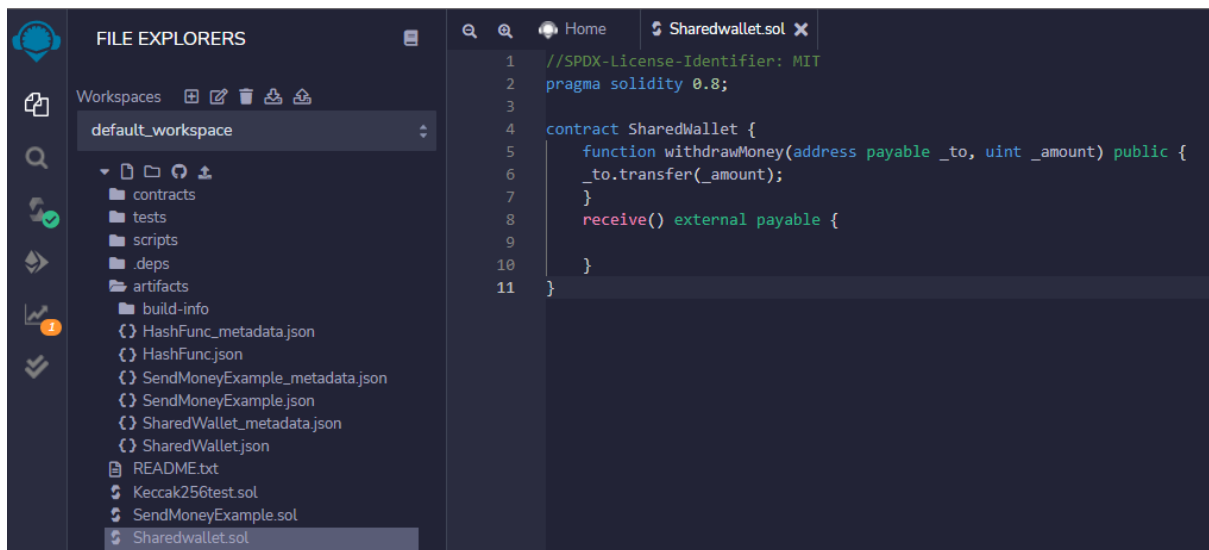


14.2 We Define the Basic Smart Contract

This is the very basic smart contract. It can receive Ether and it's possible to withdraw Ether, but all in all, not very useful quite yet. Let's see if we can improve this a bit in the next step.

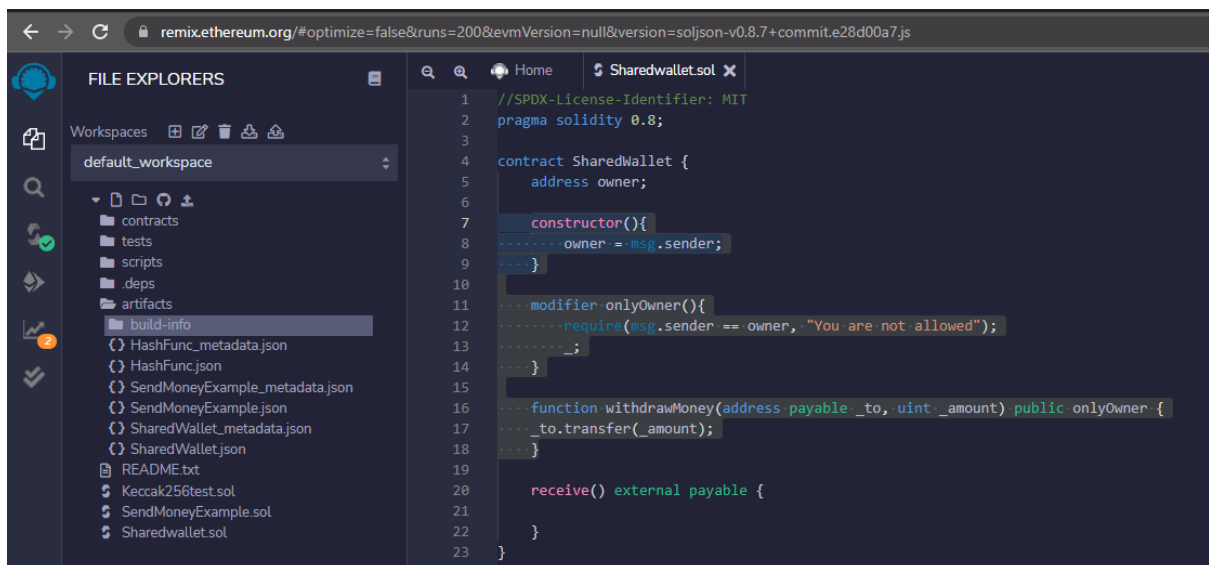


The screenshot shows the VS Code interface with the 'SharedWallet.sol' file open. The left sidebar displays the 'FILE EXPLORERS' view with a file tree for 'default_workspace'. The main editor area shows the following Solidity code:

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3
4 contract SharedWallet {
5     function withdrawMoney(address payable _to, uint _amount) public {
6         _to.transfer(_amount);
7     }
8     receive() external payable {
9     }
10 }
11 }
```

14.3 Permissions: Allow only the Owner to Withdraw Ether

In this step we restrict withdrawal to the owner of the wallet. How can we determine the owner? It's the user who deployed the smart contract.



The screenshot shows the VS Code interface with the 'SharedWallet.sol' file open, displaying the updated code that restricts withdrawal to the owner. The code is as follows:

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3
4 contract SharedWallet {
5     address owner;
6
7     constructor(){
8         owner = msg.sender;
9     }
10
11     modifier onlyOwner(){
12         require(msg.sender == owner, "You are not allowed");
13         _;
14     }
15
16     function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
17         _to.transfer(_amount);
18     }
19
20     receive() external payable {
21     }
22 }
23 }
```

Whatch out that you also add the "onlyOwner" modifier to the withdrawMoney function!

14.4 Use Re-Usable Smart Contracts from OpenZeppelin

Having the owner-logic directly in one smart contract isn't very easy to audit. Let's break it down into smaller parts and reuse existing audited smart contracts from OpenZeppelin for that. The latest OpenZeppelin contract does not have an `isOwner()` function anymore, so we have to create our own. Note that the `owner()` is a function from the `Ownable.sol` contract.

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
4
5 contract SharedWallet is Ownable {
6     function isOwner() internal view returns(bool) {
7         return owner() == msg.sender;
8     }
9     function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
10         _to.transfer(_amount);
11     }
12     receive() external payable {
13     }
14 }
```

14.5 Permissions: Add Allowances for External Roles

In this step we are adding a mapping so we can store `address => uint` amounts. This will be like an array that stores `[0x123546...]` an address, to a specific number. So, we always know how much someone can withdraw. We also add a new modifier that checks: Is it the owner itself or just someone with allowance?

```
1 pragma solidity 0.8;
2
3 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
4
5 contract SharedWallet is Ownable{
6
7     function isOwner() internal view returns(bool) {
8         return owner() == msg.sender;
9     }
10
11     mapping(address => uint) public allowance;
12
13     function addAllowance(address _who, uint _amount) public onlyOwner {
14         allowance[_who] = _amount;
15     }
16
17     modifier ownerOrAllowed(uint _amount) {
18         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
19         _;
20     }
21
22     function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
23         require(_amount <= address(this).balance, "Contract doesn't own enough money");
24         _to.transfer(_amount);
25     }
26
27     receive() external payable {
28
29     }
30 }
31
```

14.6 Improve/Fix Allowance to avoid Double-Spending

Without reducing the allowance on withdrawal, someone can continuously withdraw the same amount over and over again. We have to reduce the allowance for everyone other than the owner.

```
function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
    allowance[_who] -= _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);
    }
    _to.transfer(_amount);
}
```

14.7 Improve Smart Contract Structure

Now we know our basic functionality, we can structure the smart contract differently. To make it easier to read, we can break the functionality down into two distinct smart contracts.

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3
4 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
5
6 contract SharedWallet is Ownable{
7
8     function isOwner() internal view returns(bool) {
9         return owner() == msg.sender;
10    }
11
12    mapping(address => uint) public allowance;
13
14    function addAllowance(address _who, uint _amount) public onlyOwner {
15        allowance[_who] = _amount;
16    }
17
18    modifier ownerOrAllowed(uint _amount) {
19        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
20        _;
21    }
22
23    function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
24        allowance[_who] -= _amount;
25    }
26 }
27
28 contract SharedWallet is Allowance {
29     function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
30         require(_amount <= address(this).balance, "Contract doesn't own enough money");
31         if(!isOwner()) {
32             reduceAllowance(msg.sender, _amount);
33         }
34         _to.transfer(_amount);
35     }
36
37     receive() external payable {
38
39     }
40 }
```

14.8 Add Events in the Allowances Smart Contract

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
4
5 contract Allowance is Ownable {
6     event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
7     mapping(address => uint) public allowance;
8
9     function isOwner() internal view returns(bool) {
10         return owner() == msg.sender;
11     }
12
13     function setAllowance(address _who, uint _amount) public onlyOwner {
14         emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
15         allowance[_who] = _amount;
16     }
17
18     modifier ownerOrAllowed(uint _amount) {
19         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
20         _;
21     }
22
23     function reduceAllowance(address who, uint amount) internal ownerOrAllowed(_amount) {
24         emit AllowanceChanged(who, msg.sender, allowance[_who], allowance[_who] - _amount);
25         allowance[_who] -= _amount;
26     }
27 }
```

14.9 Add Events in the SharedWallet Smart Contract

Obviously we also want to have events in our shared wallet, when someone deposits or withdraws funds:

```
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);
    }

    receive() external payable {
        emit MoneyReceived(msg.sender, msg.value);
    }
}
```

14.10 Add the SafeMath Library safeguard Mathematical Operations

14.11 Remove the Renounce Ownership functionality

Now, let's remove the function to remove an owner. We simply stop this with a revert. Add the following function to the SharedWallet:

```
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);
    }
}
```

14.12 Move the Smart Contracts into separate Files

As a last step, let's move the smart contracts into separate files and use import functionality:

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8;
3
4 import "../Allowance.sol";
5
6 contract SharedWallet is Allowance {
7     event MoneySent(address indexed _beneficiary, uint _amount);
8     event MoneyReceived(address indexed _from, uint _amount);
9
10    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
11        require(_amount <= address(this).balance, "Contract doesn't own enough money");
12        if(!isOwner()) {
13            reduceAllowance(msg.sender, _amount);
14        }
15        emit MoneySent(_to, _amount);
16        _to.transfer(_amount);
17    }
18
19    function renounceOwnership() public override onlyOwner {
20        revert("can't renounceOwnership here"); //not possible with this smart contract
21    }
22
23    receive() external payable {
24        emit MoneyReceived(msg.sender, msg.value);
25    }
26 }
```

```
1 //SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8;
4
5 import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
6
7 contract Allowance is Ownable {
8
9     event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, uint _newAmount);
10     mapping(address => uint) public allowance;
11
12     function isOwner() internal view returns(bool) {
13         return owner() == msg.sender;
14     }
15
16     function setAllowance(address _who, uint _amount) public onlyOwner {
17         emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
18         allowance[_who] = _amount;
19     }
20
21     modifier ownerOrAllowed(uint _amount) {
22         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
23         _;
24     }
25
26     function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
27         emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount);
28         allowance[_who] -= _amount;
29     }
30 }
```