Smart Contract Documentation: Pandemic coin

Overview

The Pandemic smart contract is an ERC-20 compliant token contract with additional functionalities inherited from context, ownable, and reentrancy guard modules. It manages token minting, burning, transfers, allowances, and ownership.

Contract Structure and Functions

State Variables

- `_balances` (mapping): This mapping stores the token balances of each address. For instance:
    - `_balances[0xAddress1] = 1000`: Indicates that the address '0xAddress1' holds 1000 Pandemic tokens.
- `_allowances` (mapping): Manages the allowances for token transfers between addresses. For example:
    - `_allowances[0xOwner][0xSpender] = 500`: Represents that '0xOwner' approved '0xSpender' to spend 500 tokens on their behalf.
- `_isAutomatedMarketMaker` (mapping): Identifies whether an address is an automated market maker (AMM). This is a boolean mapping (true/false) to define AMM status for addresses.
- `MaxSupply` (uint256): Represents the maximum supply of tokens allowed in the contract. For instance:
    - `MaxSupply = 1000000000 ether`: Sets the maximum token supply to 1 billion Pandemic tokens.
- `_totalSupply` (uint256): Tracks the total supply of the Pandemic coin. Initially set to 0 and increases when tokens are minted.
- `_teamWallet` (address): Represents the wallet address designated for the team responsible for managing the contract and holding tokens.
- `_name`, `_symbol`, `_decimals` (string, string, uint8): Variables containing token name, symbol, and decimals respectively. For example:
    - `_name = "Pandemic coin"`, `_symbol = "PDC"`, `_decimals = 18`.

Functions and Usage

Constructor

- `constructor()`: Initializes the token name, symbol, and decimals, setting them to "Pandemic coin" (PDC), "PDC", and 18 respectively. It also initializes the `_teamWallet` with the team's wallet address.

Functions Explanation

`name()`, `symbol()`, `decimals()`

These functions are view functions used to access information about the token without modifying the contract's state.

- `name()`: Returns the name of the token, in this case, "Pandemic coin".
- `symbol()`: Returns the symbol of the token, which is "PDC".
- `decimals()`: Returns the number of decimal places used for token balances, set to 18 in this contract.

`totalSupply()`

This function returns the total number of tokens minted and currently in circulation.

- Example:
  - `totalSupply()`: Returns the total supply of Pandemic tokens.

`balanceOf(address account)`

Used to retrieve the token balance of a specific address.

- Example:
  - `balanceOf(0xAddress)`: Returns the balance of Pandemic tokens held by address '0xAddress'.

`mint(address to, uint256 amount)`

Enables the contract owner to mint new tokens and assign them to a specified address.

- Parameters:
  - `to`: The address to which tokens will be minted.
  - `amount`: The number of tokens to mint.

- Example:
- `mint(0xRecipient, 1000);`
- This mints 1000 Pandemic tokens and assigns them to address '0xRecipient'.

`setTeamWallet(address newTeamWallet)`

Allows the contract owner to update the team's wallet address.

- Parameters:
    - `newTeamWallet`: The new address for the team's wallet.
- Example:
- `setTeamWallet(0xNewTeamWalletAddress);`
- This changes the team's wallet address to '0xNewTeamWalletAddress'.

`transfer(address to, uint256 amount)`

Facilitates the transfer of tokens between addresses.

- Parameters:
    - `to`: The recipient address.
    - `amount`: The number of tokens to transfer.
- Example:
- `transfer(0xRecipient, 500);`
- Transfers 500 Pandemic tokens to the address '0xRecipient'.

`approve(address spender, uint256 amount)`

Allows the caller to approve a designated spender to spend a specific amount of tokens on their behalf.

- Parameters:
    - `spender`: The address authorized to spend tokens.
    - `amount`: The maximum number of tokens that can be spent.
- Example:
- `approve(0xSpender, 200);`
- Approves the address '0xSpender' to spend 200 Pandemic tokens on behalf of the caller.

`transferFrom(address sender, address recipient, uint256 amount)`

Transfers tokens from one address to another based on the allowance given by the sender.

- Parameters:

        - `sender`: The address from which tokens are being transferred.
        - `recipient`: The address receiving the tokens.
        - `amount`: The number of tokens to transfer.
- Example:
- `transferFrom(0xSender, 0xRecipient, 300);`
- Transfers 300 Pandemic tokens from '0xSender' to '0xRecipient' if the allowance is granted.
-

Internal Helper Functions

- `_approve(address owner, address spender, uint256 amount)`: Sets a spender's allowance for a specific owner.
- `_transfer(address from, address to, uint256 amount)`: Handles the transfer of tokens between addresses.
- `_mint(address account, uint256 amount)`: Mints new tokens and assigns them to a specified account.
- `_burn(address account, uint256 amount)`: Burns a specified amount of tokens from a particular account.
- `_spendAllowance(address owner, address spender, uint256 amount)`: Checks and spends an allowance between addresses.

Events

- `Transfer(address indexed from, address indexed to, uint256 value)`: Emitted on token transfers.
- `Approval(address indexed owner, address indexed spender, uint256 value)`: Emitted on approval of token allowance.

Security Considerations

- Reentrancy protection applied using `ReentrancyGuard`.
- Ownership controls through `Ownable` for restricted functions.
- Safeguards against insufficient balances and allowances in transfer functions.

Testing Guide

1. Preparation

a. First replace the .env values with actual values.

```
WALLET_PRIVATE_KEY : The private key of your Ethereum wallet.
MNEMONIC :The mnemonic phrase used to derive the Ethereum addresses.
ETHERSCAN_API_KEY: The API key used to interact with polygon for
contract verification. login to polygon and generate api key
```

b. (Optional) update Keys in scripts/hardhat.config.js

```
 const MUMBAI_API_KEY = "tMXNMqXM6uir3cg3stybGsmPM6h0vtIY";
const MUMBAI_PRIVATE_KEY = "private key";

module.exports = {
  solidity: "0.8.19",

  networks: {
    mumbai:{
      url:
`https://polygon-mumbai.g.alchemy.com/v2/${MUMBAI_PRIVATE_KEY}`,
      accounts: [`${MUMBAI_PRIVATE_KEY}`],
    }
  }
};
```

2. Deployment

a. Make sure you have some matic faucet in your account.
b. Open terminal and run the following command

```
npx hardhat --network mumbai run scripts/deploy_stake.ts
```

Command to compile and deploy

Compile : npx hardhat compile
Test : npx hardhat test test/TokenLocking.test
deploy(testnet) : npx hardhat --network mumbai run scripts/deploy_stake.ts
deploy(mainnet) : npx hardhat --network mainnet run scripts/deploy_stake.ts