Blockchain

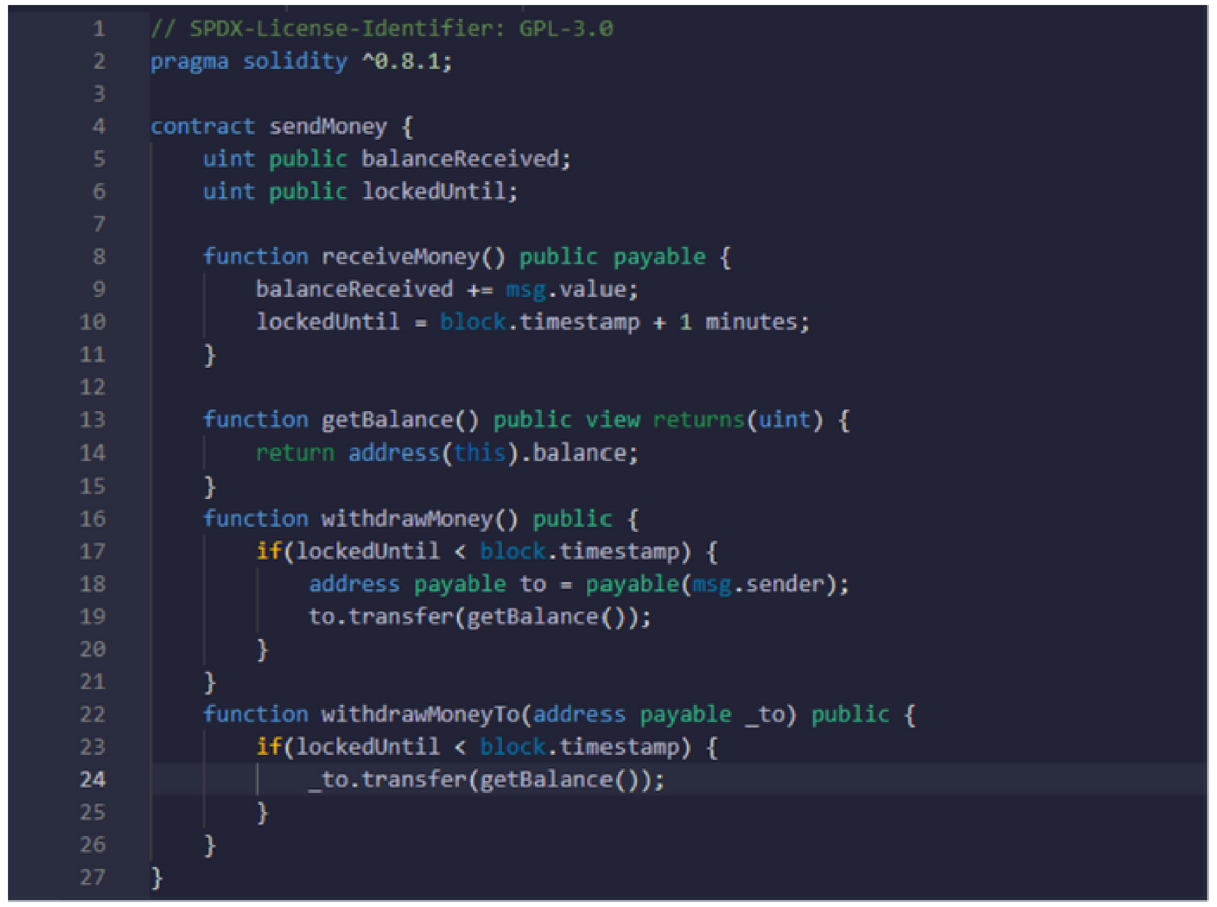
UTS

Technical Documentation

Muh Nardika | 1103184124

LAB 1
DEPOSIT / WITHDRAW ETHER

Source Code



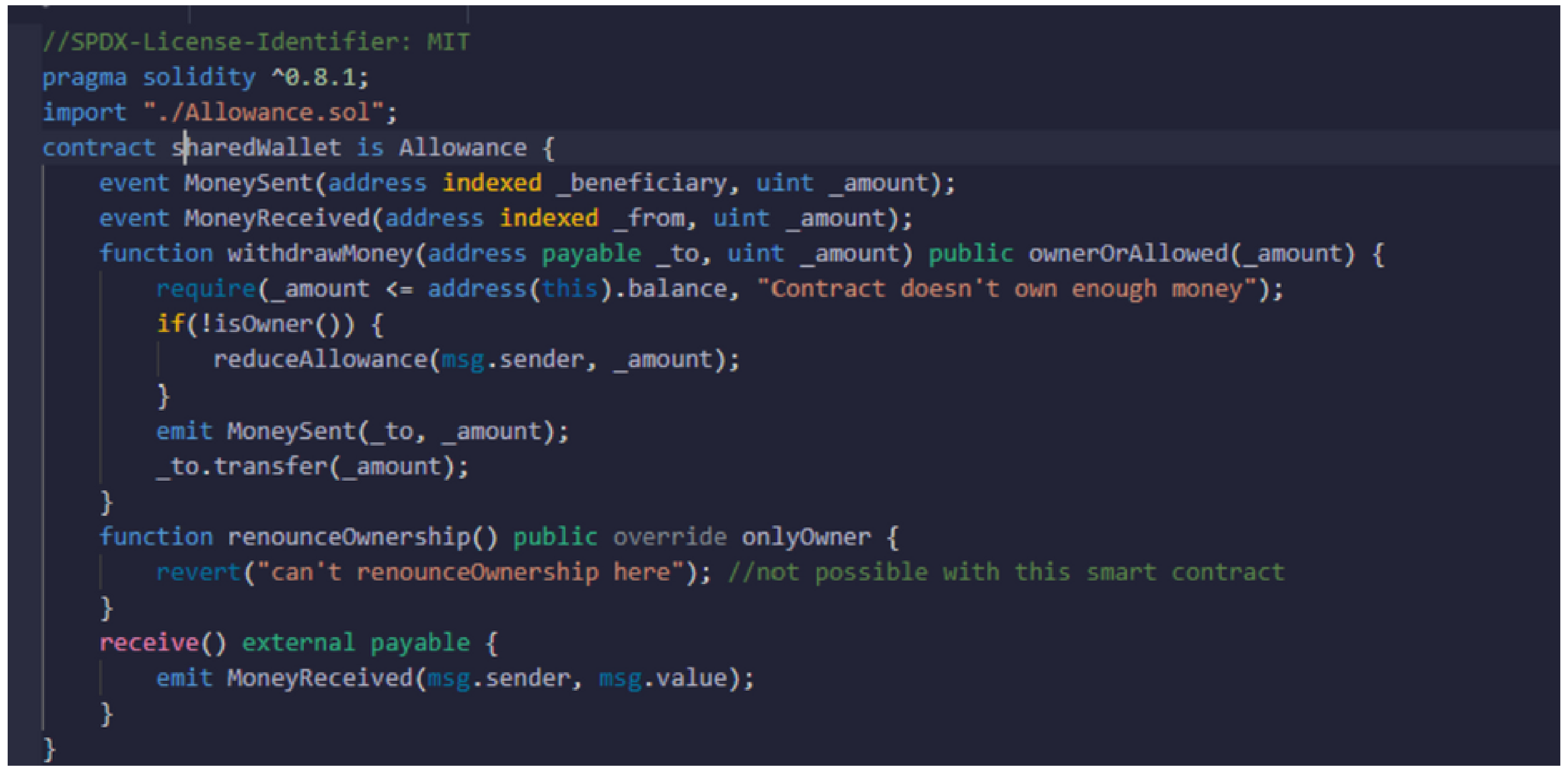
```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;

contract sendMoney {
    uint public balanceReceived;
    uint public lockedUntil;

    function receiveMoney() public payable {
        balanceReceived += msg.value;
        lockedUntil = block.timestamp + 1 minutes;
    }

    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
    function withdrawMoney() public {
        if(lockedUntil < block.timestamp) {
            address payable to = payable(msg.sender);
            to.transfer(getBalance());
        }
    }
    function withdrawMoneyTo(address payable _to) public {
        if(lockedUntil < block.timestamp) {
            _to.transfer(getBalance());
        }
    }
}
```

LAB 2
SHARED WALLET

- Bisa menyimpan dana dan membolehkan user untuk withdraw lagi.
- Membatasi fungsi ke user roles spesifik (Owner, User).
- Menggunakan kembali smart contract yang telah diaudit.

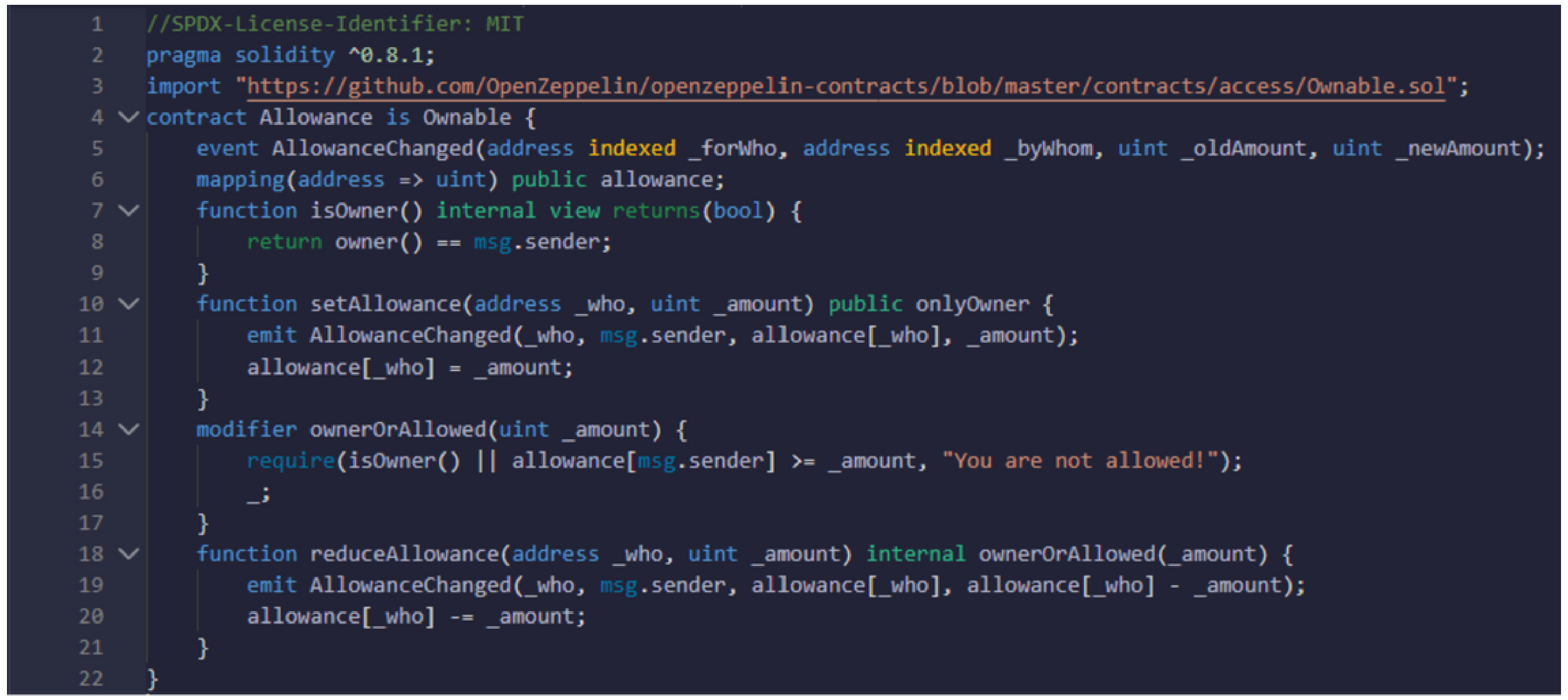
Source Code

Shared Wallet



```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "./Allowance.sol";
contract sharedWallet is Allowance {
    event MoneySent(address indexed _beneficiary, uint _amount);
    event MoneyReceived(address indexed _from, uint _amount);
    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        emit MoneySent(_to, _amount);
        _to.transfer(_amount);
    }
    function renounceOwnership() public override onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }
    receive() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

Allowance

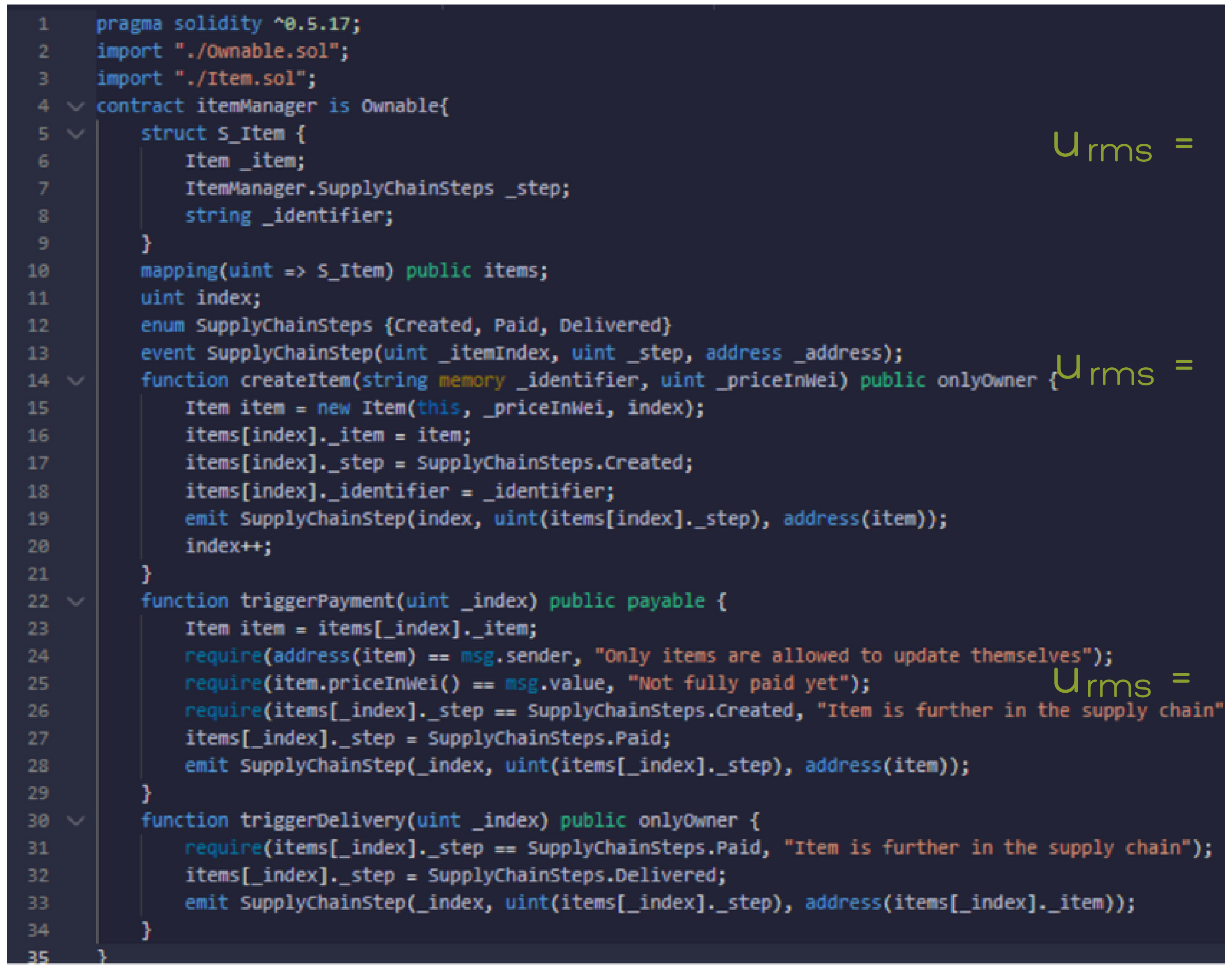


```solidity
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
contract Allowance is Ownable {
    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
    mapping(address => uint) public allowance;
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }
    function setAllowance(address _who, uint _amount) public onlyOwner {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
        allowance[_who] = _amount;
    }
    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }
    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount);
        allowance[_who] -= _amount;
    }
}
```

LAB 3
SUPPLY CHAIN

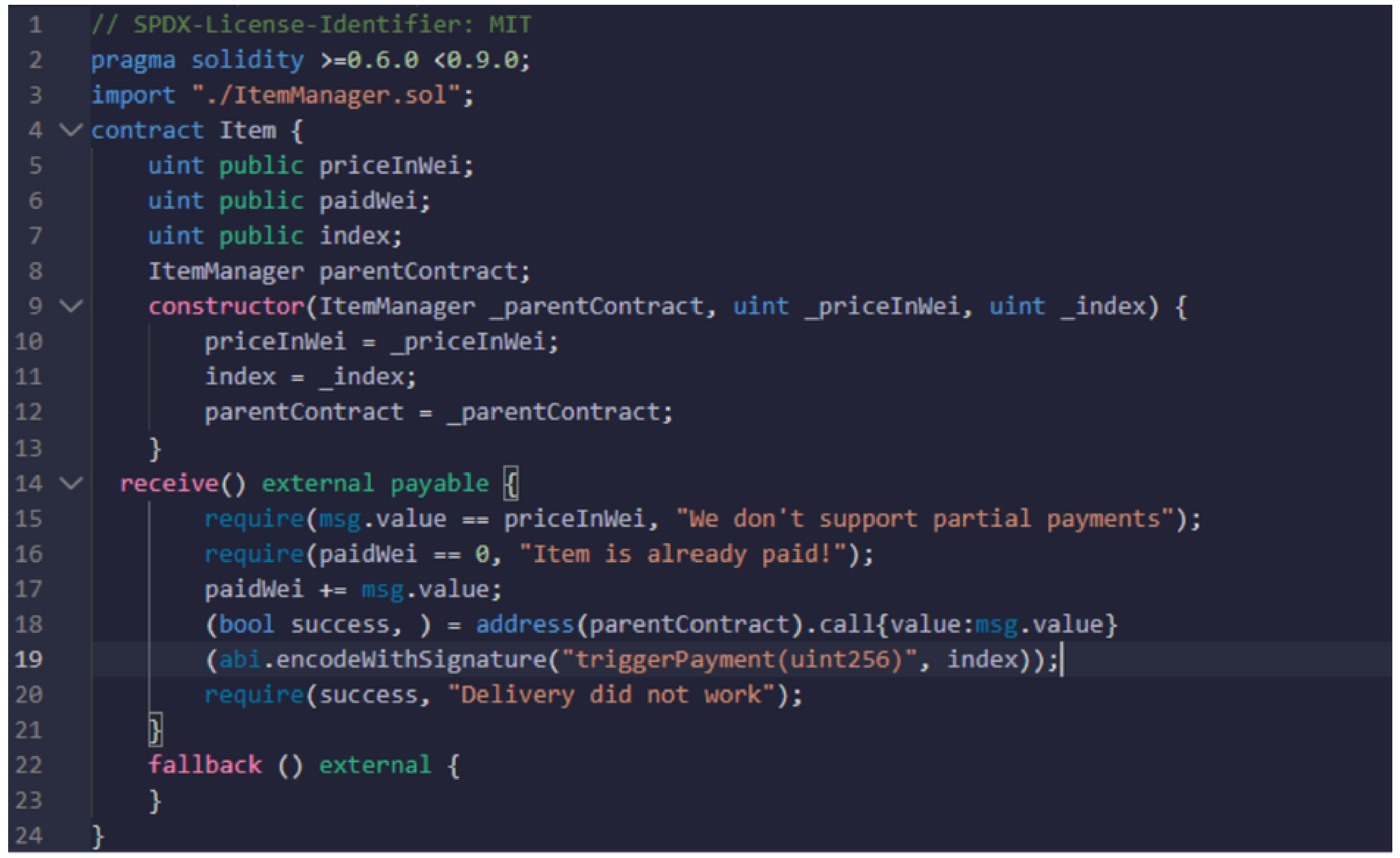
LAB INI AKAN MEMPELAJARI TENTANG SUPPLY CHAIN

SOURCE CODE



```solidity
pragma solidity ^0.5.17;
import "./Ownable.sol";
import "./Item.sol";
contract itemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
    mapping(uint => S_Item) public items;
    uint index;
    enum SupplyChainSteps {Created, Paid, Delivered}
    event SupplyChainStep(uint _itemIndex, uint _step, address _address);
    function createItem(string memory _identifier, uint _priceInWei) public onlyOwner {
        Item item = new Item(this, _priceInWei, index);
        items[index]._item = item;
        items[index]._step = SupplyChainSteps.Created;
        items[index]._identifier = _identifier;
        emit SupplyChainStep(index, uint(items[index]._step), address(item));
        index++;
    }
    function triggerPayment(uint _index) public payable {
        Item item = items[_index]._item;
        require(address(item) == msg.sender, "Only items are allowed to update themselves");
        require(item.priceInWei() == msg.value, "Not fully paid yet");
        require(items[_index]._step == SupplyChainSteps.Created, "Item is further in the supply chain"
        items[_index]._step = SupplyChainSteps.Paid;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(item));
    }
    function triggerDelivery(uint _index) public onlyOwner {
        require(items[_index]._step == SupplyChainSteps.Paid, "Item is further in the supply chain");
        items[_index]._step = SupplyChainSteps.Delivered;
        emit SupplyChainStep(_index, uint(items[_index]._step), address(items[_index]._item));
    }
}
```

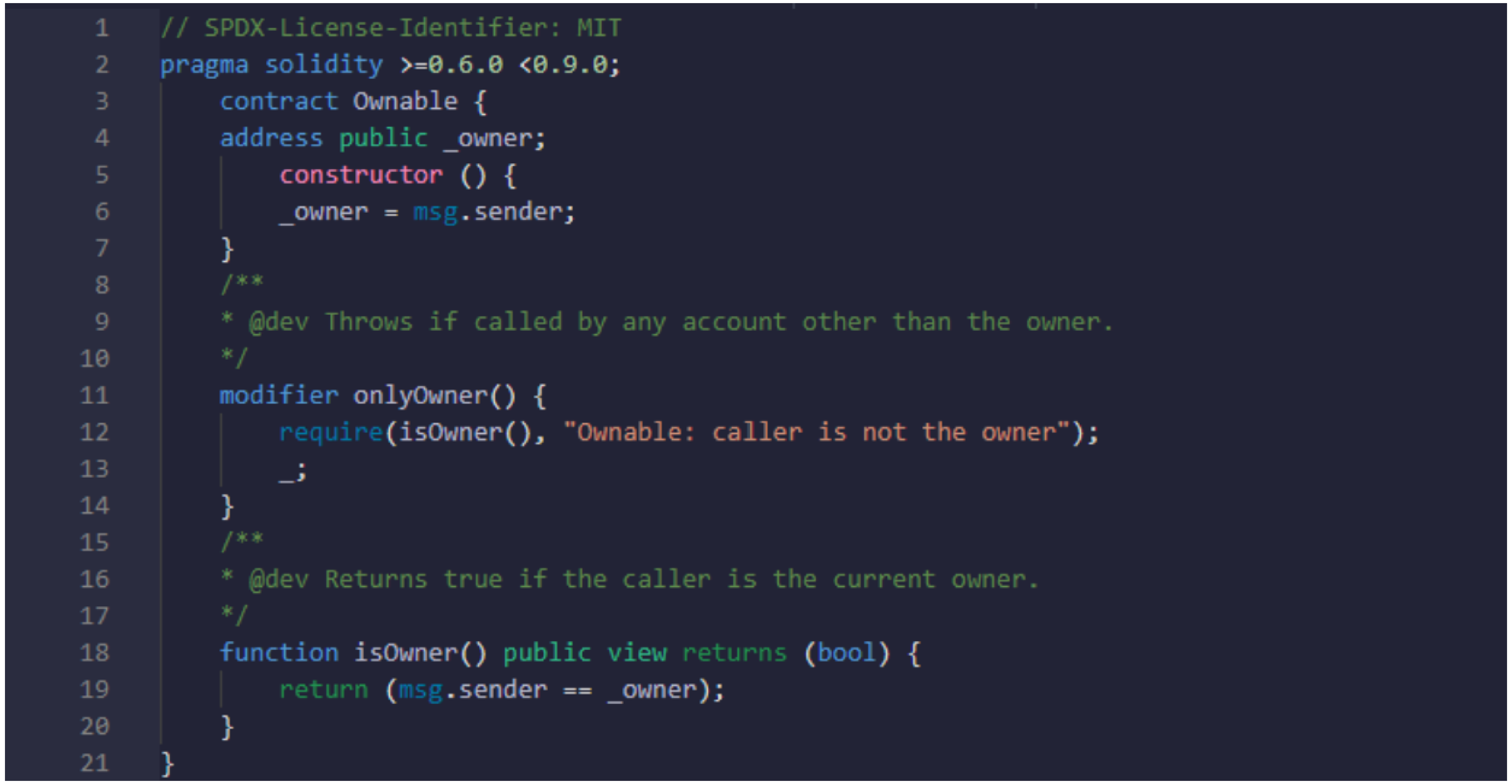
ITEM MANAGER



```solidity
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./ItemManager.sol";
contract Item {
    uint public priceInWei;
    uint public paidWei;
    uint public index;
    ItemManager parentContract;
    constructor(ItemManager _parentContract, uint _priceInWei, uint _index) {
        priceInWei = _priceInWei;
        index = _index;
        parentContract = _parentContract;
    }
  receive() external payable {
        require(msg.value == priceInWei, "We don't support partial payments");
        require(paidWei == 0, "Item is already paid!");
        paidWei += msg.value;
        (bool success, ) = address(parentContract).call{value:msg.value}
        (abi.encodeWithSignature("triggerPayment(uint256)", index));
        require(success, "Delivery did not work");
    }
    fallback () external {
    }
}
```

ITEM

SOURCE CODE



```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
    contract Ownable {
    address public _owner;
        constructor () {
        _owner = msg.sender;
    }
    /**
    * @dev Throws if called by any account other than the owner.
    */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }
    /**
    * @dev Returns true if the caller is the current owner.
    */
    function isOwner() public view returns (bool) {
        return (msg.sender == _owner);
    }
}
```

OWNABLE