Deedz Coin Smart Contract

Table of Content

1. Project Overview

The LockableToken contract manages token locking functionality and extends the ERC20 and Ownable and SupplierRole contracts. It provides methods to lock tokens, check locked token balances, extend lock duration, increase locked token amount, check unlockable token balances, unlock tokens, and retrieve the total unlockable tokens for an address.

2. Functional Requirements

2.1 Roles:

- **Owner:** The contract owner has administrative privileges, including the ability to transfer the supplier role.
- **Supplier:** The supplier role is a specialized role with privileges related to token locking and distribution.

2.2 Features:

- **Token Transfers**
  Users can transfer DeedzCoin tokens to other addresses.
- **Token Locking by Supplier**
  The supplier role, with special privileges, can lock a specified amount of tokens for a specified duration, preventing them from being transferred until the lock expires. This functionality is enhanced with more control and flexibility.
- **Locking with Actual Time by Supplier**
  The supplier can also lock tokens until a specific timestamp, ensuring they remain locked until that time.
- **Supplier Role**
  The contract introduces a "supplier" role with special privileges, including the ability to transfer tokens while locking them and managing the distribution of tokens.
- **Locking Enhancements**
  The supplier can extend the lock duration for a specific reason and increase the number of tokens locked for a reason.
- **Unlockable Tokens and Unlock Function**
  The contract introduces the concept of "unlockable" tokens that can be claimed by users after passing their lock validity period. The unlock function allows users to claim these unlockable tokens.
- **Get Unlockable Tokens Function**
  Users can query the contract to get the total number of unlockable tokens they have across all lock reasons.

2.3 Use Cases:

- **Team Token Vesting:** DeedzCoin wants to incentivize its core development team by providing them with tokens as part of their compensation package. However, the project is concerned about team members potentially selling their tokens immediately, which could negatively impact the token's price and stability. The DeedzCoin smart contract can be used to lock team members' tokens for a predefined vesting period. Tokens would be gradually released to team members over time, ensuring their commitment to the project's long-term success.
- **Partnership Lock-Up:** DeedzCoin is entering into a strategic partnership with another blockchain project. As part of the partnership agreement, DeedzCoin needs to provide a certain amount of tokens to the partner. To ensure that the partner remains committed to the collaboration and doesn't immediately sell the tokens, DeedzCoin can use the smart contract to lock the partnership tokens for a specified duration. This way, the tokens are released to the partner gradually, aligning with the partnership's goals.
- **Community Token Rewards:** DeedzCoin aims to foster an active and engaged community of token holders. To incentivize long-term holding and active participation within the DeedzCoin ecosystem, the project launches a community rewards program. Users who lock their tokens in the DeedzCoin smart contract for a specific period are eligible to receive bonus tokens or participate in exclusive events and governance decisions. This use case encourages token holders to become loyal and engaged members of the DeedzCoin community.

3 Technical Requirements

The "DeedzCoin" project has been developed using the Solidity programming language within the Hardhat development environment. TypeScript is employed for testing and scripting purposes. OpenZeppelin's libraries are also integrated into the project. To obtain more information about these libraries and installation instructions, refer to their GitHub repository.

- **Project Structure**

The project's folder structure is as follows:

```
|— contracts
|    └— DeedzCoin.sol
|— scripts
|    └— deploy.js
|— utilities
|    └— parse-custom-error.js
|    └— promise-handler.js
|    └— time-constants.js
|— test
|    └— deedzCoin.test.js
|— README.md
|— docs
|    └— Details.pdf
```

For detailed project information, including structure and scripts, consult the "**README.md**" file in the project root directory. The "**./contracts**" folder contains contract files, while the "**./docs**" directory holds documentation resources. Contract deployment is managed via the "**deploy.js**" script located in the "**./scripts**" directory. To ensure that specific files or directories, such as build artifacts or dependencies.

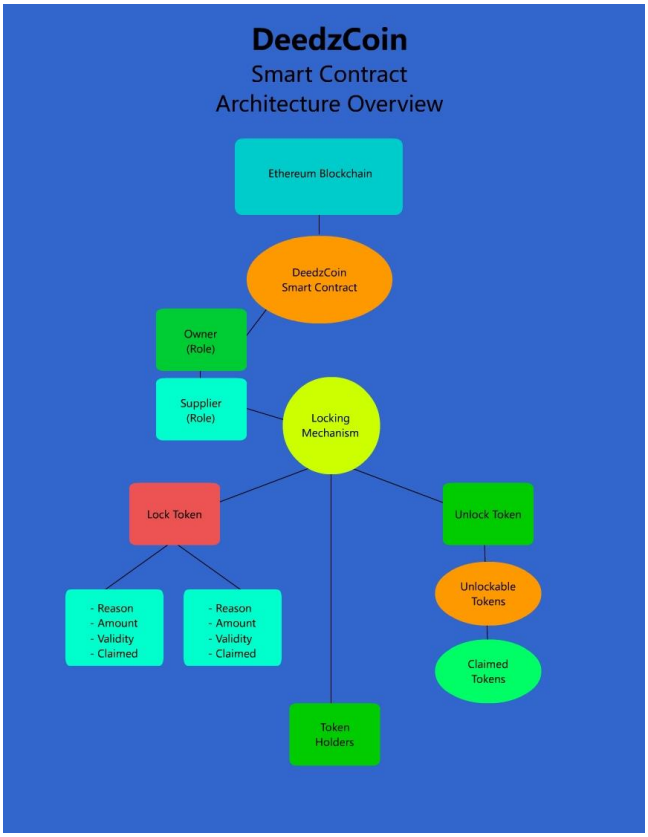
The entire project, along with its source code and documentation, is hosted on GitHub at the following repository: **https://github.com/experiencenano/DeedzCoin.git**

These technical requirements provide an overview of the development stack, project structure, documentation, and version control for the "**DeedzCoin**" project, following the style of the provided document.

3.1 Architecture Overview:

- The "Ethereum Blockchain" is the foundational layer.
- The "DeedzCoin Smart Contract" is represented as the central component.
- "SupplierRole" branches into "Owner" and "Supplier" roles.
- The "Locking Mechanism" includes multiple "Lock Tokens" representing distinct reasons for locking tokens.
- Each "Lock Token" consists of an "Amount" of tokens, "Validity" period, and "Claimed" status.

This graphical representation offers a clear visualization of the DeedzCoin smart contract's architecture, illustrating the relationships between its key components



4 Contract Information

4.1 Contract Inheritance:

- **ERC20**: Implements the ERC-20 token standard.
- **Ownable**: A contract for managing ownership functionality.
- **SupplierRole**: A contract for managing supplier functionality.

4.2 Constructor:

Parameters: supplierAddress (address), totalSupply (uint256)

- Initializes the token supply by minting the specified totalSupply number of tokens to the supplierAddress.

4.3 Assets:

- **Supplier Role:** A specialized role defined within the contract to manage token-related operations.
- **LockToken Struct:** Represents a locked token, including the amount, validity period, and whether it has been claimed.
- **lockReason Mapping:** Records reasons for token locking.
- **locked Mapping:** Stores data of all the tokens locked for specific addresses and reasons.
- **allowed Mapping:** Manages token allowances.

4.3 Modifiers:

- **onlyOwne**r: Restricts certain functions to be callable only by the contract owner.
- **onlySupplie**r: Restricts certain functions to be callable only by the address assigned as the supplier.

4.4 Functions:

- **lock**
  Parameters: _reason (bytes32), _amount (uint256), _time (uint256)
    - Locks a specified amount of tokens against the caller's address for a specific reason and duration.
    - Checks if tokens are already locked for the given reason and requires a non-zero amount.
    - Transfers the specified amount of tokens from the caller to the contract and updates the locked token information.
- **tokensLocked**
  Parameters: _account (address), _reason (bytes32)
    - Returns the number of tokens locked for a specified address and reason.

- **tokensLockedAtTime**
  Parameters: _account (address), _reason (bytes32), _time (uint256)
    - Returns the number of tokens locked for a specified address and reason at a specific time.
- **totalBalanceOf**
  Parameters: _account (address)
    - Returns the total balance of tokens held by an address, including both transferable and locked tokens.
- **extendLock**
  Parameters: _reason (bytes32), _time (uint256)
    - Extends the lock duration for a specified reason and time.
- **increaseLockAmount**
  Parameters: _reason (bytes32), _amount (uint256)
    - Increases the number of tokens locked for a specified reason.
- **tokensUnlockable**
  Parameters: _account (address), _reason (bytes32)
    - Returns the number of tokens unlockable for a specified address and reason.
- **Unlock**
  Parameters: _reason (bytes32)
    - Unlocks the unlockable tokens of the caller's address, transferring them back to the caller.
- **getUnlockableTokens**
  Parameters: _account (address)
    - Calculates the total number of unlockable tokens for a specified address.
- **transferSupplierRole**
    - Parameters: newSupplier (address)
    - Transfers the supplier role to a new address. This function can only be called by the contract owner (onlyOwner modifier).

4.5 Events:

- **TokensLocked**: Emitted when tokens are locked against an address for a specific reason and duration.
- **TokensUnlocked**: Emitted when tokens are unlocked and transferred back to the caller's address.
- **SupplierRoleTransferred**: Emitted when the supplier role is transferred from one address to another.

4.6 Ownership:

- The contract owner is established during contract deployment and maintains this status throughout the contract's existence.
- Ownership grants specific administrative privileges, controlled by the **onlyOwner** modifier, restricting certain functions to the contract owner.

4.6.1 Supplier Role Assignment:

  - The contract introduces a "**supplier**" role, which carries distinct privileges.
  - During contract deployment, the supplier role is assigned to a designated address, responsible for token locking and related operations.
  - The contract owner has the authority to transfer the supplier role to another address using the **transferSupplierRole** function, providing flexibility in supplier role management.
  - This setup ensures secure contract management and controlled access to essential functions, while the supplier role oversees token-related operations.