Quickstart ⚡

In this guide, we will create a smart account and mint an NFT with Biconomy's SDK and Node.js.

info For a detailed exploration of the SDK's inner workings, delve into our more granular API documentation .

1. Setup

Let's create a Typescript project using npm to get started. Let's start by adding some tooling:

```
mkdir biconomy-sa && cd biconomy-sa && npm init -y && npm i --save-dev @types/node tslib
```

Then add biconomy's account package:

```
npm i @biconomy/account
```

Create the following tsconfig.json (TypeScript config):

```
{ "compilerOptions": { "target": "es5", "module": "commonjs", "declaration": true, "outDir": "./lib", "strict": true, "esModuleInterop": true, "skipLibCheck": true }, "include": ["./*.ts"] }
```

The Account package will help you with creating Smart Account and an interface with them to send transactions. The same package can be used to also import the paymaster and bundler.

2. Create a Smart Account

The Externally Owned Account (EOA) corresponding to the private key will serve as the owner of the Smart Account we create. You can get the private key from wallets like MetaMask, TrustWallet, Coinbase Wallet, etc.

Create an index.ts file, copy the following code in it and replace the PRIVATE_KEY .

Be sure to never publicly expose your private key.

```
import

{ Hex , createWalletClient , encodeFunctionData , http , parseAbi , zeroAddress , }

from

"viem" ; import

{ privateKeyToAccount }

from

"viem/accounts" ; import

{ polygonMumbai }

from

"viem/chains" ; import

{ createSmartAccountClient }

from

"@biconomy/account" ;

const bundlerUrl = "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" ;

// Found at https://dashboard.biconomy.io

export

const

createAccountAndMintNft

=

async

( )

=>

{ // ----- 1. Generate EOA from private key const account =

privateKeyToAccount ( "0x"

+

"PRIVATE_KEY" ) ; const client =

createWalletClient ( { account , chain : polygonMumbai , transport :

http ( ) , } ) ; const eoa = client . account . address ; console . log(EOA address: { eoa } ) ;

// ------ 2. Create biconomy smart account instance const smartAccount =

await

createSmartAccountClient ( { signer : client , bundlerUrl , } ) ; const saAddress =

await smartAccount . getAccountAddress ( ) ; console . log ( "SA Address" , saAddress ) ; } ; createAccountAndMintNft ( ) ;
```

Next, run the following command:

```
npx ts-node index.ts
```

You should see EOA and the smart account address in the console.

```
EOA address: 0x7d4EcAc73A5574447E0F38b8451DC9B29D8Fd083 SA Address 0x4dF23B78543F5c2F9CBCDF09956288B3e97bb9a4
```

Smart accounts are designed with a pre-determined address known prior to deployment. Users transfer funds to this address and then the actual deployment transaction is internally batched with the first transaction that is done by that smart Account. Alternatively the paymaster could be used to sponsor this transaction.

caution Before continuing, now that we have our Smart Account address, we need to fund it with some test network tokens! Since we are using the Polygon Mumbai network, head over to the Polygon Faucet , paste in your smart account address, and get some test tokens! If you skip this step, you might run into the AA21 didn't pay prefund error ! Once you have t available, follow the next steps.

3. Execute your first transaction

Let's create your first transaction

- to
- : Smart Contract your Smart Account will interact with.
- data
- : For the mint function we will pass the nftData
- created using encodeFunctionData
- .
- value

- : For the mint function this won't be needed. Depending on the transaction, you can pass the value.

Add the following code in the script above at the bottom of thecreateAccountAndMintNft function.

try

{ const nftAddress =

"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; const parsedAbi =

parseAbi (["function safeMint(address _to)"]) ; const nftData =

encodeFunctionData ({ abi : parsedAbi , functionName :

"safeMint" , args :

[saAddress as Hex] , }) ;

// ------ 4. Send transaction const userOpResponse =

await smartAccount . sendTransaction ({ to : nftAddress , data : nftData , }) ; const

{ transactionHash }

=

await userOpResponse . waitForTxHash ( ) ; console . log ("transactionHash" , transactionHash) ; const userOpReceipt =

await userOpResponse . wait ( ) ; if (userOpReceipt . success ==

'true')

{ console . log ("UserOp receipt" , userOpReceipt) console . log ("Transaction receipt" , userOpReceipt . receipt) } }

catch

(error :

unknown)

{ if

(error instanceof

Error)

{ console . error ("Transaction Error:" , error . message) ; } } 1. We create a transaction object. 2. We send thetransaction 3. to our bundler. 4. We store the response in a variable calledtransactionResponse 5. . 6. We retrieve the transaction detail by callinguserOpResponse.waitForTxHash() 7. .

To wait for a specific number of network confirmations before getting the value, usewait() with a number argument. For instance,transactionResponse.wait(5) waits for 5 confirmations before returning the value.

Check out the long transaction details available now in your console! You just created and executed your first userOps using the Biconomy SDK. The entire Biconomy crew is sending you a big round of applause!

You can also view the latest transaction events on the Polygon Mumbai Testnet Explorer .

View Complete Code import

{ Hex , createWalletClient , encodeFunctionData , http , parseAbi , zeroAddress , }

from

"viem" ; import

{ privateKeyToAccount }

from

"viem/accounts" ; import

{ polygonMumbai }

from

"viem/chains" ; import

{ createSmartAccountClient }

from

"@biconomy/account" ;

export

const

createAccountAndMintNft

=

async

( )

=>

{ // ----- 1. Generate EOA from private key const account =

privateKeyToAccount ("0x"

+

"PRIVATE_KEY") ; const client =

createWalletClient ({ account , chain : polygonMumbai , transport :

http ( ) , }) ; const eoa = client . account . address ; console . log(EOA address: { eoa }) ;

// ------ 2. Create biconomy smart account instance const smartAccount =

await

```
createSmartAccountClient ( { signer : client , bundlerUrl : "https://bundler.biconomy.io/api/v2/80001/nJPK7B3ru.dd7f7861-190d-41bd-af80-6877f74b8f44" , } ) ; const saAddress =
await smartAccount . getAccountAddress ( ) ; console . log ( "SA Address" , saAddress ) ;
// ------ 3. Generate transaction data const nftAddress =
"0x1758f42Af7026fBbB559Dc60EcE0De3ef81f665e" ; const parsedAbi =
parseAbi ( [ "function safeMint(address _to)" ] ) ; const nftData =
encodeFunctionData ( { abi : parsedAbi , functionName :
"safeMint" , args :
[ saAddress as Hex ] , } ) ;
// ------ 4. Send transaction const
{ waitForTxHash }
=
await smartAccount . sendTransaction ( { to : nftAddress , data : nftData , } ) ;
const
{ transactionHash }
=
await
waitForTxHash ( ) ; console . log ( "transactionHash" , transactionHash ) ;
const userOpReceipt =
await userOpResponse . wait ( ) ; if ( userOpReceipt . success ==
'true' )
{ console . log ( "UserOp receipt" , userOpReceipt ) console . log ( "Transaction receipt" , userOpReceipt . receipt ) } } ; createAccountAndMintNft ( ) ;
```

If you run into any errors, check outtroubleshooting for common errors.

Congratulations on completing the quickstart!

Check out Tutorials

Explore the tutorials for various use cases.

Send a batch of transactions

In this you will do batching of transactions which is sending more than one transaction in an array and execute them with a single signature.

Send a gasless transaction

In this you will do a gasless transaction for which you will not have to pay any gas fees and with empty funds in your smart account you can execute transactions.

Pay gas in ERC20

In this you will execute a transaction by paying gas fees in ERC20 tokens, for this your smart account should have the specific ERC20 token for paying gas fees.

To dive deeper, check out moreuse cases . Previous Overview Next Tutorials