Decentralized Algorithm Trading Platform

Project Report

Team: Playing with Mud

Yunqiu Xu (z5096489), Xingshi Zhang (z5101900)

Yichen Zhu (z5098663),Qihai Shuai (z5119437)

25/05/2018

Content

Figures

Tables

1. Background

Recently, artificial intelligence and machine learning have attracted increasing attention and achieved great success in both research community and industry. With more research, machine learning (ML) systems have been able to surpass humans in many specific problems such as face identification [1] , speech recognition [2] and game playing [3]. There is a strong demand for the users to get access to suitable model directly so that they can test their ideas fastly. However, it requires long time and memory to train a suitable machine learning model, especially for deep learning, which is hard to choose optimal hyper parameter set and is expensive to build GPU cluster for computing. In terms of companies which can produce good machine learning models, they have the demand to make profit through selling models with improved efficiency and new capabilities. Thus it will be promising to build an trading platform where those who are good at solving machine learning problems can directly monetize their skillset, and where any organization or software agent that has a problem to solve with AI can solicit solutions from all over the world.

Blockchain technology [4,5] enables the creation of decentralized currencies, self-executing digital contracts (smart contracts) and intelligent assets that can be controlled over the Internet (smart property). Blockchain technology represents the next step in the peer-to-peer economy. The blockchain is a distributed, shared, encrypted database that serves as an irreversible and incorruptible public repository of information. By allowing people to transfer a unique piece of digital property or data to others, in a safe, secure, and immutable way, the technology can create: digital currencies that are not backed by any governmental body, self-enforcing digital contracts (called smart contracts), whose execution does not require any human intervention. A decentralized market uses the identity and reputation system as a base [6]. The marketplace would eliminate tremendous amounts of friction, lower costs for customer acquisition and offer a new income stream for consumers. Since machine learning is purely software and training it doesn't require interacting with any physical systems, we propose the idea to build a machine learning model store using blockchain for coordination betweens users and using cryptocurrency for payment.

2. Related Work

There has been a lot of traditional outsourcing platforms [7] such as Guru and Elance which are great for both the people who want to hire and want to be hired. Machine learning model, as a kind of software, can also be achieved by posting requests on such platforms. The major drawback is that these platforms are comprehensive that the requests range from programming to graphic designing that only a small part of them are machine learning related tasks, which can be too complex for those who want to focus on machine learning only. Another drawback is that we need a mechanism to evaluate the performance of model automatically, while traditional outsourcing websites can only provide clients platforms to find freelancers. Furthermore, traditional outsourcing platforms are centralized, and we want to make users interact with each other directly.

Some competition platforms, such as Kaggle [8], can also provide best models for predicting and describing the datasets uploaded by companies and users. Although models with high performance can be acquired given a dataset, it would be time-consuming and expensive to

hold a competition. Thus it goes against the aim to get suitable model quickly, especially for independent researchers and those from small institutes.

Blockchain has been used to build decentralized marketplace, where identities and reputations are not required to create a transaction. Some related applications are already implemented including virtual pets shop [9], financial trading platform [10] and decentralized app store [11, 12]. While only a few of them are related to the exchange of machine learning models, Danku [13], which is a new blockchain-based protocol for evaluating and purchasing ML models on a public blockchain such as Ethereum, is closest to our project. The smart contract of Danku allows anyone to post a dataset, an evaluation function, and a monetary reward for anyone who can provide the best trained machine learning model for the data. Participants train deep neural networks to model the data, and submit their trained networks to the blockchain. The blockchain executes these neural network models to evaluate submissions, and ensure that payment goes to the best model. Compared with Danku, our idea has 3 major differences: (1) Instead of evaluating models on blockchain directly, we add an online judge module to run the test code, then the result will be sent to blockchain. This will cost less computing resource on blockchain, i.e. save gas on Etheurum. (2) Currently Danku is more like a model-related outsource platform that the user can exchange models by posting and accepting requests. Since some machine learning models have decent generalization ability and can be fine-tuned into new domain without too many modifications, we add a new module to enable users to buy and sell models like common transactions. (3) To some extent, pure decentralization is impractical for that we still need to store some personal information into places other than blockchain to protect privacy. We also need to apply cloud storage to store big files such as datasets and models. So in our project we combine the blockchain with traditional database system.

3. Project Overview

The aim of our project is to build a trading platform specific to machine learning models. This platform combines the functionalities of online shop (short term transaction) and outsourcing platform (long term transaction), that one user can have 4 different roles: seller, buyer, client and freelancer.

The interaction between seller and buyer is for short term transaction, that they can exchange models directly like a general online shop, as shown in Figure 1.

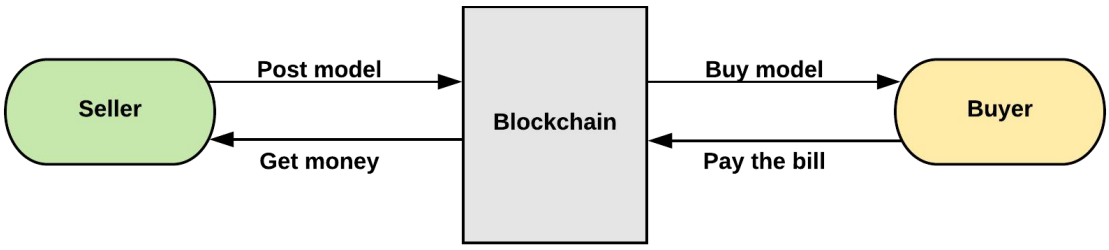

Figure 1 The short term transaction (interaction between seller and buyer)

The interaction between client and freelancer is for long term transaction, that a client can post requests to seek suitable models, while a freelancer can accept these requests and provide solutions, as shown in Figure 2. This goes beyond current outsouring platform that a

request can be accepted by more than one freelancers, which can solve the problem more efficiently via collective intelligence and competitive environment.

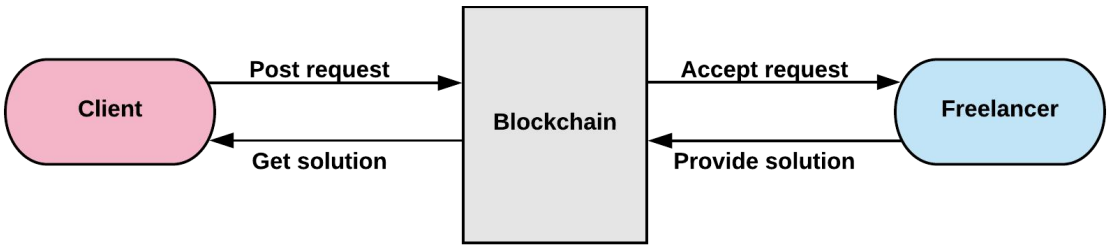

Figure 2 The long term transaction (interaction between client and freelancer)

Compared with traditional platforms which are totally based on database, we decentralize our platform by integrating blockchain. As shown in Figure 3, we only use two tables for database part that one is for personal details such as username and password, another is for shopping cart, where only necessary details of an item (e.g. seller's address and index) is stored, while all other information will be stored on blockchain. In this way we can make our platform more efficient that it the interaction does not require third party's intervention. Since the identity and reputation have already built, as said in Ethereum white paper, our platform can be performed anonymously that a user does not need to know whose model he has bought or who fulfilled his request, which makes the transaction simpler that they can focus on the models themselves. Another consideration may be that machine learning models are suitable to be exchanged online for that they are pure software thus do not need delivery.

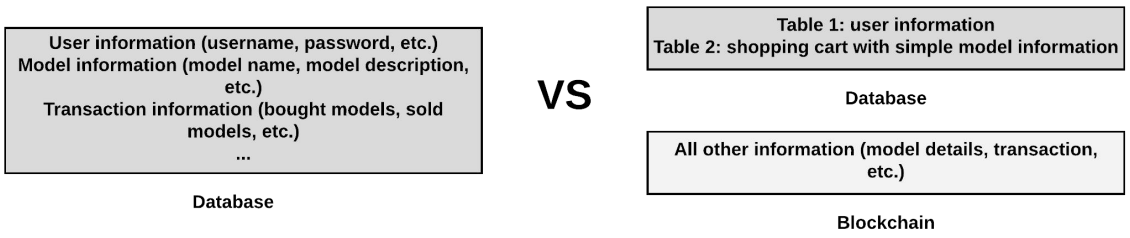

Figure 3 Comparison between traditional platform (left) and ours (right)

The project architecture is shown in Figure 4, that a user can choose either of the roles after logging in. These functionalities are built separately for clarification and simplification, e.g. searching models and searching requests are regarded as two different operations. The functional details can be seen in Section 4.

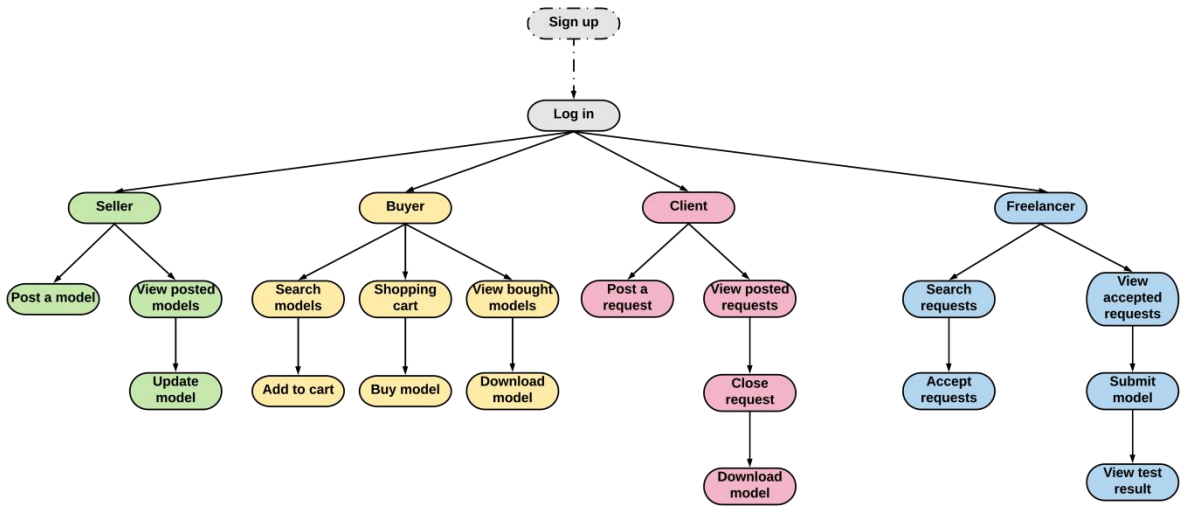

Figure 4 Project architecture

4. Functionalities

The functional details consists of basic operations (e.g. sign up, log in), and the operations corresponding to each role.

4.1. Basic Operations

The basic operations of a user consists of signing up, logging in and selecting roles. The user can create a new account by inputting username, password and user's address (MetaMask address, for our project is address on Ganache), these inputs will be posted to blockchain to generate four unique contracts for this user: seller contract, buyer contract, client contract and freelancer contract, then user inputs as well as the contract address will be saved in the first table of database. The whole process is shown in Figure 5 and Table 1 shows the first database table.

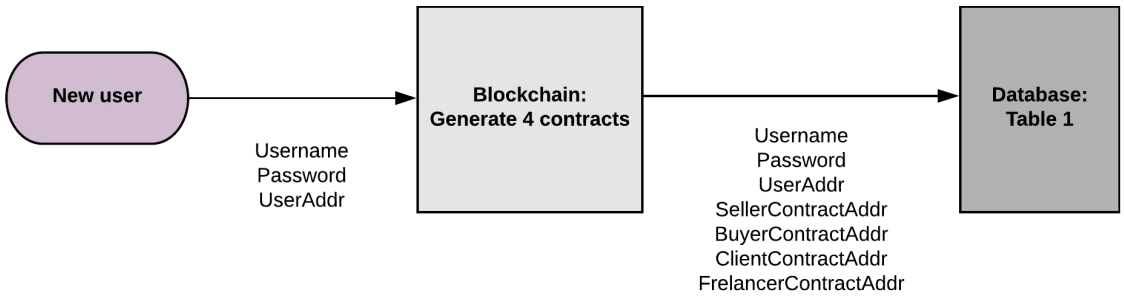

Figure 5 User creates new account

Table 1 The first table in database

Attribute	Type	Description
UserID	Integer	Unique, generated during inserting
UserName	String	Unique

Password	String	
UserAddr	String	Unique, user's address (MetaMask or Ganache)
SellerContAddr	String	Unique, seller contract's address
BuyerContAddr	String	Unique, buyer contract's address
ClientContAddr	String	Unique, client contract's address
FreelancerContAddr	String	Unique, freelancer contract's address

Figure 6 shows the logging in process. As the personal details has been stored in database, here the user only need to input username and password, which will be checked in database. If the inputs are correctly matched, the user's information will be used to generate cookie, then the user can get access to starting page and be able to choose roles.

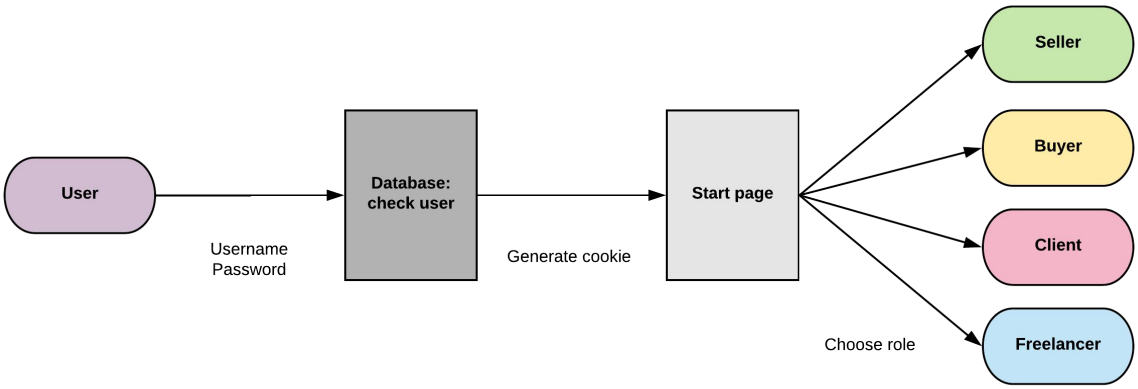

Figure 6 User logs in the platform

4.2. Seller Branch

When the user is acting as a seller, he will be able to post a new model and view posted models. For posted models, he can update the description and price, and be able to download the model. Most of the functions used in this branch are from seller contract, which is shown in Table 2:

Table 2 Functions for seller contract

Function	Input	Description
postModel()	Name, description, path, price, time	Post a new model
updateModelDesccription()	Model's index, new description	Find a model and update its description
updateModel()	Model's index, new price	Find a model and update its description
getModelCount()	-	Count the number of models in this contract
getModelByIndex()	Model's index for this contract	Get a model's information

A user can get access to a contract via his user address and the address of this contract, then he can use the functions. Figure 7 shows how the functions in seller contract be used, for other contracts, the process is similar.

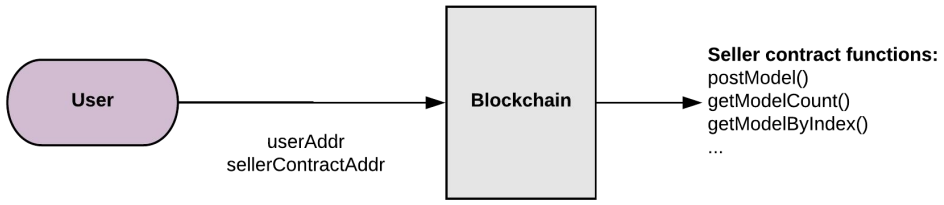

Figure 7 User gets access to seller contract

When posting a new model, a user can upload an attachment first, then this file will be saved in server and represented as path. Other inputs consist of the name, description, price of this model. A time stamp will be generated automatically to show the modified time. Then the user address and corresponded seller contract address will be extracted from cookie to get access to seller contract, and then function postModel() will be called to generate and save model object in seller contract. The process is shown in Figure 8 and Table 3 shows a model object's structure in contract.

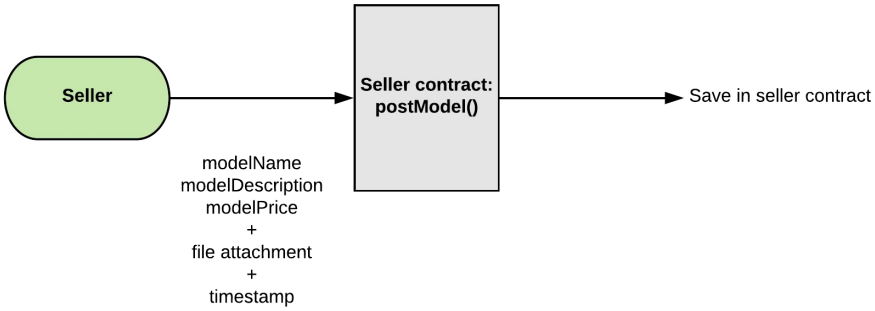

Figure 8 Seller posts a new model

Table 3 The structure of a model object (for seller and buyer)

Attribute	Type	Description
modelName	string	Model's name
modelDescription	string	Model's description
modelAddr	string	Model's path
modelPrice	uint	Model's price
modelTime	uint256	Modified time

The seller can view all posted models, the process consists of getting the number of models saved in this seller contract and showing the information of each model, as shown in Figure 9. For these posted models, the seller can update the description and price, as shown in Figure 10. After modification, the model time will also be changed automatically.

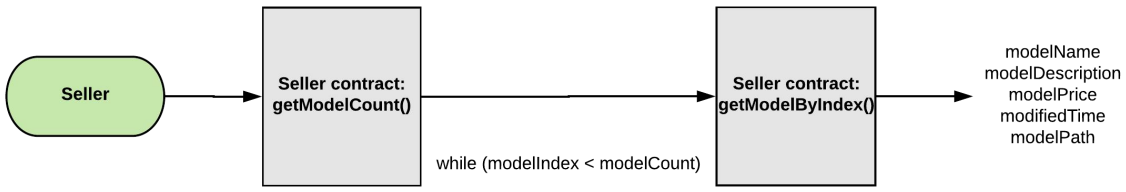

Figure 9 Seller views posted models

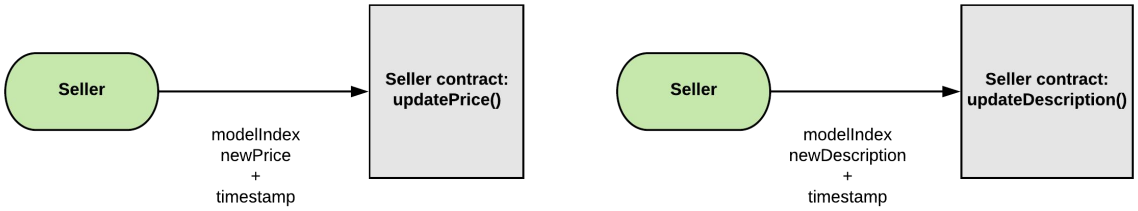

Figure 10 Seller updates a model

4.3. Buyer Branch

When the user is acting as a buyer, he can search the models by name or view all models on the platform. For those models he want to buy, he can add them to shopping cart then buy purchase the models. Then these bought models can be viewed and downloaded. Most of the functions used in this branch are from buyer contract, which is shown in Table 4. As there are interactions between seller and buyer (e.g. buy a model), the seller contract will also be used here.

Table 4 Functions for buyer contract

Function	Input	Description
buyModel()	Seller's contract address, model's name, description, path, price, time	Buy a model from a seller
getModelCount()	-	Count the number of models in this contract
getModelByIndex()	Model's index for this contract	Get a model's information
checkRepeat()	A model's path	Check whether this model has been bought (already exist in this contract)

The search process is shown in Figure 11, if there is no keyword, all models posted on this platform will be listed. The information visible to the buyer consists of model's status, description, price and modified time. There are 3 status: "my own model", "already bought" and "available model" that only those still available can be added to shopping cart. When a model is added to shopping cart (Figure 12), only simple information will be extracted and saved in database (Table 5), while detailed information will still be viewed by get accessing to this seller's contract, as shown in Figure 13. In this way the information of model can be

accessed in real time that if a seller update the model, the changes can also be checked for those in one's shopping cart.

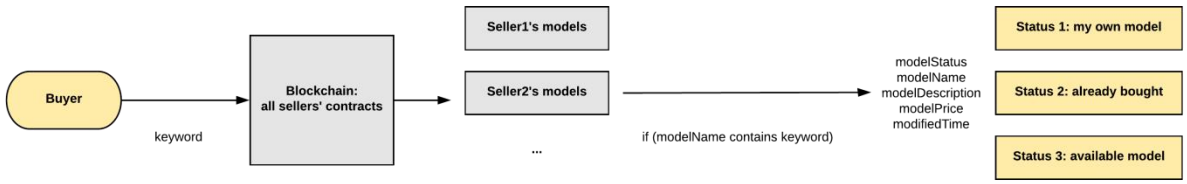

Figure 11 Buyer searches models

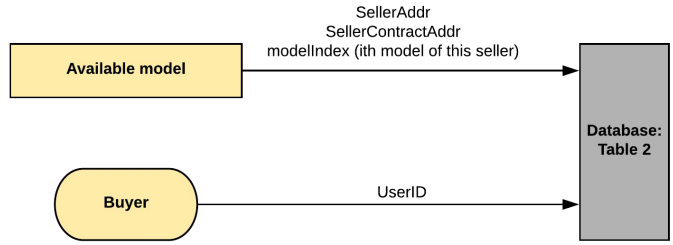

Figure 12 Buyer adds a model to shopping cart

Table 5 The second table in database

Attribute	Type	Description
UserID	Integer	Buyer's user ID
SellerAddr	String	Seller's user address
SellerContAddr	String	Seller's contract address
modelIndex	Integer	The model's index in seller's contract (i.e. this is the ith model posted by this seller)

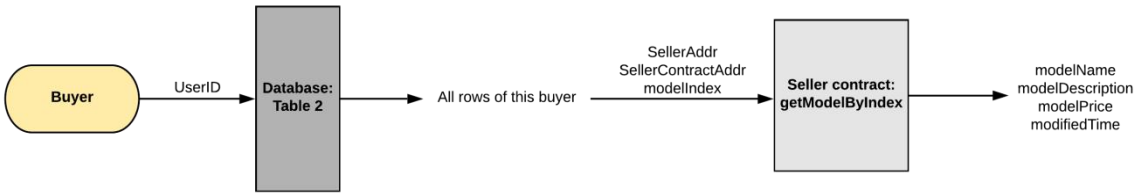

Figure 13 Buyer views shopping cart

The purchasing operation is available for those in shopping cart, which is shown in Figure 14. When a buyer buys a model, the model details is collected first from seller, and used for creating a new model object and saving in buyer's contract. Then the money will be transferred from buyer to seller through buyer's contract. If a model has been bought, it will be deleted from shopping cart. This model is visible and can be downloaded when the buyer wants to view all bought models (Figure 15). Thus the buy-and-sell interaction is finished, due to anonymity the buyer does not need to know who is the model's author while the seller does not need to know who bought his model, but they can view balance to know the changes.

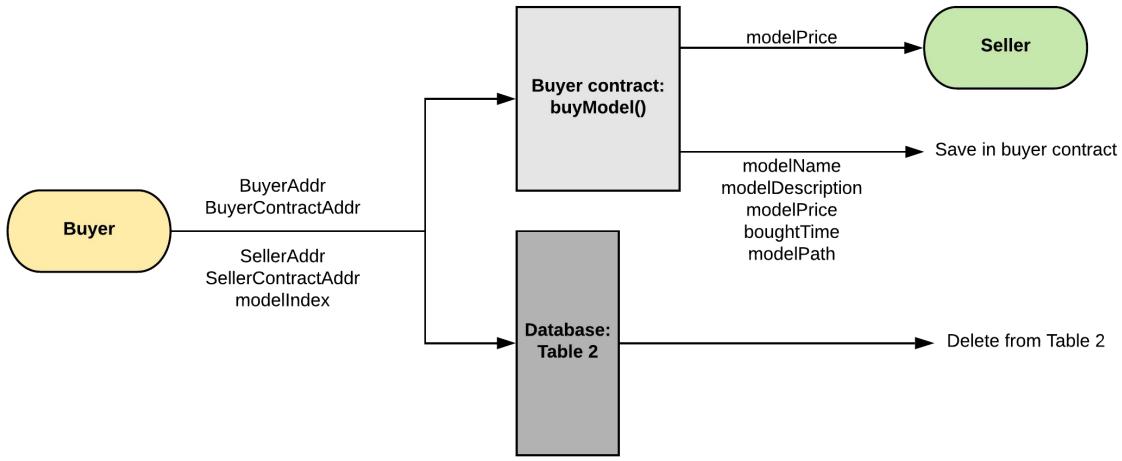

Figure 14 Buyer buys a model

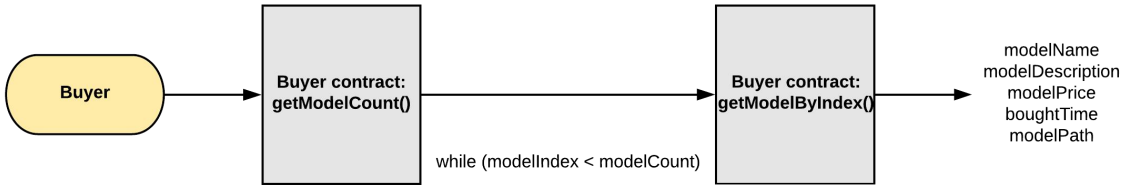

Figure 15 Buyer views bought models

4.4. Client Branch

Compared with seller and buyer, the interaction between client and freelancer is performed in long term that the transaction will not happen immediately when a freelancer accepts the request. When the user is acting as a client, he can seek help by posting requests. He can view posted requests and choose whether to close them to get results. Most of the functions used in this branch are from Client contract, which is shown in Table 6.

Table 6 Functions for client contract

Function	Input	Description
postRequest()	Request's name, description, test code path, reward, threshold, posted time	Post a new request
updateRequest()	Request's index for this contract (client), submitted model's path, test result and author's user address	Update a request, will be called when a freelancer submits a model for this request
getRequestCount()	-	Count the number of requests in this contract
getRequestByIndexVisible()	Request's index for this contract	Get a request's information (part 1)

getResquestByIndexInvisible()	Request's index for this contract	Get a request's information (part 2)
closeRequest()	Request's index for this contract	Close a request and perform transaction

The operation of posting a new request is similar to seller posting a new model, the difference is that the client should deposit the reward to contract during posting. In this process a request object will be generated with inputs and then saved in client contract, as shown in Figure 16. The structure of request object is shown in Table 7, where the result information is initialized.

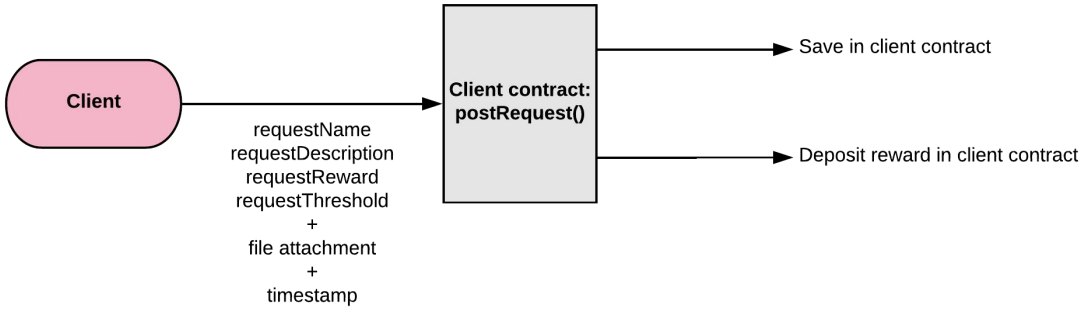

Figure 16 Client posts a new request

Table 7 The structure of a request object (for client)

Attribute	Type	Description
requestName	string	Request's name
resquestDescription	string	Request's description
testCodeAddr	string	The path of test code
reward	uint	Request's reward
threshold	uint	The threshold for test result
requestEnd	bool	If true, this request is closed. Initialized as false
modelAddr	string	The path of model with best test result (this result should also be better than threshold), initialized as null
testResult	uint	The best test result (this result should also be better than threshold), initialized as 0
freelancerAddr	address	The author's user address for model with best test result, initialized as this client's user address.

The client is able to view requests he has posted, as shown in Figure 17. As Solidity does not support too many output variables, the viewing function here is divided as two parts and the visibility of information is set based on the status of request. There are 3 status for posted requests: "closed with successful model", "closed with no suitable model", "available request". The first status means that there's already model that satisfies the threshold and the

client can view all information and download it, while the second status means there is no suitable model. The requests with Status 3 mean that they are still available and the result information (i.e. submitted model and test result) is not visible until this request is closed.

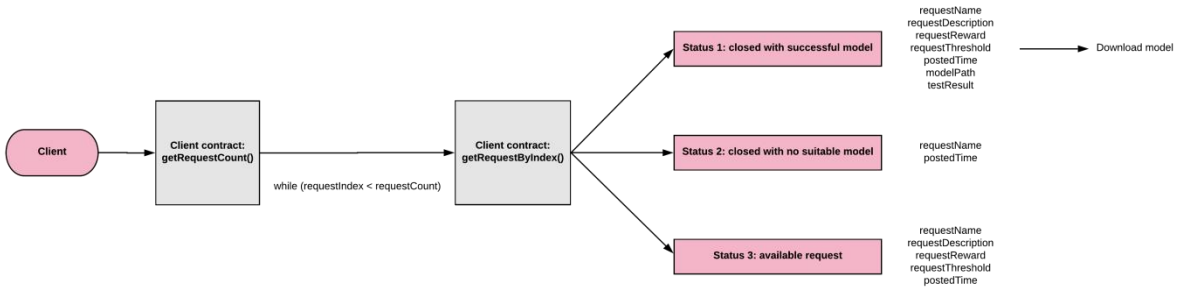

Figure 17 Client views posted requests

For a posted request, the client can decide whether or not to close it and view result. The transaction will happen in this process that if there's already suitable model, the money will be transferred to its author, otherwise as the freelancerAddr is still initialized as this client's user address, the money deposited in contract will be transferred back to the client. The whole process is shown in Figure 18. On the other hand, in order to protect freelancers in long term transaction, the client is not allowed to update description, reward or threshold, that the information of request will not change until it is closed.

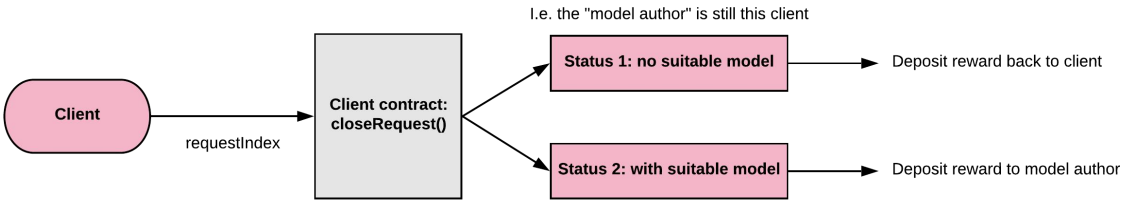

Figure 18 Client closes a request

4.5. Freelancer Branch

When a user is acting as a freelancer, he can be able to search requests by their names or view all requests posted on this platform. For search result he can decide whether or not to accept them. For those have been accepted, he can submit model for either of them if this request is not yet closed. Most functions used in this branch are from freelancer contract, which is shown in Table 8. Functions of client are also used here for interaction.

Table 8 Functions for freelancer contract

Function	Input	Description
acceptRequest()	Client's user address and the request index for this client	Accept a request
getRequestCount()	-	Count the number of requests in this contract
getModelByIndex()	Request's index for this contract	Get a request's information
checkRepeat()	Client's user address and the	Check whether this request has

	request index for this client	been accepted (already exist in this contract)

The searching operation for freelancer is similar to what in buyer branch. There are 4 status, as shown in Figure 19, that the information visible to this freelancer maybe different. For those posted by this user as a client, already accepted and closed, the freelancer can only see the request name and posted time, and can not accept them. Only those still available can be accepted, as shown in Figure 20. Compared with the request object for client, here the request object (Table 9) is much simpler that more details can be accessed by calling corresponding client's contract. In this way the freelancer can know the result of an accepted request in real time if the client close it. Different from interaction between buyer and seller, where purchasing is performed immediately when the model is bought, freelancer will not get or lose money by accepting a request, thus shopping cart is not needed here.

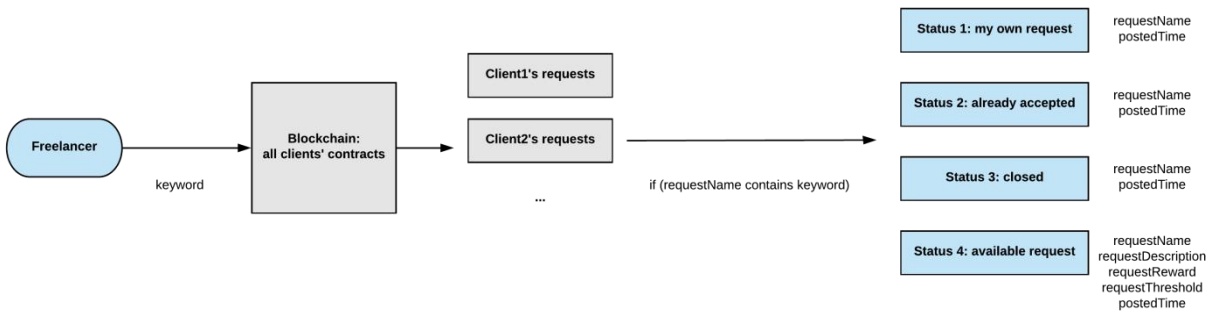

Figure 19 Freelancer searches requests

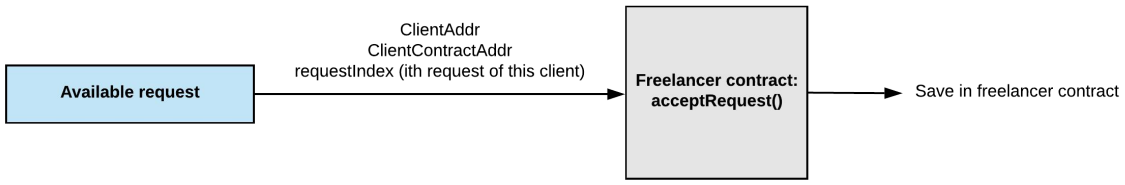

Figure 20 Freelancer accepts a request

Table 9 The structure of a request object (for freelancer)

Attribute	Type	Description
clientAddr	address	Client's user address (who posted this request)
resquestClientIndex	uint	The index of request for this client (this is the ith request posted by this client)

The freelancer can view requests he has been accepted, as shown in Figure 21. The simplified request object is used to get access to more thorough information in client's contract, that there are 3 status based on attribute "requestEnd" and "freelancerAddr". For those still available, the freelancer can submit a model as solution. This model, as well as the corresponding test code of this request, will be processed by online judge module first to get test result, then the client function "updateRequest()" will be called to update the model, test result, and author's (i.e. this freelancer) user address. There are 3 status for the updating

result, only when the test result surpasses both the threshold and current best result will these new values be assigned. For available requests the freelancer can only know the test result of his own submission. If a request is closed while this freelancer is the provider of best model, he can be reminded by the status of accepted requests. In order to make the transaction anonymously, the freelancer does not need to know who posted this request while the client does need to know who provided the solution. In this way they can focus on the requests and models themselves, regardless of the identification and reputation of authors.

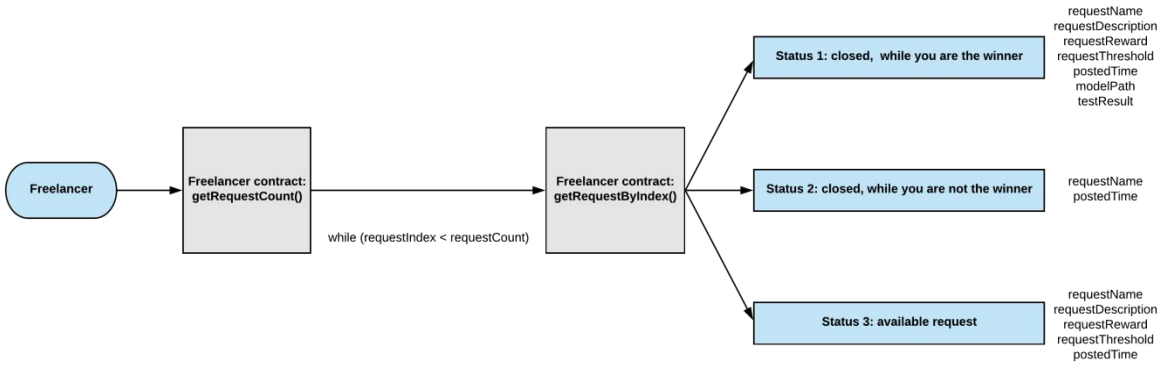

Figure 21 Freelancer views accepted requests

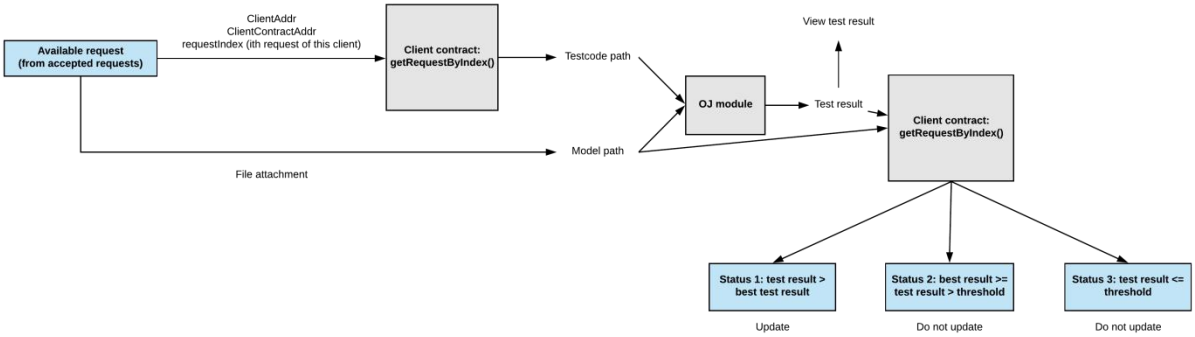

Figure 22 Freelancer submits the model

4.6. Comparison with the Proposal

Most of the deliverables in proposal have been achieved successfully. Table 10 shows differences between proposal and final implementation:

Table 10 Differences between proposal and final implementation

Proposal	Final implementation
User profile page	There's no need for specific user profile, the user can choose roles from mainpage and view different forms of transaction history (e.g. models I bought). Changing password will be implemented later as a non-functional operation.
View/edit the details of item	No specific page for item details, all the visible details is shown directly (e.g. search result, posted items, bought items and

	accepted items). The seller can edit the description and price of posted models.
View all items in the mainpage	Combined with searching operation. If there is no keyword, all items will be shown.
The due date of request.	The request will not be closed automatically on due date. The client can add due date as specification and then close the request manually.

5. Implementation

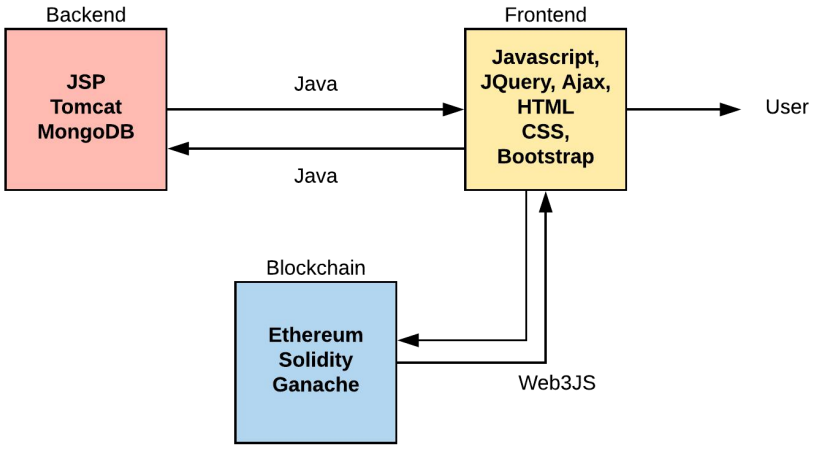

Figure 23 Technology stacks

The technology stacks used in this project are shown in Figure 23 and Table 11 lists the detailed version of techniques and tools. The server is built based on JSP and Apache Tomcat [14]. MongoDB [15] is used to store user's personal details and shopping cart information, thus key-value pairs are used instead of relational tables. Solidity [16] is used to build smart contracts and Web3JS [17] is used to interact contract functions with front end. Instead of MetaMask, Ganache [18] is used to provide simulated user accounts. In terms of front end, general techniques are used and the page architecture is shown in Figure 24.

Table 11 Techniques and tools

Techniques and tools	Version
Java & JDK	1.8
Apache Tomcat	7.0.56 / 7.0.82
MongoDB	3.6
Solidity	0.4.20

Web3JS	0.14.0
Ganache	1.0.0
NodeJS	General version
Python	General version
Operation system	Ubuntu 16.04LTS, Mac OS, Windows 7 & 10 (For Windows some paths need to be changed)
IntelliJ IDEA	Ultimate
Sublime Text	3

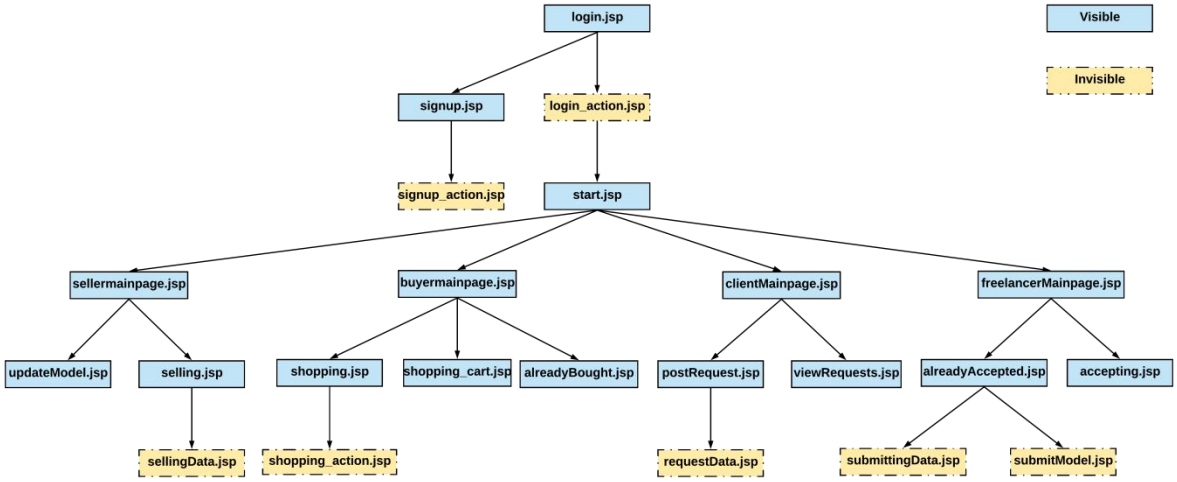

Figure 24 Page architecture

All the models as well as test codes are stored in local folder. When an item is uploaded, the relative path is like ***./upload/A/B_C.py***, where A is the contract address, B can be either "model" or "testcode" and C is the index of this submission. An example can be ***./upload/0x3AAA5AA04c6d6e81B04792448aff017490bdf664/model_0.py***, that the contract address can be either seller or freelancer, and the index means this is model for the first item. With this path the file can be downloaded or used in OJ module. Currently both the model and testcode are restricted to Python scripts, and the instructions about file name will be shown during submission.

Instead of using open-sourced online judge system, we implemented a simple OJ module to test the models. When the freelancer submits the model in alreadyAccepted.jsp, This module will take the path of model and corresponding test code as input, then run a python script on JSP to write these two files together as a new script, this new script will be ran to generate a text file which contains result. When the result is posted back to alreadyAccepted.jsp, the two generated files will be removed automatically. The whole process is shown in Figure 25. Currently this OJ module can only run simple python code and the format of test code and model should follow the user manual (read Section 6 for details).

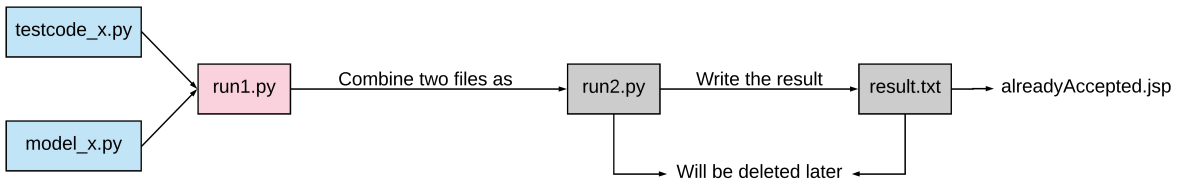

Figure 25 OJ module

6. User Manual

This section contains tutorials about how to setup, use and modify this platform. We also recorded some demos at our project web page, which may be beneficial for the user to get familiar with our platform https://sites.google.com/view/playingwithmud

6.1. Setup and Configuration

The project consists of 4 folders: ***tomcat_test, model_and_testcode***, ***original_contracts*** and ***project_final***, where the last one is the major part. Before running, all dependencies should be satisfied, as shown in Table 11. It's recommended to run the project on Ubuntu or Mac OS for that some paths need to be changed when running on Windows. There's something wrong when we were using Tomcat 8.x, so it's recommended to use v7.x, such as v7.0.56 and v7.0.82. Before doing following configurations, you should make sure that you can run JSP pages on server. We provided a simple project, ***tomcat_test***, to test the basic configuration.

The configuration of this project should follow following steps, and the tutorial video can be found on project web page:

1) Copy all .jar files in ***project_final/lib*** to Tomcat's library, for me the path is ***/opt/apache-tomcat-7.0.56/lib***. After this step you should test whether the simple project can still be run correctly. If not, do not copy ***jsp.jar***. This error is found on Ubuntu only while Windows and Mac OS still work after copying all files.

2) Open the project with IntelliJ IDEA and check project structure, as shown in Figure 26-29

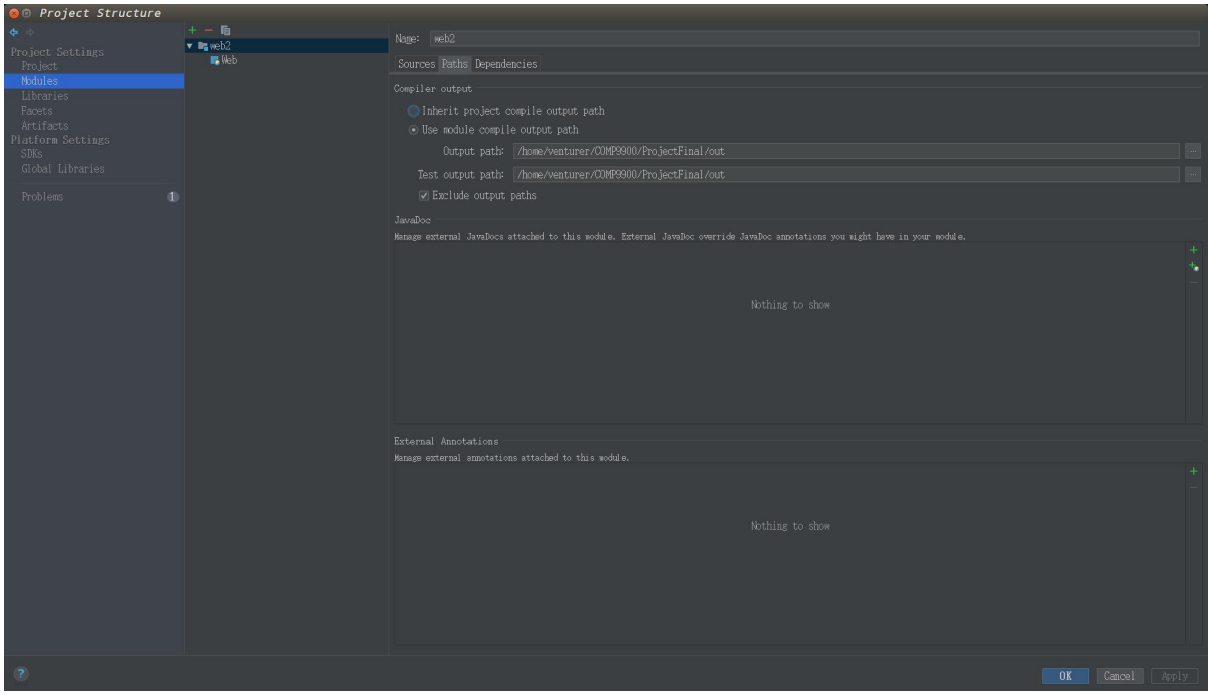

Figure 26 Tutorial : Path in Modules

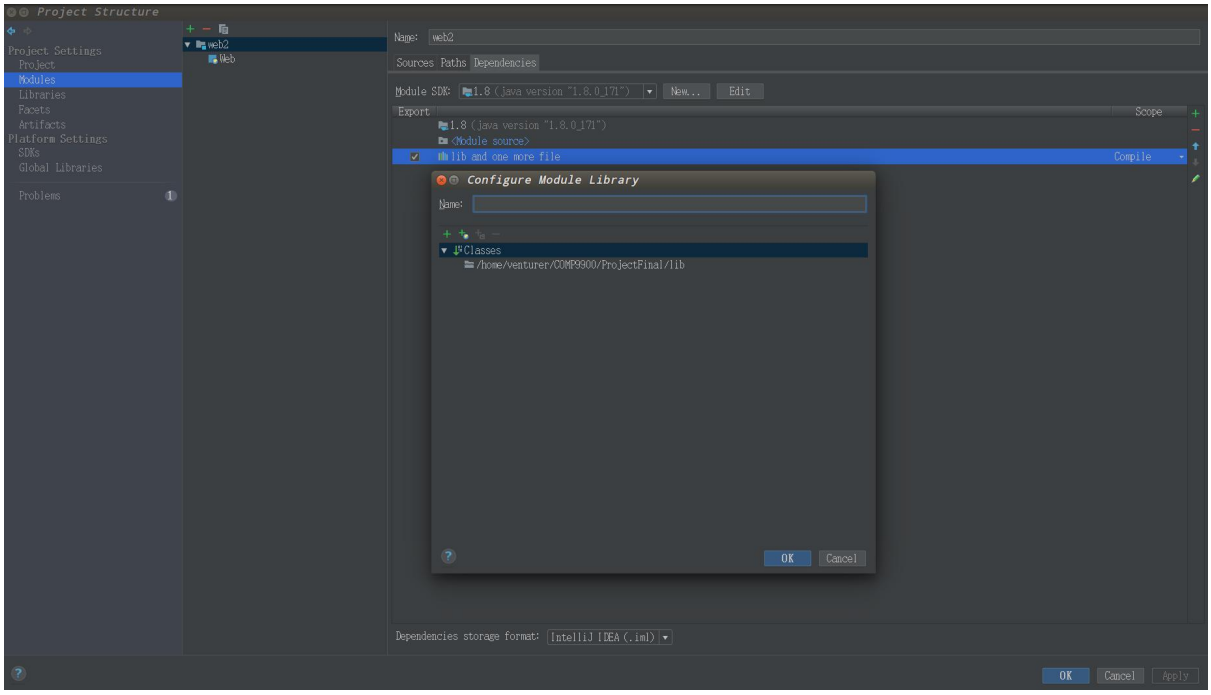

Figure 27 Tutorial : Dependencies in Modules

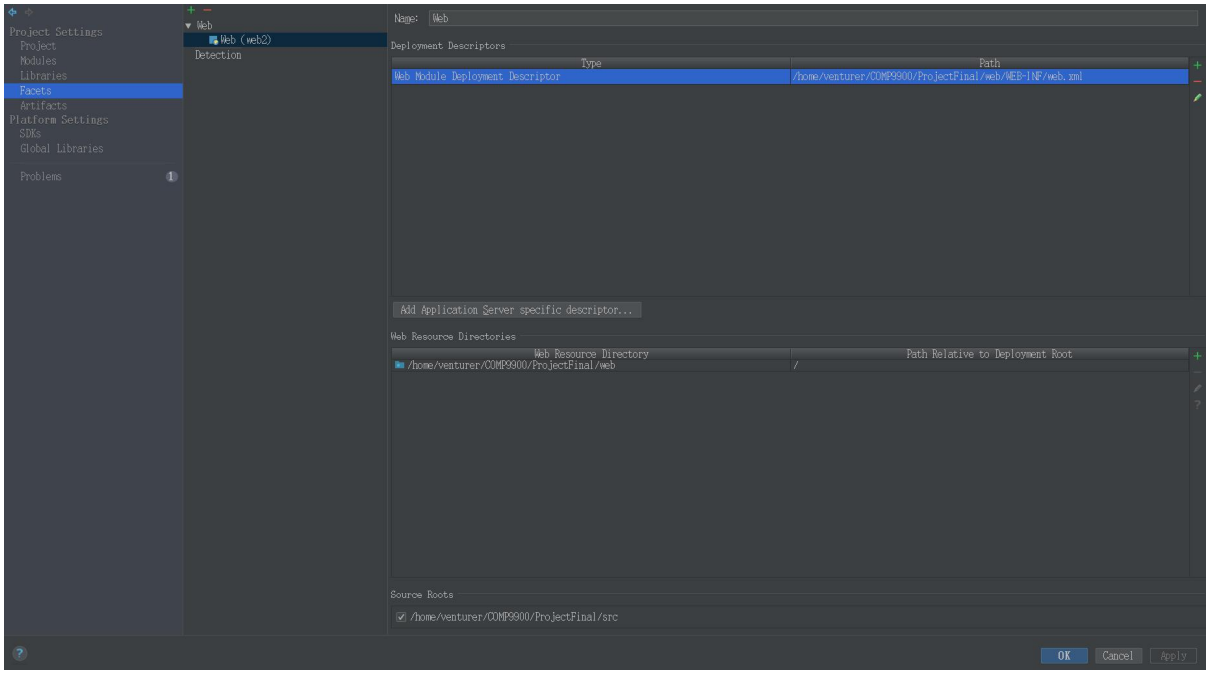

Figure 28 Tutorial : Facets

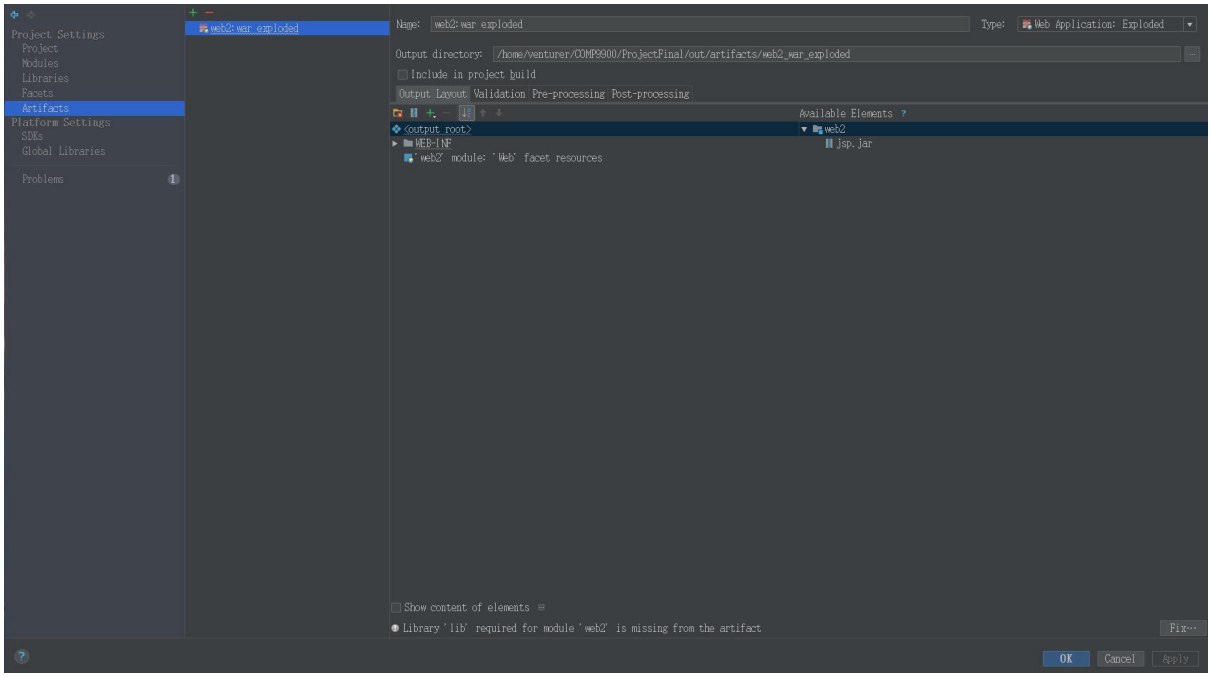

Figure 29 Tutorial : Output directory in Artifacts

3) Choose a server and edit the configuration, as shown in Figure 28 and Figure 29:

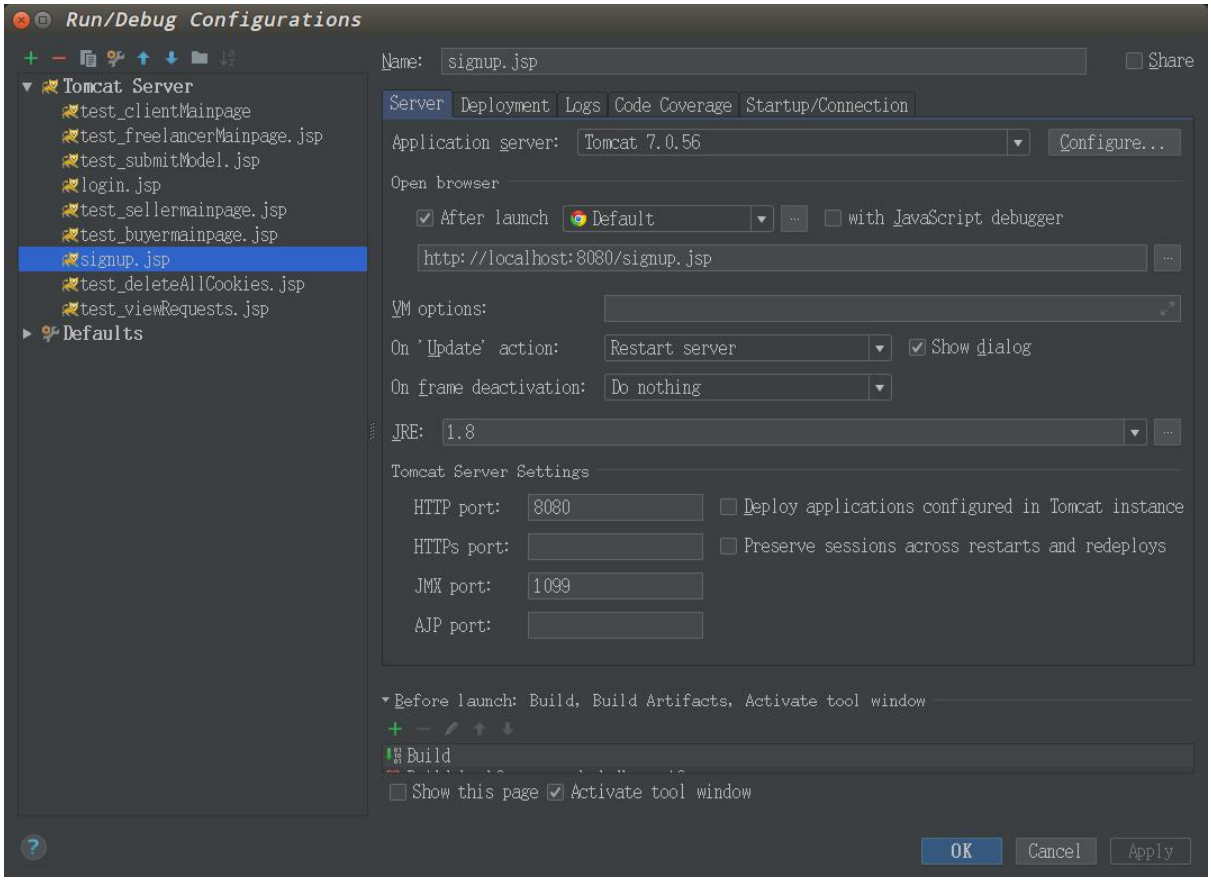

Figure 30 Tutorial : Check server

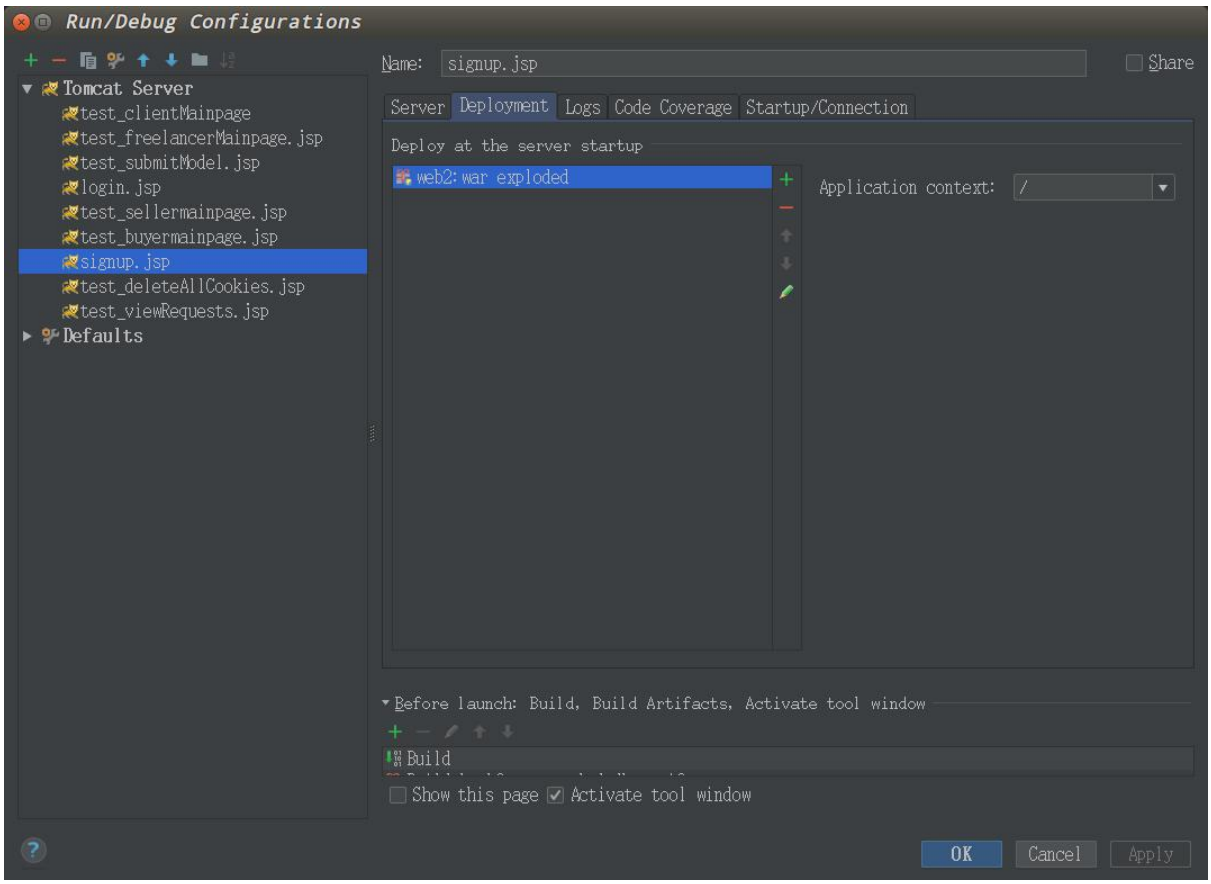

Figure 31 Tutorial : Check deployment

4) Run this server, this will take several seconds to deploy.

5) Open Ganache to simulate user address

6.2. Use the Platform

The new user should create account first, then he can log in and choose either of the roles. As illustrated in Section 4 and Figure 4, for each role there can be different operations. Some sample models and test codes are provided in To help the users to get familiar with this platform, some demos with detailed operations have been released on the project web page, as shown in Table 12.

Table 12 Demos on project webpage

Demo Index	Description
1	User creates new account and logs in
2	Seller posts new model, view posted models, and updates model
3	Buyer searches model, adds models to shopping cart, then buys model
4	Client posts new request, views and closes posted requests
5	Freelancer searches and accepts models
6	Freelancer submits model and solution to accepted request

6.3. Restore the Platform

If the user wants to restart the project, following steps should be done to restore it to original state:

1) Restart Ganache

2) Delete all submissions in out/artifacts/web3_war_exploded/upload

3) Run MongoDB on command line and delete the database ***test***

```
mongo
use test
db.dropDatabase()
exit
```

4) Restart server

Then the project can be restored to original state. Note that this is different from “restart server”, for this case, just restart server directly without doing these.

6.4. Extension

During running only the compiled contracts are used. E.g. for a seller contract, there are 3 related files: ***SellerABI.js,*** ***Seller_submit_sol_Seller_abi.js*** and ***Seller_submit_sol_Seller_abi.js***. The original contracts in Solidity are in folder ***original contracts***, if the user wants to modify them to go beyond our project, the modified contract should be compiled again and the old .js files should be replaced. Take seller's contract for example:

1) Run solc to get ABIs and BINs, this can also be done from Remix [19]

```
    solcjs --abi Seller_submit.sol
    solcjs --bin Seller_submit.sol
```

2) Modify their format to the 3 related .js files, where ABI file can be used for 2 files and BIN can be used for another one.

3) Replace the old .js files.

7. Summary

In this project we built a trading platform specific for machine learning models. It combines online shop and outsourcing platform that the user can not only exchange models directly, but also perform transaction via posting requests and providing solutions. This platform is based on blockchain that it is decentralized without intervene of third party, while all the operations are performed anonymously. We hope this platform can fulfill the need for those want to exchange models, thus boost the research of machine learning, data mining and related areas.

8. Reference

[1] Sun Y, Chen Y, Wang X, et al. Deep learning face representation by joint identification-verification[C]//Advances in neural information processing systems. 2014: 1988-1996.

[2] Amodei D, Ananthanarayanan S, Anubhai R, et al. Deep speech 2: End-to-end speech recognition in english and mandarin[C]//International Conference on Machine Learning. 2016: 173-182.

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.

[4] Raval S. Decentralized Applications: Harnessing Bitcoin's Blockchain Technology[M]. " O'Reilly Media, Inc.", 2016.

[5] Wright A, De Filippi P. Decentralized blockchain technology and the rise of lex cryptographia[J]. 2015.

[6] Buterin V. A next-generation smart contract and decentralized application platform[J]. white paper, 2014.

[7] http://rebelgrowth.com/best-outsourcing-websites/

[8] https://www.kaggle.com/

[9] truffleframework.com/tutorials/pet-shop

[10]https://www2.deloitte.com/nl/nl/pages/financial-services/articles/5-blockchain-use-cases-in-financial-services.html

[11] www.tauchain.org/

[12] https://spheris.io/

[13] Kurtulmus A B, Daniel K. Trustless Machine Learning Contracts; Evaluating and Exchanging Machine Learning Models on the Ethereum Blockchain[J]. arXiv preprint arXiv:1802.10185, 2018.

[14] tomcat.apache.org/

[15] https://www.mongodb.com/

[16] https://solidity.readthedocs.io/en/v0.4.20/

[17] https://github.com/ethereum/web3.js/

[18] https://github.com/trufflesuite/ganache

[19] http://remix.ethereum.org