Integration with Monerium

TheMoneriumPack (opens in a new tab) enables Safe users to make direct transfers of e-money tokens from their Safe accounts to an IBAN via the SEPA network. This allows them to use Monerium and Safe services together.

More info about Monerium:

- Monerium(opens in a new tab)
- Monerium Developers(opens in a new tab)

What are we going to learn?

This guide demonstrates how to use theMoneriumPack as part of theOnRampKit (opens in a new tab) and incorporate it into your web application.

Prerequisites

1. Node.js and npm(opens in a new tab)
2. Monerium account and application(opens in a new tab)
3. A web application using your favorite CLI and language. For example:
4. Vite for React projects(opens in a new tab)
5. Vue CLI(opens in a new tab)
6. Angular CLI(opens in a new tab)

Monerium offers several authentication methods for web apps. TheMoneriumPack implements theAuthorization code flow with proof key for code exchange (PKCE)(opens in a new tab) .

Install dependencies

yarn

add

@safe-global/onramp-kit

@safe-global/protocol-kit

@monerium/sdk

Login with Monerium

Creating aLogin with Monerium integration for the Safe requires a multi-step process that should be implemented in your web app. The following steps are required.

1. Load the application and initialize theMoneriumPack
2. using the following snippet:

import { MoneriumPack } from

'@safe-global/onramp-kit'

const

moneriumPack

=

new

MoneriumPack ({ clientId : { YOUR_CLIENT_ID } ,

// Get your client id from Monerium redirectUrl :

'https://your-site-redirect-url'

// return URL after authenticating in Monerium environment: 'sandbox'

// Use the proper Monerium environment ('sandbox' | 'production')}) })

await

moneriumPack .init ({ protocolKit })

TheprotocolKit is an instance of theSafe (opens in a new tab) class. For more information on how to instantiate theprotocol-kit refer to theProtocol Kit Quickstart section .

TheMoneriumPack will use the Safe account address configured in theprotocolKit to link to your Monerium account.

1. Start theLogin with Monerium
2. flow by creating a button or link in your application. Use your favorite UI library to add a handler and start the login flow. In the button handler you should start the flow by calling theopen
3. method:

await

moneriumPack .open ({ initiateAuthFlow :

true })

This action will open the Monerium web page to begin the authentication process and get the permissions to gain access to your information.

Take a look to the Monerium web page URL. You will see that the query string includes parameters such as these, among others:

- address
- : The Safe address you want to bind to your Monerium account.
- signature
- : Always "0x." It means the origin wallet is a multisig, just like the Safe. Monerium will then verify for onchain signatures in your Safe contract.
- chain
- : The Monerium selected chain. The value is automatically calculated using theprotocolKit
- instance.
- network
- : The Monerium selected network. The value is automatically calculated using theprotocolKit
- instance.
- redirect_uri
- : TheredirectUrl
- you sent in theMoneriumPack
- constructor. Once authenticated Monerium dashboard will redirect to that URL.

â ï¸ It's important to be aware of what's happening during this initial interaction. AsignMessage transaction accepting therequired Monerium message(opens in a new tab) will be proposed to your Safe the first time you try to link your Safe address. To confirm and execute it, you must also add the remaining signatures in the Safe UI and execute the transaction.

Once you are in the Monerium web page, you should login or create a new account if you don't have one already.

After that, the Monerium UI, will wait until it detects the Safe address has signed the message as explained above and the transaction is executed. You may need to reload the Monerium UI page once you know the transaction was executed in order to continue with the flow. Alternatively, you can start the process again and Monerium will detect the transaction confirmation immediately.

Note. If you use thesandbox environment, you can test this flow without KYC inGoerli . To use the mainnets forethereum ,gnosis orpolygon in production, you need to create an account in the Monerium dashboard and complete KYC. Theproduction environment involves real money, whereas thesandbox environment uses fake money and test tokens.

1. Once you authenticate through the Monerium UI, a window will appear asking for permission to access your information. If you accept, control will be returned to theredirectUrl
2. you specified in theMoneriumPack
3. constructor.
4. Once we reach theredirectUrl
5. we can call open again, now without using arguments.

const

safeMoneriumClient

=

await

moneriumPack .open ()

If the process was successful, you will be now authenticated with Monerium and your Safe will be linked!. You can start using thesafeMoneriumClient instance to interact with your Monerium account. This instance is an enhanced version of the Monerium SDK, with some additional Safe features.

To learn more about the methods you can use with thesafeMoneriumClient instance, check theMonerium SDK documentation(opens in a new tab) .

Here are some examples:

const

authContext

=

await

moneriumClient .getAuthContext () const

profile

=

await

moneriumClient .getProfile (authContext .defaultProfile) const

balances

=

await

moneriumClient .getBalances () const

orders

=

await

moneriumClient .getOrders ()

1. When you reload a page, you usually want to stay authenticated as long as the tokens are valid. To do this, just call theopen()
2. method without arguments.
3. You are now ready to place orders. In theproduction
4. environment, real funds will be transferred. In thesandbox
5. environment, fake money will be used. If you add funds using thesandbox
6. or create a transfer from your bank account to your Monerium IBAN in theproduction
7. environment, you'll receive the corresponding tokens in your Safe. For example, if your Monerium IBAN is associated with the EURe token of your Safe, and you transfer 10 euros, the EURe balance of your Safe will be 10 after the SEPA system completes the transfer.

Add tokens to your Safe using thesandbox . They should be available almost instantly. You can check the balance of your assetshere(opens in a new tab) afterwards.

Once you have tokens, you can create aredeem order(opens in a new tab) to burn them and convert them to fiat. Include a text box for an IBAN and a button to place the order. Add a handler for the button and do the following:

safeMoneriumClient .send ({ amount :

'10' , counterpart : { identifier : { standard :

'iban' , iban :

'The IBAN to send the money to' , } , details : { firstName :

'John' , lastName :

'Doe' , country :

'You ISO country code' , } , } , memo :

'Testing Safe-Monerium integration' })

Once you place your order, it will be sent to the destination account. Two things will occur:

- AsignMessage
- transaction will be proposed to the Safe services indicating in the message the amount of tokens to burn and the destination IBAN. To verify the message to be signed, refer tothe API docs(opens in a new tab)
- .
- You should confirm and execute it using the Safe UI or an alternative method as the transaction data is returned in thesend()
- call.
- An order is placed in the Monerium system, it listens for thesignMessage
- transaction above to be confirmed and executed.

Once the transaction is recorded on the blockchain, the token is burned and the transfer of real money begins, completing the order ð.

1. Listening to events is important for understanding what's happening around. It helps us to create a proper UI in our web page.

You probably want to know when the order is completed. For this you can listen to events using the Monerium API web sockets(opens in a new tab)

Connecting to the socket is easy, just use themoneriumPack instance along with thesubscribe() andunsubscribe() methods.

To subscribe, do this:

const

handler

= (notification) => { console .log (notification) }

moneriumPack .subscribe (OrderState .processed , handler)

The potential states are this ones:

OrderState { placed =

'placed' , pending =

'pending' , processed =

'processed' , rejected =

'rejected' }

If you wish to unsubscribe, you can do:

moneriumPack .unsubscribe (OrderState .processed , handler)

MoneriumPack complete React example

Check a completeexample(opens in a new tab) in thesafe-core-sdk repository. Follow the steps in theREADME.md (opens in a new tab) to run the example and configure the environment variables for the pack following theenv.sample (opens in a new tab) .

Stripe Reference

Was this page helpful?

Report issue