Hedgey StreamVesting NFTs
Smart Contract(s): StreamVestingNFT.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 contributors, and most often those tokens that are granted will contain some legal vesting arrangement. This is common in the traditional startup space with equity plans, and for DAOs a token vesting plan is established for employees and early investors. Currently many DAOs keep these vesting plans on legal documents and excel sheets, but are not enforceable in any way on-chain. Hedgey's StreamVestingNFT protocol aims to provide a decentralized tool and infrastructure to support the token vesting plans 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 schedules for those beneficiaries, and now just need a simple and elegant tool to generate those plans on-chain and administer them as necessary.

Functional Requirements:

The purpose of this tool is to automate and effectively manage the token vesting schedules and plans of DAOs. Unique to Hedgey's tool, we utilize the backbones of NFTs (ERC721) to create a vesting plan for each participant that is defined as an individual and unique NFT. Each NFT in the collection represents a single and unique vesting schedule between the DAO vesting administrator and the vesting plan beneficiary (the holder of the NFT). The NFT backbone empowers additional abilities to the contract, such as greater transparency, managing multiple vesting schedules at a time, and most importantly adding in the ability for beneficiaries to vote or delegate their votes based on the tokens that are contained in their vesting plan. The voting attribute is an addition that DAOs must actively select to use in their

snapshot or other voting mechanisms; it is an additional optional attribute that no other vesting contracts contain, but not one that is an obligation to utilize.

Tokens vest in a linear schedule, vesting a precise amount every second, as defined by the vesting schedule. When a plan beneficiary wants to claim some or all of their vested tokens, they will interact with the smart contract to redeem whatever amount of tokens has been vested. Vesting schedules may contain a single cliff date, where any tokens prior to the cliff become fully vested in a chunk on the cliff date. Additionally, the vesting schedule can define a mandatory lockup period, using a single unlock date, whereby even fully vested tokens cannot be claimed until the lockup period has expired, i.e. past the unlock date.
NOTE: These vesting NFTs are not transferable. They cannot be moved to different wallets once minted.

User Interface Overview:

In addition to the smart contracts, the Hedgey team also builds a beautiful front end interface for users to interact with and perform all of the smart contract functions, as well as view their streaming balances and administer them. The application relies on a back end database that stores all of the information from the smart contract each time a transaction occurs, storing the event values and several other key pieces of information. That information is then used to display current information for users about their balances, as well as enable them to properly perform smart contract calls. Because Hedgey's front end applications rely on a database for streamlined viewing, storing initial data in the smart contract storage is unnecessary, and so the contract data is streamlined for purpose, while the application is built to store all previous states and variables for historical data visualization.

1.1 Roles:

a. **Contract Admin**
   Contract Admin is the deployer of the smart contract, who has two primary responsibilities; 1) deploying the smart contract and verifying the authenticity of it and 2) adding a baseURI string to link the NFTs to the metadata that provides an easy way to visualize the data contained in the vesting schedule for front end applications.
b. **Vesting Plan Creators**
   Generally these are the DAO in general or its executive / HR team. This group will generate and agree upon the overall token vesting plan and then subsequent individual vesting schedules for employees and other beneficiaries. The creators are responsible for creating the on-chain plans via creation of NFTs that represent each unique vesting schedule, which requires them to deliver tokens from the DAO treasury to the StreamVestingNFT smart contract.
c. **Vesting Plan Administrator**
   The Plan Administrator is generally a group of individuals, the Plan Creators, or the collective DAO, that are responsible for revoking vesting plans when beneficiaries are no longer part of the DAO (ie fired or quit). The administrators have a huge responsibility, because they can revoke the vesting NFT from a beneficiary at any time before it becomes fully vested. When an NFT is revoked, it will result in the unvested tokens

being delivered to the vesting administrator, and any un-claimed but vested tokens will be delivered to the beneficiary. Administrators can revoke a single NFT, or they can revoke multiple at the same time, which is useful if employees have received multiple grants and upon termination multiple vesting NFTs need to be revoked.

d. **Vesting Plan Beneficiaries**
   Vesting plan beneficiaries are the end employee who is receiving the vesting tokens from the DAO. The cannot transfer their schedule to anyone else. The beneficiary can claim (redeem) any vested tokens, subject to the lockup period, which will pull tokens from the NFT contract and deliver them to the beneficiary wallet. The beneficiary can also delegate the tokens locked inside their vesting plan to an external wallet, specifically for snapshot voting strategies implemented by the DAO. Beneficiaries can redeem their tokens individually, as a group, or they can redeem and claim all tokens that are vested across all of their vesting NFTs.
e. **NFT Delegatee**
   In the circumstance where a DAO has elected to setup snapshot voting based on tokens locked inside of a vesting NFT, the beneficiary may want to delegate the votes to a separate wallet (often a 'hot wallet') that is used to participate in the voting. If a delegatee has been delegated from the beneficiary of the NFT, then they can utilize the locked token balances of any NFTs delegated to them to vote and participate in the snapshot voting. There are no smart contract interactions that they can write, only read functions to check how many locked tokens have been delegated to them.

1.2 Functions and Features

1. **updateBaseURI**
   This is a Contract Admin Only function, and used to link a URI to the NFT collection to easily represent the NFT data on-chain to off-chain metadata for easier application visualizations.
2. **deleteAdmin**
   This is a Contract Admin Only function, which deletes the administrator after they have completed the linking of the URI to the NFT collection for metadata purposes.
3. **createNFT**
   This function is what creates the basic vesting schedule. It is performed by the plan creator, who will input all of the data points surrounding the vesting schedule into the function, and create an NFT that is minted to the beneficiary representing their vesting schedule. This function will pull the DAO tokens from the creator's balance and transfer them to the NFT smart contract. This function does not allow for the optional lockup period, it will create a vesting plan that does not include the lockup period. The creator should know when they are creating the following parameters:
   A. beneficiary / recipient / holder of the NFT
   B. address of the DAO token
   C. amount of total tokens in the entire vesting schedule
   D. the start date when the plan starts vesting (can be back dated, forward dated, or now)
   E. An optional cliff date when some of the tokens vest on a single cliff date

F. the per second rate which the tokens vest upon the start date
G. the vesting administrator who is responsible for revoking the tokens, should the DAO and beneficiary part ways.

4. **createLockedNFT**
   This function creates a slightly more complex vesting schedule. This is also performed by the plan creator, inputting all of the data above the createNFT, with two additional parameters:
   H. the lockup period (unlock date), a date set when vested tokens become unlocked
   I. transferableNFTLockers - this is a field that is defined where IF an NFT is revoked and there are some vested tokens that are still subject to the lockup period (ie unlockDate is still in the future), then another single unlock date NFT is minted to the beneficiary. This NFT minted to the beneficiary contains the fully vested, but still locked tokens, and has the ability to either be transferable or not transferable.
5. **redeemNFTs**
   This function is used by the beneficiary. This is the core function that the beneficiary will call to redeem some or all of the tokens that have vested. They can enter in multiple NFTs to the function to redeem tokens from more than one vesting NFT. If all of the tokens have vested, then the NFT will be burned when this function is called. If there are no tokens vested for the NFT provided, then nothing is redeemed. Tokens are transferred from the NFT contract to the beneficiary wallet address.
6. **redeemAllNFTs**
   This function is used by the beneficiary to redeem all of their NFTs. This will pull all of the vesting NFTs the beneficiary owns, and only redeem any tokens that have vested (ie ignore any schedules with 0 vested tokens). If the tokens are fully vested, then the NFT will be burned.
7. **revokeNFTs**
   This function is the core function used by the vesting administrator. It will take a single or multiple NFTs as inputs. This function revokes the vesting NFTs - burning the NFTs and delivering any un-vested tokens to the vesting admin and any vested tokens to the beneficiary. If there is a lockup period with an unlockDate in the future, then the vested tokens will be transferred to another streaming NFT contract (StreamingNFT.sol) where the tokens have a single unlock date when the lockup period ends, and the beneficiary can then claim the unlocked tokens by redeeming that NFT.
8. **delegateTokens**
   This function is used by the beneficiary who wants to delegate the tokens vesting inside their NFT to another wallet address used for snapshot voting. The function can take a single or multiple NFTs as inputs to delegate to another wallet.
9. **delegateAllNFTs**
   This function is used by the beneficiary who wants to delegate all of their NFTs to a single wallet address for the snapshot voting purposes.

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 StreamVestingNFT.sol smart contract leverages the backbones of the ERC721 NFT standard contract, with the addition of the Enumerable extension and a custom Delegate extension. This NFT backbone allows the smart contract to actively manage multiple vesting schedules within the same smart contract - allowing for better and more efficient management by the vesting administrators and the beneficiaries. The contract holds tokens that are vesting in the smart contract as escrow - pulled to the contract when new vesting NFTs are created. When beneficiaries redeem NFTs or vesting administrators revoke NFTs, the tokens in the contract are transferred out to the beneficiary, or to the admin, or both based on what has been vested and what remains unvested. The contract combines an NFT with a special struct that stores all of the data points of each unique vesting schedule - each vesting schedule can be described with a single integer identifier, that maps to the struct that contains all of the necessary information about the vesting schedule, such as the recipient, token to be locked, total amount vesting, start date of vesting schedule, cliff date, rate that tokens vest per second, and a vesting admin.

2.1 File Imports and Dependencies

```
import '@openzeppelin/contracts/token/ERC721/ERC721.sol'
import '@openzeppelin/contracts/utils/Counters.sol';
import './ERC721Delegate/ERC721Delegate.sol';
import '@openzeppelin/contracts/security/ReentrancyGuard.sol';
import './libraries/TransferHelper.sol';
import './libraries/StreamLibrary.sol';
import './interfaces/IStreamNFT.sol';
```

- The smart contract imports the ERC721 standard from an open zeppelin contract.
- It imports counters for the use of assigning a uint to the NFT token ID and mapping the token ID to the struct that contains the vesting data in storage.
- It imports the a custom ERC721Delegate.sol contract, which is the ERC721 Enumerable extension - with the addition that NFTs can be individually delegated. The Delegate contract mirrors the logic of the Enumerable extension for delegated tokens.
- It imports Reentrancy guard to ensure security that functions adjusting the token balances of the smart contract can only be called once
- It imports a transfer helper; which assists transferring tokens from address to the NFT contract, and withdrawing from the NFT contract to other addresses - ensuring the amount that was expected to be transferred was successfully transferred.
- It imports a streaming library that helps with calculating the balances of an NFT at a given moment in time, the minimum of two integers, and calculating the end date of an NFT vesting schedule.

- It imports IStreamNFT interface for the sole purpose of creating a locked streaming NFT when an NFT is revoked - but some amount is vested but subject to the lockup period. The lockup period of the vested portion of the tokens is enforced via minting a streamingNFT with a single unlock date based on the expiry of the lockup period.

2.2 Global Variables

```
 string private baseURI;
```

This is the string representation of the baseURI that is linked to the metadata representation of the NFTs. This is generally the following:
`https://nft.hedgey.finance/${networkName}/${contractaddress}/`

```
address private admin;
```

The address of the contract administrator who is responsible for setting the baseURI after deployment. This is needed because the baseURI uses the contractAddress, and as we only get this after deployment, the admin is necessary for this one and only task after deployment.

```
mapping(bool => address) private nftLockers;
```

This is a mapping of two nftLockers, true is mapped to the StreamingHedgeys contract, and false is mapped to the StreamingBoundHedgeys contract.

```
struct Stream {
    address token;
    uint256 amount;
    uint256 start;
    uint256 cliffDate;
    uint256 rate;
    address vestingAdmin;
    uint256 unlockDate;
    bool transferableNFTLocker;
  }
```

Stream is the storage in a struct of the tokens that are currently being vested. This struct in storage represents all of the data points of a given unique vesting schedule, and is mapped to a specific NFT Token ID.

- token is the token address being streamed
- amount is the total amount of tokens in the stream, which is comprised of the balance and the remainder
- start is the start date when token stream begins, this can be set at anytime including past and future
- cliffDate is an optional field to add a single cliff date prior to which the tokens cannot be unlocked
- rate is the number of tokens per second being streamed

- vestingAdmin is the address of a vesting administor that can revoke NFTs and pull unvested tokens to its wallet any time.
- unlockDate is the date set for an optional lockup period, that the vested tokens may be subject to
- n the case where there is an unlockDate, this is enforced via vested tokens being transferred to and minting an NFT of either the StreamingHedgeys or StreamingBoundHedgeys - determined by the true or false of this param

```
mapping(uint256 => Stream) public streams;
```

This is the mapping that connects the storage struct Stream to the NFT Token ID, via this public global variable streams.

2.3 Contract Admin Only Functions

```
function updateBaseURI(string memory _uri) external
```

This is the function called by the admin to update the baseURI after deployment.

```
function deleteAdmin() external
```

This function can be called by the admin after updating the baseURI to delete itself from storage, thus not allowing any further changes to the baseURI.

2.4 Read Functions

```
function streams(uint256 tokenId)
    external
    view
    returns (
      address token,
      uint256 amount,
      uint256 start,
      uint256 cliffDate,
      uint256 rate,
      address vestingAdmin,
      uint256 unlockDate,
      bool transferableNFTLockers
    );
```

Streams takes a single input of the tokenID, and pops out with the vesting schedule details of the struct held in storage. This public function is generated via the automated getter functions from the streams variable being a public global variable.

```
function streamBalanceOf(uint256 tokenId) external view returns (uint256 balance, uint256 remainder);
```

This function will take the tokenID of an NFT and return the balance and remainder. The balance is the amount that has vested for the holder, and the remainder is what is left unvested.

The combination of the balance and the remainder is equal to the total amount held in the storage struct.

```
function getStreamEnd(uint256 tokenId) external view returns (uint256 end);
```

This function takes a unique NFT token ID and will return the end date of the vesting schedule - the date when the holder has fully vested all of the tokens.

```
function lockedBalances(address holder, address token) external view returns (uint256);
```

This function takes the address of a holder, and the address of a token, to return the entire balance that is locked inside of the smart contract on behalf of the holder. This function will aggregate all of the vesting token balances (both vested and unvested) across all NFTs that share the same underlying token address, and return the sum of all of them. This is useful for snapshot voting purposes.

```
function delegatedBalances(address delegate, address token) external view returns (uint256);
```

This function is similar to the lockedBalances function, except taking a delegate address instead of the beneficial holder of the NFT. It will aggregate the unvested and vested balances that are tied to NFTs that have been delegated to a single delegate address, based on the token that is locked in each NFT, and return the sum of the amounts. This is very useful for snapshot voting.

```
function delegatedTo(uint256 tokenId) external view returns (address);
```

This function takes a tokenId and returns the address of a delegate that the NFT has been delegated to.

```
function balanceOfDelegate(address delegate) external view returns (uint256);
```

This function takes in the address of a delegate and returns the total count of NFTs that have been delegated to that address.

```
function tokenOfDelegateByIndex(address delegate, uint256 index) external view returns (uint256);
```

This function is used for enumeration purposes, it will index all of the NFT token IDs that have been delegated to a specific address, so that we can easily count and aggregate those NFTs for a single delegate wallet address.

2.5 Write Functions

```
function createNFT(
    address recipient,
    address token,
```

```
    uint256 amount,
    uint256 start,
    uint256 cliffDate,
    uint256 rate,
    address vestingAdmin
  ) external;
```

This function is the function to create a vesting NFT without a lockup period. This function will transfer tokens from the msg.sender to the NFT contract. It takes the following arguments:

1. **Address recipient:** The address of the beneficiary, the recipient of the vesting NFT.
2. **Address token:** The address of the token that will be vesting
3. **Uint256 amount:** The total amount of tokens to be vested across the entire schedule
4. **Uint256 start:** The start date when tokens begin vesting. This can be back dated, or future dated, or use the current time stamp. This is unix time representation of seconds.
5. **Uint256 cliffDate:** Cliff date when tokens vest in a single clump, on a cliff. If this date is set at the same as the start or before it, then there is no cliff date - this only applies if the amount is set greater than the start date, also represented as Unix seconds timestamp.
6. **Uint256 rate:** The rate at which tokens vest per second. If the rate is set to 5, then 5 tokens vest per second, meaning over the course of 1 minute 300 tokens have vested. This is base version of token vesting, meaning it does not account for the decimals of the token (so if 1 token = 1ee18, then a rate of 1ee18 would be required for 1 token to vest per second)
7. **Address vestingAdmin:** The special vesting administrator address, typically a multi-signature wallet or a DAO contract address. This address is responsible for revoking NFTs for employees or beneficiaries that are no longer supposed to vest tokens. Tokens that are unvested are returned to this address.

```
function createLockedNFT(
    address recipient,
    address token,
    uint256 amount,
    uint256 start,
    uint256 cliffDate,
    uint256 rate,
    address vestingAdmin,
    uint256 unlockDate,
    bool transferableNFTLockers
  ) external;
```

This function is a core function that creates a new vesting NFT. This function will transfer tokens from the msg.sender to the NFT smart contract. It is different from the above in that it takes two additional inputs for a unlockDate and transferableNFTLockers. This should only be used if there is a mandatory lockup period for vested tokens, otherwise the createNFT function is simpler and should be used.

1. **Address recipient:** The address of the beneficiary, the recipient of the vesting NFT.
2. **Address token:** The address of the token that will be vesting
3. **Uint256 amount:** The total amount of tokens to be vested across the entire schedule
4. **Uint256 start:** The start date when tokens begin vesting. This can be back dated, or future dated, or use the current time stamp. This is unix time representation of seconds.
5. **Uint256 cliffDate:** Cliff date when tokens vest in a single clump, on a cliff. If this date is set at the same as the start or before it, then there is no cliff date - this only applies if the amount is set greater than the start date, also represented as Unix seconds timestamp.
6. **Uint256 rate:** The rate at which tokens vest per second. If the rate is set to 5, then 5 tokens vest per second, meaning over the course of 1 minute 300 tokens have vested. This is base version of token vesting, meaning it does not account for the decimals of the token (so if 1 token = 1ee18, then a rate of 1ee18 would be required for 1 token to vest per second)
7. **Address vestingAdmin:** The special vesting administrator address, typically a multi-signature wallet or a DAO contract address. This address is responsible for revoking NFTs for employees or beneficiaries that are no longer supposed to vest tokens. Tokens that are unvested are returned to this address.
8. **Uint256 unlockDate:** This is a unix timestamp parameter that defines the end of a lockup period - where vested tokens are subject to a lockup period such that even after those tokens are vested, they still cannot be claimed until after the unlockDate has passed, which describes the end of the lockup period.
9. **Bool transferableNFTLockers:** This boolean determines when a vesting admin revokes an NFT, but that NFT contains vested tokens still subject to the lockup period, whether the locked tokens the beneficiary receives can be transferred or not. The beneficiary receives another NFT representing their locked tokens - which can either be transferable or not transferable based on the creators decision.

```
function redeemNFT(uint256[] memory tokenId) external;
```

This function will redeem NFTs and claim any vested tokens. It is used by the beneficiary / holder of the NFT. The holder can input as many tokenIds as they like, subject to them owning the NFTs and there being a vested balance that can be claimed. This function then will iterate through the array of tokenIds input, and redeem them that have vested balances. The function leverages an internal function that checks the msg.sender is the owner of the NFT, and that there is a balance to be claimed. Then it withdraws the vested balance of tokens from the NFT contract and delivers to the msg.sender. The internal function will also update the storage struct to reset the amount to the remaining tokens that are still vesting, and it will reset the start date to the current block timestamp. This is critical to make sure it continues vesting in the same linear function it was initially constructed to vest along. If the NFT is entirely redeemed - ie the vested balance equals the total amount in the vesting struct, then the NFT will be burned and streams struct deleted.

```
function redeemAllNFTs() external;
```

This function will redeem all of the NFTs a wallet holds. This function iterates through the enumerable balances of the msg.sender, and then redeems all of the NFTs that have a balance. This is a useful function for holders that do not need to keep their tokenIds handy, but rather can simply redeem and claim their vested tokens in bulk. This function utilizes the same internal redeem NFT function, performing the same checks and withdrawing the vested balances, updating the storage data and burning any NFTs that have been fully redeemed.

```
function revokeNFT(uint256[] memory tokenIds) external
```

This function is used by the vestingAdmin to revoke tokens. TokenIds can be added in an array to bulk revoke for a user across all of their vesting schedules. This function calls an internal _revokeNFT method that performs necessary checks to ensure that the revoker is the vestingAdmin, to confirm that there is a remainder in the address still to be revoked (unvested balance). The internal function also burns the NFT and deletes the storage struct Stream that contains the vesting schedule data. The function will withdraw tokens from the NFT contract and deliver unvested balances to the vestingAdmin. Any vested balances that have not been claimed will either be transferred to the beneficiary (holder) of the NFT, or if they are subject to the lockup period as denoted by an unlockDate in the future, then instead of receiving the vested tokens, the beneficiary will receive a new time locked token NFT, which locks the vested tokens inside an NFT where the unlock date of the tokens is the unlockDate that the lockup period manded.

```
function delegateToken(address delegate, uint[] memory tokenIds) external
```

This function allows the owner of an NFT to delegate the NFT(s) to another wallet address. This is how NFTs are delegated for snapshot voting. It records the delegate address mapped to the tokenId.

```
function delegateAllNFTs(address delegate) external
```

This function performs the same function as the delegateToken function, but will delegate all of the tokens owned by a wallet, rather than a specific NFT or set of NFTs.