**6. Contract Factory Development:** Develop a contract factory that can create the ERC-20 tokens. Ensure that only the Owner can generate new tokens and include functionality to assign a batch ID and specify the number of tokens during creation.

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;

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

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

// Custom ERC-20 token contract with expiry date

contract ExpirableERC20Token is ERC20, Ownable {

mapping(uint256 => uint256) private tokenExpiryDates;

event TokenMintedWithExpiry(address indexed to, uint256 value, uint256 expiryDate);

constructor(

string memory name,

string memory symbol,

address initialHolder,

uint256 initialSupply

) ERC20(name, symbol) Ownable(initialHolder) {

\_mint(initialHolder, initialSupply);

}

function mintWithExpiry(address to, uint256 value, uint256 expiryDate) external onlyOwner {

require(expiryDate > block.timestamp, "Expiry date must be in the future");

\_mint(to, value);

tokenExpiryDates[\_tokenId(to, value)] = expiryDate;

emit TokenMintedWithExpiry(to, value, expiryDate);

}

function transfer(address to, uint256 value) public override returns (bool) {

require(isTokenValid(to, value), "Token has expired");

return super.transfer(to, value);

}

function transferFrom(address from, address to, uint256 value) public override returns (bool) {

require(isTokenValid(from, value), "Token has expired");

return super.transferFrom(from, to, value);

}

function isTokenValid(address owner, uint256 value) public view returns (bool) {

uint256 tokenId = \_tokenId(owner, value);

return (tokenExpiryDates[tokenId] == 0 || tokenExpiryDates[tokenId] > block.timestamp);

}

function \_tokenId(address owner, uint256 value) internal pure returns (uint256) {

return uint256(keccak256(abi.encodePacked(owner, value)));

}

}

// Contract factory for creating ExpirableERC20Token instances

contract TokenFactory is Ownable {

event TokenCreated(address indexed tokenAddress, string name, string symbol, uint256 batchId, uint256 quantity);

constructor() Ownable(msg.sender) {} // Pass the initial owner to Ownable

function createToken(

string memory name,

string memory symbol,

uint256 initialSupply,

uint256 expiryDate,

uint256 batchId,

uint256 quantity

) external onlyOwner {

ExpirableERC20Token newToken = new ExpirableERC20Token(name, symbol, msg.sender, initialSupply);

newToken.mintWithExpiry(msg.sender, quantity, expiryDate);

emit TokenCreated(address(newToken), name, symbol, batchId, quantity);

}

}