Hedgey Batch Vester Documentation
Smart Contract(s): BatchVester.sol;

1.0 Overview & Purpose:

Background: Hedgey Finance is a group of builders that develops token tools for treasury managers of crypto companies and decentralized organizations (DAOs). DAOs in this context have a token, ERC20 standard, which represents governance, ownership or utility over the DAO and its ecosystem. Before and or after a token is launched by a DAO, they will issue tokens to their employees and investors, and most often those tokens that are granted will contain some legal lockup arrangement, where the tokens cannot be sold or claimed until after the end of this lockup period. Currently many DAOs keep these token vesting and lockup plans on legal documents and excel sheets, but are not enforceable in any way on-chain. Hedgey's StreamvestingNFT smart contract aims to provide decentralized tools and infrastructure to support the token vesting for DAOs, bringing the off-chain agreements on-chain, in a transparent, trusted and decentralized tool.

As with any plan, we imagine that a DAO has a group of people, or even the entire DAO itself, that have predefined and determined the token vesting and unlock schedules for those beneficiaries, and now just need a simple and elegant tool to generate those plans on-chain and wish for a way to view them as necessary.

The BatchVester is a contract that acts as a simple tool in addition to the StreamvestingNFT contract to mint several vesting NFTs in a single transaction. This is highly useful for a team that is initiating their vesting plan for their contributors or employees and can do so in a bulk batch fashion.

Functional Requirements

The purpose is to help automate token unlocks and token vesting for DAOs. This contract helps create those unlocking and vesting plans in a single batch transaction. This contract can interface with either the StreamVestingNFT contract. .

1.1 Roles:

a. **Vesting Plan creators**
   Generally the plan creator is the DAO or executive, finance, or HR team responsible for employee compensation or investor vesting. The team will establish a plan for the

investor or employees, and then use the function (with the Hedgey Application) to create the on-chain vesting schedules for each beneficiary. The creators are the holders of the DAO tokens, and when setting up the on-chain plans via smart contract interaction, will provide the details of each plan to the smart contract - whereupon tokens are pulled from their wallet or DAO address, and delivered to the underlying StreamVestingNFT contract for protocol escrow.

1.2 Functions and Features

There are two versions of vesting plans, one that has a token lockup period, where tokens are locked even after they are vested, and the more common version where tokens are immediately redeemable as soon as they are vested. Lockup periods are more relevant for DAOs launching their tokens, who want to enforce a global lockup period for employees regardless of their vesting dates, and thus the contract accommodates for both styles of vesting plan.

1. **createBatch**
   This function creates a batch of several vesting NFTs, taking inputs in arrays for each of the underlying vesting plans; the recipients, token amounts, start dates, cliff dates, vesting rates, and the token address, and importantly the vesting administrator address. It then iterates through the arrays and creates a vesting NFT per each item in the set of arrays. It does not have the lockup period. This function will transfer tokens from the plan creator (msg.sender) to the StreamVestingNFT contract, and each recipient will receive an NFT representing their vesting plan tokens.
2. **createLockedBatch**
   This function works similarly to the createBatch, except that it adds the additional functionality of the lockup period. So it also iterates through an array of all of the same fields of recipients, amounts, start dates, cliff dates, vesting rates, token address and vesting administrator, plus it takes an array for the lockup dates (unlocks), and whether vested but locked tokens may be transferable when wrapped inside another NFT.

1.3 Tokenomics & Fees

Hedgey does not have a governance token as of this product development, and so there are no tokenomics or fees associated with the tool. It is a free to use open tool built for the betterment of the crypto and web3 ecosystem.

Technical Requirements

2.0 Smart Contract Architecture Overview

The BatchVester.sol contract is a simple tool that allows a vesting plan creator to create multiple vesting plans for their beneficiaries in a single transaction. It leverages the power of the StreamVesting smart contract, interfacing with them to call the createNFT function or createLockedNFT function. The contract loops over the array of the inputs of the vesting plan data for each recipient, and then calls the createNFT or createLockedNFT function at the address of the StreamVestingNFT contract.

2.1 File Imports and Dependencies

```
import './interfaces/IVestingNFT.sol';
import './libraries/TransferHelper.sol';
```

- The smart contract imports the interface necessary to interact with the StreamVestingNFT, being the IVestingNFT.sol interface.
- The contract imports a TransferHelper.sol library that assists the contract transferring tokens from the msg.sender, and then increasing the allowance of the StreamVestingNFT contract so that it can pull the necessary tokens into the contract and mint each recipient a streaming NFT.

2.2 Global Variables

There are no global variables.

2.3 Write Functions

A vesting NFT is defined by the following parameters:

1. Recipient
2. Token address
3. The total amount of tokens locked inside the NFT
4. The start date when the tokens begin to vest or unlock
5. An optional cliff date, before which tokens cannot be redeemed
6. The rate per second which the tokens linearly vest / unlock
7. The vesting administrator address
8. The unlock date (for a lockup period)
9. A boolean to determine if vested but locked tokens, when wrapped in a new streamingNFT during revocation, are transferable or not.

Thus the BatchVeser inputs an array of these values to batch create several of these vesting NFTs at the same time. Each index in the array, as detailed below, represents a single vesting NFT, so it is important to ensure that the index of each item within each array matches perfectly together, ie the inputs at index 3 in each of the arrays correspond to a single vesting NFT, and are thus critical to keep aligned. The arrays also must be of the same length or the function will fail.

There are two versions of the external createBatch function:

createBatch (version 1)

```
function createBatch(
    address vester,
    address[] memory recipients,
    address token,
    uint256[] memory amounts,
    uint256[] memory starts,
    uint256[] memory cliffs,
```

```
    uint256[] memory rates,
    address vestingAdmin
  ) external
```

1. **Address vester**: the address of the StreamVestingNFT contract
2. **Address[] memory recipients**: the array of addresses of each recipient who will receive a vesting NFT
3. **Address token**: the address of the ERC20 token that will be locked and vesting in the NFTs
4. **Uint256[] memory amounts**: the array of uint256 amount that represents the total amount of tokens to be locked for each recipient.
5. **Uint256[] memory starts**: the array of unix time stamp dates that record the start date when the vesting stream begins
6. **Uint256[] memory cliffs**: the array of unix time stamp dates that record the cliff date for each vesting stream
7. **Uint256[] memory rates**: the array of rates of how the vesting tokens unlock. Each rate defines a per second rate in which tokens
8. **Address vestingAdmin**: the very import address of the vesting administrator who is responsible for administering and revoking the vestingNFTs

createBatch (version 2)

```
function createBatch(
    address vester,
    address[] memory recipients,
    address token,
    uint256[] memory amounts,
    uint256[] memory starts,
    uint256[] memory cliffs,
    uint256[] memory rates,
    address vestingAdmin,
    uint256 mintType
  ) external
```

The only difference between this function and the first, is the additional **uint256 mintType** field. This is an internal field that is called when using the Hedgey front end application. The field is picked up in an event and stored in the Hedgey database, and used for certain special visualizations on the front end. It is simply an identifier that Hedgey maintains internally for this extra usage, and has no technical or mechanical impact to the functionality of the smart contract itself.

The other pair of functions are for creating the locked batch, which takes the additional parameters of the unlockDates, as well as the transferability boolean.

createLockedBatch (version 1)

```
function createLockedBatch(
    address vester,
    address[] memory recipients,
    address token,
    uint256[] memory amounts,
    uint256[] memory starts,
    uint256[] memory cliffs,
    uint256[] memory rates,
    address vestingAdmin,
    uint256[] memory unlocks,
    bool transferableNFTLocker
  ) external
```

1. **Address vester**: the address of the StreamVestingNFT contract
2. **Address[] memory recipients**: the array of addresses of each recipient who will receive a vesting NFT
3. **Address token**: the address of the ERC20 token that will be locked and vesting in the NFTs
4. **Uint256[] memory amounts**: the array of uint256 amount that represents the total amount of tokens to be locked for each recipient.
5. **Uint256[] memory starts**: the array of unix time stamp dates that record the start date when the vesting stream begins
6. **Uint256[] memory cliffs**: the array of unix time stamp dates that record the cliff date for each vesting stream
7. **Uint256[] memory rates**: the array of rates of how the vesting tokens unlock. Each rate defines a per second rate in which tokens
8. **Address vestingAdmin**: the very import address of the vesting administrator who is responsible for administering and revoking the vestingNFTs
9. Uint256[] memory unlocks: the array of unlock dates that define the lockup period for each vesting schedule
10. **Bool transferableNFTLocker**: whether or not the NFT Locker that contains the vested but locked tokens is transferable.

createLockedBatch (version 2)

```
function createLockedBatch(
    address vester,
    address[] memory recipients,
    address token,
    uint256[] memory amounts,
    uint256[] memory starts,
    uint256[] memory cliffs,
```

```
    uint256[] memory rates,
    address vestingAdmin,
    uint256[] memory unlocks,
    bool transferableNFTLocker,
    uint256 mintType
  ) external
```

Similar to the version 2 of the createBatch function, this is the same as the createLockedBatch function but with the additional mintType parameter used internally by Hedgey's applications.