Integrating FT Payments into an NFT Marketplace

In this tutorial, you'll learn the basics of how an NFT marketplace contract works and how you can modify it to allow for purchasing NFTs using Fungible Tokens. In the previous tutorials, you went through and created a fully fledged FT contract that incorporates all the standards found in theFT standard .

Introduction

Throughout this tutorial, you'll learn how a marketplace contract could work on NEAR. This is meant to be an example and there is no canonical implementation. Feel free to branch off and modify this contract to meet your specific needs.

Using the same repository as the previous tutorials, if you visit themarket-contract directory, you should have the necessary files to complete the tutorial.

File structure

This directory contains both the build script, dependencies as well as the actual contract code as outlined below.

market-contract ├─── Cargo.lock ├─── Cargo.toml ├─── README.md ├─── build.sh └─── src ├─── external.rs ├─── ft_balances.rs ├─── internal.rs ├─── lib.rs ├─── nft_callbacks.rs ├─── sale.rs └─── sale_views.rs Let's start by building both the finished FT contract and the marketplace contract. Make sure you're in the root directory and run the following command in your terminal:

yarn build && cd market-contract && ./build.sh && cd .. This will install the dependencies for the marketplace contract as well as the FT contract. In addition, it will compile them towasm such that yourft-tutorial/out directory looks like this:

ft-tutorial └─── out ├─── contract.wasm ├─── nft-contract.wasm └─── market.wasm Note that there's also a pre-build nft contract wasm file in the directory which you'll use to place the NFTs for sale.

Understanding the contract

The marketplace contract used in this tutorial is a modified version of the contract created at the end of the NFT zero to hero tutorial. If you'd like to learn about the backbone of how the NFTs are put for sale and the process by which they are minted / sold, check out theNFT zero to hero tutorial .

The core functionalities are the same in the sense that both this contract and the marketplace contract built in the NFT zero to hero have the following features:

- Users can put NFTs for sale and specify sale conditions
- Users must pay for storage deposit to put NFTs for sale and they can withdraw the storage at any time
- Users can update the price of an NFT or remove the sale entirely
- Buyers can purchase the NFTs by callingoffer
- .

The only difference is that this marketplace has removed the ability to purchase NFTs forNEAR and instead allows users to buy them with Fungible Tokens. The fungible token is specified when the contract is initialized and only1 type of fungible token can be used to purchase NFTs. You can't, for example, offer 100 Team Tokens for an NFT and 5000 Mike Tokens for another.

In addition, the marketplace doesnot support royalties. This is because FT transfers are less Gas efficient than regular NEAR transfers. In addition, each user would have to be registered and it's much easier to say "hey seller, make sure you're registered before your NFT is sold" rather than enforcing that the seller andall accounts in the payout object are registered. When an NFT is sold, the entire price is sent directly to the seller.

Purchasing Flow

In order to purchase an NFT, the contract utilizes the "transfer and call" workflow that the FT contract provides. The marketplace contract implements theft_on_transfer method that is called whenever someone transfers FTs to the marketplace contract.

The marketplace keeps track of a balance for each user that outlines how many FTs they've sent to the contract. Each timeft_on_transfer is called, the marketplace contract will update the balance for the user. When that user wishes to purchase an NFT, they calloffer and pass in the amount of tokens they're willing to spend. The marketplace will then decrement from their balance and transfer the NFT to the buyer / send the FTs to the seller.

It's important to note that the sellermust be registered in the FT contract before a sale is made otherwise theft_transfer method will panic and the seller won't receive any tokens.

Looking at the Code

Most of the code is the same as what's been outlined in the NFT zero to hero tutorial but we'll go through a refresher in case you're new or have forgotten some of the details.

Main Library File

Starting at thelib.rs file, this outlines what information is stored on the contract as well as some other crucial functions that you'll learn about below.

Initialization function

The first function you'll look at is the initialization function. This takes anowner_id as well as theft_id as the parameters and will default all the storage collections to their default values. Theft_id outlines the account ID for the fungible token that the contract will allow.

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

Storage management model

Next, let's talk about the storage management model chosen for this contract. Users will need to deposit NEAR onto the marketplace to cover the storage costs. Whenever someone puts an NFT for sale, the marketplace needs to store that information which costs NEAR. Users can either deposit a large amount of NEAR so that they never have to worry about storage again or they can deposit the minimum amount to cover 1 sale on an as-needed basis.

You might be thinking about the scenario when a sale is purchased. What happens to the storage that is now being released on the contract? This is why we have a storage withdrawal function. This allows users to withdraw any excess storage that is not being used. Let's go through some scenarios to understand the logic. The required storage for 1 sale is 0.01 NEAR on the marketplace contract.

Scenario A

- Benji wants to list his NFT on the marketplace but has never paid for storage.
- He deposits exactly 0.01 NEAR using thestorage_deposit
- method. This will cover 1 sale.
- He lists his NFT on the marketplace and is now using up 1 out of his prepaid 1 sales and has no more storage left. If he were to callstorage_withdraw
- , nothing would happen.
- Dorian loves his NFT and quickly purchases it before anybody else can. This means that Benji's sale has now been taken down (since it was purchased) and Benji is using up 0 out of his prepaid 1 sales. In other words, he has an excess of 1 sale or 0.01 NEAR.
- Benji can now callstorage_withdraw
- and will be transferred his 0.01 NEAR back. On the contract's side, after withdrawing, he will have 0 sales paid for and will need to deposit storage before trying to list anymore NFTs.

Scenario B

- Dorian owns one hundred beautiful NFTs and knows that he wants to list all of them.
- To avoid having to callstorage_deposit
- everytime he wants to list an NFT, he calls it once. Since Dorian is a baller, he attaches 10 NEAR which is enough to cover 1000 sales. He now has an excess of 9 NEAR or 900 sales.
- Dorian needs the 9 NEAR for something else but doesn't want to take down his 100 listings. Since he has an excess of 9 NEAR, he can easily withdraw and still have his 100 listings. After callingstorage_withdraw
- and being transferred 9 NEAR, he will have an excess of 0 sales.

With this behavior in mind, the following two functions outline the logic.

market-contract/src/lib.rs loading ... See full example on GitHub In this contract, the storage required for each sale is 0.01 NEAR but you can query that information using thestorage_minimum_balance function. In addition, if you wanted to check how much storage a given account has paid, you can query thestorage_balance_of function.

FT Deposits

If you want to learn more about how NFTs are put for sale, check out the NFT zero to hero tutorial. Once NFTs are put for sale, the owner has three options:

- Update the price of the NFT
- Remove the sale from the marketplace

- Wait for somebody to purchase it

In order to purchase NFTs, buyers need to deposit FTs in the contract and call theoffer function. All the logic for FT deposits is outlined in thesrc/ft_balances.rs file. Starting with theft_on_approve function, this is called when a user transfers FTs to the marketplace contract. The logic can be seen below.

market-contract/src/ft_balances.rs loading ... See full example on GitHubOnce FTs are deposited into the contract, users can either withdraw their FTs or they can use them to purchase NFTs. The withdrawing flow is outlined in theft_withdraw function. It's important to note that you should decrement the user's balancebefore calling theft_transfer function to avoid a common exploit scenario where a user spams theft_withdraw . If you were to decrement their balance in the callback function (if the transfer was successful), they could spam theft_withdraw during the time it takes the callback function to execute. A better pattern is to decrement their balance before the transfer and then if the promise was *unsuccessful , revert the balance back to what it was before.

market-contract/src/ft_balances.rs loading ... See full example on GitHub

Purchasing NFTs

Now that you're familiar with the process of both adding storage and depositing FTs on the marketplace, let's go through what you can do once a sale has been listed. Thesrc/sale.rs file outlines the functions for updating the price, removing, and purchasing NFTs. In this tutorial, we'll focusonly on the purchasing flow. If you'd like to learn about the sale objects, updating the price, and removing a sale, check out theNFT zero to hero tutorial .

For purchasing NFTs, you must call theoffer function. It takes annft_contract_id ,token_id , and the amount you wish to offer as parameters. Behind the scenes, this function will make sure your offer amount is greater than the list price and also that you have enough FTs deposited. It will then call a private methodprocess_purchase which will perform a cross-contract call to the NFT contract to invoke thenft_transfer function where the NFT will be transferred to the seller.

market-contract/src/sale.rs loading ... See full example on GitHubOnce the transfer is complete, the contract will callresolve_purchase where it will check the status of the transfer.If the transfer succeeded, it will send the FTs to the seller. If the transfer didn't succeed, it will increment the buyer's FT balance (acting as a refund).

market-contract/src/sale.rs loading ... See full example on GitHub

View Methods

There are several view functions that the marketplace contract exposes. All of these functions are the same as theNFT zero to hero tutorial except for one extra function we've added. In thesrc/ft_balances.rs file, we've added theft_balance_of function. This function returns the balance of a given account.

Testing

Now that youhopefully have a good understanding of how the marketplace contract works and how you can use the powers of the FT standard to purchase NFTs, let's move onto testing everything.

Deploying and Initializing the Contracts

The first thing you'll want to do is deploy a new FT, NFT, and marketplace contract. Start by building and then you can usedev-accounts to quickly deploy.

yarn build && cd market-contract && ./build.sh && cd .. To deploy the FT contract and export an environment variable, run the following command:

export FT_CONTRACT= near create-account FT_CONTRACT --useFaucet near deploy FT_CONTRACT out/contract.wasm Next, you'll deploy the NFT and marketplace contracts.

export NFT_CONTRACT= near create-account NFT_CONTRACT --useFaucet near deploy NFT_CONTRACT out/nft-contract.wasm

export MARKETPLACE_CONTRACT= near create-account MARKETPLACE_CONTRACT --useFaucet near deploy MARKETPLACE_CONTRACT out/market.wasm Check and see if your environment variables are all correct by running the following command. Each output should be different.

echo FT_CONTRACT && echo MARKETPLACE_CONTRACT && echo NFT_CONTRACT An example output is:

dev-1660831615048-16894106456797 dev-1660831638497-73655245450834 dev-1660831648913-23890994169259 Once that's finished, go ahead and initialize each contract by running the following commands.

near call FT_CONTRACT new_default_meta '{"owner_id": "'FT_CONTRACT'", "total_supply":

"10000000000000000000000000"}' --accountId FT_CONTRACT near call MARKETPLACE_CONTRACT new '{"owner_id": "'MARKETPLACE_CONTRACT'", "ft_id": "'FT_CONTRACT'"}' --accountId MARKETPLACE_CONTRACT near call NFT_CONTRACT new_default_meta '{"owner_id": "'NFT_CONTRACT'"}' --accountId NFT_CONTRACT Let's check if each contract was initialized correctly. You can do this by checking the metadata of the FT and NFT contracts:

near view FT_CONTRACT ft_metadata && near view NFT_CONTRACT nft_metadata In addition, you can check the sales of the marketplace contract and it should return 0.

near view MARKETPLACE_CONTRACT get_supply_sales

Placing a Token For Sale

Now that the marketplace and NFT contracts are initialized, let's place a token for sale. Start by creating a new buyer and seller account by running the following command. In this case, we'll create a sub-account of the FT contract.

near create-account buyer.FT_CONTRACT --masterAccount FT_CONTRACT --initialBalance 25 && export BUYER_ID=buyer.FT_CONTRACT near create-account seller.FT_CONTRACT --masterAccount FT_CONTRACT --initialBalance 25 && export SELLER_ID=seller.FT_CONTRACT Check if everything went well by running the following command.

echo BUYER_ID && echo SELLER_ID This should return something similar to:

buyer.dev-1660831615048-16894106456797 seller.dev-1660831615048-16894106456797 The next thing you'll want to do is mint a token to the seller.

near call NFT_CONTRACT nft_mint '{"token_id": "market-token", "metadata": {"title": "Marketplace Token", "description": "testing out the marketplace", "media": "https://bafybeiftczwrtyr3k7a2k4vutd3amkwsmaqyhrdzlhvpt33dyjivufqusq.ipfs.dweb.link/goteam-gif.gif"}, "receiver_id": "'SELLER_ID'"}' --accountId NFT_CONTRACT --amount 0.1 Now you'll need to place the token for sale. This requires paying for storage as well as calling thenft_approve function.

near call MARKETPLACE_CONTRACT storage_deposit --accountId SELLER_ID --amount 0.1 In this case, we'll place the token for sale for10 gtNEAR .

near call NFT_CONTRACT nft_approve '{"token_id": "market-token", "account_id": "'MARKETPLACE_CONTRACT'", "msg": "{\"sale_conditions\":\"10000000000000000000000000\"}"}' --accountId SELLER_ID --amount 0.1 If you now query for the supply of sales again on the marketplace, it should be 1.

near view MARKETPLACE_CONTRACT get_supply_sales In addition, if you query for the sales by the owner ID, it should reflect the10 gtNEAR price.

near view MARKETPLACE_CONTRACT get_sales_by_owner_id '{"account_id": "'SELLER_ID'"}' Expected output:

[{ owner_id: 'seller.dev-1660831615048-16894106456797', approval_id: 0, nft_contract_id: 'dev-1660831648913-23890994169259', token_id: 'market-token', sale_conditions: '10000000000000000000000000' }]

Deposit FTs into the Marketplace

Now that you have an NFT up for sale for10 gtNEAR on the marketplace contract, the buyer needs to deposit some FTs. The first thing you need to do is register both the marketplace contract and the buyer on the FT contract otherwise you won't be able to transfer any FTs.

near call FT_CONTRACT storage_deposit '{"account_id": "'MARKETPLACE_CONTRACT'"}' --accountId FT_CONTRACT --amount 0.1 near call FT_CONTRACT storage_deposit '{"account_id": "'BUYER_ID'"}' --accountId FT_CONTRACT --amount 0.1 After this, you should transfer the buyer some FTs so that they can deposit at least10 gtNEAR . Lets start with 50gtNEAR . Run the following command to send the buyer FTs on behalf of the FT contract owner.

near call FT_CONTRACT ft_transfer '{"receiver_id": "'BUYER_ID'", "amount": "50000000000000000000000000", "memo": "Go Team!"}' --accountId FT_CONTRACT --depositYocto 1 You'll now need to deposit those tokens into the marketplace contract.

near call FT_CONTRACT ft_transfer_call '{"receiver_id": "'MARKETPLACE_CONTRACT'", "amount": "50000000000000000000000000", "msg": "Woooooooo!"}' --accountId BUYER_ID --depositYocto 1 --gas 200000000000000 If you now query for your balance on the marketplace contract, it should be50 gtNEAR .

near view MARKETPLACE_CONTRACT ft_deposits_of '{"account_id": "'BUYER_ID'"}'

Purchasing the NFT

Now that the buyer has deposited FTs into the marketplace and the token is up for sale, let's go ahead and make an offer! If you try to offer more FTs than what you have, the contract will panic. Similarly, if you try to offer lower than the sale price,

the contract will also panic. Since the sale price is10 gtNEAR , let's try to offer20 gtNEAR and see what happens. The expected outcome is:

- The NFT will be transferred to the buyer
- 20 gtNEAR
- will be sent to the seller
- The buyer will have30 gtNEAR
- left to withdraw.

There is one thing we're forgetting, however. We need to make sure that the seller is registered on the FT contract so let's go ahead and do that now.

near call FT_CONTRACT storage_deposit '{"account_id": "'SELLER_ID'"}' --accountId FT_CONTRACT --amount 0.1 Now let's make an offer!

near call MARKETPLACE_CONTRACT offer '{"nft_contract_id": "'NFT_CONTRACT'", "token_id": "market-token", "amount": "20000000000000000000000000"}' --accountId BUYER_ID --depositYocto 1 --gas 300000000000000 If everything went well, you should see 2 events in your terminal. One event is the NFT transfer coming from the NFT contract when the token was transferred from the seller to the buyer. The other event is the FT transfer for when the seller receives their fungible tokens.

Log [dev-1660831638497-73655245450834]: Memo: payout from market Log [dev-1660831638497-73655245450834]: EVENT_JSON:{"standard":"nep171","version":"nft-1.0.0","event":"nft_transfer","data":[{"authorized_id":"dev-1660831638497-73655245450834","old_owner_id":"seller.dev-1660831615048-16894106456797","new_owner_id":"buyer.dev-1660831615048-16894106456797","token_ids":["market-token"],"memo":"payout from market"}]} Receipt: BBvHig5zg1n2vmxFPTpxED4FNCAU1ZzZ3H8EBqqaeRw5 Log [dev-1660831638497-73655245450834]: EVENT_JSON:{"standard":"nep141","version":"1.0.0","event":"ft_transfer","data":[{"old_owner_id":"dev-1660831638497-73655245450834","new_owner_id":"seller.dev-1660831615048-16894106456797","amount":"20000000000000000000000000","memo":"Sale from marketplace"}]} Let's call some view methods to double check if everything went well. First let's check if the seller now has20 gtNEAR .

near view FT_CONTRACT ft_balance_of '{"account_id": "'SELLER_ID'"}' Next, let's check if the buyer has30 gtNEAR left to withdraw.

near view MARKETPLACE_CONTRACT ft_deposits_of '{"account_id": "'BUYER_ID'"}' Finally, let's check if the NFT is now owned by the buyer.

near view NFT_CONTRACT nft_token '{"token_id": "market-token"}'

Withdrawing the Excess Deposits

Now that the buyer purchased the NFT with20 gtNEAR , they should have30 gtNEAR left to withdraw. If they withdraw the tokens, they should be left with a balance of30 gtNEAR on the FT contract.

near call MARKETPLACE_CONTRACT ft_withdraw '{"amount": "30000000000000000000000000"}' --accountId BUYER_ID --depositYocto 1 --gas 300000000000000 If you now query for the buyer's balance, it should be30 gtNEAR .

near view FT_CONTRACT ft_balance_of '{"account_id": "'BUYER_ID'"}' And just like that you're finished! You went through and put an NFT up for sale and purchased it using fungible tokens!Go team

Conclusion

In this tutorial, you learned about the basics of a marketplace contract and how it works. You went through the core logic both at a high level and looked at the code. You deployed an NFT, marketplace, and FT contract, initialized them all and then put an NFT for sale and sold it for fungible tokens! What an amazing experience! Go forth and expand these contracts to meet whatever needs you have. The world is your oyster and thank you so much for following along with this tutorial series. Don't hesitate to ask for help or clarification on anything in our discord or social media channels.Go Team! Edit this page Last updatedonFeb 9, 2024 bygagdiez Was this page helpful? Yes No

Previous Transferring FTs Next Minting NFTs