CSE 526: BLOCKCHAIN TERM PROJECT DIRECTFUND-DAPP

Name : Mohammed Mahaboob Khan
Person # : 50318613
Email : mkhan32@buffalo.edu

Phase 1

Project Description

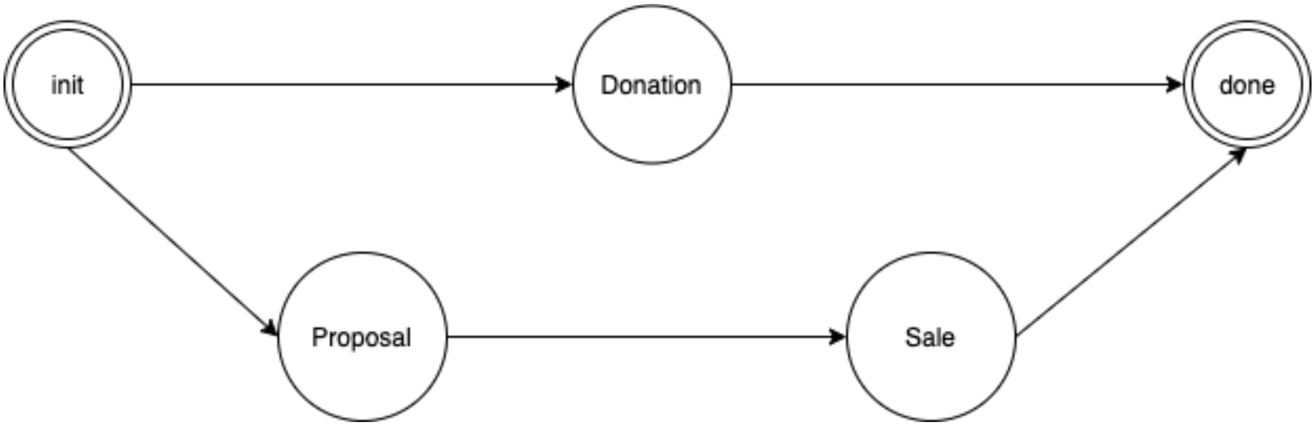
- **Major Area**: Charity/ Donations/ fund raising
- **Title**: Dapp for fund raising
- **Dapp Name**: DirectFund-Dapp
- **Clients**: charitable fund/ donation recipient, donors, buyers
- **Abstract**:
  - We often come across fund raisers on Facebook and other social media platforms where people donate money to raise funds for a particular person or an organization or for a certain cause. There are many entities involved in this donation process other than just the donor and the recipient of the donation. Because of this we almost always worry if our donations are reaching the recipient intact without intervention from any middlemen.
  - This issue can be easily addressed by making the donation process decentralized. DirectFund-Dapp will allow anyone across the world to make donations directly to the person or the fund without worrying about anyone interfering.
  - Another feature that I wish to incorporate in my Dapp is, a feature to allow a person to sell anything to whoever bids highest for it in order to raise funds. Even in this case, the funds will go directly from the buyer to the recipient of the donation/charity and not to the seller instead.

Phase 2

Design Diagrams

In this phase, I develop the design diagrams, namely FSM diagram, contract diagram and the use case diagram for the design and development of my Dapp.

I. Finite State Machine Diagram

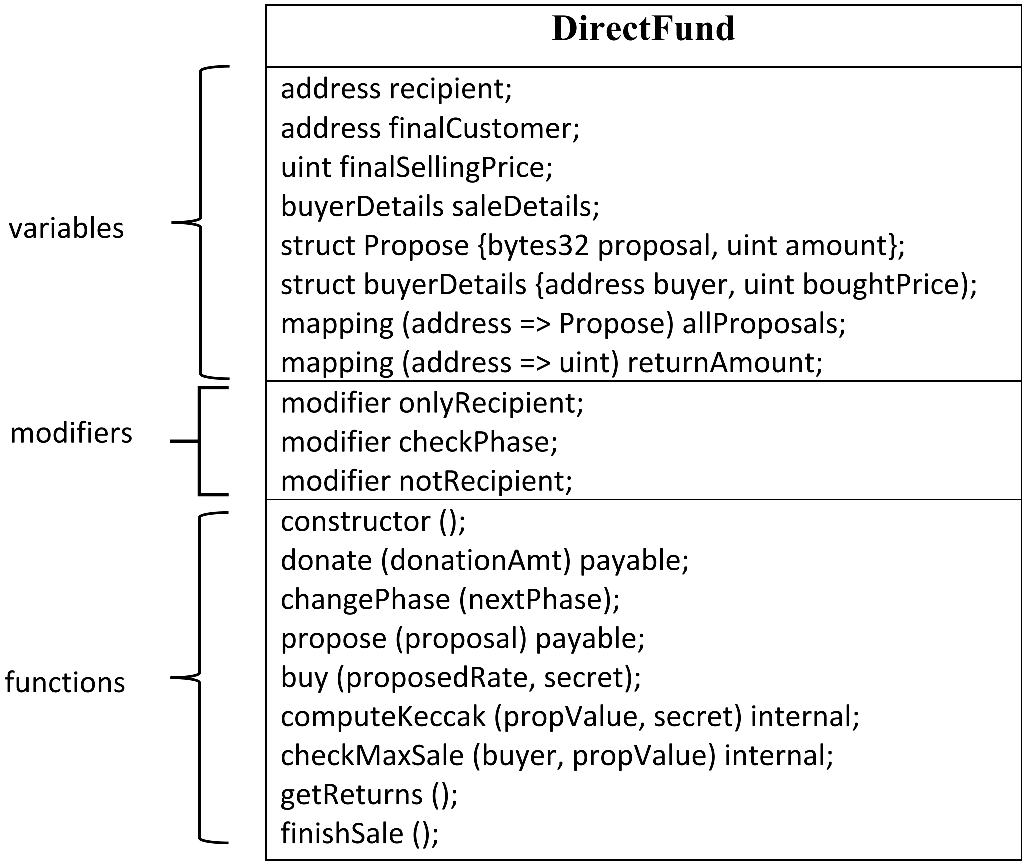


As shown in the FSM above, the main states are:

1. **Init:** Initialization phase
2. **Donation:** The phase/state in which a donor sends (donates) money.
3. **Proposal:** The state where buyers propose their rates.
4. **Sale:** The state where the item is sold to the buyer with the highest rate.
5. **Done:** End phase

The transition between the states is managed by the recipient who is the contract owner in this case.

II. Contract Diagram



DirectFund

```
address recipient;
address finalCustomer;
uint finalSellingPrice;
buyerDetails saleDetails;
struct Propose {bytes32 proposal, uint amount};
struct buyerDetails {address buyer, uint boughtPrice);
mapping (address => Propose) allProposals;
mapping (address => uint) returnAmount;
```

```
modifier onlyRecipient;
modifier checkPhase;
modifier notRecipient;
```

```
constructor ();
donate (donationAmt) payable;
changePhase (nextPhase);
propose (proposal) payable;
buy (proposedRate, secret);
computeKeccak (propValue, secret) internal;
checkMaxSale (buyer, propValue) internal;
getReturns ();
finishSale ();
```

Variables:

- Recipient – contract owner
- buyerDetails – struct that has final buyer details
- Propose – struct for proposal details
- allProposals – mapping of all addresses to their proposal details.
- returnAmount – mapping of addresses to the amount that must be returned.

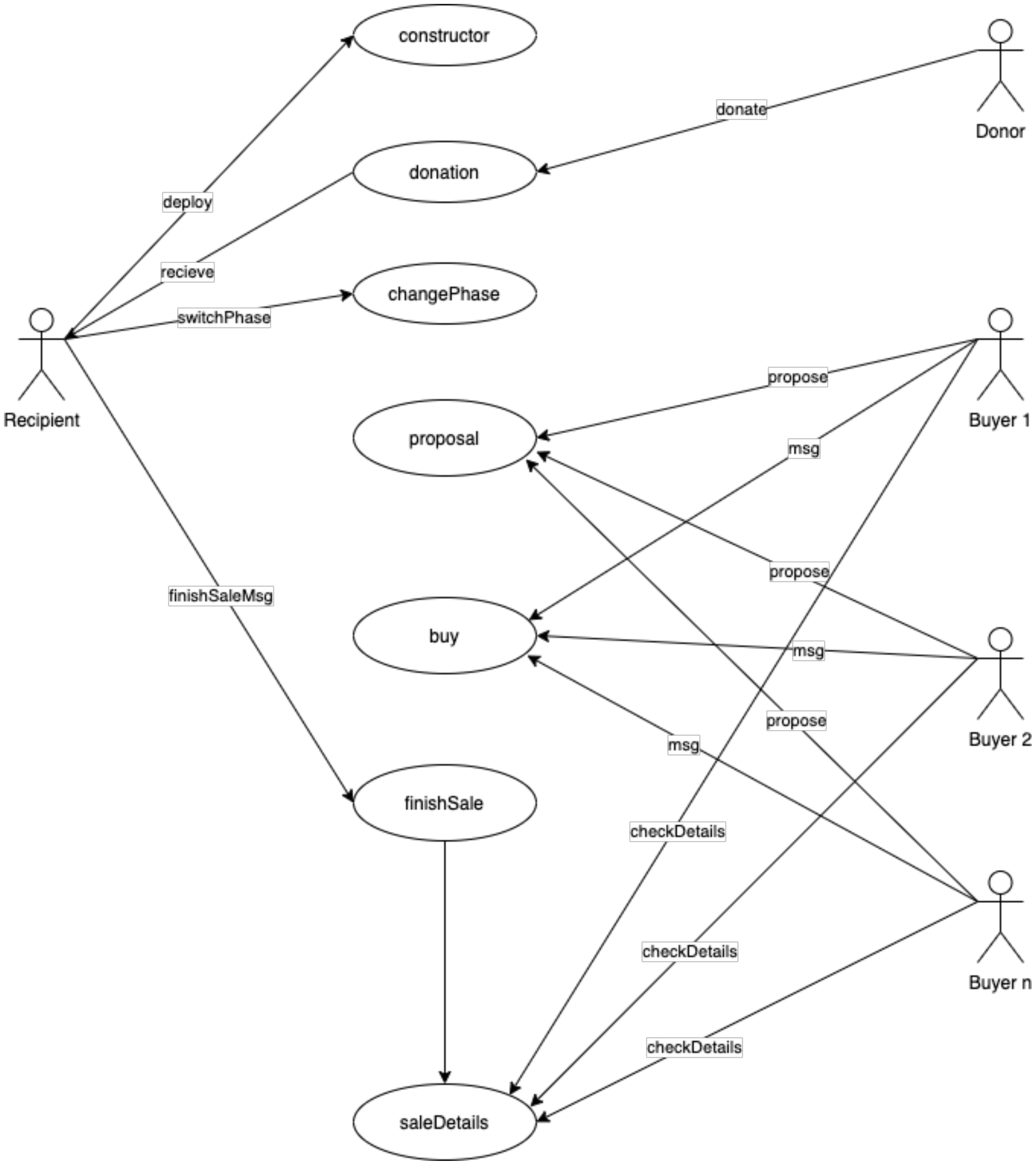
Modifiers:

- onlyRecipient – check that the user is the recipient/ contract owner
- checkPhase – check whether the current phase is valid
- notRecipient – check that the user is not the recipient

Functions:

- donate() – donate money directly to the recipient.
- changePhase() – used by recipient to change the phases shown in the FSM.
- propose() – buyers propose their rates for the item being sold.
- buy() – buyers attempt to buy the item at the proposed rates.
- getReturns() – to make sure everyone receives their balance money at the end of the sale.
- finishSale() – changes the phase to mark the end of the sale.

III. Use Case Diagram

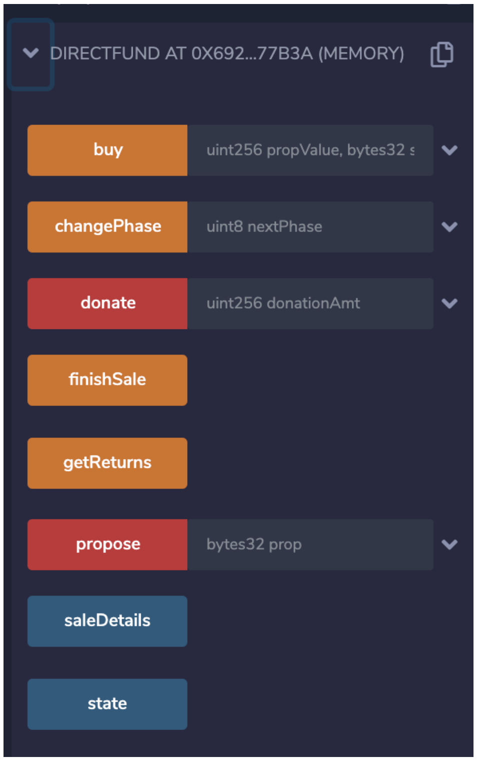


Phase 3

Smart Contract Development

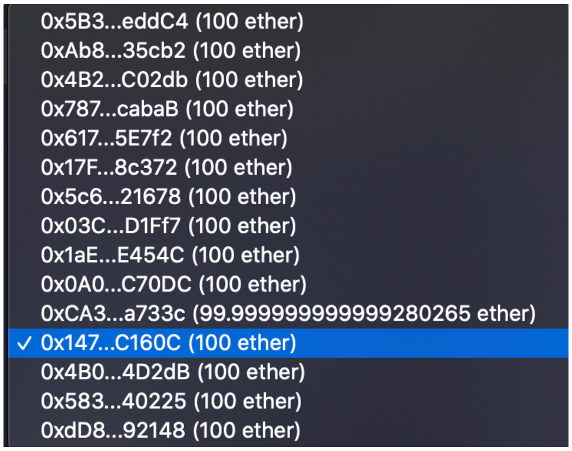
In this phase I successfully develop and deploy the smart contract part of my Dapp. The smart contract **DirectFund.sol** performs two main functions:

1. **Donation**: Any person can send/donate any amount of ether to the recipient.
2. **Sale**: Buyers can propose their rates for purchasing the item being sold and buyer with the highest proposed rate gets the item. These funds are directly transferred to the recipient.

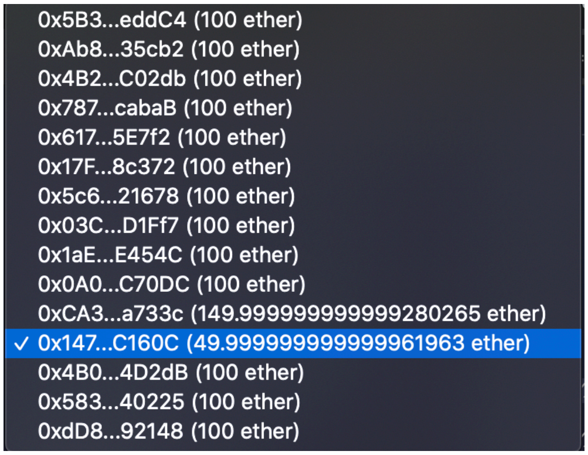


The above picture is a screenshot of Remix IDE after deploying the smart contract.

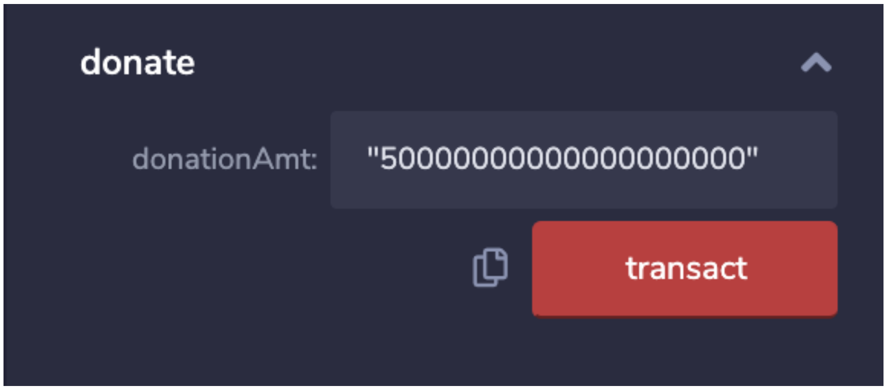
Donation:



```
0x5B3...eddC4 (100 ether)
0xAb8...35cb2 (100 ether)
0x4B2...C02db (100 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0xCA3...a733c (99.999999999999280265 ether)
✓ 0x147...C160C (100 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (100 ether)
0xdD8...92148 (100 ether)
```



```
0x5B3...eddC4 (100 ether)
0xAb8...35cb2 (100 ether)
0x4B2...C02db (100 ether)
0x787...cabaB (100 ether)
0x617...5E7f2 (100 ether)
0x17F...8c372 (100 ether)
0x5c6...21678 (100 ether)
0x03C...D1Ff7 (100 ether)
0x1aE...E454C (100 ether)
0x0A0...C70DC (100 ether)
0xCA3...a733c (149.999999999999280265 ether)
✓ 0x147...C160C (49.999999999999961963 ether)
0x4B0...4D2dB (100 ether)
0x583...40225 (100 ether)
0xdD8...92148 (100 ether)
```



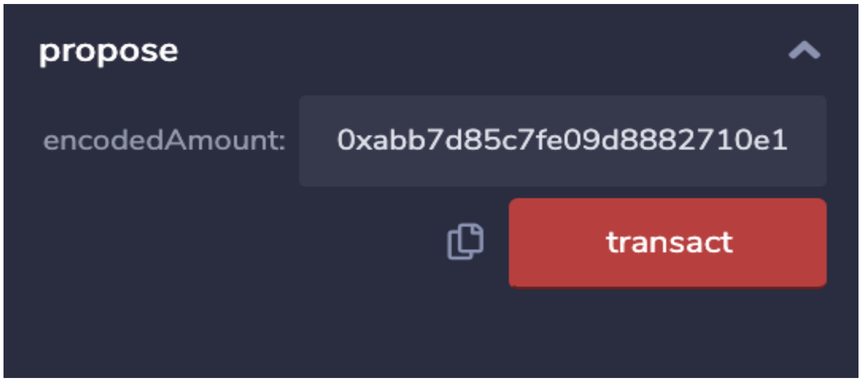
donate

donationAmt: "50000000000000000000"

Account number ‘0x147...’ donates 50 ether successfully.

Buying and Selling:

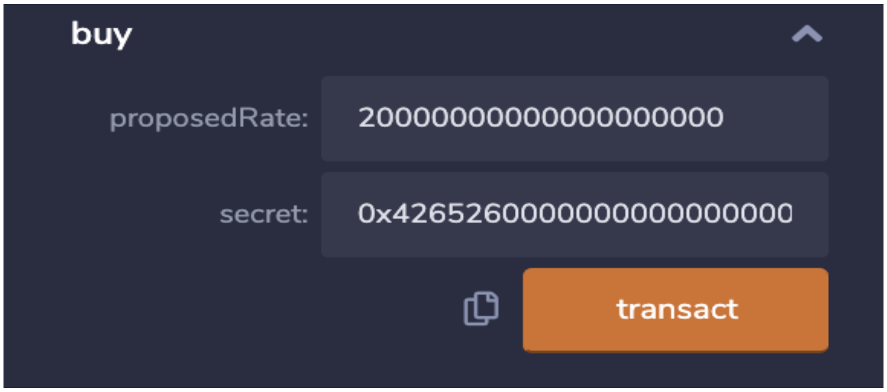
1. In the **‘Propose’** phase, the buyers will first propose their rates by sending the keccak encoded value of the rate to the ***propose()*** function.



propose

encodedAmount: 0xabb7d85c7fe09d8882710e1

2. Once all buyers have proposed their rates, the recipient will change the phase to **‘Sale’.** In this phase, the buyers will validate their identity by sending their original proposed rates along with the keccak secret as the password. The buyer with the highest rate is chosen at the end of this phase.

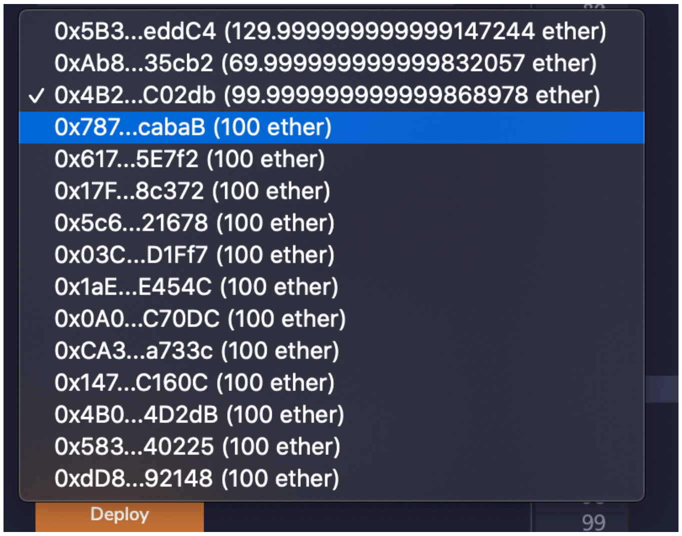


buy

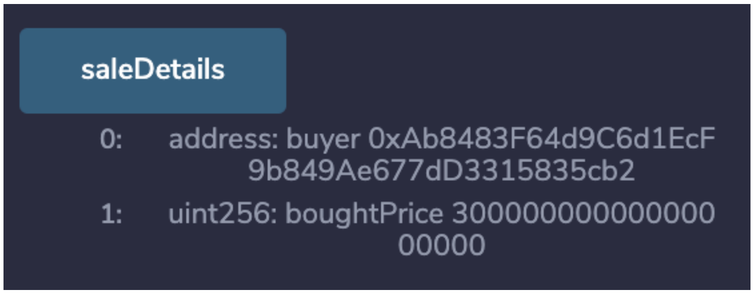
proposedRate: 20000000000000000000

secret: 0x4265260000000000000000

3. In the final phase **‘Done’**, the buyers can make use of ***getReturns*()** to get any remaining balance after the sale. After this phase, the buyers can click on **saleDetails** which is a public struct, to view the address of the final buyer and the rate at which the sale was made.



```
  0x5B3...eddC4 (129.999999999999147244 ether)
  0xAb8...35cb2 (69.999999999999832057 ether)
✓ 0x4B2...C02db (99.999999999999868978 ether)
  0x787...cabaB (100 ether)
  0x617...5E7f2 (100 ether)
  0x17F...8c372 (100 ether)
  0x5c6...21678 (100 ether)
  0x03C...D1Ff7 (100 ether)
  0x1aE...E454C (100 ether)
  0x0A0...C70DC (100 ether)
  0xCA3...a733c (100 ether)
  0x147...C160C (100 ether)
  0x4B0...4D2dB (100 ether)
  0x583...40225 (100 ether)
  0xdD8...92148 (100 ether)
```



saleDetails

```
0:   address: buyer 0xAb8483F64d9C6d1EcF
              9b849Ae677dD3315835cb2
1:   uint256: boughtPrice 300000000000000
              00000
```

We can see that buyer with address '0xAb8…' has purchased the item successfully for 30 ether.

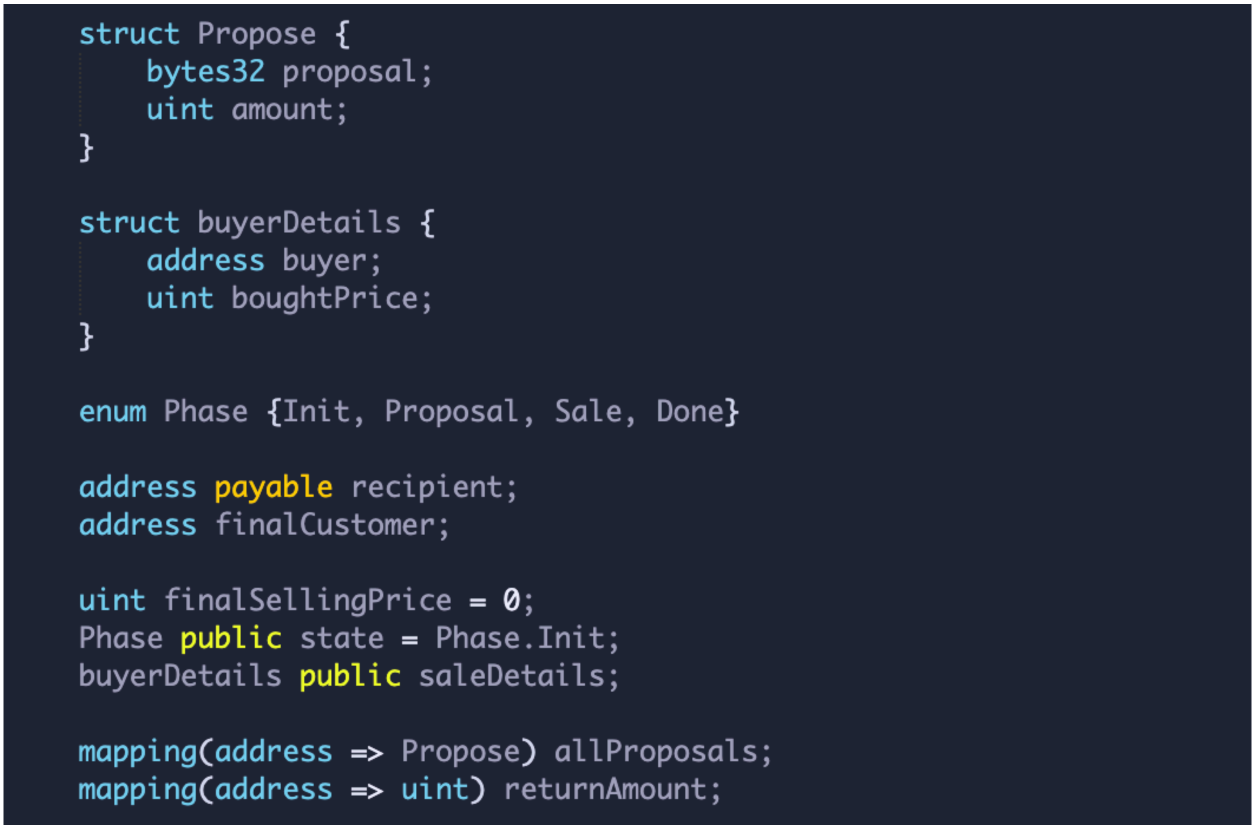
Conclusion of Phase3:

- Successfully implemented donation part of the Dapp
- Implemented selling and buying part of the Dapp and also added security using keccak encryption/ decryption.

Note: *Slight modifications in Phase2 contract diagram. Added new modifier, struct and functions.*

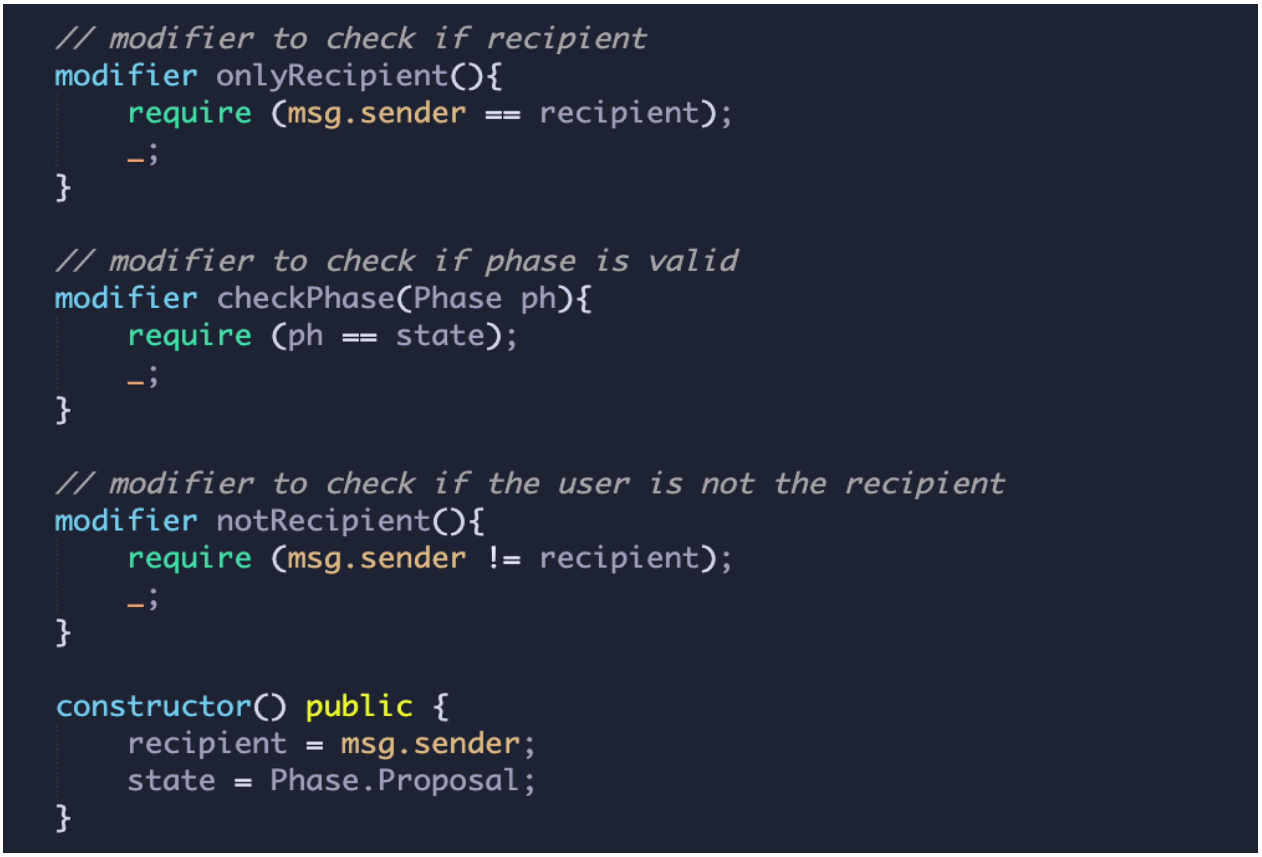
Screenshots of code snippets for reference:

- Structs, variables and mappings:



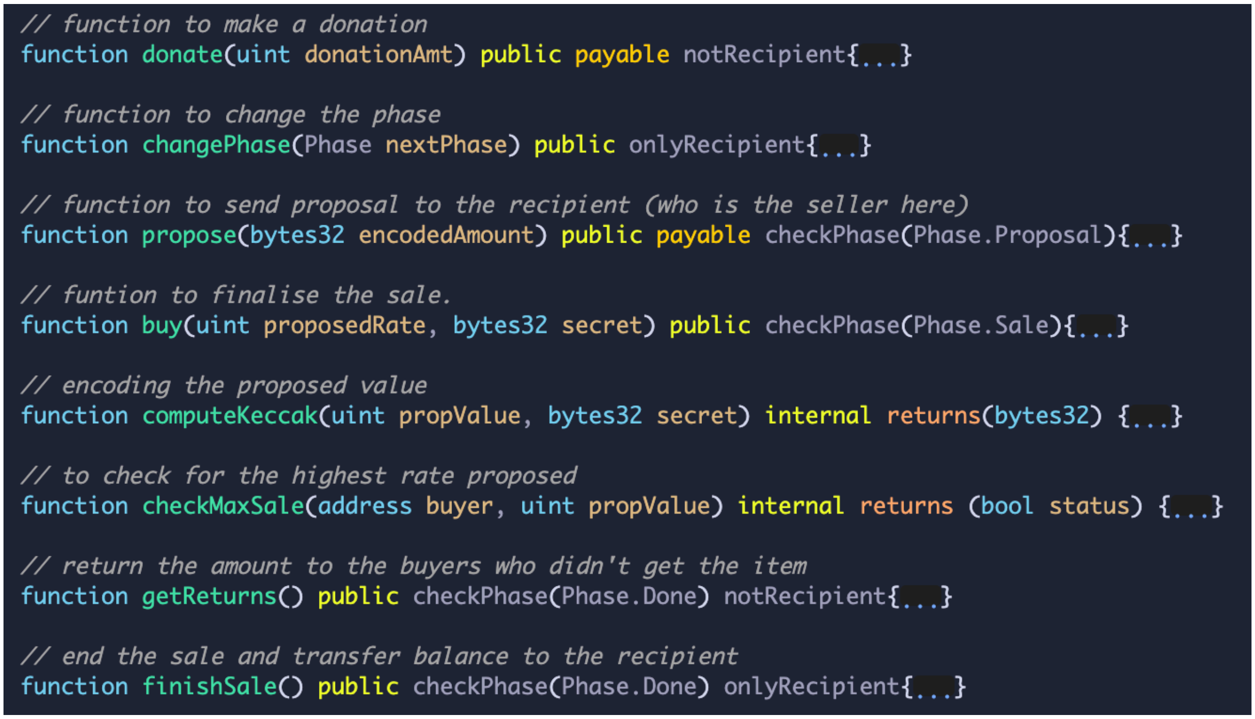
```
struct Propose {
    bytes32 proposal;
    uint amount;
}

struct buyerDetails {
    address buyer;
    uint boughtPrice;
}

enum Phase {Init, Proposal, Sale, Done}

address payable recipient;
address finalCustomer;

uint finalSellingPrice = 0;
Phase public state = Phase.Init;
buyerDetails public saleDetails;

mapping(address => Propose) allProposals;
mapping(address => uint) returnAmount;
```

- Modifiers and constructor:



```
// modifier to check if recipient
modifier onlyRecipient(){
    require (msg.sender == recipient);
    _;
}

// modifier to check if phase is valid
modifier checkPhase(Phase ph){
    require (ph == state);
    _;
}

// modifier to check if the user is not the recipient
modifier notRecipient(){
    require (msg.sender != recipient);
    _;
}

constructor() public {
    recipient = msg.sender;
    state = Phase.Proposal;
}
```

- Functions:



```
// function to make a donation
function donate(uint donationAmt) public payable notRecipient{...}

// function to change the phase
function changePhase(Phase nextPhase) public onlyRecipient{...}

// function to send proposal to the recipient (who is the seller here)
function propose(bytes32 encodedAmount) public payable checkPhase(Phase.Proposal){...}

// funtion to finalise the sale.
function buy(uint proposedRate, bytes32 secret) public checkPhase(Phase.Sale){...}

// encoding the proposed value
function computeKeccak(uint propValue, bytes32 secret) internal returns(bytes32) {...}

// to check for the highest rate proposed
function checkMaxSale(address buyer, uint propValue) internal returns (bool status) {...}

// return the amount to the buyers who didn't get the item
function getReturns() public checkPhase(Phase.Done) notRecipient{...}

// end the sale and transfer balance to the recipient
function finishSale() public checkPhase(Phase.Done) onlyRecipient{...}
```