Frontend Development

Developing the frontend of a dApp on Sei involves connecting to wallets, interacting with the blockchain via RPC endpoints, and signing and broadcasting transactions. dApps should choose either EVM or Cosmos for their connection, but can use interoperability features such as precompiles and pointer contracts to support both environments.

Wallet Connection

Connecting to wallets is a crucial step in developing dApps. Here are some recommended libraries for wallet connection, each with its specific advantages:

EVM and EVM RPC dApps:

- Wagmi
- : A React-based library for Ethereum dApps that simplifies wallet connection and interaction. Provides hooks for interacting with Ethereum wallets and contracts for use with modern frontend libraries and frameworks.* Wagmi Documentation(opens in a new tab)
- Viem
- : A lightweight and flexible library for Ethereum development.* Viem Documentation(opens in a new tab)
- Ethers.js
- : A complete and compact library for interacting with the Ethereum blockchain and its ecosystem. Known for its simplicity and extensive functionality.* Ethers.js Documentation(opens in a new tab)

CosmWasm/Cosmos RPC dApps

- CosmosKit
- : A React-based library for Cosmos ecosystem dApps, facilitating wallet connection and interaction. Supports all Sei native wallets as well as cross-chain wallets like Keplr and Leap.* CosmosKit Documentation(opens in a new tab)
- CosmJS
- : A JavaScript library for interacting with Cosmos blockchains, providing tools to handle wallet connections, transactions, and more. Offers comprehensive tools for Cosmos SDK based blockchains.* CosmJS Documentation(opens in a new tab)

RPC Endpoints

dApps need to connect to an RPC provider in order to broadcast transactions or to query the chain. There are many free providers on testnet and devnet, and a few rate limited providers on mainnet. See the RPC providers section for more info and links

@sei-js Helper Libraries

The@sei-js package provides various helper libraries to facilitate interaction with the Sei blockchain. Use@sei-js/cosmjs and@sei-js/proto for Cosmos side interactions, and@sei-js/evm for EVM side interactions.

- /cosmjs
- : Provides helpful wrappers around CosmJS, CosmosKit, and more; specifically tailored for the Cosmos side of Sei.* @sei-js/cosmjs(opens in a new tab)
- /evm:
- A library designed to simplify EVM interaction from Typescript/Javascript, this library includes precompile contracts and helpers for interaction with Wagmi/Viem and Ethers.js.* @sei-js/evm(opens in a new tab)
- /proto
- : A TypeScript typed library for Cosmos and CosmWasm dApps that provides query clients and typed message and response structures to help avoid runtime errors.* @sei-js/proto(opens in a new tab)

Polyfills Warning

When developing frontend applications for the blockchain, it's important to be aware that some libraries may require polyfills, especially when used in browser environments. For instance, theBuffer class and other Node.js-specific features are not natively available in browsers and need to be polyfilled.

If you are using Vite or another rollup based frontend library you can add the following to the entry point of your app.

import { Buffer } from

'buffer' ;

// Polyfill self for browser and global for Node.js const globalObject

=

typeof self !==

'undefined'

? self : global;

Object .assign (globalObject , { process : process , Buffer : Buffer }); If you are using a Webpack based bundling tool you can use the following plugin in you Webpack config.

yarn add -D node-polyfill-webpack-plugin import NodePolyfillPlugin from

'node-polyfill-webpack-plugin' ;

...

// the rest of your webpack config plugins : [... new

NodePolyfillPlugin () , ...] , ... Using proto files (@sei-js/proto)

Tips for New Developers

1. Understanding Gas Fees
2. : Gas fees are crucial for executing transactions on the blockchain. Ensure you understand how they work and how to optimize your contracts to minimize gas usage.
3. Security Practices
4. : Always follow best security practices. Regularly audit your code and stay updated with the latest security vulnerabilities and patches.
5. Testing
6. : Thoroughly test your dApps in different environments to ensure they work correctly. Use testnets and faucets to test transactions without spending real tokens.
7. Documentation
8. : Make use of the extensive documentation available for each library and tool. Good documentation can significantly speed up your development process.
9. Community and Support
10. : Join developer communities, forums, and chat groups. Engaging with other developers can provide valuable insights and help you solve problems more efficiently.

Last updated onMay 23, 2024 Smart Contracts Wallets