How to use a Dynamic signer with permissionless.js

permissionless.js allows you to plug in custom signers to control the accounts that you create. Dynamic is an embedded wallet provider that allows you to easily onboard users to your dapp. It is possible to use Dynamic as a signer with permissionless.js, allowing you to use Dynamic to create and control smart accounts and sign transactions.

Install the dependencies

```
npmi @dynamic-labs/sdk-react-core @dynamic-labs/wagmi-connector @dynamic-labs/ethereum permissionless viem wagmi
```

Create the Dynamic provider

Following Dynamic'squickstart guide , set up the Dynamic provider in your app. Also integrate the DynamicWagmiConnector, which will allow you to use Dynamic as a signer with permissionless.js.

```
import{ DynamicContextProvider, DynamicWidget, }from"@dynamic-labs/sdk-react-core"; import{ DynamicWagmiConnector }from"@dynamic-labs/wagmi-connector";

import{ EthereumWalletConnectors }from"@dynamic-labs/ethereum";

exportconstApp=()=>{ return( <DynamicContextProvider settings={{ // Find your environment id at https://app.dynamic.xyz/dashboard/developer environmentId:"REPLACE-WITH-YOUR-ENVIRONMENT-ID", walletConnectors: [EthereumWalletConnectors], }}

        ); };
```

Create the SmartAccountClient

Create the smart account client using the Dynamic signer. Note: DynamicWagmiConnector internally sets up the WagmiConfig, so there is no need to do it separately. This is where you would configure what smart account implementation (e.g.Safe ,Kernel , Biconomy,SimpleAccount) and what paymaster logic you want to use.

```
import{ createSmartAccountClient, walletClientToSmartAccountSigner }from"permissionless"; import{ signerToSimpleSmartAccount }from"permissionless/accounts"; import{ useWalletClient }from"wagmi";

const{ data:walletClient }=useWalletClient()

constsigner=walletClientToSmartAccountSigner(walletClient)

constsimpleSmartAccountClient=awaitsignerToSimpleSmartAccount(publicClient, { entryPoint:"0x5FF137D4b0FDCD49DcA30c7CF57E578a026d2789", signer: signer, factoryAddress:"0x9406Cc6185a346906296840746125a0E44976454" })

constsmartAccountClient=createSmartAccountClient({ account: simpleSmartAccountClient, chain: sepolia,// or whatever chain you are using bundlerTransport:http(""), entryPoint:ENTRYPOINT_ADDRESS_V06, middleware: { sponsorUserOperation: paymasterClient.sponsorUserOperation,// optional, if using a paymaster }, })
```

Send a transaction

You can now send transactions as normal. ThesponsorUserOperation function will be called before each transaction is signed and sent, applying the custom paymaster logic you have set.

```
consttxHash=awaitsmartAccountClient.sendTransaction({ to: zeroAddress, data:"0x", value:BigInt(0) })
```