# Build on MiniPay

## Building DApps for MiniPay Wallet

Welcome to the MiniPay wallet integration guide.MiniPay is one of the fastest growing wallets that was built out by Opera on Celo that seeks to create a simple user experience to use DApps. MiniPay is available inside Opera Mini browser on Android phones thereby allowing DApp developers to tap into a distribution of 100M users on integration. This guide will help you build DApps for MiniPay.

### Overview

We'll walk you through the process of integrating the windows.provider object provided by the MiniPay wallet's webview with popular web3 JavaScript libraries.

When your website is loaded inside the MiniPay app's webview, the MiniPay wallet injects a Web3 provider via thewindows object, akin towindow.ethereum . This provider is compatible with a majority of web3 libraries that accept a provider object.

To check if your app is in the context of MiniPay, you can verify it usingwindow.ethereum.isMiniPay .

### Installing MiniPay

MiniPay is available inside Opera Mini on Android phones only. To ensure that your DApp functions as expected within the MiniPay environment, it's crucial to test it inside the MiniPay app. Here's how you can set up and test your DApp:

- Download Opera Mini Beta:
- Start by downloading the Opera Mini Beta application from the PlayStore on your Android phoneDownload Opera Mini Beta
- Access MiniPay through Opera Mini Beta:
- Once you've downloaded the latest version of Opera Mini Beta application, click on the following link on your phone to access MiniPay:Access MiniPay
- Create an Account:
- Before you can test your DApp, you need to create an account on the MiniPay app. Follow the on-screen instructions to set up your account.

### Test your DApp inside MiniPay

1. Open the MiniPay app on your phone and click on compass icon.
2. Click on "Test Page" to open the MiniPay test page.
3. Enter the URL of your DApp and click on "Go".

### Integration with Popular Web3 Libraries

#### 1. Web3.js

Web3.js is a widely-used Ethereum JavaScript API. To utilize thewindows.ethereum withWeb3.js :

const

Web3

=

require ( "web3" ) ;

// Ensure MiniPay provider is available if

( window . ethereum

&&

window . ethereum . isMiniPay )

{ const web3 =

new

Web3 ( window . ethereum ) ; }

else

{ console . error ( "MiniPay provider not detected" ) ; }

**2. Ethers.js**

Ethers.js offers a comprehensive Ethereum wallet implementation and utilities in both JavaScript and TypeScript. Here's how to integrate it with MiniPay:

const

{ ethers }

=

require ( "ethers" ) ;

// Ensure MiniPay provider is available if

( window . ethereum

&&

window . ethereum . isMiniPay )

{ const provider =

new

ethers . providers . Web3Provider ( window . ethereum ) ; }

else

{ console . error ( "MiniPay provider not detected" ) ; }

**3. Wagmi**

Wagmi.js is an emerging library in the Ethereum ecosystem. To integrate it with MiniPay's provider, you can use theuseConnect hook:

import

{ useConnect }

from

"wagmi" ; import

{

InjectedConnector

}

from

"wagmi/connectors/injected" ;

const

{ connect }

=

useConnect ( { connector :

new

InjectedConnector ( ) , } ) ;

useEffect ( ( )

=>

{ connect ( ) ; } ,

[ ] ) ; This code sets up anInjectedConnector and then utilizes theconnect method from theuseConnect hook. TheuseEffect ensures that the connection is established when the page loads.

**4. Viem**

Viem is another library gaining traction. Here's how you can integrate it:

import

{ createWalletClient , custom }

from

"viem" ; import

{ mainnet }

from

"viem/chains" ;

const client =

createWalletClient ( { chain : mainnet , transport :

custom ( window . ethereum ) , } ) ;

const

[ address ]

=

await client . getAddresses ( ) ; In the Viem example, we're creating a wallet client that specifies the chain and a custom transport usingwindow.ethereum . ThegetAddresses method then retrieves the connected addresses.

## Testing Local Development with MiniPay

If you're developing your DApp locally (e.g., onlocalhost:3000 using Next.js or a similar framework), you can usengrok to tunnel the traffic over HTTP. This allows you to test the localhost version in real-time inside MiniPay.

- Install ngrok:
- If you haven't already, install ngrok. You can find instructions on theirofficial website
- .
- Start Your Local Server:
- Ensure your local development server is running. For instance, if you're using Next.js, you might runnpm run dev
- to start your server atlocalhost:3000
- .
- Tunnel Traffic with ngrok:
- In your terminal, run the following command to start an ngrok tunnel:

ngrok http 3000 This will provide you with a public URL that tunnels to your localhost.

- Test in MiniPay:
- Copy the provided ngrok URL and use it inside the MiniPay app to test your DApp.

## Important Notes

- Ensure the "Connect Wallet" button is hidden when your DApp is loaded inside the MiniPay app, as the wallet connection is implicit.
- Always verify the existence ofwindow.provider
- before initializing your web3 library to ensure seamless compatibility with the MiniPay wallet.
- When usingngrok
- , remember that the tunneling URL is temporary. You'll get a new URL every time you restart ngrok.
- Be cautious about exposing sensitive information or functionality when using public tunneling services like ngrok. Always use them in a controlled environment.
- MiniPay currently supports setting thefeeCurrency
- property when runningeth_sendTransaction
- . However, currency support is limited tocUSD

- . More currencies might be supported in future.
- MiniPay only accepts legacy transactions at the moment. EIP-1559 properties won't be considered when handling requests.
- Message signing is not currently supported.
- A developer mode will be available soon, allowing the use of testnet.

Edit this page Previous Verify with Hardhat Next Android Studio Setup