

# Technial Documentation

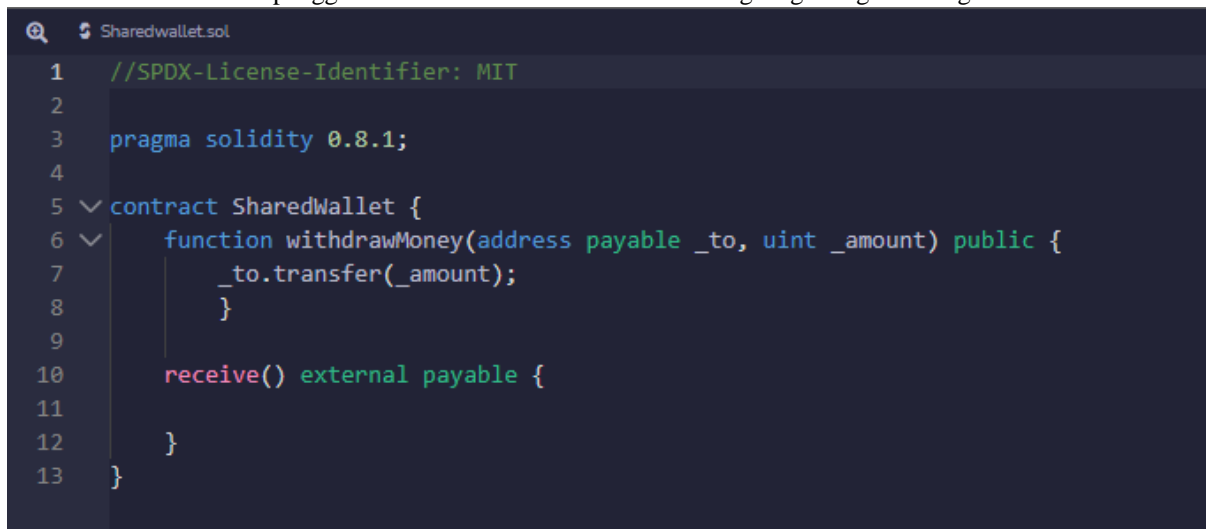
## LAB 2 : Shared Wallet

By : Fajri Nurfauzan | 1103180184

Pada lab kedua untuk UTS ini akan mempelajari cara sebuah Wallet yang dapat menyimpan dana dan memungkinkan penggunaanya untuk menarik kembali dananya dengan contoh kasus seperti :

- Tunjangan Anak per hari/minggu/bulan untuk dapat membelanjakan dana dalam jumlah tertentu.
- Majikan memberi karyawan tunjangan untuk biaya perjalanan mereka.
- Bisnis memberi kontraktor tunjangan untuk menghabiskan anggaran tertentu.

Setelah memahami teori penggunaan dan contoh studi kasus bisa langsung mengikuti langkah berikut :



```
1 //SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract SharedWallet {
6     function withdrawMoney(address payable _to, uint _amount) public {
7         _to.transfer(_amount);
8     }
9
10    receive() external payable {
11
12    }
13 }
```

Buat file remix sederhana seperti berikut dan beri nama SharedWallet.sol

Ini adalah kontrak pintar (*Smart Contract*) yang sangat mendasar yang dapat menerima Eter dan dimungkinkan untuk menarik Eter, tetapi secara keseluruhan, sebenarnya ini tidak terlalu berguna atau cukup belum digunakan secara kasus.

```

1 //SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 contract SharedWallet {
6
7     address owner;
8
9     constructor() {
10         owner = msg.sender;
11     }
12
13     modifier onlyOwner() {
14         require(msg.sender == owner, "You are not allowed");
15         _;
16     }
17
18     function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
19         _to.transfer(_amount);
20     }
21
22     receive() external payable {
23
24     }
25 }

```

Pada langkah ini dapat dilihat code akan membatasi penarikan kepada pemilik dompet. Bagaimana cara menentukan pemiliknya? Itu adalah dengan pengguna yang menyebarkan kontrak pintar (*Smart Contract*). Ketahuilah bahwa code yang sudah diketik perlu menambahkan pengubah "onlyOwner" ke fungsi withdrawMoney!

```

//SPDX-License-Identifier: MIT

pragma solidity 0.8.1;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet {

    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }

    function withdrawMoney(address payable _to, uint _amount) public {
        _to.transfer(_amount);
    }

    receive() external payable {

    }
}

```

Untuk memiliki logika seperti pemilik secara langsung dalam satu kontrak pintar (*Smart Contract*) merupakan hal yang tidak mudah untuk diaudit. Maka dari itu perlu diuraikan menjadi bagian-bagian yang lebih kecil dan akan digunakan kembali kontrak pintar (*Smart Contract*) yang sudah diaudit dari OpenZeppelin untuk itu. Kontrak OpenZeppelin terbaru tidak memiliki isOwner() fungsi lagi, jadi kita harus membuat sendiri. Perhatikan bahwa pemilik() adalah fungsi dari kontrak Ownable.sol.

```

1  //SPDX-License-Identifier: MIT
2
3  pragma solidity 0.8.1;
4
5  import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
6
7  contract SharedWallet is Ownable {
8
9      function isOwner() internal view returns(bool) {
10         return owner() == msg.sender;
11     }
12
13     mapping(address => uint) public allowance;
14
15     function addAllowance(address _who, uint _amount) public onlyOwner {
16         allowance[_who] = _amount;
17     }
18
19     modifier ownerOrAllowed(uint _amount) {
20         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
21         _;
22     }
23
24     function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
25         require(_amount <= address(this).balance, "Contract doesn't own enough money");
26         _to.transfer(_amount);
27     }
28
29     receive() external payable {
30     }
31 }

```

Pada langkah ini perlu menambahkan pemetaan sehingga code dapat menyimpan alamat => jumlah uint. Code ini akan seperti array yang menyimpan [0x123546...] alamat, ke nomor tertentu. Jadi, code selalu tahu berapa banyak yang bisa ditarik seseorang. dan juga menambahkan yang baru modifikator yang akan memeriksa: Apakah pemiliknya sendiri atau hanya seseorang dengan uang saku?

Tanpa mengurangi tunjangan penarikan, seseorang dapat terus menerus menarik jumlah yang sama berulang-ulang

```

1  //SPDX-License-Identifier: MIT
2
3  pragma solidity 0.8.1;
4
5  import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable.sol";
6
7  contract Allowance is Ownable {
8
9      function isOwner() internal view returns(bool) {
10         return owner() == msg.sender;
11     }
12
13     mapping(address => uint) public allowance;
14
15     function setAllowance(address _who, uint _amount) public onlyOwner {
16         allowance[_who] = _amount;
17     }
18
19     modifier ownerOrAllowed(uint _amount) {
20         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
21         _;
22     }
23
24     function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
25         allowance[_who] -= _amount;
26     }
27
28 }
29
30 contract SharedWallet is Allowance {
31     function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
32         require(_amount <= address(this).balance, "Contract doesn't own enough money");
33         if(!isOwner()) {
34             reduceAllowance(msg.sender, _amount);
35         }
36         _to.transfer(_amount);
37     }
38
39     receive() external payable {
40     }
41 }

```

Sekarang setelah mengetahui fungsi dasar hal selanjutnya dapat menyusun kontrak pintar (*Smart Contract*) secara berbeda. Untuk membuatnya lebih mudah dibaca, mungkin dengan bisa mengistirahatkan fungsionalitas menjadi dua kontrak pintar (*Smart Contract*) yang berbeda.

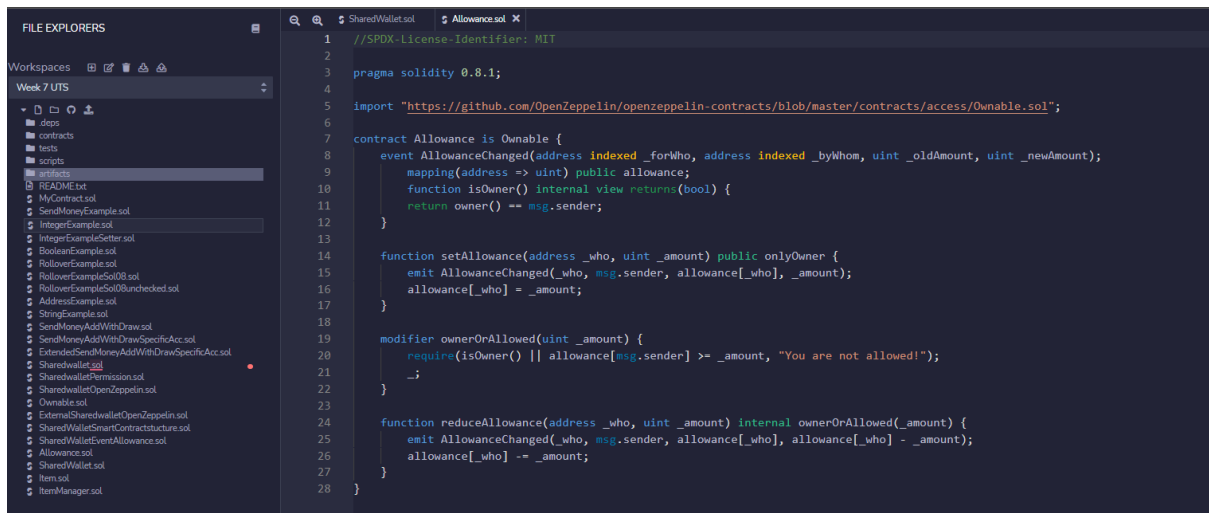
```

1 //SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
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
11     mapping(address => uint) public allowance;
12
13     function isOwner() internal view returns(bool) {
14         return owner() == msg.sender;
15     }
16
17     function setAllowance(address _who, uint _amount) public onlyOwner {
18         emit AllowanceChanged(_who, msg.sender, allowance[_who], _amount);
19         allowance[_who] = _amount;
20     }
21
22     modifier ownerOrAllowed(uint _amount) {
23         require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
24         _;
25     }
26
27     function reduceAllowance(address _who, uint _amount) internal ownerOrAllowed(_amount) {
28         emit AllowanceChanged(_who, msg.sender, allowance[_who], allowance[_who] - _amount);
29         allowance[_who] -= _amount;
30     }
31 }
32
33 contract SharedWallet is Allowance {
34
35     event MoneySent(address indexed _beneficiary, uint _amount);
36     event MoneyReceived(address indexed _from, uint _amount);
37
38     function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
39         require(_amount <= address(this).balance, "Contract doesn't own enough money");
40         if(!isOwner()) {
41             reduceAllowance(msg.sender, _amount);
42         }
43         emit MoneySent(_to, _amount);
44         _to.transfer(_amount);
45     }
46
47     receive() external payable {
48         emit MoneyReceived(msg.sender, msg.value);
49     }
50 }

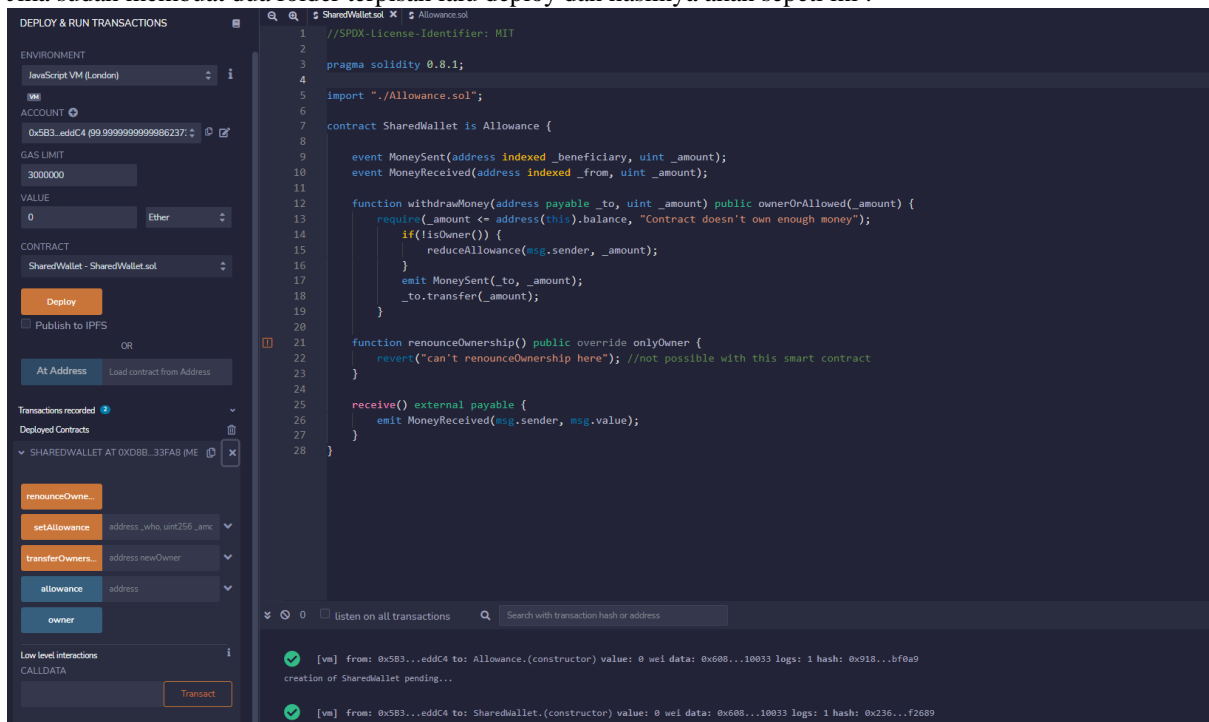
```

Dan terakhir ini merupakan fungsi penambahan event Allowance dan SharedWallet setelah mengetahui dasar bisa mencoba untuk membuat Smart Contract dengan folder terpisah dan gunakan fungsi impor dan akan menjadi seperti ini :





Jika sudah membuat dua folder terpisah lalu deploy dan hasilnya akan seperti ini :



DEPLOY & RUN TRANSACTIONS

ENVIRONMENT  
JavaScript VM (London)

ACCOUNT  
0x5B3...eddC4 (99.99999999999994998)

GAS LIMIT  
3000000

VALUE  
0 Ether

CONTRACT  
Allowance - Allowance.sol

Deploy

☐ Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded

Deployed Contracts

ALLOWANCE AT 0xD91...39138 (MEMO)

renounceOwne...

setAllowance address \_who, uint256 \_amc

transferOwners... address newOwner

allowance address

owner

Low level interactions

CALLDATA

Transact

SharedWallet.sol x Allowance.sol

```
1 //SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.1;
4
5 import "../Allowance.sol";
6
7 contract SharedWallet is Allowance {
8
9     event MoneySent(address indexed _beneficiary, uint _amount);
10    event MoneyReceived(address indexed _from, uint _amount);
11
12    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed(_amount) {
13        require(_amount <= address(this).balance, "Contract doesn't own enough money");
14        if(!isOwner()) {
15            reduceAllowance(msg.sender, _amount);
16        }
17        emit MoneySent(_to, _amount);
18        _to.transfer(_amount);
19    }
20
21    function renounceOwnership() public override onlyOwner {
22        revert("can't renounceOwnership here"); //not possible with this smart contract
23    }
24
25    receive() external payable {
26        emit MoneyReceived(msg.sender, msg.value);
27    }
28 }
```

0 ☐ listen on all transactions

Search with transaction hash or address

✓ [vm] from: 0x5B3...eddC4 to: SendMoneyExample.withdrawMoneyTo(address) 0xd91...39138 value: 0 wei data: 0x0ff...c02db logs: 0 creation of Allowance pending...

✓ [vm] from: 0x5B3...eddC4 to: Allowance.(constructor) value: 0 wei data: 0x608...10033 logs: 1 hash: 0x918...bf0a9