Using PnP iOS SDK

Logging in a Userâ

login()

â

Trigger login flow will navigate the user to a browser model allowing the user to login into the service. You can pass in the supported providers to the login method for specific social logins such as Google, Apple, Facebook, etc., and do whitelabel login.Web3Auth.login() takes inW3ALoginParams as a required input.

Argumentsâ

W3ALoginParams

- Table
- Struct

Parameter Description loginProvider It sets the OAuth login method to be used. You can use any of the supported values areGOOGLE ,FACEBOOK ,REDDIT ,DISCORD ,TWITCH ,APPLE ,LINE ,GITHUB ,KAKAO ,LINKEDIN ,TWITTER ,WEIBO ,WECHAT ,EMAIL_PASSWORDLESS . extraLoginOptions? It can be used to set the OAuth login options for correspondingloginProvider . For instance, you'll need to pass user's email address as. Default value for the field isnil , and it acceptsExtraLoginOptions as a value. redirectUrl? Url where user will be redirected after successfull login. By default user will be redirected to same page where login will be initiated. Default value for the field isnil , and acceptsURL as a value. appState? It can be used to keep track of the app state when user will be redirected to app after login. Default isnil , and acceptsString as a value. mfaLevel? Customize the MFA screen shown to the user during OAuth authentication. Default value for field isMFALevel.DEFAULT , which shows MFA screen every 3rd login. It acceptsMFALevel as a value. dappShare? Custom verifier 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. It acceptsString as a value. curve? It will be used to determine the public key encoded in the jwt token which returned ingetUserInfo function after user login. This parameter won't change format of private key returned by We3Auth. Private key returned bygetPrivKey is always secp256k1. To get the ed25519 key you can usegetEd25519PrivKey method. The default value isSUPPORTED_KEY_CURVES.SECP256K1 . public

struct

W3ALoginParams :

Codable

{ public

init ( )

{ loginProvider =

nil dappShare =

nil extraLoginOptions =

nil redirectUrl =

nil appState =

nil mfaLevel =

nil curve =

. SECP256K1 }

let loginProvider :

String ? var dappShare :

String ? let extraLoginOptions :

ExtraLoginOptions ? let redirectUrl :

String ? let appState :

String ? let mfaLevel :

MFALevel ? let curve :

SUPPORTED_KEY_CURVES }

getPrivkey()

â

Use getPrivkey() to get the private key of the user. The method returns an EVM compatible private key which can be used to sign transactions on EVM compatible chains.

getEd25519PrivKey()

â

Use getEd25519PrivKey() to get the Ed25519 private key of the user. This private key can be used to sign transactions on Solana.

getUserInfo()

â

Use getUserInfo() to get the user info of the user.

UserInfo Responseâ

{ "userInfo": { "email": "w3a-heroes@web3auth.com", "name": "Web3Auth Heroes", "profileImage": "https://lh3.googleusercontent.com/a/Ajjjsdsmdjmnm...", "verifier": "torus", "verifierId": "w3a-heroes@web3auth.com", "typeOfLogin": "google", "aggregateVerifier": "w3a-google-sapphire", "dappShare": "", // 24 words of 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 "isMfaEnabled": false // Returns whether the user has enabled MFA or not } } * Google * Facebook * Discord * Twitch * Email Passwordless * JWT

Usage func

login (provider :

Web3AuthProvider)

{ Task

{ do

{ let result =

try

await

Web3Auth ( ) . login (W3ALoginParams (loginProvider : provider)) // Perform action upon success }

catch

{ print ("Error") } } } login (provider :

. GOOGLE) Usage func

login (provider :

Web3AuthProvider)

{ Task

{ do

{ let result =

try

await

```
Web3Auth ( ) . login ( W3ALoginParams ( loginProvider : provider ) ) // Perform action upon success }
catch
{ print ( "Error" ) } } } login ( provider :
. FACEBOOK ) Usage func
login ( provider :
Web3AuthProvider )
{ Task
{ do
{ let result =
try
await
Web3Auth ( ) . login ( W3ALoginParams ( loginProvider : provider ) ) // Perform action upon success }
catch
{ print ( "Error" ) } } } login ( provider :
. DISCORD ) Usage func
login ( provider :
Web3AuthProvider )
{ Task
{ do
{ let result =
try
await
Web3Auth ( ) . login ( W3ALoginParams ( loginProvider : provider ) ) // Perform action upon success }
catch
{ print ( "Error" ) } } } login ( provider :
. TWITCH ) Usage func
login ( provider :
Web3AuthProvider )
{ Task
{ do
{ let result =
try
await
Web3Auth ( ) . login ( W3ALoginParams ( loginProvider : provider ) ) // Perform action upon success }
catch
{ print ( "Error" ) } } } login ( provider :
. EMAIL_PASSWORDLESS ) Usage func
```

login (provider :

Web3AuthProvider)

{ Task

{ do

{ let result =

try

await

Web3Auth ( ) . login (W3ALoginParams (selectedLoginProvider , extraLoginOptions :

. init (domain : "your-domain" , id_token :

"your_jwt_token")))

// Perform action upon success }

catch

{ print ("Error") } } } login (provider :

. JWT)

Session Managementâ

The Session Management feature allows you to check the existing sessions with Web3Auth. TheWeb3AuthState will allow users to remain authenticated with Web3Auth for 1 day by default, or a maximum of 7 days, or until the user logout or session data is cleared.

TheWeb3Auth initialization accepts asessionTime parameter.

let sessionTime =

86400

// 1 day Usage import

Foundation // IMP START - Quick Start import

Web3Auth // IMP END - Quick Start

class

ViewModel :

ObservableObject

{ var web3Auth :

Web3Auth ? @Published

var loggedIn :

Bool

=

false @Published

var user :

Web3AuthState ? @Published

var isLoading =

false @Published

var navigationTitle :

String

=

""

func

setup ( )

async

{ guard web3Auth ==

nil

else

{

return

} await

MainActor . run (body :

{ isLoading =

true navigationTitle =

"Loading" })

web3Auth

await

Web3Auth (W3AInitParams (clientId :

"" , network :

. sapphire_mainnet , sessionTime :

86400 ,

// 1 day

))

await

MainActor . run (body :

{ if

self . web3Auth ? . state !=

nil

{ user = web3Auth ? . state loggedIn =

true } isLoading =

false navigationTitle = loggedIn ?

"UserInfo"

:

"SignIn" }) }

func

login (provider :

```
Web3AuthProvider )
{ Task
{ do
{
let result =
try
await web3Auth ? . login ( W3ALoginParams ( loginProvider : provider ) )
await
MainActor . run ( body :
{ user = result loggedIn =
true } )
}
catch
{ print ( "Error" ) } } }
func
logout ( )
throws
{ Task
{ try
await web3Auth ? . logout ( ) await
MainActor . run ( body :
{ loggedIn =
false } ) } }
```

Selecting Curveâ

TheWeb3Auth().login(W3ALoginParams) method accepts acurve parameter. This parameter can be used to select the elliptic curve to use for the signature.

```
let result =
try
await
Web3Auth ( ) . login ( W3ALoginParams ( loginProvider : provider , curve :
SUPPORTED_KEY_CURVES . SECP256K1 ) ) * SECP256K1 * ED25519
Usage func
login ( provider :
Web3AuthProvider )
{ Task
{ do
{ let result =
```

try

await

Web3Auth ( ) . login (W3ALoginParams (loginProvider : provider , curve :

SUPPORTED_KEY_CURVES . SECP256K1)) // Perform action on success }

catch

{ print ("Error") } } }

login (provider :

. GOOGLE) Usage func

login (provider :

Web3AuthProvider)

{ Task

{ do

{ let result =

try

await

Web3Auth ( ) . login (W3ALoginParams (loginProvider : provider , curve :

SUPPORTED_KEY_CURVES . ED25519)) // Perform action on success }

catch

{ print ("Error") } } }

login (provider :

. GOOGLE)

Logging out a userâ

Web3Auth().logout()

â

This method will logout the user and remove the session id from the device. The user will need to login again to use the dApp next time the dApp is opened.

await

Web3Auth ( ) . logout ( )Edit this page Previous Initialize Next Whitelabel