10. **Code Review:** Submit the smart contracts for a code review. Highlight how your implementation ensures security, efficiency and adheres to Ethereum best practices.

Ans:

// 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 {

uint256 public expiryDate;

// Constructor to initialize the token with a given name, symbol, initial supply, and expiry date

constructor(

string memory name,

string memory symbol,

address initialHolder,

uint256 initialSupply,

uint256 \_expiryDate

) ERC20(name, symbol) {

\_mint(initialHolder, initialSupply);

expiryDate = \_expiryDate;

}

// Function to transfer tokens, checking for expiry before allowing the transfer

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

require(block.timestamp < expiryDate, "Token has expired");

return super.transfer(to, value);

}

// Function to transfer tokens on behalf of someone else, checking for expiry

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

require(block.timestamp < expiryDate, "Token has expired");

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

}

// Function to update the expiry date (only callable by the owner)

function updateExpiryDate(uint256 newExpiryDate) external onlyOwner {

expiryDate = newExpiryDate;

}

}

**Security Considerations:**

The use of the OpenZeppelin library for the ERC20 token (@openzeppelin/contracts/token/ERC20/ERC20.sol) and the Ownable contract (@openzeppelin/contracts/access/Ownable.sol) adds a layer of security, as these libraries are well-audited and widely used in the Ethereum community.

The constructor initializes the token's expiry date, providing transparency and clarity about the token's lifecycle.

The transfer and transferFrom functions include a check for the token's expiry date, preventing transfers after the specified expiry time, enhancing the security of token holders.

**Efficiency:**

The contract utilizes the OpenZeppelin ERC20 implementation, which is known for its efficiency and gas optimization.

The use of block.timestamp for checking the expiry date is a gas-efficient approach to validate time-based conditions.

**Adherence to Best Practices:**

The SPDX-License-Identifier is correctly specified at the beginning of the contract.

The code structure is clean and well-organized, making it readable and maintainable.

The constructor initializes the contract state variables, including the expiry date.

The onlyOwner modifier is used to restrict the updateExpiryDate function to only be callable by the owner, ensuring proper access control.

**Gas Optimization:**

The contract uses standard ERC20 functions, and the additional check for expiry in transfer functions is a reasonable security measure without significant impact on gas costs.