Building a Frontend

Introduction

In this tutorial, we'll demonstrate how to set up a frontend project using React. You'll learn how to connect to a Sei wallet and interact with EVM and CosmWasm smart contracts deployed on Sei.

Select one of the tabs below to get started!

EVM CosmWasm In this section, we'll explore Sei's unique interoperability features by building an EVM compatible DApp that interacts with a CosmWasm smart contract. We will useethers.js(opens in a new tab) to build a React app that interacts with a CosmWasm smart contract using the SeiCosmWasm precompile .

Prerequisites

- Complete the tutorial incosmwasm-general
- to deploy a CosmWasm counter contract on our devnet (arctic-1).

Requirements

Before starting, ensure you have:

- Node.js & NPM installed
- One of the Sei wallets listedhere
- The wallet should be funded with sufficient Sei on our devnet (arctic-1). Refer to the section onfaucets
- for instructions on how to get Devnet tokens.

Creating a React Project

Start by creating a new React project using Vite's TypeScript template for streamlined development:

npm

create

vite@latest

my-counter-frontend

--

--template

react-ts This command creates a new folder with a React project using TypeScript. Openmy-counter-frontend in your favorite IDE.

The rest of this tutorial will be in TypeScript. If you're not using TypeScript, you can easily adjust by removing the types.

Installing Dependencies

Installethers , an Ethereum library that facilitates interaction with the Ethereum blockchain:

npm

install

ethers

Defining Contract Addresses and ABI

In this tutorial, we will be using theWasm Precompile to interact with our CosmWasm contract from the EVM. Precompiles (short for Precompiled contracts) are EVM compatible contracts that are built into the chain. The Wasm Precompile is a unique smart contract on Sei that enables EVM clients to query and execute CosmWasm contracts. Refer to the docs oninteroperability for more details about precompiles.

First, import the address and ABI of the CosmWasm precompile from@sei-js/evm .

@sei-js contains NPM libraries for writing applications that interact with Sei. Learn morehere(opens in a new tab) . @sei-js/evm is an npm package that contains useful constants and helpers for interacting with the EVM on Sei.

To install sei-js:

npm

install

@sei-js/evm At the top ofApp.tsx you can then importWASM_PRECOMPILE_ADDRESS ,WASM_PRECOMPILE_ABI . These constants allow us to interact with the Wasm Precompile.

import { WASM_PRECOMPILE_ADDRESS , WASM_PRECOMPILE_ABI , WasmPrecompileContract } from

'@sei-js/evm' ; import { ethers } from

'ethers' ; These values will be used in the app to query and execute a contract.

Connecting to the Wallet and Initializing the Contract

Replace your mainApp component with the following:

App.tsx import { WASM_PRECOMPILE_ADDRESS , SeiChainInfo , getWasmPrecompileEthersV6Contract } from

'@sei-js/evm' ; import { useEffect , useState } from

"react" ; import { BrowserProvider , Contract , toUtf8Bytes , toUtf8String } from

"ethers" ; import

"./App.css" ;

function

App () { const [count ,

setCount] =

useState < string

    (); const [contract ,

setContract] =

useState < Contract

    (); const [isIncrementing ,

setIsIncrementing] =

useState (false);

// TODO: Replace this with your CosmWasm contract address here const

COUNTER_CONTRACT_ADDRESS

=

"sei14hj2tavq8fpesdwxxcu44rty3hh90vhujrvcmstl4zr3txmfvw9sh9m79m" ;

const

fetchCount

=

async () => { if (! contract) { return ; } // Query to get the count on the counter contract const

queryMsg

= { get_count : {} }; const

queryResponse

=

await

contract .query (COUNTER_CONTRACT_ADDRESS , toUtf8Bytes (JSON .stringify (queryMsg))); const { count } =

JSON .parse (toUtf8String (queryResponse)); setCount (count); };

useEffect (() => { fetchCount (); } , [contract]);

const

connectWallet

=

async () => { if (window .ethereum) { const

provider

=

new

BrowserProvider (window .ethereum); const { chainId } =

await

provider .getNetwork (); const

devnetChainId

=

SeiChainInfo . devnet .chainId if (chainId !==

BigInt (devnetChainId)) { alert ("Wallet is not connected to Sei EVM devnet"); return ; }

const

signer

=

await

provider .getSigner (); const

contract

=

getWasmPrecompileEthersV6Contract (WASM_PRECOMPILE_ADDRESS , signer)

setContract (contract); } else { alert ("No EVM compatible wallet installed"); } };

const

incrementCount

=

async () => { if (! contract) { return ; }

setIsIncrementing (true); // Execute message to increment the count on the contract const

executeMsg

= { increment : {} }; const

executeResponse

=

await

contract .execute (COUNTER_CONTRACT_ADDRESS , toUtf8Bytes (JSON .stringify (executeMsg)) , toUtf8Bytes (JSON .stringify ([])) // Used for sending funds if needed); // Wait for the transaction to be confirmed await

executeResponse .wait (); console .log (executeResponse); setIsIncrementing (false); await

fetchCount (); };

return (<> < div

className

"card"

{contract ? (< div

< h1 Count is {count}</ h1

< button

disabled

{isIncrementing} onClick = {incrementCount}> {isIncrementing ?

"incrementing..."

:

"increment" } </ button

</ div

) : (< button

onClick

{connectWallet}>Connect Wallet</ button

)} </ div

); }

export

default App;

Detailed outline ofApp.tsx

State Declarations

- count
- : Holds the current count fetched from the smart contract.
- contract
- : An instance of the ethers Contract object, used for interacting with the blockchain.
- isIncrementing
- : A boolean to manage UI state during contract execution

Effect Hooks

A singleuseEffect hook to fetch the current count whenever the contract state changes, indicating that the contract instance is ready for interaction.

Connecting to EVM Wallet

A function namedconnectWallet that:

- Checks for any EVM compatible wallet extension.

- Establishes a connection to the Ethereum network via the connected wallet, using ethers.js BrowserProvider.
- Verifies the correct network (Sei EVM devnet) by comparing chainId.
- Creates an ethers.js Contract instance with the signer from the wallet, setting it in the contract state for later use.

Fetching Contract Data

A function namedfetchCount that:

- Executes a contract query to get the current count.
- Parses and updates the count state with the response.

Incrementing the Counter

A function namedincrementCount that:

- Sends a transaction to the smart contract to increment the count.
- Waits for the transaction to be confirmed.
- Refetches the count to update the UI with the new value.

To see your app in action, runnpm run dev to spin up a local version of the application. Once you connect your wallet, you should see a counter, as well as a button you can use to increment the counter on the contract.

Congrats on deploying your first interoperable dApp on Sei!

Conclusion

Congratulations on creating a website for querying and executing a smart contract on Sei! Explore more possibilities with your frontend at our@sei-js repo(opens in a new tab) .

Last updated onMay 27, 2024 Installing seid CLI CosmWasm (General)