# UTS BLOCKCHAIN

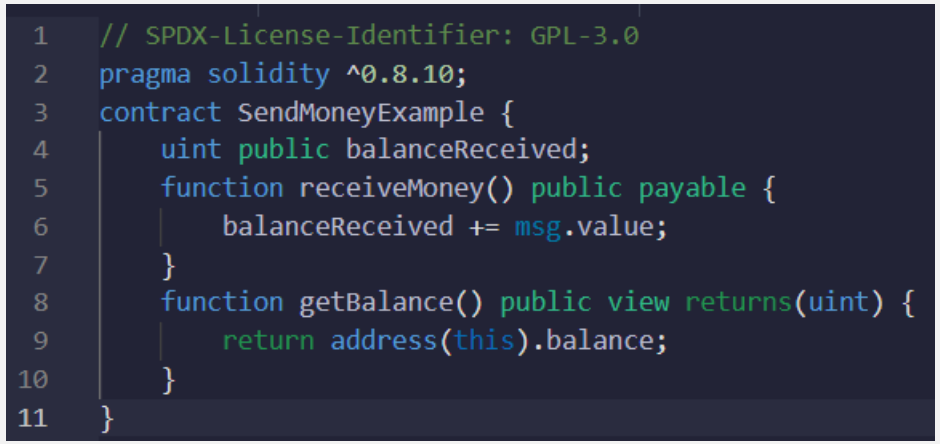
Hikmah Nisya

1103184094

# LAB 1 (DEPOSIT/WITHDRAW ETHER)

Pada lab ini akan mempelajari tentang bagaimana cara membuat smart contract yang akan mengatur keuangan. Kita akan mengirim either pada smart contract anda, lalu smart contract akan mengatur ethernya dan dapat dikirim kepada siapapun. Mengetahui contract address dan sebuah global msg-object bagaimana smart contract mengatur pendanaan. Bagaimana untuk mengirim dan mengambil ether dari dan ke smart contract. Adapun hal yang dibutuhkan dan perlu di siapkan yaitu: browser, koneksi internet, dan waktu.

- Smart contract
  membuat file di remix kemudian memasukan code berikut:



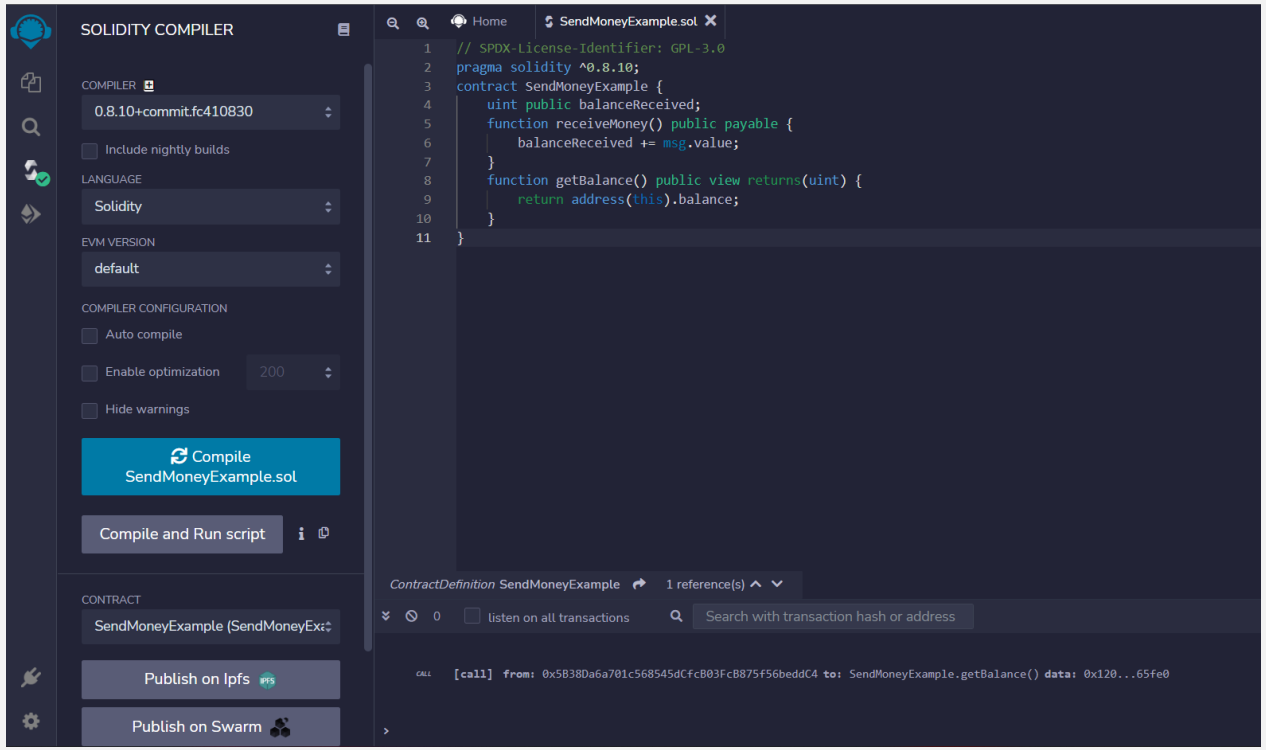
```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.10;
contract SendMoneyExample {
    uint public balanceReceived;
    function receiveMoney() public payable {
        balanceReceived += msg.value;
    }
    function getBalance() public view returns(uint) {
        return address(this).balance;
    }
}
```

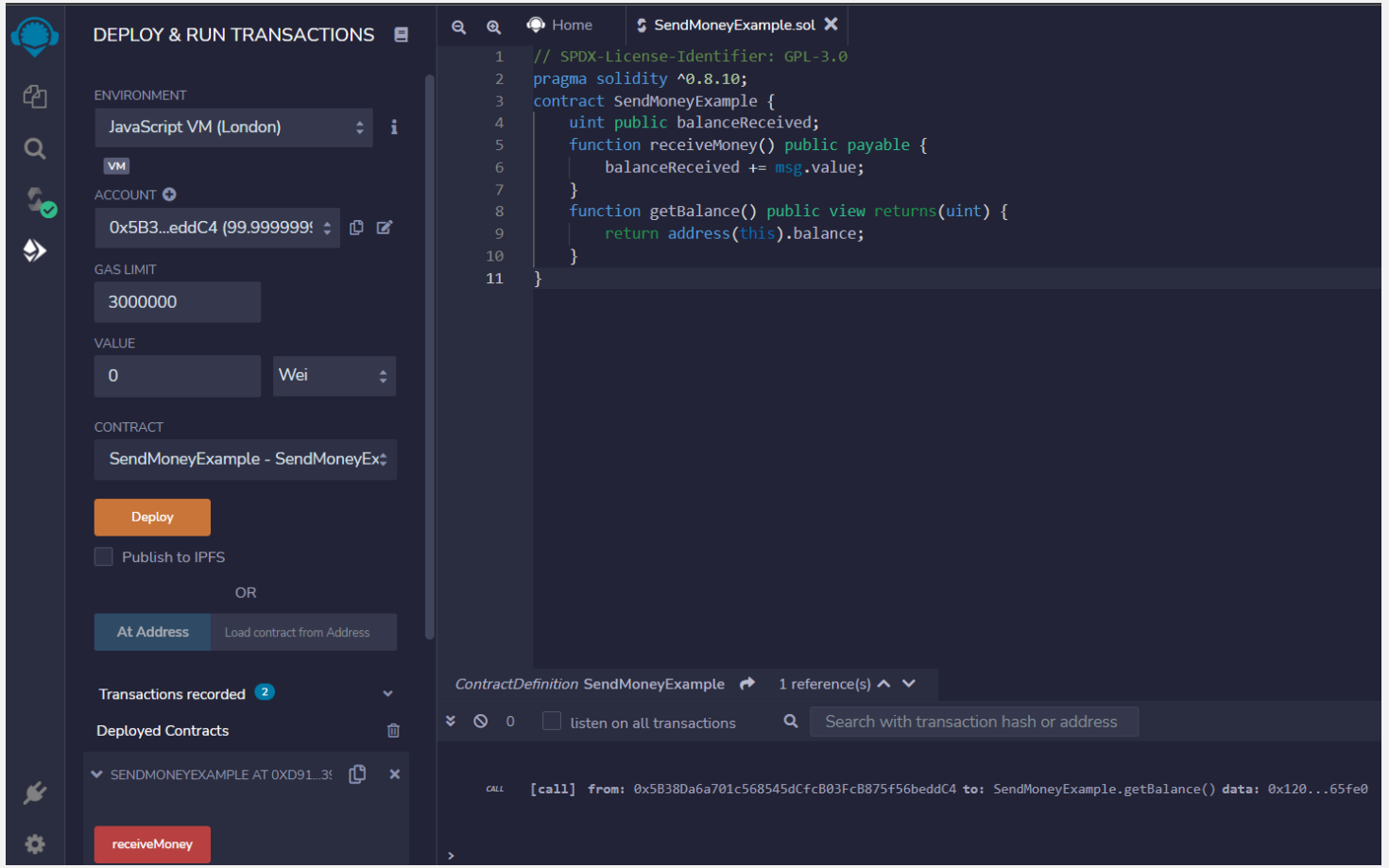
Uint public balance received adalah sebuah variable public storage balance received += msg.value.

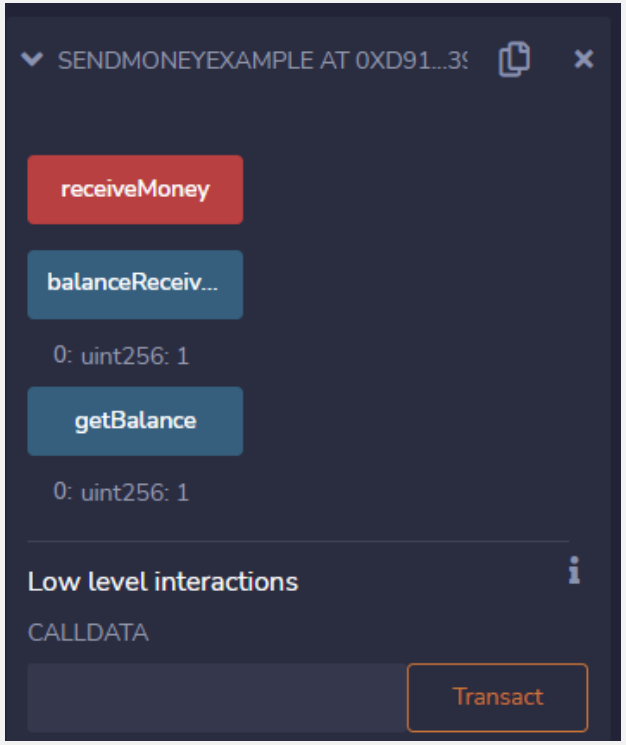
kemudian msg-object adalah object global yang selalu ada dengan beberapa informasi terkait transaksi yang berlangsung.

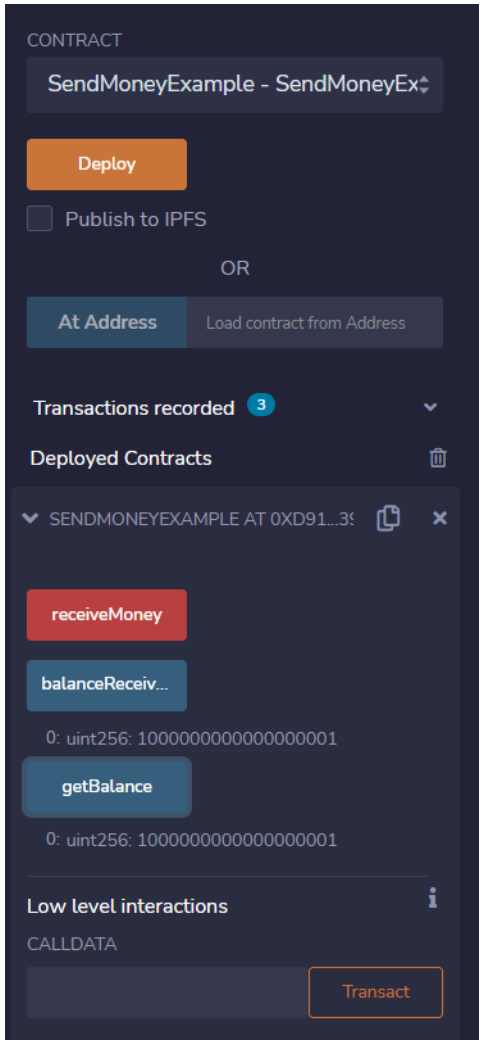
view function adalah sebuah fungsi yang tidak disebut balance yang dimana artinya memberikan anda jumlah ether yang disimpan pada alamat tersebut.

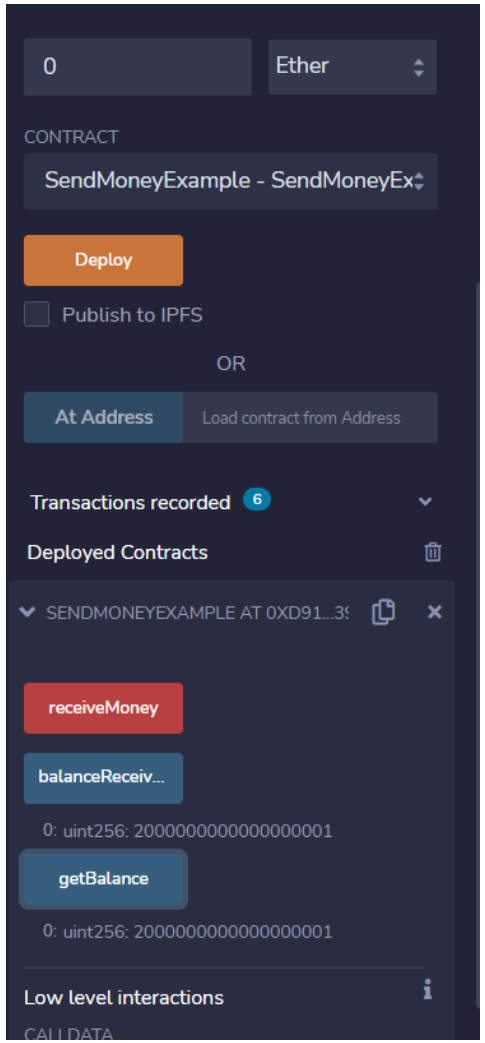
- Mendeploy dan menggunakan smart contract
  pertama-tama kita harus mendeploy smart contract kita. Lalu kita baru dapat melihat jika kita dapat menyimpan ether dan mendapatkan balance ether kita dari smart contract.
- Mendeploy smart contract




DEPLOY & RUN TRANSACTIONS
ENVIRONMENT
JavaScript VM (London)
VM
ACCOUNT
0x5B3...eddC4 (99.999999
GAS LIMIT
3000000
VALUE
0
Wei
CONTRACT
SendMoneyExample - SendMoneyEx
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 2
Deployed Contracts
SENDMONEYEXAMPLE AT 0XD91...3
receiveMoney
Home
SendMoneyExample.sol
1 // SPDX-License-Identifier: GPL-3.0
2 pragma solidity ^0.8.10;
3 contract SendMoneyExample {
4 uint public balanceReceived;
5 function receiveMoney() public payable {
6 balanceReceived += msg.value;
7 }
8 function getBalance() public view returns(uint) {
9 return address(this).balance;
10 }
11 }
ContractDefinition SendMoneyExample
1 reference(s)
0
listen on all transactions
Search with transaction hash or address
CALL [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: SendMoneyExample.getBalance() data: 0x120...65fe0


SENDMONEYEXAMPLE AT 0XD91...3
receiveMoney
balanceReceiv...
0: uint256: 1
getBalance
0: uint256: 1
Low level interactions
CALLDATA
Transact


CONTRACT
SendMoneyExample - SendMoneyEx
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 3
Deployed Contracts
SENDMONEYEXAMPLE AT 0XD91...3
receiveMoney
balanceReceiv...
0: uint256: 1000000000000000001
getBalance
0: uint256: 1000000000000000001
Low level interactions
CALLDATA
Transact


0
Ether
CONTRACT
SendMoneyExample - SendMoneyEx
Deploy
Publish to IPFS
OR
At Address
Load contract from Address
Transactions recorded 6
Deployed Contracts
SENDMONEYEXAMPLE AT 0XD91...3
receiveMoney
balanceReceiv...
0: uint256: 2000000000000000001
getBalance
0: uint256: 2000000000000000001
Low level interactions
CALLDATA

# LAB II (SHARED WALLET)

- Project shared wallet (kegunaan dunia nyata untuk proyek ini)
  tunjangan dana untuk anak per/hari yang dapat digunakan, kemudian pegawah memberi tunjangan ke pegawai lainnya untuk keperluan biaya perjalanan mereka, dan bisnis memberikan kontraktor keperluan dana yang dapat dikeluarkan untuk menggunakan anggaran.
- Pencapaian dalam pembuatan
  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 ke spesifik user berdasarkan alamat user. Menggunakan Kembali surat smart contract yang telah dibuat sebelumnya.

- Mendefinisikan smart contract
  dibawah ini adalah bentuk sederhana smartcontract. Dapat menerima ether dan memungkinkan untuk menarik ether. Tetapi fungsi dari smart contract ini masih belum cukup berguna.

- Permissions: Mengizinkan
  pada Langkah ini kita akan membatasi pengeluaran saldo ke pemilik wallet.

```
//SPDX-License-Indentifier: MIT
pragma solidity ^0.8.10;
contract shareWallet {
    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 {

    }
}
```

- Pada code diatas kita juga dapat menambahkan fungsi ‘onlyOwner’ untuk mengubah ke fungsi ‘withdrawMoney’.

- Menggunakan kontrak dari OpenZeppelin
  mempunyai logika “owner-logic” langsung di dalam smart contract bukanlah hal yang mudah untuk di audit. Maka dari itu coba untuk memecahnya menjadi beberapa bagian kecil dan menggunakan smart contract yang telah di audit dari OpenZeppllin. Pada build OpenZeppelin yang terbaru sudah tidak memiliki fungsi “isOwner” maka dari itu kita menambahkannya sendiri.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;
import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/access/Ownable.sol";
contract shareWallet is Ownable {
    function isOwner() internal vie returns(bool){
        return owner() == msg.sender;
    }
    function withdrawMoney(address payable _to, uint _amount) public onlyOwner {
        _to.transfer(_amount);
    }
    receive() external payable {

    }
}
```

- Permission: menambahkan pengeluaran untuk Roles luar pada Langkah ini kita menambahkan mapping, jadi kita dapat menyimpan address => uint amounts. Ini akan seperti array Ketika disimpan

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;
import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/access/Ownable.sol";
contract shareWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }
    mapping(address => uint) public allowance;
        function addAllowance(address _who, uint _amount) public onlyOwner {
            allowance[_who] = _amount;
        }
    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] => _amount, "You are not allowed!");
        _;
    }
    function withdrawMoney(address payable _to, uint _amount) public ownerOrAllowed (_amount) {
        require(_amount) {
            require(_amount <= address(this).balance, "contract doesn't own enough money");
            _to.transfer(_amount);
        }
        receive()external payable {
        }
    }
}
```

Jika kita jeli smart contract yang kita buat ini masih ada bug, yaitu bug double spending

- Improve/Fix pengeluaran guna menghindari double spending dengan tidak mengurangi dana pada transaksi, seseorang dapat bertransaksi secara terus menerus dengan jumlah yang sama secara terus menerus juga. Pada kode di bawah ini kita mencoba membuat smart contract dengan mengurangi saldo untuk semuanya selain pemilik.

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.10;
import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/access/Ownable.sol";
contract shareWallet is Ownable {
    function isOwner() internal view returns(bool) {
        return owner() == msg.sender;
    }
    mapping(address => uint) public allowance;
        function addAllowance(address _who, uint _amount) public onlyOwner {
            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) {
        allowance[_who] -= _amount;
    }
    function withdrawMoney(address payable _to, uint _amount) internal ownerOrAllowed(_amount){
        require(_amount <= address(this).balance, "Contract doesn't own enough money");
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }
    receive() external payable{
    }
}
```

- Improve structure smart contract
  hingga disini kita sudah mengetahui fungsi basic dari structure smart contract. Untuk dapat mudah dibaca oleh developer lainnya, ada baiknya kita memecah dari fungsi smart contractnya.

```
//SPDX-License-Identifier: MIT
pragma
 solidity 0.8.1;
 import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/access/Ownable.sol";
 contract allowance is Ownable {
     function isOwner() internal view return(bool) {
         return owner() == msg.sender;
     }
     mapping(address => uint) public allowance;
     function setAllowance(address _who, uint _amount) public onlyOwner {
         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) {
         allowance[_who] -= _amount;
     }
 }
 contract shareWallet is allowance {
     function withdrawMoney(address payable _to, uint _amount) public ownnerOrAllowed(_amount) {
         require(_amount <= address(this).balance, "Contract doesn't own enough money");
         if(!isOwner()) {
             reduceAllowance(msg.sender,_amount);
         }
         _to.transfer(_amount);
         receive() external payable {
         }
     }
 }
```

- Menambahkan event di dalam allowance smart contract
  disini kita akan menambahkan contract tetapi biarkan kosong terlebih dahulu

```
//SPDX-License-Identifier: MIT
pragma
 solidity 0.8.1;
 import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/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() = _amount;
    }
    modifier setAllowance(address _wh0, 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;
    }
 }
 contract shareWallet is allowance {
    //...
 }
```

- Menambahkan event di dalam kontrak shared wallet.

```
//SPDX-License-Identifier: MIT
pragma
 solidity 0.8.1;
 import "https://github.com/openzeppelin/openzeppelin-contracts/blob/master/contract/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() = _amount;
    }
    function setAllowance(address _wh0, 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;
    }
 }
 contract shareWallet is allowance {
   event MoneySent(address indexed _from, 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);
```

- Menambahkan Library SafeMath untuk operasi Aritmatika
  berdasarkan dari source code Safemath Library, operasi aritmatika pada solidity dibungkus di dalam overlow. Ini dapat menghasilakan bug, karena programmer biasanya mengasumsikan terjadi eror overlow, dimana merupakan sebuah perilaku standard di dalam high level programming languages. Safemath mengembalikan pemikiran ini dengan mengembalikan transaksi jika operasi terjadi overflow. Pada update solidity terbaru tipe variable integer sudah tidak bisa overflow lagi, ini karna akibat dari versi solidity di atas 0.8

```
pragma solidity ^0.6.1;
import "https://github.com/OpenZeppelin/OpenZeppelin-contracts/blob/master/contracts/access/Ownable.sol";
import "https://github.com/OpenZeppelin/Openzeppelin-contracts/contracts/math/SafeMath.sol";
contract allowance is Ownable {
    using SafeMath for uint;
    event allowanceChanged(address indexed _forWho, address indexed _bywhom, uint _oldAmount, uint _newAmount);
    function isOwner() internal view return(bool) {
        return owner() == msg.sender;
    }
    function setAllowance(address _whp, uint _amount) public onlyOwner {
        //...
    }
    modifier ownerOrAllowed(uint _amount) {
        //...
    }
    function reduceAllowance(address _who, uint _amount) internal
    ownerOrAllowed(_amount){
        emit allowanceChanged([_who], msg.sender, allowance[_who],
        allowance[_who].sub(_amount));
        allowance[_who] = allowance[_who].sub(_amount);
    }
    contract shareWallet is allowance {
        //...
    }
```

- Menghapus dari fungsi “Renounce Ownership”
  Langkah selanjutnya hilangkan fungsi untuk menghapus pemilik. Pada Langkah ini kita menghentikan ini dengan proses pengembalian. Tambahkan fungsi berikut ke Shared Wallet.

```
contract shareWallet is allowance {
    //...
    function renounceOwnershop() public override onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart contract
    }
    //...
}
```

- Memindahkan Smart Contract menjadi file yang terpisah
  pada tahap ini kita akan memisahkan file dan menggunakan fungsi import SharedWallet.sol

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "./Allowance.sol";
contract shareWallet 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 <=(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);
    }
}
```

```
//SPDX-License-Identifier: MIT
pragma solidity ^0.8.1;
import "https://github.com/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(_wh0, msg.sender, allowanceChanged[_who], _amount);
    }
    modifier ownerOrAllowed(uint _amount) {
        require(isOwner() || allowance[msg.sender] >= _amount, "You are not allowed!");
        _;
    }
    funciton reduceAllowance (address _who, uint _amount) internal ownerOrAllowed(_amount) {
        emit allowanceChanged (_who, msg.sender[_who], allowance[_who] - _amount);
        allowance[_who] -= _amount
    }
}
```

# LAB III (SUPPLY CHAIN WALLET)

- Project supply chain, dapat menjadi bagian solusi dari chain, memiliki pengiriman otomatis pada proses pembayaran dan proses pengumpulan payment tanpa orang tengah.
- Development goal, memahami fungsi tingkat rendah address.call.value()(), memahami alur kerja dengan truffle, memahami pengujian unit dengan truffle, dan memahami events dalam HTML

- Smart contract ItemManagemer, kita membutuhkan sebuah smartcontract yang Bernama “management”, maka kita akan menggunakan code di bawah ini:

```
// 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;
    }
```

- Smart contract item, kita membutuhkan sebuah smartcontract Bernama “item”, maka kita akan menggunakan code di bawah ini:

```
// 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 {
    }
}
```

- Setelah itu kita membuat fungsi kepemilikan

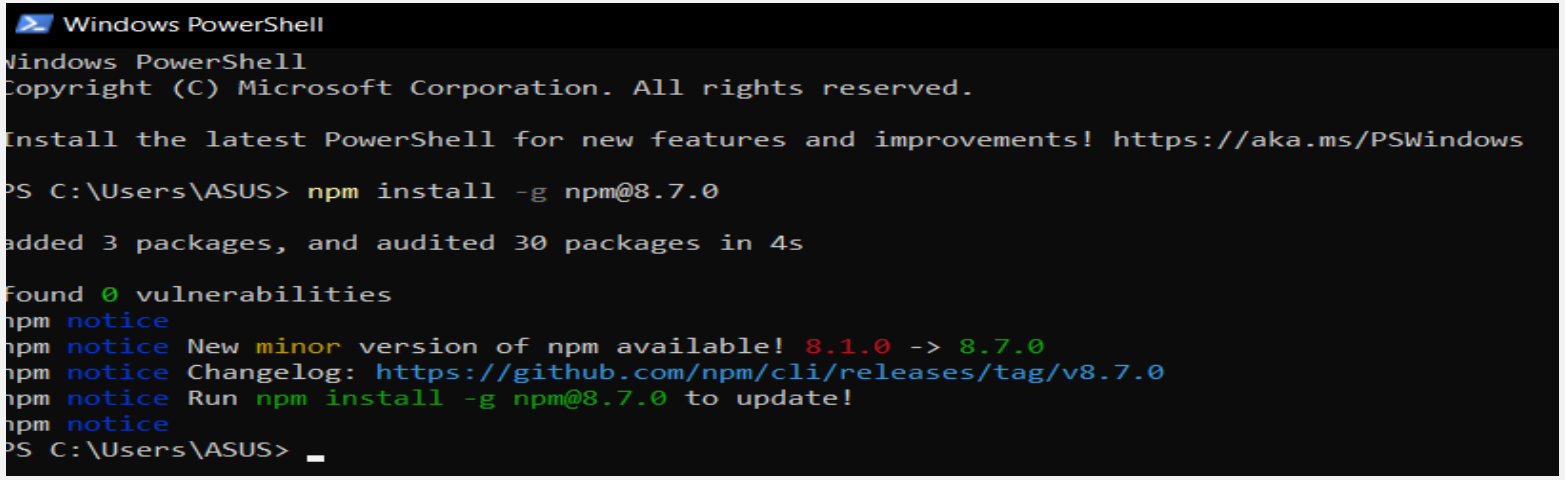
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
    contract Ownable {
    address public _owner;
        constructor () {
        _owner = msg.sender;
    }
    /**
    * @dev Throws if called by any account other than the owner.
    */
    modifier onlyOwner() {
        require(isOwner(), "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);
    }
}
```

- Kemudian melakukan sedikit perubahan/editing pada code "ItemManager" agar proses eksekusi hanya bisa dilakukan oleh owner saja.

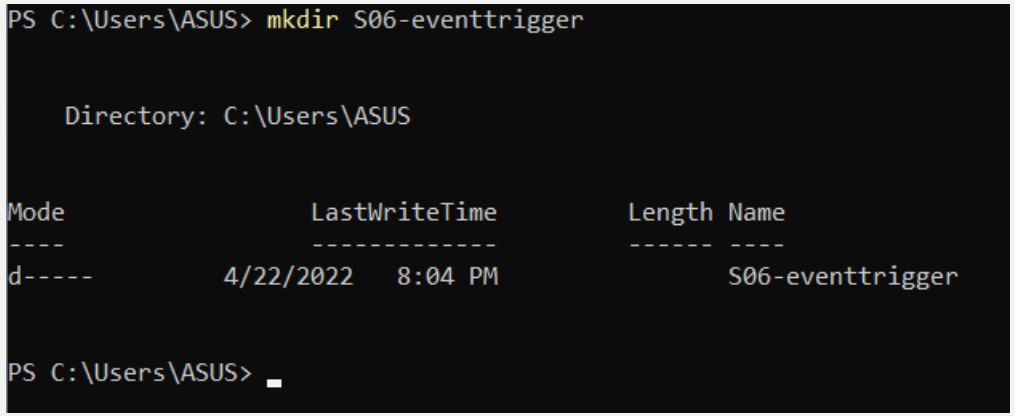
```
// SPDX-License-Identifier: MIT
pragma solidity >=0.6.0 <0.9.0;
import "./Ownable.sol";
import "./Item.sol";
contract ItemManager is Ownable{
    struct S_Item {
        Item _item;
        ItemManager.SupplyChainSteps _step;
        string _identifier;
    }
    mapping(uint => S_Item) public items;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

- Install Truffle

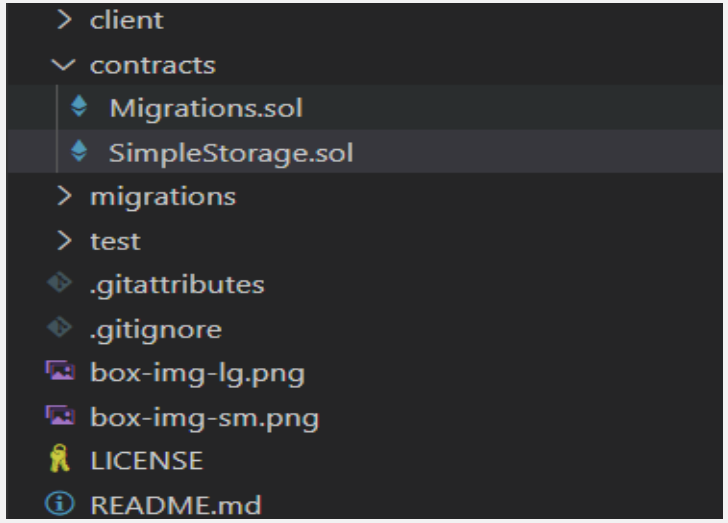
1. gunakan windows powershell lalu masukan perintah “npm install –g npm@8.7.0”



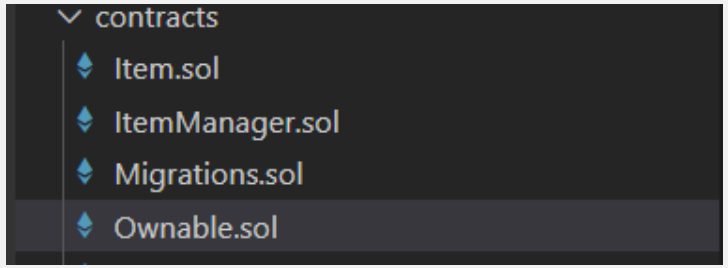

2. Lalu membuat folder dengan penamaan S06-Eventtrigger menggunakan perintah “mkdir S06-eventtrigger”



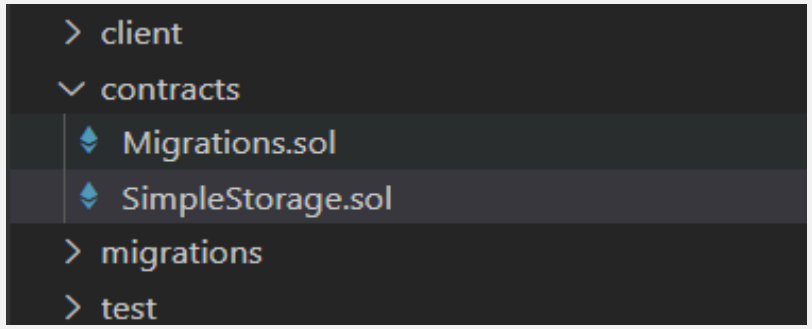
3. Lalu unbox react box nya, disini  kita akan mengguakan visitul studio code.



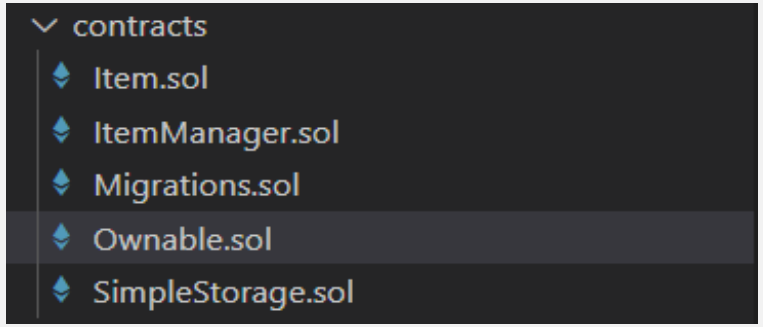

4. Setelah itu masukkan file contract yang sebelumnya telah kita buat di remix.org.



6. Selanjutnya buka text editor pada vsc lalu arahkan ke folder yang telah dibuat sebelumnya, lalu lakukan penghapusan pada file “SimpleStorage.sol”.




7. Setelah menghapus file di atas seperti petunjuk, maka selanjutnya kita memasukan lagi folder yang telah kit buat di remix.org.



7. Setelah menghapus file di atas seperti petunjuk, maka selanjutnya kita memasukan lagi folder yang telah kit buat di remix.org.

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

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

```
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"
    }
  }
};
```