Dynamic Embedded Wallets

What is an Embedded Wallet?

You can think of an embedded wallet like a powerful web-account. An embedded wallet is a programmable web3 crypto wallet that can be issued invisibly to customers on your website. Customers with an embedded wallet can immediately receive digital tokens and make on-chain interactions without needing to go through the complexities of understanding the intricacies of typical EOA wallets like metamask or Phantom.

Dynamic embedded wallets are fully non custodial.

Initial setup

Using Embedded Wallets is as simple as one, two, three!

1 Add Dynamic to your application

Get your SDK setup as normal using thequickstarts . 2 Toggle Embedded Wallets on

Toggle on “Enable Dynamic embedded wallets” in the “Email/ Social Auth & Embedded Wallets” section of theDynamic dashboard . 3 Check your other settings

Make sure you haveconnect-and-sign enabled,email OTP enabled andCORS set up. That’s it! By default, we will launch the wallet creation flow after the user confirms OTP.

You can try this flow in action yourself athttps://demo.dynamic.xyz/ .

Further Configuration

Multi-chain

Dynamic offers embedded wallets on EVM compatible networks and Solana. If you enable both, they will both be created at once and whichever you have marked as “primary” will be shown as the primary address in their profile upon sign in.

To enable embedded wallets for EVM or Solana networks the respective chains must also be enabled. You can find more information about enabling chains and networkshere .

Signing & Security Methods

After the wallet is generated and attached to the user, they still need to authenticate with their embedded wallet in order to transact and sign. There are two ways you can allow them to do this:

1. Passkey
2. : The user will be prompted to create a passkey. This passkey will be used to sign transactions and messages. This is the default option.
3. One Time Code
4. : The user will be prompted to enter a one time code sent to their email. This code will be used to generate a session key which can sign transactions and messages.

Security Prompt Behavior

In this section of the dashboard you can choose whether the user needs to authenticate (“claim”) the embedded wallet at the point at which it’s first generated for them, or if this action should be deferred to when they send their first on-chain transaction or off-chain message.

If you toggle on the “On first transaction” option, a pre-generated wallet will be created for the user. They will be prompted to

add a signing method when they need to use the wallet for the first time such as sending a on-chain transaction or signing an off-chain message.

That's compared to the second option ("At initial signup") which will require the user to add a signing method before they can continue with the sign-up flow.

Deferring can be very helpful if you want to reduce friction in the sign-up flow, but it's important to note that if you choose this option, the user will not be able to sign messages or transactions until they have completed, however their wallet can still receive assets.

Manual Mode

Inside the dashboard configuration section for Dynamic wallet as-a-service, you will see that you are provided with the choice as to whether "Manual mode" is toggled on or off. When it is off (the default), the wallet creation flow will be triggered automatically during signup. If you toggle it on, you will need to trigger the wallet creation flow yourself. You can find more information about this in theHooks and Callbacks section below.

Wallet recovery

You can learn more about this toggle in the Recovery guidehere .

Hooks and Callbacks

If you have chosen Manual wallet creation, you will need to trigger the wallet creation flow yourself. To do this, you can use the new "createEmbeddedWallet" hook. Here's an example!

import { useEmbeddedWallet} from "@dynamic- labs/ sdk- react- core"const { createEmbeddedWallet, userHasEmbeddedWallet} = useEmbeddedWallet ( ) ; const onClick = async ( ) => { if (! userHasEmbeddedWallet ( )) { try { const walletId= await createEmbeddedWallet ( ) ; // do whatever you want with that Id } catch (e) { // handle error } } } return (< button onClick= { ( ) => onClick ( ) }

> Create Wallet < / button>) You can find the complete specification of this hook in the SDK reference sectiohere .

There is also a callback available for you if you need to hook into the action of a wallet being successfully created. It's called "onEmbeddedWalletCreated" and the spec can be foundhere .

Account Abstraction

You can turn these embedded wallets into smart contract wallets using ouraccount abstraction feature.

Example code

Sending an Ethereum transaction

const publicClient= createPublicClient ({ transport : http ('https://rpc.ankr.com/eth_goerli') , }) ; const client : WalletClient = (await primaryWallet?. connector. getSigner ( )) as WalletClient ; const txRequest= { to : '0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045' as 0x { string} , value : 1000000000000000000n , } ; const txHash= await client. sendTransaction (txRequest) ; console . log (Success! Transaction broadcasted with hash { txHash}) ; await publicClient. waitForTransactionReceipt ({ hash : txHash, }) ;

Sending a legacy Solana transaction

const { primaryWallet} = useDynamicContext ( ) ; const connection : any= await (primaryWalletas any) . connector .

```
getConnection ( ) ; if ( ! connection) return ; const fromKey= new PublicKey ( primaryWallet. address ) ; const toKey= new PublicKey ( address) ; const amountInLamports= Number ( amount) * 1000000000 ; const transferTransaction= new Transaction ( ) . add ( SystemProgram . transfer ( { fromPubkey : fromKey, lamports : amountInLamports, toPubkey : toKey, } ) , ) ; const blockhash= await connection. getLatestBlockhash ( ) ; transferTransaction. recentBlockhash = blockhash. blockhash ; transferTransaction. feePayer = fromKey; await ( primaryWalletas any) . connector . signAndSendTransaction ( { transaction : transferTransaction} ) . then ( ( res : any ) => { console . log ( Transaction successful: https://solscan.io/tx/ { res} ?cluster=devnet , ) ; } ) . catch ( ( reason : any ) => { console . error ( reason) ; } ) ; } ;
```

Sending a versioned Solana transaction

```
const { primaryWallet} = useDynamicContext ( ) ; const connection : any= await ( primaryWalletas any) . connector . getConnection ( ) ; if ( ! connection) return ; const fromKey= new PublicKey ( primaryWallet. address ) ; const toKey= new PublicKey ( address) ; const amountInLamports= Number ( amount) * 1000000000 ; const instructions= [ SystemProgram . transfer ( { fromPubkey : fromKey, lamports : amountInLamports, toPubkey : toKey, } ) , ] ; const blockhash= await connection. getLatestBlockhash ( ) ; // create v0 compatible message const messageV0= new TransactionMessage ( { instructions, payerKey : fromKey, recentBlockhash : blockhash. blockhash , } ) . compileToV0Message ( ) ; const transferTransaction= new VersionedTransaction ( messageV0) ; await ( primaryWalletas any) . connector . signAndSendTransaction ( { transaction : transferTransaction} ) . then ( ( res : any ) => { console . log ( Transaction successful: https://solscan.io/tx/ { res} ?cluster=devnet , ) ; } ) . catch ( ( reason : any ) => { console . error ( reason) ; } ) ; } ;
```

Was this page helpful?