Negotium, Ethereal Contracts

Savada Wilson
Dr.Banda
CSC 4980
May 28th 2020

I. Introduction

Ethereum, as a blockchain, has the capability to function as a platform for other projects. Through the use of its Solidity programming language one can create complex automated protocols called smart contracts. The ledger aspect of blockchains also generally provide near ideal record keeping systems being that they are immutable. Using code in fusion with the inherent boons that come with using a blockchain, or more specifically Ethereum, one can solve problems that previously had no clear answer. One such problem that blockchain can solve and was even created for is, the rebalancing of global inequalities. For instance , In the present day, all that any entrepreneurial minded person, business guru , or even investor can talk about, is the upcoming 4th industrial revolution. However most denizens of the world will not get to prosper from the event in the same way legacy multinational corporations will. Instead, MNCs will once again internationally monopolize yet another aspect of the global commerce sphere. This scenario highlights a key point in which blockchain can provide a cheap yet scalably feasible solution to the current issue of decentralizing global industry and commerce. Smart Contracts can be used to automate the transfer of funds and data at the behest of specific parties, or when codified criteria inside the contract have been met. The key aspect of this , is that it eliminates the necessity or mitigates the interface of a middleman, governmental trade system, or human headed fiduciary system. Discounting smart contracts, one can even send funds to the address of a non-governmental arbitrating party's Escrow system as they judge disputes and proper allocation of funds between international parties. In this project, the broad aim was to display how replicable a court-like decentralized database of contracts is. This Decentralized App has been designed with the target of providing a platform where people internationally can create their own smart contracts to self govern how they will ,trade with, invest in, and interact with other like minded people globally.

II. Methods

To create something in the same vein as this project, there are a few tools necessary for the creator to procure first. To start, one must have an IDE in mind that they plan to use Solidity on. While many may like using Remix, an online IDE. It is suggested that one uses Visual Studio as it is an IDE that many are familiar with. After choosing the IDE one plans to use, all they must do is determine a location on their machine for their project and simply create the containing folder for it. Subsequently, after creating the folder for their project One must install Truffle on their machine, which if using a Linux operating system can be done using "npm install -f truffle". Next, using " Truffle unbox pet-shop" can save the developer lots of time in creating the files they would normally need in their end product. In fact It is imperative that one sets up Truffle inside the folder correctly, and using the aforementioned command surely eliminates many of the common pitfalls one can run into. After the pet-shop has loaded, the developer can delete all unnecessary files such as: Adoption.sol, pets.json etc. Afterwards, one must then install Ganache or retrieve the appimage both of which can be done with Github. With all that taken care of one may do preliminary development on their smart contract or preemptively get the MetaMask extension on their default browser.

Using these four main tools one could reproduce the work of this project.

Moving on, many helpful tutorials on creating similar Dapp projects can be found online for those who may want to take a variant approach to their development. One should be mindful however that the official documentation of the functions and data types used is the best source for cross referencing information given in the tutorials. Ideally, in total, for the creation of the Dapp one will use a stack of languages that span Solidity, Javascript, Html, and CSS. On the Javascript layer one would use MetaMask and Web3 to interface with the calls to one's Smart Contract. Similarly Javascript functions will also be called from their HTML file, when the user clicks buttons and submits information. Lastly the CSS file will be used to display the visual effects and animations of the Smart Contract.

III. Discussion

The project overall turned out to be successful as it displays the ability of blockchain to function as a database, where users can enter their own information and interact with the content that others create. Specifically in this Dapp one is able to enter the name of their contract, the amount of collateral or capital to be put in escrow, and a Description that that details the obligations and articles of the contract. Although more complexity was initially intended at the conceptualization of the project, the final version of the Dapp still has many key actions one that can be carried out using the final product.

Beginning with the smart contracts ability to differentiate users, upon running "truffle migrate --reset" the constructor for the contract is called and preliminary data is entered into the contract for the sake of viewing. Using CSS and HTML the Dapps page is displayed along with three tabs that give users three options. At this stage if MetaMask is not enabled, one cannot click the tabs and their Account number will not be shown on the screen. Upon activating Metamask and reloading, if not already the smart contract will send a request to Metamask that it would like to connect to it. Upon connecting and reloading the webpage, One is given three abilities, the ability to create a contract, the ability to view deployed contracts, and the ability to view contracts that they have created only.

Hypothetically, if one wanted to create a contract, they would click "Create a Legally Binding Contract". Afterwards, one would enter in the Title of the contract, the Escrow amount, and the Description. Following suit, they would click the submit button, which will then push the entered data onto the Smart Contract. Now the newly created contract is only visible to the user via the "View My Active Contracts" tab. For the contract to be visible to all other users, it would have to be deployed using the deploy button that should appear under the contract in the "View My Active Contracts" tab. Now it would be visible along with all the other contracts that users had created on the Dapp. After clicking to view available contracts they would see a slew of data entered by each user who had already created a contract. The data comes in columns and each contract deployed is given its own row. The columns are: contract id, name, escrow holdings, description, and creation date. These columns allow a user to see the most important details prevalent to the contract they intend to accept. One can then accept such a contract using the accept button, at which point the address they are accepting from would be recorded inside the parties struct within the Solidity code of the DApp. This was designed in such a way that the information could be retrieved by the Arbitrators for contact or correct transfer of funds later. However despite this being the intent, implementation of an interface tailored to Arbitrators or non-user administrators of the database was not completed. To substitute for this, after one accepts the contract, it is removed from the "View Available Employment Contracts" tab. Next the escrow account for which the contract creator must deposit to will then be displayed in their "View My Active Contracts" tab. This humble work around is meant to display the potential for growth in alternative ways one can create a fiduciary system.

gas on the database is used a total of three times throughout the lifespan of a contract or "entry" on the Dapp. Each time, Metamask is requested for a payment which the user can accept or reject. Initially, at the inception of this project it was hoped

that the amount of gas deployed would be few in an attempt to lower the overhead of utilizing the Dapp for a user. As a compromise, the locations at which the gas payment requests occur are strategic in their aid of the function of the Dapp. The gas request payments occur at: user contract creation, user contract deployment, and user contract acceptance. The initial payment at contract creation is meant to secure a place of one's contract in the network as they are then writing information to the smart contract Dapp. The second payment of contract deployment was initially made for the purpose of taking the necessary payment entered in by the user during contract creation and sending it to an arbitrator controlled address. This was ditched as it proved more challenging than originally thought, still the deploy button writes to the smart contract but does not request payment of the Escrow Amount. Lastly payment is requested at the contract acceptance button as a person's public key is written to the blockchain and depending on the contract one is accepted, payment may even be necessary from their end to even participate.

IV. Conclusion

To conclude, the project was completed and the full product of the Dapp panned out to be good. Albeit a notable number of setbacks and compromises were encountered and had to be made, the overall goal of displaying the Database capabilities of the Ethereum ledger through the use of Smart Contracts. The ability to make something with the potential to be radical and innovative was shown to be possible with only a stack of 4 languages: Solidity, Javascript, HTML, and CSS. This Smart contract does have many rough edges but all things considered it proves to be a significant use of Blockchain technology, in its desired end-goal and function.