// SPDX-License-Identifier: MIT

pragma solidity ^0.8.26;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

import "@openzeppelin/contracts/security/ReentrancyGuard.sol";

/// @title DogeJindo Token Contract

contract DogeJindoToken is ERC20, Ownable, ReentrancyGuard {

uint256 private constant TOTAL\_SUPPLY = 10\_000\_000\_000 \* 10 \*\* 18;

uint256 public constant COMMUNITY\_SUPPLY = (TOTAL\_SUPPLY \* 30) / 100;

uint256 public constant PRESALE\_SUPPLY = (TOTAL\_SUPPLY \* 30) / 100;

uint256 public constant TEAM\_SUPPLY = (TOTAL\_SUPPLY \* 10) / 100;

uint256 public constant RESERVE\_SUPPLY = (TOTAL\_SUPPLY \* 20) / 100;

uint256 public constant LIQUIDITY\_SUPPLY = (TOTAL\_SUPPLY \* 10) / 100;

/// @dev Constructor that initializes the contract and mints the total supply to the contract owner

constructor() ERC20("DogeJindo", "DOJ") Ownable(msg.sender) {

\_mint(msg.sender, TOTAL\_SUPPLY);

}

/// @dev Hook to prevent token transfers to the zero address

/// This function includes from and amount parameters to avoid future compatibility issues.

function \_beforeTokenTransfer(

address from,

address to,

uint256 amount

) internal pure {

require(to != address(0), "Transfer to the zero address is not allowed");

require(from != address(0), "Transfer from the zero address is not allowed");

require(amount > 0, "Transfer amount must be greater than zero");

}

/// @dev Disable receiving Ether directly to the contract

receive() external payable {

revert("Direct Ether transfers not allowed");

}

/// @dev Disable fallback function calls to the contract

fallback() external payable {

revert("Fallback not allowed");

}

/// @dev Prevent renouncing ownership to maintain contract control

function renounceOwnership() public view override onlyOwner {

revert("Renouncing ownership is disabled");

}

/// @dev Allows the owner to transfer ownership to a DAO contract

function transferToDAO(address daoAddress) external onlyOwner nonReentrant {

require(daoAddress != address(0), "Invalid DAO address");

require(\_isContract(daoAddress), "DAO address must be a contract");

transferOwnership(daoAddress);

}

/// @dev Utility function to check if an address is a contract

function \_isContract(address addr) internal view returns (bool) {

uint32 size;

assembly {

size := extcodesize(addr)

}

return size > 0;

}

}