Etherium Blockchain Developer Guide

Raudhatul Rafiqah Assyahiddini

1103180225

UTS Blockchain

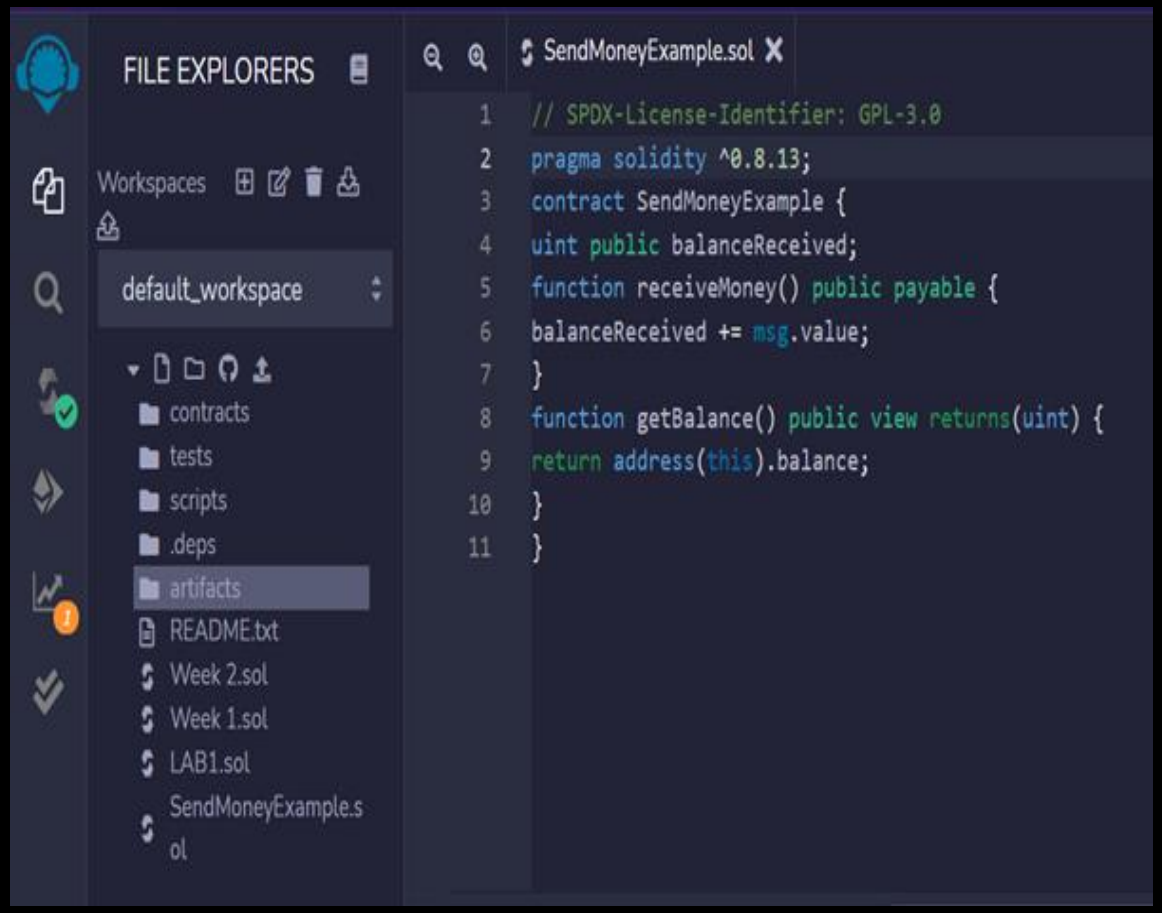
LAB 1: 8. Deposit/Withdraw Ether

Bagian ini akan mempelajari tentang bagaimana cara membuat smart contract yang akan mengatur keuangan anda. Anda akan mengirim Ether pada smart contract anda, lalu smart contract akan mengatur Ethernya dan dapat dikirim kepada siapapun. Ini seperti akun bank dengan code programming di dalamnya. Bisa digunakan sebagai escrow ether kedalam smart contract. Pertama yang kita perlukan adalah contoh setor dan Tarik tunai secara simple, lalu saya akan menunjukan kepada anda bagaimana smartcontract dapat mengunci sebuah dana menggunakan aktivasi waktu.

Smart Contract

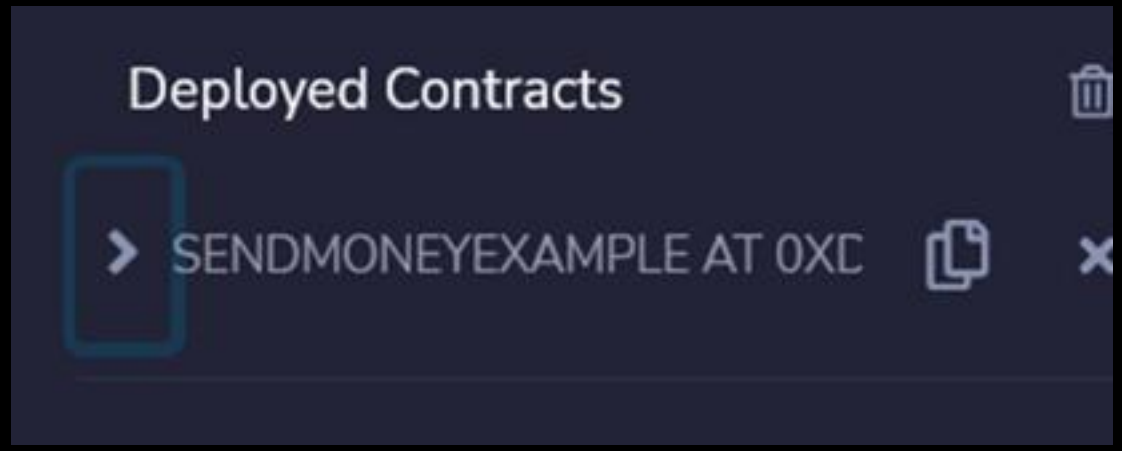
Dibuat di file (SendMoneyExample.sol) di remix dan paste kode ini

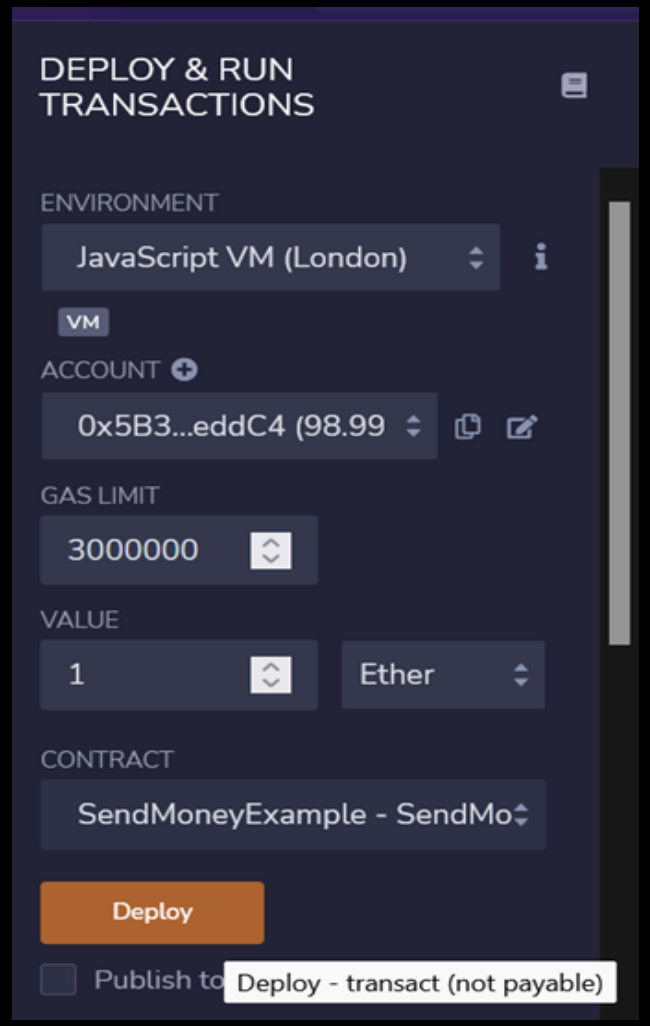
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.13;
contract SendMoneyExample {
uint public balanceReceived;
function receiveMoney() public payable {
balanceReceived += msg.value;
}
function getBalance() public view returns(uint) {
return address(this).balance;
}
}
```

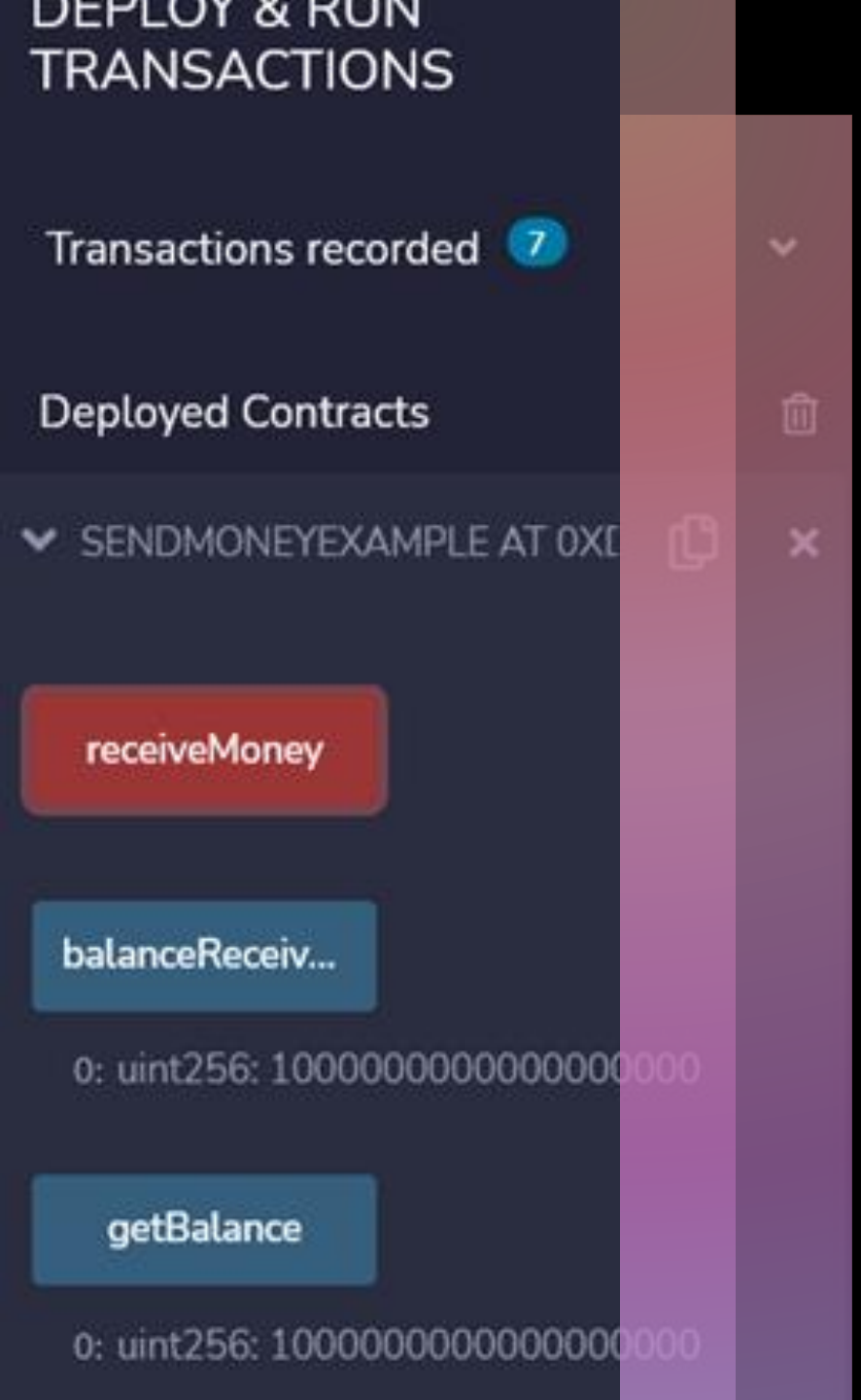


Deploy and Use the Smart Contract

1. Setelah itu, program akan di deploy, seperti pada gambar disamping.
1. Klik bagian panah untuk membuka step / button selanjutnya, seperti gambar dibawah



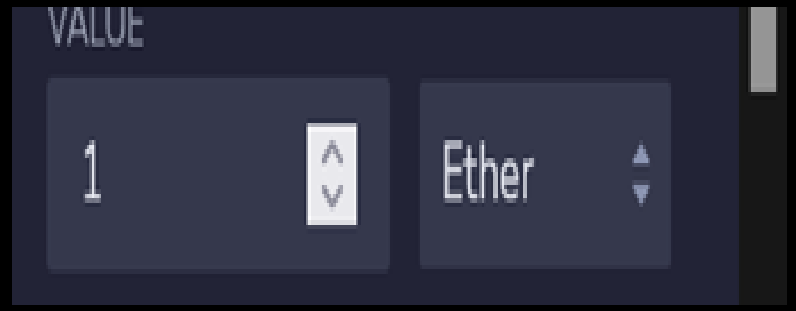




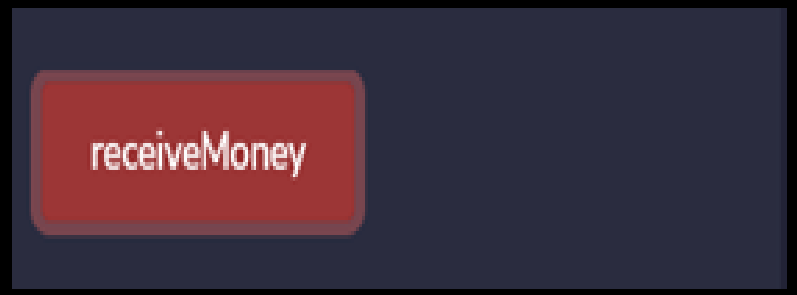
- Bagian tadi akan memunculkan step Untuk melihat
- smart contract yang telah terdeploy,
- setelah itu kita klik button yang ada pada deployed contracts
- atau ketiga button disamping

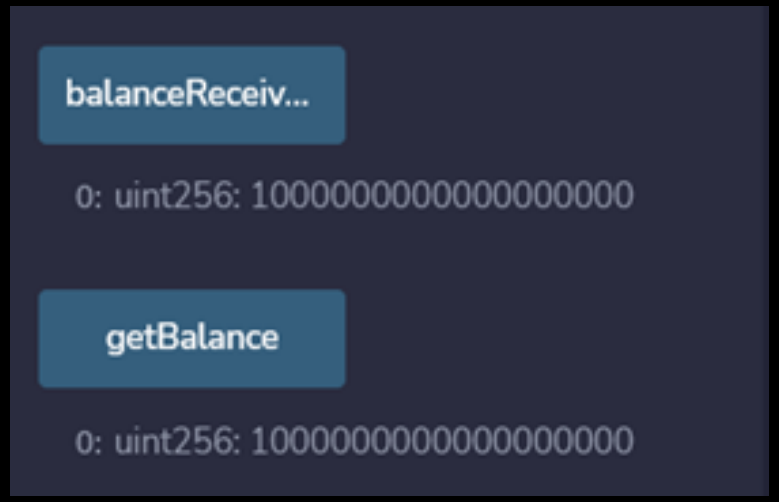
Send Ether To The Smart Contract

- Untuk dapat mengirim ether ke dalam Smart Contract kita harus memasukan value yang akan kita kirim ke alamat Ethereum kita. Pertama scroll keatas dan kita dapat lihat value , lalu rubah nilai value menjadi 1 dan ‘wei’ menjadi ether.

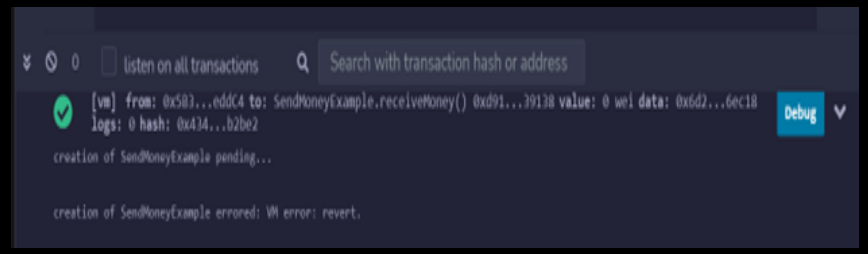


- Lalu scroll kebawah ke deployed contract dan kita dapat melihat dan klik tombol merah “receiveMoney”





- Cek saldo pada smart contract



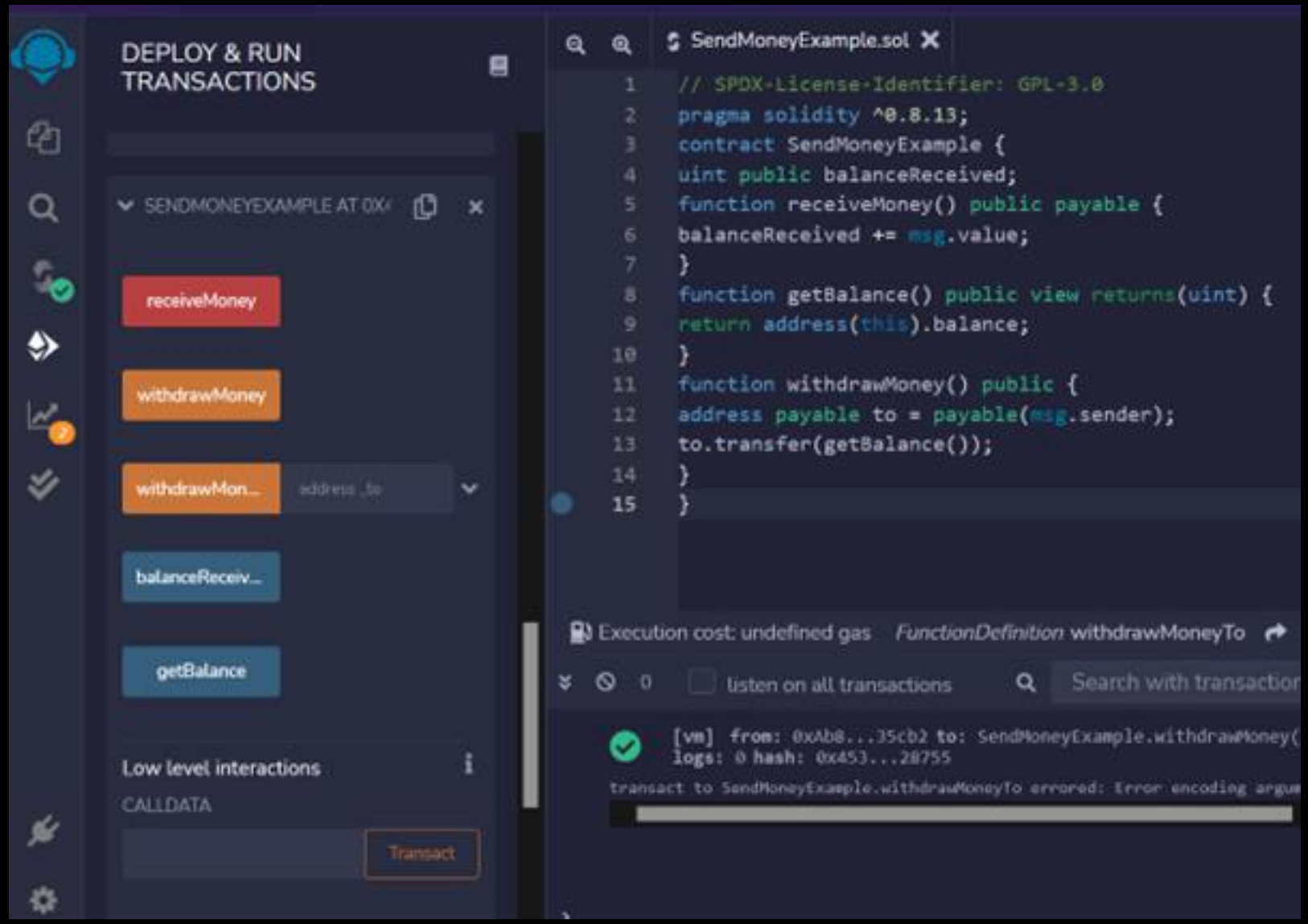
- Status berhasil tertulis pada bagian bawah program tadi

Withdraw Ether From Smart Contract

- 8.4.1 Add a Withdraw Function
- pada bagian ini akan ditambah
- kan bagian withdraw function

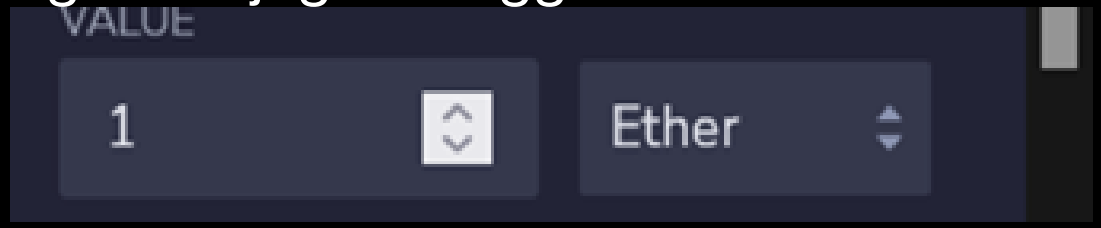
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.1;
contract SendMoneyExample {
uint public balanceReceived;
function receiveMoney() public payable {
balanceReceived += msg.value;
}
function getBalance() public view returns(uint) {
return address(this).balance;
}
function withdrawMoney() public {
address payable to = payable(msg.sender);
to.transfer(getBalance());
}
}
```

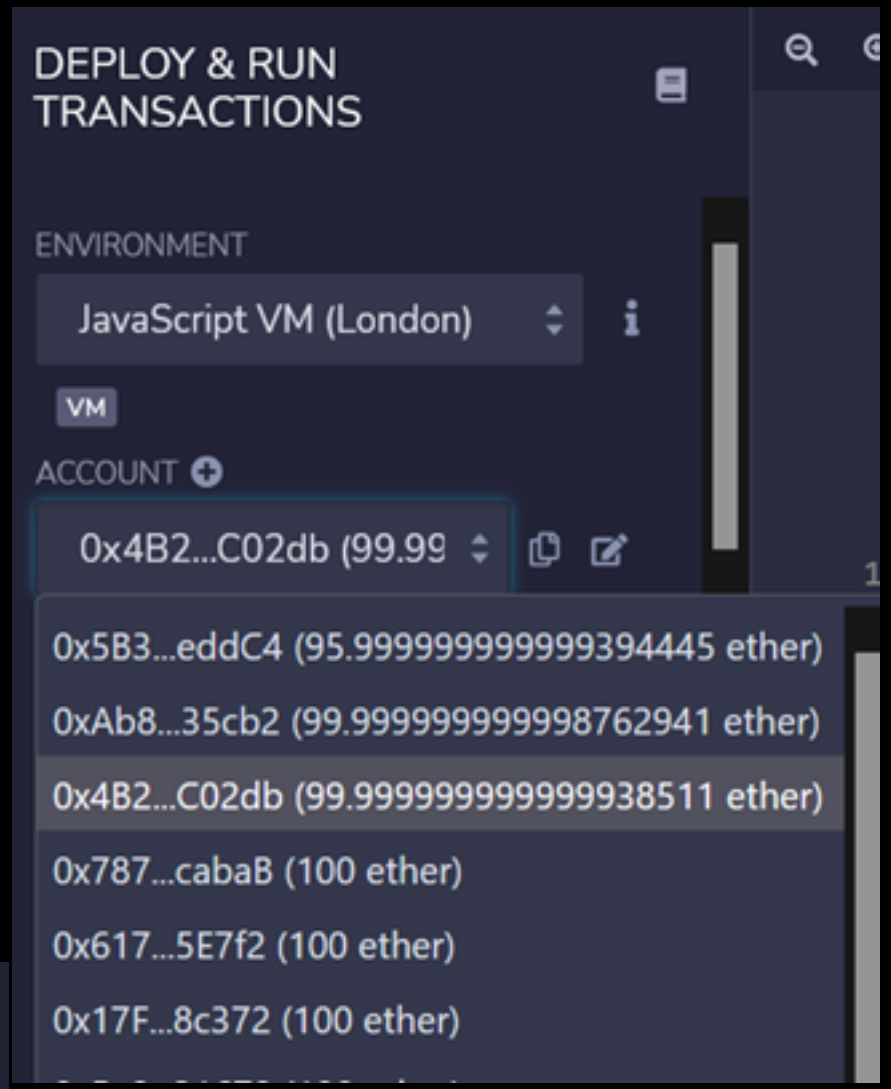
Deploy Kembali Smart Contract Kita dan kirim satu ether ke akun dan cek Kembali apakah ether sudah di terima

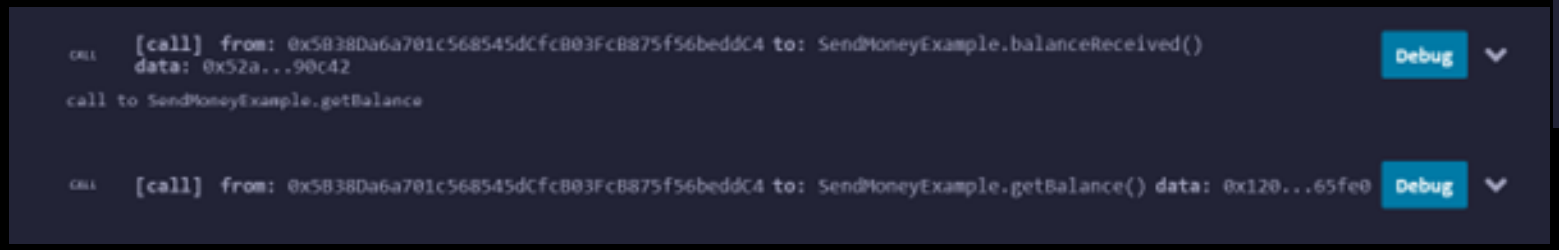


Menggunakan akun ke 3 pada list akun saat test fungsi withdrawMoneyTo, seperti pada gambar disamping.

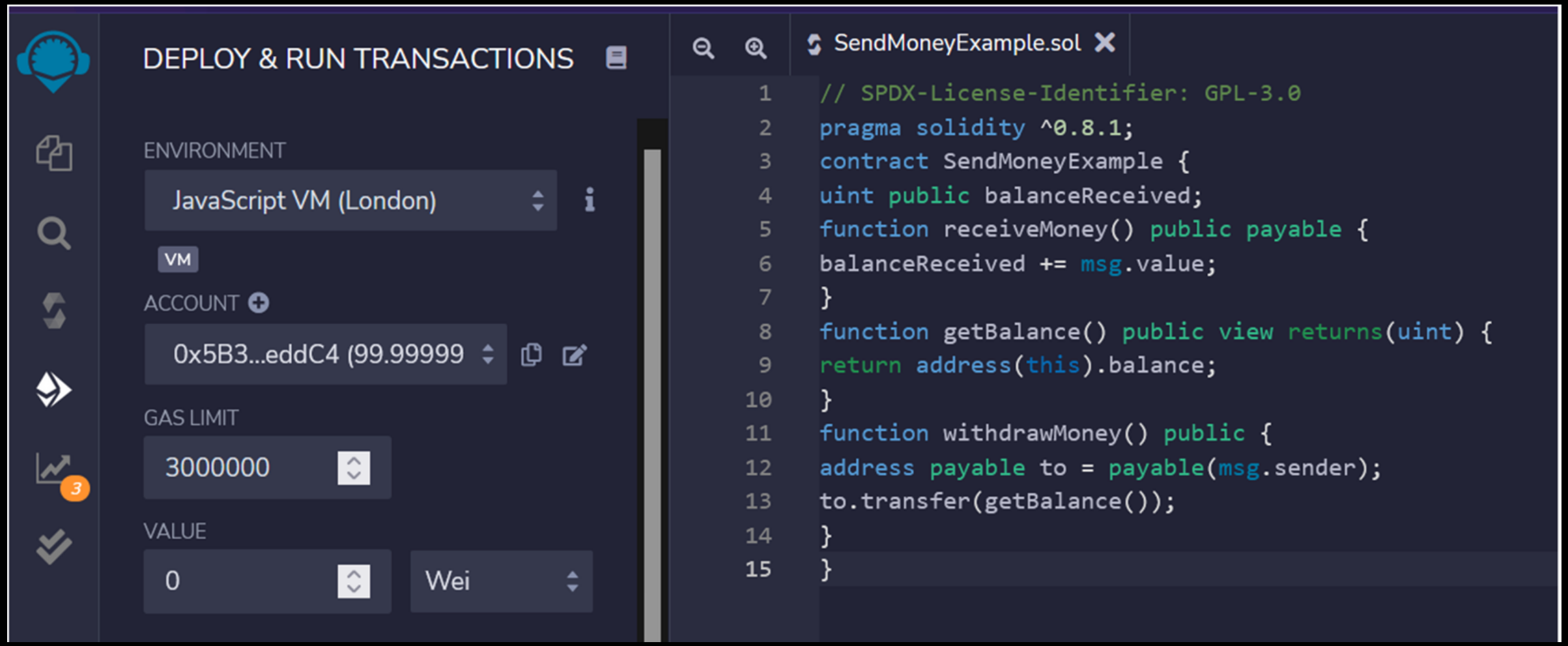
bagian ini juga mengganti nilai 0 value menjadi 1

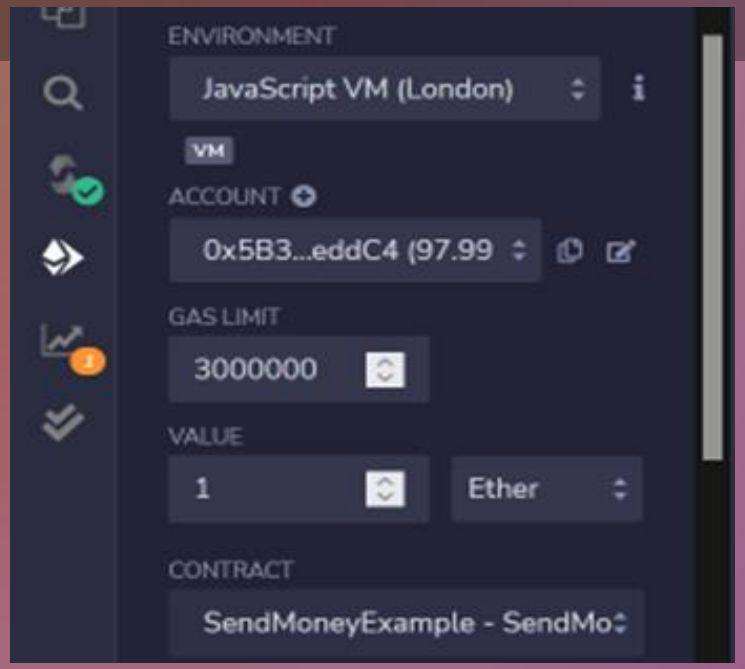






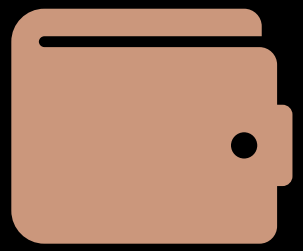
Withdraw To Specific Account





- lakukan Deploy ulang, Lalu ulangi lagi pengecekan value yang diganti menjadi 1 ether tadi Kembali sampai nilai account terganti nilainya.
- copy code pada account dan paste disebelah withdrawMoneyTo seperti dibawah

LAB 2: 14. Shared Wallet



- Pada bagian ini, kita akan mempelajari mengenai bagaimana Memiliki sebuah “on-chain wallet smart contract”. Contract wallet dapat menyimpan saldo dan mengizinkan user untuk mengambil dana. Dapat memberikan tunjangan ke orang lain atau pada spesifik user berdasarkan alamat si user. Menggunakan Kembali smart contract yang telah di buat sebelumnya

- Definisikan smart contract terlebih dahulu dan diberi nama Sharedwallet.sol seperti pada ss program dibawah ini

Home | Sharedwallet.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
contract SharedWallet {
    function withdrawMoney(address payable _to, uint _amount) public {
_to.transfer(_amount);
}
receive() external payable {
} }
```

- Pada code dibawah ini, kita juga dapat menambahkan fungsi “onlyOwner” untuk merubah ke fungsi “withdrawMoney”

Home | Sharedwallet.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
contract SharedWallet {
    address owner;

    constructor() {
owner = msg.sender;
}
    modifier onlyOwner() {
        require(msg.sender == owner, "You are not allowed");
_;
}
    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
_to.transfer(_amount);
}
    receive() external payable {
}}
```

- Mengizinkan

- Pada langkah ini kita dapat membatasi pengeluaran saldo ke pemilik wallet

Home | Sharedwallet.sol

```
//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);
    }
}
```

- Menggunakan Kembali kontrak dari OpenZeppllin
- Mempunyai logika “owner-logic” langsung didalam smart contract bukan lah hal yang mudah untuk di audit. Maka dari itu cobalah untuk memecah menjadi bagian-bagian kecil dan menggunakan smart contract yang telah di audit dari OpenZeppelin. Pada build OpenZepplin yang terbaru sudah tidak memiliki fungsi “isOwner” maka dari itu kita menambahkannya sendiri.

Home | Sharedwallet.sol

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownal
    contract SharedWallet is Ownable {
        function isOwner() internal view returns(bool) {
    return owner() == msg.sender;
}
        function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
_to.transfer(_amount);
}
    receive() external payable {
}
}
```

- Menambahkan pengeluaran untuk roles luar

- Pada Langkah ini kita menambahkan mapping, jadi kita dapat menyimpan address => uint amounts. Ini akan seperti array Ketika disimpan.

Home | Sharedwallet.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "https:github.com/OpenZeppelin-contracts/blob/master.contracts/access/ownable.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);
    }
}
```

- Definisikan smart contract terlebih dahulu dan diberi nama Allowance.sol
- seperti pada ss program dibawah ini

Home | Allowance.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownal
contract Allowance is Ownable {
    event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint _oldAmount, ui
    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;
    }
}
```

- Menambahkan event di dalam Allowance Smart Contract

Home | Sharedwallet.sol | Allowance.sol

```
//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 Wallet

Pada bagian ini, kita akan mempelajarin atau memahami bagaimana fungsi tingkat rendah address.call.value()()

Pahami Alur Kerja dengan Truffle

Memahami Pengujian Unit dengan Truffle

Memahami Events dalam HTML

```
//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);
    }
}
```

- Definisikan smart contract terlebih dahulu dan diberi nama ItemManager.sol seperti pada ss program dibawah ini

Smart Contract ItemManager

- Pertama kita membutuhkan Smart Contract ItemManager

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./Ownable.sol";
import "./Item.sol";
contract ItemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
```

- Item.sol

```
// 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
        require(success, "Delivery did not work");
    }
    fallback () external {
    }
}
```

Smart Contract Item

- Kita akan membuat satu Smart Contract lagi yang bernama Item

```
// 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)", inc
        require(success, "Delivery did not work");
    }
    fallback () external {
    }
}
```

- Fungsi kepemilikan
- Owanable.sol

```
// 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);
    }
}
```

- Lalu kita rubah sedikit pada smartcontract “ItemManager” kita dan kita set untuk dapat di eksekusi oleh pemilik saja

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
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;
```

Install Truffle

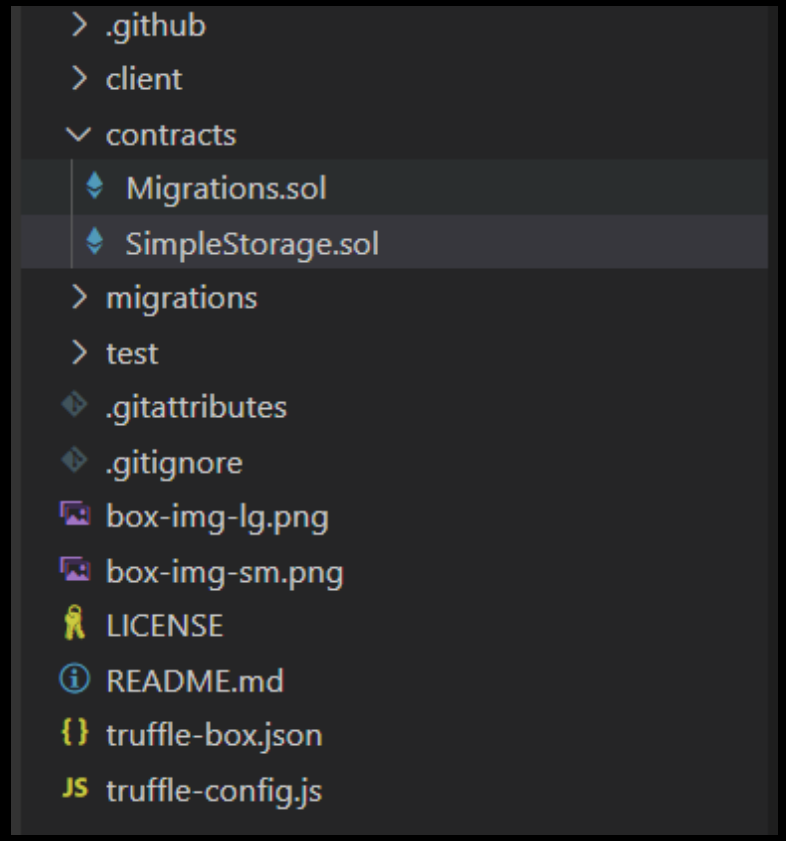
- Untuk meninstall Truffle pada windows, kita menginstal berbasiskan CLI bisa menggunakan Windows Powershell dengan mengetikkan " npm install -g npm@8.7.0 "
- Lalu buat folder disini dengan penamaan "s06-eventtrigger"
- Lalu unbox react boxnya

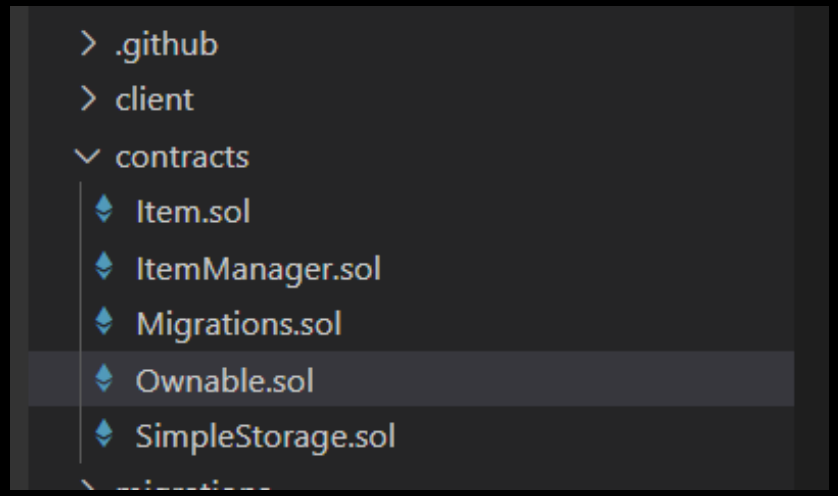
```
npm install truffle
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements

PS C:\Users\62812> npm install -g npm@8.7.0

changed 14 packages, and audited 197 packages in 11s

10 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS C:\Users\62812> npm install -g truffle
[                    ] | idealTree:ganache: sill
```

```
ers\62812> mkdir s06-eventtrigger


tory: C:\Users\62812


                LastWriteTime         Length Nam
                -------------         ------ ---
        4/22/2022   5:48 AM                s06


ers\62812>
```

- Langkah selanjutnya kita buka visual studio code. Setelah di buka arahkan ke folder yang kita sudah buat sebelumnya lalu di dalam folder contract hapus file “SimpleStorage.sol”

- Setelah mengghapus file “SimpleStorage.sol” kita masukan kontrak kita yang sebelumnya sudah kita siapkan dari remix. Jika ada pesan error maka kita abaikan saja terlebih dahulu.





- Setelah itu kita dapat mengubah di envoirment kita agar dapat di compile

Item.sol | ItemManager.sol | Ownable.sol | 2_deploy_contra

migrations > 2_deploy_contracts.js > SimpleStorage

```javascript
var SimpleStorage = artifacts.require("./ItemManager.sol");

module.exports = function(deployer) {
  deployer.deploy(SimpleStorage);
};

```

truffle-config.js > <unknown> > compilers

```javascript
const path = require("path");

module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  contracts_build_directory: path.join(__dirname, "client/src/contracts"),
  networks: {
    develop: {
      port: 8545
    }
  },
  compilers: {
    solc: {
      version: "^0.6.0"
    }
  }
};

```

Thank You