Skeleton and Rust Architecture

In this article, you'll learn about the basic architecture behind the NFT contract that you'll develop while following this"Zero to Hero" series. You'll discover the contract's layout and you'll see how the Rust files are structured in order to build a feature-complete smart contract. New to Rust? If you are new to Rust and want to dive into smart contract development, ourQuick-start guide is a great place to start.

Introduction

This tutorial presents the code skeleton for the NFT smart contract and its file structure. You'll find how all the functions are laid out as well as the missing Rust code that needs to be filled in. Once every file and function has been covered, you'll go through the process of building the mock-up contract to confirm that your Rust toolchain works as expected.

File structure

Following a regularRust project, the file structure for this smart contract has:

- Cargo.toml
- file to define the code dependencies (similar topackage.json
-)
- src
- folder where all the Rust source files are stored
- target
- folder where the compiledwasm
- will output to
- build.sh
- script that has been added to provide a convenient way to compile the source code

Source files

File Description approval.rs Has the functions that controls the access and transfers of non-fungible tokens.enumeration.rs Contains the methods to list NFT tokens and their owners. lib.rs Holds the smart contract initialization functions.metadata.rs Defines the token and metadata structure. mint.rs Contains token minting logic.nft_core.rs Core logic that allows you to transfer NFTs between users. royalty.rs Contains payout-related functions. nft-contract ├── Cargo.lock ├── Cargo.toml ├── README.md ├── build.sh └── src ├── approval.rs ├── enumeration.rs ├── lib.rs ├── metadata.rs ├── mint.rs ├── nft_core.rs └── royalty.rs tip Explore the code in ourGitHub repository .

approval.rs

This allows people to approve other accounts to transfer NFTs on their behalf. This file contains the logic that complies with the standard'sapprovals management extension. Here is a breakdown of the methods and their functions:

Method Description nft_approve Approves an account ID to transfer a token on your behalf. nft_is_approved Checks if the input account has access to approve the token ID. nft_revoke Revokes a specific account from transferring the token on your behalf. nft_revoke_all Revokes all accounts from transferring the token on your behalf. nft_on_approve This callback function, initiated duringnft_approve , is a cross contract call to an external contract. nft-contract/src/approval.rs loading ... See full example on GitHubYou'll learn more about these functions in theapprovals section of the Zero to Hero series.

enumeration.rs

This file provides the functions needed to view information about NFTs, and follows the standard'senumeration extension. Method Description nft_total_supply Returns the total amount of NFTs stored on the contract. nft_tokens Returns a paginated list of NFTs stored on the contract regardless of their owner. nft_supply_for_owner Allows you view the total number of NFTs owned by any given user. nft_tokens_for_owner Returns a paginated list of NFTs owned by any given user. nft-contract/src/enumeration.rs loading ... See full example on GitHubYou'll learn more about these functions in theenumeration section of the tutorial series.

lib.rs

This file outlines what information the contract stores and keeps track of. Method Description new_default_meta Initializes the contract with defaultmetadata so the user doesn't have to provide any input. new Initializes the contract with the user-providedmetadata . Keep in mind The initialization functions (new ,new_default_meta) can only be called once. nft-contract/src/lib.rs loading ... See full example on GitHub You'll learn more about these functions in theminting section of the tutorial series.

metadata.rs

This file is used to keep track of the information to be stored for tokens, and metadata. In addition, you can define a function to view the contract's metadata which is part of the standard'smetadata extension. Name Description TokenMetadata This structure defines the metadata that can be stored for each token (title, description, media, etc.). Token This structure outlines what information will be stored on the contract for each token. JsonToken When querying information about NFTs through view calls, the return information is stored in this JSON token. nft_metadata This function allows users to query for the contact's internal metadata. nft-contract/src/metadata.rs loading ... See full example on GitHub You'll learn more about these functions in theminting section of the tutorial series.

mint.rs

Contains token minting logic. Method Description nft_mint This function mints a non-fungible token. nft-contract/src/mint.rs loading ... See full example on GitHub

nft_core.rs

Core logic that allows you to transfer NFTs between users. Method Description nft_transfer Transfers an NFT to a receiver ID. nft_transfer_call Transfers an NFT to a receiver and calls a function on the receiver ID's contract. The function returnstrue if the token was transferred from the sender's account. nft_token Allows users to query for the information about a specific NFT. nft_on_transfer Called by other contracts when an NFT is transferred to your contract account via thenft_transfer_call method. It returnstrue if the token should be returned back to the sender. nft_resolve_transfer When you start thenft_transfer_call and transfer an NFT, the standard also calls a method on the receiver's contract. If the receiver needs you to return the NFT to the sender (as per the return value of thenft_on_transfer method), this function allows you to execute that logic. nft-contract/src/nft_core.rs loading ... See full example on GitHub You'll learn more about these functions in thecore section of the tutorial series.

royalty.rs

Contains payout-related functions. Method Description nft_payout This view method calculates the payout for a given token. nft_transfer_payout Transfers the token to the receiver ID and returns the payout object that should be paid for a given balance. nft-contract/src/royalty.rs loading ... See full example on GitHub You'll learn more about these functions in theroyalty section of the tutorial series.

Building the skeleton

- If you haven't cloned the main repository yet, open a terminal and run:

git clone https://github.com/near-examples/nft-tutorial/ * Next, switch to the1.skeleton * branch and build the contract withyarn * :

cd nft-tutorial git switch 1.skeleton yarn build Since this source is just a skeleton you'll get many warnings about unused code, such as:

Compiling nft_simple v0.1.0 (/Users/dparrino/near/nft-tutorial/nft-contract) warning: unused imports:LazyOption, LookupMap, UnorderedMap, UnorderedSet --> src/lib.rs:3:29 | 3 | use near_sdk::collections::{LazyOption, LookupMap, UnorderedMap, UnorderedSet}; | ^^^^^^^^^^ ^^^^^^^^^ ^^^^^^^^^^^^ ^^^^^^^^^^^^ | = note: #[warn(unused_imports)] on by default ... ... warning: constant is never used: NO_DEPOSIT --> src/nft_core.rs:6:1 | 6 | const NO_DEPOSIT: Balance = 0; | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

warning: nft_simple (lib) generated 50 warnings Finished release [optimized] target(s) in 22.58s ✨ Done in 22.74s. Don't worry about these warnings, you're not going to deploy this contract yet. Building the skeleton is useful to validate that your Rust toolchain works properly and that you'll be able to compile improved versions of this NFT contract in the upcoming tutorials.

Conclusion

You've seen the layout of this NFT smart contract, and how all the functions are laid out across the different source files. Usingyarn , you've been able to compile the contract, and you'll start fleshing out this skeleton in the nextMinting tutorial .

Versioning for this article At the time of this writing, this example works with the following versions:

- rustc:1.75.0
- near-sdk-rs:4.1.1
- NFT standard:NEP171
- , version1.1.0 Edit this page Last updatedonFeb 16, 2024 bygarikbesson Was this page helpful? Yes No

Previous Pre-deployed Contract Next Minting