Final modifications

Let's modify ournew_puzzle method a bit, and demonstrate how a smart contract author might use base64-encoded arguments.

In the previous chapter we had a fairly long NEAR CLI command that called thenew_puzzle , providing it the parameters for all the clues. Having these lengthy parameters on the CLI might get cumbersome. There may be issues needing to escape single or double quotes, and each operating system may wish for a different format on the Terminal or Command Prompt.

We're going to send all the arguments as a base64-encoded string, and make this a bit simpler. For this, we're going to usBase64VecU8 from the SDK .

Base64VecU8 is great for binary payloads What we're doing makes sense, but it's worth noting that it's perhaps more common to useBase64VecU8 when sending binary parameters.

Read moreabout it here . First we'll set up a struct for the arguments we're expecting:

contract/src/lib.rs loading ... See full example on GitHub Then we modify ournew_puzzle method like so:

contract/src/lib.rs loading ... See full example on GitHub We can take our original arguments and base64 encode them, using whatever method you prefer. There are plenty of online tool, Terminal commands, and open source applications likeBoop .

We'll copy this:

{ "answer_pk": "ed25519:7PkKPmVUXcupA5oU8d6TbgyMwzFe8tPV6eV1KGwgo9xg", "dimensions": { "x": 11, "y": 10 }, "answers": [{ "num": 1, "start": { "x": 0, "y": 1 }, "direction": "Across", "length": 12, "clue": "NEAR transactions are more _ *instead of atomic." }, { "num": 2, "start": { "x": 6, "y": 0 }, "direction": "Down", "length": 7, "clue": "In a smart contract, when performing an Action, you use this in Rust." }, { "num": 3, "start": { "x": 9, "y": 0 }, "direction": "Down", "length": 6, "clue": "In docs.rs when you search for the near-sdk crate, these items a considered a what: collections, env, json*types." }, { "num": 4, "start": { "x": 1, "y": 1 }, "direction": "Down", "length": 10, "clue": "A series of words that can deterministically generate a private key." }, { "num": 5, "start": { "x": 1, "y": 3 }, "direction": "Across", "length": 3, "clue": "When doing high-level cross-contract calls, we import this that ends in _contract. When calling ourselves in a callback, it is convention to call it THIS_self." }, { "num": 6, "start": { "x": 0, "y": 8 }, "direction": "Across", "length": 8, "clue": "Use this to determine the execution outcome of a cross-contract call or Action." }, { "num": 7, "start": { "x": 4, "y": 6 }, "direction": "Across", "length": 4, "clue": "You chain this syntax onto a promise in order to schedule a callback afterward." }] } and base64 encode it:

Now we can build and run the new crossword puzzle contract as we have before:

./build.sh

export NEAR_ACCT=crossword.friend.testnet export PARENT_ACCT=friend.testnet near delete NEAR_ACCT PARENT_ACCT near create-account NEAR_ACCT --masterAccount PARENT_ACCT near deploy NEAR_ACCT --wasmFile res/crossword_tutorial_chapter_3.wasm --initFunction new --initArgs '{"owner_id": "'NEAR_ACCT'", "creator_account": "testnet"}' near call NEAR_ACCT new_puzzle '{ "args": "ewogICJhbnN3ZXJfcGsiOiAiZWQyNTUxOTo3UGtLUG1WVVhjdXBBNW9VOGQ2VGJneU13ekZlOHRQVjZlVjFLR3dnbzl4ZyIsCiAgImRpbWVuc2lvbnMiOiB7CiAgICJ4IjogMTEsCiAgICJ5IjogMTAKICB9 }' --accountId NEAR_ACCT Back at the project root (not in thecontract directory) we can run our app and see the new crossword puzzle:

CONTRACT_NAME=crossword.friend.testnet npm run start

Wrapping up

Once you understand cross-contract calls and callbacks and where the logic should go, you can build just about anything on NEAR.

This might be a good time for a reminder that this crossword puzzle, which checks permissions to methods based on a public key, is a bit unusual. It's more common to have simple collections or mappings for allowed users, or utilize theowner_id field we set up. The account and access key system in NEAR is quite powerful, and hopefully this tutorial helps stretch the limits of what's possible, like the seamless onboarding we have with the crossword puzzle.

Again, the final code for this chapter:

https://github.com/near-examples/crossword-tutorial-chapter-3

Happy hacking! Edit this page Last updatedonJan 19, 2024 byDamián Parrino Was this page helpful? Yes No

Previous Cross-contract calls, etc. Next Posts Indexer