Introduction to Blockchain: Technology and Applications
Programming Assignment # 3

Out: April 9, 2022.
Deliverable-1 due: April 17, 2022 at 11:55 pm via LMS
Deliverable-2 due: April 18, 2022 at 11:55 pm

Course Rules

In the previous assignment, we learned using Remix for developing smart contracts and executing those in a simulated environment. The objective of this assignment is to learn deploying smart contracts (developed in Remix) on a real-world blockchain (Ropsten) using MetaMask (browser-based crypto wallet and blockchain interface to facilitate transactions on a real blockchain). You are welcome to discuss any issues and confusions with the course staff, however **DO NOT ask other students for guidance.**

Setup

You will need to do the following steps:

- Install MetaMask Chrome Extension (or equivalent for your browser).
- Create an account on MetaMask using the extension (follow the interactive procedure). **You will be required to submit your account address**.
- Open MetaMask from your extensions. On the top right you will see the available networks you can connect to (by default it is Ethereum Mainnet).
- Click Show/hide test networks. It will take you to the advanced settings from where you can enable the setting Show test networks to let you view the test networks.
- Go back to the MetaMask main page/screen and select the Ropsten Test Network from the Networks menu on the top right.
- On the deploy page (accessible by clicking the Deploy & run transactions sidebar button) in Remix, change your environment to Injected Web3. If you have MetaMask set up correctly, it will automatically connect itself to Remix (after asking for your permission to do so) and will allow you to deploy your contracts on Ropsten Testnet blockchain.

You will now be able to interact with contracts on the test network (Ropsten). For deployment of your own contracts and interacting with certain functions of already deployed contracts, you will require a cryptocurrency called Ropsten test Ether (rETH). You can acquire Ropsten tokens (rETH) from various faucets for free. We recommend the following:

- https://app.mycrypto.com/faucet
- https://moonborrow.com/

Feel free to ask the TAs for more rETH sources. Disclaimer: some faucets take a while to give you the coins, and some have restrictions for how often your IP address can request them. **Please start your assignment on time so you do not face any difficulties in acquiring tokens for deployment.**

Moreover, it takes a couple of seconds to make changes to the blockchain (for confirmation of transactions), unlike in the previous assignment where it gave you a simulated environment, in which case the transactions were instantaneous.

Introduction

The purpose of this assignment is to give you a real experience of developing a smart contract and deploying it on a network where everyone can access it and use it. We will require you to create a bidding smart contract in which the ownership of a particular asset (in our case just a hash representation) will be sold. The smart contract you develop will reflect the changes in ownership of one asset. As all of you (we hope) will be developing a smart contract for different assets and deploying it on the same network (the Ropsten Testnet), it will result in a network where you can place bids on assets of your peers (other students in the class). **Additionally, you are required to place bids for 5 assets, made available by 5 different smart contracts deployed by your peers**. We encourage you to share your contract addresses (**but not the code**) with one another to achieve this. Be sure to do this with the account address you submit to us. We will be checking your bids during the grading phase.

Bidding contract

Similar to the previous assignment, you will be writing a solidity contract. For this assignment you will simulate a bidding scenario for the ownership of a particular asset (in our situation: instead of the asset itself, you will calculate a **representative keccak256 hash value of your unofficial transcript pdf**).

You will need to alter the following functions in the skeleton code (provided) but make sure to not change the functions headers (name, input/output parameters and their types):

- constructor: You will be required to initialize the current owner of the asset, an array that will hold the addresses of past (and current) owners, as well as the representative hash value of the asset that is up for bidding. Initially, the owner will be the deployer of the contract and the history will only have the deployer. Note that the ownership can change over time if the owner sells the asset to a successful bidder. The owner is not an intrinsic part of the contract, rather a variable that identifies who currently owns the stored hash value. The hash value (representing the asset) will remain constant for the duration of the contract's lifetime.

- setMinimumPrice: This is where only the current owner of the asset will be able to set/change the minimum price they are willing to sell the ownership for. On calling this function, the contract should remove any bids below the new minimum price and return the Ether submitted in those bids to the corresponding bidders' addresses. Before this function is called for the first time, the minimum price should default to zero Wei. If someone other than the owner attempts to call this function, your code should return the error statement **"Only the owner can call this function"**.

- submitBid: This is the function to be called to submit a bid. There are a couple of checks that need to be in place for this function.

  1. Ensure that the owner cannot submit a bid. If the owner attempts to submit a bid, return the error statement **"Owner can't call this function"**
  2. Ensure that the bid is always greater than or equal to the minimum sell price. If the bid does not meet this condition, return the error statement **"Bids must offer more than minimumPrice"**
  3. If an address has already submitted a bid, then submitting another bid should **add** to the value of the original bid. An account submitting a second bid doesn't need to make sure the second bid is greater than or equal to the minimum sell price.

  NOTE: This is a payable function and execution of this will cause transfer of rETH from the bidder's wallet to the contract.

- sell: Firstly, keep a check to ensure that only the owner can call this function and that there is at least one bid. Find the highest bid and transfer the corresponding rETH from the contract to the current owner. The unsuccessful bidders (if any) are refunded their rETH. Call the Register contract that we have already deployed and submit your Roll Number (in the form: *20100031*) as well as the value of the highest bid. Change the owner of this asset to the address that made the highest bid and update the history of owners. If someone other than the owner tries to call this

function, return the error statement: **"Only the owner can call this function"**. In case there is no bid for an asset, calling this function as the owner should return **"There should be at least 1 bid"**.

- viewBid: A getter to view a particular bid at a certain index. Only the owner can call this function. The bids should not be visible to everyone. If someone other than the owner tries to call this function, return the error statement: **"Only the owner can call this function"**.

Apart from these, there are various getters that you will need to complete to ensure that the test cases work. They are specified in the skeleton code.
NOTE: The smart contract Register is deployed on the Ropsten network at the following address:
0x5801bd1Be6e6093745FCafA18De4b11217dDD0CE

NOTE: In order to call a function in a deployed contract, you would need the address and name of the contract as well as the full function declaration (which includes the name of the function, its input and output parameters, and the types of each of those parameters). This is provided in the skeleton code; please do not edit or remove this.

Interacting with smart contracts

Once you are done with your smart contract and have deployed it on the Testnet, you will have to interact with your peers' smart contracts. For this you will need their smart contract addresses so we encourage you to share your contract address with your peers (feel free to post on piazza), and ask them for theirs. What you will then have to do is use their smart contract addresses to bid on their assets. You will have to bid for **5 different assets** (which basically means you will have to bid in 5 different smart contracts).
Please note: You do not have to win the bid in this part. All you have to do is make sure you place 5 bids. We will only be checking that.

Submission

There are **two** deliverables for this assignments; both are REQUIRED.
Deliverable-1 (due on Sunday April 17, 2022 at 11:55pm):

Deploy the latest version of your contract and submit the following on LMS Assignments **before the above deadline**:

1. biddingContract_<your-roll-number>.sol
2. <your-roll-number>.pdf

where your-roll-number is your 8-digit LUMS email ID such as 20100031. The file <your-roll-number>.pdf will include the following information:

1. Your Ropsten account address (which deployed your final contract).
2. Your **final** contract address (you can deploy multiple times, just be sure to submit the address of the one you want graded).

Deliverable-2 (due on Monday April 18, 2022 at 11:55pm):

You are required to place bids for 5 assets, made available by 5 different smart contracts deployed by your peers. Be sure to do this with the account address you submitted to us the previous day. It is okay to place the bids even before the first deadline.

We encourage you to share the address of your deployed contract with one another. It is perfectly fine to anonymously post your contract address on Piazza for this purpose.