SDK Integration

Now let's get to work on our App.tsx file!

First lets update some of our imports. I've changed the initial imports to the following:

import

"./App.css" ; import

"@Biconomy/web3-auth/dist/src/style.css" ; import

{ useState , useEffect , useRef }

from

"react" ; import

SocialLogin

from

"@biconomy/web3-auth" ; import

{

ChainId

}

from

"@biconomy/core-types" ; import

{ ethers }

from

"ethers" ; import

{

IBundler ,

Bundler

}

from

"@biconomy/bundler" ; import

{ BiconomySmartAccount , BiconomySmartAccountConfig , DEFAULT_ENTRYPOINT_ADDRESS , }

from

"@biconomy/account" ; import

{

IPaymaster ,

BiconomyPaymaster

}

from

"@biconomy/paymaster" ; import

Counter

from

"./Components/Counter" ; We are importing some css styles here but you can build your own login UI as well if needed.

Here is information about the rest of the imports:

- useState
- ,useEffect
- ,useRef
- : React hooks for managing component state
- and lifecycle.
- SocialLogin
- from@biconomy/web3-auth
- : A class from Biconomy SDK that
- allows you to leverage Web3Auth for social logins.
- ChainId from@biconomy/core-types
- : An enumeration of supported blockchain
- networks.
- ethers
- : A library for interacting with Ethereum.
- Ibundler
- andbundler
- will take UserOperations included in a mempool and
- and handle sending them to an entry point contract to be executed as a
- transaction onchain.
- BiconomySmartAccount
- ,BiconomySmartAccountConfig
- ,DEFAULT_ENTRYPOINT_ADDRESS
- from@biconomy/account
- to handle the
- configuration and methods of smart accounts
- IPaymaster
- andPaymaster
- will be used to sponsor gas fees for an
- account, provided specific predefined conditions are satisfied.

Now, let's setup our paymaster and bundler :

const

bundler :

IBundler

=

new

Bundler ({ bundlerUrl : "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" , chainId :

ChainId . POLYGON_MUMBAI , entryPointAddress :

DEFAULT_ENTRYPOINT_ADDRESS , }) ;

const

paymaster :

IPaymaster

=

new

BiconomyPaymaster ({ paymasterUrl : "https://paymaster.biconomy.io/api/v1/80001/[YOUR_API_KEY_HERE]" , }) ; You can get your Paymaster URL and bundler URL from Biconomy Dashboard. Follow the steps mentionedhere . :::

Let's take a look at some state variables that will help us with our implementation:

const

[smartAccount , setSmartAccount]

= useState < any

null ; const

[interval , enableInterval]

=

useState (false) ; const sdkRef =

(useRef <

SocialLogin)

|

(null

null) ; const

[loading , setLoading]

= useState < boolean

false ; const

[provider , setProvider]

= useState < any

null ; Here we have some state that will be used to track our smart account that will be generated with the sdk, an interval that will help us with checking for login status, a loading state, provider state to track our web3 provider and a reference to our Social login sdk.

Next let's add auseEffect hook:

useEffect (( )

=>

{ let

configureLogin : any ; if

(interval)

{ configureLogin =

setInterval (( )

=>

{ if

(! ! sdkRef . current ?. provider)

{ setupSmartAccount ( ) ; clearInterval (configureLogin) ; } } ,

1000) ; } } ,

[interval]) ; This use effect will be triggered after we open our login component, which we'll create a function for shortly. Once a user opens the component it will check if a provider is available and run the functions for setting up the smart account.

Now let's build our login function:

async

function

login ( )

{ if

(! sdkRef . current)

{ const socialLoginSDK =

new

SocialLogin ( ) ; const signature1 =

await socialLoginSDK . whitelistUrl ("http://127.0.0.1:5173/" ,) ; await socialLoginSDK . init ({ chainId : ethers . utils . hexValue (ChainId . POLYGON_MUMBAI) . toString ( ) , network :

"testnet" , whitelistUrls :

{ "http://127.0.0.1:5173/" : signature1 , } , }) ; sdkRef . current

= socialLoginSDK ; } if

(! sdkRef . current . provider)

{ sdkRef . current . showWallet ( ) ; enableInterval (true) ; }

else

{ setupSmartAccount ( ) ; } } Thelogin function is an asynchronous function that handles the login flow for the application. Here's a step-by-step explanation:

1. SDK Initialization
2. : The function first checks if thesdkRef
3. object
4. (which is a reference to the Biconomy SDK instance) is null. If it is, it
5. means that the SDK is not yet initialized. In this case, it creates a new
6. instance ofSocialLogin
7. (a Biconomy SDK component), whitelists a local URL
8. (http://127.0.0.1:5173/
9.), and initializes the SDK with the Polygon Mumbai
10. testnet configuration and the whitelisted URL. After initialization, it
11. assigns the SDK instance to sdkRef.current.
12. Provider Check
13. : After ensuring the SDK is initialized, the function
14. checks if the provider of thesdkRef
15. object is set. If it is not, it means
16. the user is not yet logged in. It then shows the wallet interface for the
17. user to login usingsdkRef.current.showWallet()
18. , and enables the interval
19. by callingenableInterval(true)
20. . This interval (setup in a useEffect hook
21. elsewhere in the code) periodically checks if the provider is available and
22. sets up the smart account once it is.
23. Smart Account Setup
24. : If the provider of sdkRef is already set, it means
25. the user is logged in. In this case, it directly sets up the smart account by
26. callingsetupSmartAccount()
27. .

In summary, thelogin function handles the SDK initialization and login flow. It initializes the SDK if it's not already initialized, shows the wallet interface for the user to login if they're not logged in, and sets up the smart account if the user is logged in. caution It is important to make sure that you update the whitelist URL with your production url when you are ready to go live! Now lets actually set up the smart account:

async

function

setupSmartAccount ( )

{ if

(! sdkRef ?. current ?. provider)

return ; sdkRef . current . hideWallet ( ) ; setLoading (true) ; const web3Provider =

new

ethers . providers . Web3Provider (sdkRef . current . provider) ; setProvider (web3Provider) ;

try

{ const

biconomySmartAccountConfig :

BiconomySmartAccountConfig

=

{ signer : web3Provider . getSigner ( ) , chainId :

ChainId . POLYGON_MUMBAI , bundler : bundler , paymaster : paymaster , } ; let biconomySmartAccount =

new

BiconomySmartAccount (biconomySmartAccountConfig) ; biconomySmartAccount =

await biconomySmartAccount . init ( ) ; console . log ("owner: " , biconomySmartAccount . owner) ; console . log ("address: " , await biconomySmartAccount . getSmartAccountAddress ( )) ; console . log ("deployed: " , await biconomySmartAccount . isAccountDeployed (await biconomySmartAccount . getSmartAccountAddress ( ))) ;

setSmartAccount (biconomySmartAccount) ; setLoading (false) ; }

catch

(err)

{ console . log ("error setting up smart account... " , err) ; } } ThesetupSmartAccount function is an asynchronous function used to initialize a smart account with Biconomy and connect it with the Web3 provider. Here's a step-by-step explanation of what it does:

1. if(!sdkRef?.current?.provider) return:
2. Checks if the sdkRef object
3. exists, and if it does, whether it has a provider property. If either of
4. these conditions is not met, the function returns early and does not proceed
5. further.
6. sdkRef.current.hideWallet():
7. Line calls the hideWallet() method on the
8. sdkRef.current object. It appears to be a method provided by the sdkRef
9. object, and it is likely used to hide the wallet or authentication interface
10. for the user.
11. setLoading(true):
12. Sets the state variable loading to true. It seems
13. like loading is used to indicate that some asynchronous operation is in
14. progress, and the UI might display a loading indicator during this time.
15. const web3Provider
16. = new
17. ethers.providers.Web3Provider(sdkRef.current.provider):
18. Creates a new
19. Web3Provider instance using the sdkRef.current.provider as the Web3 provider.
20. It assumes that sdkRef.current.provider is a valid Web3 provider, possibly
21. obtained from Biconomy's SDK.
22. setProvider(web3Provider):
23. Sets the web3Provider created in the
24. previous step as the state variable provider. This step likely enables other
25. parts of the application to access the Web3 provider.

Setting up BiconomySmartAccount:

1. const biconomySmartAccountConfig
2. Creates a configuration object for setting up the BiconomySmartAccount. The

3. configuration includes the following properties:

4. signer:

5. The signer (wallet) associated with the web3Provider.
6. chainId:
7. The chain ID, which is set to ChainId.POLYGON_MUMBAI. This

8. specifies the blockchain network where the BiconomySmartAccount is being
9. used (Polygon Mumbai, in this case).
10. bundler:
11. The bundler used for optimizing and bundling smart contracts. It
12. is expected that the bundler variable is defined elsewhere in the code.
13. paymaster:
14. The paymaster used for handling payment processing. It is
15. expected that the paymaster variable is defined elsewhere in the code.
16. let biconomySmartAccount
17. = new
18. BiconomySmartAccount(biconomySmartAccountConfig):
19. Creates a new instance of
20. BiconomySmartAccount using the provided configuration.
21. biconomySmartAccount
22. = await biconomySmartAccount.init():
23. Initializes
24. the BiconomySmartAccount instance by calling the init() method. It likely
25. performs some internal setup and prepares the account for use.
26. LoggingBiconomySmartAccount
27. information:
28. console.log("owner: ", biconomySmartAccount.owner):
29. Logs the owner of
30. the BiconomySmartAccount. The owner property might represent the Ethereum
31. address of the smart account owner.
32. console.log("address: ", await
33. biconomySmartAccount.getSmartAccountAddress()):
34. Logs the Ethereum address
35. of the BiconomySmartAccount using the getSmartAccountAddress() method. This
36. address is the entrypoint address mentioned earlier, and it serves as the
37. point of entry for interacting with the smart account through Biconomy.
38. console.log("deployed: ", await biconomySmartAccount.isAccountDeployed(await biconomySmartAccount.getSmartAccountAddress()))
39. :
40. Logs whether the smart account has been deployed or not. It calls the
41. isAccountDeployed() method on the BiconomySmartAccount instance, passing the
42. entrypoint address as an argument.
43. setSmartAccount(biconomySmartAccount)
44. :
45. Sets the biconomySmartAccount as
46. the state variable smartAccount. This step makes the BiconomySmartAccount
47. instance available to other parts of the application.
48. setLoading(false):
49. Sets the state variable loading to false, indicating
50. that the asynchronous operation is complete.
51. Error handling:
52. If any errors occur during the execution of the
53. function, the catch block will catch the error, and it will be logged to the
54. console.

So, in summary, thesetupSmartAccount function checks the availability of the Biconomy provider, hides the wallet interface, sets up a Web3 provider, creates and initializes a smart account, and then saves this account and the Web3 provider in the state. If any error occurs during this process, it is logged to the console.

Finally our last function will be a logout function:

const

logout

=

async

( )

=>

{ if

(! sdkRef . current)

{ console . error ("Web3Modal not initialized.") ; return ; } await sdkRef . current . logout ( ) ; sdkRef . current . hideWallet ( ) ; setSmartAccount (null) ; enableInterval (false) ; } ; Thelogout function is an asynchronous function that handles the logout flow for the application. Here's a breakdown of its functionality:

1. Check SDK Initialization
2. : The function first checks if thesdkRef
3. object (which is a reference to the Biconomy SDK instance) is null. If it is,
4. it means that the SDK is not yet initialized. In this case, it logs an error
5. message and returns immediately without executing the rest of the function.
6. Logout and Hide Wallet
7. : If the SDK is initialized, it logs the user out
8. by callingsdkRef.current.logout()
9. . This is an asynchronous operation,
10. hence the await keyword. It then hides the wallet interface by callingsdkRef.current.hideWallet()
11. .
12. Clear Smart Account and Interval
13. : After logging the user out and hiding
14. the wallet, it clears the smart account by callingsetSmartAccount(null)
15. ,
16. and disables the interval by callingenableInterval(false)
17. .

In summary, the logout function checks if the SDK is initialized, logs the user out and hides the wallet if it is, and then clears the smart account and disables the interval. If the SDK is not initialized, it logs an error message and does not execute the rest of the function. Previous Initialize Frontend Next Creating a Gasless Transaction