CS731

BLOCKCHAIN TECHNOLOGY & APPLICATIONS

ETHEREUM ASSIGNMENT

The aim of this assignment is to get you started with Solidity Smart Contract development for the Ethereum Blockchain. You will understand how to develop and deploy smart contracts and how to interact with them.

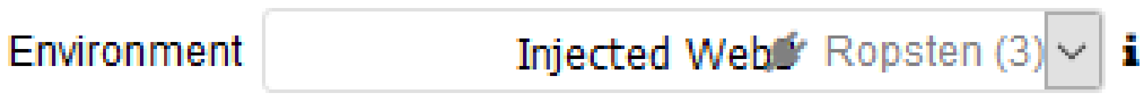
PART A
PREPARATION

This assignment will be on the **Ropsten** testnet of the Ethereum Blockchain. So, like demonstrated in the class you are required to download the **MetaMask** plugin (for Firefox or Chrome, depending on the browser of your choice). Remember to keep your password and the seed phrase in a safe and secure location. Select the network as Ropsten test network on Metamask.

Create an account and then, go to the **Ropsten faucet** (https://faucet.ropsten.be/) and get 1 ether for your account (on the Ropsten testnet). You will need at least two-three accounts for this assignment (you can add more for testing purposes). The faucet will give you 1 ether every 24 hours (unless you use VPNs), so start this assignment well in time or distribute that 1 ether among all your accounts.

Go to **Remix IDE** (http://remix.ethereum.org/) and get used to the interface. Try developing a few contracts to understand how it all works.

Running the contracts with **JavaScript VM** will run them locally and will not affect the blockchain in any way. If you want to deploy on the blockchain, set the environment to **Injected Web3.** If MetaMask is correctly configured, you'll see Ropsten written in grey as shown:



Environment Injected Web3 Ropsten (3)

Note: Remember, that contracts once deployed on the blockchain cannot be changed in any way. So, test your contracts thoroughly locally using JavaScript VM first before you begin deploying on Ropsten.

Understanding basics of solidity and developing smart contracts will be needed for this assignment. A few good references are:

- https://solidity.readthedocs.io/en/v0.5.1/
- https://ethereumbuilders.gitbooks.io/guide/content/en/solidity_tutorials.html

PART B
WARM UP: INTERACTING WITH A DEPLOYED CONTRACT

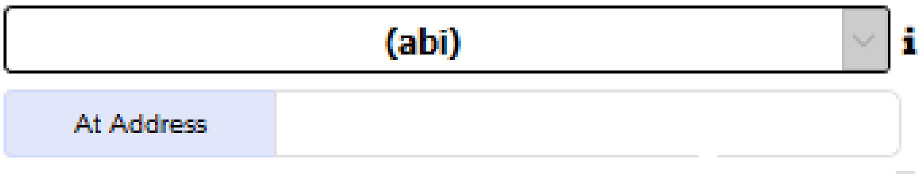
Now that you are familiar with Remix, it's time for the first part of this assignment. In this part you have to just interact with an already deployed contract.

We have a contract called `GetYourAnswer` deployed on Ropsten at

`0x956320c6e3607d3bd7e93312a804b068aff92ca3`

The contract ABI for that function is given to you in `part-b.abi`

Load that ABI in Remix IDE and go to the run tab. There, you can give the address of the contract in the box and load it from there. Make sure the environment is set to Injected Web3.



Do the following:
1. Call the `greeter()` function and it will give you a warm welcome.
2. Call the `myAnswer()` function with your roll number as a parameter. You will be given an output with data type as `bytes32`.
3. Call the `addMe()` function with your roll No. You'll see that MetaMask will be activated. Approve the transaction. The contract stores your roll number. We'll check it later.

Congratulations! You just interacted with a deployed contract!

Fill `part-b.csv` with the required information of the transaction hash, your address, your roll number, the output of `myAnswer()`, etc. This file will be your deliverable for this part of the assignment.

PART C
DEPLOYING YOUR OWN CONTRACT & INTERACTING WITH IT

Now that you have interacted with a deployed contract, it's time to start developing your own contract!

You are a school administrator and you want to put the course registration and grades management on the blockchain.

There are three main parties (create different addresses for them) –

- **The School Administrator** – your main account that will be used for deploying contracts on the chain. It also has the privilege to kill the deployed contracts (you should always have this option as a privileged option if you find a bug in the contract later). Also, only the school admin can add new courses.
- **Instructor(s)** – Each Course has an associated instructor. It has privileges to upload the midsem, endsem marks and also mark the attendance of the student.
- **Students(s)** – Anyone on the network can become a student. They can register for any course that is being offered and can view their marks.

For this you will create two contracts according to the given specifications –

- **Manager Contract –** main contract created by the school administrator. Used for adding courses. Also serves as endpoint for students to register for courses, view their grades, etc.
  - **Variables**
    - `admin` – address of the school administrator
    - `student` – mapping from address to roll number of students
    - `isStudent` – whether an address is a student or not
    - `course` – mapping from course number to course contract instance
    - `isCourse` – whether a course number is valid or not
    - `rollCount` – starting number for roll numbers
  - **Methods**
    - `Constructor` – initialize the admin address
    - `kill()` – kill the contract. Can be called only by the school administrator only
    - `addStudent()` – function that can be called by anyone on the chain to become a student in the school. Check if the address is not already a student.
    - `addCourse(int num, address i)` – add a new course with course number as `num` and instructor at address `i`
    - `regCourse(int cnum)` – register a valid student in a valid course with course number `cnum`
    - `getMyMarks()` – used by valid students to get marks `(int midsem, int endsem. int attendance)`
    - `getMyRollNo()` – utility to be used by a valid student to get his/her roll number.

- **Course Contract** – every course has an associated course contract. A contract is deployed for every course on the blockchain by the Manager Contract.
  - **Variables**
    - `admin` – address of the school administrator
    - `managerContract` – address of the manager contract
    - `instructor` – address of the course instructor
    - `courseNo` – integer course number
    - `struct Marks` – struct to store midsem, endsem marks and attendance count
    - `student` – mapping from rollNo to marks

        - `isEnrolled` – whether student with a roll number is enrolled in a course or not
    - **Methods**
        - `constructor` – initialize the addresses
        - `kill` – kill the contract. Can be called only by the school administrator only
        - `enroll(int rn)` – can be called by the Manager Contract only. Registers the student if not in the course.
        - `markAttendance(int rn)` – increase attendance of student with roll number `rn` by 1. Can only be used by instructor
        - `addMidSemMarks(int rn, int marks)` – Can only be called by the instructor. Uploads midsem marks of students with roll number `rn`
        - `addEndSemMarks(int rn, int marks)` – Can only be called by the instructor. Uploads endsem marks of students with roll number `rn`
        - `getMidsemMarks(int rn)` – Can only be called by Manager Contract. Return midsem marks of student with roll number `rn`
        - `getEndsemMarks(int rn)` – Can only be called by Manager Contract. Return endsem marks of student with roll number `rn`
        - `getAttendance(int rn)` – Can only be called by Manager Contract. Return attendance marks of student with roll number `rn`
        - `isEnroll(int rn)` – returns true if roll number `rn` is enrolled in the course. False otherwise. Can be accessed by anyone.

You are given skeleton code with function definitions for both these contracts. Complete these contract definitions and test on a local JavaScript VM. Create multiple addresses – administrator (who will deploy the manager contract), instructor(s) and students(s). Add Courses, add Students and check the workings of all the functions.

Note – Authorization is a big part of this assignment. Make sure that privileged functions are not called by anyone except the users intended.

After testing on the local machine, for **deliverables** for this part you need to do the following –

- Deploy Manager Contract with your address on the Ropsten test network
- Add a course with course number **731** and instructor address as **`0xE2AC47d5310478450111B2b4F7CA3bfa3751D9Cd`**
- You have now deployed two contracts on the Ropsten test network.
- Also upload your source code files on the network using the **verify contract** option available on etherscan

The graders will check whether your contract works as intended (stuff like - anyone can become a student, only the instructor can grade, only the admin can kill a contract, etc.), so make sure to thoroughly check and test on your own before submitting.

You also need to fill the information like the addresses, transaction hashes, etc. in the file named `part-c.csv.`

You also need to provide the completed solidity files `Manager.sol` and `Course.sol` along with the ABI files in `Manager.abi` and `Course.abi`

PART D
THEORETICAL QUESTIONS

Answer these questions in one-two lines only -

1. Why `tx.origin` should not be used for authorization?
2. What is contract ABI and why is it needed?
3. How does having the function declared as view change the way we interact with that function of the contract?
   Hint – try having functions with and without `view` on Remix
4. Difference between `assert()` and `require()`?
5. How is a checksum ethereum address different from a non-checksum address?

Keep your answers short and to the point. Put them in a file called `answers.txt`. This file will be part of the final submission.

DELIVERABLES

- Part A has no deliverables
- Part B – `part-b.csv` complete with all the required information
- Part C – `part-c.csv` complete with all the required information, deployed two contracts with verified source codes on etherscan, `Manager.sol`, `Course.sol`, `Manager.abi` and `Course.abi`
- Part D – `answers.txt` with the answers to the questions

Put all the deliverables in a **zip file** with your roll number (like `18111000.zip`) as the name and upload in time.

Note: Make sure that you create your own addresses (as shown in part A) and don't use a friend's ethereum addresses as this might be classified as plagiarism.