I have deployed the contract on goerli: https://goerli.etherscan.io/address/0x7e9f99d8baebad8aff47467241db277228836dae

Could someone help me review the contract?

//SPDX-License-Identifier: MIT pragma solidity 0.8.17;

import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol"; import "@openzeppelin/contracts/utils/Strings.sol"; import "@openzeppelin/contracts/utils/Address.sol";

contract BankLockTest { address owner; uint128 counter; mapping(address => bool) private legalErc20Tokens; mapping(string => Receipt) private receiptRepo; mapping(string => bool) private hasReceipt; struct Receipt { address customer; address token; uint256 amount; uint256 unlockTime; bool isEther; }

```
using SafeERC20 for IERC20;

event Read(
    address customer,
    address token,
    uint256 amount,
    uint256 unlockTime
);
event DepositErc20Token(
    string receiptKey,
    address customer,
    address token,
    uint256 amount,
    uint256 lockDays,
    uint256 unlockTime
);
event WithdrawErc20Token(
    string receiptKey,
    address customer,
    address token,
    uint256 amount,
    uint256 time
);
event DepositEther(
    string receiptKey,
    address customer,
    uint256 amount,
    uint256 lockDays,
    uint256 unlockTime
);
event WithdrawEther(
    string receiptKey,
    address customer,
    uint256 amount,
    uint256 time
);
event AddToken(address token);

constructor() {
    owner = msg.sender;
    counter = 0;
}

function _computeReceiptKey(Receipt memory _receipt, uint256 _counter)
    private
    view
    returns (string memory)
{
    return
        Strings.toString(
            uint256(
                keccak256(
                    abi.encode(
                        _receipt.customer,
                        _counter + block.timestamp
                    )
                )
            )
        );
}

modifier _isLegalErc20Token(address _token) {
    require(legalErc20Tokens[_token], "not legal token");
    _;
}

modifier _notContractAddress(address _address) {
```

```
    require(!Address.isContract(_address), "not support contract address");
    _;
}

function _getUnlockTime(uint256 _lockDays) private view returns (uint256) {
    return _lockDays * 86400 + block.timestamp;
}

function addToken(address _token) public {
    require(msg.sender == owner, "only owner can add tokens");
    legalErc20Tokens[_token] = true;
    emit AddToken(_token);
}

function getReceipt(string memory _receiptKey) public {
    require(hasReceipt[_receiptKey], "has not receipt or already draw");
    Receipt memory receipt = receiptRepo[_receiptKey];
    emit Read(
        receipt.customer,
        receipt.token,
        receipt.amount,
        receipt.unlockTime
    );
}

function depositEther(uint256 lockDays)
    public
    payable
    _notContractAddress(msg.sender)
{
    require(msg.value > 0, "amount <= 0");
    if ((lockDays <= 0) || (lockDays > 180)) {
        lockDays = 1;
    }
    unchecked {
        counter = counter + 1;
    }
    uint256 unlockTime = _getUnlockTime(lockDays);
    address etherAddress = address(0);
    Receipt memory receipt = Receipt(
        msg.sender,
        etherAddress,
        msg.value,
        unlockTime,
        true
    );
    string memory receiptKey = _computeReceiptKey(receipt, counter);
    require(!hasReceipt[receiptKey], "same receipt key collision");
    receiptRepo[receiptKey] = receipt;
    hasReceipt[receiptKey] = true;

    emit DepositEther(
        receiptKey,
        msg.sender,
        msg.value,
        lockDays,
        unlockTime
    );
}

function depositErc20Token(
    address token,
    uint256 amount,
    uint256 lockDays
) public _isLegalErc20Token(token) _notContractAddress(msg.sender) {
    require(amount > 0, "amount <= 0");
    require(
        !Address.isContract(msg.sender),
        "not support contract address"
    );
    if ((lockDays <= 0) || (lockDays > 180)) {
        lockDays = 1;
    }
    unchecked {
        counter = counter + 1;
    }

    uint256 unlockTime = _getUnlockTime(lockDays);
    Receipt memory receipt = Receipt(
        msg.sender,
        token,
```

```
        amount,
        unlockTime,
        false
    );
    string memory receiptKey = _computeReceiptKey(receipt, counter);
    require(!hasReceipt[receiptKey], "same receipt key collision");
    receiptRepo[receiptKey] = receipt;
    hasReceipt[receiptKey] = true;
    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

    emit DepositErc20Token(
        receiptKey,
        msg.sender,
        token,
        amount,
        lockDays,
        unlockTime
    );
}

function withdraw(string memory receiptKey) public {
    require(hasReceipt[receiptKey], "has not receipt or already draw");
    require(receiptRepo[receiptKey].unlockTime < block.timestamp, "unlock time not reached");
    hasReceipt[receiptKey] = false;
    Receipt memory receipt = receiptRepo[receiptKey];

    if (receipt.isEther) {
        payable(receipt.customer).transfer(receipt.amount);

        emit WithdrawEther(
            receiptKey,
            receipt.customer,
            receipt.amount,
            block.timestamp
        );
    } else {
        IERC20(receipt.token).safeTransfer(
            receipt.customer,
            receipt.amount
        );

        emit WithdrawErc20Token(
            receiptKey,
            receipt.customer,
            receipt.token,
            receipt.amount,
            block.timestamp
        );
    }
    delete hasReceipt[receiptKey];
    delete receiptRepo[receiptKey];
}

}
```

my test results:

[

1

1015×632 59.4 KB

](https://ethresear.ch/uploads/default/original/2X/a/a8d657d466f9a1b5a64e055051013f7646d79cc6.png)