Creating a Gasless Transaction

Now with our SDK integration set up let's render our UI and execute some transactions! First let's create a new import in our App.tsx file:

import

Counter

from

"./Components/Counter" ; In the return for this component lets add the following JSX:

< div

< h1

Biconomy Smart Accounts using social login + Gasless Transactions </ h1

{ ! smartAccount &&

! loading &&

< button

onClick

{ login }

Login </ button

} { loading &&

< p

Loading account details... </ p

} { ! ! smartAccount &&

(< div

className

" buttonWrapper "

< h3

Smart account address: </ h3

< p

{ smartAccount . address } </ p

< Counter

smartAccount

{ smartAccount }

provider

{ provider }

/> < button

onClick

{ logout }

Logout </ button

</ div

) } < p

Edit < code

src/App.tsx </ code

and save to test </ p

< a href = " https://docs.biconomy.io/docs/overview " target = " _blank " className = " read-the-docs "

Click here to check out the docs </ a

</ div

If you followed all instructions from the last step to now your file should look something like this:

```
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' ; import
styles
from
'@/styles/Home.module.css'
const
bundler :
IBundler
=
new
Bundler ( { bundlerUrl :
'https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44' ,
// you can get this value from biconomy dashboard. chainId :
ChainId . POLYGON_MUMBAI , entryPointAddress :
DEFAULT_ENTRYPOINT_ADDRESS , } )
const
paymaster :
IPaymaster
=
new
BiconomyPaymaster ( { paymasterUrl :
'https://paymaster.biconomy.io/api/v1/80001/cIhIeS-I0.7e1f17b1-6ebb-454c-8499-c5f66dd098c6' } )
export
default
function
Home ( )
{ 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) ;

useEffect (( )

=>

{ let

configureLogin : any if

(interval)

{ configureLogin =

setInterval (( )

=>

{ if

(! ! sdkRef . current ?. provider)

{ setupSmartAccount ( ) clearInterval (configureLogin) } } ,

1000) } } ,

[interval])

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 ( ) } }

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) } }

const

logout

=

async

( )

=>

{ if

(! sdkRef . current)

{ console . error ('Web3Modal not initialized.') return } await sdkRef . current . logout ( ) sdkRef . current . hideWallet ( ) setSmartAccount (null) enableInterval (false) }

return

(< div

< h1

Biconomy

Smart

Accounts using social login +

Gasless

Transactions < / h1

{ ! smartAccount &&

! loading &&

< button onClick = { login }

Login < / button

} { loading &&

< p

Loading account details ... < / p

} { ! ! smartAccount &&

(< div className = "buttonWrapper"

< h3

Smart account address : < / h3

< p

{ smartAccount . address } < / p

< Counter smartAccount = { smartAccount } provider = { provider }

/

< button onClick = { logout }

Logout < / button

< / div

) } < p

Edit

< code

src / App . tsx < / code

and save to test < / p

< a href = "https://docs.biconomy.io/docs/overview" target = "_blank" className = "read-the-docs"

Click here to check out the docs < / a

< / div

) } Now lets create our Counter component!

If you do not already have a Components folder go ahead and create one within source and create a new file called Counter.tsx

We will also addreact-toastify for a nice toast to update our users about their transactions.

yarn add react-toastify Let's add our imports for this file:

import

React ,

{ useState , useEffect }

from

"react" ; import

{

BiconomySmartAccount

}

from

"@biconomy/account" ; import

{ IHybridPaymaster , SponsorUserOperationDto , PaymasterMode , }

from

"@biconomy/paymaster" ; import

abi

from

"../utils/counterAbi.json" ; import

{ ethers }

from

"ethers" ; import

{

ToastContainer , toast }

from

"react-toastify" ; import

"react-toastify/dist/ReactToastify.css" ; Make sure to get the abi from the smart contract we deployed in the first step. You can create utils folder and add the ABI json file there.

Here is an interface we will use for the props of this component:

interface

Props

{ smartAccount :

BiconomySmartAccount provider : any }

Now we can start building out this Counter Component:

TotalCountDisplay will be our functional component where the value ofcount will be passed to the component when it's used.

const

TotalCountDisplay :

React . FC < { count :

number

}

=

({ count })

=>

{ return

< div

Total count is { count } </ div

; } ; Counter is react function component. This component takes two props smartAccount and provider, which are expected to be passed when the component is used. The component uses React's useState hook to manage three states:count, counterContract, and isLoading . Additionally, the counterAddress variable is set to the value ofVITE_COUNTER_CONTRACT_ADDRESS .

const Counter : React . FC < Props

=

({ smartAccount , provider })

=>

{ const

[count , setCount]

=

useState < number

(0) ; const

[counterContract , setCounterContract]

=

useState < any

(null) ; const

[isLoading , setIsLoading]

=

useState < boolean

(false) ;

const counterAddress =

import . meta . env . VITE_COUNTER_CONTRACT_ADDRESS ; With this done let's go into the two functions this component will have:

useEffect (( )

=>

{ setIsLoading (true) ; getCount (false) ; } ,

[ ]) ;

const

getCount

=

async

(isUpdating :

boolean)

=>

{ const contract =

new

ethers . Contract (counterAddress , abi , provider) ; setCounterContract (contract) ; const currentCount =

await contract . count ( ) ; setCount (currentCount . toNumber ( )) ; if

(isUpdating)

{ toast . success ("Count has been updated!" ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ; } } ; We useUseEffect hook so that it sets the isLoading state to true initially and then calls the getCount(false). ThegetCount function is an asynchronous function responsible for fetching the current value of the counter from a smart contract on a blockchain and setting the state of the contract and count in the component.

1. Create Contract Instance
2. : It first creates an instance of the smart contract using theethers.Contract
3. constructor. It takes three arguments - the contract address, the contract ABI (Application Binary Interface), and the Web3 provider. This contract instance allows you to interact with the contract on the blockchain.
4. Set Contract State
5. : It then callssetCounterContract(contract)
6. to set the state of counterContract to the instance of the contract it just created.
7. Fetch Current Count
8. : It fetches the current value of the count from the smart contract by calling thecount()
9. function on the contract instance. Since this function interacts with the blockchain, it returns a Promise, hence theawait
10. keyword.
11. Set Count State
12. : It then callssetCount(currentCount.toNumber())
13. to set the state of count to the current value it just fetched. It converts the value to a number since the value returned from the contract is a BigNumber.
14. Display Toast Notification
15. : If theisUpdating
16. parameter istrue
17. , it displays a toast notification indicating that the count has been updated. This notification includes configuration for its position, auto-close time, behavior on click and hover, and theme.

In summary, thegetCount function creates a contract instance, sets the state of the contract, fetches the current value of the count from the contract, sets the state of the count, and optionally displays a toast notification.

const

incrementCount

=

async

( )

=>

{ try

{ toast . info ('Processing count on the blockchain!' ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ; TheincrementCount function is an asynchronous function that increments the count on the smart contract by sending a transaction to the blockchain.

1. Display Initial Toast Notification
2. : The function begins by displaying a toast notification to inform the user that the count is being processed on the blockchain.
3. Create Transaction
4. : It then creates a transaction object using the counterContract.populateTransaction.incrementCount()
5. function. This function prepares the data needed to call theincrementCount
6. function on the smart contract but does not send the transaction. The returned transaction object includes the data needed to call the function.
7. Define Transaction Parameters
8. : It defines the parameters for the transaction intx1
9. . This includes the address of the contract in to and the data needed to call the function indata
10. .
11. Send Transaction
12. : It sends the transaction to the blockchain using thesmartAccount.sendTransaction()
13. function. This function takes an object with the transaction parameters and sends the transaction. Since this function interacts with the blockchain, it returns a Promise, hence the await keyword.
14. Wait for Transaction Confirmation
15. : It then callstxResponse.wait()
16. to wait for the transaction to be confirmed on the blockchain. This function also returns a Promise, hence theawait
17. keyword.
18. Log Transaction Hash
19. : Once the transaction is confirmed, the transaction hash is logged to the console withconsole.log(txHash)
20. .
21. Update Count
22. : It then callsgetCount(true)
23. to fetch the updated count from the smart contract and update the state in the component. The true argument means that a toast notification will be displayed to indicate that the count has been updated.
24. Error Handling
25. : If an error occurs during this process, it is caught by thecatch
26. block. The error is logged to the console withconsole.log({error})
27. and a toast notification is displayed to inform the user that an error occurred.

In summary, theincrementCount function sends a transaction to a smart contract on a blockchain to increment a count, waits for the transaction to be confirmed, fetches the updated count, and handles any errors that occur during this process.

Now, let's work on our transaction data :

const incrementTx =

new

ethers . utils . Interface (["function incrementCount()"]) ; const data = incrementTx . encodeFunctionData ("incrementCount") ;

const tx1 =

{ to : counterAddress , data : data , } ;

let partialUserOp =

await smartAccount . buildUserOp ([tx1]) ;

const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

;

let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , // optional params... } ; * FunctionincrementCount * of the smart contract is being prepared using theethers.utils.Interface * to encode the function data. * A transaction object tx1 is created with the target contract address(counterAddress) * and the encoded function data (data), representing the "incrementCount()" function call. * The smartAccount is used to build a partial user operationpartialUserOp * that includes tx1. ThepaymasterServiceData * is prepared with optional parameters, specifying that the operation is sponsored. The IHybridPaymaster type ensures that thesmartAccount.paymaster * supports the sponsored mode for handling payment processing. * Here, we are supporting gasless transaction, which is why we setupmode: PaymasterMode.SPONSORED

Now, let's build try and catch block :

try

{ const paymasterAndDataResponse =

await biconomyPaymaster . getPaymasterAndData (partialUserOp , paymasterServiceData) ; partialUserOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ;

const userOpResponse =

await smartAccount . sendUserOp (partialUserOp) ; const transactionDetails =

await userOpResponse . wait ( ) ;

console . log ("Transaction Details:" , transactionDetails) ; console . log ("Transaction Hash:" , userOpResponse . userOpHash) ;

toast . success (Transaction Hash: { userOpResponse . userOpHash } ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ;

getCount (true) ; }

catch

(e)

{ console . error ("Error executing transaction:" , e) ; // ... handle the error if needed ... } }

catch

(error)

{ console . error ("Error executing transaction:" , error) ; toast . error ('Error occurred, check the console' ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ; } } ; Now, let's break down what's happening above :

- const paymasterAndDataResponse
- = await biconomyPaymaster.getPaymasterAndData(partialUserOp, paymasterServiceData);
- : Calls the getPaymasterAndData function on the biconomyPaymaster instance. It sends the partialUserOp and paymasterServiceData as arguments to fetch the necessary information and data related to the sponsored user operation.
- partialUserOp.paymasterAndData
- = paymasterAndDataResponse.paymasterAndData;
- : The paymasterAndData received from the previous step is added to the partialUserOp object. This likely includes data and configuration needed for the sponsored user operation.
- const userOpResponse
- = await smartAccount.sendUserOp(partialUserOp);
- : The partialUserOp containing the transaction details and paymaster information is sent as a user operation (sendUserOp) to the smartAccount. The smartAccount handles the meta-transaction and submits it to the blockchain.
- const transactionDetails
- = await userOpResponse.wait();
- : The wait() function is called on the userOpResponse, which awaits the completion of the user operation transaction. It returns the transaction details once the transaction is confirmed on the blockchain.

Logging and displaying transaction information:

a.console.log("Transaction Details:", transactionDetails); : The transaction details are logged to the console.

b.console.log("Transaction Hash:", userOpResponse.userOpHash); : The transaction hash (identifier) is logged to the console.

c.toast.success(...); : A toast notification is displayed to the user to indicate that the transaction was successful. The transaction hash is displayed in the notification.

getCount(true); : Calls the getCount function with true as an argument. It's likely used to fetch and update the latest count value from the smart contract after the successful transaction.

Error handling:

a. If any error occurs in the try block of code, the catch block will catch the error and log it to the console. Additionally, a toast notification is displayed to the user indicating that an error occurred. The details of the error are logged to the console for further investigation.

b. If any error occurs in the inner try block (getPaymasterAndData, sendUserOp, etc.), it will be caught in the corresponding catch block, and an error toast notification is displayed to the user. The error message will be logged to the console for debugging purposes.

Finally we round all this up by displaying the UI for our Toast and button:

return

(<

< TotalCountDisplay count = { count }

/

< ToastContainer position = "top-right" autoClose = { 5000 } hideProgressBar = { true } newestOnTop = { false } closeOnClick rtl = { false } pauseOnFocusLoss draggable pauseOnHover theme = "dark" /

< br

< / br

< button onClick = { ( )

=>

incrementCount ( ) }

Increment Count < / button

< /

) ; } ;

export

default Counter ; Congratulations you just created your first AA powered dApp. Users can now log in and have a smart account created for them and interact with a smart contract without the need to paying gas fees. Here is the complete implimintation ofCounter.tsx :

import React ,

{ useState , useEffect }

from

"react" ; import

{ BiconomySmartAccount }

from

"@biconomy/account" import

{ IHybridPaymaster , SponsorUserOperationDto , PaymasterMode , }

from

'@biconomy/paymaster' import abi from

"../utils/counterAbi.json" ; import

{ ethers }

from

"ethers" ; import

{ ToastContainer , toast }

from

'react-toastify' ; import

'react-toastify/dist/ReactToastify.css' ;

interface

Props

{ smartAccount : BiconomySmartAccount provider :

any }

const TotalCountDisplay : React . FC < { count :

number

```
}
=
( { count } )
=>
{ return
< div
        Total count is
{ count } < / div
        ; } ;
const Counter : React . FC < Props
=
( { smartAccount , provider } )
=>
{ const
[ count , setCount ]
=
useState < number
        ( 0 ) ; const
[ counterContract , setCounterContract ]
=
useState < any
        ( null ) ; const
[ isLoading , setIsLoading ]
=
useState < boolean
        ( false ) ;
const counterAddress =
import . meta . env . VITE_COUNTER_CONTRACT_ADDRESS ;
useEffect ( ( )
=>
{ setIsLoading ( true ) ; getCount ( false ) ; } ,
[ ] ) ;
const
getCount
=
async
( isUpdating :
```

boolean)

=>

{ const contract =

new

ethers . Contract (counterAddress , abi , provider) ; setCounterContract (contract) ; const currentCount =

await contract . count ( ) ; setCount (currentCount . toNumber ( )) ; if

(isUpdating)

{ toast . success ('Count has been updated!' ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ; } } ;

const

incrementCount

=

async

( )

=>

{ try

{ toast . info ('Processing count on the blockchain!' ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ;

const incrementTx =

new

ethers . utils . Interface (["function incrementCount()"]) ; const data = incrementTx . encodeFunctionData ("incrementCount") ;

```
const tx1 =

{ to : counterAddress , data : data , } ;

let partialUserOp =

await smartAccount . buildUserOp ( [ tx1 ] ) ;

const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

        ;

let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , // optional params... } ;

try

{ const paymasterAndDataResponse =

await biconomyPaymaster . getPaymasterAndData ( partialUserOp , paymasterServiceData ) ; partialUserOp .
paymasterAndData = paymasterAndDataResponse . paymasterAndData ;

const userOpResponse =

await smartAccount . sendUserOp ( partialUserOp ) ; const transactionDetails =

await userOpResponse . wait ( ) ;

console . log ( "Transaction Details:" , transactionDetails ) ; console . log ( "Transaction Hash:" , userOpResponse .
userOpHash ) ;

toast . success ( Transaction Hash: { userOpResponse . userOpHash }  ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ;

getCount ( true ) ; }

catch

( e )

{ console . error ( "Error executing transaction:" , e ) ; // ... handle the error if needed ... } }

catch

( error )

{ console . error ( "Error executing transaction:" , error ) ; toast . error ( 'Error occurred, check the console' ,

{ position :

"top-right" , autoClose :

5000 , hideProgressBar :

false , closeOnClick :
```

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , }) ; } } ;

return

(<

< TotalCountDisplay count = { count }

/

< ToastContainer position = "top-right" autoClose = { 5000 } hideProgressBar = { true } newestOnTop = { false } closeOnClick rtl = { false } pauseOnFocusLoss draggable pauseOnHover theme = "dark" /

< br

< / br

< button onClick = { ( )

=>

incrementCount ( ) }

Increment Count < / button

< /

) ; } ;

export

default Counter ; If you would like to see the completed project on github you can use the template below:https://github.com/vanshika-srivastava/scw-gasless-bico-modular Previous SDK Integration Next Next JS Fullstack Tutorial