# Execute a Gasless Transaction

In the previous section our work has primarily been in theindex.tsx file. Let's now create a new component that will handle all of our mint logic.

In yoursrc directory create a new folder calledcomponents and within the folder create aMinter.tsx file.

Doing any transaction with a smart contract requires the ABI of that contract. As a reminder if you are using the already deployed contract you can get the ABI directly fromhere on basescan .

In yoursrc firectory create a folder namedutils and create a file calledabi.json . Copy the abi into that folder.

Let's get started with building our component, first the imports:

import

{ useState }

from

"react" ; import

{ ethers }

from

"ethers" ; import abi from

"../utils/abi.json" ; import

{ IHybridPaymaster , SponsorUserOperationDto , PaymasterMode , }

from

"@biconomy/paymaster" ; import

{ BiconomySmartAccount }

from

"@biconomy/account" ; import styles from

"@/styles/Home.module.css" ; These are all of the imports you need to execute the gasless transaction.

I also added the NFT address below the imports as a variable:

const nftAddress =

"0x0a7755bDfb86109D9D403005741b415765EAf1Bc" ; If you deployed your own make sure to replace the address.

For type safety we're going to create an interface for our the Props of our component:

interface

Props

{ smartAccount : BiconomySmartAccount ; address :

string ; provider : ethers . providers . Provider ; } With our interface created lets start scaffolding out our component:

const Minter : React . FC < Props

=

( { smartAccount , address , provider } )

=>

{ return ( < { address &&

< button onClick = { handleMint } className = { styles . connect }

```
    Mint NFT < / button
```

}

< /

) } We're going to need to pass three items to this component: the instance of the smartAccount we saved to state in the index, the address of the smart account, as well as the provider from Particle Auth for signing the transactions before executing them. I have also added a Mint NFT button that needs a handleMint funciton. Let's write that function now:

const

handleMint

=

async

( )

=>

{ const contract =

new

ethers . Contract ( nftAddress , abi , provider ) ; try

{ const minTx =

await contract . populateTransaction . safeMint ( address ) ; console . log ( minTx . data ) ; const tx1 =

{ to : nftAddress , data : minTx . data , } ; let userOp =

await smartAccount . buildUserOp ( [ tx1 ] ) ; console . log ( { userOp } ) ; const biconomyPaymaster = smartAccount . paymaster as IHybridPaymaster < SponsorUserOperationDto

; let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , } ; const paymasterAndDataResponse = await biconomyPaymaster . getPaymasterAndData ( userOp , paymasterServiceData ) ;

userOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; const userOpResponse =

await smartAccount . sendUserOp ( userOp ) ; console . log ( "userOpHash" , userOpResponse ) ; const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; console . log ( "txHash" , receipt . transactionHash ) ; }

catch

( err :

any )

{ console . error ( err ) ; console . log ( err ) ; } } ; Here is what the above code does:

- we connect to the contract using ethers
- we use the etherspopulateTransaction
- method in order to create a raw transaction object
- we start constructing our transaction which is simply the start of our userOperation object:

const tx1 =

{ to : nftAddress , data : minTx . data , } ; The to value is what contract we are interacting with and the data field takes the data from our raw transaction object.

- We now use built in smartAccount methods to begin building the userOperation object.

let userOp =

await smartAccount . buildUserOp ( [ tx1 ] ) ; The next few lines are important in making sure this becomes a gasless transaciton. We need to update the userOp to also include thepaymasterAndData field so when the entry point contract executes the transaction, our gas tank on our paymaster will pay for the transaction cost.

let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , } ; const paymasterAndDataResponse =

await biconomyPaymaster . getPaymasterAndData ( userOp , paymasterServiceData , ) ; In the background the sdk is making a call to our Paymaster API (which is something you can actually interact with yourself!) and returning the data we need for this operation.

Finally we add the data to the userOp and send the userOp!

userOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; const userOpResponse =

await smartAccount . sendUserOp ( userOp ) ; const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; Thewait function optionally takes a number here if you want to wait for a specific number of network confirmations before considering this a success. In this case I just passed the number 1 in order to make sure there was at least 1 confirmation before showing the user any success messages.

Let's add two more things here for a better user experience:

First an additional state variable that keeps track of user having minted an NFT within this session:

const

[ minted , setMinted ]

=

useState < boolean

    ( false ) ; By default it will be set to false and after our userOpResponse has completed we can update the state:

const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; setMinted ( true ) ; Let's update the JSX:

<

    { minted &&

< a href = { https://testnets.opensea.io/ { address } }

    Click to view minted nfts for smart account < / a

    } < /

    Now after succesfully minting we'll show a link users can click to view their NFT on opensea.

We'll also add in React Toastify to send updates to the user regarding the transaction.

yarn add react-toastify We'll add these imports:

import

{ toast , ToastContainer }

from

"react-toastify" ; import

"react-toastify/dist/ReactToastify.css" ; And make another update to thehandleMint function:

```
const

handleMint

=

async

( )

=>

{ const contract =

new

ethers . Contract ( nftAddress , abi . abi , provider ) ; try

{ toast . info ( "Minting your NFT..." ,

{ position :

"top-right" , autoClose :

15000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ; const minTx =

await contract . populateTransaction . safeMint ( address ) ; console . log ( minTx . data ) ; const tx1 =

{ to : nftAddress , data : minTx . data , } ; console . log ( "here before userop" ) ; let userOp =

await smartAccount . buildUserOp ( [ tx1 ] ) ; console . log ( { userOp } ) ; const biconomyPaymaster = smartAccount .
paymaster as IHybridPaymaster < SponsorUserOperationDto

    ; let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , } ; const paymasterAndDataResponse = await biconomyPaymaster .
getPaymasterAndData ( userOp , paymasterServiceData ) ;

userOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; const userOpResponse =

await smartAccount . sendUserOp ( userOp ) ; console . log ( "userOpHash" , userOpResponse ) ; const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; console . log ( "txHash" , receipt . transactionHash ) ; setMinted ( true ) ; toast . success
( ` Success! Here is your transaction: { receipt . transactionHash }

` , { position :

"top-right" , autoClose :

18000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :
```

undefined , theme :

"dark" , } , ) ; }

catch

( err :

any )

{ console . error ( err ) ; console . log ( err ) ; } } ; To make sure that this toast shows up in our dApp we need to add the component into the JSX, here is the last update:

<

{ address &&

< button onClick = { handleMint } className = { styles . connect }

Mint NFT < / button

} { minted &&

< a href = { https://testnets.opensea.io/ { address } }

Click to view minted nfts for smart account < / a

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

< /

Don't forget to also import the Minter function into yourindex.tsx

< main className = { styles . main }

< h1

Based Account Abstraction < / h1

< h2

Connect and Mint your AA powered NFT now < / h2

{ ! loading &&

! address &&

< button onClick = { connect } className = { styles . connect }

Connect to Based Web3 < / button

} { loading &&

< p

Loading Smart Account ... < / p

} { address &&

< h2

Smart Account :

{ address } < / h2

} { smartAccount && provider &&

< Minter smartAccount = { smartAccount } address = { address } provider = { provider }

/

} < / main

Now you're all set, you created a Next JS application that leverages Account Abstraction and Social Logins via the Biconomy SDK and Particle Auth. If you need to review the completed code check out the full snippets below for theindex.tsx andMinter.tsx files!

```
index.tsx import Head from

'next/head' import

{ ParticleAuthModule , ParticleProvider , }

from

"@biconomy/particle-auth" ; import styles from

'@/styles/Home.module.css' import

{ useState }

from

'react' ; import

{ IBundler , Bundler }

from

'@biconomy/bundler' import

{ BiconomySmartAccount , BiconomySmartAccountConfig ,

DEFAULT_ENTRYPOINT_ADDRESS

}

from

"@biconomy/account" import

{ ethers }

from

'ethers' import

{ ChainId }

from

"@biconomy/core-types" import

{ IPaymaster , BiconomyPaymaster , }

from

'@biconomy/paymaster' import Minter from

'@/components/Minter' ;

export

default

function

Home ( )

{ const

[ address , setAddress ]

=

useState < string
```

```
        ( "" ) const
[ loading , setLoading ]
=
useState < boolean
        ( false ) ; const
[ smartAccount , setSmartAccount ]
=
useState < BiconomySmartAccount |
null
        ( null ) ; const
[ provider , setProvider ]
=
useState < ethers . providers . Provider |
null
        ( null )
const particle =
new
ParticleAuthModule . ParticleNetwork ( { projectId :
"bb8d58f8-0d3c-4306-a5f1-6cc7aa73b012" , clientKey :
"c9rwyb2a3pQhHapL1EphoNKYnFsVQkAEHgWP5TRm" , appId :
"bd23aa64-ef27-4054-a823-25aa32d903a4" , wallet :
{ displayWalletEntry :
true , defaultWalletEntryPosition : ParticleAuthModule . WalletEntryPosition . BR , } , } ) ;
const bundler : IBundler =
new
Bundler ( { bundlerUrl :
'https://bundler.biconomy.io/api/v2/84531/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44' , chainId : ChainId .
BASE_GOERLI_TESTNET , entryPointAddress :
DEFAULT_ENTRYPOINT_ADDRESS , } )
const paymaster : IPaymaster =
new
BiconomyPaymaster ( { paymasterUrl :
'https://paymaster.biconomy.io/api/v1/84531/m814QNmpW.fce62d8f-41a1-42d8-9f0d-2c65c10abe9a' } )
const
connect
=
async
( )
```

```
=>

{ try

{ setLoading ( true ) const userInfo =

await particle . auth . login ( ) ; console . log ( "Logged in user:" , userInfo ) ; const particleProvider =

new

ParticleProvider ( particle . auth ) ; console . log ( { particleProvider } ) const web3Provider =

new

ethers . providers . Web3Provider ( particleProvider , "any" ) ; setProvider ( web3Provider ) const biconomySmartAccountConfig : BiconomySmartAccountConfig =

{ signer : web3Provider . getSigner ( ) , chainId : ChainId . BASE_GOERLI_TESTNET , bundler : bundler , paymaster : paymaster } let biconomySmartAccount =

new

BiconomySmartAccount ( biconomySmartAccountConfig ) biconomySmartAccount =

await biconomySmartAccount . init ( ) setAddress (

await biconomySmartAccount . getSmartAccountAddress ( ) ) setSmartAccount ( biconomySmartAccount ) setLoading ( false ) }

catch

( error )

{ console . error ( error ) ; } } ;

return

( <

        < Head

        < title

        Based Account Abstraction < / title

        < meta name = "description" content = "Based Account Abstraction"

/

        < / Head

        < main className = { styles . main }

        < h1

        Based Account Abstraction < / h1

        < h2

        Connect and Mint your AA powered NFT now < / h2

        { ! loading &&

! address &&

< button onClick = { connect } className = { styles . connect }

        Connect to Based Web3 < / button

        } { loading &&

< p

        Loading Smart Account ... < / p
        } { address &&
< h2
        Smart Account :
{ address } < / h2
        } { smartAccount && provider &&
< Minter smartAccount = { smartAccount } address = { address } provider = { provider }
/
        } < / main
        < /
        ) } Minter.tsx import
{ useState }
from
'react' ; import
{ ethers }
from
"ethers" ; import abi from
"../utils/abi.json" import
{ IHybridPaymaster , SponsorUserOperationDto , PaymasterMode }
from
'@biconomy/paymaster' import
{ BiconomySmartAccount }
from
"@biconomy/account" import
{ toast , ToastContainer }
from
'react-toastify' ; import styles from
'@/styles/Home.module.css' import
'react-toastify/dist/ReactToastify.css' ;
const nftAddress =
"0x0a7755bDfb86109D9D403005741b415765EAf1Bc"
interface
Props
{ smartAccount : BiconomySmartAccount , address :
string , provider : ethers . providers . Provider , }
const Minter : React . FC < Props
=
( { smartAccount , address , provider } )
```

```
=>

{ const

[ minted , setMinted ]

=

useState ( false )

const

handleMint

=

async

( )

=>

{ const contract =

new

ethers . Contract ( nftAddress , abi . abi , provider , ) try

{ toast . info ( 'Minting your NFT...' ,

{ position :

"top-right" , autoClose :

15000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ; const minTx =

await contract . populateTransaction . safeMint ( address ) ; console . log ( minTx . data ) ; const tx1 =

{ to : nftAddress , data : minTx . data , } ; console . log ( "here before userop" ) let userOp =

await smartAccount . buildUserOp ( [ tx1 ] ) ; console . log ( { userOp } ) const biconomyPaymaster = smartAccount .
paymaster as IHybridPaymaster < SponsorUserOperationDto

        ; let paymasterServiceData : SponsorUserOperationDto =

{ mode : PaymasterMode . SPONSORED , } ; const paymasterAndDataResponse = await biconomyPaymaster .
getPaymasterAndData ( userOp , paymasterServiceData ) ;

userOp . paymasterAndData = paymasterAndDataResponse . paymasterAndData ; const userOpResponse =

await smartAccount . sendUserOp ( userOp ) ; console . log ( "userOpHash" , userOpResponse ) ; const

{ receipt }

=

await userOpResponse . wait ( 1 ) ; console . log ( "txHash" , receipt . transactionHash ) ; setMinted ( true ) toast . success (
` Success! Here is your transaction: { receipt . transactionHash }

` ,

{ position :
```

"top-right" , autoClose :

18000 , hideProgressBar :

false , closeOnClick :

true , pauseOnHover :

true , draggable :

true , progress :

undefined , theme :

"dark" , } ) ; }

catch

( err :

any )

{ console . error ( err ) ; console . log ( err ) } } return ( <

    { address &&

< button onClick = { handleMint } className = { styles . connect }

    Mint NFT < / button

    } { minted &&

< a href = { https://testnets.opensea.io/ { address } }

    Click to view minted nfts for smart account < / a

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

    < /

    ) }

export

default Minter ; Previous SDK Integration Next Smart Accounts : V1 to V2 Upgrade