BatchNFTMinter Technical Documentation

Contract: BatchNFTMinter.sol

Background: Hedgey Finance is the group that develops token tools for DAOs to utilize their treasury tokens along various life cycle events of the DAO, from inception to maturity. A DAO, generally known as Decentralized Autonomous Organization, will be used generally as a term to describe a group of people in the "Web3" ecosystem, namely building products in the Ethereum and similar EVMs Blockchains. DAOs often have a Token, generally an ERC20 standardized Token, that represents some form of governance, purpose, or just general community fun that is associated with the DAO itself. DAOs use these tokens in many different ways, and this specific protocol is targeted at assisting DAOs responsibly distribute tokens from their treasury to contributors and investors via token locking mechanism. This contract is a periphery contract that allows DAOs to batch together large minting sets, as large as 100 NFTs have been minted on mainnet in a single transaction.

Why an NFT?

The locking of tokens inside an NFT has many advantages of the standard token locking mechanisms. Firstly, the NFT contract can store different tokens, and store tokens for more than one user, unlike the standard vesting contracts. Additionally, the NFT, being owned by a wallet, can be used in other cases where the NFT primitive is used such as; voting with the locked tokens, borrowing against them, and of course selling the NFTs in a secondary OTC marketplace while the tokens remain locked. As a fun element, the Hedgeys contract adds in custom artwork for each NFT which gives them a fun element that owners can view and feel ownership of beyond a simple vesting or locking contract.

Functional Overview

Purpose: The purpose of the BatchNFTMinter.sol contract is to make it easy for DAOs to mint multiple NFTs to distribute locked tokens to a group of contributors or investors in a single transaction. As DAOs often require on-chain governance proposals to distribute any tokens from the treasury, the purpose of this is to alleviate the necessity for multiple proposals to distribute tokens to a single group in a single transaction. For further reading and understanding on the core FuturesNFT.sol and Hedgeys NFTs, please see the Hedgeys FuturesNFT Technical Documentation contained in this repository folder.

Roles: There is just one role, the Minter. The minter's role is solely to mint many NFTs in a single batch transaction. They will call a single function that is batchMint, and input the necessary details of the recipients (ie "holders"), the token they are sending, how much tokens they are sending to each holder, and the specific unlockDates for each token locked NFT.

Functions: There are two external writable functions, though they are essentially the same. Both are called batchMint, but we will illustrate the two separate use cases for each one.

1. **batchMint** (*without mintType)*
   This function is called by the Minter to mint as many NFTs as they wish, given a block size limit (which we estimate for mainnet is about 105 NFTs). The minter will need to know the address of the FuturesNFT contract, as there are at least two versions at the time of this writing that the minter can interact with and generate NFTs for. The minter will input an array of the holders (recipients of the NFTs), an array of the amounts of tokens each nft will lock, an array of the unlock dates when the NFTs will be unlocked, and of course the token address for the ERC20 token that will be locked inside each NFT.
2. **batchMint**(*with mintType*)
   All of the params and types are the same for this function, with the addition of a final parameter of 'mintType'. This function is generally called via a dedicated UI interaction on the Hedgey dApp interface, or via another partner UI integration. The purpose of this is for tagging in database(s) the NFTs that are minted so that more detailed analytics can be performed, as well as differentiated user experiences can be created based on the mint type that tags each hedgey in the metadata.

Technical Overview

Contract Dependencies & Imports

1. Library **TransferHelper.sol**
   Purpose: Assist with transferring tokens between users and contracts. This is a helper library that ensures the correct amount of tokens is transferred and that the before and after balances match with what is the expected outcome. Additionally the library checks to ensure sufficient balances are owned by the transferrer prior to attempting to transfer the tokens.

2. Interface **INFT.sol**
   Purpose: Assist with locking tokens into a FuturesNFT contract, which mints an NFT and locks tokens with a precise unlock date, by calling the createNFT function on the FuturesNFT.sol contract.

Global Variables

There are no global variables in this very basic contract

Write Functions

1. *Function* **batchMint**(*address* nftContract, *address[] memory* holders, *address* token, *uint256[] memory* amounts, *uint256[] memory* unlockDates) *public*
   This batchMint function is called to mint several Hedgeys NFTs in a single transaction. It is required that the array of the holders, amounts and unlockDates are the same length,

and that all of the amounts are greater than 0 and that all of the unlockDates are in the future. This prevents unwanted failures / reversions at the base FuturesNFT contract call level when the createNFT function is called. This function will calculate the sum of all the amounts and performed the prior mentioned checks, then pull in the total token amount from the minter (msg.sender). Subsequently the contract will increase the allowance of the token with the nftContract as the spender so that it will not fail when next it iterates through the array of the amounts and mints an individual nft for each row in the array. It is assumed that the holders, amounts, and unlockDates arrays are ordered in such a way that the index of each item in each array is associated with eachother, in other words the second index in the holders array should receive the amount in the second index of the amounts array, and be locked based on the unlockDate in the second index of the unlockDates array. More details on the inputs can be described as follows:
address nftContract: This is the address of the nftContract, aka the FuturesNFT (Hedgeys NFTs) that the batchminter contract will point at, and mint NFTs from. It will call the createNFT function to the nftContract for each individual item in the batch, minting an NFT with each row iteration.

address[] memory holders: This is the array of holders, ie recipients of the NFTs. Each holder will receive to their address a Hedgey NFT representing locked tokens

address token: The token is the address of a standard ERC20 token that will be used for all of the batch minting – note that only a single token address can be used as the total amount of the same token is pulled into this contract

uint256[] memory amounts: Amounts is the array of the amounts of the tokens that will be locked into each NFT for each recipient. All amounts must be greater than 0

uint256[] memory unlockDates: unlockDates is the array of dates in which the locked tokens will become unlocked. All of the unlockDates must be in the future compared with the current block timestamp.

2. *Function* **batchMint**(*address* nftContract, *address[] memory* holders, *address* token, *uint256[] memory* amounts, *uint256[] memory* unlockDates, *uint256* mintType) *public*
This function is the same as the previous batchMint function with the exception of the addition of the mintType parameter. This parameter is used internally by Hedgey Finance team for tagging NFTs based on various UIs, integrations, and other parameters based on the SDK that allows for specialized user experiences and analytics.