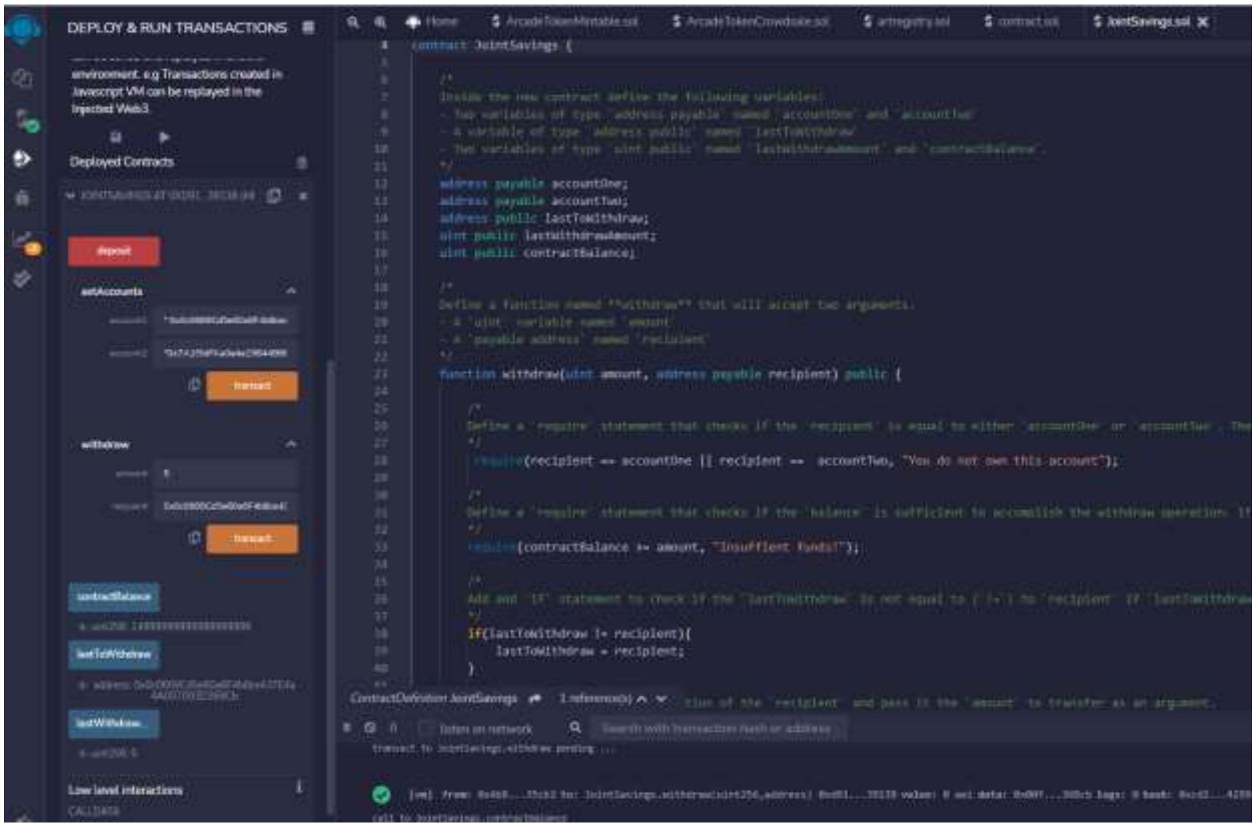
Withdrawal 5 into account 1

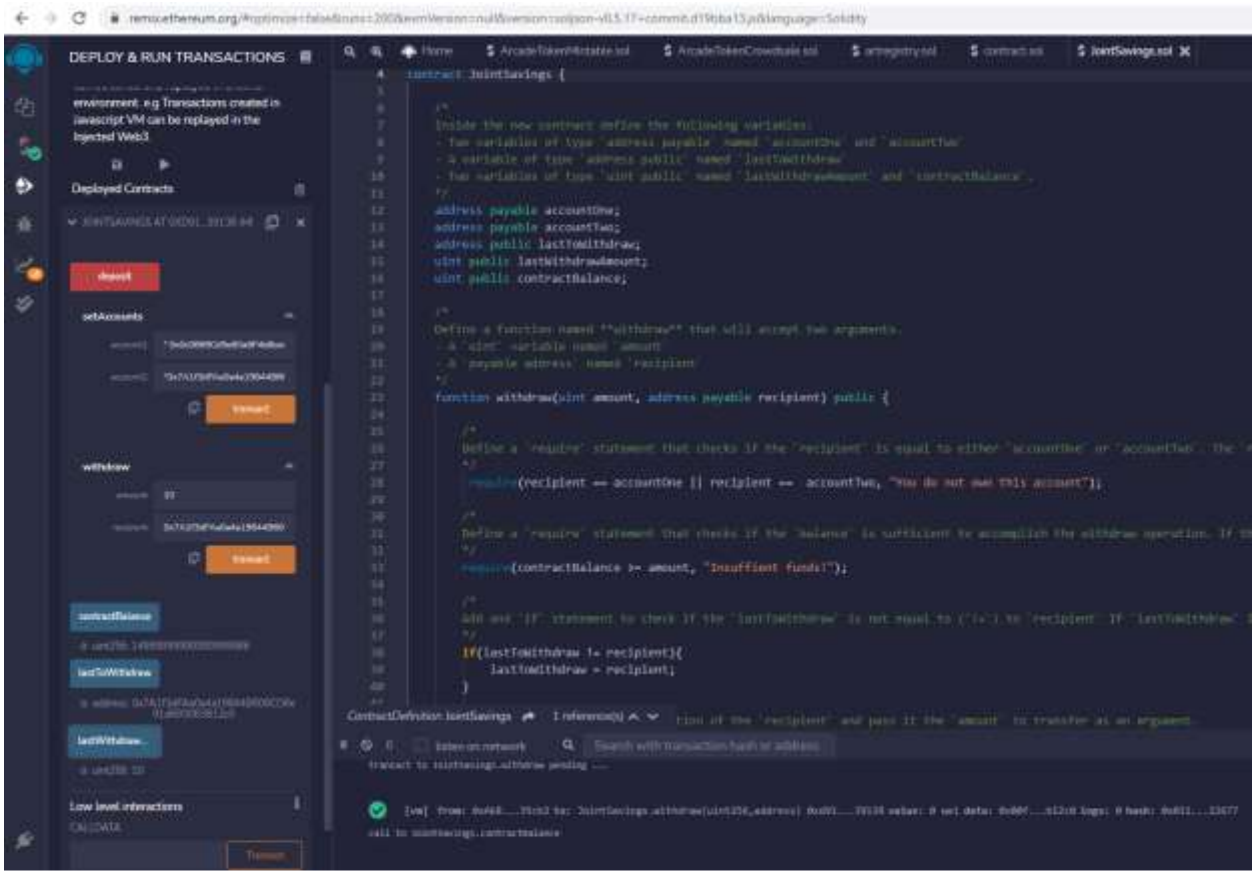


environment. e.g Transactions created in Javascript VM can be replayed in the Injected Web3.

```
contract JointSavings {

    /*
    Inside the new contract define the following variables:
    - Two variables of type `address payable` named `accountOne` and `accountTwo`
    - A variable of type `address public` named `lastToWithdraw`
    - Two variables of type `uint public` named `lastWithdrawAmount` and `contractBalance`.
    */
    address payable accountOne;
    address payable accountTwo;
    address public lastToWithdraw;
    uint public lastWithdrawAmount;
    uint public contractBalance;

    /*
    Define a function named **withdraw** that will accept two arguments.
    - A `uint` variable named `amount`
    - A `payable address` named `recipient`
    */
    function withdraw(uint amount, address payable recipient) public {

        /*
        Define a `require` statement that checks if the `recipient` is equal to either `accountOne` or `accountTwo`. The
        */
        require(recipient == accountOne || recipient ==  accountTwo, "You do not own this account");

        /*
        Define a `require` statement that checks if the `balance` is sufficient to accomplish the withdraw operation. If
        */
        require(contractBalance >= amount, "Insuffient funds!");

        /*
        Add and `if` statement to check if the `lastToWithdraw` is not equal to (`!=`) to `recipient` If `lastToWithdraw
        */
        if(lastToWithdraw != recipient){
            lastToWithdraw = recipient;
        }
```

```
transact to JointSavings.withdraw pending ...

[vm] from: 0x4b0...35cb2 to: JointSavings.withdraw(uint256,address) 0xd91...39138 value: 0 wei data: 0x00f...300cb logs: 0 hash: 0xcd2...4299
call to JointSavings.contractBalance
```

Withdraw 10 into account 2



DEPLOY & RUN TRANSACTIONS

environment. e.g Transactions created in Javascript VM can be replayed in the Injected Web3

Deployed Contracts

JOINTSAVINGS AT 0XD91...39138 (M)

```
contract JointSavings {

    /*
    Inside the new contract define the following variables:
    - Two variables of type `address payable` named `accountOne` and `accountTwo`
    - A variable of type `address public` named `lastToWithdraw`
    - Two variables of type `uint public` named `lastWithdrawAmount` and `contractBalance`.
    */
    address payable accountOne;
    address payable accountTwo;
    address public lastToWithdraw;
    uint public lastWithdrawAmount;
    uint public contractBalance;

    /*
    Define a function named **withdraw** that will accept two arguments.
    - A `uint` variable named `amount`
    - A `payable address` named `recipient`
    */
    function withdraw(uint amount, address payable recipient) public {

        /*
        Define a `require` statement that checks if the `recipient` is equal to either `accountOne` or `accountTwo`. The `
        */
        require(recipient == accountOne || recipient ==  accountTwo, "You do not own this account");

        /*
        Define a `require` statement that checks if the `balance` is sufficient to accomplish the withdraw operation. If th
        */
        require(contractBalance >= amount, "Insufficient funds!");

        /*
        Add and `if` statement to check if the `lastToWithdraw` is not equal to (`!=`) to `recipient` If `lastToWithdraw` 
        */
        if(lastToWithdraw != recipient){
            lastToWithdraw = recipient;
        }
```

```
transact to JointSavings.withdraw pending ...

[vm] from: 0x4B2...35cb2 to: JointSavings.withdraw(uint256,address) 0xd91...39138 value: 0 wei data: 0x00f...812c0 logs: 0 hash: 0x811...22677
call to JointSavings.contractBalance
```