Coin Flip

Coin Flip is a game where the player tries to guess the outcome of a coin flip. It is one of the simplest contracts implementing random numbers.

Starting the Game

You have two options to start the example:

1. Recommended:
2. use the app through Gitpod (a web-based interactive environment)
3. Clone the project locally.

Gitpod Clone locally https://github.com/near-examples/coin-flip-examples.git If you choose Gitpod, a new browser window will open automatically with the code. Give it a minute, and the front-end will pop up (ensure the pop-up window is not blocked).

If you are running the app locally, you should build and deploy a contract (JavaScript or Rust version) and a client manually.

Interacting With the Counter

Go ahead and log in with your NEAR account. If you don't have one, you can create one on the fly. Once logged in, use thetails andheads buttons to try to guess the next coin flip outcome.

Frontend of the Game

Structure of a dApp

Now that you understand what the dApp does, let us take a closer look to its structure:

1. The frontend code lives in the/frontend
2. folder.
3. The smart contract code in Rust is in the/contract-rs
4. folder.
5. The smart contract code in JavaScript is in the/contract-ts
6. folder.

note Both Rust and JavaScript versions of the contract implement the same functionality.

Contract

The contract presents 2 methods:flip_coin , andpoints_of .

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... See full example on GitHub contract-rs/src/lib.rs loading ... See full example on GitHub

Frontend

The frontend is composed by a single HTML file (/index.html). This file defines the components displayed in the screen.

The website's logic lives in/assets/js/index.js , which communicates with the contract through awallet . You will notice in/assets/js/index.js the following code:

- JavaScript

frontend/index.js loading ... See full example on GitHub It indicates our app, when it starts, to check if the user is already logged in and execute eithersignedInFlow() orsignedOutFlow() .

Testing

When writing smart contracts, it is very important to test all methods exhaustively. In this project you have integration tests. Before digging into them, go ahead and perform the tests present in the dApp through the commandyarn test for the JavaScript version, or./test.sh for the Rust version.

Integration test

Integration tests can be written in both Rust and JavaScript. They automatically deploy a new contract and execute methods on it. In this way, integration tests simulate interactions from users in a realistic scenario. You will find the integration tests for thecoin-flip incontract-ts/integration-tests (for the JavaScript contract) andcontract-rs/sandbox-rs (for the Rust contract).

- JavaScript
- Rust

contract-ts/integration-tests/src/main.ava.ts loading ... See full example on GitHub contract-rs/sandbox-rs/src/tests.rs loading ... See full example on GitHub

A Note On Randomness

Randomness in the blockchain is a complex subject. We recommend you to read and investigate about it. You can start with oursecurity page on it . Edit this page Last updatedonMar 15, 2024 bygarikbesson Was this page helpful? Yes No

Previous Cross Contract Call Next Factory