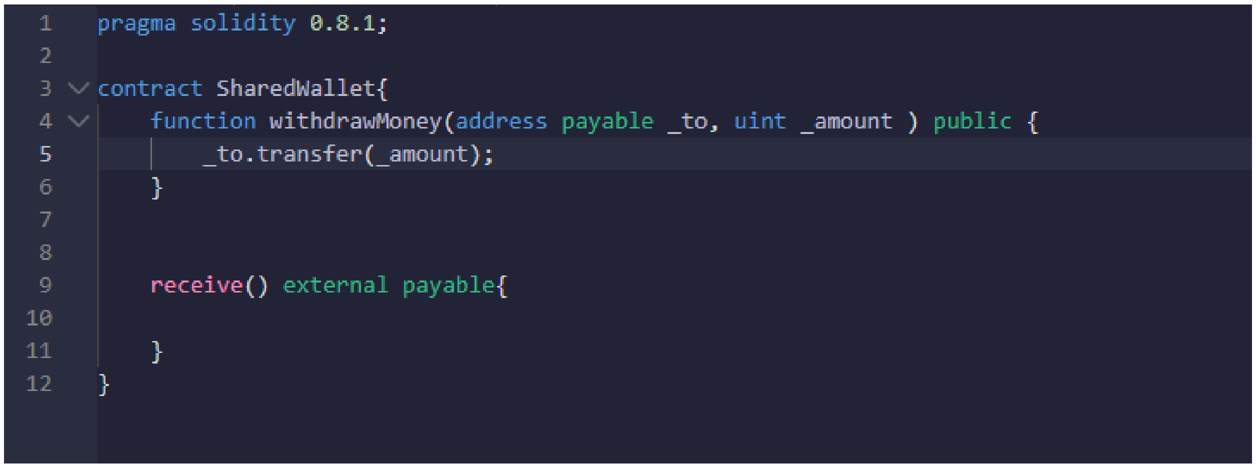
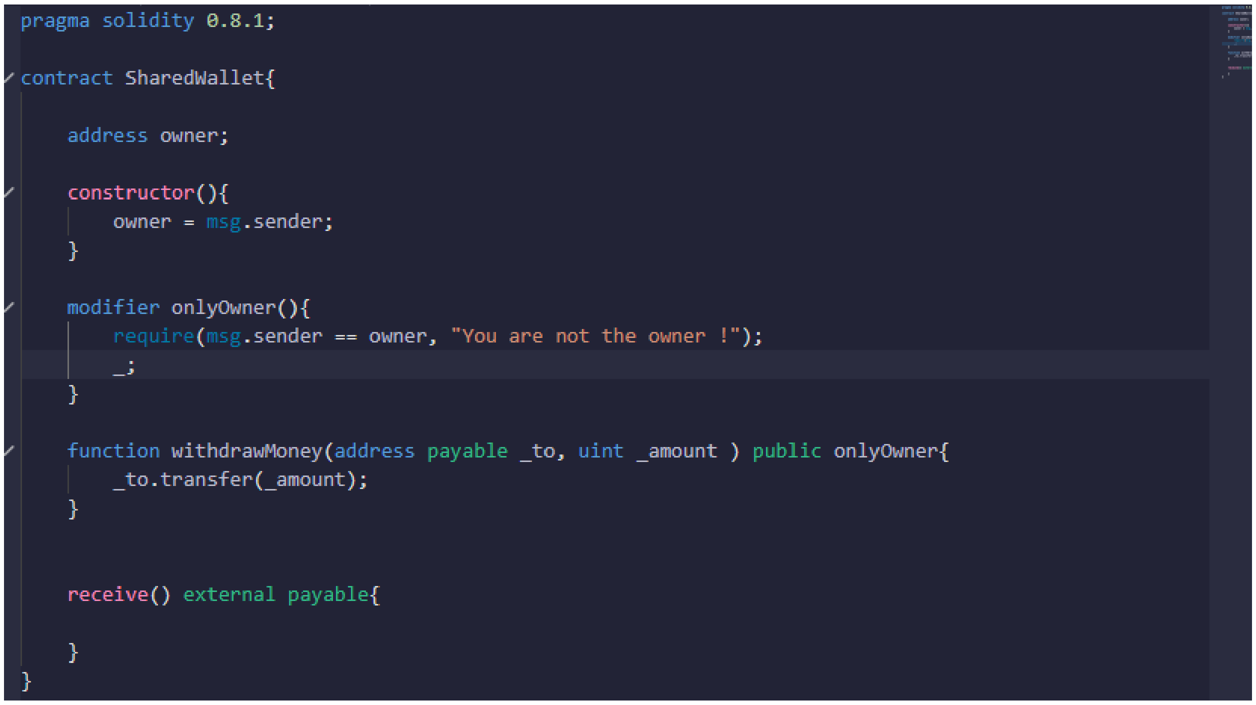
Technical Documentation for Lab 2 Reproduce – Shared Wallet

1. First we will define the basic smart contract for shared wallet, with capabilities to withdraw and receive Ether :



```
pragma solidity 0.8.1;

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


    receive() external payable{

    }
}
```

2. Add "onlyOwner" modifier to adjust permission so that only owner is allowed to withdraw Ether :



```
pragma solidity 0.8.1;

contract SharedWallet{

    address owner;

    constructor(){
        owner = msg.sender;
    }

    modifier onlyOwner(){
        require(msg.sender == owner, "You are not the owner !");
        _;
    }

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


    receive() external payable{

    }
}
```

3. Owner-logic directly in one smart contract isn't very easy to audit. Therefore, it was needed to break it down into smaller parts and re-use existing audited smart contracts from **OpenZeppelin** for that.

   Zeppelin import source :

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

   Codes :

```
pragma solidity 0.8.1;
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

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

    address owner;

    constructor(){
        owner = msg.sender;
    }

    modifier onlyOwner(){
        require(msg.sender == owner, "You are not the owner !");
        _;
    }


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


    receive() external payable{

    }
}
```

4. In this step we add mapping so we can store address in form of uint amounts. This will be like an array that stores [0x123546...] an address, to a specific number.

```
pragma solidity 0.8.1;
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";

contract SharedWallet is Ownable{

    address owner;

    constructor(){
        owner = msg.sender;
    }


    function isOwner() internal view returns(bool) {
                return Ownable.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) {
        _to.transfer(_amount);
    }

    receive() external payable{

    }
}
```

5. Improve/Fix Allowance to avoid Double-Spending

```
pragma solidity 0.8.1;
```

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

contract SharedWallet is Ownable{

    address owner;

    constructor(){
        owner = msg.sender;
    }


    function isOwner() internal view returns(bool) {
        return Ownable.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) public
ownerOrAllowed(_amount) {
        if(!isOwner()) {
            reduceAllowance(msg.sender, _amount);
        }
        _to.transfer(_amount);
    }


    receive() external payable{

    }
}
```

6. After all the step above are completed, we can structure the smart contract differently. To make it easier to read, we can break the functionality down into two distinct smart contracts.

Note : since Allowance is Ownable, and the SharedWallet is Allowance, therefore by commutative property, SharedWallet is also Ownable.

```
//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 {

    function isOwner() internal view returns(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 SharedWallet is Allowance {
    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);
    }

    receive() external payable {
    }
}
```

7. Add event and emit on set and reduce allowance :

```
//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);

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

    mapping(address => uint) public allowance;

    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;
    }
}
```

```
contract SharedWallet is Allowance {
    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);
    }

    receive() external payable {
    }
}
```

8. Add money send and receive on SharedWallet smart contract :

```
//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);

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

    mapping(address => uint) public allowance;

    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;
    }
}

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);
    }
}
```

9. Next we will import safemath to use on reduce allowance functionality :

Import library :

import "https://github.com/OpenZeppelin/openzeppelin-contracts/contracts/math/SafeMath.sol";

```
//SPDX-License-Identifier: MIT
pragma solidity 0.8.1;
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 returns(bool) {
        return owner() == msg.sender;
    }

    mapping(address => uint) public allowance;

    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].sub(_amount));
        allowance[_who] = allowance[_who].sub(_amount);
    }
}
```

10. Lastly , on the SharedWallet smart contract , we remove user remove functionality by doing a revert :

```
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);
    }

    function renounceOwnership() public override onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart
contract
    }


}
```

11. And then separate the contract into two files :

Allowance Smart Contract :

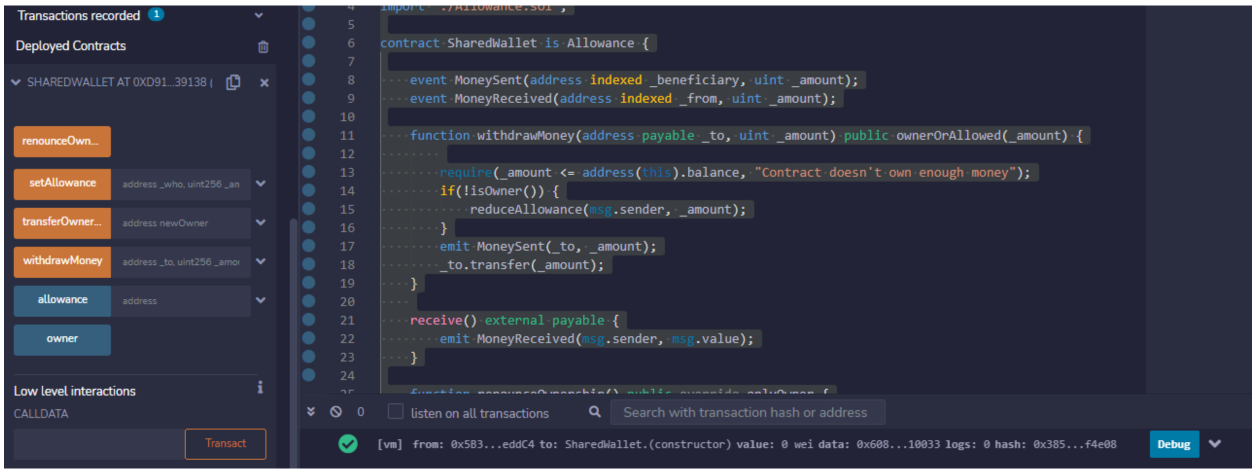
```
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/blob/master/contracts/access/Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-
contracts/contracts/math/SafeMath.sol";

contract Allowance is Ownable {
    using SafeMath for uint;
   event AllowanceChanged(address indexed _forWho, address indexed _byWhom, uint
_oldAmount, uint _newAmount);

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

    mapping(address => uint) public allowance;

    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].sub(_amount));
        allowance[_who] = allowance[_who].sub(_amount);
    }
}
```

Shared Wallet Shared Contract :

We import the allowance contract before defining the shared wallet contract

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

    function renounceOwnership() public override onlyOwner {
        revert("can't renounceOwnership here"); //not possible with this smart
contract
    }


}
```

12. Smart contract successfully deployed :



```
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);
    }

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

    function renounceOwnership() public override onlyOwner {
```

```
[vm] from: 0x5B3...eddC4 to: SharedWallet.(constructor) value: 0 wei data: 0x608...10033 logs: 0 hash: 0x385...f4e08
```