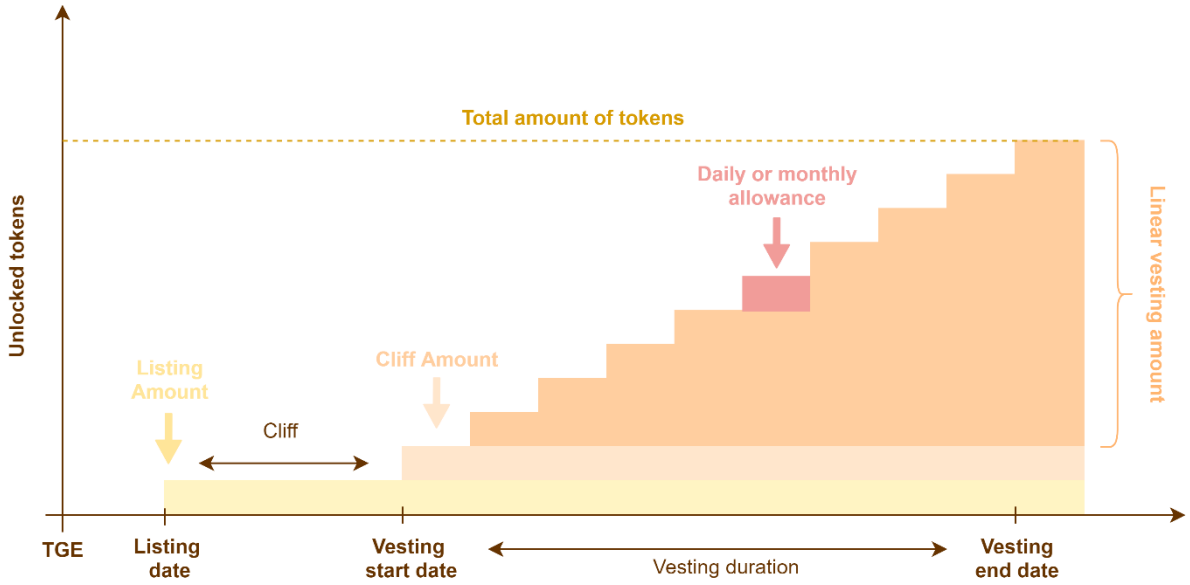
Sahara Vesting Pools

High level description

Business description of the project, one-two sentences.

Vesting contract for the specified platform tokens where owners can add vesting pools and wallets whereas beneficiaries can claim vested tokens during the set vesting period.



Main actors, actions, flows

List of roles and their main actions:

- **Contract owner** can add vesting pools with specified listing, cliff percentage and vesting duration.
- **Contract owner** can add beneficiary wallets that are eligible for claiming tokens.
- **Contract owner** can change listing date.
- **Contract owner** can remove beneficiary.
- **Beneficiary** can claim vested tokens if there are any.
- **All users** can check vesting pool data.

Technical details

Design and architecture choices, clever tricks and hacks, complex logic.:

- The contract is upgradeable (using @OpenZeppelin transparent proxy).
- Contract uses IERC20 interface for transferring claimable tokens.
- Contract uses number of modifiers to restrict functionality based on roles. Modifiers are also create to minimize the risk of writing unwanted, ineligible values.

Deployed contracts

Number of deployed contracts, connections between them. E.g.:

- We deploy a separate `VestingMain` and a `TransparentUpgradeableProxy` contract (latter is OpenZeppelin standard and is uploaded using truffle-upgrade plugin (code is not stored in the repository).

Main logic

There are two structures (objects) that users will interact with: pools and beneficiaries.

Main parameters for pools are:

- Name;
- Cliff period in days;
- Vesting period in months;
- Listing percentage (fractional form. E.g. 7.5 % = 3/40);
- Cliff percentage (fractional form. E.g. 5 % = 1/20);
- Total allocated pool amount;
- Linear vesting unlock type: daily or monthly.

Main parameters for beneficiaries are:

- Wallet address;
- Pool ID;
- Token amount (absolute format: e.g. 10 SHRA tokens = 10 * 10^{18} , or 10000000000000000000)

Token amount for beneficiary is recalculated this way:
Total amount = Listing amount + Cliff amount + Vesting amount

Unlocked token amount is calculated this way:

- **If listing has started :** listing token amount;
- **If cliff has ended :** listing token amount + cliff token amount + vested unlocked tokens:

$$unlockedTokens = listingAmount + cliffAmount + \frac{vestingAmount * periodsPassed}{duration}$$

- **If vesting period ended :** transfer all allocated and unclaimed tokens.

Restrictions

There are number of modifiers that restrict and moderates usage of functions.

- **onlyOwner** – permits pool and beneficiary additions, modifications only for contract owners;
- **onlyWhitelisted** – lets only beneficiaries to invoke claimable function;
- **addressNotZero –** checks that there is no way to add beneficiary with address 0;
- **poolExists –** checks that pool that is being updated really exists;
- **validListingDate –** Checks whether the listing date is not in the past;
- **nameDoesNotExist -** Checks whether new pool's name does not already exist;
- **nonReentrant -** Reentrancy check. Disables multiple calls from one address at once;
- **tokenNotZero –** Token amount is higher than 0;

Vesting contract functions

In this section you will find ta description of main and more complex functions which are available for contract owner and beneficiaries in vesting contract: what are function parameters, their types, usage explanation and code examples.

Formats

- **Timestamps** are passed in Unix epoch time, see https://www.epochconverter.com/
- **Token amounts** are passed in absolute form, see https://eth-converter.com/.

Owner functions

addVestingPool

Adds a new vesting pool.

Parameter	Type	Explanation
name	string	Pool name
listingDate	timestamp / uint256	The start of token distribution. Format in seconds since epoch start (timestamp).
listingPercentageDividend	uint256	Tokens unlocked on listing date. If listing percentage is 5%, then **dividend** is: 1 and **divisor** is: 20 (1/20 = 0,05)
listingPercentageDivisor	uint256	
cliff	uint256	Cliff period in **days.**
cliffPercentageDividend	uint256	Tokens unlocked after cliff period has ended. If cliff percentage is 7,5%, then **dividend** is: 3 and **divisor** is: 40 (3/40 = 0,075)
cliffPercentageDivisor	uint256	
vestingDurationInMonths	uint256	Duration of vesting period when tokens **are linearly unlocked**.

unlockType	UnlockTypes (enum)	0 for DAILY; 1 for MONTHLY;
poolTokenAmount	uint256	Total amount of tokens available for claiming from pool. **absolute token amount!** If pool has 5 000 000 tokens, contract will accept "5000000000000000000000000" ← value as a parameter which is (5 000 000 * 10^{18})

User desired values to be set	Conversion to contract parameters
• **Name**: Test 1 • **Listing**: 5% • **Cliff**: 1 day • **Cliff Percentage**: 0% • **Vesting Period**: 3 months • **Unlock Type**: Daily • **Total pool amount**: 1 000 000	• **Name**: 'Test 1' • **Listing Dividend**: 1 • **Listing Divisor**: 20 • **Cliff**: 1 • **Cliff Dividend**: 0 • **Cliff Divisor**: 1 • **Vesting Period**: 3 • **Unlock Type:** 1 • **Total pool amount:** 100000000000000000000000

addToBeneficiariesList

Adds beneficiary's wallet address with purchased token amount to vesting pool.

Parameter	Type	Explanation
poolIndex	uint256	Pool index
address	address	Beneficiary wallet address
tokenAmount	uint256	Total amount of tokens that **address** can claim from vesting pool. **absolute token amount!**

Solidity example

```
addToBeneficiariesList(
  0,
 "0x21E6B6ad11C058EEDe8C0041615bd040b590CB4c",
 "10000000000000000000");
```

Values	Contract parameters
• **Pool**: Test 1 • **Beneficiary**: 0x21E6B6ad11C058EEDe8C00416 15bd040b590CB4c • **Amount** : 1000 tokens	• **poolIndex**: 0 • **address** : '0x21E6B6ad11C058EEDe8C00416 15bd040b590CB4c'; • **amount:** 100000000000000000000

addToBeneficiariesListMultiple

Adds beneficiaries wallet addresses with purchased token amount to the beneficiary list.

Parameter	Type	Explanation
poolIndex	uint256	Pool index
addresses	address[]	Beneficiary wallet addresses
tokenAmount	uint256[]	Amounts of tokens that can be claimed by beneficiaries. **absolute token amount!**

Passable parameters example

```
addToBeneficiariesListMultiple(
  0,
["0x21E6B6ad11C058EEDe8C0041615bd040b590CB4c",
0xadE6B6ad11C058ff3e8C0041615bd040b590C3aC],
["100000000000000000000", "150000000000000000000"]);
```

Values	Contract parameters

• **Pool**: Test 1
• **Beneficiary List**:
  ○ 0x21E6B6ad11C058EEDe8C0 041615bd040b590CB4c : 1000 Tokens
  ○ 0xadE6B6ad11C058ff3e8C00 41615bd040b590C3aC : 1500 Tokens
	• **poolIndex**: 0
	• **addresses** : [“0x21E6B6ad11C058EEDe8C004 1615bd040b590CB4c”, “0xadE6B6ad11C058ff3e8C00416 15bd040b590C3aC”]
	• **tokenAmount:** [“100000000000000000000”, “150000000000000000000”]

Beneficiary functions

claimTokens

Lets user claim unlocked tokens.

Requirements:

- Wallet is whitelisted;
- Contract must have some balance;
- Unlocked tokens > 0