Decentralised application incentivising daily steps count

Koh, Zhuang Chean

2022

Koh, Z. C. (2022). Decentralised application incentivising daily steps count. Final Year Project (FYP), Nanyang Technological University, Singapore. https://hdl.handle.net/10356/156624

https://hdl.handle.net/10356/156624

CZ4079 – Final Year Project

<u>Decentralised Application Incentivising Daily Steps Count</u>

Project ID: **SCSE21-0505**

Submitted By: **Koh Zhuang Chean**

Matriculation Number: **U1820520A**

Supervisor: **Dr Sourav Sen Gupta**

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Abstract

With the advancement of blockchain technology and its unique meanwhile promising characteristics in every aspect of a good software system, a plethora of business use-cases have shifted their designs towards a more decentralised manner and paved the way for inclusive decentralisation through innovative incentives and smart contracts. However, the scalability of the blockchain system remains a prominent issue in the community. In other words, there is still very little coverage of blockchain on the diverse array of day-to-day applications due to the difficulty scaling the system. This acts as a concern to the blockchain technology as there are still an enormous number of individuals who do not even know what blockchain technology is able to bring for us.

On the other hand, despite the relentless campaigns held across the globe to raise the health awareness, there are still reluctant people choosing not to care about it. One of the reasons being no immediate rewards whatsoever in consuming more time to act in maintaining a healthy lifestyle.

Hence, the aim of this final year project is to leverage the ease of developing a decentralised application (DApp) that Ethereum community has to offer to develop a DApp that incentivises daily steps to individuals in the form of ERC20 compatible token which can then be used to purchase in-game Non-Fungible Tokens (NFTs). The DApp is meant to promote health awareness by having more daily steps and now individuals like you and me can be rewarded by your daily steps.

Acknowledgement

I would like to express my sincere gratitude to all parties that have helped me learn and understand the concept of an Ethereum stack from Ethereum states to smart contracts, above all, to Dr Sourav Sen Gupta, for the relentless guidance as well as putting altogether my unstructured thoughts.

Besides, I would like to thank those who made CZ4153 – Blockchain module possible, including Ms Gweneth, Mr Alex and Mr Anton for leading us to the right path to mitigate blockchain distribution overheads. All those helps have paved the way for me to deal with the obstacles along the way and made the learning curve for this final year project less steep.

Acronyms

DApp	Decentralised Application
NFT	Non-Fungible Token
UI/UX	User Interfaces/User Experience
ERC	Ethereum request for comment
MERN	MongoDB, Express, React, Node
REST	Representational State Transfer
API	Application Programming Interface
EVM	Ethereum Virtual Machine
AJAX	Asynchronous JavaScript and XML
IDE	Integrated Development Environment
M2E	Move to Earn
EOA	Externally Owned Account
CRUD	Create, Read, Update, Delete
MVC	Model, View, Controller
1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal Form
NPM	Node Package Manager
ODM	Object Data Modelling
VDOM	Virtual Document Object Model
CLI	Command Line Interface

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1. Background and Overview

As decentralised applications have started blending in our daily lives at an extremely quick rate that we might not even notice, be it decentralised finance (DeFi), games, non-fungible tokens (NFTs) or some other kinds of decentralised derivatives, we are exposed with more decentralised applications (DApps) day by day [1]. This has created an opportunity for software engineers to have a change at their specialties or even better, integrating their specialties with the mighty blockchains.

Studies have shown that the DApps have caught the attention of Gaming and they are booming in the gaming world at a remarkable rate. Gamers from all over the world are intrigued by the act of owning their in-game items and/or collectibles in the game regardless of whether their accounts are hacked or stolen, provided that they have access to their private keys, which are used as proof that they own the items [2]. Furthermore, blockchain in gaming enables the projection of value on intangible assets and hence motivates those who find gaming is a waste of time to really dedicate themselves to playing those games [3]. As a matter of fact, there are already games out there that are solely built from Ethereum and/or Ethereum-based cryptocurrency with NFTs as its collectibles, such as Alien Worlds, Splinterland, just to name a few. A NFT is a unit of data stored on a blockchain, which certifies a digital asset to be unique and therefore not interchangeable and immutable. NFTs can be used to represent assets such as photos, videos, audio, and other types of digital files [4].

1.2. Aim

Despite the advantages of gaming in blockchain technology, the limitation of blockchain's functionality has narrowed down the types of the game that can be based solely on blockchain technology. As such, the gaming world in blockchain needs some out-of-the-box ideas in gaming to cater to all types of gamers. Therefore, the aim of this Final Year Project is to develop a Ethereum-based Mobile DApp in a game format that rewards people StepTokens, which is ERC-20 compatible, from having more daily steps which can be used to claim the StepTokens.

1.3. Objectives

The main objective of the project is to develop a game which promotes exercising in general, incentivizing people to jog more or travel to nearer places walking by letting them claim their daily steps as StepTokens which in turn could be used to purchase in-game NFTs and have fun in the games.

The nice-to-have objective of the project is to study and utilize a gas-optimized method to distribute the StepTokens to users so that the users can receive the StepTokens without having to pay the costly gas price on Ethereum Blockchain. Do note that live testnet such as Kovan and/or local blockchain such as Ganache will be used to simulate the real-life blockchain scenarios.

1.4. Scope

The scope of this project is to develop a full-stack simple mobile game using the MongoDB, Express, React, Node (MERN) stack. The mobile game will be integrating with Ethereum smart contracts written in Solidity to tokenize the in-game assets such as StepTokens and equipment as NFTs.

The scope of the project encompasses:

1.4.1. Frontend Development

Developing intuitive user interface/user experience (UI/UX) for users to claim rewards, view rewards, level up their NFTs as well as auction their NFTs. IOS environment will be the main focus here due to the time constraints and hence only iOS ipa file will be built and distributed to the test devices.

1.4.2. Backend Development

Developing the backend system hosted on Heroku that is responsible of providing Representational State Transfer Application Interface (REST API) endpoints for other components to exchange messages. It includes integrating with Ethereum Blockchain and Frontend.

1.4.3. Ethereum Blockchain

Developing and deploying smart contracts written in Solidity that will be executed on the Ethereum Virtual Machine (EVM). Ganache and Truffle are used as local blockchain environment for testing purposes. The fully tested smart contracts are deployed to Kovan live testnet to resemble real-world blockchain.

1.4.4. Integration Interfaces

Ether JS is used to handle the communications between the Backend and the Ethereum Blockchain. Whereas, Mongoose is used to handle the communications between the Backend and the MongoDB. Lastly, Axios, a JavaScript library which uses AJAX, is used to handle the communications between Backend and Frontend.

1.5. Market Competitor

There is an application on market called StepN. It is a Web3 lifestyle app with inbuilt Game-Fi and Social-Fi elements that is the first project to effectively bring to life a functioning Move and Earn (M2E) concept [5].

1.5.1. Disclaimer

Before starting off, I would like to clarify that ideas of my final year project were rightfully mine. The abovementioned application was discovered by accident upon near completion of this project. It turned out that the objectives and scope of this project were almost exactly the same as StepN, which is simply just M2E.

1.5.2. Comparison

Applications **Fields**	**Final Year Project**	**StepN**
Blockchain	Ethereum	Solana
Game Token	ERC20 Step Token (SPT)	SPL Green Satoshi Token (GST)
Governance Token	None	Green Metaverse Token (GMT)
NFT	Game characters with associated level	Sneakers, Shoebox, Gems
Earning type	Move-to-earn	Move-to-earn
Environment	Anywhere	Outdoors with GPS
GPS tracking	No	Yes
Decentralised Wallet	In-game Wallet	In-game and multi-chain wallets
Marketplace	In-app blind auction page	In-app Marketplace
Voting	No	Yes, using GMT

Table 1: Comparison of FYP and STEPN

1.5.3. Designs of STEPN

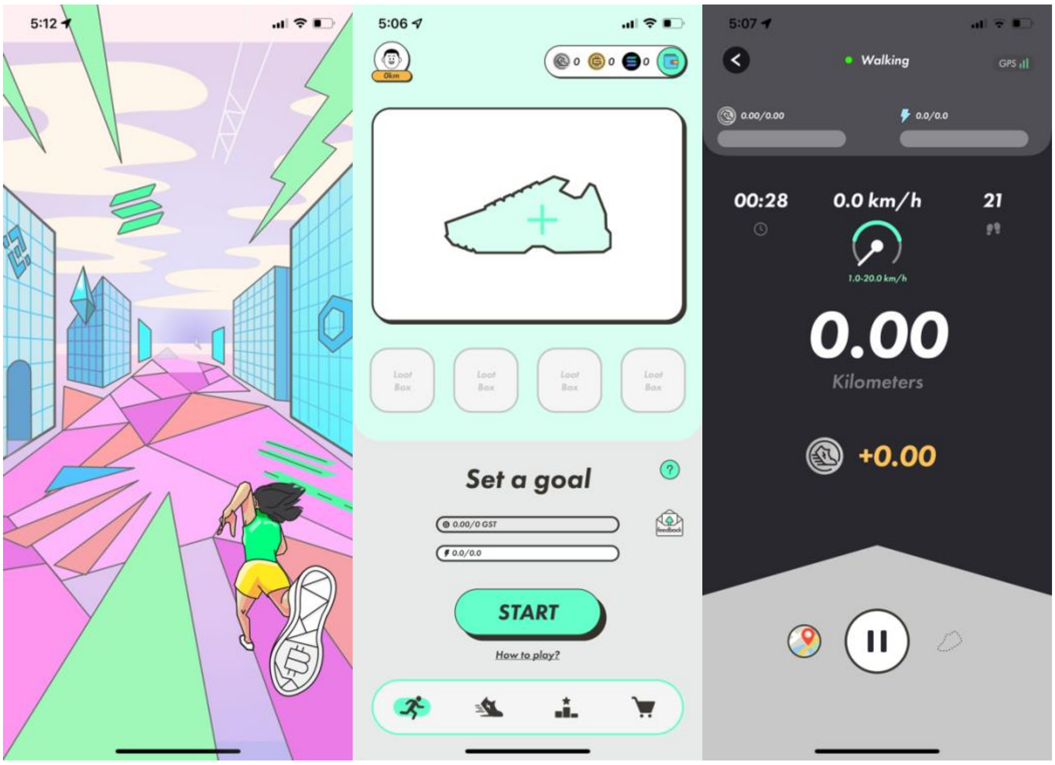
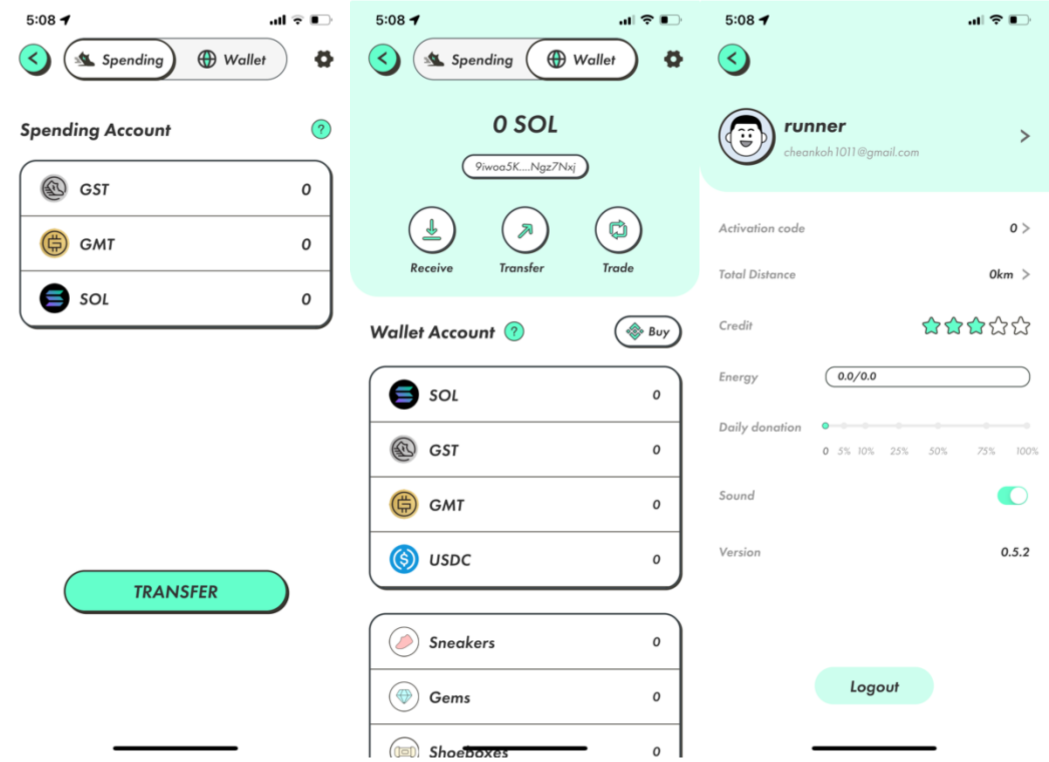


Figure 1: Loading, Home and Start pages of STEPN



Spending Account

GST	0
GMT	0
SOL	0

0 SOL

9iwoa5K....Ngz7Nxj

Wallet Account

SOL	0
GST	0
GMT	0
USDC	0

Sneakers	0
Gems	0
Shoeboxes	0

runner

cheankoh1011@gmail.com

Activation code	0
Total Distance	0km
Credit	★★★☆☆
Energy	0.0/0.0
Daily donation	0 5% 10% 25% 50% 75% 100%
Sound	●
Version	0.5.2

Figure 2: Spending, Wallet, Setting pages of STEPN

1.5.4. Conclusion and Insights

The idea of a M2E is a feasible token economics (tokenomics) to attract users by referencing the existing example, STEPN. It can be downloaded on App Store or Google Play Store but an activation code needs to be obtained through the means of joining the community on renown social medias like telegram, discord, just to name a few.

It is indeed an intriguing experience to realise that there are also people working on the same ideas around the same period as I am. According to their timeline, they participated Solana Hackathon and got 4th in rank and started out their journey around Aug 2021, which was the same time that I commenced my final year project, too.

1.6. Software Environments

Different platforms and tools required to develop the project are listed here.

Software	Description
Visual Studio Code	Visual Studio Code serves as a lightweight but powerful source code editor or sometimes is referred to as integrated development environment (IDE). It comes with built-in support for a lot of languages including the ones that are required for this project, namely Javascript, Typescript, Node.js, React-native and Solidity.
Git	Git is a free and distributed version control system that helps to coordinate work among the different developers even though in this case it will just be developed by myself. The reason it is still used is because it can track changes in the project and provide rollback to certain version with ease.

Github	Github serves as the provider of internet hosting for software development. It complements Git by providing an easy-to-learn user interface which eases the management of the software projects and provide hosting for the code repository.
Truffle	Truffle is a world class development environment, testing framework and asset pipeline for blockchains using the EVM aiming to make life as a developer easier in the Ethereum ecosystem.
Ganache	Ganache is sometimes called as a One Click Blockchain. It is a personal blockchain for Ethereum development you can use to deploy contracts, develop your applications, and run tests on local computers. Ganache is available for Windows, Mac, and Linux.
MongoDB	MongoDB serves as a NoSQL database program which works very well with Node.js applications.
Express.js	Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications such as simple re-routing of APIs’ urls.
React-Native	React Native allows the creation of truly native apps and does not compromise users' experiences. It provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform’s native UI building blocks.

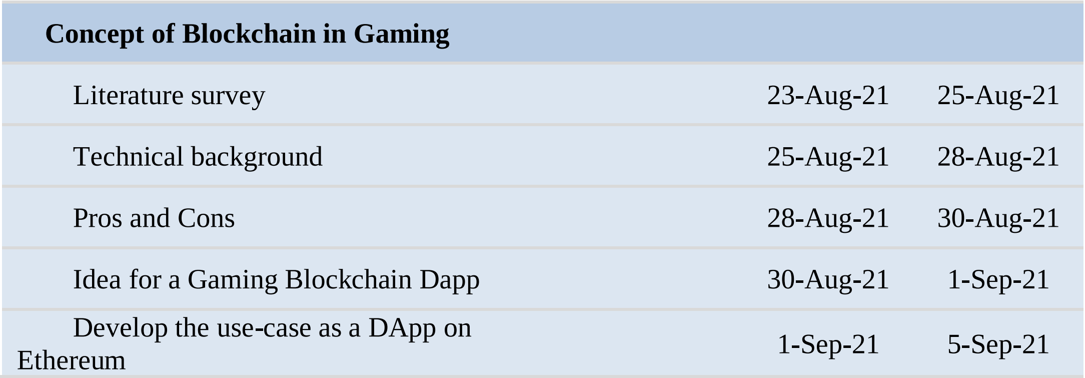
Node.js	Node.js serves as an asynchronous event-driven JavaScript runtime which is designed to build scalable network applications.
Expo	Expo serves as a framework and a platform for universal React applications. It offers a set of tools and services for React Native that help in developing, building, deploying, and quickly iterating on iOS, Android, and web apps from the same codebase.
Xcode	Xcode is a must have IDE for building iOS applications. It is used to handle native compilation and build fat binary files that are in Mach-O executable format.
IOS simulator	IOS simulator can run any iOS device for development purposes.
Visual Paradigm	Visual Paradigm provides ease of drawing UML diagrams that are needed for software development such as software architecture diagrams, Entity Relation diagrams and whatnot.
Procreate	Procreate is a raster graphics editor app for digital painting. It simplifies the management for UI designs. It is often used for prototyping and serving as a central location for the UI design in larger projects.
Postman	Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can

	create better APIs—faster. It is used to validate the backend APIs and predict their behaviours under different circumstances.
Infura	Infura is globally distributed, a cloud-hosted node network which helps connecting to Ethereum blockchain very quickly and easily.
Metamask	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, in this case, this project.
Microsoft Teams	Microsoft teams is an online communication platform which provides features such as video conferencing that assists teams in remote discussions. This is the main platform used for the stand-up meetings.

Table 2: Platforms and Tools

1.7. Project Schedule

The initial project schedule is shown in *Table 1* while the revised project schedule is shown in *Table 2*.



Concept of Blockchain in Gaming		
Literature survey	23-Aug-21	25-Aug-21
Technical background	25-Aug-21	28-Aug-21
Pros and Cons	28-Aug-21	30-Aug-21
Idea for a Gaming Blockchain Dapp	30-Aug-21	1-Sep-21
Develop the use-case as a DApp on Ethereum	1-Sep-21	5-Sep-21

Design of the Entire Ecosystem for a PoC		
Configuration - On Truffle using Ganache	5-Sep-21	12-Sep-21
Connecting with Flutter - Web3dart	12-Sep-21	19-Sep-21
Smart Contract - Step Count to Step Coin	19-Sep-21	3-Oct-21
Smart Contract - Step Coin to Game Coin	3-Oct-21	17-Oct-21
Development of the Dapp		
Design of the main Gaming System	17-Oct-21	23-Jan-22
Integration with Smart Contract	27-Nov-21	26-Dec-21
Resolving dependencies for iOS ipa	16-Jan-22	20-Jan-22
Future Applications and Integrations		
Determine if building plugin is possible	20-Jan-22	23-Jan-22
Market survey and look for possible partnerships	23-Jan-22	6-Feb-22
Design iOS plugin (if possible)	1-Feb-22	22-Feb-22
Design Android plugin (if possible)	22-Feb-22	13-Mar-22
Deployment of plugins (if possible)	13-Mar-22	20-Mar-22
Introspection on current Dapp	20-Jan-22	27-Jan-22
Tuning existing DApp based on Introspection	27-Jan-22	20-Mar-22

Table 3: Initial Project Schedule

Concept of Blockchain in Gaming		
Literature survey	23-Aug-21	25-Aug-21
Technical background	25-Aug-21	28-Aug-21
Pros and Cons	28-Aug-21	30-Aug-21
Idea for a Gaming Blockchain Dapp	30-Aug-21	1-Sep-21
Develop the use-case as a DApp on Ethereum	1-Sep-21	5-Sep-21
Design of the Entire Ecosystem for a PoC		
Configuration - On Truffle using Ganache	5-Sep-21	19-Sep-21
Connecting with React-native - Etherjs	19-Sep-21	27-Sep-21
Smart Contract - Step Count to Step Tokens	19-Sep-21	17-Oct-21

Smart Contract - Step Token to NFTs	17-Oct-21	31-Oct-21
Development of the Dapp		
Design of the main Gaming System	1-Nov-21	16-Mar-22
Integration with Smart Contract	5-Jan-22	5-Mar-22
Build and Distribute ipa to test devices	16-Mar-22	20-Mar-22
Final Year Project Report		
Final report preparation and submission	1-Mar-22	21-Mar-22
Amended final report preparation and submission	23-Mar-22	15-Apr-22
Oral presentation preparation	1-May-22	6-May-22

Table 4: Revised Project Schedule

2. System Design & Implementation

This chapter elaborates the design and implementation corresponding the abovementioned objectives and scope of the project.

2.1. System Architecture

Like most of the DApps out on the market, the DApp here also comprises four main components, namely Ethereum blockchain, Backend, Frontend and Database. This project is a full stack project started out from scratch and hence the design will be split into four main sections explaining the implementations for each of the components.

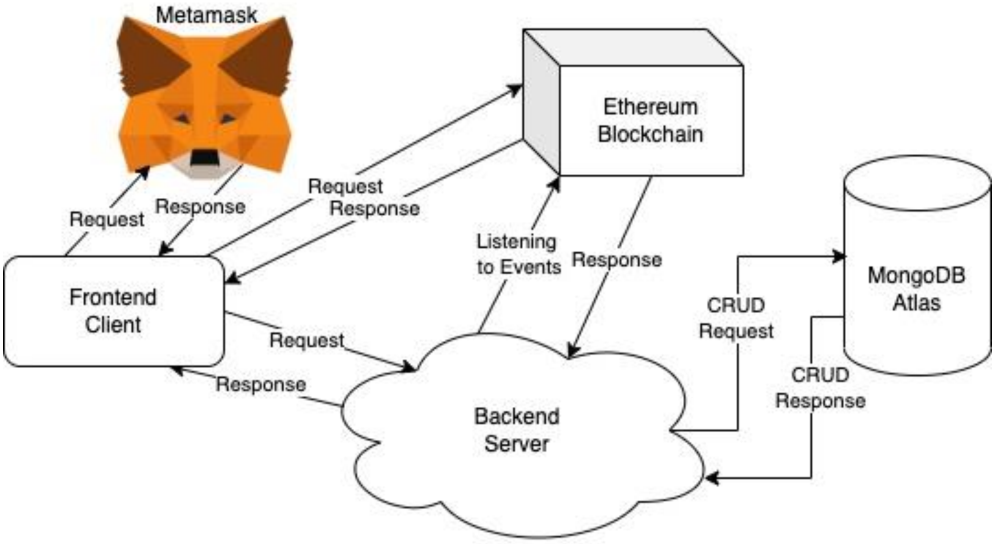


Figure 3: System Architecture

The figure above depicts how the five components interact with one another. In the architecture, backend server is the backbone like any centralised system out on the market. The jobs of the backend server here consists of processing system logics, serving as APIs endpoint for frontend requests and handling Create, Read, Update, Delete (CRUD) operations with MongoDB. However, in the world of decentralised systems, the job of the backend is slightly different. Frontend clients must directly sign their transaction without going pass backend and invoke the smart contracts on their own. This enables a fully decentralised mechanism whereby the backend listens to the events from the Ethereum blockchain should there be any events emitted by the Ethereum blockchain from the clients invocations.

On the other hand, the frontend clients and the MongoDB are only able to communicate with each other via backend server. All the communications must go through backend server for some processing.

The backend communicates with the Ethereum blockchain by connecting to an archive node such as Infura node, using Ether Js as an interface which allows the invocation codes to be written in JavaScript.

Moreover, the backend server exchanges messages with the frontend clients through the means of using REST API. The node.js server, which is hosted on Heroku, is always active and listening for incoming requests from frontend clients.

Apart from that, Metamask, a cryptocurrency wallet used to interact with the Ethereum blockchain is used to ensure the Externally Owned Account (EOA) is well managed and provide ease of signing the clients' transactions.

Finally, a cloud-based database is used, and it is called the MongoDB. An Atlas of the MongoDB is created for the backend server to connect to. The node.js server uses Mongoose library to connect to the atlas through a connection string generated by MongoDB.

2.1.1. Software Design Pattern – Backend

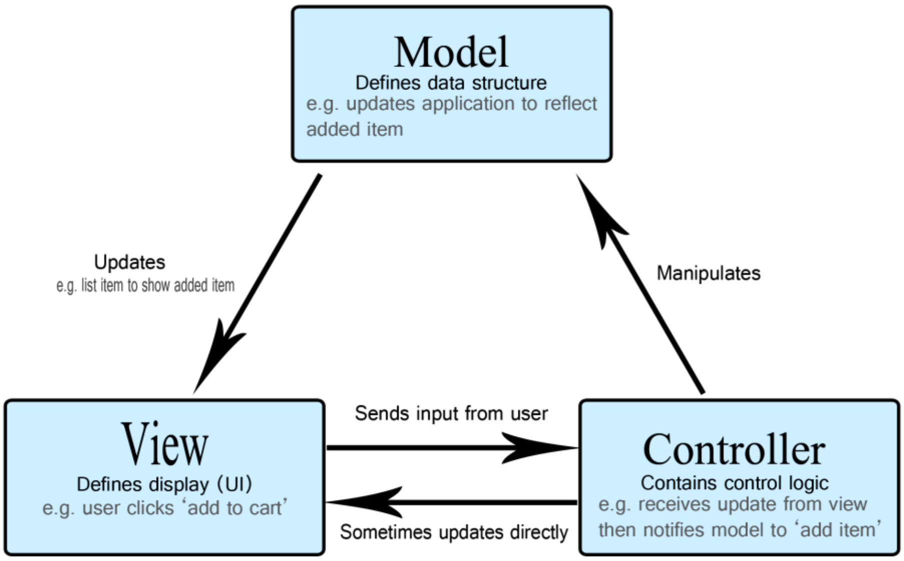

Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Figure 4: Model View Controller (MVC) Architecture

The figure above depicts the architecture that the project's backend codes follow throughout and will be mentioned below for quite several times. It is a design pattern that is used to separate business logic and presentation details.

It is one of the most commonly used design patterns for simple business solutions.

The Model here is the Entity of the Database which will be discussed under Database section, the Controller is the backend logic of the application in which each controller file manages its corresponding View and manipulates its corresponding Model(s), whereas the View here is responsible for routing the urls or endpoints of the REST API that backend provides. It is considered as a web services MVC pattern in which the View here is not the actual UI/UX interfaces of the application.

2.1.2. Software Design Pattern – Frontend

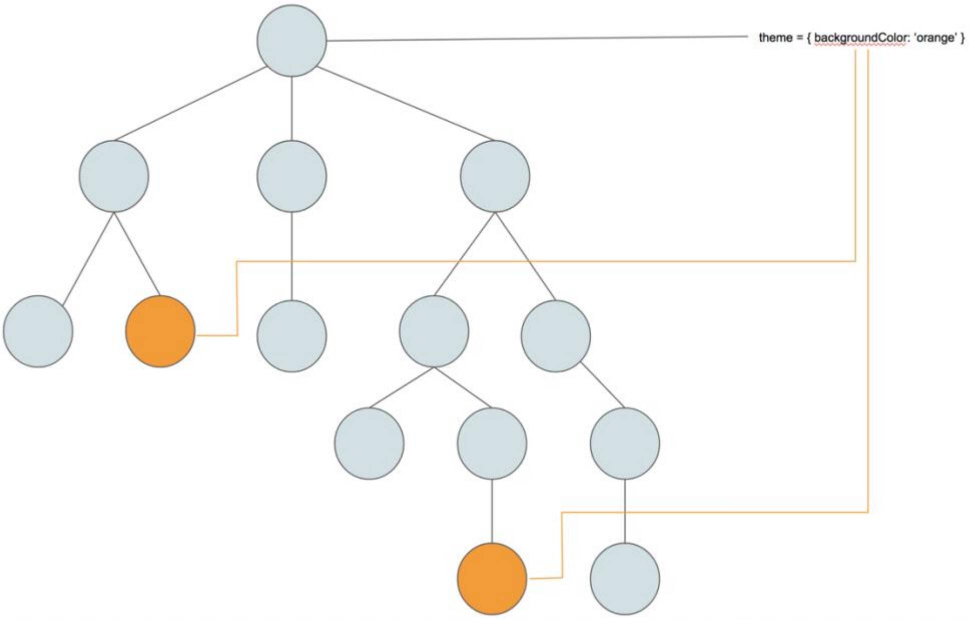


Figure 5: Provider Pattern in React

The figure above illustrates the provider pattern in React, which is the frontend library used in this project. The reason that this design pattern was chosen is because of a major problem that is faced by React developers, which is the prop drilling. It is when data (prop) is passed down to different components until it gets to the component where the data (prop) is needed, more and more unrelated components share the data that is passed down which brings unnecessary overheads. The provider pattern enables storing data in a central location, the React Context object or the Redux store. With the use of Context Provider, data can be passed to any component that is wrapped under the component without the need of drilling data (prop) from top all the way until the component.

2.1.3. Use Case Diagram

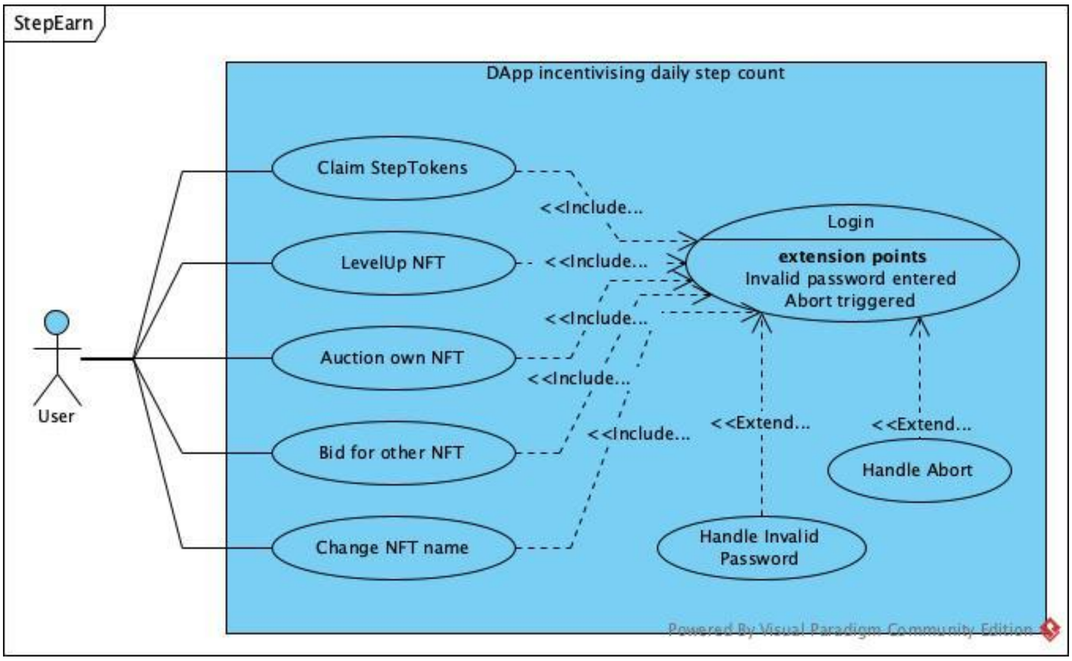


Figure 6: Use Case Diagram

The figure above shows six use cases including login that user can perform on the application. The detailed explanations on each use case are as below:

Use Case	Descriptions
Claim StepTokens	After the user successfully login, use can press the claim token button in order to exchange their steps on the Health App on their mobile devices to exchange their steps to the in-app StepTokens.
LevelUp NFT	After the user successfully login, an amount needed to level their NFT is shown and if the user has enough amount of StepTokens, user can burn/spend

	the equivalent amount of StepTokens and level up his/her NFTs.
Auction Own NFT	After the user successfully login, user who wants to auction his/her NFT can input the price of the NFT and place it onto the auction page so that other users are able to see it on the auction listing page.
Bid for other NFT	After the user successfully login, user can browse the auction listing page and opt for an NFT that he/she is interested in and then bid for the NFT. Upon termination of the auction, winners will spend the amount that he/she bids for and win over the NFT, and losers will be returned the amount of StepTokens that they bid for.
Change NFT Name	After the user successfully login, if the level of NFT of the user is above 2, he/she can rename the NFT

Table 5: Use Case Description

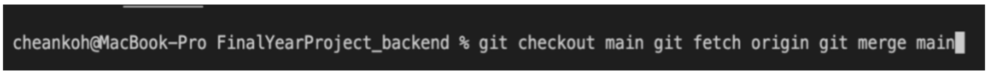
2.1.4. Git Workflow

Despite working alone throughout the whole project, good practices still can be abided by, especially for a graduating software engineer. In most of the software applications, there are more than one individual are involved and hence using a standardised workflow is imperative to ensure synchronisation throughout the whole project among several developers contributing to the same project. Git and Github are used to provide just that as well as a user-friendly interface to manage the repositories of the software applications.

The whole backend and database related files were written under the github repository of FinalYearProject_backend at https://github.com/cheankoh/FinalYearProject_backend. Whereas the whole frontend as well as the Solidity smart contracts reside in the directory of FinalYearProject at https://github.com/cheankoh/FinalYearProject.

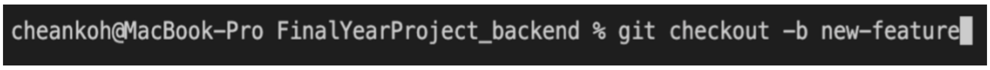
A simple git workflow is followed throughout the whole software development process. The step-by-step instructions are as below:

1. **Start off with pulling the latest version from main branch and overrides your current version.**



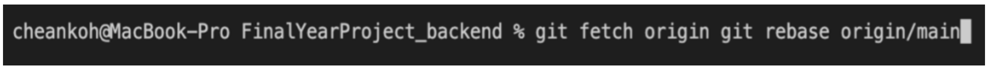
```
cheankoh@MacBook-Pro FinalYearProject_backend % git checkout main git fetch origin git merge main
```

2. **Branch off to work on new feature without changing the main.**



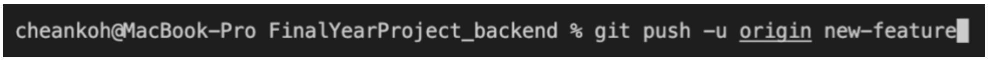
```
cheankoh@MacBook-Pro FinalYearProject_backend % git checkout -b new-feature
```

3. **Work on the feature.**
4. **Keep your branch updated if there are people merging to the main branch**



```
cheankoh@MacBook-Pro FinalYearProject_backend % git fetch origin git rebase origin/main
```

5. **Upon completion, push the branch remotely and request a pull request**



```
cheankoh@MacBook-Pro FinalYearProject_backend % git push -u origin new-feature
```

2.2. Database

Due to the ease of use and a huge community that can be leveraged should there be any potential errors and bugs that others have already encountered, the “M” under the MERN stack, MongoDB was chosen to be the database of this project.

2.2.1. Entity Relationship Diagram

For this project, five entities were identified and normalized. Users entity which stores the data of the users, Claims entity which stores the records of the claims from steps on the Health App to the StepTokens, Auctions entity which stores the required data for an auction created by users, Bids entity which stores the bidding details of the bids and lastly NonFungibleTokens entity which stores simple credentials of the users' NFTs, are the five entities required for this project.

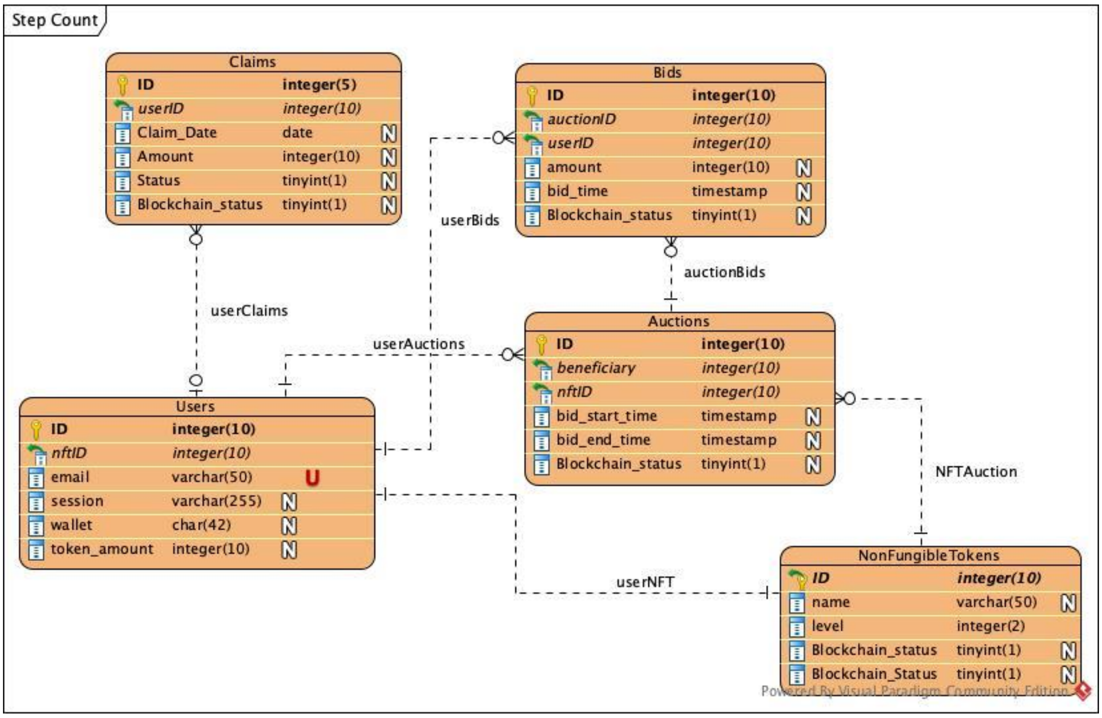


Figure 7: Entity Relationships Diagram

Figure 3 depicts the entities as well as their relationships with one another. The explanations of the relationships are as below:

1. **userClaims**: each user can have zero or more claims throughout the lifetime using the application. Hence, it forms a one-to-many relationship.
2. **userNFT**: the application is limiting the user to have only one NFT which is the character with its associated name and level. In case an auction of that NFT is successfully bidden by a user, a newly created NFT with specified name and level of one will be created and assigned to the beneficiary of the auction. Hence, it forms a one-to-one relationship of Users and NonFungibleTokens.

3. **userAuctions**: each user can have zero or more auctions throughout the lifetime of using the application. Hence, it forms a one-to-one relationship of Users and Auctions.
4. **userBids**: each user can have zero or more bids throughout the lifetime of using the application. Hence, it forms a one-to-one relationship of Users and Bids.
5. **auctionBids**: for each auction that has been placed by a user, it can have zero or more bids before the auction has ended. Hence it forms a one-to-many relationship with Auctions and Bids.
6. **NFTAuction**: each NFT can be auctioned to other users zero or more times. Hence, it forms a one-to-many relationship with NonFungibleTokens and Auctions.

2.2.2. Normalisation techniques

To design a robust database with less redundancy and meanwhile ensuring zero loss of data, **Third Normal Form (3NF)** normalisation technique is used to design the schemas.

To normalise the database schemas into **3NF**, the schemas must be in **1NF** and **2NF** as well.

Firstly, to normalise the schemas into **1NF**, make sure that every column in the schemas must be single valued. Furthermore, the data stored under the same column must be of the same kind or type. This can be known using "Common Sense".

Once the schemas are in **1NF**, it is time to normalize them into **2NF**. To normalise the schemas into **2NF** is just simply make sure there is **no partial dependencies**. In other words, it means that no column can be derived from a non-primary key column. In case there is a **partial dependency**, remove the column from the schema and place it under a schema that will not cause any partial dependency.

Ultimately when all the schemas are in **2NF**, it is time to normalize them into **3NF**. To simply put, there should not be any non-trivial dependency that is dependent on a non-prime column or attribute that is not part of the primary key. This kind of dependency is called **transitive dependency**. In case there is a **transitive dependency**, take out the non-prime column and the one that it is dependent on from the schema and put them into a separate table, and use an id for the non-prime column whenever necessary.

2.2.3. Indexes

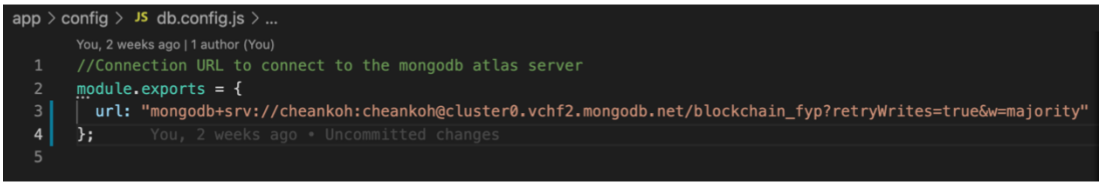
Indexes of a schema support efficient execution of queries in MongoDB. Without indexes, MongoDB must perform a collection scan in which it iteratively goes through every document in a collection to retrieve those which match the query statement. MongoDB uses a **b-tree data structure** for storing its indexes by default. **B-tree data structure indexes** allow efficient retrieval of documents for range-based query operations.

Creating indexes for a certain field under a schema is a very common thing to do to speed up the query process. Fortunately, with the help of MongoDB built-in collection's function, indexes can be simply created by invoking the *collection.createIndex()* function. Therefore, every collection corresponding to each entity has its own *unique index* that is created by MongoDB by default to ensure the uniqueness of the values in the _id field.

For this project, only single field index was created.

2.2.4. Mongoose

Mongoose was used in this project throughout to lessen the hassle of writing MongoDB validation, casting and business logic boilerplate.


app > config > JS db.config.js > ...
You, 2 weeks ago | 1 author (You)

```
//Connection URL to connect to the mongodb atlas server
module.exports = {
  url: "mongodb+srv://cheankoh:cheankoh@cluster0.vchf2.mongodb.net/blockchain_fyp?retryWrites=true&w=majority"
};       You, 2 weeks ago • Uncommitted changes
```

Figure 8: Codes – Mongoose Connection String

The above figure shows the configuration file of the project. It is just using the *module.export* to export the object containing the whole database connection and entities.

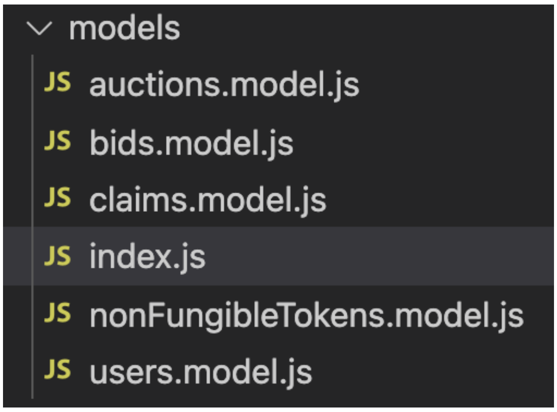
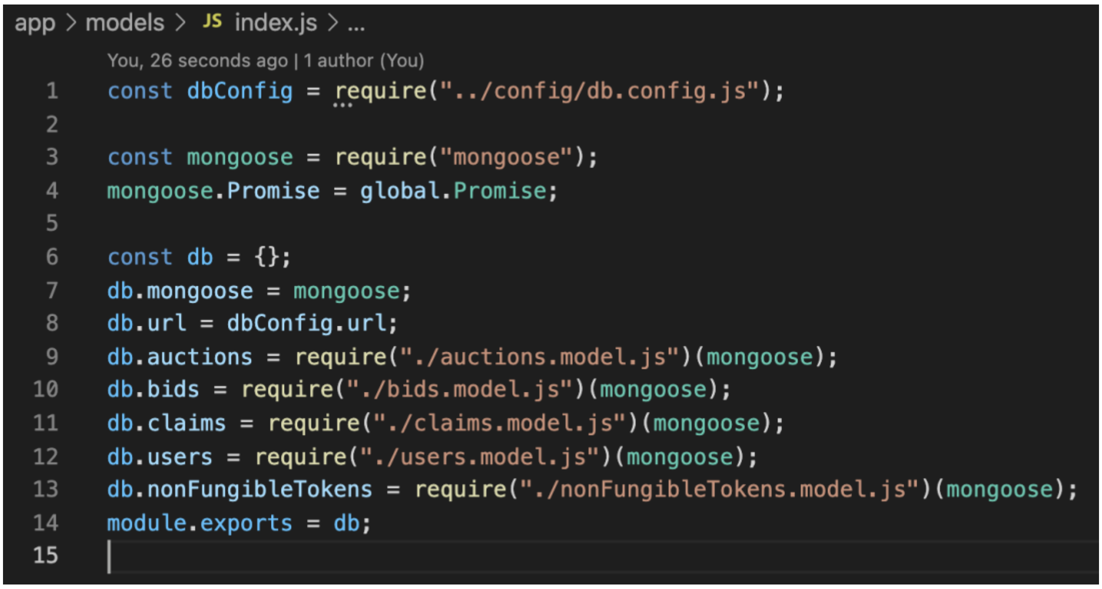


Figure 9: Files of Entities (Models)

The above figure shows the files each of which represents an entity on *Figure 4* except the index.js. The architecture that is used is the MVC architecture. As the name suggests, every model or entity file depending on which naming is preferred, corresponds to each entity in *Figure 4*.



app > models > JS index.js > ...
You, 26 seconds ago | 1 author (You)

```
const dbConfig = require("../config/db.config.js");

const mongoose = require("mongoose");
mongoose.Promise = global.Promise;

const db = {};
db.mongoose = mongoose;
db.url = dbConfig.url;
db.auctions = require("./auctions.model.js")(mongoose);
db.bids = require("./bids.model.js")(mongoose);
db.claims = require("./claims.model.js")(mongoose);
db.users = require("./users.model.js")(mongoose);
db.nonFungibleTokens = require("./nonFungibleTokens.model.js")(mongoose);
module.exports = db;

```

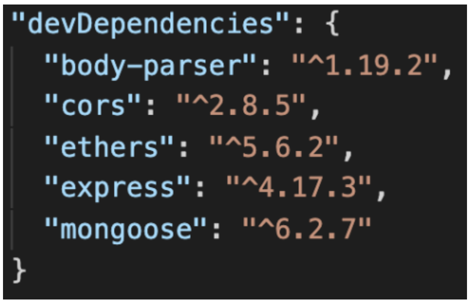
Figure 10: Codes – index.js of Models

The figure above shows the codes of the entry point of the directory app/models/, which is the *index.js* file. It is responsible of importing the configuration file, using it to connect to the MongoDB using mongoose, as well as importing each model as the field of the 'db' object and eventually export the database object, which will then be accessed by the controllers.

2.3. Backend Server

Backend server usually does and processes what the users cannot see from an application and what is really happening under the hood. The backend of this project uses Node.js, a JavaScript runtime environment that runs on the V8 JavaScript engine and executes JavaScript code outside web browsers. It is best known for its event-driven architecture meanwhile being capable of asynchronous I/O. Therefore, being single-threaded does not make Node.js incapable of efficient performance and hence was chosen to be the backend of this project.

2.3.1. Versions and Dependencies



```
"devDependencies": {
  "body-parser": "^1.19.2",
  "cors": "^2.8.5",
  "ethers": "^5.6.2",
  "express": "^4.17.3",
  "mongoose": "^6.2.7"
}
```

Figure 11: Codes – Versions of Dependencies

The above figure shows the development dependencies under the backend repository of this project. It is under the package.json and can be installed via Node Package Manager (NPM).

Dependency	Usage
body-parser	It is a Node.js body parsing middleware which means it is responsible for parsing the incoming

	requests in this project such as requests of application/json content type.
cors	It is the acronym for Cross-Origin Resource Sharing. It is used to allow or restrict requested resources on the backend server depending on where the HTTP request was initiated.
ethers	Also known as Ethers.js. It is a library used for interacting with the Ethereum Blockchain and its ecosystem. It is used by both frontend and backend in this project to invoke the smart contracts that were deployed onto the Kovan testnet.
express	A web application framework used in this project. Its main usage in this project is to reroute the urls.
mongoose	It is an Object Data Modeling (ODM) library for MongoDB distributed as a npm package. It was mentioned under Section 2.2.4..

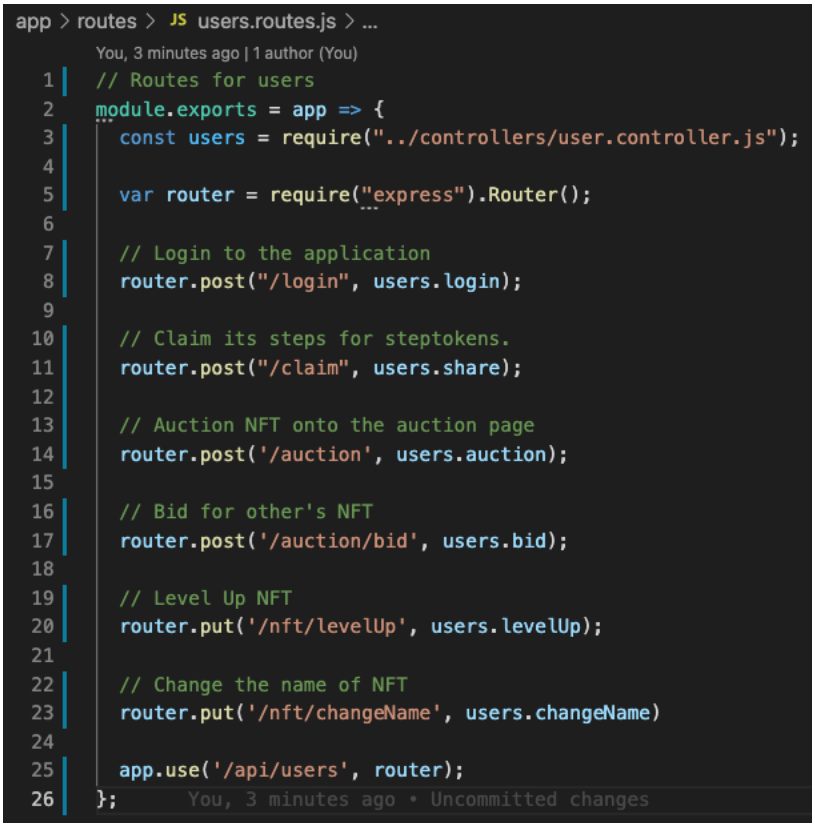
Table 6: Dependencies and their usages

2.3.2. Routing

It is of utmost importance to separate endpoints into descriptive urls for the frontend clients to interact with. One of the best practices is to have API specification document to ensure the url paths, parameters, responses and etc are in sync among the frontend and the backend developers.

The routing here refers to how the application respond to a client request at a particular endpoint with the specified HTTP request method namely GET, POST, PUT and DELTE.

Express.js is used to handle those routing and rerouting of the endpoints in this project. Even though Express.js comes with a set of very handy features, only the express.Router() function was used in this project.


app > routes > JS users.routes.js > ...
You, 3 minutes ago | 1 author (You)
You, 3 minutes ago • Uncommitted changes

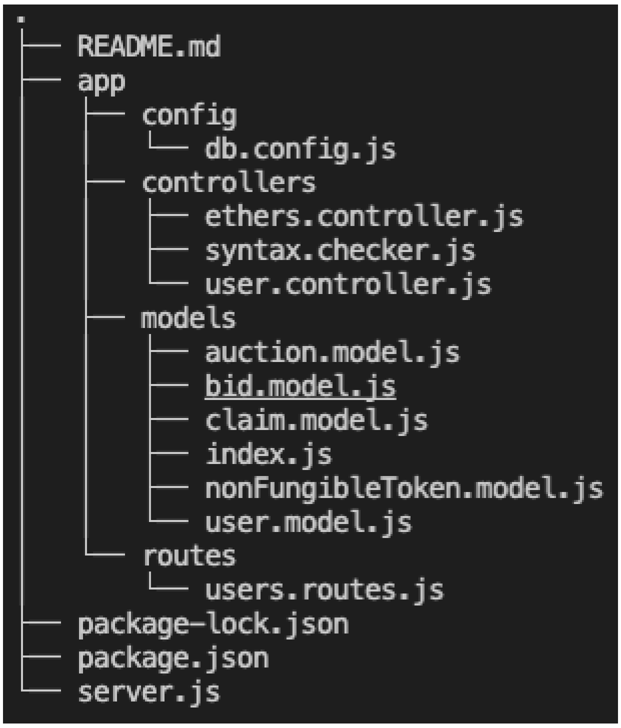
```
1  // Routes for users
2  module.exports = app => {
3    const users = require("../controllers/user.controller.js");
4
5    var router = require("express").Router();
6
7    // Login to the application
8    router.post("/login", users.login);
9
10   // Claim its steps for steptokens.
11   router.post("/claim", users.share);
12
13   // Auction NFT onto the auction page
14   router.post('/auction', users.auction);
15
16   // Bid for other's NFT
17   router.post('/auction/bid', users.bid);
18
19   // Level Up NFT
20   router.put('/nft/levelUp', users.levelUp);
21
22   // Change the name of NFT
23   router.put('/nft/changeName', users.changeName)
24
25   app.use('/api/users', router);
26 };
```

Figure 12: Codes – Routes for user-related features

The above figure shows the codes for routing the user-related functions into descriptive and intuitive endpoints relative to this project’s url that is hosting on Heroku. In line 3, the code for importing the user controller from the directory of controllers is shown. In line 5, the express.Router() is used for routing the endpoints. In line 25, it shows that the “app”, which is an instance of the express constructor that would be passed in when using the file *users.routes.js*, is appending the */api/users* to the url of this project which is at *https://aqueous-stream-78468.herokuapp.com*. Therefore, the APIs related to users’ features in this project would be something similar as *https://aqueous-stream-78468.herokuapp.com/api/users/nameOfFeatures*.

2.3.3. Implementations

As mentioned in *Section 2.1.1.*, the design pattern of the backend server followed the web services MVC pattern.



```
.
├── README.md
├── app
│   ├── config
│   │   └── db.config.js
│   ├── controllers
│   │   ├── ethers.controller.js
│   │   ├── syntax.checker.js
│   │   └── user.controller.js
│   ├── models
│   │   ├── auction.model.js
│   │   ├── bid.model.js
│   │   ├── claim.model.js
│   │   ├── index.js
│   │   ├── nonFungibleToken.model.js
│   │   └── user.model.js
│   └── routes
│       └── users.routes.js
├── package-lock.json
├── package.json
└── server.js
```

Figure 13: Backend Structure (To be updated)

The above figure depicts the directory structure of the backend server's code in a tree format. The server starts by running the *server.js* as shown in the last line of the tree structure. All the codes are under the directory *app* and there are *config*, *controllers*, *models* and *routes* directories. The *config* here is just to store the connection string to MongoDB using mongoose as mentioned in Section 2.2.4.. Whereas the rest, *controllers*, *models* and *routes* correspond to the Controllers, Models and View of the MVC architecture.

3. **Controllers**

The controllers are what process the application logics and are responsible for coordinating between Models and the Routes, i.e., when Routes receive requests to store something in the database, it needs to go through the controllers in order access the Models. This helps in separation of concerns as well as maintenance of codebase in the long run.

Controller	Associated Functions
user.controller.js	Login: login and gain access to the application.
	Claim: claim steps for StepTokens.
	Auction: auction his/her NFT.
	Bid: bid for NFT that is being auctioned
	Level Up: level up his/her NFT
	Change Name: change the name of NFT
ethers.controller.js	Claim: invoke the smart contract to mint the amount that the user claims to him/her
auction.controller.js	Refund: refund to all the bidders who did not win
	Auction End: transfer the token from the smart contract to the beneficiary and the ownership of NFT from beneficiary to the highest bidder

Table 7: Controllers and their respective functions

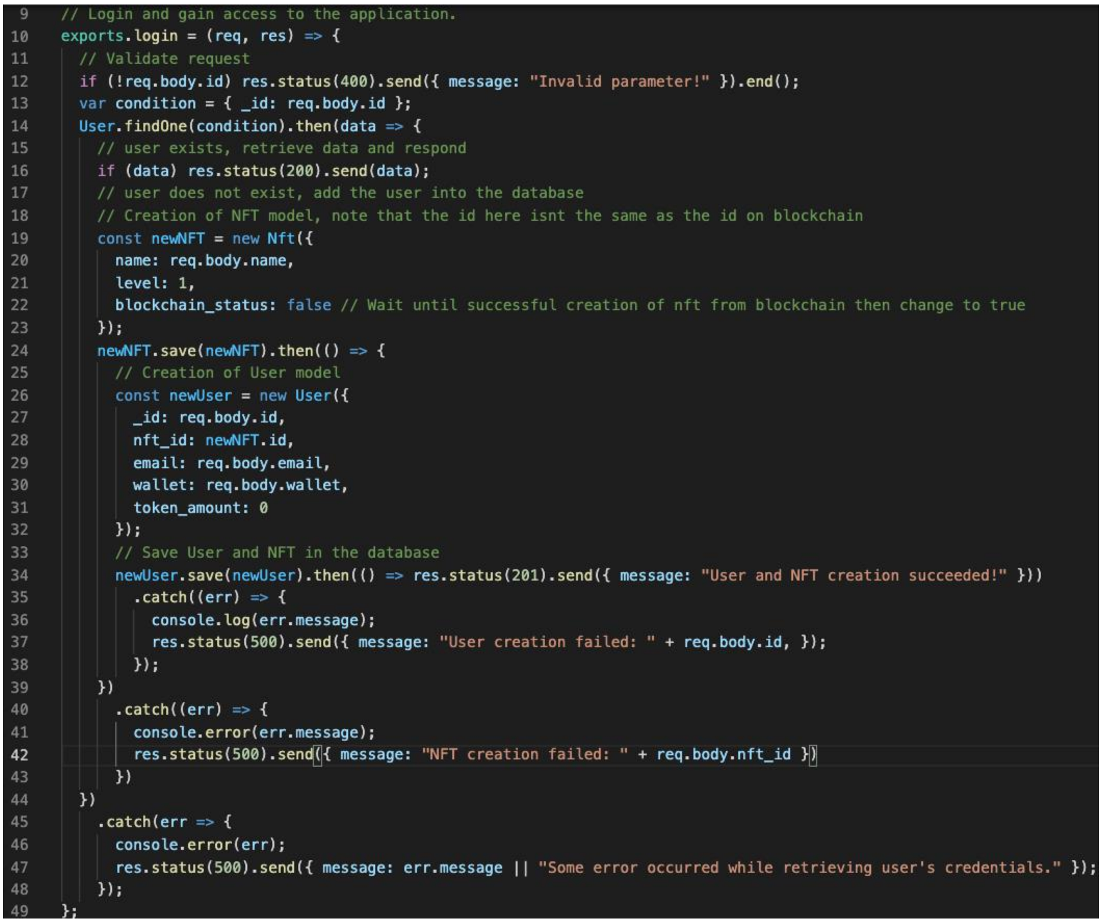
The above table shows the controllers that the backend has and their respective functions' names. The ethers.controller.js only has one function which is to invoke the smart contract to claim the steps for StepTokens. The reason being all the other invocations of the smart contracts are done directly from frontend to the Ethereum Blockchain in order to achieve the decentralisation in this project.

Some examples of the controllers and their respective functions are as below:

**Note that only a few of the codes are shown here as examples.*

a. *users.controller.js*:

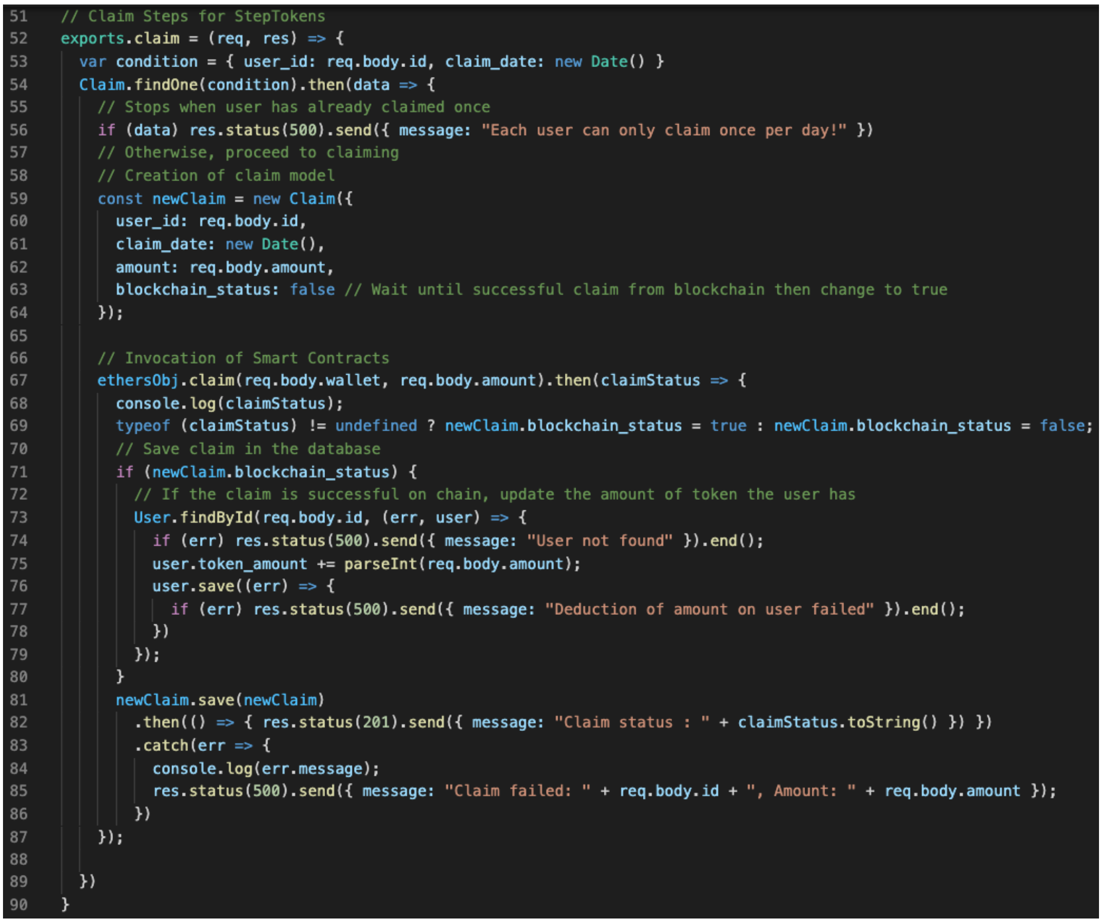
    i. Login: login and gain access to the application.



```javascript
// Login and gain access to the application.
exports.login = (req, res) => {
  // Validate request
  if (!req.body.id) res.status(400).send({ message: "Invalid parameter!" }).end();
  var condition = { _id: req.body.id };
  User.findOne(condition).then(data => {
    // user exists, retrieve data and respond
    if (data) res.status(200).send(data);
    // user does not exist, add the user into the database
    // Creation of NFT model, note that the id here isnt the same as the id on blockchain
    const newNFT = new Nft({
      name: req.body.name,
      level: 1,
      blockchain_status: false // Wait until successful creation of nft from blockchain then change to true
    });
    newNFT.save(newNFT).then(() => {
      // Creation of User model
      const newUser = new User({
        _id: req.body.id,
        nft_id: newNFT.id,
        email: req.body.email,
        wallet: req.body.wallet,
        token_amount: 0
      });
      // Save User and NFT in the database
      newUser.save(newUser).then(() => res.status(201).send({ message: "User and NFT creation succeeded!" }))
        .catch((err) => {
          console.log(err.message);
          res.status(500).send({ message: "User creation failed: " + req.body.id, });
        });
    })
      .catch((err) => {
        console.error(err.message);
        res.status(500).send({ message: "NFT creation failed: " + req.body.nft_id })
      })
  })
    .catch(err => {
      console.error(err);
      res.status(500).send({ message: err.message || "Some error occurred while retrieving user's credentials." });
    });
};
```

Figure 14: Codes – users.login()

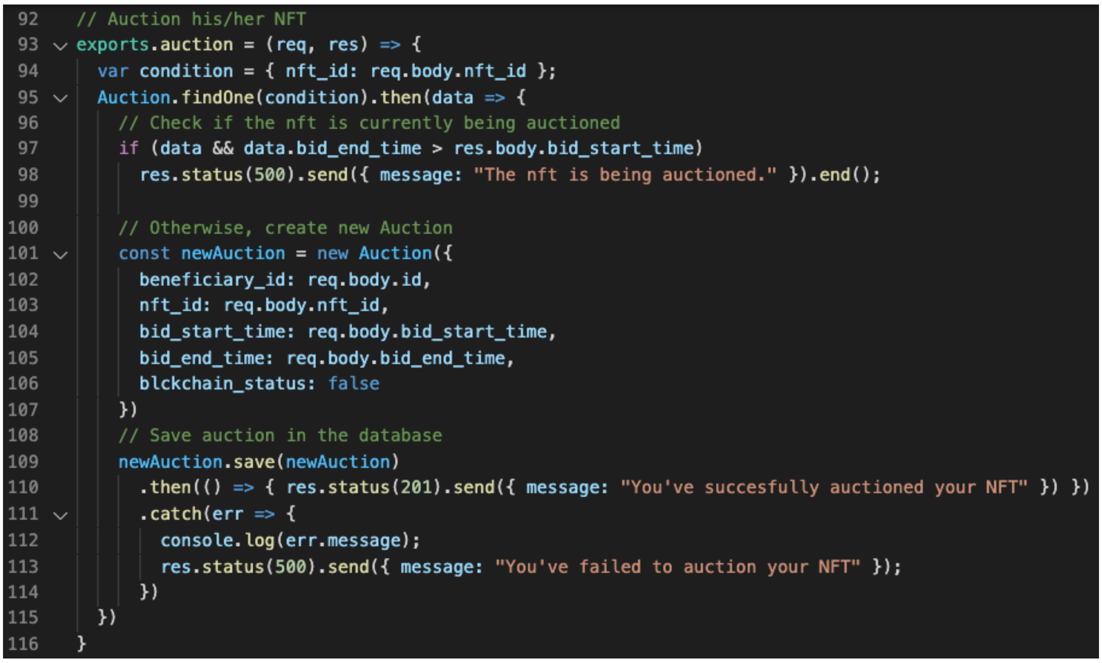
    ii. Claim: claim steps for StepTokens.



```
// Claim Steps for StepTokens
exports.claim = (req, res) => {
  var condition = { user_id: req.body.id, claim_date: new Date() }
  Claim.findOne(condition).then(data => {
    // Stops when user has already claimed once
    if (data) res.status(500).send({ message: "Each user can only claim once per day!" })
    // Otherwise, proceed to claiming
    // Creation of claim model
    const newClaim = new Claim({
      user_id: req.body.id,
      claim_date: new Date(),
      amount: req.body.amount,
      blockchain_status: false // Wait until successful claim from blockchain then change to true
    });

    // Invocation of Smart Contracts
    ethersObj.claim(req.body.wallet, req.body.amount).then(claimStatus => {
      console.log(claimStatus);
      typeof (claimStatus) != undefined ? newClaim.blockchain_status = true : newClaim.blockchain_status = false;
      // Save claim in the database
      if (newClaim.blockchain_status) {
        // If the claim is successful on chain, update the amount of token the user has
        User.findById(req.body.id, (err, user) => {
          if (err) res.status(500).send({ message: "User not found" }).end();
          user.token_amount += parseInt(req.body.amount);
          user.save((err) => {
            if (err) res.status(500).send({ message: "Deduction of amount on user failed" }).end();
          })
        });
      }
      newClaim.save(newClaim)
        .then(() => { res.status(201).send({ message: "Claim status : " + claimStatus.toString() }) })
        .catch(err => {
          console.log(err.message);
          res.status(500).send({ message: "Claim failed: " + req.body.id + ", Amount: " + req.body.amount });
        })
    });

  })
}
```

Figure 15: Codes – users.claim()

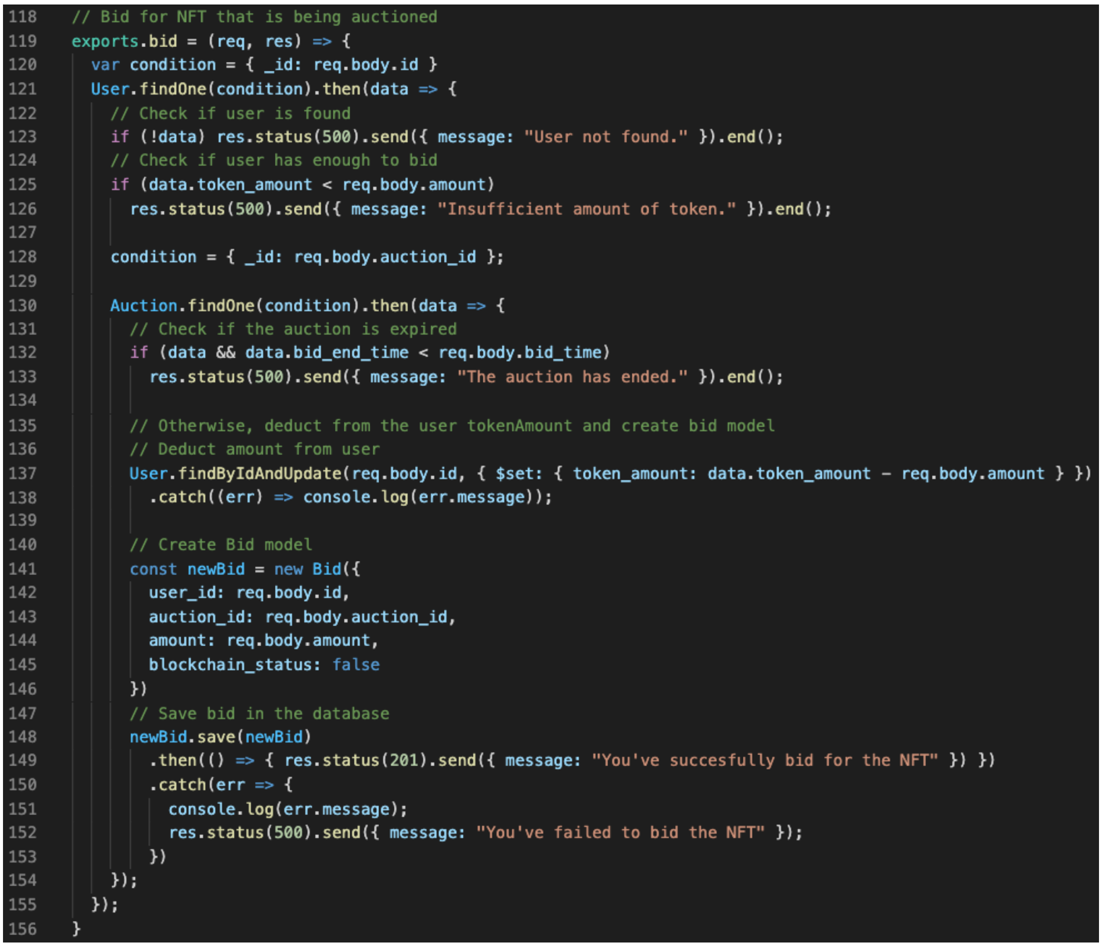
iii. Auction: auction his/her NFT



```
// Auction his/her NFT
exports.auction = (req, res) => {
  var condition = { nft_id: req.body.nft_id };
  Auction.findOne(condition).then(data => {
    // Check if the nft is currently being auctioned
    if (data && data.bid_end_time > res.body.bid_start_time)
      res.status(500).send({ message: "The nft is being auctioned." }).end();

    // Otherwise, create new Auction
    const newAuction = new Auction({
      beneficiary_id: req.body.id,
      nft_id: req.body.nft_id,
      bid_start_time: req.body.bid_start_time,
      bid_end_time: req.body.bid_end_time,
      blckchain_status: false
    })
    // Save auction in the database
    newAuction.save(newAuction)
      .then(() => { res.status(201).send({ message: "You've succesfully auctioned your NFT" }) })
      .catch(err => {
        console.log(err.message);
        res.status(500).send({ message: "You've failed to auction your NFT" });
      })
  })
}
```

Figure 16: Codes – users.auction()

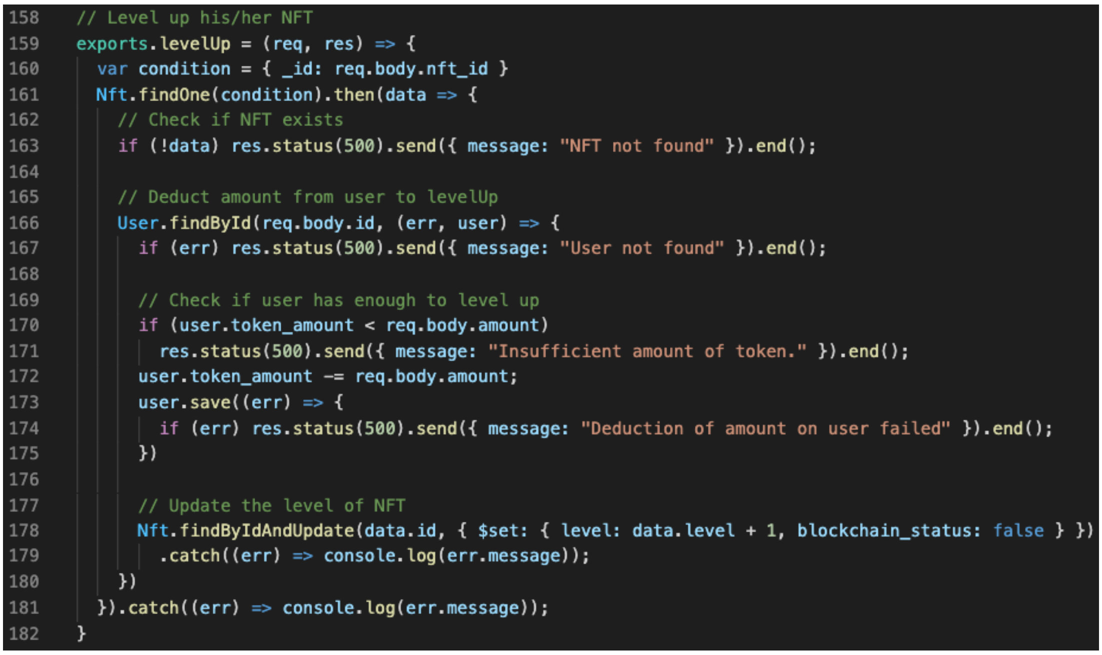
iv. Bid: bid for NFT that is being auctioned



```
// Bid for NFT that is being auctioned
exports.bid = (req, res) => {
  var condition = { _id: req.body.id }
  User.findOne(condition).then(data => {
    // Check if user is found
    if (!data) res.status(500).send({ message: "User not found." }).end();
    // Check if user has enough to bid
    if (data.token_amount < req.body.amount)
      res.status(500).send({ message: "Insufficient amount of token." }).end();

    condition = { _id: req.body.auction_id };

    Auction.findOne(condition).then(data => {
      // Check if the auction is expired
      if (data && data.bid_end_time < req.body.bid_time)
        res.status(500).send({ message: "The auction has ended." }).end();

      // Otherwise, deduct from the user tokenAmount and create bid model
      // Deduct amount from user
      User.findByIdAndUpdate(req.body.id, { $set: { token_amount: data.token_amount - req.body.amount } })
        .catch((err) => console.log(err.message));

      // Create Bid model
      const newBid = new Bid({
        user_id: req.body.id,
        auction_id: req.body.auction_id,
        amount: req.body.amount,
        blockchain_status: false
      })
      // Save bid in the database
      newBid.save(newBid)
        .then(() => { res.status(201).send({ message: "You've succesfully bid for the NFT" }) })
        .catch(err => {
          console.log(err.message);
          res.status(500).send({ message: "You've failed to bid the NFT" });
        })
    });
  });
}
```

Figure 17: Codes – users.bid()

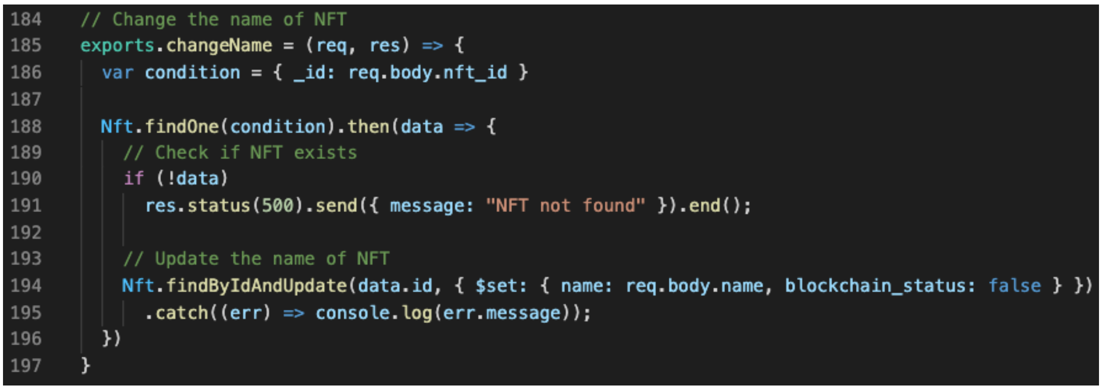
v. Level Up: level up his/her NFT



```
// Level up his/her NFT
exports.levelUp = (req, res) => {
  var condition = { _id: req.body.nft_id }
  Nft.findOne(condition).then(data => {
    // Check if NFT exists
    if (!data) res.status(500).send({ message: "NFT not found" }).end();

    // Deduct amount from user to levelUp
    User.findById(req.body.id, (err, user) => {
      if (err) res.status(500).send({ message: "User not found" }).end();

      // Check if user has enough to level up
      if (user.token_amount < req.body.amount)
        res.status(500).send({ message: "Insufficient amount of token." }).end();
      user.token_amount -= req.body.amount;
      user.save((err) => {
        if (err) res.status(500).send({ message: "Deduction of amount on user failed" }).end();
      })

      // Update the level of NFT
      Nft.findByIdAndUpdate(data.id, { $set: { level: data.level + 1, blockchain_status: false } })
        .catch((err) => console.log(err.message));
    })
  }).catch((err) => console.log(err.message));
}
```

Figure 18: Codes – users.levelUp()

vi. Change Name: change the name of NFT

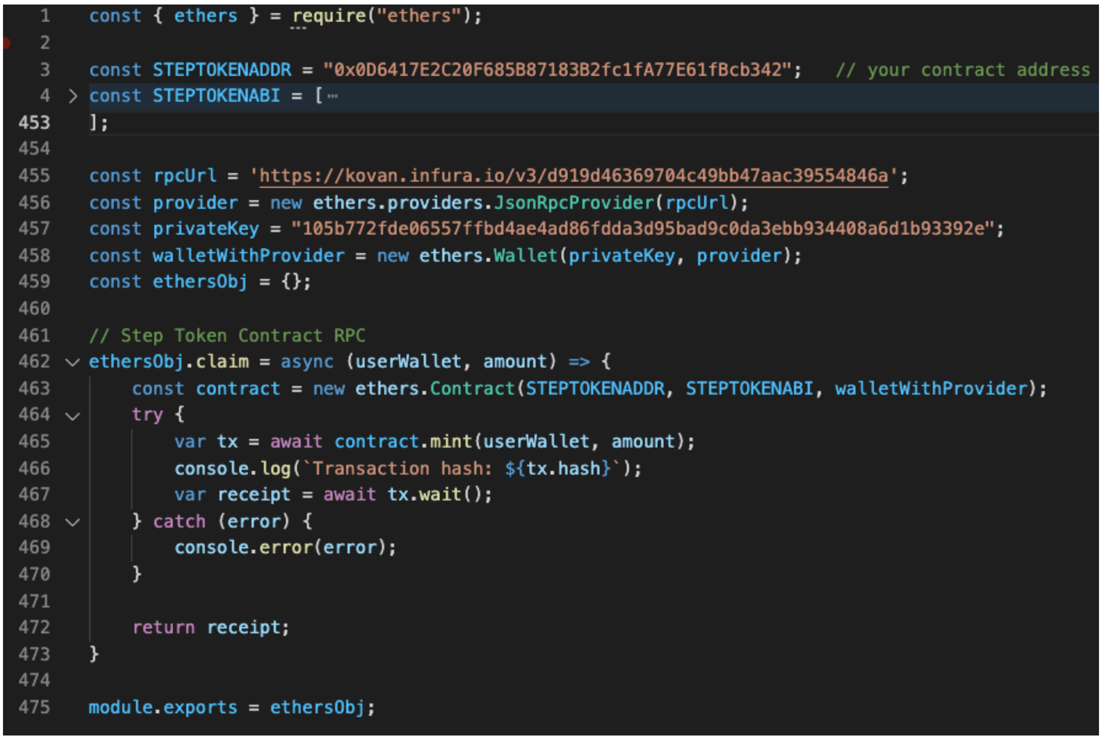


```
// Change the name of NFT
exports.changeName = (req, res) => {
  var condition = { _id: req.body.nft_id }

  Nft.findOne(condition).then(data => {
    // Check if NFT exists
    if (!data)
      res.status(500).send({ message: "NFT not found" }).end();

    // Update the name of NFT
    Nft.findByIdAndUpdate(data.id, { $set: { name: req.body.name, blockchain_status: false } })
      .catch((err) => console.log(err.message));
  })
}
```

Figure 19: Codes – users.changeName()

b. *ethers.controller.js*

i. Claim: invoke the smart contract to mint the amount that the user claims to him/her



```
const { ethers } = require("ethers");

const STEPTOKENADDR = "0x0D6417E2C20F685B87183B2fc1fA77E61fBcb342";   // your contract address
const STEPTOKENABI = [⋯
];

const rpcUrl = 'https://kovan.infura.io/v3/d919d46369704c49bb47aac39554846a';
const provider = new ethers.providers.JsonRpcProvider(rpcUrl);
const privateKey = "105b772fde06557ffbd4ae4ad86fdda3d95bad9c0da3ebb934408a6d1b93392e";
const walletWithProvider = new ethers.Wallet(privateKey, provider);
const ethersObj = {};

// Step Token Contract RPC
ethersObj.claim = async (userWallet, amount) => {
    const contract = new ethers.Contract(STEPTOKENADDR, STEPTOKENABI, walletWithProvider);
    try {
        var tx = await contract.mint(userWallet, amount);
        console.log(`Transaction hash: ${tx.hash}`);
        var receipt = await tx.wait();
    } catch (error) {
        console.error(error);
    }

    return receipt;
}

module.exports = ethersObj;
```

Figure 20: Codes – ethers.claim()

2.3.4. Testing using Postman

It is a software developer’s courtesy to ensure that whatever features or pieces of codes are well functioning at least in unit testing before making a pull request. Even though this is a solo project and yet testing the codes before the integration with other components like frontend clients and Ethereum blockchain simplifies integration complexity as well as shortens the integration time.

In this project, before deploying the codes onto the Heroku cloud, every endpoint is tested using the Postman on localhost development environment.

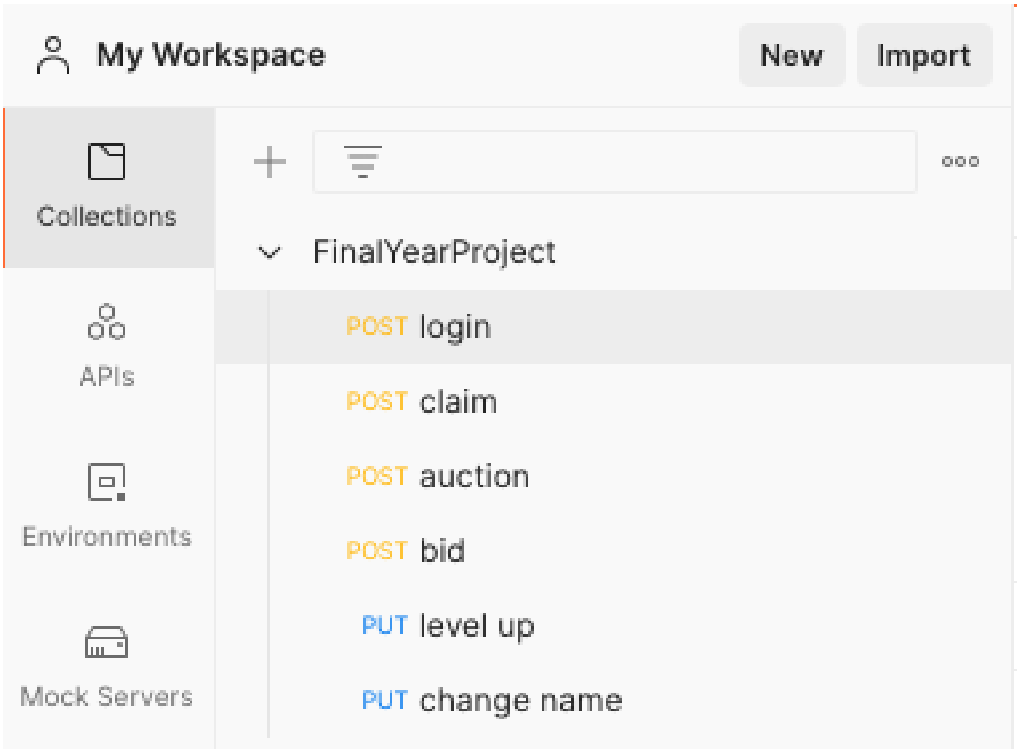
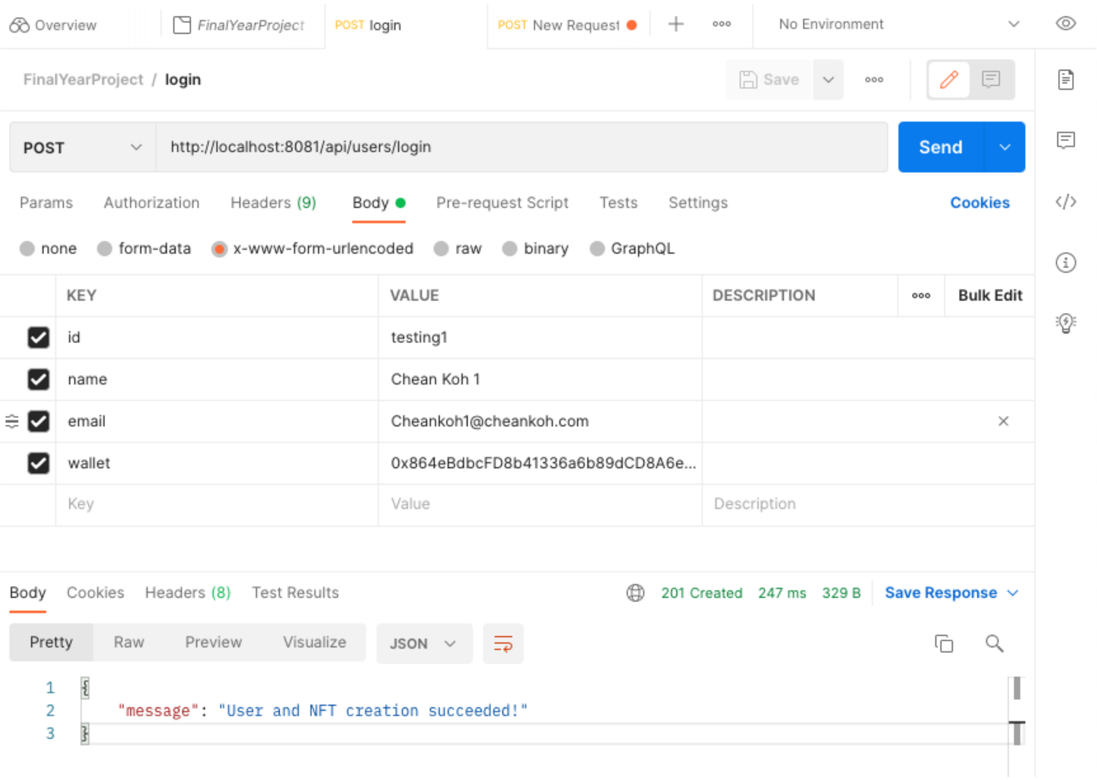


Figure 21: Postman API Structure

The figure above shows the functions that were tested using Postman.



POST http://localhost:8081/api/users/login

○ none ○ form-data ● x-www-form-urlencoded ○ raw ○ binary ○ GraphQL

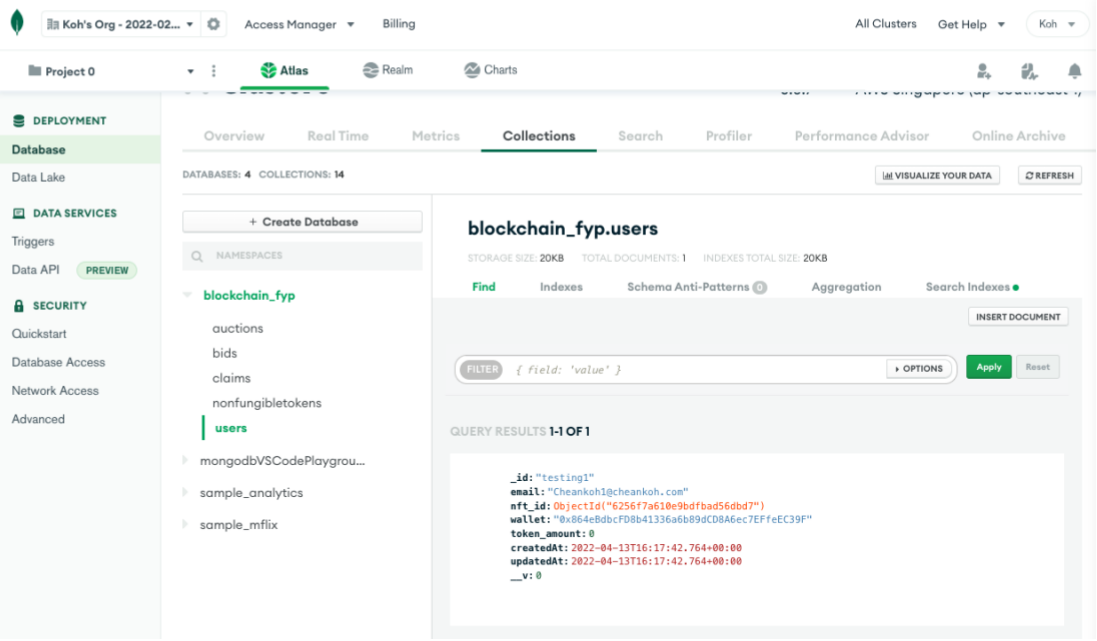
	KEY	VALUE	DESCRIPTION
☑	id	testing1	
☑	name	Chean Koh 1	
☑	email	Cheankoh1@cheankoh.com	
☑	wallet	0x864eBdbcFD8b41336a6b89dCD8A6e...	
	Key	Value	Description

201 Created 247 ms 329 B

```
{
    "message": "User and NFT creation succeeded!"
}
```

Figure 22: Postman login test

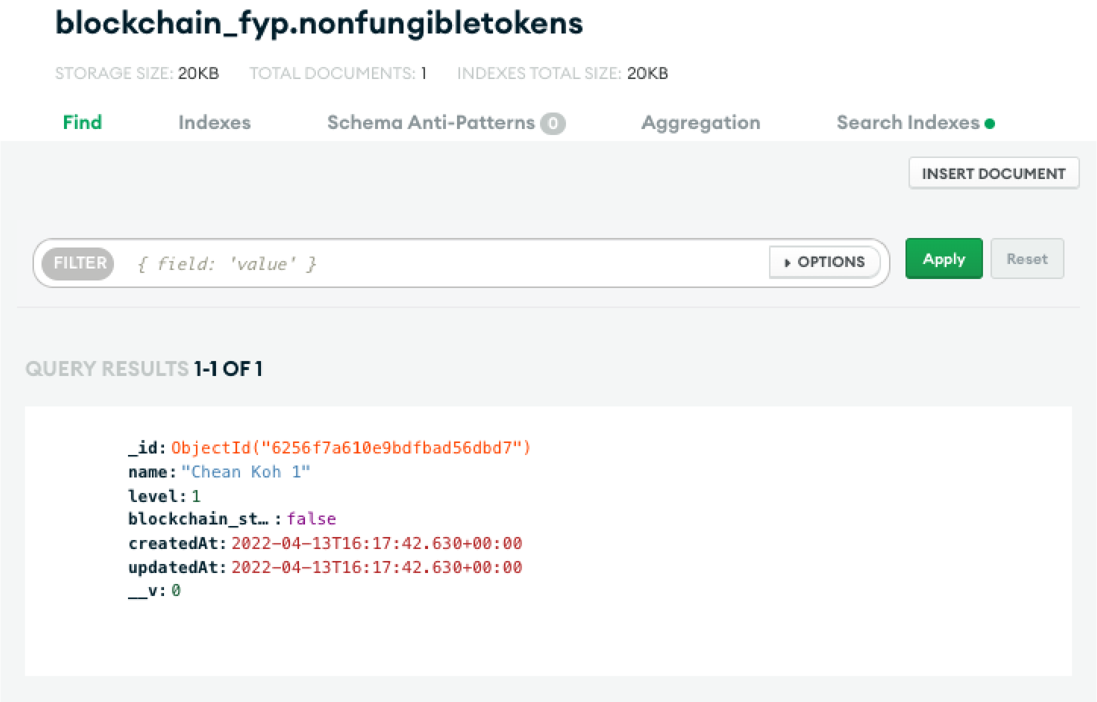
The figure above shows the configuration and the request parameters that were passed in to test the login API. At the bottom where the response from the server resides, status *201 Created* and the response message are shown. This means that the API succeeded without having errors and the user with id equal to *testing1* is logged in successfully. Now, let's have a look at the MongoDB collections.



```
_id: "testing1"
email: "Cheankoh1@cheankoh.com"
nft_id: ObjectId("6256f7a610e9bdfbad56dbd7")
wallet: "0x864eBdbcFD8b41336a6b89dCD8A6ec7EFfeEC39F"
token_amount: 0
createdAt: 2022-04-13T16:17:42.764+00:00
updatedAt: 2022-04-13T16:17:42.764+00:00
__v: 0
```

Figure 23: MongoDB users collection

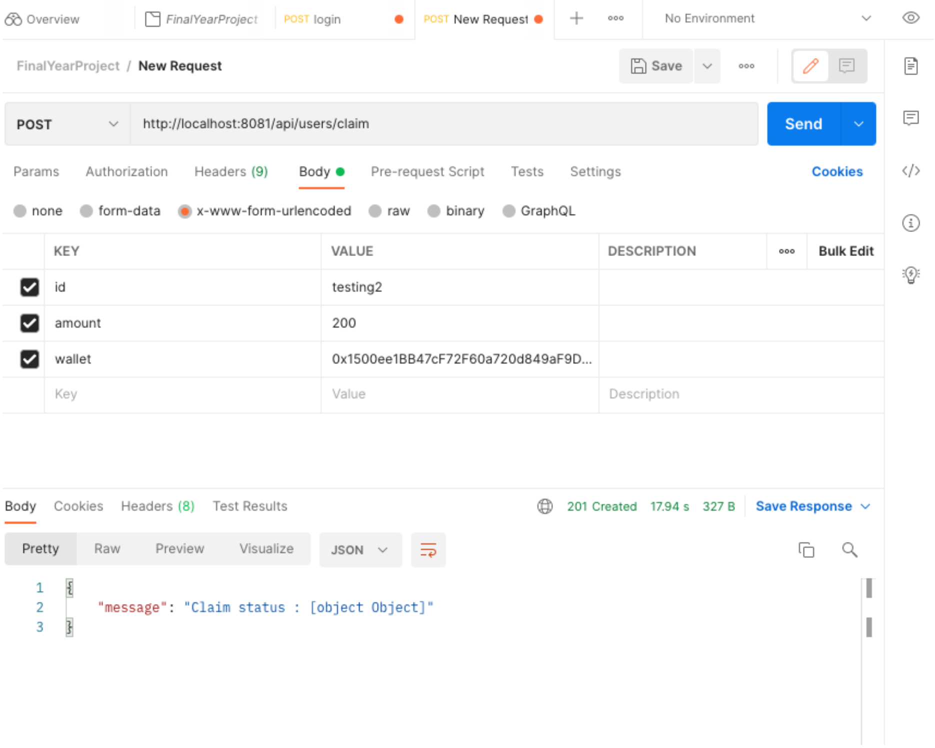
The above figure shows the browse collections page on MongoDB Atlas cloud website. At the bottom-right corner there is the document of the user *testing1*. For first timer that logged in to the application, the credentials will be stored as a new document and a NFT will be created for him/her. Whereas for existing users, their credentials will be retrieved directly.



```
_id: ObjectId("6256f7a610e9bdfbad56dbd7")
name: "Chean Koh 1"
level: 1
blockchain_st… : false
createdAt: 2022-04-13T16:17:42.630+00:00
updatedAt: 2022-04-13T16:17:42.630+00:00
__v: 0
```

Figure 24: MongoDB nonfungibletokens collection

The above figure shows the document of NFT created for the first timer that logged in to the application. Therefore, the API responding a *status 201* really means that the documents were successfully created.



POST http://localhost:8081/api/users/claim

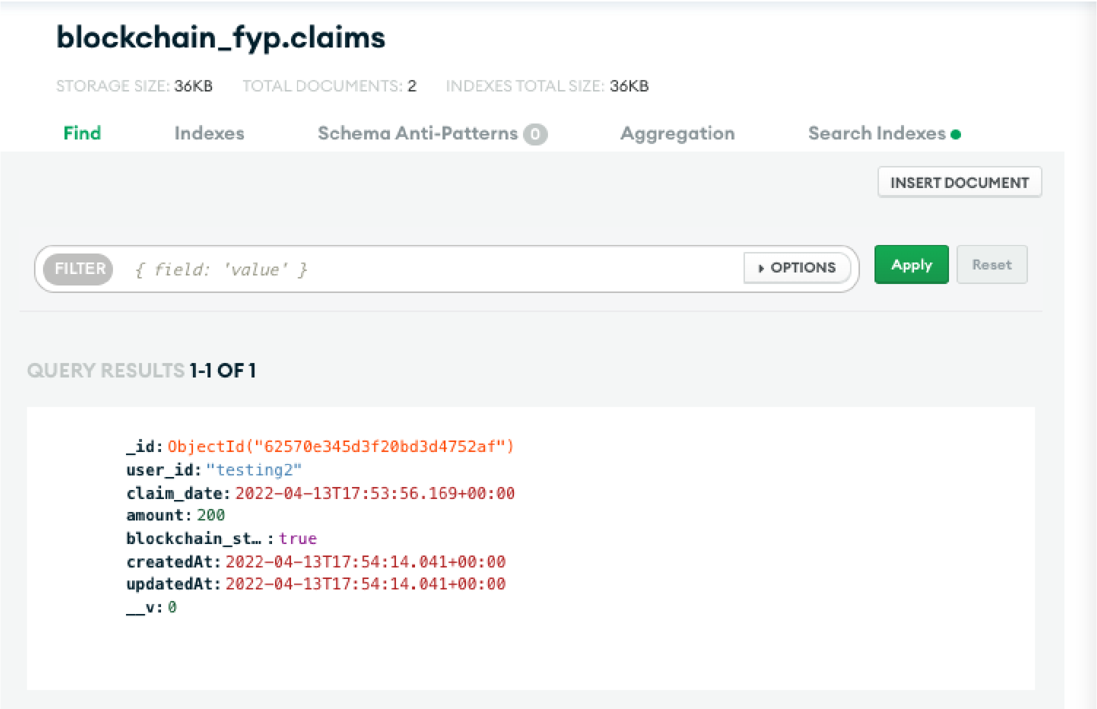
	KEY	VALUE	DESCRIPTION
☑	id	testing2	
☑	amount	200	
☑	wallet	0x1500ee1BB47cF72F60a720d849aF9D...	
	Key	Value	Description

201 Created 17.94 s 327 B

```
{
    "message": "Claim status : [object Object]"
}
```

Figure 25: Postman claim test

It was mentioned earlier in this section that the claim would be done by the backend server as we could not simply allow anyone to mint the token for the users otherwise the *tokenomics* of this project would be less appealing to users. Tokenomics refer to how the asset works, as well as the psychological or behavioural forces that could affect its value long term [6]. Therefore, the claim on chain can only be invoked by the owner of the smart contract, the backend server.



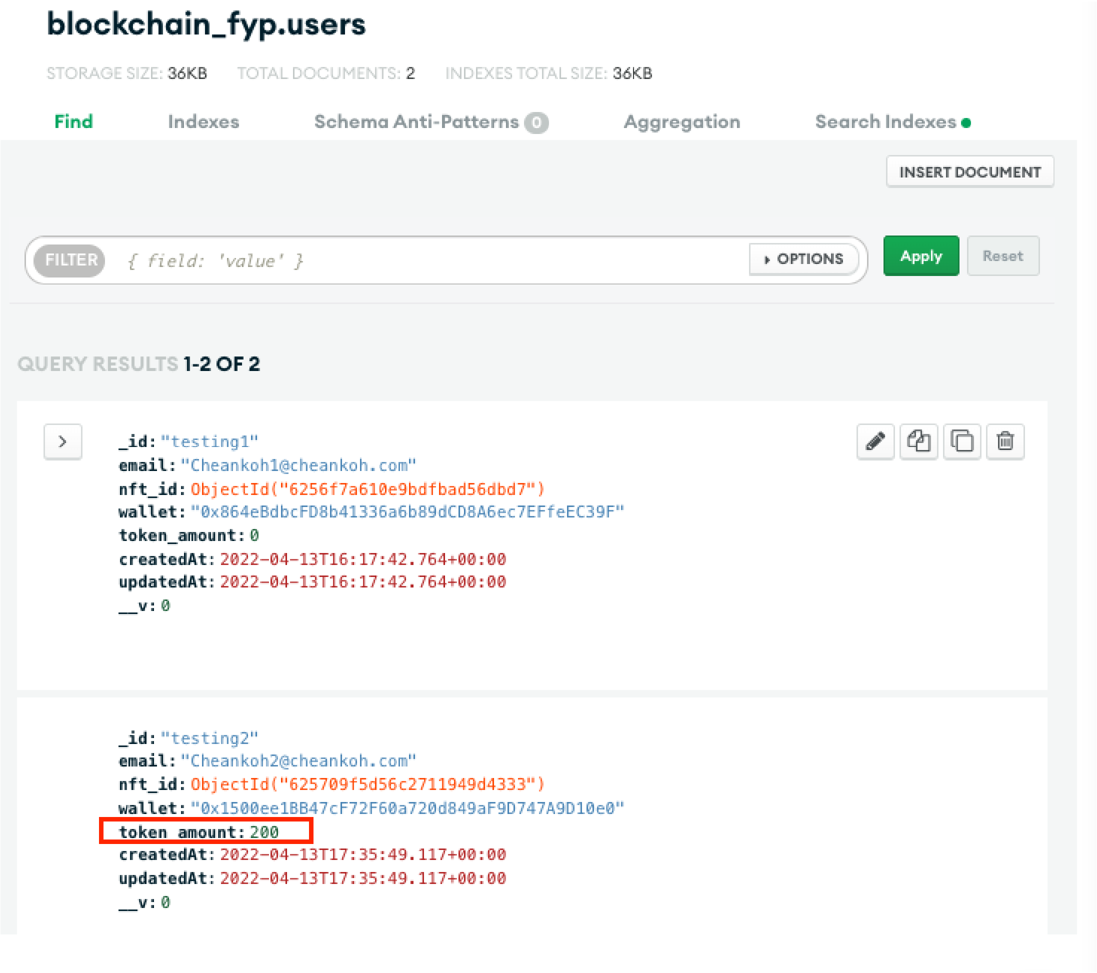
blockchain_fyp.claims

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

QUERY RESULTS 1-1 OF 1

```
_id: ObjectId("62570e345d3f20bd3d4752af")
user_id: "testing2"
claim_date: 2022-04-13T17:53:56.169+00:00
amount: 200
blockchain_st… : true
createdAt: 2022-04-13T17:54:14.041+00:00
updatedAt: 2022-04-13T17:54:14.041+00:00
__v: 0
```

Figure 26: MongoDB claims collection



blockchain_fyp.users

STORAGE SIZE: 36KB TOTAL DOCUMENTS: 2 INDEXES TOTAL SIZE: 36KB

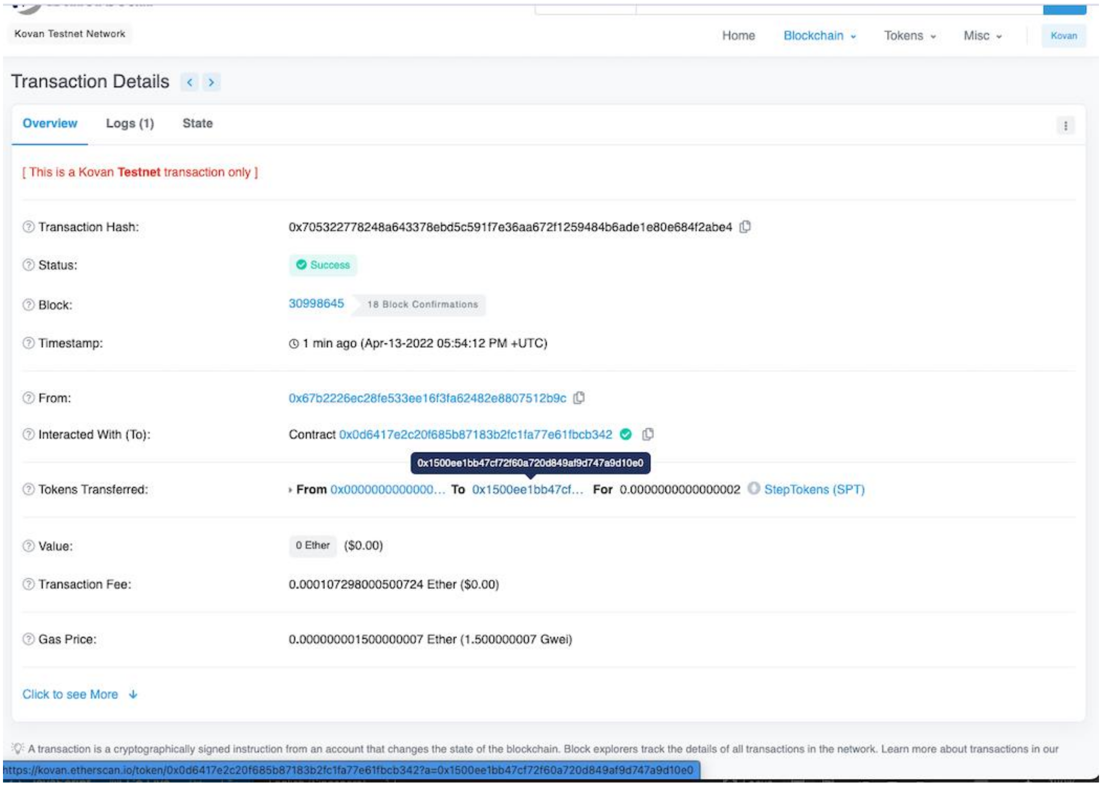
QUERY RESULTS 1-2 OF 2

```
_id: "testing1"
email: "Cheankoh1@cheankoh.com"
nft_id: ObjectId("6256f7a610e9bdfbad56dbd7")
wallet: "0x864eBdbcFD8b41336a6b89dCD8A6ec7EFfeEC39F"
token_amount: 0
createdAt: 2022-04-13T16:17:42.764+00:00
updatedAt: 2022-04-13T16:17:42.764+00:00
__v: 0
```

```
_id: "testing2"
email: "Cheankoh2@cheankoh.com"
nft_id: ObjectId("625709f5d56c2711949d4333")
wallet: "0x1500ee1BB47cF72F60a720d849aF9D747A9D10e0"
token_amount: 200
createdAt: 2022-04-13T17:35:49.117+00:00
updatedAt: 2022-04-13T17:35:49.117+00:00
__v: 0
```

Figure 27: MongoDB users collection

Figures 26 and 27 show the updated documents on both claims and users collections after calling claim using Postman. Note that the *token_amount* of the user with id equal to *testing2* was updated to 200.



Transaction Details

[This is a Kovan **Testnet** transaction only]

Transaction Hash: 0x705322778248a643378ebd5c591f7e36aa672f1259484b6ade1e80e684f2abe4

Status: Success

Block: 30998645 18 Block Confirmations

Timestamp: 1 min ago (Apr-13-2022 05:54:12 PM +UTC)

From: 0x67b2226ec28fe533ee16f3fa62482e8807512b9c

Interacted With (To): Contract 0x0d6417e2c20f685b87183b2fc1fa77e61fbcb342

0x1500ee1bb47cf72f60a720d849af9d747a9d10e0

Tokens Transferred: **From** 0x0000000000000... **To** 0x1500ee1bb47cf... **For** 0.0000000000000002 StepTokens (SPT)

Value: 0 Ether ($0.00)

Transaction Fee: 0.000107298000500724 Ether ($0.00)

Gas Price: 0.000000001500000007 Ether (1.500000007 Gwei)

A transaction is a cryptographically signed instruction from an account that changes the state of the blockchain. Block explorers track the details of all transactions in the network. Learn more about transactions in our

https://kovan.etherscan.io/token/0x0d6417e2c20f685b87183b2fc1fa77e61fbcb342?a=0x1500ee1bb47cf72f60a720d849af9d747a9d10e0

Figure 28: Etherscan of claim transaction on Kovan testnet

The figure above shows the screenshot on Etherscan on Kovan testnet after the user claims for StepTokens. Note that the wallet of the user which can be seen on *figure 27* is the same as the recipient highlighted on the *Tokens Transferred* on the figure above.

2.4. Frontend Client

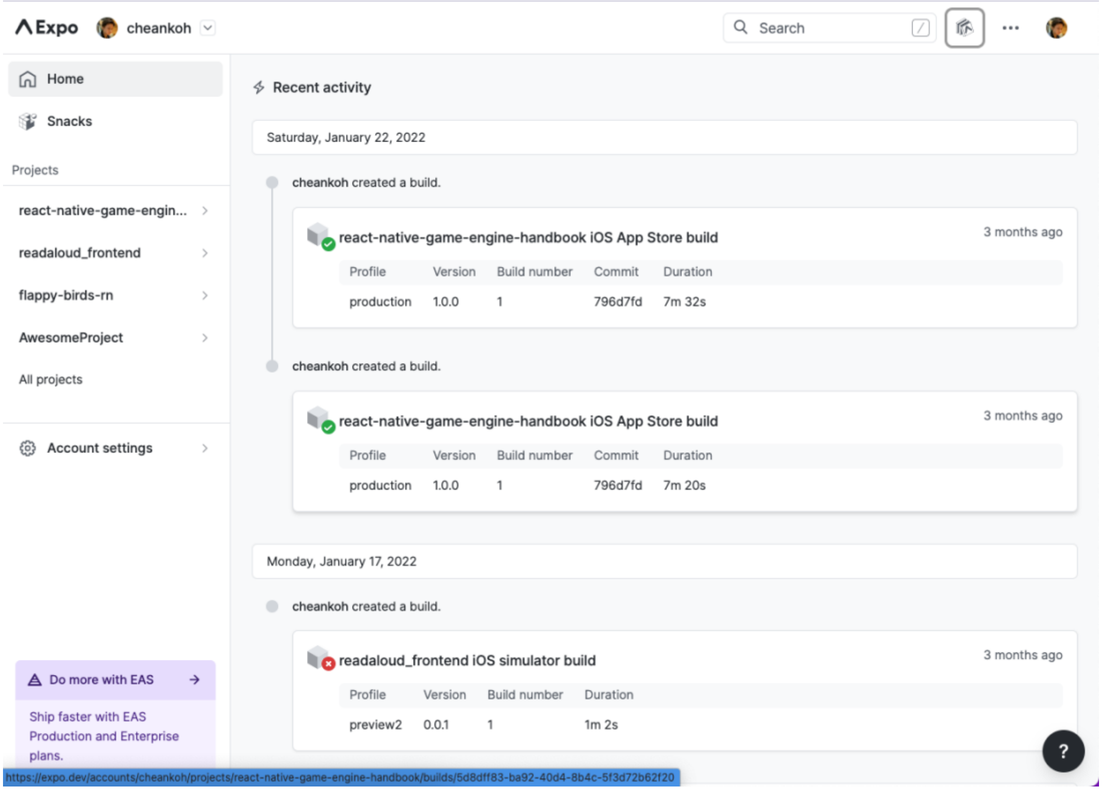
Frontend usually describes the development of graphical UIs that users can interact with on a device like computer or mobile phone. The frontend of this project uses React library which was developed by Facebook and has become one of the most influential UI libraries on recent memory. The reason that it is so widely used is the efficiency that it has to offer by manipulating the Virtual Document Object Model (VDOM) corresponding to the actual DOM, instead of the actual DOM itself. React can determine what have been changed in the DOM using VDOM, then reflect the minimum changes onto the DOM. This is such a genius workaround that saves tons of time on rendering the UIs.

2.4.1. React-native

Learn once, write anywhere is the motto of react-native. One of the reasons that react-native was chosen to be the frontend software framework in this project is that it has such huge community that building anything is possible using open-source libraries or packages.

2.4.2. Expo

Expo is the example of what a huge community can offer to the users. It is framework as well as a platform dedicated to bringing React application universal. It comes with a set of tools and services built around react-native that make developing, building, deploying, and quickly iterating on any devices from the same codebase very handy.



Recent activity

Saturday, January 22, 2022

cheankoh created a build.

react-native-game-engine-handbook iOS App Store build — 3 months ago

Profile	Version	Build number	Commit	Duration
production	1.0.0	1	796d7fd	7m 32s

cheankoh created a build.

react-native-game-engine-handbook iOS App Store build — 3 months ago

Profile	Version	Build number	Commit	Duration
production	1.0.0	1	796d7fd	7m 20s

Monday, January 17, 2022

cheankoh created a build.

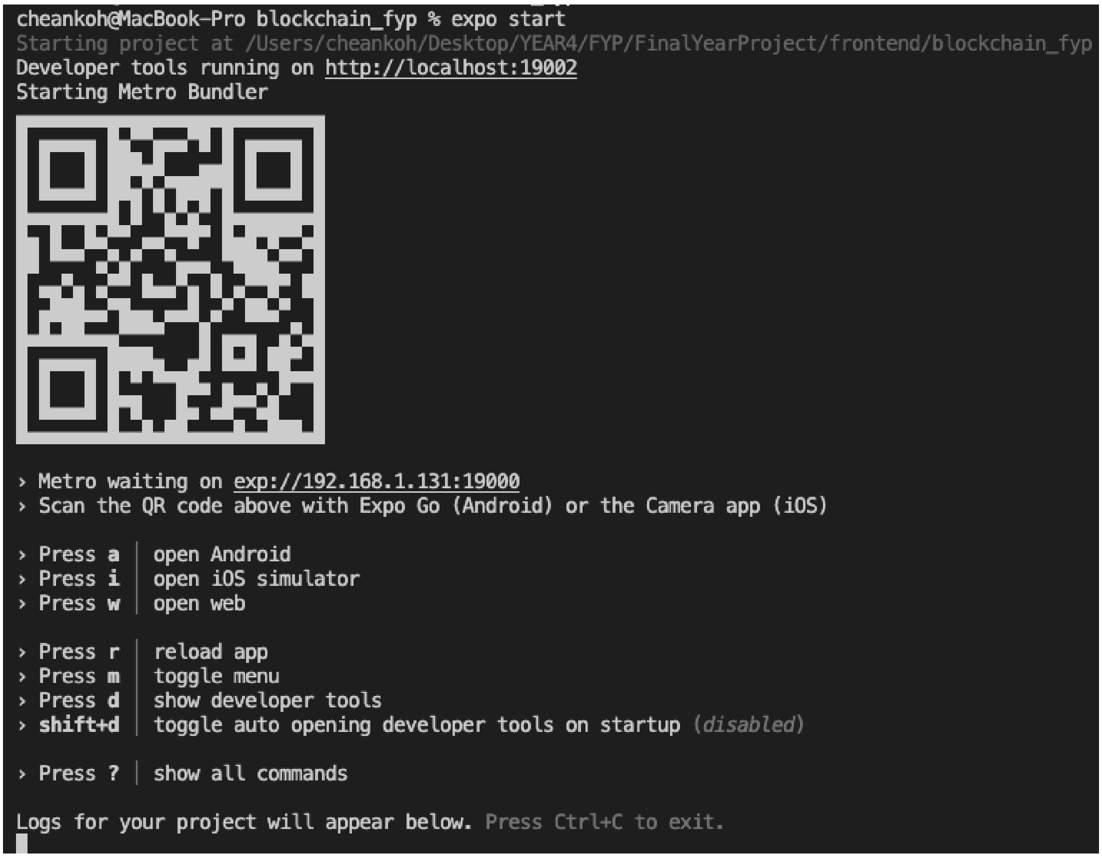
readaloud_frontend iOS simulator build — 3 months ago

Profile	Version	Build number	Duration
preview2	0.0.1	1	1m 2s

https://expo.dev/accounts/cheankoh/projects/react-native-game-engine-handbook/builds/5d8dff83-ba92-40d4-8b4c-5f3d72b62f20

Figure 29: Expo Dashboard

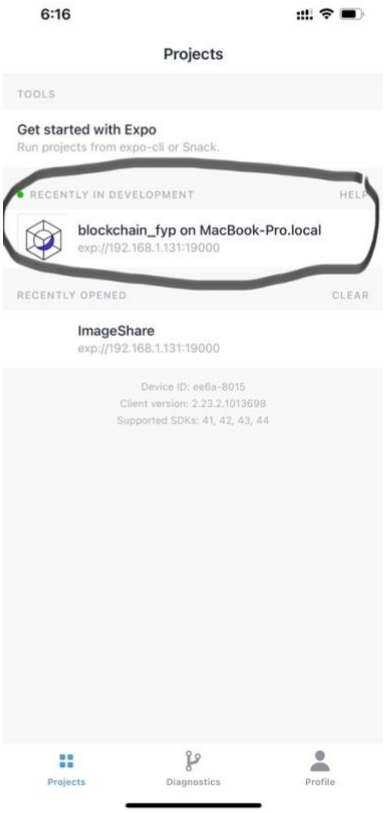
Expo enables easy deployment of *.ipa* and *.apk* files onto the cloud and provides a url pointing to the location of those files so that users could download onto their test devices.



```
cheankoh@MacBook-Pro blockchain_fyp % expo start
Starting project at /Users/cheankoh/Desktop/YEAR4/FYP/FinalYearProject/frontend/blockchain_fyp
Developer tools running on http://localhost:19002
Starting Metro Bundler

› Metro waiting on exp://192.168.1.131:19000
› Scan the QR code above with Expo Go (Android) or the Camera app (iOS)

› Press a │ open Android
› Press i │ open iOS simulator
› Press w │ open web

› Press r │ reload app
› Press m │ toggle menu
› Press d │ show developer tools
› shift+d │ toggle auto opening developer tools on startup (disabled)

› Press ? │ show all commands

Logs for your project will appear below. Press Ctrl+C to exit.
```

Figure 30: Console UI of expo

The above figure shows the UI on console after running a localhost development server on the frontend in this project. Expo allows developers to connect to the server using their mobile devices and reflect the latest changes in codes on the go using the Expo Go App on both iOS and Android. Even without using a cable to manually connect a phone to a computer and build the app bundle on that phone, changes are still visible on the go using Expo.



6:16

Projects

TOOLS

Get started with Expo

Run projects from expo-cli or Snack.

RECENTLY IN DEVELOPMENT HELP

blockchain_fyp on MacBook-Pro.local

exp://192.168.1.131:19000

RECENTLY OPENED CLEAR

ImageShare

exp://192.168.1.131:19000

Device ID: ee6a-8015

Client version: 2.23.2.1013698

Supported SDKs: 41, 42, 43, 44

Figure 31: Expo Go on mobile device

The figure is the screenshot on a mobile device on the Expo Go application. After running the localhost server, the blockchain_fyp application can be seen on Expo Go. The blockchain_fyp could be browsed and the latest changes of the codebase could be reflected real time.

2.4.3. Xcode

IOS development is the focus in this project and hence it is inevitable to use Xcode. Even though expo has hidden away the complexity of configuring the Xcode, some features like enabling logging in with Apple ID still requires an identifier of the application linked to an Apple Developer account.

Xcode also provides the use of Simulator of any iOS devices. The go-to device in this project is the iPhone 13 pro simulator in which was prompted to open by typing ‘*i*’ on the Expo console. Having a simulator that resembles actual device almost one hundred percent allows better testing especially for user-to-user interactions.

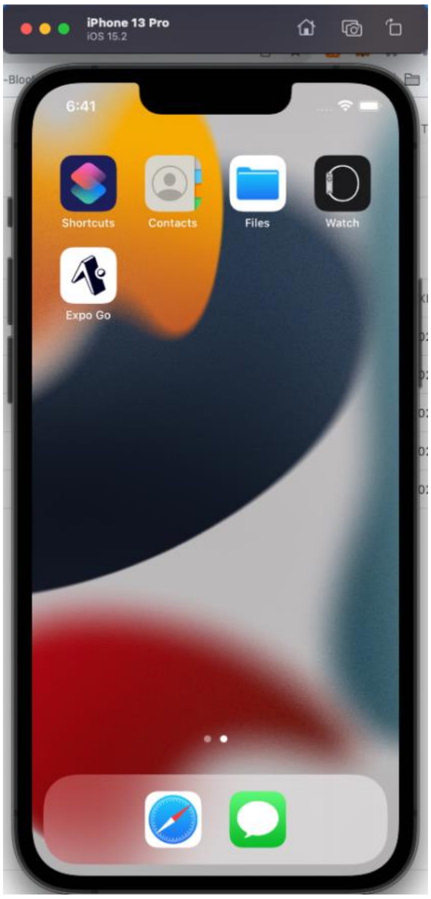


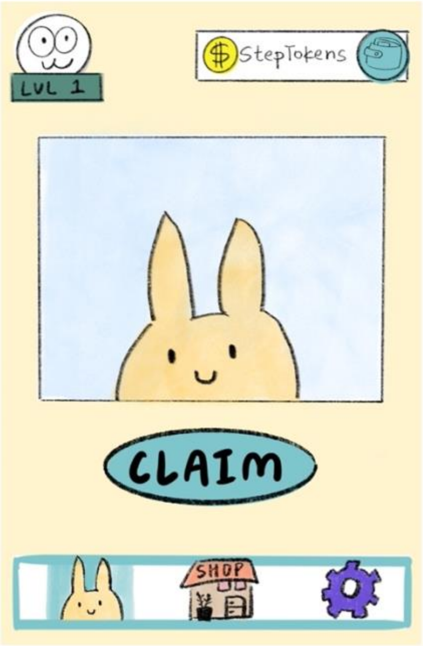
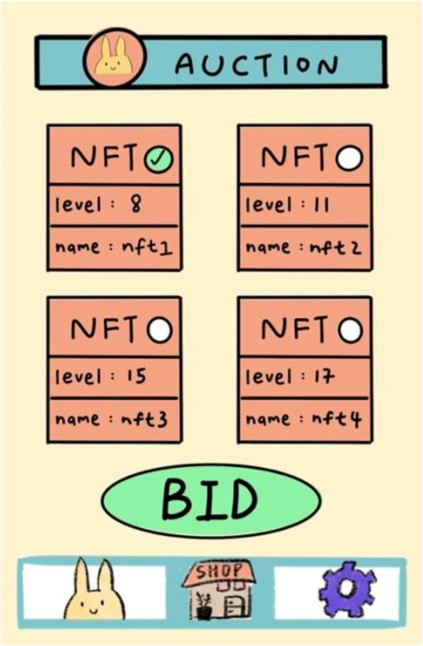
Figure 32: iOS 15.2 simulator (iPhone 13 Pro)

The above figure is the screenshot of the simulator with the Expo Go installed in it so that it can connect to the localhost server of Expo to reflect the real-time changes.

2.4.4. User Interfaces

A good UI/UX entails many topics including Human Computer Interaction, Neuroscience Marketing on User Interface, just to name a few. It is not practical to incorporate them into this project as the main focus here is the ecosystem of a DApp. Therefore, in this project, the UIs are intuitive and simple UIs that only allowed users to make simple interaction with the backend server and the Ethereum Blockchain.

The UI designs are as below:

User Interface	Description
 *Figure 33: Home Page*	After logging in to the application, a NFT with its associated name and level is created as shown on the top-left corner. Under home page, the user is allowed to claim the steps in exchange for the StepTokens. The top-right corner shows the amount of token the user has.
 AUCTION NFT level : 8 name : nft1 NFT level : 11 name : nft2 NFT level : 15 name : nft3 NFT level : 17 name : nft4 BID SHOP *Figure 34: Auction Page*	The auction page allows users to view at ongoing auction. Users are able to bid for any NFT that they are interested in, once one of the users is announced as winner, he/she will get the NFT and level up to the level of the NFT he/she bid.

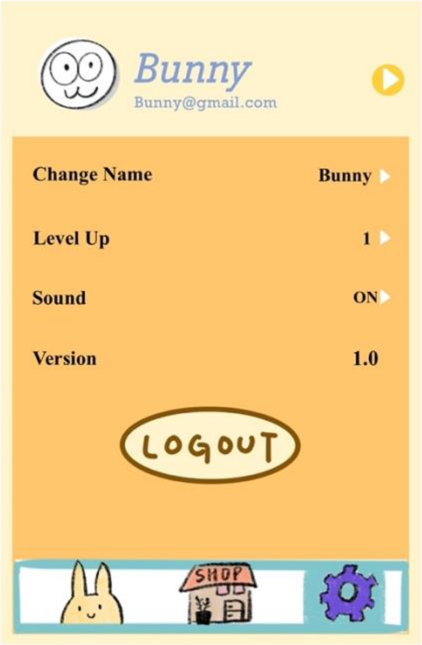
 Bunny Bunny@gmail.com Change Name Bunny Level Up 1 Sound ON Version 1.0 *Figure 35: Profile Page*	Users can change the name, level up their NFT here. Changing the name here does not cost any tokens but level up does. The higher the level of the NFT, the higher the cost to level up.

Table 8: User Interfaces

2.5. Ethereum Blockchain

Ethereum is the first blockchain that offers smart contract functionality. It has a smart contract functionality which is written in a Turing Complete language such as Solidity and compiled into a low-level bytecode and executed by the Ethereum Virtual Machine (EVM).

2.5.1. Solidity

Solidity is one of the most popular languages used in the Ethereum stack for building smart contracts. Smart contracts are programs or pieces of codes that govern the behaviour of accounts within the Ethereum state. Solidity is statically typed, supports inheritance, has a huge community as well to provide libraries that can help solving complex problem by simply importing them into your smart contracts. Undeniably, Solidity was chosen to be the language to develop smart contracts in this project.

One example among the huge community has to offer is the OpenZeppelin, best known for providing security products such as @openzeppelin package installable on npm, that helps build, automate, and operate decentralised applications. They made sure the contract standards such as ERC-20, ERC-721 are well complied with in any versions of Solidity. As such, @openzeppelin contracts are vastly imported and referenced in this project. The contracts that are imported from @openzeppelin are as follow:

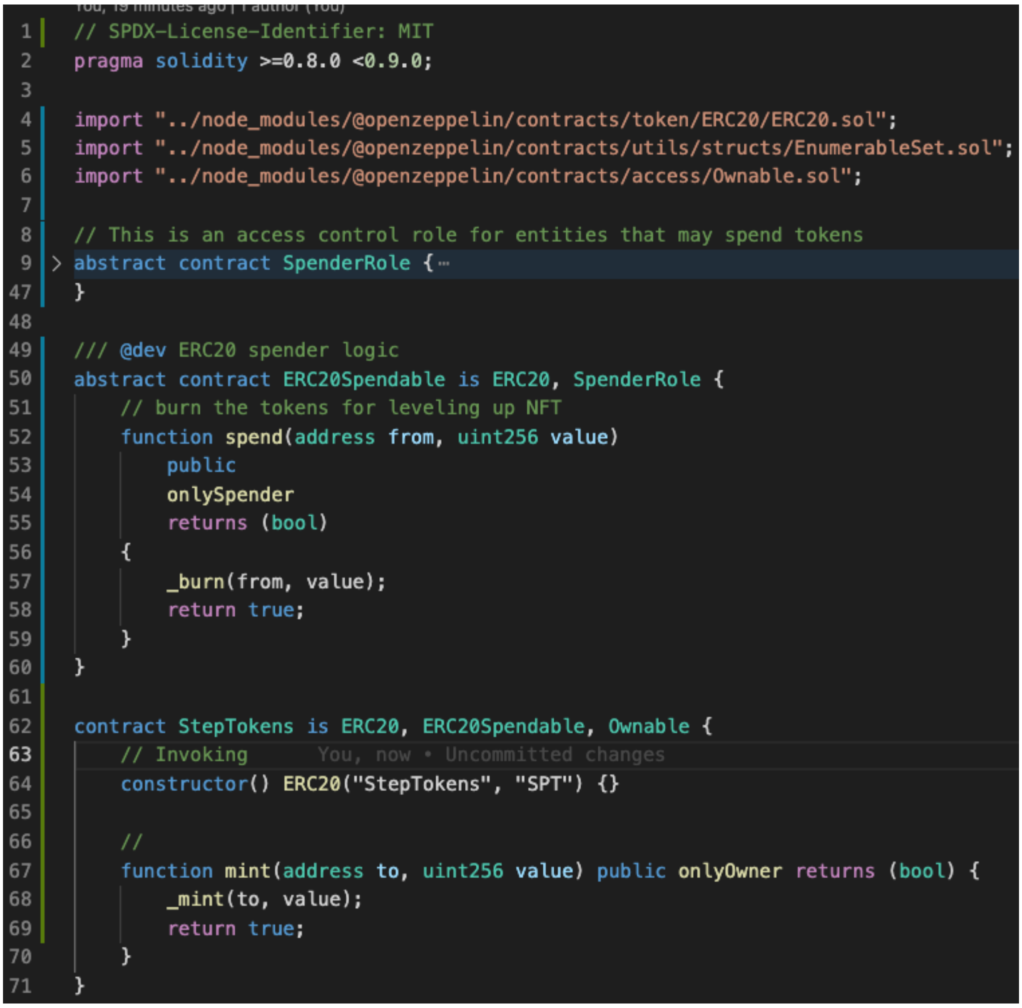
**The following contracts precede by @openzeppelin/contracts/*

Contracts	**Usage**
token/ERC20/ERC20.sol	Implements the IERC20 interface and provides standard features of a ERC20 contract. Inherited in *StepTokens.sol*.
utils/structs/EnumerableSet.sol	Adds, removes, checks for existence of elements in constant time complexity. Used by *StepTokens.sol* to control the access the role that may spend tokens that is stored in a set.
access/Ownable.sol	Provides a basic access control mechanism whereby only the owner can be granted exclusive access to specific functions. Inherited in *StepTokens.sol* and *StepNFTFactory.sol*.
token/ERC721/ERC721.sol	Implements the IERC721 interface and provides standard features of a ERC721 contract. Inherited in *StepOwnership.sol*.

Table 9: @openzeppelin contracts used in this project

2.5.2. StepTokens.sol

This is an ERC-20 compatible contract. Only owner of this contract is able to mint the tokens for the users of this project.



```
        You, 19 minutes ago | 1 author (You)
 1   // SPDX-License-Identifier: MIT
 2   pragma solidity >=0.8.0 <0.9.0;
 3
 4   import "../node_modules/@openzeppelin/contracts/token/ERC20/ERC20.sol";
 5   import "../node_modules/@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
 6   import "../node_modules/@openzeppelin/contracts/access/Ownable.sol";
 7
 8   // This is an access control role for entities that may spend tokens
 9 > abstract contract SpenderRole {⋯
47   }
48
49   /// @dev ERC20 spender logic
50   abstract contract ERC20Spendable is ERC20, SpenderRole {
51       // burn the tokens for leveling up NFT
52       function spend(address from, uint256 value)
53           public
54           onlySpender
55           returns (bool)
56       {
57           _burn(from, value);
58           return true;
59       }
60   }
61
62   contract StepTokens is ERC20, ERC20Spendable, Ownable {
63       // Invoking      You, now • Uncommitted changes
64       constructor() ERC20("StepTokens", "SPT") {}
65
66       //
67       function mint(address to, uint256 value) public onlyOwner returns (bool) {
68           _mint(to, value);
69           return true;
70       }
71   }
```

Figure 36: Codes – StepTokens.sol

The above figure is the code for *StepTokens.sol*. In line 67, there is a onlyOwner which is the modifier from *@openzeppelin/contracts/access/Ownable.sol* that restricts addresses other than the owner's address from invoking this function.

2.5.3. StepOwnership.sol

This contract inherits both *StepLevelUp.sol* and *@openzeppelin/contracts/token/ERC721/ERC721.sol* whereby

StepLevelUp.sol inherits from *StepNFTFactory.sol*. The hierarchy of the contracts is as follow:

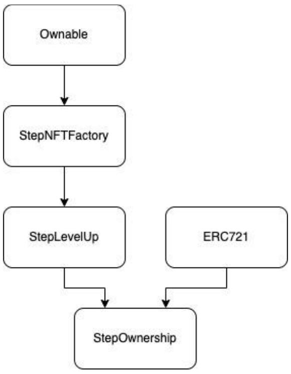
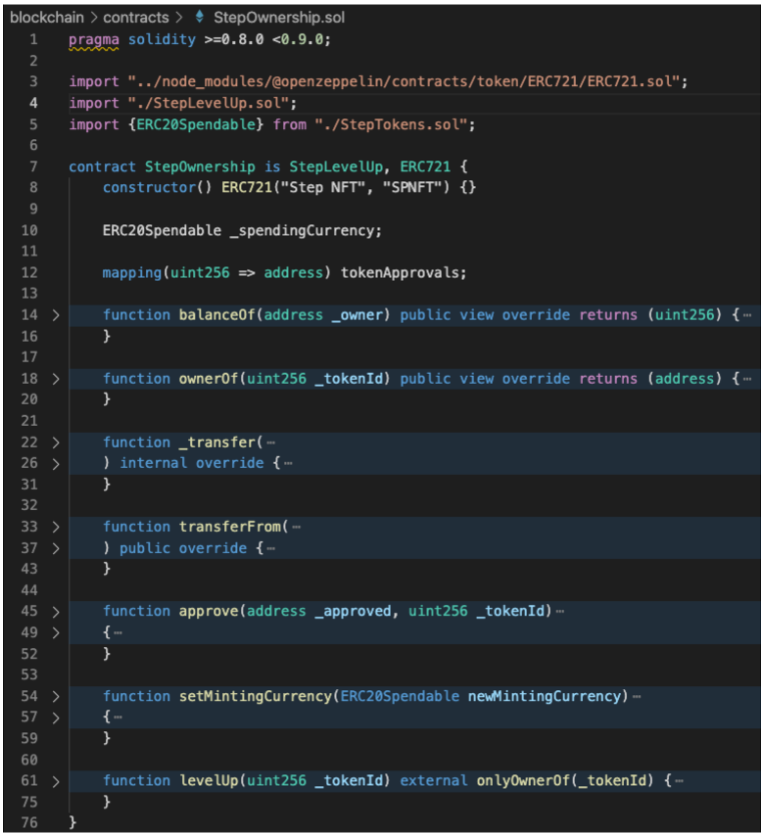


Figure 37: Hierarchy of NFT contracts

The above figure depicts the hierarchy of the NFT contracts. The upper ones are the parents of the bottom ones. In other words, the *StepOwnership.sol* inherited all the methods, functions, modifiers, mappings and fields of all the contracts above it.



```
pragma solidity >=0.8.0 <0.9.0;

import "../node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "./StepLevelUp.sol";
import {ERC20Spendable} from "./StepTokens.sol";

contract StepOwnership is StepLevelUp, ERC721 {
    constructor() ERC721("Step NFT", "SPNFT") {}

    ERC20Spendable _spendingCurrency;

    mapping(uint256 => address) tokenApprovals;

    function balanceOf(address _owner) public view override returns (uint256) {⋯
    }

    function ownerOf(uint256 _tokenId) public view override returns (address) {⋯
    }

    function _transfer(⋯
    ) internal override {⋯
    }

    function transferFrom(⋯
    ) public override {⋯
    }

    function approve(address _approved, uint256 _tokenId)⋯
    {⋯
    }

    function setMintingCurrency(ERC20Spendable newMintingCurrency)⋯
    {⋯
    }

    function levelUp(uint256 _tokenId) external onlyOwnerOf(_tokenId) {⋯
    }
}
```

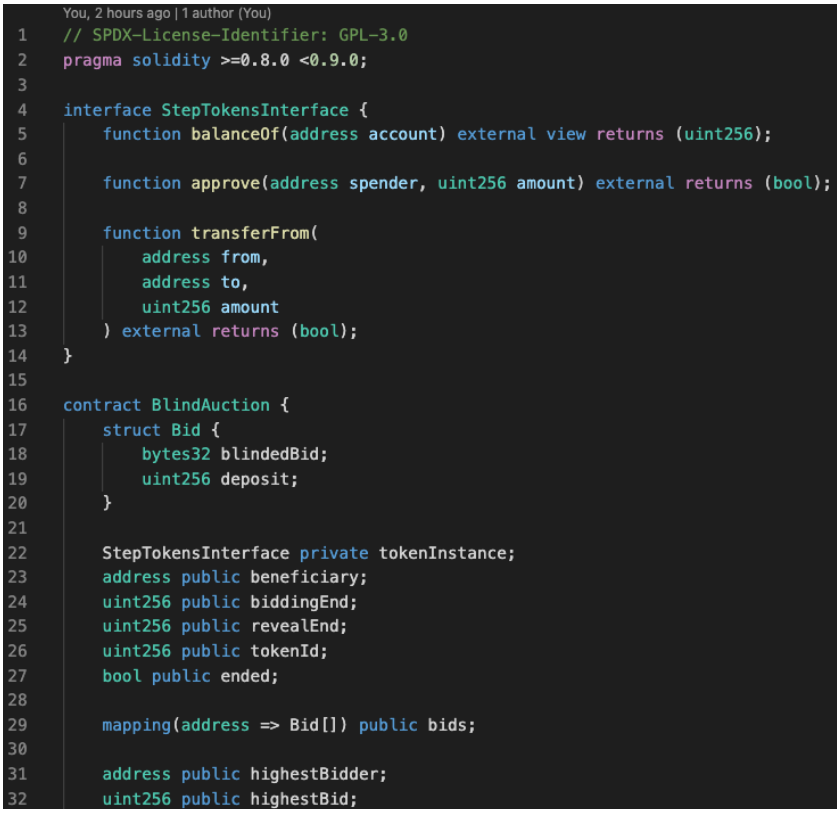
Figure 38: Codes – StepOwnership.sol

2.5.4. GameAuction.sol

In this project, blind auction is implemented for the users to bid for the NFT that wish. Blind auction works by hiding the prices and the number of bidders of a specified item and upon termination of the auction, the highest bidder is announced as the winner and pays the beneficiary.

However, in a decentralised system, there should not be a centralized entity like backend server that holds the token until the termination of auction. Hence, a slightly different mechanism needs to be implemented for temporary hold of these tokens until termination of auction on the smart contract.

Fortunately, there is a smart contract template for blind auction written on the Solidity Documentation that can be used and the *GameAuction.sol* in this project follow the previously mentioned template. There are some changes made to cater to the usage of this project such as the currency used for the auction is StepTokens instead of ether.



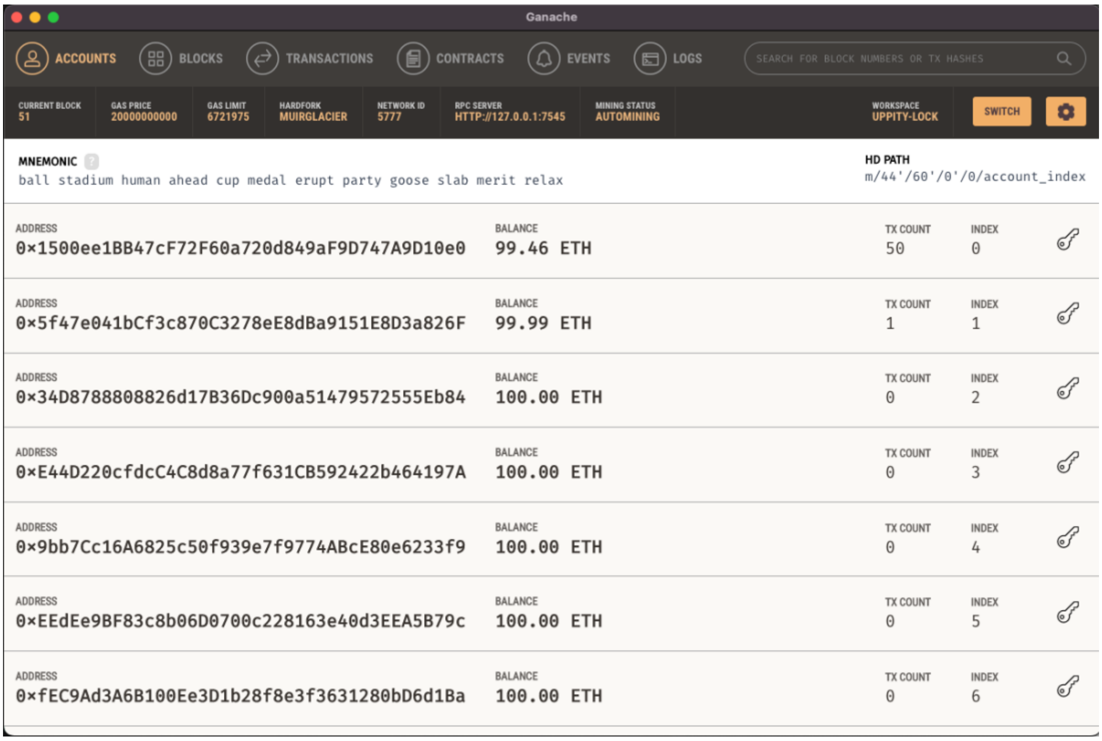
You, 2 hours ago | 1 author (You)

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.8.0 <0.9.0;

interface StepTokensInterface {
    function balanceOf(address account) external view returns (uint256);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 amount
    ) external returns (bool);
}

contract BlindAuction {
    struct Bid {
        bytes32 blindedBid;
        uint256 deposit;
    }

    StepTokensInterface private tokenInstance;
    address public beneficiary;
    uint256 public biddingEnd;
    uint256 public revealEnd;
    uint256 public tokenId;
    bool public ended;

    mapping(address => Bid[]) public bids;

    address public highestBidder;
    uint256 public highestBid;
```

Figure 39: Codes – Snapshot of GameAuction.sol

2.5.5. Ganache

As mentioned in Section 1.6., with just one click and a local blockchain is fired up. An extremely nice feature that Ganache offers is the ability to link the Truffle project which will be mentioned in the later section. After linking a Truffle project, the smart contracts of this project are shown on the UI and their respective statuses about whether they have been deployed to Ganache.



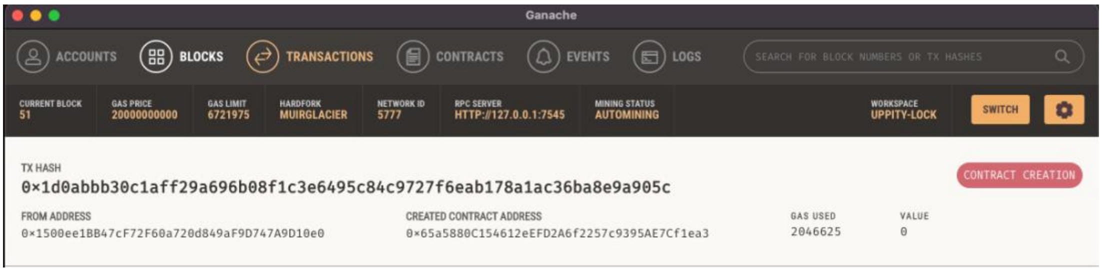
CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
51	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	AUTOMINING	UPPITY-LOCK

MNEMONIC
ball stadium human ahead cup medal erupt party goose slab merit relax

HD PATH
m/44'/60'/0'/0/account_index

ADDRESS	BALANCE	TX COUNT	INDEX
0×1500ee1BB47cF72F60a720d849aF9D747A9D10e0	99.46 ETH	50	0
0×5f47e041bCf3c870C3278eE8dBa9151E8D3a826F	99.99 ETH	1	1
0×34D8788808826d17B36Dc900a51479572555Eb84	100.00 ETH	0	2
0×E44D220cfdcC4C8d8a77f631CB592422b464197A	100.00 ETH	0	3
0×9bb7Cc16A6825c50f939e7f9774ABcE80e6233f9	100.00 ETH	0	4
0×EEdEe9BF83c8b06D0700c228163e40d3EEA5B79c	100.00 ETH	0	5
0×fEC9Ad3A6B100Ee3D1b28f8e3f3631280bD6d1Ba	100.00 ETH	0	6

Figure 40: Ganache UI, Account Page



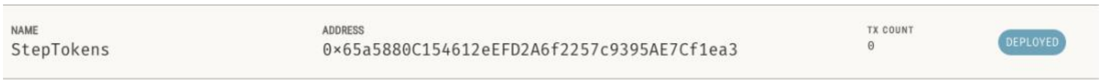
CURRENT BLOCK	GAS PRICE	GAS LIMIT	HARDFORK	NETWORK ID	RPC SERVER	MINING STATUS	WORKSPACE
51	20000000000	6721975	MUIRGLACIER	5777	HTTP://127.0.0.1:7545	AUTOMINING	UPPITY-LOCK

TX HASH
0×1d0abbb30c1aff29a696b08f1c3e6495c84c9727f6eab178a1ac36ba8e9a905c

FROM ADDRESS	CREATED CONTRACT ADDRESS	GAS USED	VALUE
0×1500ee1BB47cF72F60a720d849aF9D747A9D10e0	0×65a5880C154612eEFD2A6f2257c9395AE7Cf1ea3	2046625	0

Figure 41: Transaction of Deployment

The above figure shows the transaction hash as well as other information regarding the deployment of *StepTokens.sol*. It cannot be seen that this is *StepTokens.sol* but there is a *Contract Creation* on the right that is covered in red.



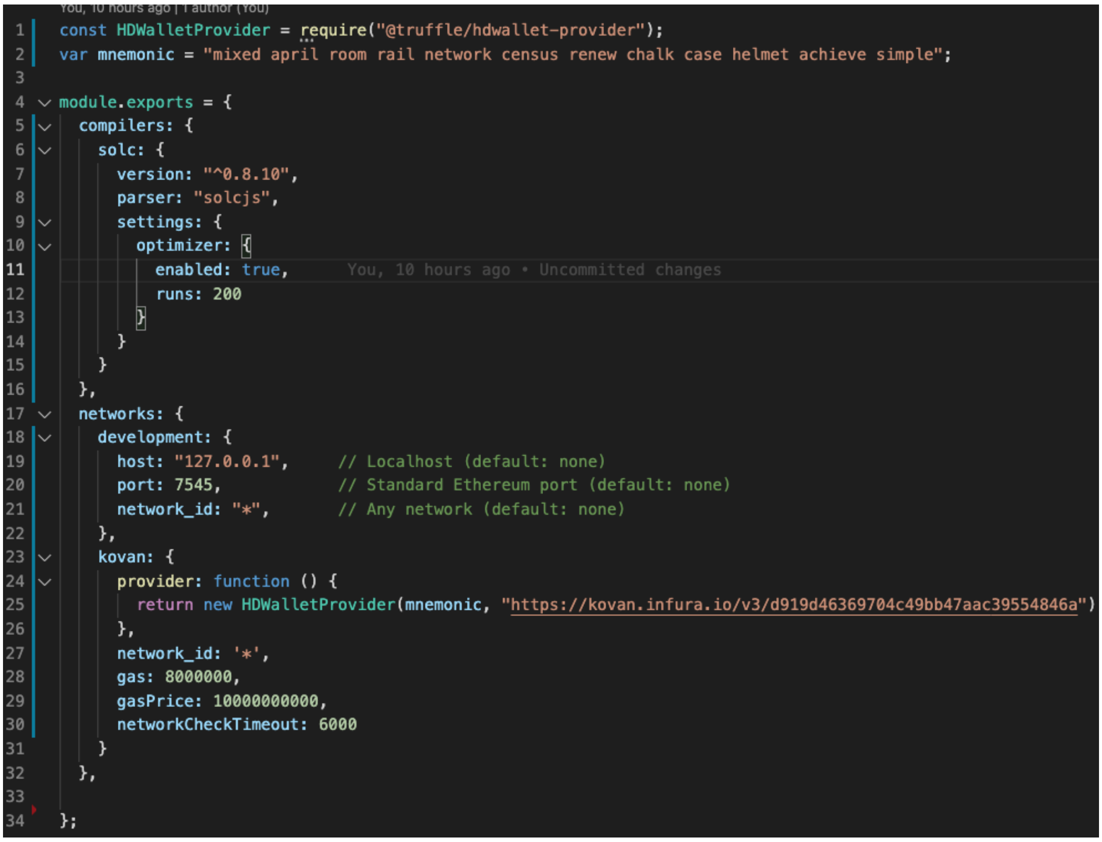
NAME	ADDRESS	TX COUNT	
StepTokens	0×65a5880C154612eEFD2A6f2257c9395AE7Cf1ea3	0	DEPLOYED

Figure 42: Status of StepTokens.sol

Due to the fact that this project is linked to the Ganache, all the smart contracts are shown on the *Contracts Tab*. It can be seen that the *StepTokens* contract was deployed to Ganache and the address that it was deployed to matches the one in figure 41.

2.5.6. Truffle

Truffle offers its own Command Line Interface (CLI) that incorporates a set of features such as init, deploy, compile, just to name a few. After installing truffle globally using npm, a truffle project can simply be created by running *truffle init* under the directory of the blockchain project. Truffle was widely used in this project for compiling, unit testing and deploying.



```
You, 10 hours ago | 1 author (You)
const HDWalletProvider = require("@truffle/hdwallet-provider");
var mnemonic = "mixed april room rail network census renew chalk case helmet achieve simple";

module.exports = {
  compilers: {
    solc: {
      version: "^0.8.10",
      parser: "solcjs",
      settings: {
        optimizer: {
          enabled: true,      You, 10 hours ago • Uncommitted changes
          runs: 200
        }
      }
    }
  },
  networks: {
    development: {
      host: "127.0.0.1",     // Localhost (default: none)
      port: 7545,            // Standard Ethereum port (default: none)
      network_id: "*",       // Any network (default: none)
    },
    kovan: {
      provider: function () {
        return new HDWalletProvider(mnemonic, "https://kovan.infura.io/v3/d919d46369704c49bb47aac39554846a")
      },
      network_id: '*',
      gas: 8000000,
      gasPrice: 10000000000,
      networkCheckTimeout: 6000
    }
  },

};
```

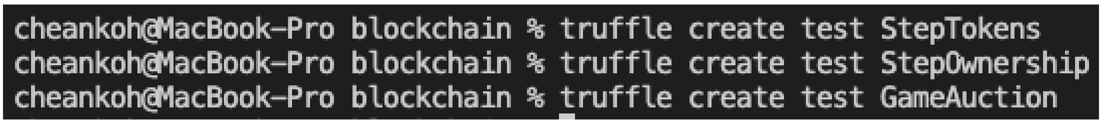
Figure 43: Codes – truffle-config.js

The above figure is the screenshot of *truffle-config.js*. It contains the configuration settings for Ganache starting from line 18 which written as development, as well as for kovan testnet starting from line 23.

2.5.7. Truffle Testing

Truffle offers command line testing on truffle console which is extremely tedious as all the testing command lines need to be stored elsewhere and be copied and pasted line by line on the truffle console. Thus, it was not chosen to be the testing method in this project.

Instead, JavaScript testing was chosen despite being more complicated than the command lines.



```
cheankoh@MacBook-Pro blockchain % truffle create test StepTokens
cheankoh@MacBook-Pro blockchain % truffle create test StepOwnership
cheankoh@MacBook-Pro blockchain % truffle create test GameAuction
```

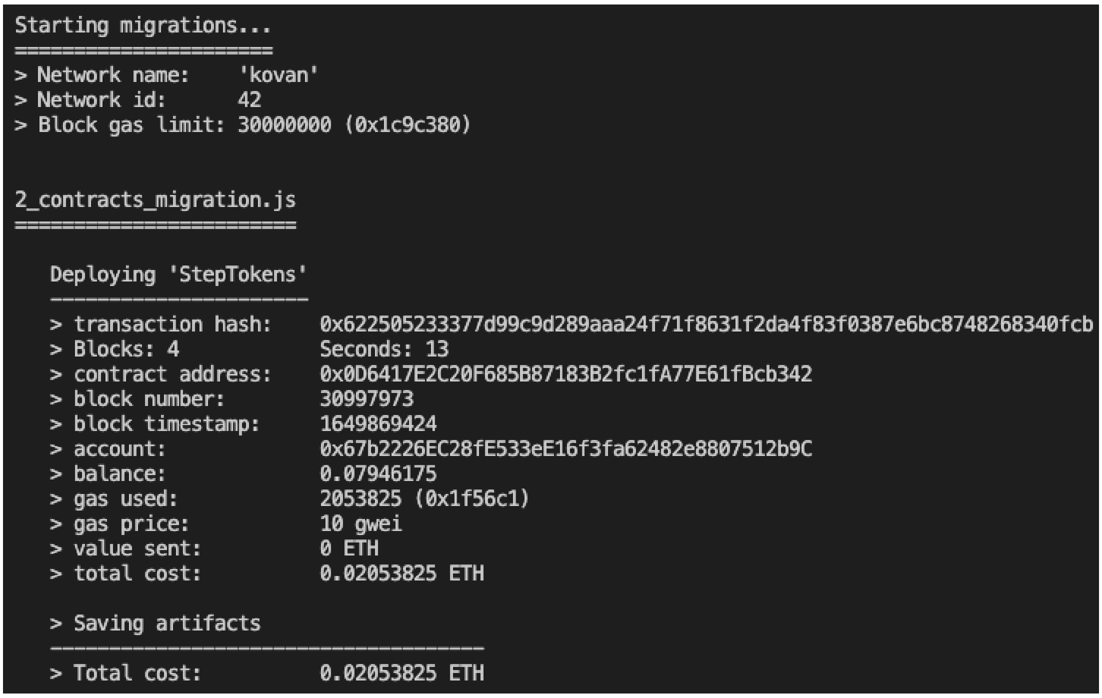
Figure 44: Command lines for creating tests

The above figure shows the command lines to create JavaScript tests for smart contracts. The StepTokens, StepOwnership and GameAuction correspond to *StepTokens.sol*, *StepOwnership.sol* and *GameAuction.sol* respectively.

2.5.8. Deployment to Kovan

There are many testnet more widely used than the Kovan testnet but most of them ran out of faucet services, which are meant to give fake ethers to people. Fortunately, Kovan is not one of them and hence was chosen to be the testnet of this project.

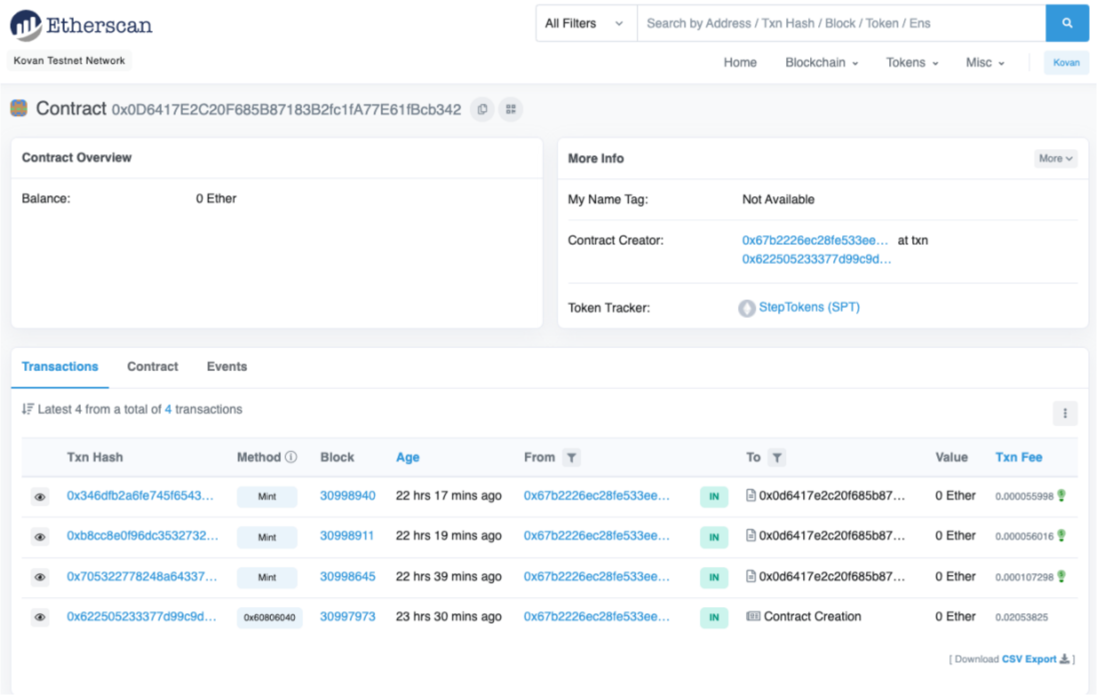
There are in total three contracts that were deployed to the Kovan testnet. The *StepTokens.sol*, *StepOwnership.sol* and *GameAuction.sol* were deployed to the Kovan testnet. The example of invoking the functions on *StepTokens.sol* was shown in *Section 2.4.3.*.



```
Starting migrations...
======================
> Network name:    'kovan'
> Network id:      42
> Block gas limit: 30000000 (0x1c9c380)


2_contracts_migration.js
========================

   Deploying 'StepTokens'
   ----------------------
   > transaction hash:    0x622505233377d99c9d289aaa24f71f8631f2da4f83f0387e6bc8748268340fcb
   > Blocks: 4            Seconds: 13
   > contract address:    0x0D6417E2C20F685B87183B2fc1fA77E61fBcb342
   > block number:        30997973
   > block timestamp:     1649869424
   > account:             0x67b2226EC28fE533eE16f3fa62482e8807512b9C
   > balance:             0.07946175
   > gas used:            2053825 (0x1f56c1)
   > gas price:           10 gwei
   > value sent:          0 ETH
   > total cost:          0.02053825 ETH

   > Saving artifacts
   -------------------------------------
   > Total cost:          0.02053825 ETH
```

Figure 45: StepTokens deployed to Kovan

The figure above shows the *StepTokens.sol* being deployed to the Kovan testnet and some other information regarding the deployment. The important information here is the contract address, 0x0D6417E2C20F685B87183B2fc1fA77E61fBcb342. The most intriguing thing in blockchain is that everything is transparent to public. Every information about this contract and every transaction associated with it are visible on etherscan.io.



Contract 0x0D6417E2C20F685B87183B2fc1fA77E61fBcb342

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

Contract Creator: 0x67b2226ec28fe533ee... at txn 0x622505233377d99c9d...

Token Tracker: StepTokens (SPT)

Latest 4 from a total of 4 transactions

Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
0x346dfb2a6fe745f6543...	Mint	30998940	22 hrs 17 mins ago	0x67b2226ec28fe533ee...	IN	0x0d6417e2c20f685b87...	0 Ether	0.000055998
0xb8cc8e0f96dc3532732...	Mint	30998911	22 hrs 19 mins ago	0x67b2226ec28fe533ee...	IN	0x0d6417e2c20f685b87...	0 Ether	0.000056016
0x705322778248a64337...	Mint	30998645	22 hrs 39 mins ago	0x67b2226ec28fe533ee...	IN	0x0d6417e2c20f685b87...	0 Ether	0.000107298
0x622505233377d99c9d...	0x60806040	30997973	23 hrs 30 mins ago	0x67b2226ec28fe533ee...	IN	Contract Creation	0 Ether	0.02053825

Figure 46: Etherscan of StepTokens.sol on Kovan

The figure above is the screenshot of the *StepTokens.sol* after being deployed to the Kovan testnet.

2.5.9. Insights

DApp contradicts how a traditional centralised application works. The backend server here does not have much information about the invocation status of the users. In other words, even though the implementation allows the users to invoke smart contract and send the backend server at the same time, the server does not know when the transaction will be mined to the block and if the transactions of the users are valid. What the backend server can do is to keep track of the users' requests, store the information needed with an additional field like *blockchain_status* in database, and lastly subscribe to the events to retrieve some important information that the application needs to store. Upon receiving the events from the blockchain, the server keeps track of the status of the users' invocations to the smart contracts and store the historical data received from the events that are needed in the application.

Despite developing an application that would be considered as a rather simple one on a traditional centralised manner, replicating the same exact set of features on a decentralised manner makes it so much more difficult. There are gas fees to bear in mind, there are security issues all the time as the smart contracts are transparent and thus are even more susceptible to hackers, and community of folks or users to entertain to make the tokenomics of the application to be more valuable.

Throughout the development of this project, there were so many companies that had launched their own DApps, boomed in market capitalisation, raised fund to scale and maintain their existing products. All these are the signs that the world is currently shifting from Web2 to Web3 an alarming rate. Even the STEPN that was mentioned just boomed and their telegram official group chat has over 120 thousands of people. In a nutshell, the world is really shifting to Web3 whether we like it or not.

3. Result

The result of this project is a full stack DApp that has frontend UIs that connects to the MetaMask EOA which can then invoke the smart contracts of Ethereum blockchain, backend that serves as endpoints to both frontend for requests and Ethereum blockchain for the events that it subscribed to, as well as a database to store processed historical result for ease of access.

The tokenomics of this application is somewhat appealing as the Game-fi elements in the game are not entertaining enough due to the time constraint and manpower limitation. The result of this application is also the proof that even day to day basic operations like walking and jogging can be leveraged to develop an application that is appealing enough to grow on people of this era. Lastly, innovations are desired in the DApp world and a different kind of thinking like Web3 needs to be cultivated to public so that more and more innovative developers can enter the Web3 world.

4. Conclusion

As a full stack developer in this project, the author experienced the use of blockchain in mobile application. The use of blockchain is just like normal backend but in a decentralised way in which the write requests cost ethers.

The author gained exposure and is glad to get to learn how decentralised system works and how to prove the invocation is done by whom using signing of transaction. Ultimately, the author developed a DApp that promotes health awareness by using their daily steps to exchange for tokens and realized that an incentivisation using blockchain technology in creating great tokenomics is what makes a DApp worth playing.

5. Recommendations

5.1. Merkle Distribution

The current implementation on minting the StepTokens is that the owner pays all the gas fee to mint for the users. However, this is not a practical way in the long run as the owner will just simply run out of ether to pay for it. A more practical way would be to use a merkle distribution method which allows a concise proof of membership. Any of the users that was granted the tokens can prove that they were granted and pay in ether to claim the tokens for themselves. This way, the owner no longer bears the gas fee to mint for the users.

5.2. Integration Test

The current application only covers unit testing for each component and does not implement integration testing. This resulted longer integration time than expected and hence it can be implemented in the future to help shorten and simplifies the integration cycle.

5.3. Continuous Integration/continuous deployment (CI/CD) workflow

Even though there is a git workflow in this project, there is no CI/CD workflow which automates the development and operation activities and in the long run could save a ton of time and hassle.

References

[1] Kaidong Wu^{1}, “An Empirical Study of Blockchain-based Decentralized Applications” 1Key Lab of High-Confidence Software Technology, MoE (Peking University), Beijing, China, 2019.

[2] Tian Min, Hanyi Wang, Yaoze Guo and Wei Cai, “Blockchain Games: A Survey” School of Science and Engineering, The Chinese University of Hong Kong, Shenzhen, China, 2019

[3] Aran Davies, “How Blockchain Could Redefine the Gaming Industry”, DevTeam.Space^{TM}, 2021. Accessed on: Sep. 17, 2021. [Online]. Available: https://www.devteam.space/blog/how-blockchain-could-redefine-the-gaming-industry/#4

[4] Mitchell Clark, “NFTs, explained”, The Verge, Aug 2021. Accessed on: Sep. 17, 2021. [Online]. Available: https://www.theverge.com/platform/amp/22310188/art-club

[5] Jerry H., Yawn Rong, “StepN Whitepaper”, Aug 2021. Accessed on: Mar. 17, 2022. [Online]. Available: https://www.whitepaper.stepn.com

[6] Nat Eliason, “Tokenomics 101: The Basics of Evaluating Cryptocurrencies – DeFriday #19”, Dec. 17, 2021. Accessed on: Mar. 20, 2022. [Online]. Available: https://every.to/almanack/tokenomics-101