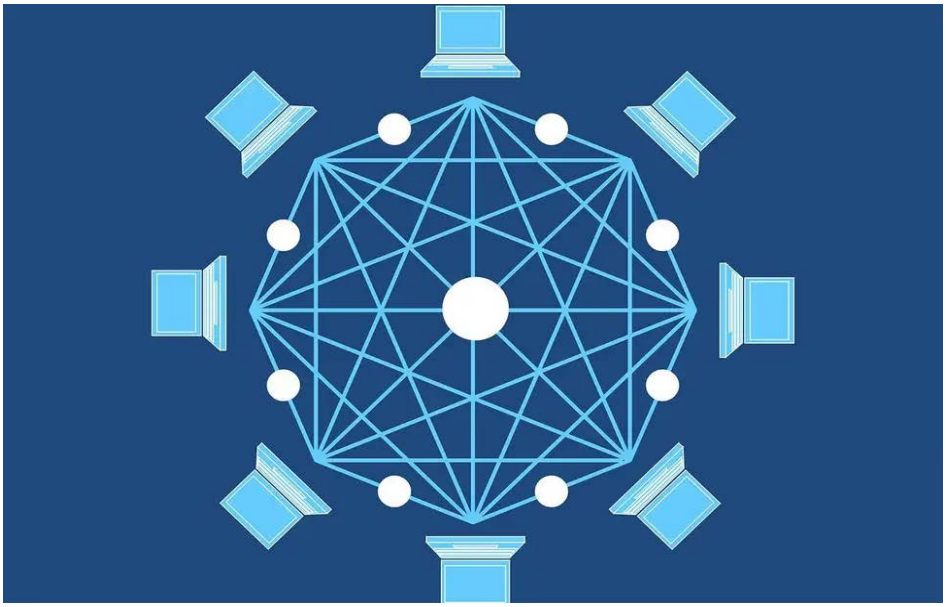
UNIVERSITY OF ILLINOIS AT CHICAGO

CS 540: Advanced Software Engineering

Topic: *Design and Implementation of Solidity Blockchain: Spring 19 CS courses*



Mentor:
Ugo Buy

Developed by:
Aditya Ramakrish
Adarsh JV
Pranali Loke
Utkarsh Gupta
Vanisre Jaiswal
Yogesh Patil

INDEX

(1) Design alternatives considered:

Tried and Tested IDEs:

- Remix
- Parity
- Truffle

Truffle vs Remix vs Parity:

- Truffle is a development environment/framework for smart contracts which can be accessed in the terminal, and Remix is an IDE in the browser.
- They both provide the ability to test and deploy contracts, but truffle can be included in projects as a build dependency, whereas remix contains an editor.
- We used truffle when building JavaScript projects based on smart contracts (e.g. DApps) and used remix for its debugging tools.
- Parity is an Ethereum client that is integrated directly into your web browser. Not only does it allow the user to access the basic Ethereum and token wallet functions, it is also an Ethereum GUI browser that provides access to all the features of the Ethereum network including DApps.
- Parity is novice in comparison with truffle and remix. The open source community is very limited. Proof of Authority is not well defined enough to list specific attacks.

Data Structures used:

- We have used a **struct** to store our Course model which consists of the following variables: a rubric, course name, instructor, venue and capacity.
- We are using a **mapping** from the index to the course model which we use to store and search for course details given a rubric.
- We store the addresses of the users who have access to add courses to our blockchain by pushing them into an **address array** at the time the contract is deployed.
- We also have a **rubric_list array** to check for duplicates course additions.

(2) Description of the implementation:

Implementation details:

- Created a smart contract containing course information and functions to add blocks (courses) to the blockchain and get course information.
- Created a web client to add and access course information to and from the Ethereum Blockchain.
- Used Web3.js as thin client to interact with the solidity code and display it on the web browser.
- Leveraged the tools below to create a blockchain and deploy it on the IPFS public network.

Tools Used:

- **Truffle framework** - It is a development environment, testing framework and asset pipeline for Ethereum. It provides:
  - Built-in smart contract compilation, linking, deployment and binary management.
  - Scriptable deployment & migrations framework.
  - Network management for deploying to many public & private networks.
  - Interactive console for direct contract communication.
- **Ganache** - It is used to fire up a personal Ethereum blockchain that you can use to run tests, execute commands, and inspect state while controlling how the chain operates. It acts as a GUI for the server that displays your transaction history and chain state.
- **MetaMask** - MetaMask is an extension for accessing Ethereum enabled distributed applications, or "DApps" in your browser. Also, acts as an Ethereum Wallet in your Browser.
- **Npm** - a package manager for the JavaScript programming language. It is the default package manager for the JavaScript runtime environment Node.js.
- **Web3 module in npm** - web3.js is a collection of libraries which allow you to interact with a local or remote ethereum node.
- **INFURA** - We used Infura which is a scalable back-end infrastructure for building DApps on the Ethereum blockchain. It is a method for connecting to the Ethereum network without having to run a full node locally on your system.
- **IPFS -** A peer-to-peer file distribution system (eg limewire). The front-end files

containing the address of the Ethereum blockchain of the application is deployed on the IPFS and is given a cryptographic hash. Only people with the URL containing cryptographic hash can access the DApp and interact with the Ethereum blockchain.

Functionality of the Contract:

- The **struct** is used to store our Course model which consists of the following variables: a rubric, course name, instructor, venue and capacity
- The **mapping** function is used to index the course model and retrieve to get the course details.
- As soon as the contract is deployed the constructor is called which appends all the addresses which need to be given access to add courses to the blockchain.
- The addCourse() function takes in parameters as rubric, course name, instructor, venue and capacity.
- As soon as the addCourse () function is called, a function modifier ifIssuer() is called.
- Function modifier is mainly used to check a condition prior to executing the function. So, you could re-use the same modifier in multiple functions if you are checking for the same condition. It reduces code redundancy. For example, the modifier ifIssuer() is used to make sure that the caller of a function has the access privilege to call the function addCourse(). The symbol _; is used to continue executing the rest of the function body.
- As soon as the ifissuer() modifier continues the execution of the function body the isOkay() function is called which checks for duplicate courses from the rubric_list. If a course already exists, the contract throws an exception and the contract stops executing.
- If the course does not already exist, the course information is stored in the mapping function and the transaction successfully executes and gets appended to the blockchain.

Tried Approaches for deployment:

- Github pages (Failed to interact with contract. Only static HTML page deployed)
- AWS EC2 instance (Failed to interact with contract. Only static HTML page deployed)
- Geth and Kovan testnet (Failed to link to the front-end HTML page. Successfully deployed Contract to the test-net.)
- IPFS and Rinkeby test network (Worked!!!)

Deployment details:

- We used INFURA to get a Rinkeby API key which can be used to access the Rinkeby test-network.
- This API key is given in the truffle config file to deploy the contract on the test-net.
- Also, we need truffle “HD Wallet Provider” which is a node dependency used to interact with the contract. This wallet provider can be added to the truffle’s config file before deploying the contract.
- Then truffle console is used to deploy the contract to the Rinkeby test-net using “truffle migrate --network Rinkeby”
- Our Blockchain is now live on the Rinkeby test-net.
- Next, we used IPFS a distributed file-sharing protocol to add our front-end files to interact with the blockchain in the Rinkeby test-net.
- We take the build directory containing contract’s json files to store the files on the IPFS system.
- This allows us to access the blockchain on the test-net using its address stored in the contract’s json file.
- After we deploy the front-end files on the IPFS system it is given a cryptographic hash which can be used in the URL to access and interact with the contract.

Positives of using our system:

- Re-entrancy attack avoided (No fallback function). We have used only a single contract for our app.
- Low transaction cost.
- Restriction through permissions.
- Efficient execution.
- No possibility of contract manipulation as no other contract interacts with ours.

(3) Testing Performed:

The steps performed for testing are as follows:

- Debugged the app using Remix IDE.
- Checked whether contract is getting created on our blockchain and a user is able to view and add course on our local system.
- Checked whether only permitted users can add courses to the blockchain.
- Duplicates were checked to ensure that no two courses exist with the same rubric on our blockchain.
- Null checks to ensure that users do not leave any field blank i.e. a course with insufficient information is not added.
- Ensured that proper alerts are in place to ensure that the user is aware in case of a failed transaction.
- Finally deployed the app and checked whether users are able to access our system using the link with the cryptographic hash to access the IPFS instance which has the front-end files.
- Checked that the users are able to access the Ethereum wallet in MetaMask and are able to interact with the blockchain deployed on the Rinkeby test-net.

(4) Reflection on the overall experience:

- It was an enriching experience to put to practical use the concepts we learnt through the many research papers.
- It was a novel project for everyone that helped us to dive into a completely different domain.
- It gave us a good hands-on experience to explore this diverse topic (blockchain) by developing it from scratch.
- We had an amazing opportunity to work on many tools to implement and deploy the blockchain on a private network. The overall experience gave us insight into creating a blockchain and securing it from various vulnerabilities.
- We got an experience of building a DApp which included implementing an end-to-end application right from scratch. This included the contract, the front-end HTML and web3.js, a framework to interact with the contract using the front-end HTML page and then deploying the DApp into production.
- The blockchain papers coupled with this project gave us a really good insight on the overall working of the blockchain including its flaws and merits. The papers introduced us to this fairly new domain and helped in understanding its basic theoretical concepts. And the project helped with the practical application of those concepts. The project specifically involved a lot of researching and exploring various tools which could be best suited for a successful deployment of our project.

(5) Steps to use the DApp:

- Firstly, install the go-ipfs from the website using the given link and install the pre-built package called go-ipfs. (https://docs.ipfs.io/introduction/install/)
- Extract the zip file and open your terminal and navigate to the directory of the go-ipfs and follow the installation instructions given in the above link depending on the OS version.
- Run the following commands to have ipfs up and running:
  -> **ipfs init**
  -> **ipfs daemon**

- Make sure you have the extension "**IPFS Companion**" added to your chrome browser which can be found on the chrome web store.
- Use the URL shared by us with the cryptographic hash to access the DApp.
- Make sure you are connected to MetaMask and "Rinkeby test network" is selected.
- You are now set to use our DApp and test its functionality. But you only have access to the "Get Course" functionality.
- To get course details just type in the Rubric of the course and click on "Get Course" button to get all the details of that course. Anyone with access to the link and a MetaMask account on Rinkeby test-net can view the course details with the Course rubric if it already exists in the blockchain.
- In order for you to have "Add Course" functionality make sure you give your Rinkeby address to us so that we can grant you access to "Add Course" functionality.
- Also, you need to get some fake ethers for your address. For that go to https://www.rinkeby.io/#faucet and tweet your address on twitter. Then use the tweet's URL to request for 3 ethers. The ethers should be added in 30 seconds.
- You now have access to add course functionality.
- Now you can give course details and confirm the transaction to add the course onto the blockchain using the "Add Course" button on the Dapps. Only privileged users have access to "Add Course" i.e. only few addresses have been added to the contract who will have access to "Add Course".

Code Locations:

- The contract should be in ATSE V2/Contracts/Course.sol
- The Web3.js code should be in ATSE V2/src/js/app.js
- The HTML page should be in ATSE V2/src/index.html

(6) Security Considerations:

Smart contracts are “immutable”. Once they are deployed, their code is impossible to change, making it impossible to manipulate the code and fix any bugs.

So, we took care of some of the few security issues as follows:

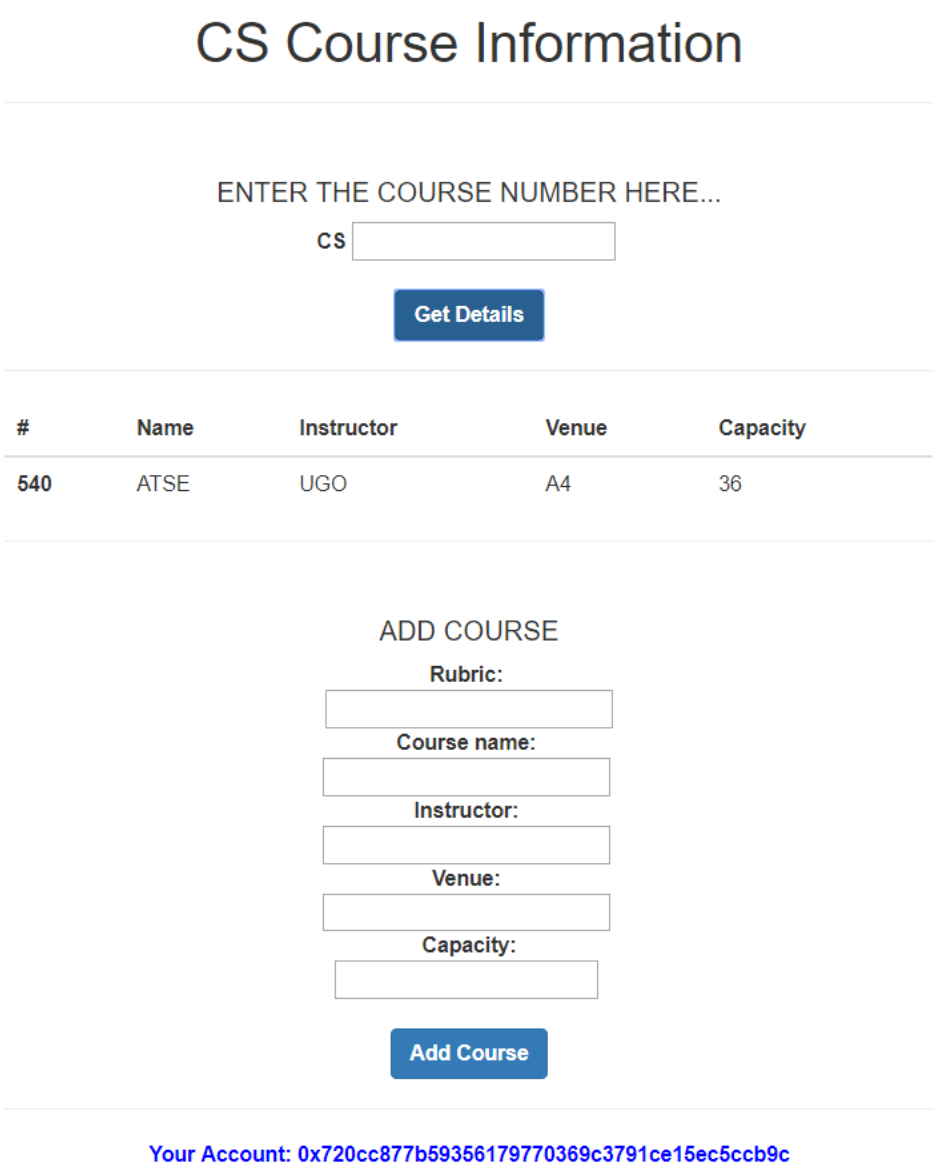
- **Overflow:** An overflow is when a number gets incremented above its maximum value. Solidity can handle up to 256 bit numbers (up to $2^{256}-1$), so incrementing by 1 would result into 0. Chances of overflow to happen is highly unlikely as number of courses won’t reach $2^{256}-1$.

- **Underflow:** When the number is unsigned, decrementing will underflow the number, resulting in the maximum possible value. There’s no scope of underflow as no decrementing of *uint* is present in our smart contract.

- **Visibility & delegatecall:** Not possible as access is restricted to limited people.

- **Re-entrancy:** Re-entrancy attack is avoided as there is no fallback function in our smart contract.

Snapshot of our DApp UI:



(7) References:

- Parity- https://www.parity.io/
- Hackernoon.com - building a private blockchain network (https://hackernoon.com/heres-how-i-built-a-private-blockchain-network-and-you-can-too-62ca7db556c0)
- Medium.com https://medium.com/@lhartikk/a-blockchain-in-200-lines-of-code-963cc1cc0e54
- Ethereum Github repositories https://www.youtube.com/watch?v=3681ZYbDSSk&t=5167s
- Tutorial for building an Ethereum DApp with integrated Web3 Monitoring (https://www.moesif.com/blog/blockchain/ethereum/Tutorial-for-building-Ethereum-DApp-with-Integrated-Error-Monitoring/#)
- Guild to create first decentralized application (https://www.youtube.com/watch?time_continue=6&v=gSQXq2_j-mw)
- Beginner's Guide: Smart Contracts Programming Tutorial in Solidity (https://www.youtube.com/watch?v=R_CiemcFKis)

- Solidity for Beginners - Smart Contract Development (http://www.DAppuniversity.com/articles/solidity-tutorial)
- A Guide to Building Your First Decentralized Application https://www.youtube.com/watch?v=gSQXq2_j-mw
- IPFS Companion - https://chrome.google.com/webstore/search/ipfs%20companion?hl=en
- Github pages - https://pages.github.com/
- Security - https://medium.com/loom-network/how-to-secure-your-smart-contracts-6-solidity-vulnerabilities-and-how-to-avoid-them-part-1-c33048d4d17d
- How to Host a Website using EC2 instance https://www.youtube.com/watch?v=30MqiWnazew&t=872s
- Remix - https://remix.readthedocs.io/en/latest/