GROUP 22
Submitted By:
Pankaj Kumar Singh **15116038**
Sunil Saini **15116058**
Sandeep Kumar Marandi **15116049**
Hemant Yadav **15116024**

Project Link: https://github.com/sunildahiya/cmanage

Decentralized App for Credit Management

Introduction

Our project involves making a decentralized web app which can be used to store transaction details mainly money lending. The entire application set up and build was done on a fresh installation of Linux Mint 18.02 Cinnamon.

1.Set up the development environment.

Dependencies:

1. Nodejs
2. Ethereumjs-testrpc
3. Web3
4. Solc
5. Live-server

Install missing dependencies with npm.

Ethereumjs-testrpc: Create local blockchain network
Web3: API to interact with the blockchain
Solc: Programming language to write smart contracts in Ethereum

Instead of developing the app against the live blockchain, we will use an in- memory blockchain called testrpc. The testrpc creates 10 test accounts to play with automatically. These accounts come preloaded with 100 (fake) ethers.

This project currently works with web3js version 0.20.1, Node version 6.11.05 npm 3.10.10.
Materialize-css framework is used to style the HTML page.
Live-server used to make the local server which automatically reloads the page in your browser when any of file changes.

2. Writing the contract

We used the solidity programming language to write our contract.We wrote a contract called Transaction with a constructor which initializes an array of people we can lend and borrow from. We then wrote the neccessary functions for the working of the sending and recieving in the application. The constructor is invoked once and only once when we deploy the contract to the blockchain. Unlike in the web world where every deploy of your code overwrites the old code, deployed code in the blockchain is immutable. i.e, If we update our contract and deploy again, the old contract will still be in the blockchain untouched along with all the data stored in it, the new deployment will create a new instance of the contract.
All the below codes are combined into a script that does all the jobs together i.e. running the testrpc, running the live server, compiling and deploying the contract. It will output a contract address the needs to be pasted into the contract address in the JavaScript.

We used solc library within a node console to compile our contract. web3js lets us interact with the blockchain through RPC. We used this library to deploy our application and interact with it.

```
> code = fs.readFileSync('Transaction.sol').toString()
> solc = require('solc')
> compiledCode = solc.compile(code)
```

We then deploy the contract. For this we create a contract object (TransactionContract as shown below) which is used to deploy and initiate contracts in the blockchain.

```
> abiDefinition = JSON.parse(compiledCode.contracts[':Transaction'].interface)
> TransactionContract = web3.eth.contract(abiDefinition)
> byteCode = compiledCode.contracts[':Transaction'].bytecode
> deployedContract=TransactionContract.new(['Pankaj','Sunil','Sandip','Hemant'],{data:
byteCode, from: web3.eth.accounts[0], gas: 4700000})
> deployedContract.address
> contractInstance = TransactionContract.at(deployedContract.address)
```

TransactionContract.new above deploys the contract to the blockchain.. Let's see what are all in the hash in the second argument:

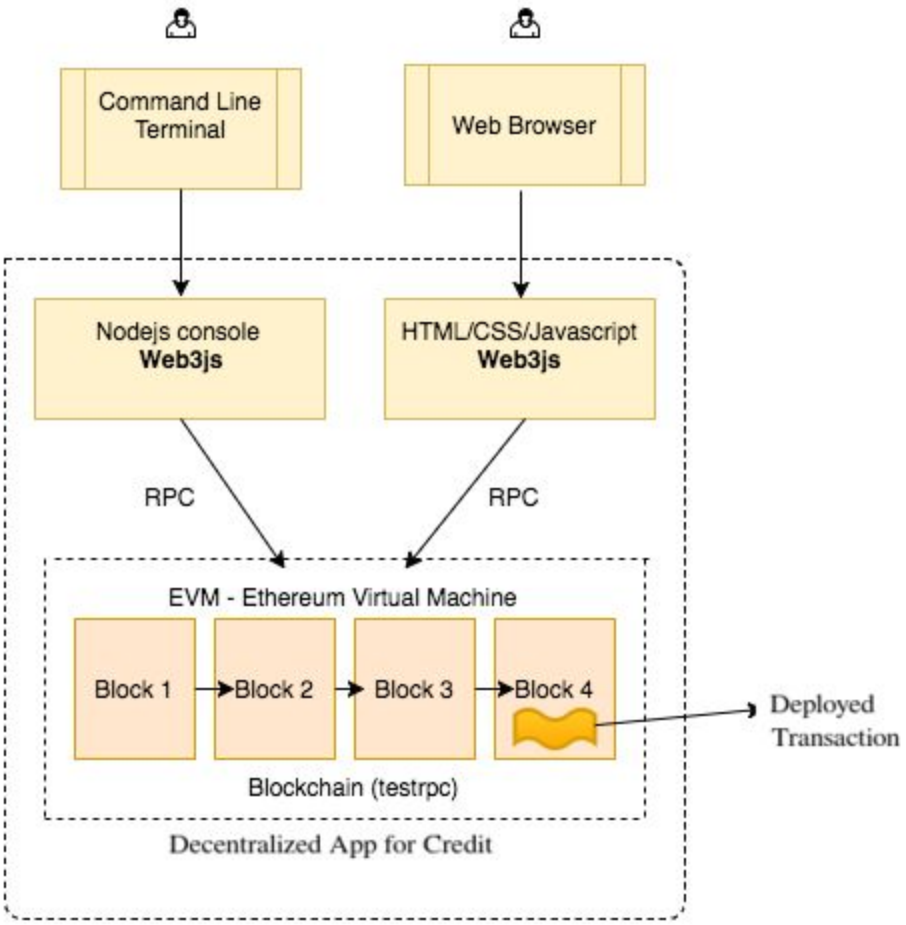
1. **data**: This is the compiled bytecode which we deploy to the blockchain.
2. **from**: The blockchain has to keep track of who deployed the contract. In this case, we are just picking the first account we get back from calling web3.eth.accounts to be the owner of this contract (who will deploy it to the blockchain). Remember that web3.eth.accounts returns an array of 10 test accounts testrpc created when we started the test blockchain. In

the live blockchain, you can not just use any account. You have to own that account and unlock it before transacting. You are asked for a passphrase while creating an account and that is what you use to prove your ownership of that account. Testrpc by default unlocks all the 10 accounts for convenience.

3. **gas**: It costs money to interact with the blockchain. This money goes to miners who do all the work to include your code in the blockchain. You have to specify how much money you are willing to pay to get your code included in the blockchain and you do that by setting the value of 'gas'. The ether balance in your 'from' account will be used to buy gas. The price of gas is set by the network.

GUI to interact with the application

Now that most of the backend work is done, all we have to do now is create a frontend for the application. We created a simple HTML file file with a table of debts and credits and an option to lend money to someone by typing in their name and amount and clicking the send button. We also created a CSS file to style the application. Then we created a JavaScript file to invoke the transaction commands. JSON is used to send and receive data from blockchain database.



Executing the Application:

- To execute the application go to the project folder and open terminal.
- Run the commant ./terminal.sh
- Wait for the contractAddress. Two extra terminal windows will open. One has the blockchain running and the other has live-server running.
- Copy the contractAddress and paste it in the contract address field in the javascript.
- The index.html will be opened by itself by now.
- Now we are ready to use the application
- Type in Name and Amount to send and click the Send button.

Application:

This application is functional to send and receive credit from person to person. The main motive was to have a decentralized system and this idea can be carried on to create many other decentralized applications.

Further Work:

This application has only been tested on local blockchain created on laptop and need to be deployed into the global test network of blockchain . There are still problems with using multiple accounts on a local network so we need to test it on a global test network. It also needs to be linked to Ethereum accounts to add to the functionality of the application.

References:

https://ethereumbuilders.gitbooks.io/guide/content/en/glossary.html
https://github.com/ethereum/wiki/wiki/JavaScript-API
https://github.com/ConsenSys/smart-contract-best-practices
https://github.com/ethereum/wiki/wiki/JSON-RPC
https://github.com/ethereum
https://github.com/ethereum/web3.js