COMP90019 Distributed Computing Project (25 Credit Points)

Software Development Project

Machine Learning Marketplace on Blockchain

Chenhan Ma – 823289

Department of Computing and Information Systems

The University of Melbourne

chenhanm@student.unimelb.edu.au

Supervised by

Dr. Gideon Aschwanden

June 2018

TABLE OF CONTENTS

TABLE OF CONTENTS I

LIST OF FIGURES II

LIST OF TABLES III

1. Introduction 1

2. Background 2

2.1 Blockchain Technology 2

2.1.1 Blockchain Concepts 2

2.1.2 Ethereum 3

2.1.3 Decentralized Application 3

2.1.4 Inter Planetary File System 4

2.2 Machine Learning 4

2.2.1 Neural Network Algorithm 4

2.2.2 Machine Learning Dataset 5

2.3 Project Motivation 6

2.4 Objective Statement 6

3. Requirement Analysis 7

3.1 Use Case 7

3.2 Requirement Analysis 8

3.2.1 Functional Requirements 8

3.2.2 Non-Functional Requirements 9

3.2.3 Domain Requirements 9

4. Application Design 10

4.1 Technologies Involved 10

4.1.1 Programming Language and Framework 10

4.1.2 Ethereum Development Dependencies 11

4.1.3 Web Development Dependencies 13

4.2 Dapp Architecture 15

4.3 Software Design Pattern 16

5. Application Implementation 17

5.1 Smart Contract Implementation 17

5.1.1 Smart Contract Pattern 17

5.1.2 Model Tree Data Structure 18

5.1.3 IPFS Contract 20

5.2 Web Application Implementation 20

5.3 Error Handling 27

6. Conclusion **28**
6.1 Contribution Summary 28
6.2 Future Directions 28
6.3 Conclusion 29

Reference **30**
Appendix **31**

LIST OF FIGURES

FIGURE 1. *DECENTRALIZED NETWORK REPRESENTATION* 2
FIGURE 2. *DECENTRALIZED APPLICATION COMPOSITION* 4
FIGURE 3. *NEURAL NETWORK* 5
FIGURE 4. *PERFORMANCE OF A NAÏVE BAYES MODEL* 5
FIGURE 5. *MACHINE LEARNING MARKETPLACE USER CASE* 7
FIGURE 6. *USER INTERACTION EXAMPLE* 8
FIGURE 7. *GANACHE INTERFACE* 12
FIGURE 8. *METAMASK NETWORK CONFIGURATION* 13
FIGURE 9. DAPP ARCHITECTURE 15
FIGURE 10. MODEL VIEW CONTROLLER PATTERN 16
FIGURE 11. FACTORY CONTRACT PATTERN 18
FIGURE 12. MODEL OBJECT ATTRIBUTES 18
FIGURE 13. MODEL TREE DATA STRUCTURE 19
FIGURE 14. GET ALL DETAILS OF A MODEL CALLING METHOD 19
FIGURE 15. IPFS CONTRACT 20
FIGURE 16. WEB3 INSTANCE 21
FIGURE 17. MARKETPLACE CONTRACT INSTANCE 21
FIGURE 18. SAMPLE DASHBOARD 22
FIGURE 19. CATEGORY SELECTION PAGE 23
FIGURE 20. MODEL LIST OF A USER 24
FIGURE 21. MODEL LIST OF A CATEGORY 24
FIGURE 22. MODEL DETAIL PAGE 25
FIGURE 23. SAMPLE MODEL DETAIL DATA ENCAPSULATION 25
FIGURE 24. UPLOAD MODEL COMPONENT 26
FIGURE 25. METAMASK INTERFACE WHEN UPLOADING MODEL 27
FIGURE 26. ERROR HANDLING 27

LIST OF TABLES

TABLE 1. *FRONT-END FRAMEWORK COMPARISON* 11
TABLE 2. WEB APPLICATION ROUTING 23

CHAPTER 1

Introduction

A new revolution has begun since a man called Satoshi Nakamoto invented Bitcoin: A Peer-to-Peer Electronic Cash System in 2008, sweeping across the industries all around the world. The invention of Bitcoin brought a new technology to world: Blockchain. After years of development of the technology and growth of the industry, currently there are thousands of projects that are built based on Blockchain technology. Ethereum is one of them, occupies the second place in terms of cryptocurrency market cap. The nature of this project is to build a Decentralized Application (DApp) on the Ethereum network, as the Ethereum is seen as the second generation of the Blockchain technology with the support of Smart Contract, while Bitcoin is considered as the first generation. Since Blockchain possesses the feature of decentralization, the purpose of this project is to integrate Blockchain technology with current Machine Learning and Artificial Intelligence processes: from collecting data to training models, by building a marketplace for people on both ends to improve the current processes.

Significantly, the Blockchain technology, community and industry are still in their early stages. Also, this project is not aimed to provide a complete software solution or a comprehensive tool for the proposed objective that can be put into production immediately. Instead, it is more tailored to a proof of concept and experimental work that pursues a combination of the decentralized network with the open-sourced ML and AI fields. Overall, the project includes three sub-ideas: ① Model iteration system, ② Machine Learning contest, ③ Dataset crowdsourcing.

This project will focus on the first idea: ML model iteration, a community that allows people to contribute and share their ML models. To implement this idea, a DApp is build using technologies such as Solidity, Web3, React.js. The design and implementation will be the main focus of this report. Another emphasis of this project is to tackle the challenges and problems raised through the development. Therefore, Chapter 6 will mainly illustrate the future directions and challenges of this project and the Blockchain technology involved.

CHAPTER 2

Background

This chapter gives a basic introduction and background to the technology involved in the project. More details in depth can be found in the resources in the Appendix.

2.1 Blockchain Technology

2.1.1 Blockchain Concepts

First of all, in order to form a fundamental understanding of the project, introduction of the Blockchain technology is indispensable. Blockchain is known as a distributed, decentralized and public ledger, runs on a peer to peer network while each node maintains a copy of the ledger. The structure of a Blockchain contains a chain of blocks, while every block is connected by the cryptographic hash of their previous block's transactions. This data structure most commonly used in the Blockchain is called Merkle Tree. [2]

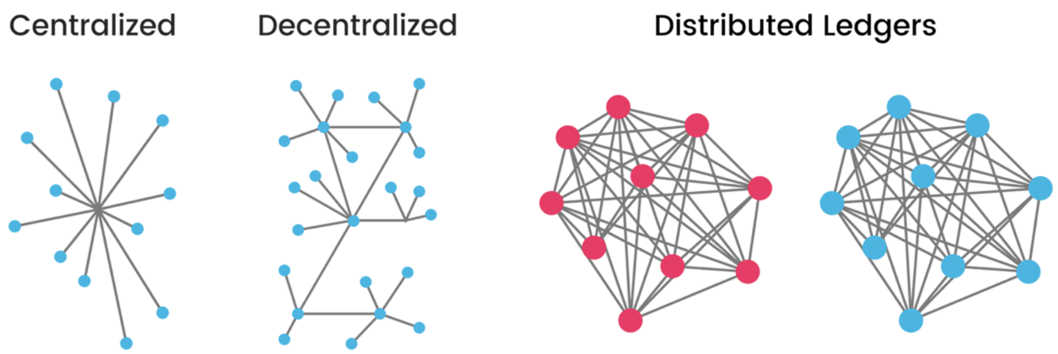


Figure 1. Decentralized Network Representation

Blockchain technology has many features, such as authenticity, transparency, immutability and decentralization [1], which are endowed by its base level technology. However, as the technology evolves, different kinds of Blockchain projects have different features and purposes, such as some projects aim to maintain a higher privacy level. Some problems that the public Blockchain technology wants to solve are: Byzantine problems and distributed system consensus.

In this project, we will mainly take advantage of the decentralization feature. Decentralization is also closely related to the concept of open source.

2.1.2 Ethereum

Ethereum is a decentralized Blockchain platform that provides the environment for the Smart Contract. The cryptocurrency used on Ethereum is called Ether. The main feature of Ethereum is that it provides a platform for Smart Contract, Decentralized Application and Autonomous Organizations (DAO) [10]. The Ethereum platform is for now the most popular choice for developer to develop a DApp, since Ethereum has its own programming language: Solidity and a suite of SDK supporting the development. In Ethereum development, there are three kinds of nets: one is Ethereum main net, which is the real Ethereum network; one is test net, which is a test network for developers to deploy their DApp; and the last one is a private network, runs on local computers. This project will host the application on the local computer for now.

2.1.3 Decentralized Application (DApp)

In a nutshell, DApp is the combination of Smart Contract with any front-end web technologies. This means instead of running a server as a back-end part of a web application, such as a NodeJS server lives on an AWS instance, the back-end now runs on the Ethereum Virtual Machine (EVM) using Solidity compiled Smart Contract.

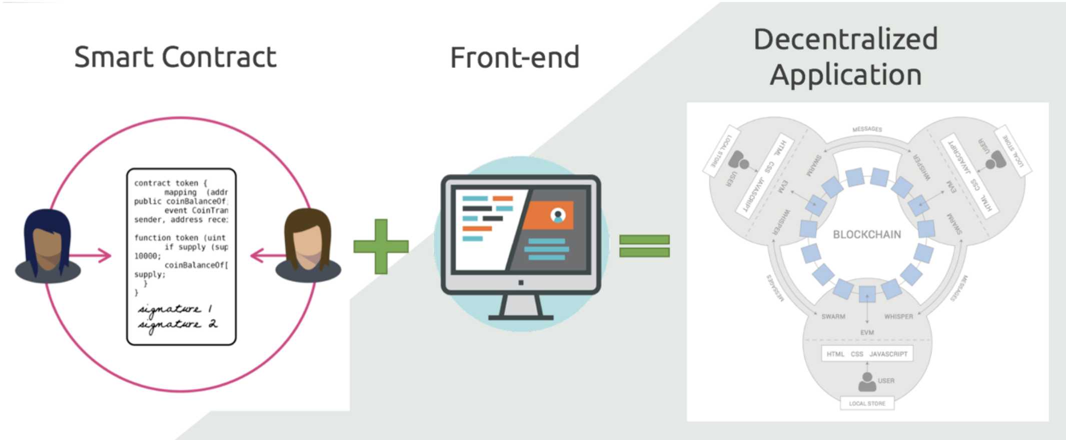


Figure 2. Decentralized Application Composition

2.1.4 Inter Planetary File System (IPFS)

IPFS is a peer to peer hypermedia protocol, or in some definitions, a distributed file system, able to exchange objects in a BitTorrent and Git like system which running nodes all over the world. It is similar to the normal web technology: HTTP, and have a potential of replacing HTTP. In this project, in order to minimize the data size stored on the chain, large data objects are uploaded to and retrieved from the IPFS network.

2.2 Machine Learning

2.2.1 Neural Network Algorithm

One of the key algorithms used in the ML and AI area is Neural Network (NN). A NN normally consist of an input layer, multiple hidden layers, and an output layer. Furthermore, the NN algorithm owns various variants such as Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN). A NN can be seen as a Machine Learning model, for example an RNN can be trained to analyze the sentiment of some sentences. In this project, a model will be referring to a Machine Learning model built by some kind of NN algorithm, which can be supervised learning or unsupervised learning.

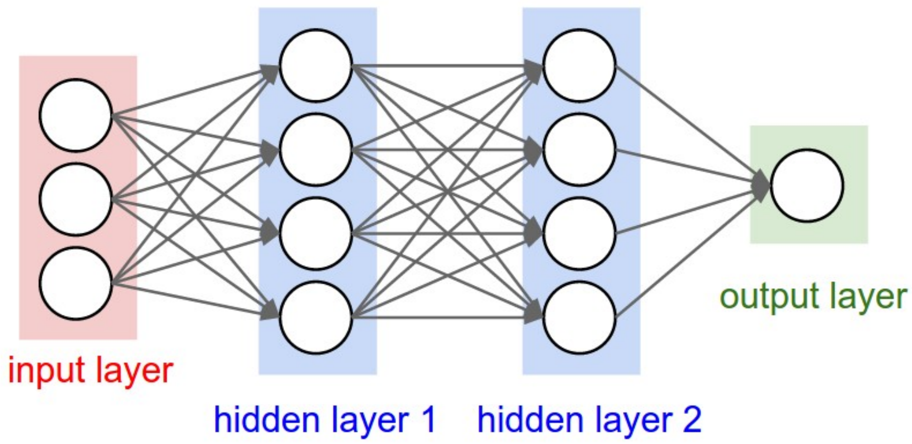


Figure 3. Neural Network

2.2.2 Machine Learning Dataset

In the Machine Learning area, algorithms can able to gain knowledge from the dataset by training them. Normally, a dataset used for ML is divided into training set, test set and validation set with customized percentage. Significantly, in normal situations, as the size of the dataset grows, a same algorithm is able to provide a better performance.

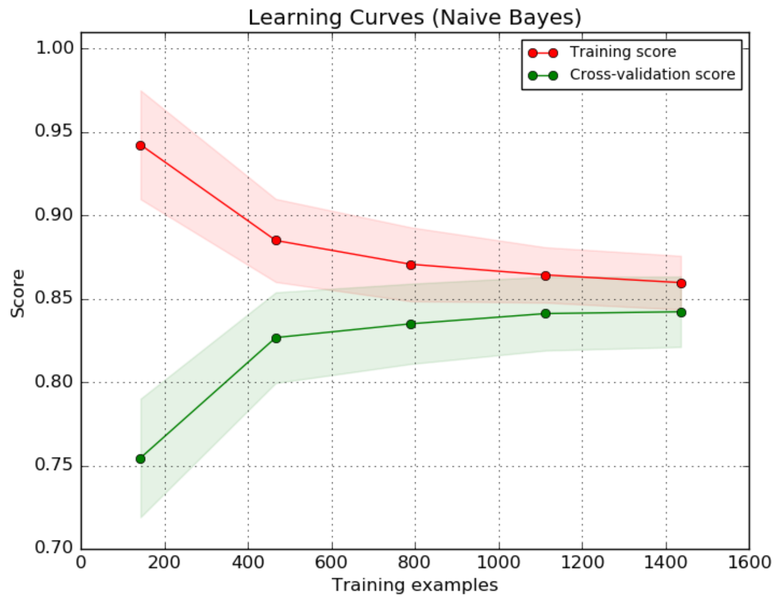


Figure 4. Performance of a Naïve Bayes Model

This brings the third idea introduced in the Introduction section. Based on [5], If the size of a medical dataset is large enough, the researchers and doctors may be able to provide a more accurate diagnosis based on ML.

2.3 Project motivation

As introduced in the previous sections, AI technology is changing the world, and Blockchain technology maybe the next one, since they are all bottom-level technologies essentially. Therefore, many people are trying to find out a combination of these two buzzwords. From the technology's point of view, conforming to the current open source community, the decentralized Blockchain network may play its role in helping the current ML and AI research processes. This leads an idea of building a marketplace of ML models.

2.4 Objective statement

As stated in the introduction, rather than provide an integrated solution, this project aims to provide a prototype: a DApp running on Ethereum network with a React.js as front-end, to implement the idea introduced above: a marketplace for sharing ML models. Challenges and infeasible ideas are frequently encountered through the process; however, the objective is to seek a reasonable approach under current technology allowance.

CHAPTER 3

Requirement Analysis

3.1 Use case

In this project, we will only demonstrate the implementation details of the model iteration system idea. However, in this section, in order to give an overview of the whole marketplace, all three ideas will be mentioned in the use case diagram.

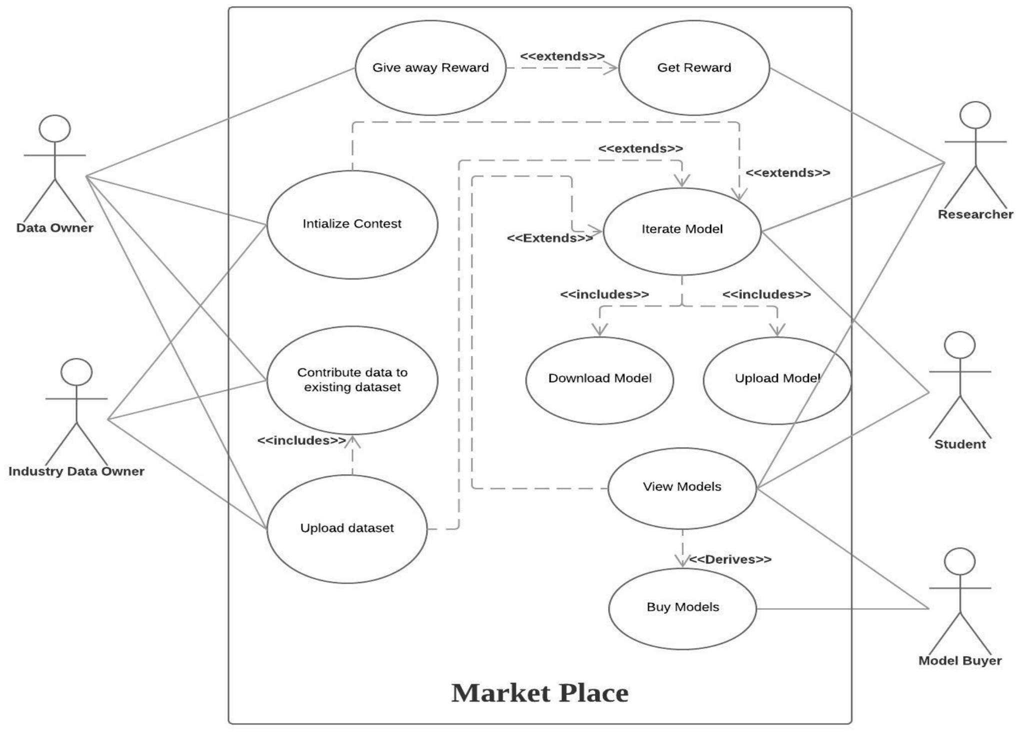


Figure 5. Machine Learning Marketplace User Case

As shown in the diagram, a full user interaction lifecycle with the marketplace can be described as the following flow:

1. The user that owns a dataset upload their dataset to the marketplace, initializes a contest and sets a reward for the model performs the best of

some criteria, e.g. accuracy. (In progress)

2. Others can also contribute their data of the same format to an existing dataset to make the relating algorithms more powerful. (Future work)
3. Researchers and students as another party of the marketplace, download the dataset, build and train their own ML model, then upload it to the marketplace. (Implemented)
4. People can also contribute to the models in marketplace, iterate it, similar to an open source community. (Implemented)
5. People that need to use a model can download the model (Implemented) and pay the owner an amount of token. (In progress)

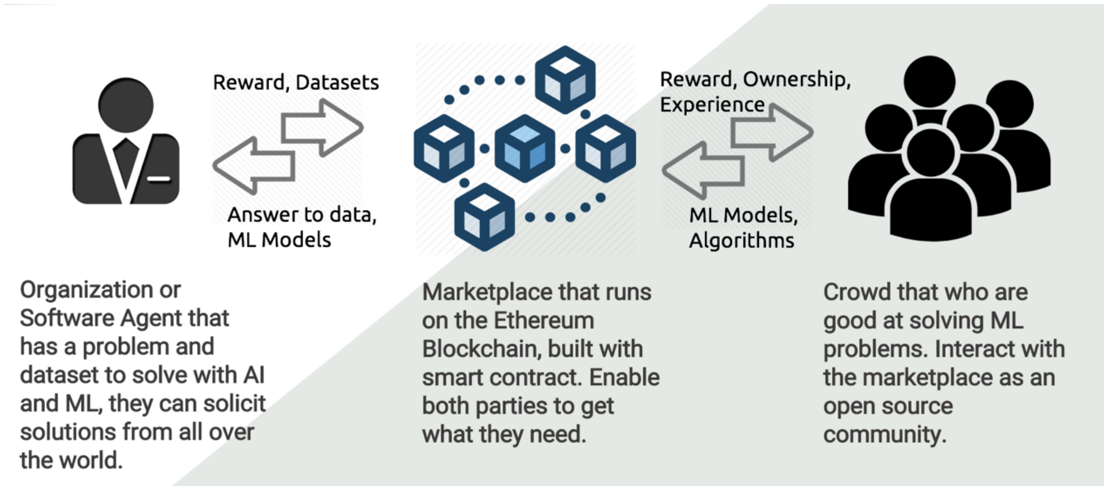


Figure 6. User Interaction Example

3.2 Requirement Analysis

In a software development, it is critical to conduct the requirement management to set an expectation and a plan of the project. Therefore, the following requirements are identified through the project development. Note that: Implemented requirements are marked as I, some of them are not implemented, marked as NI.

3.2.1 Functional Requirements:

Functional requirements are considered as the functionalities that the application should provide to the user in order to achieve the use case.

- The application should allow the user to browse a list of models. (I)
- The application should allow the user to view the model details. (I)
- The application should allow the user to download and upload the model to IPFS. (I)
- The application should allow the user to login with an Ethereum account. (I)
- The application should allow users to upload their dataset or initialize a contest. (NI)
- The application should autonomously send the reward to users. (NI)
- The application should be able to format and verify the new data added to the dataset. (NI)
- The application should be able to check a model's performance on a dataset. (NI)

3.2.2 Non-functional Requirements

Different from functional requirements, non-functional requirements refer to the feature or characteristic that the application should achieve.

- The application should have a good fault-tolerance feature.
- The application shouldn't write too much data to the Blockchain.
- The application should have an easy-to-use interface.
- The application should ensure security and prevent from attacks.

3.2.3 Domain Requirements

Domain requirements are the requirements that specify the domain of the application, it can be both function and non-functional.

- The application should have a front-end for user to interact with the Smart Contract.
- The application should minimize the Gas used when making a transaction on the Blockchain.
- The application should maintain fewer data on the Blockchain.

CHAPTER 4

Application Design

This Chapter will introduce the architecture of the DApp, technologies involved in the project and how data is transferred between front-end framework and the Smart Contract (Blockchain).

4.1 Technologies involved

4.1.1 Programming Language and Framework

As introduced in section 2.1.3, a DApp consists of Smart Contract and a front-end framework. In this project, Solidity is used to write the Smart Contract on Ethereum, and React.js is chosen as the front-end framework, which uses JSX to replace the normal JavaScript, HTML and CSS.

- **Solidity**
  Solidity is a contract-oriented, high-level static programing language that is used to implement smart contracts on Ethereum. In Solidity, term contract replaces the class in normal object-oriented programming languages. However, Solidity is still under development with many drawbacks, this project uses the latest version of Solidity, which is 0.4.24.

- **React.js**
  React.js is a JavaScript framework developed by Facebook for building web front-end sites. React.js is component-based, each part of the page is encapsulated into a component maintains its own state. And React.js uses JSX, a HTML-like syntax consists of JavaScript and HTML to write components. The advantages of React.js have readability and scalability, although it is difficult to learn.

Attribute	AngularJS	Angular 2	React
DOM	Regular DOM	Regular DOM	Virtual DOM
Learning Curve	High	Medium	Low
Packaging	Weak	Medium	Strong
Abstraction	Weak	Strong	Strong
Debugging General	Good HTML / Bad JS	Good JS/Good HTML	Good JS / Bad HTML
Debug Line NO	No	No	Yes
Unclosed Tag Mentioned?	No	No	Yes
Fails When?	Runtime	Runtime	Compile-Time
Binding	2 Way	2 Way	Uni-Directional
Templating	In HTML	In TypeScript Files	In JSX Files
Component Model	Weak	Strong	Medium
Building Mobile?	Ionic Framework	Ionic Framework	React Native
MVC	Yes	Yes	View Layer Only
Rendering	Client Side	Server Side	Server Side

Table 1. Front-end Framework Comparison [9]

4.1.2 Ethereum Development Dependencies

The Ethereum development topic involves a very board of segments. This section will only give a basic introduction of the dependencies used in this project, more information about them can be found in the Appendix.

- **Truffle versus Geth:**

  Truffle is a development environment, testing framework and asset pipeline for Ethereum [6]. It is always used for Smart Contract development on a testnet. In this project, we will main use the Truffle suite.

  Geth is a Go implementation of Ethereum protocol, it implements a full Ethereum node. What this means is that by using Geth, user will connect to the real Ethereum main net instead of a testnet.

  Normally, a fully developed and tested DApp will be migrated from Truffle suite to Geth.

- **Ganache-CLI (TestRPC):**

  Ganache is one of the Truffle suite, it is the replacement of TestRPC with a user interface instead of running it through command line. What Ganache does is that it creates a local Ethereum Virtual Machine, generates some fake accounts that contains some Ether for development.

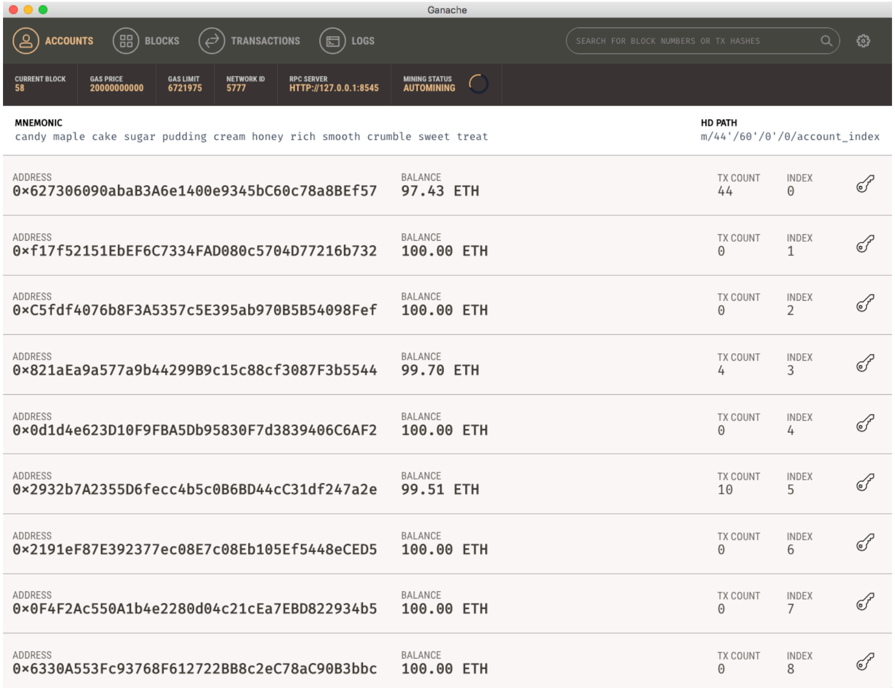
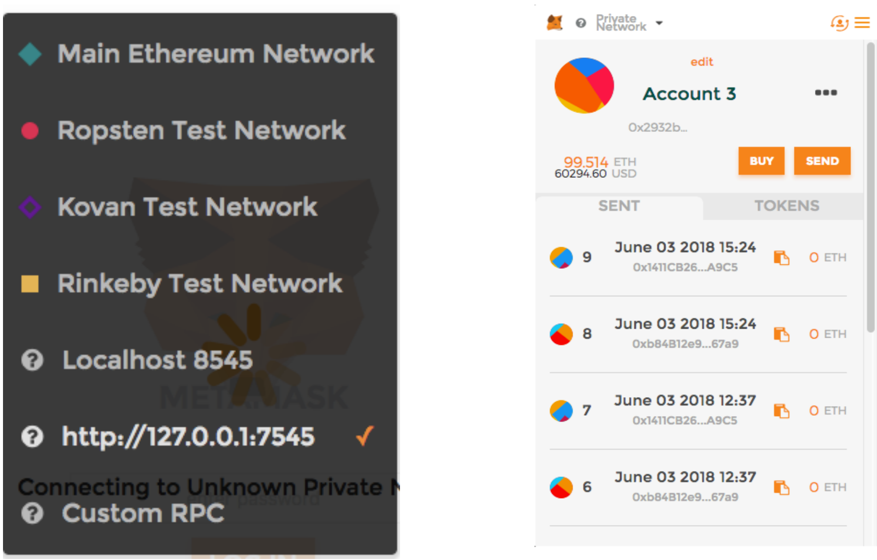


Figure 7. *Ganache Interface*

- **MetaMask:**

  MetaMask is a Google Chrome extension developed by Consensys. It can turn the browser into an Ethereum browser, allowing browser retrieve data from the local Ganache Ethereum Blockchain, and sign transactions. In this project, a transaction (Consumes Gas) initiated by the current signed-in account (Created by Ganache) is being processed through MetaMask to execute.



a). MetaMask Network Configure *b). MetaMask Account Transaction*

Figure 8. MetaMask Network Configuration

4.1.3 Web Development Dependencies

On the web development side of this project, except React.js for the main framework, various libraries and dependencies are involved in the development process. Some primary libraries and dependencies are introduced below:

- **Truffle-Contract:**

Truffle-contract is one of the most significant packages in this project. Normal Ethereum projects require developers to manually copy the Application Binary Interface (ABI, a JSON format interface that encode the solidity contracts) into the web application directory, which is not user-friendly and time consuming. With Truffle-contract, developers don't need to copy and paste the bytecode every time after re-deploy them. In addition, Truffle-contract supports synchronized transactions using promises instead of callbacks, which makes development much easier.

- **Web3.js:**

Web3.js is an Ethereum JavaScript API, which is the most important package used in the application. Web3.js provides various methods that allow you to interact with an Ethereum node. In this project, Web3.js is used as a package injected in the front-end part in order to communicate with our Smart Contract. By using Web3.js, we are able get Blockchain status, account details, and call the Smart Contract methods using JavaScript in our web application.

- **npm:**

npm is a JavaScript package management tool, allow JavaScript developers easily access the wheels built by other developers. npm improves the flexibility and scalability of the application.

- **webpack:**

Webpack is a module bundler responsible for configuration management. Webpack takes modules with dependencies with user defined entries and compile them based on user configuration into browser readable output. In addition, we also use webpack in order to hot-render the web application.

- **Ant-Design:**

Ant-Design is a UI design system with React.js-based implementation, developed by Alibaba. It provides basic web page components just like Bootstrap. Ant-Design is used as the main UI system in this project.

4.2 DApp Architecture

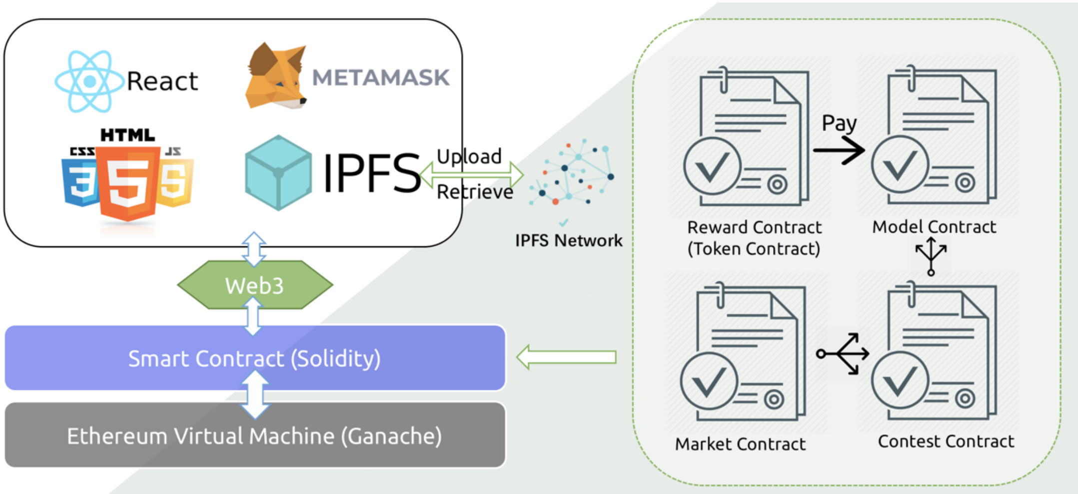


Figure 9. DApp Architecture

The diagram above demonstrates a general architecture of the Marketplace DApp. The system consists of 3 layers:

1) Ethereum Virtual Machine:
   Ganache runs a local Ethereum Blockchain as the environment for the Smart Contract. To some extent, EVM is similar to an EC2 instance server, and data is also recorded on the Blockchain.

2) Smart Contract:
   Smart Contracts written in Solidity are deployed on the Blockchain, and connect to the local Ethereum network by localhost and a port number.

3) Web Application:
   Web application part consists of React.js framework, client side Ethereum wallet: MetaMask, and other dependencies such as Webpack, Babel. The web application communicates with smart contracts and the Blockchain network using Web3.js.

4.3 Software Design Pattern

In modern software development, a widely used design pattern is called Model-View-Controller (MVC), shown in the diagram below. In this project, the model layer consists of the EVM and the Smart Contract, and they communicate through JSON-RPC. The React.js front-end includes the view, which is the rendered page shown to the user, and the controller, which is the smart contract invoking logic written in JavaScript using Web3.js.

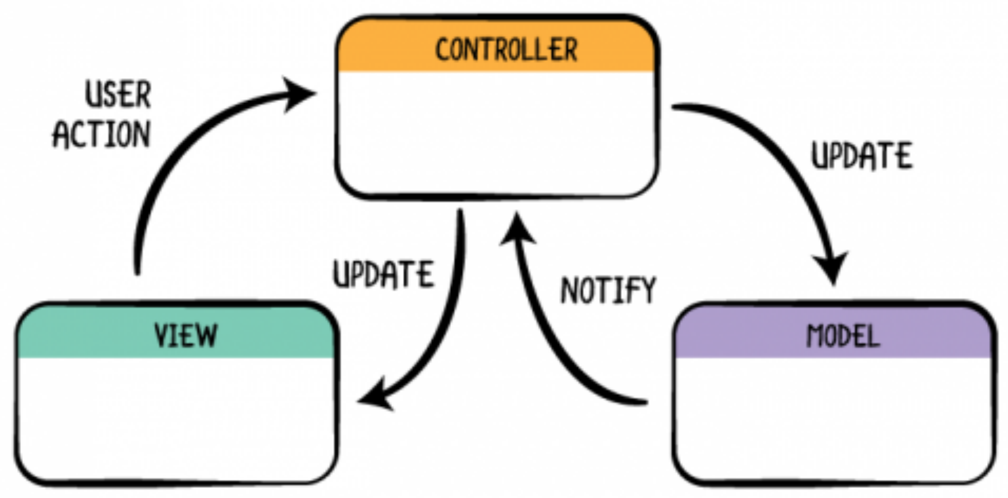


Figure 10. Model View Controller Pattern

CHAPTER 5

Implementation

This section will introduce the implementation details of the smart contracts in the project, such as, a tree-type data structure we created for model iteration system, and the smart contract architecture and relationship.

5.1 Smart Contract Implementation

5.1.1 Smart Contract Pattern

Since a Blockchain project normally does not have a database (e.g. MySQL, MongoDB) storing the data, the relationship of the objects is reflected embodied by the relationship of smart contracts.

In this DApp, the factory pattern is applied in order to implement the one-to-many relationship. Marketplace.sol holds contract Marketplace, which is the main entry and logic of the DApp. In the Marketplace contract, by holding an array of model IDs and a mapping which maps user to model IDs, we are able to retrieve all the Model contract addresses. And by calling the "create_model" method, we directly invoke the constructor of the Model contract in the Marketplace contract, to create a new Model instance, and store the address in our Marketplace contract global variables. The way we create a Contest Object is the same (Contest system front-end part still in progress).

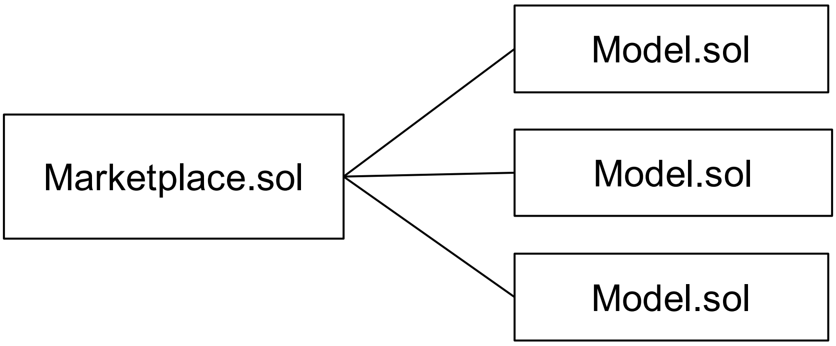


Figure 11. Factory Contract Pattern

5.1.2 Model Tree Data Structure

Specifically, for implementing the iteration mechanism, we propose a type of tree data structure to maintain the relationship and history of models (Machine Learning model object).

First of all, each Model object has the following attributes, shown in the figure 12 below.

```
address public owner;          // Model creator
bytes public ipfs_address;     // IPFS storage address of current model
int public id;                 // Unique identifier of models
string public name;            // Model name
string public description;     // Model description
int public parent;             // Parent model ID, 0 if current model is Genesis
int[] public children;         // List of subordinate model IDs
bool public genesis;           // Boolean if genesis model
int256 public accuracy;        // Float type model accuracy
string public category;        // Model belonged category
int public iterationLevel;     // Level of current model in its tree
int256 public price;           // Model price set by the creator
```

Figure 12. Model Object Attributes

Based on those attributes, we use the *Int* type model ID as a pointer to point the current model to another model. The data structure is described in the figure 13 below:

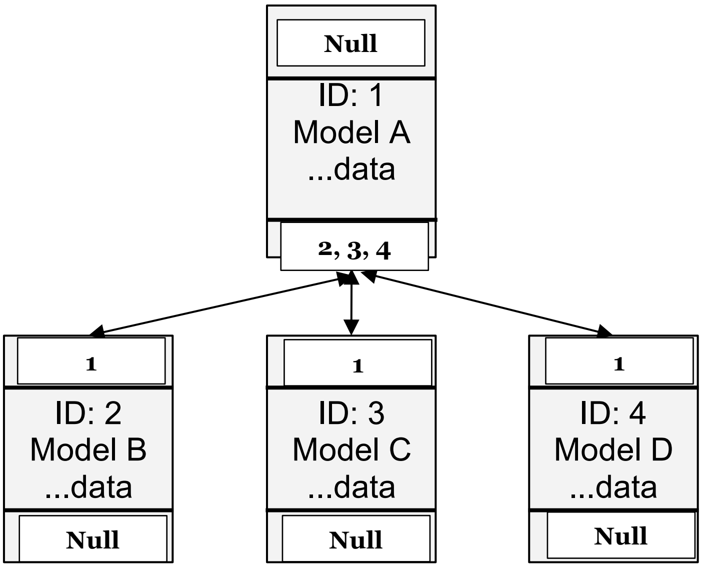


Figure 13. Model Tree Data Structure

Therefore, given a 20-byte Model contract address retrieved by an ID from the ID => Model mapping in Marketplace contract, we are able to call some get functions using the Marketplace instance from Web3.js. For example, method in Figure 14.a will call the method in 14.b and will return all the information about a Model object.

```
function get_model_all(int _id) public view returns (
  int id_,
  address owner_,
  string name_,
  int256 accuracy_,
  string category_,
  int price_,
  int parent_,
  bool genesis_,
  bytes ipfs_,
  int iterationLevel_,
  string description_)
{
  return models[_id].get_model_all();
}
```

a). Get Model Method in Marketplace

```
function get_model_all() public constant
    returns(
        int id_,
        address owner_,
        string name_,
        int256 accuracy_,
        string category_,
        int256 price_,
        int parent_,
        int[] children_,
        bool genesis_,
        bytes ipfs_,
        int iterationLevel_
    )
{
    return (
        id,
        owner,
        name,
        accuracy,
        category,
        price,
        parent,
        children,
        genesis,
        ipfs_address,
        iterationLevel
    );
}
```

b). Get Model Method in Model

Figure 14. Get All Details of a Model Calling Method

Since all the Model contract addresses and IDs are stored in the Marketplace contract, using different setter and getter functions, we are able to get any information through our model tree data structure.

5.1.3 IPFS Contract

The smart contract part for integrate with IPFS network only has two methods: one for set an IPFS hash, one for get an IPFS hash.

```
contract Ipfs {
    string ipfsHash;

    function sendHash(string x) public {
        ipfsHash = x;
    }

    function getHash() public view returns (string x) {
        return ipfsHash;
    }
}
```

Figure 15. IPFS Contract

5.2 Web Application Implementation

According to the requirement analysis, corresponding smart contracts methods have been implemented. Since React.js uses many components to represent different parts of a web page, different functionalities can also be encapsulated into components. The following sections will introduce the web application based on different components.

- **Web Application Objects**

React.js uses props and state to pass and store the data objects in different components. In this project, different components are closely related to each other, forming a nested relationship. For example, a model list page may lead to multiple model detail pages. There are two most important data objects almost exists in every component, that is web3 instance and a Marketplace contract instance. The Marketplace instance is a data object created by Truffle-Contract, enables the application to access variables and methods in the contract. The web3 instance is created by connecting to the truffle network using Web3.js, which enable us to call the Ethereum JavaScript API.

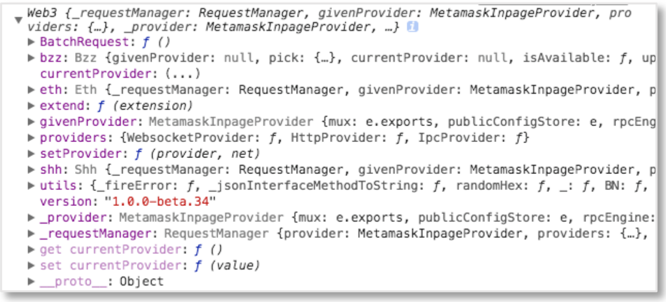


Figure 16. Web3 Instance

Figure 17. Marketplace Contract Instance

- **Dashboard and Router**

Dashboard is the main frame of the web application. It contains a header on the top, a navigation bar on the left and a main content area. Based on different paths directed to, React.js will render different components that matches the path, and also pass the parameters to the component using React Router.

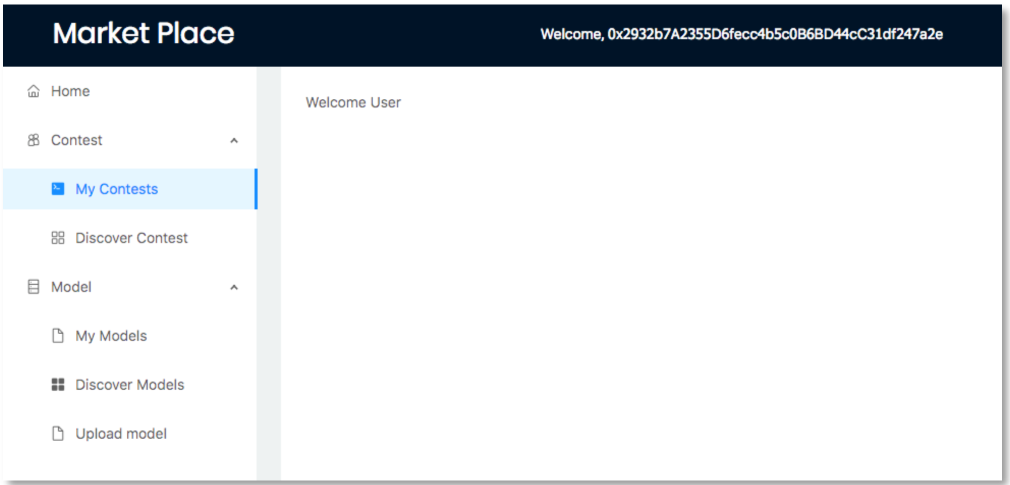


Figure 18. Sample Dashboard

The logic of React Router is implemented in the "dashboard.js" file, which contains the following routes:

Pathname	Component Description	Parameter Description	Development Status
"/"	Home page		Implemented
"/models"	All models, choose a category first		
"/models/:param /:paramKey"	A list of models, filtered by "param=paramKey"	":param" refers to "category", "children" or "user". :paramKey refers to "category name", "parent model ID" or "user address". Example: "/models/children/1" will render all the children of model with ID:1	Implemented

"/model/:modelID"	Model details	Render the details of model matched by ":modelID". Example: "model/1" will render the model details of model 1	Implemented
"/mycontest"	User's contest list		In progress
"/contests"	All contests		In progress

Table 2. Web Application Routing

- **Discover Models**

In this marketplace, Machine Learning models are categorized by their category first. Currently, there are two built-in categories: Image recognition and Medical diagnosis. We are expected to support more categories in the future. Figure 19 below shows the first page of "Discover Models" route, the user will be able to see a model list of the corresponding category after clicks on it.

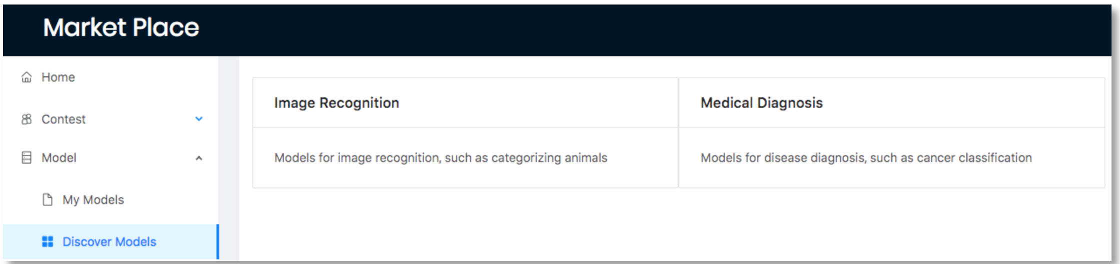


Figure 19. Category Selection Page

- **View List of Models**

Based on designed functionalities, the model list can be generated from a category, a user or a parent model id. Those filters will match the route path's ":param" parameter. Figure 20 below shows a sample model list rendered.

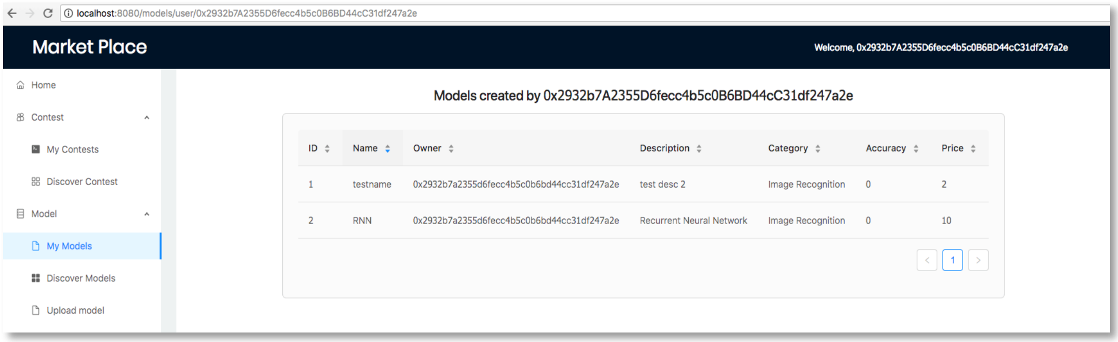


Figure 20. Model List of a User

By extracting the parameters in the route paths, we call the corresponding methods in the smart contract use Web3.js and Marketplace contract instance. After the data is returned from the Blockchain, they will be processed into a list of dictionaries ready to be rendered.

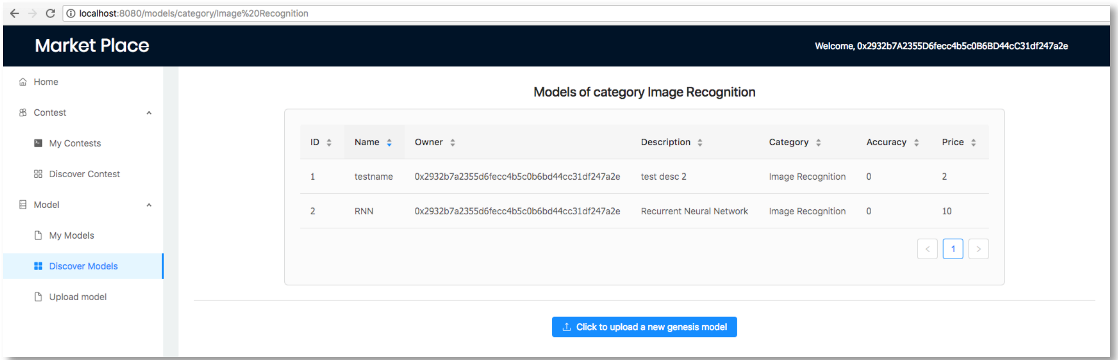


Figure 21. Model List of a Category

- **View Model Details**

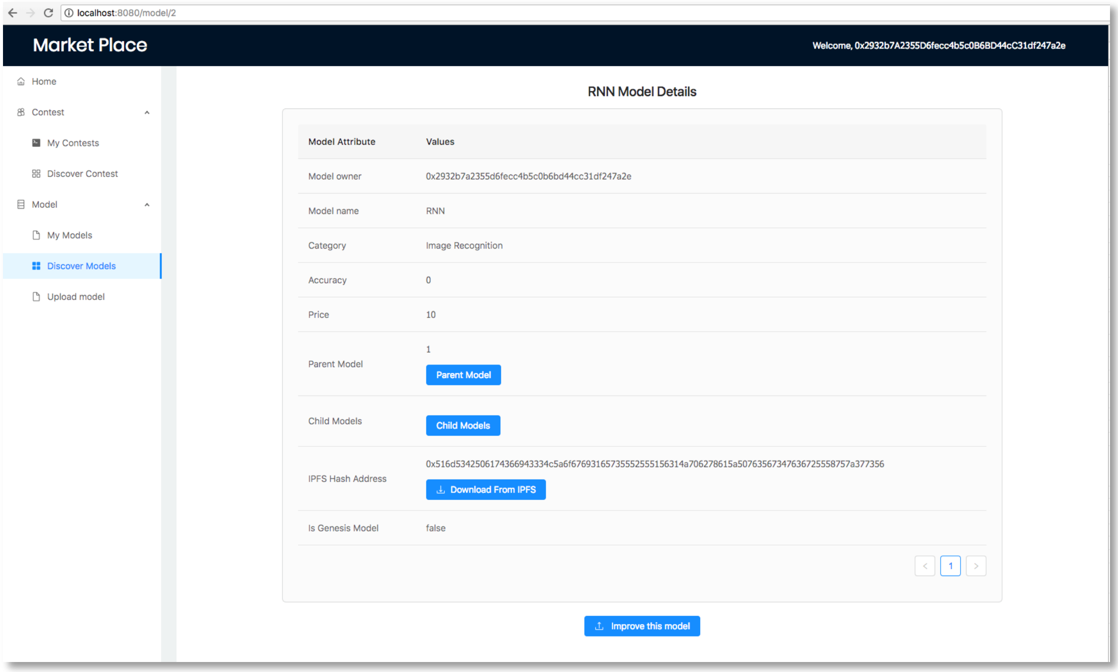


Figure 22. Model Detail Page

When a user clicks on the row in the model list table, React Router will direct the user to the model detail page, shown in Figure 22. From this page, user is able to check this model's parent model, child models or download this model from the IPFS network. Moreover, user can also choose to improve this model, and the current model will be the parent model of the subsequent model.

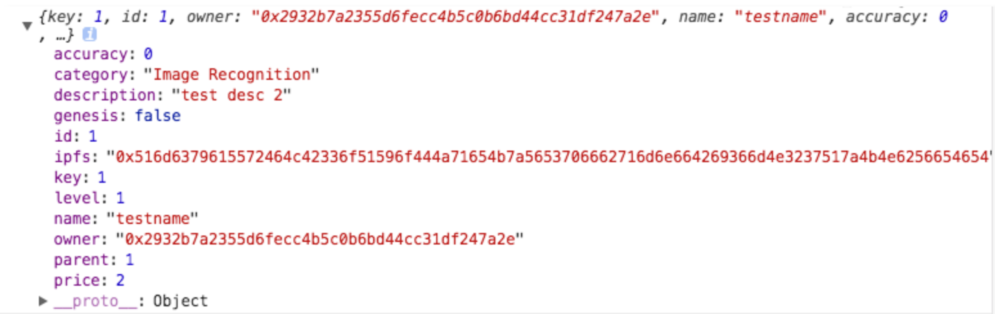


Figure 23. Sample Model Detail Data Encapsulation

- **Upload Model**

The upload model page can be triggered in the following two situations (For now, more use cases to be implemented):

- User clicks into a model category, user can choose to upload a genesis model for that category.
- In the model detail page, user is able to upload an improved model.

Parameters such as model id, parent model id, and bool type genesis will be handled by the logic automatically. However, the calculation of accuracy will require the integration of Machine Learning packages, such as Jupyter Notebook, and this will be one of the biggest challenges in this project.

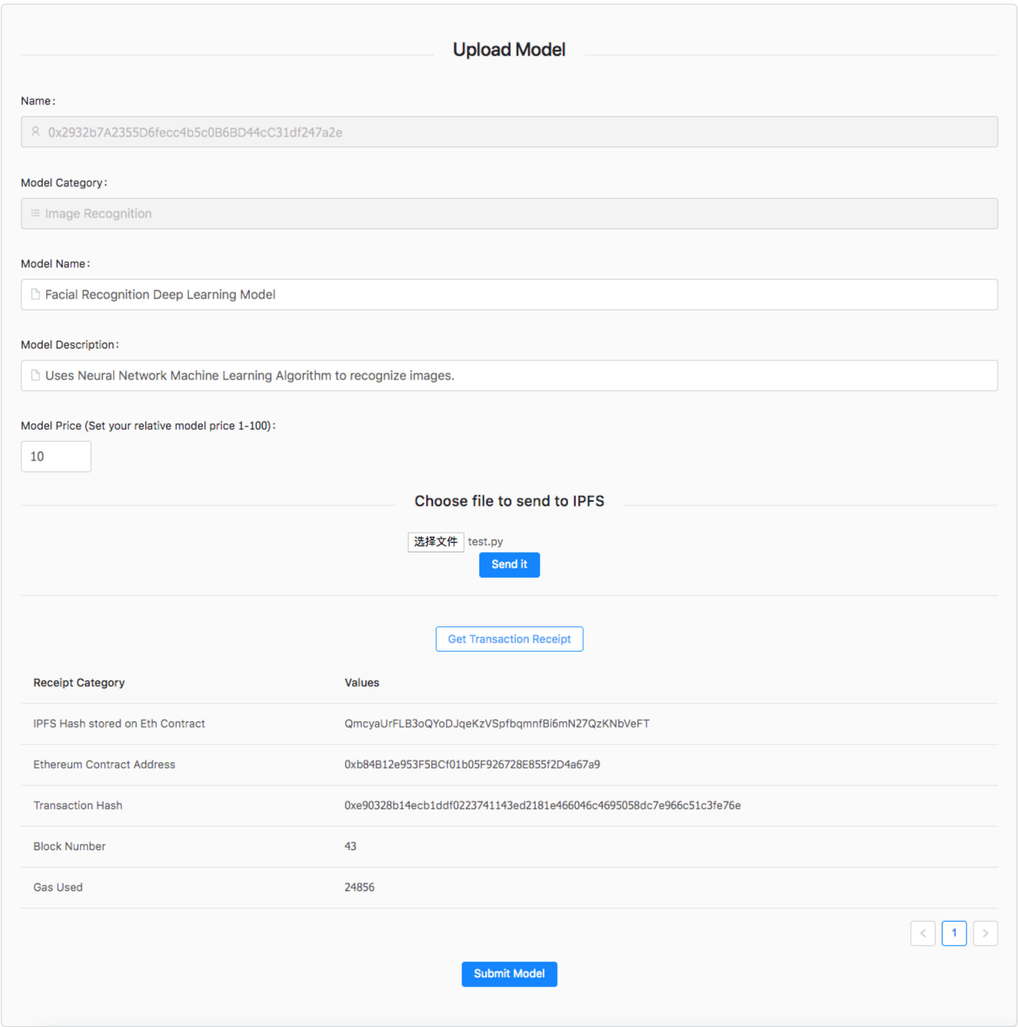
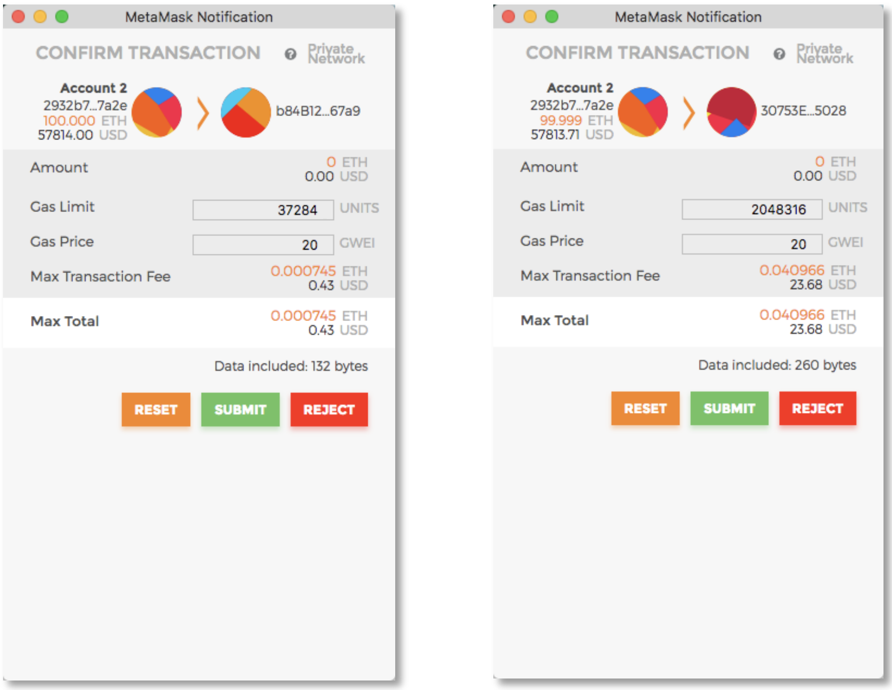


Figure 24. Upload Model Component

Figure 24 above shows the form for uploading a model, model creator and model category values are passed from previous paths and cannot be modified. User is required to input the name, description and price of the model, followed by uploading the model file to IPFS. After the user clicks on “Send it” button, Shown in Figure 25 below, MetaMask will pop up a notification requires the user action to accept the transaction. Finally, if all form fields are valid, MetaMask will ask the user again for submitting the model. However, uploading a model will cost approximately 0.05 Eth even though data size has been minimized.



a). Send Model to IPFS *b). Submit Model to Blockchain*

Figure 25. MetaMask Interface When Uploading Model

5.3 Error Handling

In order to make the application more robust and fault-tolerant, certain error handling functions are required. Generally speaking, the web application contains two kinds of functionalities: data display and data entry. In data entry components, the application needs to handle invalid form values. In data display components, the application needs to handle invalid table values or null data case. Below are some notifications and alerts when errors occur.

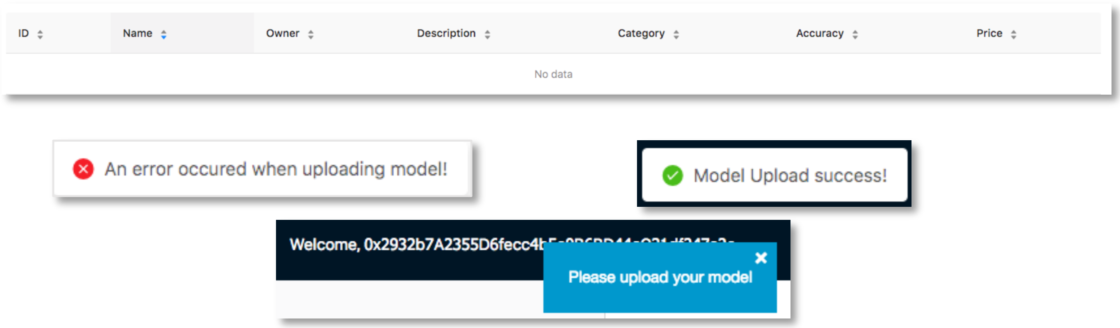


Figure 26. Error Handling

CHAPTER 6

Conclusion

6.1 Contribution Summary

The individual contribution through the project lifecycle can be summarized below:

1. Design feasible use cases and generate requirements after conducting research in relevant fields.
2. Design and implement smart contracts that satisfy the requirements and optimize performance.
3. Design the web application architecture and UI. Develop web application based on identified requirements.
4. Iterate functionalities and use cases in an Agile manner, based on challenges identified to optimize the rationality of use cases.

6.2 Future Directions

This section will introduce the work to be accomplished and discussion to be conducted in the future to improve current application, in both vertical (Improve current approach) and horizontal (Add new features) scaling direction. As briefly introduced in previous chapter, the directions on the two ideas have not been implemented will be introduced.

1) Implement ML model accuracy automatic judging system. (Requires integration with Python and Deep Learning framework API, such as TensorFlow)
2) Implement ML and AI contest system, a user is able to initiate a contest with some judging criteria, and other users are able to participate into the contest by uploading their model. (Reward system needs further consideration)

3) Implement dataset crowdsourcing system. Many details are to be defined, such as: how to merge the same type of datasets to a unified format, how to assure user data privacy, and how is the system works? Enterprise level or public faced?
4) Uploading datasets and models is a temporary approach to avoid store data on the Blockchain. Is there a better solution to improve performance, such as scalability, availability?
5) Holding DApp on Ethereum is a good approach for now. If consider the future case and constraints on the Ethereum, should we migrate the DApp to our own chain?
6) Since a DApp once deployed cannot be modified later, comprehensive unit tests are required to ensure its fault-tolerance, either written in Solidity or JavaScript.

6.3 Conclusion

To conclude, motivated by the decentralization idea of Blockchain and the passion for ML and AI algorithms, we applied cutting-edge technologies to build an Ethereum Blockchain Decentralized Application. The application is designed and developed as a prototype, which is a marketplace and allow users to share and improve their ML models. This project demonstrates our idea of the future decentralized world, in terms of ML and AI field. And the initial project objective has been fulfilled, although questionable cases still exist. Hopefully this project's ideas will have some similarities against the technology 5 years later. Moreover, vast discussions were held around the topic of the use cases and the feasibility of the DApp, based on current technology level and resources on hand.

Reference

[1] En.wikipedia.org. (2018). Blockchain. [online] Available at: https://en.wikipedia.org/wiki/Blockchain [Accessed 3 Jun. 2018].

[2] Blockgeeks. (2018). *What is Blockchain Technology? A Step-by-Step Guide for Beginners*. [online] Available at: https://blockgeeks.com/guides/what-is-blockchain-technology/ [Accessed 3 Jun. 2018].

[3] Nakamoto, Satoshi (31 October 2008). "Bitcoin: A Peer-to-Peer Electronic Cash System" (PDF). bitcoin.org. Archived (PDF) from the original on 20 March 2014. Retrieved 28 April 2014.

[4] Benet, Juan. "IPFS-content addressed, versioned, P2P file system." arXiv preprint arXiv:1407.3561 (2014).

[5] Nature.com. (2018). AI researchers embrace Bitcoin technology to share medical data. [online] Available at: https://www.nature.com/articles/d41586-018-02641-7 [Accessed 3 Jun. 2018].

[6] GitHub. (2018). *ethereum/wiki*. [online] Available at: https://github.com/ethereum/wiki/wiki/White-Paper [Accessed 3 Jun. 2018].

[7] GitHub. (2018). ipfs/ipfs. [online] Available at: https://github.com/ipfs/ipfs [Accessed 3 Jun. 2018].

[8] Truffle Suite. (2018). Documentation | Truffle Suite. [online] Available at: http://truffleframework.com/docs/ [Accessed 3 Jun. 2018].

[9] Quora.com. (2018). Which is better 'AngularJS or Angular2 or React Js' and why? - Quora. [online] Available at: https://www.quora.com/Which-is-better-AngularJS-or-Angular2-or-React-Js-and-why [Accessed 3 Jun. 2018].

[10] Hacker Noon. (2018). An Introduction to Ethereum – Hacker Noon. [online] Available at: https://hackernoon.com/an-introduction-to-ethereum-68fb9b95fc62 [Accessed 3 Jun. 2018].

Appendix

1. Source code and installation guide:

- GitHub:https://github.com/matutu22/Machine-Learning-Marketplace-on-Ethereum
- Installation guide: See the Readme file in the Repo. (May take some effort)

2. Blockchain (Ethereum) Development Dependencies:

- MetaMask: https://metamask.io/
- Truffle: https://github.com/trufflesuite/truffle
- Ganache: http://truffleframework.com/ganache/
- Truffle-Contract: https://github.com/trufflesuite/truffle-contract
- Web3.js: https://github.com/ethereum/web3.js/
- Ethereum Wiki: https://github.com/ethereum/wiki/wiki
- Ethereum JSON RPC: https://github.com/ethereum/wiki/wiki/JSON-RPC
- Ethereum JS API: https://github.com/ethereum/wiki/wiki/JavaScript-API
- IPFS: https://github.com/ipfs/ipfs
- Ethereum Whitepaper: https://github.com/ethereum/wiki/wiki/White-Paper
- Infura IPFS network: https://infura.io/
- Solidity: http://solidity.readthedocs.io/en/v0.4.24/
- Remix: http://remix.ethereum.org/

3. Web Development Packages and SDK:

- React.js: https://reactjs.org/
- React Router: https://github.com/ReactTraining/react-router
- Webpack: https://github.com/webpack/webpack
- Babel: https://babeljs.io/
- Ant Design: https://ant.design/