enabling you to do things like Whitelabeling and Custom Authentication among other things.

tip Checkout theopenlogin-adapter SDK Reference for more details on different configurations you can pass for customizations.

loginConfig

â

TheloginConfig parameter ofadapterSettings inopenlogin-adapter contains the following properties:

- Table
- Type Declarations

loginConfig: { "identifier of social login": { params } }

params

Parameter Description verifier The name of the verifier that you have registered on the Web3Auth Dashboard. It's a mandatory field, and acceptsstring as a value. typeOfLogin Type of login of this verifier, this value will affect the login flow that is adapted. For example, if you choosegoogle , a Google sign-in flow will be used. If you choosejwt , you should be providing your own JWT token, no sign-in flow will be presented. It's a mandatory field, and acceptsTypeOfLogin as a value. clientId Client id provided by your login provider used for custom verifier. e.g. Google's Client ID or Web3Auth's client Id if using 'jwt' as TypeOfLogin. It's a mandatory field, and acceptsstring as a value. name? Display name for the verifier. If null, the default name is used. It acceptsstring as a value. description? Description for the button. If provided, it renders as a full length button. else, icon button. It acceptsstring as a value. verifierSubIdentifier? The field in JWT token which maps to verifier id. Please make sure you selected correct JWT verifier id in the developer dashboard. It acceptsstring as a value. logoHover? Logo to be shown on mouse hover. It acceptsstring as a value. logoLight? Light logo for dark background. It acceptsstring as a value. logoDark? Dark logo for light background. It acceptsstring as a value. mainOption? Show login button on the main list. It acceptsboolean as a value. Default value is false. showOnModal? Whether to show the login button on modal or not. Default value is true. showOnDesktop? Whether to show the login button on desktop. Default value is true. showOnMobile? Whether to show the login button on mobile. Default value is true. showOnSocialBackupFactor? If we are using social logins as a backup factor, then this option will be used to show the type of social login on the social backup login screen. jwtParameters? Custom jwt parameters to configure the login. Useful for Auth0 configuration. It acceptsJwtParameters as a value. export

type

LoginConfig

=

Record < string , { verifier :

string ; /* * *The type of login. Refer to enum**LOGIN_TYPE* / typeOfLogin :

TypeOfLogin ; /* * *Display Name. If not provided, we use the default for openlogin app*/ name ? :

string ; /* * *Description for button. If provided, it renders as a full length button. else, icon button*/ description ? :

string ; /* * *Custom client_id. If not provided, we use the default for openlogin app*/ clientId ? :

string ; verifierSubIdentifier ? :

string ; /* * *Logo to be shown on mouse hover. If not provided, we use the default for openlogin app*/ logoHover ? :

string ; /* * *Logo to be shown on dark background (dark theme). If not provided, we use the default for openlogin app*/ logoLight ? :

string ; /* * *Logo to be shown on light background (light theme). If not provided, we use the default for openlogin app*/ logoDark ? :

string ; /* * *Show login button on the main list*/ mainOption ? :

boolean ; /* * *Whether to show the login button on modal or not*/ showOnModal ? :

boolean ; /* * *Whether to show the login button on desktop*/ showOnDesktop ? :

boolean ; /* * *Whether to show the login button on mobile*/ showOnMobile ? :

boolean ; /* * *If we are using social logins as a backup factor, * then this option will be used to show the type of social login * on the social backup login screen.* / showOnSocialBackupFactor ? :

boolean ; /* * *Custom jwt parameters to configure the login. Useful for Auth0 configuration*/ jwtParameters ? :

JwtParameters ; }

;

export

type

TypeOfLogin

= |

"google" |

"facebook" |

"reddit" |

"discord" |

"twitch" |

"apple" |

"github" |

"linkedin" |

"twitter" |

"weibo" |

"line" |

"email_password" |

"passwordless" |

"jwt" |

"webauthn" ;

Logging in through your Custom JWT Token â

While using the Web3Auth Plug and Play No Modal SDK, you have the option to use theconnectTo function, which enables you to customize the login process according to the parameters you have for your custom authentication service.

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 ; } ; tip Know more about theconnectTo function in the Usage SDK Reference important It is mandatory to pass theidToken parameter in theextraLoginOptions object when logging in through your custom JWT token. Further, to enable Custom Authentication, theloginParams parameter takes in another object calledextraLoginOptions which contains the following properties:

- 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 ; }

Exampleâ

Since we're using the@web3auth/no-modal , ie. the Plug and Play No Modal SDK, theloginConfig can include custom JWT-

based authentication as well. This way, we can use any of our preferred login providers and further setup their configs while logging the user in and passing over theextraLoginOptions in theconnectTo function.

import

{

OpenloginAdapter

}

from

"@web3auth/openlogin-adapter" ;

const openloginAdapter =

new

OpenloginAdapter ({ adapterSettings :

{ clientId ,

//Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code network :

"sapphire_mainnet" ,

// Optional - Provide only if you haven't provided it in the Web3Auth Instantiation Code uxMode :

"popup" , loginConfig :

{ jwt :

{ verifier :

"YOUR-VERIFIER-NAME-ON-WEB3AUTH-DASHBOARD" , typeOfLogin :

"jwt" , clientId :

"YOUR-CLIENTID-FROM-LOGIN-PROVIDER" , } , } , } , privateKeyProvider , }) ;

web3auth . configureAdapter (openloginAdapter) ; * 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" , } ) ;
```
Edit this page Previous Whitelabel Next Multi Factor Authentication