MYTRANSCRIPT: AN ACADEMIC TRANSCRIPT DECENTRALISED WEB APPLICATION SYSTEM BASED ON THE ETHEREUM BLOCKCHAIN

ONWUASOANYA N.C; & EZE B.E.
Department of Computer Engineering, Federal polytechnic Ilaro.

ABSTRACT

MyTranscript is an academic transcript web application built on blockchain technology. The system leverages the Ethereum blockchain to create a decentralised academic transcript database whose aim is to simplify academic transcript sharing among network participants (the Institution, the Student, and an Employer) while providing the security benefits of a decentralised system. For privacy reasons, the system implements a restriction to only allow an Employer access to records he has been permitted to view. This access is given only by the Student whose record is concerned. However, the Institution have unrestricted access to Students' academic transcript. The system is built on the latest software technologies: Node.JS, REACT, Truffle etc., that enhances security and provide intuitive user interfaces for better interactions among its participants. The immutability, unique hashing algorithms, timestamps and transparency provided by a blockchain network makes records stored and generated on this system trustworthy and prevents the occurrence of counterfeits documents being distributed by cyber criminals. The testing phase of the development cycle was done immediately after each system logic function was implemented to avoid the debugging difficulty that comes with ambiguous codes, using dedicated JavaScript tests libraries: chai, mocha, and chai-as-promised. The application was built to be responsive, so, page contents were optimised and arranged to display perfectly across the different device screen sizes the system was tested on. The result during system use performed as expected, although, there is a significant drop in speed during activities involving the system interaction with the blockchain. All other activities, such as page navigation, exam record arithmetic, and page preview had no noticeable delay with response occurring instantaneously.

***Keywords**: Blockchain, Ethereum, Transcript, Decentralised system, Intuitive User Interfaces, Hashing algorithms*

INTRODUCTION

A blockchain network is a peer-to-peer network in which a peer (also known as a miner, node, or bookkeeper) is in charge of transaction validation, block generation, and block validity verification. A bigger number of network peers must agree on a network rule not just to certify the authenticity of transactions in a block, but also to force a newly produced block to follow the predetermined agreement. To put it another way, all peers must adhere to a recognized consensus mechanism to guarantee data integrity without the need to determine if a node in the network is trustworthy or malicious. The most common consensus protocols are Proof-of-Work (PoW), Proof-of-Stake (PoS), and Practical Byzantine Fault Tolerant (PBFT).” (Lamport, Shostak, & Pease, 1982)**.**Blockchain technology births a new generation of applications, commonly referred to as *decentralized applications* (DApp), which can for example; support the execution of cross-organizational business processes. In a few years of its inception, these technological innovations have become a trend to follow. International leaders, its members and organizations are starting to look meticulously at the application of this technology, and some have already adopted it. The crux of this “innovative technology is the production of immutable trustworthy records without the need of a trusted third party.” (Lemieux, 2017). According to Redman (2016), it is a technology to watch.

In most school worldwide, documenting and issuing student’s transcripts are traditionally done uploading the results in a central server and printing the downloaded transcript when required in a paper format which will be sent to a demanding party by a trusted courier service company. This process takes time, is bound for counterfeiting and costly.

For the following reasons, a blockchain system for academic transcripts will solve the aforementioned issues:

1. Transcripts, both unofficial and official, are recorded on a distributed ledger so that participants (school and employer) may access them from anywhere in the globe through the internet. In the local scene, students can view their academic transcripts at their leisure while remaining confident in their authenticity. For employers wishing to view employees’ academic records they can simply request the transcripts from the concerned student via the application and be rest assured that supplied information is genuine.
2. Transcripts stored on a distributed ledger are tamper-proof; because it is difficult to update transcript data in numerous nodes at the same time, a malefactor cannot edit a transcript by himself.

3. It saves time and money and ensures document reliability and accessibility over the internet.

LITERATURE REVIEW

BLOCKCHAIN ARCHITECTURE

Figure 1 depicts a class diagram of a general blockchain network, which includes the basic components of peer (miner, node, or bookkeeper), consensus, and blockchain. “The application program interface (API) is a basic class that allows clients to connect with the blockchain network. Clients can be designed as a desktop, internet, or mobile application that permits users to conduct transactions after seeking IDs.” (Nguyen, 2018)

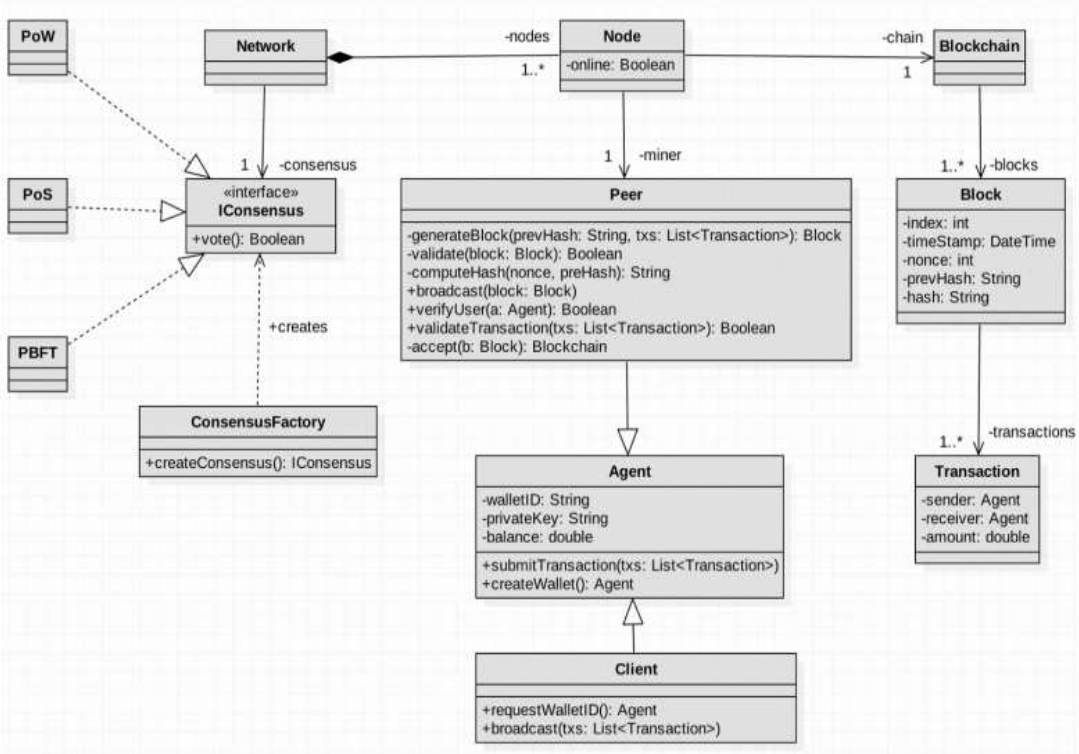

Figure 11: Blockchain class diagram

Blockchain

A block is a storage vessel for transaction and meta data. A block header is used to hold meta data. In the Bitcoin blockchain, for example, a block stores hash value, preceding block hash value, nonce, date, size, number of transactions, block difficulty, and so on. A block also contains a list of transactions, which might be

legitimate or invalid based on the blockchain solution's design. A blockchain is therefore a series of blocks linked together by pointers that are represented by the hash values of the blocks before them. “Because changing a single bit of data in the current block will affect the hash value of the preceding blocks, this design assures that data in the current block cannot be modified. It decouples not just the present block from the next block, but also the next block from the block after that, and so on, in a sequential manner.” (Nguyen, 2018)

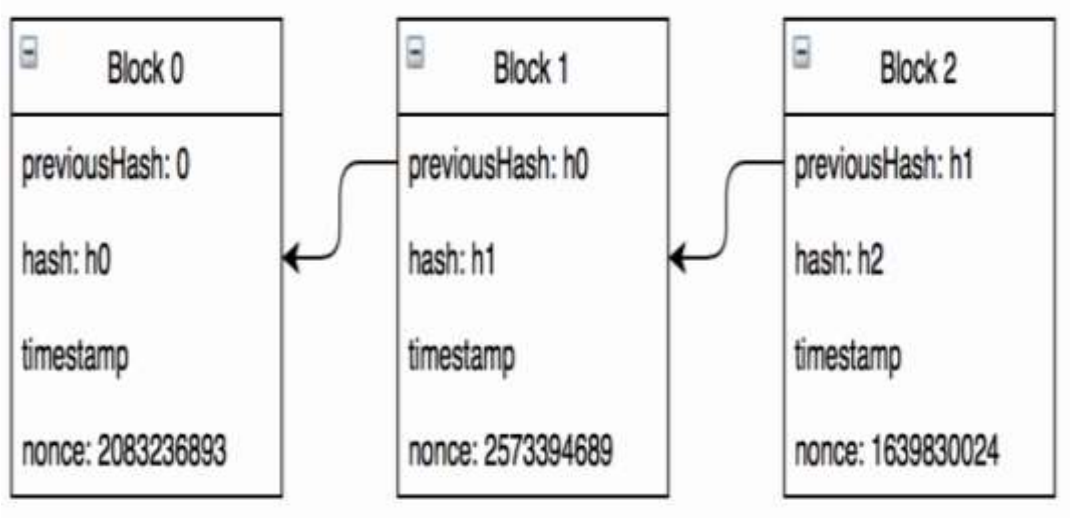

Figure 12: Blockchain

THE ETHEREUM BLOCKCHAIN

For the sake of the decentralized application we’ll be building later in this paper, it is pertinent to discuss the Ethereum blockchain as it is the blockchain technology our application will be built on.

Vitalik Buterin and his colleagues suggested Ethereum as a solution for a new blockchain platform to construct decentralized apps in 2013 (Ethereum Wiki, 2015). The goal was to provide a “platform for creating smart contracts, automating company payment processes, and a "world machine" or environment for executing smart contracts.”

To understand how the blockchain state is recorded within a block, we explore the core components below.

Ethereum Virtual Machine (EVM)

“The Ethereum Virtual Machine (EVM) is a sandbox and run-time environment for smart contracts that are loaded on Ethereum network nodes (miners)” (Wood,

2015). Because EVM is separated from the host machine's network, file system, and other processes, hackers cannot gain control of it to modify smart contracts. Solidity is a programming language for creating smart contracts, which are then converted into EVM bytecode and published to the Ethereum blockchain using Ethereum client software.

Gas and Payment

Each transaction on the EVM includes a fee that a requester must pay to a miner that hosts the EVM in order for the transaction to be completed. A miner is paid in Ether to validate transactions, execute chain codes, and produce a new block to update the status of the Ethereum network. The term "gas" is used to describe the total cost of all EVM operations. A sender specifies a gas limit and price for each transaction to indicate the charge that the sender is prepared to pay for the miner to perform the transaction. The gas limit and price are calculated in Wei (ethereum's smallest unit), with 1 Ether equalling 1018 Wei" (Wood, 2015).

Smart Contract

A smart contract is an agreement between accounts on the Ethereum network to actualize a transaction when a pre-defined set of criteria in the contract are satisfied. The phrase "smart contract" refers to a contract that may be used to initiate a transaction without the intervention of any other party. A smart contract is, in reality, a software program written in the Solidity programming language that is subsequently compiled to bytecode and published to the Ethereum blockchain. "This bytecode is performed in isolated EVMs that are deployed on Ethereum network dispersed nodes. When a smart contract is submitted, no one can change the code because it is protected by the Ethereum blockchain's security features."

Account

The Ethereum network has two types of accounts: external and contract accounts. An external account is comparable to the Bitcoin network's wallet idea, which allows users to submit transactions to the network. A contract account is a storage location for smart contracts that run on EVMs. "A contract account executes the chain code (smart contract) on behalf of people, whereas an external account is where humans interact with the Ethereum network" (Dannen, 2017).

Characteristics of an external account include: Ether balance (ETH), Ability to make transactions, Controlled by a private key, No contract code.

The contract accounts have the following characteristics: Ether balance (ETH), Keep contract code in the memory, Can be triggered by an external account or another contract account sending a message, Can perform complex operations when executed, Possess no owner after release to the EVM, Possess their own persistent state and have the ability to activate other contracts.

RELATED WORKS

- Student Academic Record Management System (Eludire, 2011). The solution suggested here is based on a centralized database server. It has the ability to keep track of pupils' academic records. However, it was incapable of generating academic transcripts.
- "Students Record Analysis and Examination Result Computation Algorithm" (Osagie & Mallam, 2014). The application proposed here was capable of processing students' results, however, like Eludire's system, it was incapable of generating academic transcripts.
- "A Centralized Transcript Processing System" (Omogbhemhe & Akpojare, 2018) built on a MySQL server database. The system was comprehensive and processed students' transcript as intended. However, it lacked the trustworthiness and transparency a decentralised database would have provided.
- Gradubique (Nguyen, 2018), which is a Blockchain-based academic transcript database. Gradubique was built on Linux Hyperledger Framework (HLF), unlike Ethereum which is a permissionless (i.e. participation is allowed to all members on the internet) Blockchain, HLF is a permissioned (i.e. participation is restricted to a select network) Blockchain, and is most suitable for B2B transactions.

METHODOLOGY

This section describes the implementation of a web-application that stores and retrieves student academic results over the Ethereum blockchain network.

We have chosen to build this Decentralized Application (DApp) on the Ethereum blockchain because ethereum is a platform for many other blockchain solutions. Several blockchain solutions run on the Ethereum blockchain, for this reason, it is the most popular blockchain network that favours the development of DApps and has extensive development/usage resources on the internet.

DESIGN ANALYSIS

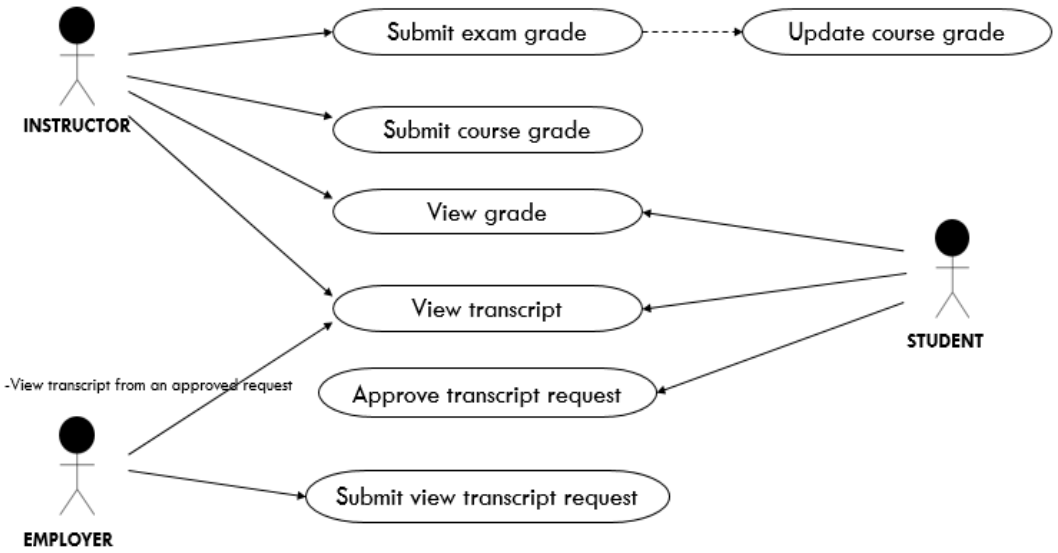

Figure 3: Use case diagram

Figure 3 above describes the use case diagram for the system.

SELECTION OF MATERIALS

This section describes the technologies that make up MyTranscript and the reason behind the selections of the technology.

NODE.JS

One of the core dependencies we have chosen to build this application with is Node.js. “Node.js is a server-side framework for creating fast and scalable network applications that is built on Chrome's JavaS6cript runtime. Node.js is lightweight and efficient because it employs an event-driven, non-blocking I/O architecture. It's ideal for data-intensive real-time applications that operate across several devices.”

Node.js also comes with a large library of JavaScript modules, which makes it easier to construct web applications that rely heavily on Node.js.

TRUFFLE

Truffle is the most widely used Ethereum programming framework. We have chosen Truffle for the following reasons:

1. Smart Contract Lifecycle Management – Truffle takes care of maintaining smart contract artifacts automatically. Support for bespoke deployments, library linking, and complicated Ethereum applications is also included.
2. Automated Contract Testing – Truffle allows automated tests to be written for contracts in both JavaScript and Solidity, and facilitates fast contracts development.
3. Scriptable Deployment and Migrations – Truffle fosters dapp's evolution and ensures that contracts can be maintained far into the future.
4. Simple Network Management – Truffle allows the developer focus on dapp development while it manages all network artifacts.
5. Powerful Interactive Console – Truffle provides an interactive console that allows the developer interacts with his built smart contracts using powerful Truffle commands. This greatly saves development time.

GANACHE

Ganache is a personal Ethereum Blockchain for deploying contracts, developing dapps, and running tests. We have chosen to use Ganache for the following reasons:

1. It is available as both a desktop application as well as a command-line tool.
2. Ganache is a popular Blockchain development tool with wide range of online resources that can prove useful for reference during personal development.
3. Ganache has good visual mnemonic and well defined account information that allows one to quickly see the current status of all accounts, including their addresses, private keys, transactions and balances.
4. Ganache also provides a comprehensive Blockchain log output and responses that are vital debugging information.
5. Ganache has a built-in block explorer and powerful mining settings.

REACT

"React is a JavaScript library for building user interfaces. React makes it easy to create interactive user interfaces." We have basically chosen React for its ability to create interactive user interfaces. By simply designing views for each state of our application, React automatically updates and renders components when state data changes.

INSTALLING THE DEVELOPMENT DEPENDENCIES

This system was developed on a Windows 10 computer system. The following installation methods discussed are that for Microsoft Windows 10 operating system.

Node.js Installation

Node.js can be downloaded directly from the node.js website by simply navigating to https://www.nodejs.org/en/download/. To get the most recent default version, click the Windows Installer icon. The most recent version at the time this project paper was produced was node-v16.13.1. The NPM package manager is included in the Node.js installation. After installation, open the download link in your browser and double-click the file, or navigate to the folder where the file is saved and double-click it to run it. Run the software when prompted by the system and follow the window prompts to install the software package.

To verify the installation, open a command prompt (or PowerShell), and enter the following commands:

node -v *//The system should display the Node.js version installed*

npm -v *//The system should display the NPM version installed*

Truffle Installation

Installing Truffle requires that Node.js was successfully installed along with the Node Package Manager (NPM) that comes with it. NPM is the world's largest software registry and is used by open source developers and organizations to share, borrow and manage software packages.

To install Truffle, open a command prompt and enter:

npm install -g truffle

Ganache Installation

Ganache can be installed on a Windows operating system by simply navigating to https://www.trufflesuite.com/ganache/ and clicking on the ***DOWNLOAD (WINDOWS)*** button to download the *.appx* package. Next, run the package and follow the window prompts to install the package.

React Installation

This project makes use of React, React-DOM, and React-router-DOM. I have created a project directory that would be a container for the Transcript application. This folder holds all of the dependencies used throughout this paper. All command line activities are also performed within the project directory.

To install React, React-DOM, and React-router-DOM, open a command prompt and enter the following command:

npm install react react-dom react-router-dom react-bootstrap --save

The packages can also be installed individually as shown:

npm install react --save

npm install react-dom --save

npm install react-router-dom --save

npm install react-bootstrap --save

React-router-dom makes it possible to create multiple page react applications by allowing routing through different components when there is a change in state; e.g. a clicked button or link, a selection etc.

React-bootstrap is a react package that makes it possible to add styling to the application without the need to have a separate style sheet(s). This is an easier and faster way to create user interfaces that are organized and coloured nicely.

Git Bash Installation

Git Bash is a Microsoft Windows program that acts as an emulation layer for the Git command line interface. Bash is an acronym for Bourne Again Shell. Commands on Git Bash are of Linux type which is my most preferred method of interacting with a computer system via the command line interface.

Git Bash can be installed with the procedures given below:

1. Download the Git Bash setup from the official website: https://www.git-scm.com/
2. Download the installer.
3. Run the .exe file and follow the instructions in the installer.
4. Right-click any folder and pick the Git Bash Here option from the context menu to launch Git Bash (right-click menu).

Chai and Mocha test packages Installation

Mocha is a Node.js-based and browser-based JavaScript test framework. Asynchronous testing, test coverage reports, and assertion libraries from any source are all possible with Mocha.

Chai is a BDD (Behaviour Driven Development) / TDD (Test Driven Development) assertion library for NodeJS and the browser that can be delightfully paired with any javascript testing framework.

For the sake of this project I have installed chai-as-promised from the chai library along with chai and mocha. The below command line installs the aforementioned:

npm install mocha chai chai-as-promised --save-dev

The above package can also be installed individually as with React.

SETTING UP THE DEVELOPMENT ENVIRONMENT

With all development dependencies installed, the next step is to set it up for use in the Transcript project. The following paragraphs describe the procedures to setting up the development environment.

Create React App

The first step is to create the react app. This process creates new directories and files essential for building the projects' client side application.

To create the react app, navigate into the project directory via the command prompt and run the following command:

npx create-react-app my-transcript*//creates a react app named my-transcript*

The above command creates a **client** directory within which a number of other directories and files .

Truffle Initialization

The next step is to initialize Truffle by typing ***truffle init*** into the command window (Git Bash in my case) and hitting the Enter/Return key

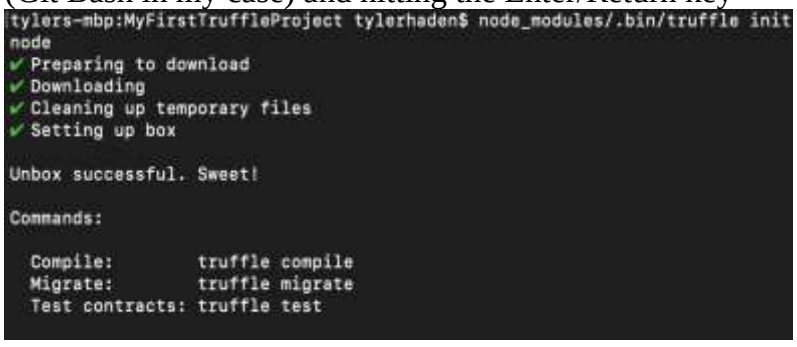

Figure 413: "truffle init" operation as shown in the CLI

NOTE: Truffle initialization is done within the client directory created by the react app. This is necessary to prevent files, directories redundancy that results from initializing npm.

Within the project directory, Truffle Initialization (truffle init) produces three folders (contracts/, migrations/, and test/), as well as three files (Migrations.sol, 1 initial migrations.js, and truffle-config.js). The following is a list of the folders and files created:

- **contracts/** stores all Solidity (.sol) files. Any smart contracts, libraries, or interfaces required at compilation time are included here.
- Migrations.sol is a complete, fully functional Solidity smart contract. Truffle uses it to verify that a project's deployment to the blockchain is completed in the correct order.
- **migrations/** will store truffle "deployer" Javascript files. Every time a contract is deployed, it is important to tell truffle which one, and what constructor arguments is needed.
- **1_initial_migration.js** is the Migration contract's deployment script. Because it does not require library linking or constructor parameters, it is the most basic sort of deployment.
- **test/** Depending on the testing language used, it may contain.js or .sol files. It starts off empty, and the developers must place their test files here.
- **truffle-config.js** is the most important setting for a Truffle project. This is where we specify which networks to utilize, how much gas to use, which addresses to use, and a few other parameters.

Truffle Configuration

The truffle configuration file is truffle-config.js, and it is situated in the project directory's root directory. This is a Javascript file that can run any code required to build a setup. As demonstrated in the screenshot below, it exports an object that represents my project settings.

Figure 5: truffle-config.js file showing the network objects

The Network object (default values given) indicates which networks are accessible for deployment during migrations, as well as transaction settings to use when dealing with each network (such as gas price, from address, etc.). If no transaction settings are supplied for a network, the following values will be used by default:

- **gas:** Gas limit used for deploys. Default is 4712388.
- **gasPrice:** Gas price used for deploys. Default is 100000000000 (100 Shannon).
- **from:** From address used during migrations. The first accessible account given by your Ethereum client is used by default.
- **provider:** Default web3 provider using host and port options: new Web3.providers.HttpProvider("http://<host>:<port>").

The truffle-config.js file can contain multiple network configurations, but in general, only one network can be worked with at a time.

NPM Initializer (npm init)

The project's package.json file is created by the npm init command. Any Node project's package.json file is its beating heart. It keeps track of key metadata about a project that is necessary before it can be published to NPM, as well as functional properties that npm utilizes to install dependencies, run scripts, and identify the package's entry point. It also contains metadata for the project such as version number, author, and description.

Not all fields are included in the bundle. Although json will be used in this project, I may get some significant advantages by capturing information about the program in its package.json. An example of such benefit is a case where I accidentally lose

the node_modules folder for this project. By simply running "npm install" in the command line interface, npm will automatically scan the package.json folder and Install the dependencies with their exact versions from there. The npm init command creates a package.json file for your project's frontend as shown below;

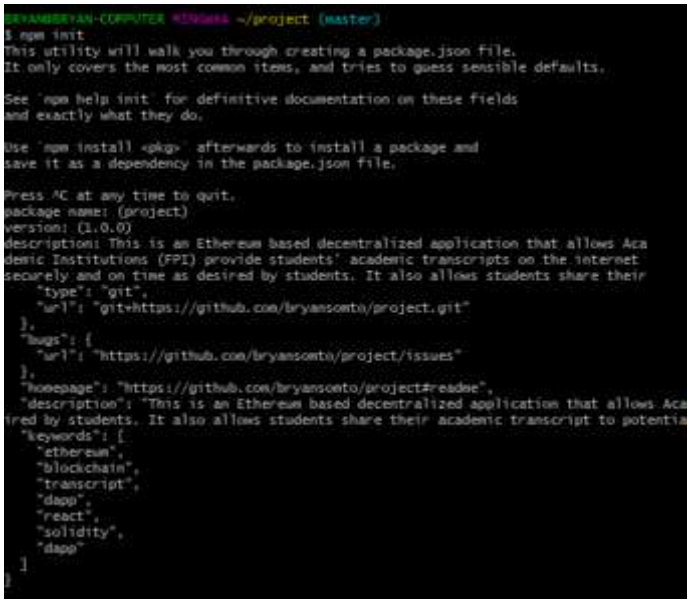

Figure 6: "npm init" operation as shown in the CLI

Setting up Ganache

It is necessary to set up the Ethereum client (Ganache) for this project. Any interaction made with truffle that is concerned with this project will require that Ganache is running. Some of those interactions include: contract migration, contract deployment, contract compilation, and contract testing.

Ganache GUI was set up through the procedures shown below:

- Open Ganache by double-clicking (left) the Ganache icon.
- Under WORKSPACES, left-click NEW WORKSPACE to select.
- Input a desired name in the WORKSPACE NAME column.
- Under the TRUFFLE PROJECTS column, left-click ADD PROJECT. This will open up the file browser window
- Navigate to the Project directory and select the **truffle-config.js** file. Double-click to select (left) or left-click and click open to select.
- The selected file will be seen within the TRUFFLE PROJECTS column.
- On the top right corner click SAVE WORKSPACE to save changes.

NOTE: The NETWORK ID, HOSTNAME and PORT NUMBER can be set under the SERVER tab.

MetaMask Installation

Metamask is a browser-based cryptocurrency wallet that works with Chrome, Firefox, and Brave. It's also available as a browser add-on. This implies it acts as a link between standard web browsers and the Ethereum Blockchain. MetaMask is a software cryptocurrency wallet used to interact with the Ethereum blockchain. It allows users to access their Ethereum wallet through a browser extension or mobile app, which can then be used to interact with decentralized applications. MetaMask can be used on Chrome, Firefox and Brave browsers. The following steps were followed to download the MetaMask extension on Chrome browser.

- Visit https://chrome.google.com/webstore/category/extensions?hl=en-GB on a chrome browser.

NOTE: The link above lands you on the Chrome web store page in the extensions category.

- In the "Search the store" bar, input metamask and hit the Enter/Return key.

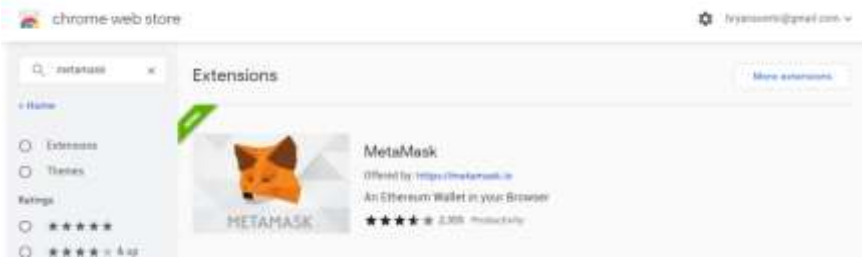

Figure 14: MetaMask Installation

- Left-click to select the app (shown in the figure above).
- Click on "Add to Chrome" to download the browser extension.

MetaMask provides an option to change from the Ethereum Mainnet to the network used by the Ethereum Client (Ganache). By changing to the network used by Ganache (Localhost:8545 as configured for this project) it is possible to import the accounts on Ganache and even make transfers to other accounts on the network.

THE SMART CONTRACT

Smart contracts are contracts that exist across a decentralized, distributed Blockchain network. They are self-executing contracts whose terms of agreement are directly written into lines of codes. This section focuses on the smart contracts' – logic, writing, compilation, migration and deployment.

Writing the Smart Contract

It is necessary to determine the functions required of the MyTranscript application before writing the smart contract. This will make the development process faster, precise and help the developer write functional contracts that meets the clients' needs.

The smart contract for the MyTranscript application is named ***Transcript.sol***. The extension ".sol" tells us it is a solidity file. *Transcript.sol* has the contract called *Transcript* and comprises a few globally and locally declared variables, constructors, structs, events, and functions.

A **constructor** in solidity is a special function declared using the constructor keyword. It is used to initialize state variables of a contract. When you build a contract, the constructor code is only run once.

A **struct** is a container-type structure like arrays, lists etc., however unlike its counterparts, structs allows a user store group of elements with different type.

An **event** in solidity is an inheritable member of a contract. It is emitted, and the parameters are saved in transaction logs on the blockchain, which are accessible via the contract's address while the contract is active.

Function like in every other programming languages are grouped code that are reusable and can be called anywhere in a program. In solidity however, there are four types of functions: *external, internal, public and private.*

- Internal functions may only be used inside the context of the current contract and cannot be used in other situations.
- External functions are invoked from the outside and through transactions as well.
- Private functions are only accessible within the contract in which they are invoked.
- The public functions are called from any place. (BitDegree, 2021)

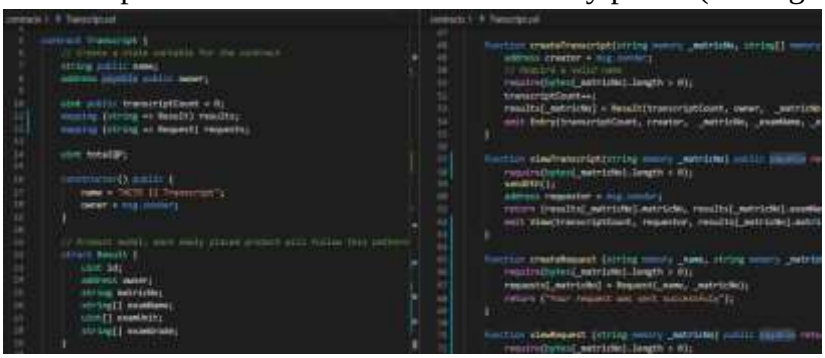

Figure 815: Transcript.sol

The figure above shows some of the smart contract terms discussed under this this section. It is worth noting some functions written in the smart contract as they perform the operations required of the smart contract.

- **createTranscript** function provides the functionality that allows a Lecturer/Examiner create and submit the details for a student academic transcript.
- **viewTranscript** function provides the functionality that enables the lecturer and student view their academic transcript.
- **createRequest** function provides the functionality that allows an Employer make a request to view a student transcript. This request is sent to the student and is only available to the Employer if the concerned student approves the request.
- **viewRequest** function provides the functionality that enables a student view pending transcript requests from potential Employers.

NOTE: Contracts (*Transcript.sol)* in solidity is saved in the *contracts* directory created during truffle initialization (*truffle init*).

Compiling the Smart Contract

After writing *Transcript.sol*, the next step is to compile the contract. Solidity is a high level programming language, hence, the need for compilation. Compilation converts the high level solidity code into bytecode (the language used by the Ethereum Virtual Machine).

Compilation is done via the command line interface (CLI). The following steps were taken to compile the contract:

- In a CLI window, navigate to the project directory
- In the project directory, input ***truffle compile***
- If compilation succeeds, a new *.json* file will be added to the build\contracts directory as shown in the last statement below (*Writing artifacts to .\build\contracts*)

```
BRYAN@BRYAN-COMPUTER MINGW64 ~/project (master)
$ truffle compile
Compiling .\contracts\Transcript.sol...

Compilation warnings encountered:

/C/Users/BRYAN/project/contracts/Transcript.sol:3:1: Warning: Experimental features are turned on.
pragma experimental ABIEncoderV2;
^------------------------------^

Writing artifacts to .\build\contracts
```

Figure 916: Compiling the smart contract

- If compilation fails, an error ("*Compilation failed. See above.")*, and the source of the error is displayed in the CLI.

Testing the Smart Contract

Using the test packages installed earlier (chai, mocha, chai-as-promised) we can test the smart contract. The goal of testing the smart contract is too find gaps, errors, or missing requirements in comparison with the actual requirements. By writing test codes, we can test the solidity functions by passing parameters where necessary and observing the behaviour of the function while making necessary adjustments until they meet the actual requirements.

```
describe('transcript', async () => {

    before(async () => {
        entry = await transcript.createTranscript('h/cte/19/0554', ['CTE 413', 'EEC 312'], [2,3], ['A','B'])
        transcriptCount = await transcript.transcriptCount()
    })

    it ('creates transcript', async () => {
        assert.equal(transcriptCount, 1)
        // console.log(entry.logs)
        let event = entry.logs[0].args
        assert.equal(event.examName[0], 'CTE 413', 'Course code is correct')
    })
```

Figure 170: Test code sample

Figure 16 above shows a snapshot of the *Transcript_test.js* file stored in the test directory of our project folder. The snapshot captures a test for the *createTranscript* function in the smart contract. Parameters such as the student matric number, course codes, course unit and grade are sent as parameters to the functions and tested to ensure it behaves as required. If not, necessary adjustments are made until the user is satisfied.

Migrating the Smart Contract

A migration file is created for each contract written. Migration files are JavaScript files that aid in the deployment of Ethereum contracts. These files are used to stage deployment tasks, and they are written with the assumption that deployment requirements will change over time. As your project grows, you'll need to write fresh migration scripts to keep up with the blockchain's progress. (Truffle, 2021)

truffle migrate //run migrations

NOTE: for migrations to run successfully, the Ethereum client must be running.

DEVELOPING THE CLIENT SIDE APPLICATION (MyTranscript)

This section describes the client application. The client side application or (front-end application) provides the interface the application users (Examiner/Lecturer, Student and Employer) interacts with during usage. Information about all the application package dependencies and the respective versions used can be viewed from the *package.json* file. The following sub-sections describes the client side application logically:

Connecting to the Ethereum Clients' API (MetaMask)

A central application that connects the Blockchain network with the client application is necessary to allow users interact with the smart contract. This central applications makes the functions written in the smart contract accessible via the user interface, it processes payments associated with performing transactions specified in the smart contract, and outputs transaction receipts to determine transactions status.

The function code that provides the connection is written on the client side application with the **web3.js – Ethereum JavaScript API**. Web3.js is a set of libraries that let you use HTTP, IPC, or WebSocket to communicate with a local or distant ethereum node. (web3.js, 2016)

Interacting with the Browser Console

In the process of application development, it is a given the developer will encounter errors no matter how experienced. How to deal with this errors (debug) greatly affects the speed of development and help actualize neat codes.

Debugging for web applications (especially JavaScript) has been made easier thanks to modern browsers with informative and advanced debugging tools. Google Chrome browser is my preferred browser, its debugging console can be

accessed via ***Menu > More tools > Developer tools*** or with the shortcut key ***Ctrl + Shift + I***. React generates its error messages to help developers easily debug errors on the console. However, to observe code behaviours, the element(s) can be logged to the console manually by adding **console.log(*element*)** to the line of code.

As this is a dynamic web application that handles data and performs arithmetic and logic calculations on those data, interaction with the console was heavy. It was necessary to generate pages and values, and observe their behaviours so as to create page functions that meets requirements.

Also, during connections to the Ethereum clients' API (Application Program Interface) it is necessary to ensure all parameters (network, wallet address, smart contract methods, etc.) are observed via the console to ensure its general behaviour meets requirement.

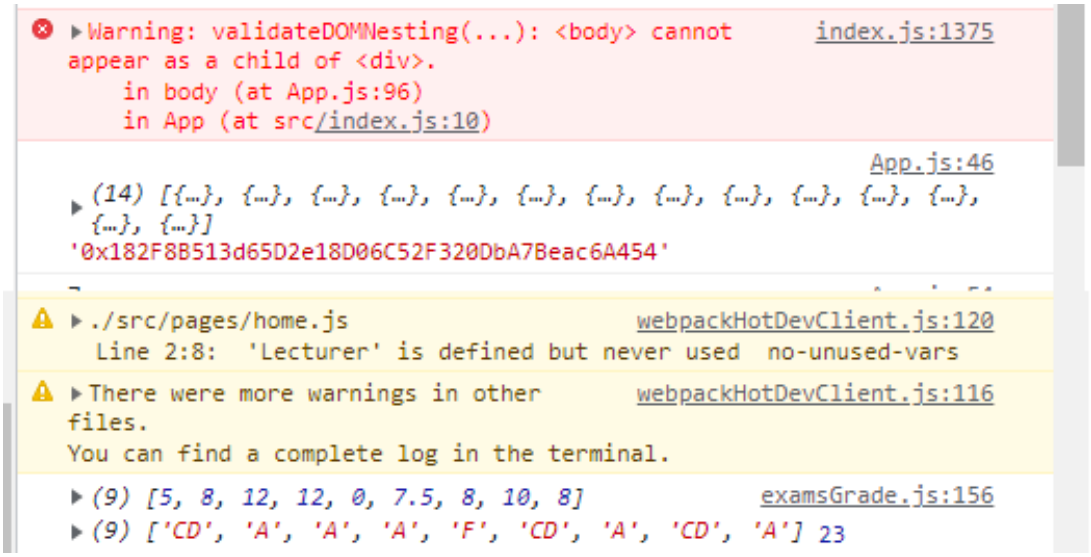

Figure 181: Developer console

Figure 11 shows a snapshot of Chrome browser developer console. Warnings on the console are shown in red and yellow. Warnings in red are fatal and will halt program flow until they are resolved. Warnings in yellow does not affect program flow and can be ignored. The white row (top) shows sets of arrays logged to the console. They can be expanded by left-clicking on them to get a detailed view. *'0x182F…'* shows the account wallet address currently used by the application. The white row (bottom) shows some values (grade points, grade, unit total) being logged to the console to observe its behaviour.

RESULTS AND DISCUSSION

Performance Test

Figure 12 below shows a snapshot of the smart contract functions being tested by the JavaScript test file. Tests were made to ensure; the smart contract deployed successfully, has a name and owner, creates transcripts, views transcripts etc. From the snapshot, tests descriptions can be seen after the green tick (shows a test pass). The *"5 passing..."* at the end, indicates all five tests passed. Other information shown are values logged to the truffle console for further observations.

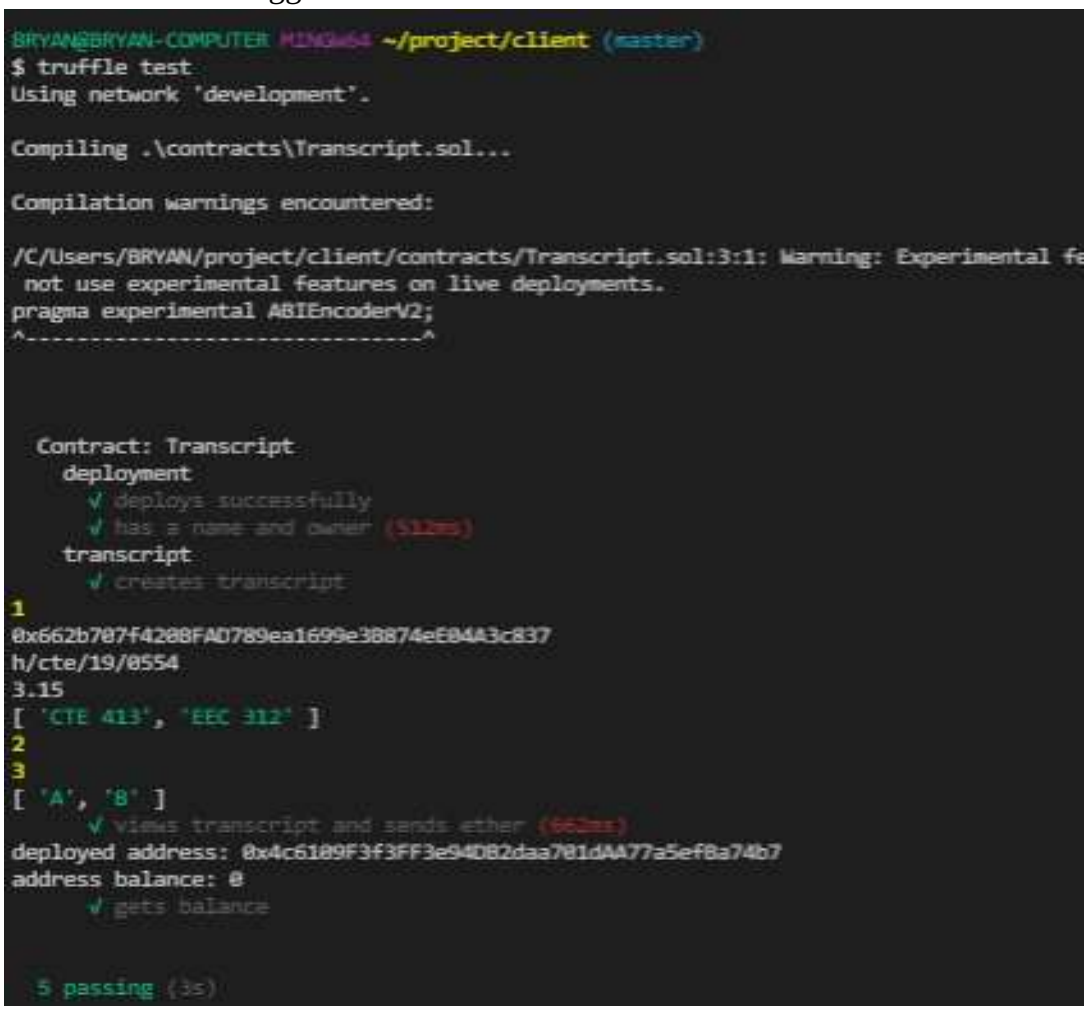

Figure 12: Running the smart contract test

Presentation of Results

The following figures show some important pages and operations result of the *MyTranscript* application:

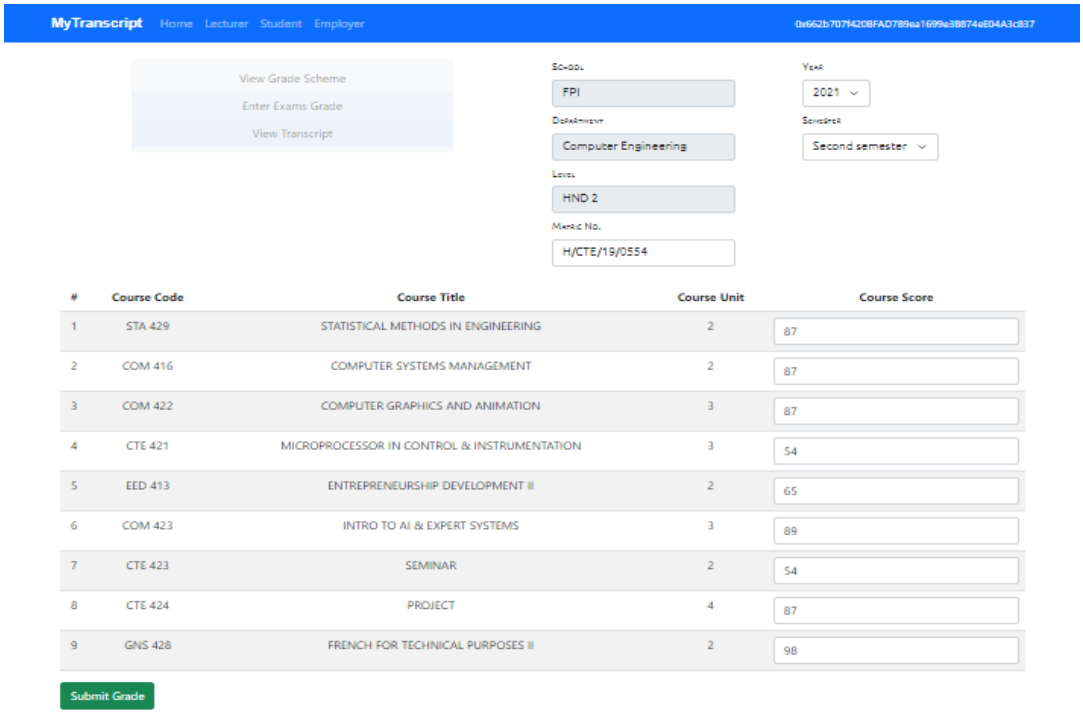

Figure 193: Enter exams grade

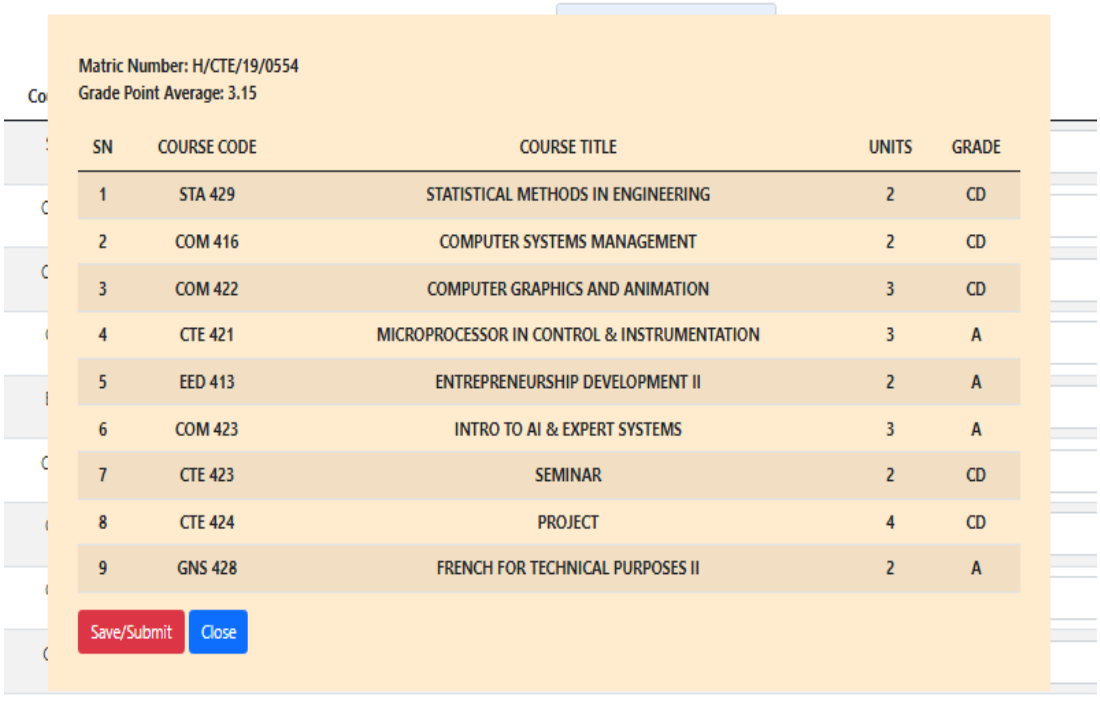

Figure 14: Grades preview

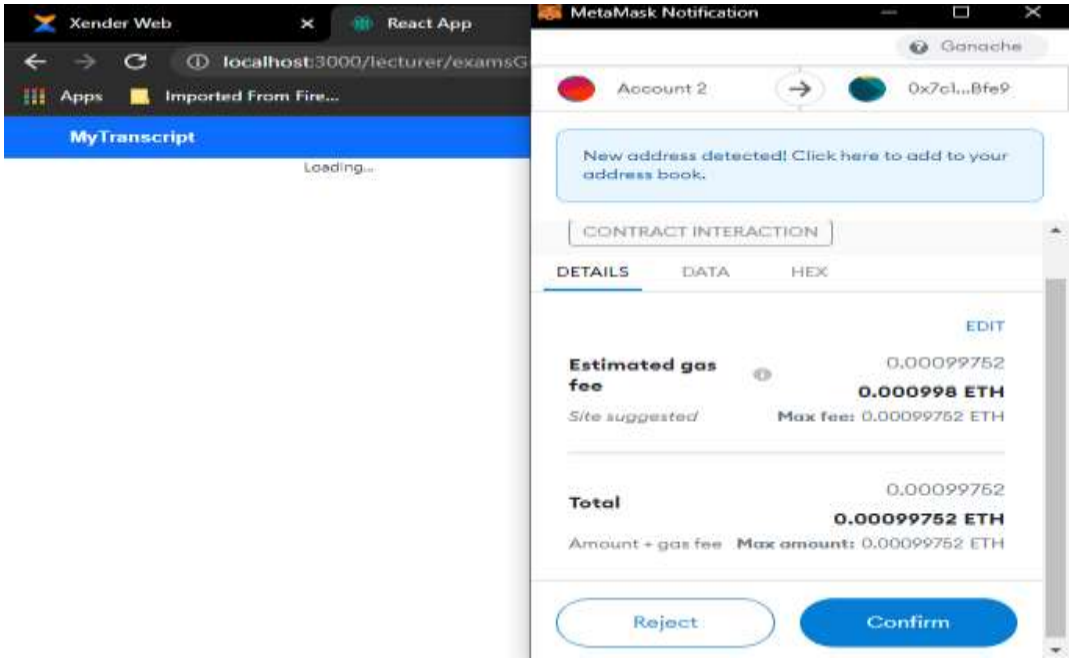

Figure 205: MetaMask transaction notification

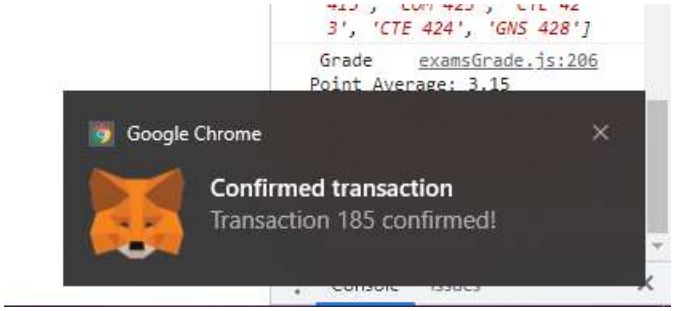

Figure 21: MetaMask popup to confirm transaction success

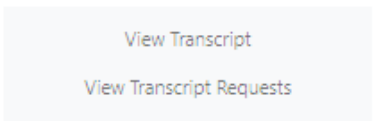

Figure 17: Student page

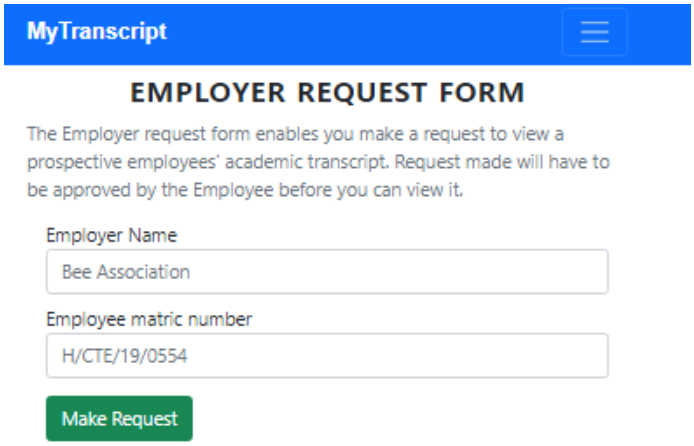

Figure 18: Employer request page

Discussion of Results

The MyTranscript decentralized web application was built using JavaScript and JavaScript Extensible Markup Language (which React is based on) on the client side to perform its logic and arithmetic functions where necessary. The user interface was styled with React Bootstrap CSS (Cascading Style Sheet) library and a few of my own locally written CSS. MyTranscript provides three major navigation options; *Lecturer*, *Student and Employer*.

The ***Lecturer*** page is subdivided into three main categories and provides a Lecturer or an Examiner the option to perform the functions listed below:

- **View Grade Scheme**: this provides the interface to view course information (code, unit, and title) of the two semesters. It is a page that is readily available to its users, hence, providing a way to access course information on the go.
- **Enter Exams Grade**: this provides the interface that enables the user input a students' academic grades for all courses. Like the *view grade scheme*, it provides the option to also switch between semesters and input its associated grades. On submission of inputted grades, a preview page with all entered information is shown to enable the user confirm all information provided were correct. On further submission, an alert box is displayed to inform the user of the Blockchain interaction and requests the users' intervention. On confirmation, the submitted information is processed on MetaMask and saved to the Blockchain.

- **View Transcript:** this allows the user view students' academic transcript previously saved on the Blockchain.

The ***Student*** page is subdivided into two main categories described below:

- **View Transcript:** this provides the functionality to allow students view their academic transcript.
- **View Transcript Request:** this provides the student with the functionality to view all requests made by Employers who have requested to view their academic transcript. This function gives the students privacy control over how their academic records are shared on the internet.

The ***Employer*** page provides the interface to allow Employers seeking to view a student academic record. The Employer fills the request form, and on submission, the student-of-interest can view the request and choose to (or not to) approve the request.

CONCLUSION AND RECOMMENDATION

CONCLUSION

The *MyTranscript* application is a web based decentralised application that employs a considerably large implementation of modern technologies involved in application development on the web. Technologies like NodeJs, Truffle, Ganache and React were the fundamental building block of this project.

In application development, it is good practice and necessary for the developer(s) to subject the application to potential use cases within the application scope in order to observe the application functionality from the end user perspective. For this reason, *MyTranscript* employs the Chai and Mocha test frameworks to ensure its functionalities were tested and behaves as required. The test frameworks employed makes up the application development dependencies.

RECOMMENDATION

The success of a product largely depends on the resources invested in it. The *MyTranscript* application can be made much more powerful by incorporating a centralised database management system to handle creation and storage of user accounts. This will enable institutions, students and employers to create user accounts which are stored on the centralised database and used to sign in to the application. Using the centralised database management system in conjunction with the Blockchain will facilitate detailed user tracking, allow application extensibility (provide an enabling environment for growth), and most importantly

reduce the application interactions with the smart contract which in turn will save a lot of Ether (money) used in processing transactions and save application load time.

REFERENCES

Atomh33Is. (2018). Retrieved from Ethereum Stack Exchange: https://i.stack.imgur.com/YZGxe.png

BitDegree. (2021). *Master Solidity Functions: Modifiers and Overloading Explained*. Retrieved from BitDegree: https://www.bitdegree.org/learn/solidity-functions

Dannen, C. (2017). Introducing Ethreum and Solidity - Foundation of Cryptocurrency and Blockchain Programming for Beginners. Brooklyn, New York: Springer Science + Business Media Finance Inc.

Eludire, A. A. (2011). The Design and Implementation of Student Accademic Record Management System. *Research Journal of Applied Sciences, Engineering and Technology*.

Ethereum Wiki. (2015). Retrieved from Github: https://github.com/ethereum/wiki/wiki/White-Paper

Jambhulkar, S. S., & Ratnaparkhi, V. P. (2020, September). Government fund distribution and tracking system using Blockchain technology.

Lamport, L., Shostak, R., & Pease, M. (1982). *The Byzantine Generals Problem.*

Lemieux, V. (2017). *Blockchain and distributed ledgers as trusted record keeping systems: An archival theoreotic evaluation framework.* University of Columbia.

Maryville University. (2021). *How Blockchain is used in Education*. Retrieved from Maryville University: https://online.maryville.edu/blog/blockchain-in-education

Master Solidity Functions: Modifiers and Overloading Explained. (2021). Retrieved from BitDegree: https://www.bitdegree.org/learn/solidity-functions.

Nguyen, T. (2018). *GRADUBIQUE: An Academic Transcript Database Using Blockchain Architecture.* doi:https://doi.org/10.31979/etd.42nu-nsnp

Omogbhemhe, M., & Akpojare, J. (2018). *Development of Centralised Transcript Processing System.* College of Applied Sciences, Adeboyega University, Mathematical and Physical Sciences.

Osagie, A. U., & Mallam, A. (2014). Students Record Analysis and Examination Result Computation Algorithm (SRAERCA). *International Journal of Technology Enhancements and Emerging Engineering Research*.

Redman, J. (2016, September 1). *We've Hit Peak Blockchain Hype, Says New Report.* Retrieved from https://news.Bitcoin.com/Blockchain-hype-peak-newreport/

Truffle. (2021). *Running Migrations.* Retrieved from Truffle: https://trufflesuite.com/docs/truffle/getting-started/running-migrations.html.

Web3.js. (2016). *Ethereum Revision.* Retrieved from Web3.js: https://www.web3js.readthedocs.io/en/v1.5.2/

Wood, G. (2015). *Ethereum: A secure decentralised generalised transaction ledger* (EIP-150 Revision ed.).