Distributed ledgers application in Science: Smart Papers on the Ethereum

YixuanXu (yx5u17@ecs.soton.ac.uk)
University of Southampton

September 3, 2018

Contents

1 Introduction **1**
- 1.1 Project aim 1
- 1.2 Outline 2

2 Background Research **4**
- 2.1 Modern publishing models example 4
- 2.2 Blockchain 5

3 Design **7**
- 3.1 Requirements 8
- 3.2 Data structure 9
- 3.3 Smart Contract 10
- 3.4 The Smart Paper workflow in Ethereum 11

4 Implementation **13**
- 4.1 New web service 13
- 4.2 Development workflow 15
- 4.3 Workspace setup 16
- 4.4 The Paper contract 16
  - 4.4.1 Source code 16
- 4.5 Compile, Test, Deploy 19
  - 4.5.1 Compile 19
  - 4.5.2 Test 20
- 4.6 Cost analysis design 22
- 4.7 Visualization 22

5 Discussion **23**
- 5.1 Good 23
- 5.2 Bad 23

6 Conclusion **24**

List of Figures

3.1 Traditional publish model 7
3.2 Smart paper publish model 9
3.3 Smart Paper Design 11

4.1 Traditional Web Service 13
4.2 Decentralized web service 14
4.3 Development workflow 15
4.4 Compile Result 20
4.5 Test Result 21

List of Tables

Abstract

This work is all about ...

Chapter 1

Introduction

1.1 Project aim

Digitization and Web technologies are now changing the way of publishing and disseminating the knowledge. It becomes more convenient and less expensive for people to access the knowledge. The knowledge creation process is more dynamic right now. Text/graphics/rich media can be changed quickly and easily while at the same time being available to all the audiences. However, most of current methods of academic publication are static, that means, they cannot be revised over time[1]. Web technologies actually have the power to make it more dynamic but is currently underused. On the other hands, journals, publishers and funders fully control the entire process of academic publishing. The view of authors who should also participate in the publishing process tends to be underrepresented. Despite that fact that the current academic publishing system is advance and productive, authors still want a more open and decentralize publishing process [2].

In the past, the scholarly books were keeping improving and updating for the centuries by releasing the new editions. Mistakes would be corrected, new result would be added and feedbacks would be used for improvement. Revising books allowed author to keep track with novel development [1]. Many handbooks and schoolbooks have been revised over and over again, resulting massive amount of quality publications. In the contrast to books, academic paper were a kind of snapshot of certain scientific knowledge. Most of them were just published once. If there is a new finding, usually a new articles need to be published. But this kind of process are currently under debate and development [1]. The number of authors who want a more open process in scientific publishing is increasing rapidly. When it comes to the traditional of academic publishing, publishers play an important role of filtering good research, rejecting papers without sufficient conclusions. They make their decisions based on the peer-review process which is fully controlled by themselves. Since this kind of peer-review process usually will take a significant amount of time and is one of the main reason of delaying publications, researcher came up with a new idea of doing publishing. A initial version will be firstly released, then it could be updated after receiving the feedbacks from pre-peer-review. All the version will be always available and the changes made in the pre-peer-review process will also be stored after the publishing of final version. The model allows the tracking of the development of academic papers. This improvements make the process of publishing process more dynamic and flexible. But, there are some vital problems that have been discussed under such models which is the mechanism to manage the interactions between authors and contributor in a trust way [3]. How can authors make agreements with each other about which version should be available ? How can authors determine their contributions to the papers in a unprejudiced way ? On the other hand, the contributions of reviewers is ignored in this model.

In last few years, Distributed Ledger Technology have attracted public attentions as the most advanced tool that can provide a decentralized solutions to manage the interaction between people that may not trust each other. It also could guarantee the security and consistence without the need for admin. The special tools which could achieve such functionality is called Smart Contract. For the questions about the publishing model, Smart contract could be programed to help authors to making decision in a decentralized way. The aim of this project is trying to provide a prototype of decentralize application to help authors to manage their publish and their attribution agreements in a dynamic and trusted way. The application itself will use the blockchain technology [4], so nobody can fully control the whole process. It could be more reliable than the current publishing system. The implementation will be evaluated by the cost of using such system. A detailed cost analysis and data visualization will also be presented.

1.2 Outline

In Section 2, firstly it will present some modern publishing systems based on the blockchain. Then a general background of blockchain will be presented. After that, a more specific description about the smart contract on Ethereum would be shown for the following part

In Section 3, it would give detailed design of the application. By comparing the blockchain technology with the current Web technology, It should gives more clear explanation why blockchain is preferred for this project

In Section 4, it will focus on the Implementation. All the important implementation details will be described here

Section 5 demonstrate the cost of using the decentralize application. The analysis consists of the visualization and specific code review.

Section 6 will discuss the advantage and disadvantage of the current implementation. Potential improvement would also be covered in this section.

Section 7 will give a summary about the project and conclusion.

Chapter 2

Background Research

2.1 Modern publishing models example

Because of the development of web technologies, several models have been provided based on the modern web technologies to improve the process of the academic publish, making authors manage and produce their works more easier. DEIP [5] is a platform that aimed at effective and fair distribution of resource allocated to scientific and research activities. It proposed a community-driven models that encourage the open knowledge without restriction. The users of this platform could be both author and reviewers and have free access to all the publications. All the records of reviews will be recorded and reviewers will be awarded by the decentralized protocol. Scienceroot [6] is another blockchain-based publishing platform, it tries to creating a scientific publishing model which will reward and sustain researchers instead of maximizing the publishers' profits by using a fully decentralized storage platform called IPFS [7]. The Pluto [8] is also a platform that is trying to help researchers to get funding in a decentralized ways. It also wants to establish a more proper evaluation index for the academic publications. The ideas of these kinds of platforms are really aggressive but inspiring. They are trying to give the power back to authors and reviewers and to make the knowledge more open and accessible to everyone. It could be found that modern publishing system requires more transparent review process and more trustable collaborations between authors. The current Blockchain technologis have huge potential in aspect of creating decentralized application. So it could be found that all of above platform are using the Ethereum [4] as their core framework.

2.2 Blockchain

Technology On its most basic level, the blockchain is a new kind of information technology which is the combination of encryption methods with distributed computing. Satoshi Nakamoto [9] combine them to make new ways to create a model where a network of computers collaborate to maintain a shared and secure database. This database consists of a string of blocks each one a record of data that has been encrypted and given a unique identifier called a hash. Mining computers on the network validate transactions, add them to the block they are trying to build, and then broadcast the completed block to other nodes so that all have a copy of the database. Because there is no centralized server to validate the transaction to the database, the blockchain depends upon a distributed consensus algorithm. In order to make an entry on to the blockchain database, all the computers have to agree about its state so that no one computer can make an alteration without the consensus of all others. Once completed, a block goes into the blockchain as a permanent record. Each time a block get completed, a new one will be created. Countless number of block connect to each other, like links in a chain. The kind of data structure makes the transactions immutable. Every block contains hash value that is dependent on the hash of the previous block, if one is changed than all the other blocks linked to it going forward will be altered. This works to make the data entered tamper-proof. This model is the working mechanism of the first generation blockchain which is known as Bitcoin. Its functionalities are mainly focused on the small data storage, which is the history of transactions. Blockchain are trying to create a secured, trusted, shared database and they do this through encryption and hashing, proof of work and network consensus. The hashing and linking of blocks makes it difficult to go back and change a previous block once entered. And the proof of work system intentionally makes it computationally more difficult to alter the database, thus making it extremely difficult to change all the blocks. It leads to a distributed consensus mechanism so that even if someone did it, it is almost impossible to convince others to accept as the valid record. The bitcoin blockchain is a very good proof of the stability of such system. It now secure hundreds of billions of dollars using this method without the network having been hacked. What this technology enables is a database that is secure with automatic trust which is powered by open source community and encryption. It is tamper proof, once information is put into the database it can not be altered. It is a shared database as many people would have a up-to-date copy so that all have a single source of truth. Likewise, it is transparent, all the transactions and alteration made to the blockchain is visible to everyone.

Ethereum With in a few years, the second generation of blockchain emerged design as a network on which developers could build applications. It is going to be more like a distributed virtual computer than just database. This was made technically possible by the development of the Ethereum platform. Ethereum is an open-source, public, blockchain-based distributed computing platform featuring smart contract functionality. It provides a decentralize Turing-complete virtual machine, which can execute computer programs using a global network of nodes. Ethereum was initial described in a white paper by Vitalik Buterin [4]. The system has been very successful attracting a large and dedicated community of developers, supports and enterprises. The important contribution of Ethereum as the second generation of blockchain is that it worked to extend the capacity of the technology from primary being a database to becoming more a general platform for running decentralized application and smart contracts. It provides developers with a powerful but simple turing-complete language called Solidity that used for build smart contracts. As of 2018 Ethereum is largest and most popular platform for building distributed application. It has been a major step forward to become a global distributed computer, a massive globally distributed cloud computing platform upon which we can run any application at scale and speed, with the assurance that it has the security, resilience and trustworthiness of today's blockchain.

By reviewing the blockchain technology, it is easy to find out that Ethereum is really powerful to help authors to make agreements and set their outcomes in stone so that they cannot be later repudiated. In [3], the Smart Paper model is proposed to provide a collaborative platform that preserves a single version of the truth through the collaborative process. Authors will use the smart contract to publish their works and collaborate with others. Once decision is made, smart contract will guarantee the result of decision won't be change by anyone. All the versions and meta data about contributions will be safely stored on the Ethereum.

Chapter 3

Design

For a information system, the state flow and data structure are the two key point. The state flow is related to the service logic. It concerns how to operator data, how data changes. The application interface and the implementation of interact should be built based on those information For a traditional publish model, it could be described as 3.1.

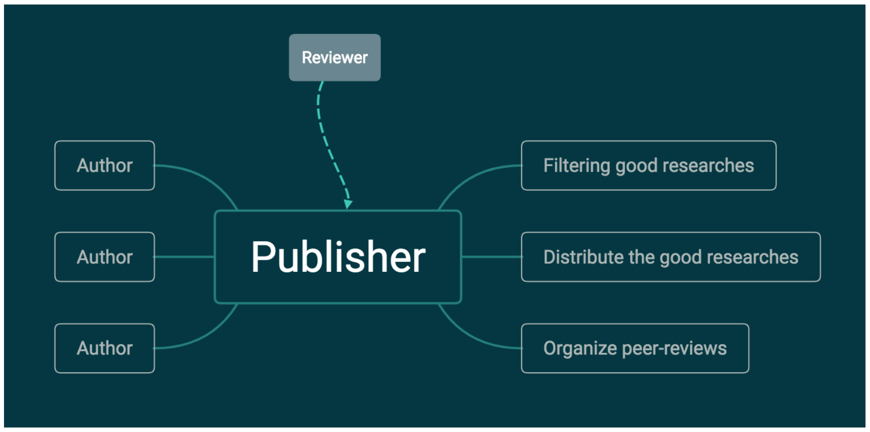


Figure 3.1: Traditional publish model

In this model, publishes takes all the responsibilities of filtering the good researches, rejecting papers with methods that are insufficient to draw the stated conclusion and the distributing widely[1]. The whole system is too centralized. On the other hand, the process of producing paper is not transparent to the public. It is hard for authors to determine their contributions for the paper. The peer-review process is also not transparent enough.

There should be a channel for authors and publishers to exchange those vital information.

3.1 Requirements

The first step of software developing is to identify the requirements from the specification, then finding the correspond user cases. The aim of this project is to provide a prototype of decentralize application to help authors to manage their publish and their attribution agreements. Based on this, we could have these simple user cases:

- Researchers should have their identities based on addresses of their Ethereum accounts
- Researchers should be able to use smart contract to publish a research artifacts, which will be represented by the hash of files
- Researchers should be able approve the contracts that belongs to themselves after verifying their local files with the online files with respect to a specific version
- The paper contracts should contains the description, metadata and version list of the research artifacts.
- The contents of papers could be update with the agreements of all authors.
- New researchers could be added to an ongoing paper with the agreements of all the previous researchers.
- The application should be open sourced and decentralized. Anyone could deploy their own instances.
- The review process could be community-driven and powered by publisher.

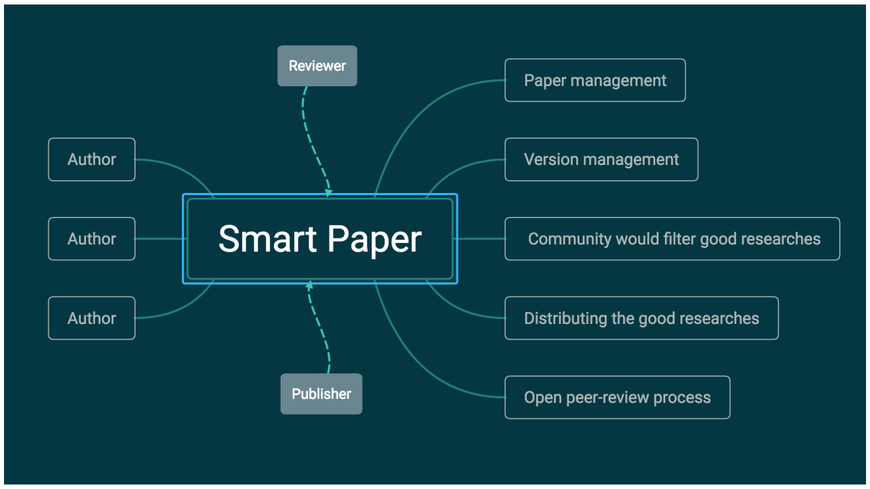


Figure 3.2: Smart paper publish model

The smart paper be design based these requirements. Based on the power of smart paper, we could have a new mechanism described as figure 3.2. In this model, the whole process of publishing would not be controlled by any individuals or organizations. Authors could use the smart paper to do their publishing which will keep the full records of producing process. publisher could use smart paper to organize open peer-review. The smart paper will be a powerful but user-friendly tool for both authors and publishers. Their contribution will be recorded permanently on the Ethereum. The smart paper consists of smart contracts which is used to implement those functionalities.

3.2 Data structure

The fundamental data unit for the smart contract used in these project is paper, which could be created and managed by authors and be reviewed by reviewers. the paper has it own basic properties such as description, metadata, hash of files and the list of authors. Here are the properties that should be included in the paper contract:

- **Description** the description of these paper, which help reviewer to understand the area of the paper.
- **Metadata** the metadata of these paper, which contains the contributions, comments and all the other important metadata of the paper.

- **latestMd5** the hash of files, which is the latest version.
- **versionNumber** the number of latest version, which will shows the latest published version to the public.
- **authorList** the list of Authors, which is the users who could control this contract
- **versionList** the list of versions, which contains of all the version in these contract.
- **isAuthor** a mapping of address to boolean value, which indicate that if the user is the author of paper.

The version of paper has complicated structure so we have to use a special keywords called 'struct' to construct it. This keyword works like 'struct' in C programing language. For version, there are some properties:

- **versionNumber** the unique Id for the current version.
- **md5** the hash of current version.
- **isPublished** the status of current version.
- **isSigned** the mapping of address to boolean, which indicate if the author is agreed this version
- **voterCount** the number of how many authors is agreed with this version, if the number is equal to number of authors, the state of version will be changed

These is the key design of the data structure of the project. The rest of design of the other contracts could be found at the source code.

3.3 Smart Contract

Three kinds of contracts, which are RBAC(Role based access control), smartPaper and smartPaperList, will be used in this project.

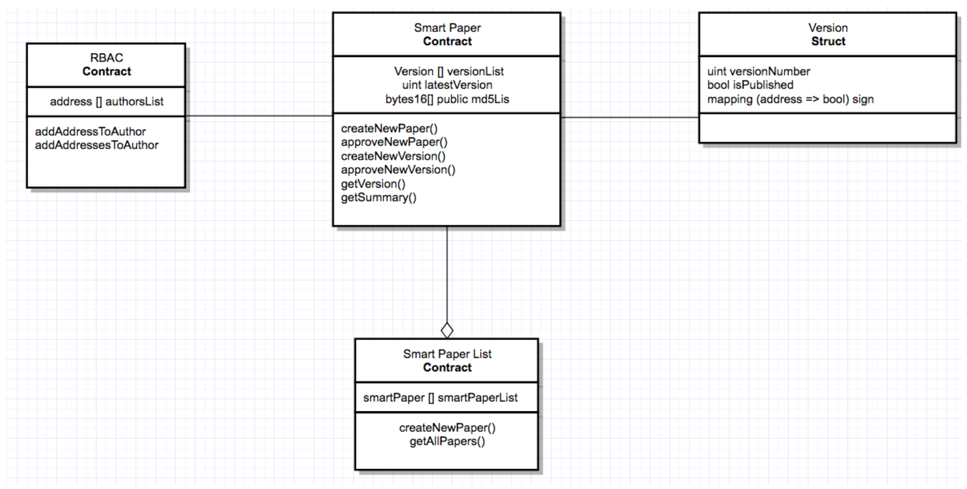


Figure 3.3: Smart Paper Design

The initial design in design in [3] consider the single version of paper as a standalone contract. However, it will increase the cost of smart paper. The current design is more cheaper by making the single version just a special data structure. RBAC is a special contracts used for access managements. It maintain a hash table of the address of authors. Every time authors want to publish or approve a new version, RBAC will check the if authors belongs to this paper. Smart paper contains all the core functionalities, in the prototype application the hash of file will be stored instead of real file. The description and metadata will also stored in the contract. Author can use the smart paper contract to publish new versions, then others authors need to approve the new version. Once the new version is confirmed, it would be public available to the reviewers. The smart paper list contract is used to control all the smart paper and create new smart papers. It is the entry point of application.

3.4 The Smart Paper workflow in Ethereum

To begin with, a description of paper, metadata of paper, an array of addresses of authors and the file is submitted by a writer. The smart paper list will take those information and create a new smart paper contract which contains all the information. Then all the authors need to use the new smart paper contract to approve the initial version. Once a new version is finished, follow the same step, writer need to use smart paper contract to release a new version by providing the new description, metadata and file. Then all the authors also need to approve the new version to make it public. The smart paper workflow involves multiple working versions with dynamic collaborators. Version can become published and made available for annotating. The smart paper list contract will keep the records of all the smart paper create by itself. User can access addresses of all the smart papers through this contract and check specific information of a paper by a specific address.

Chapter 4

Implementation

4.1 New web service

What the Ethereum platform really provides with the developers is a new kind of fundamental infrastructure for the current web service.

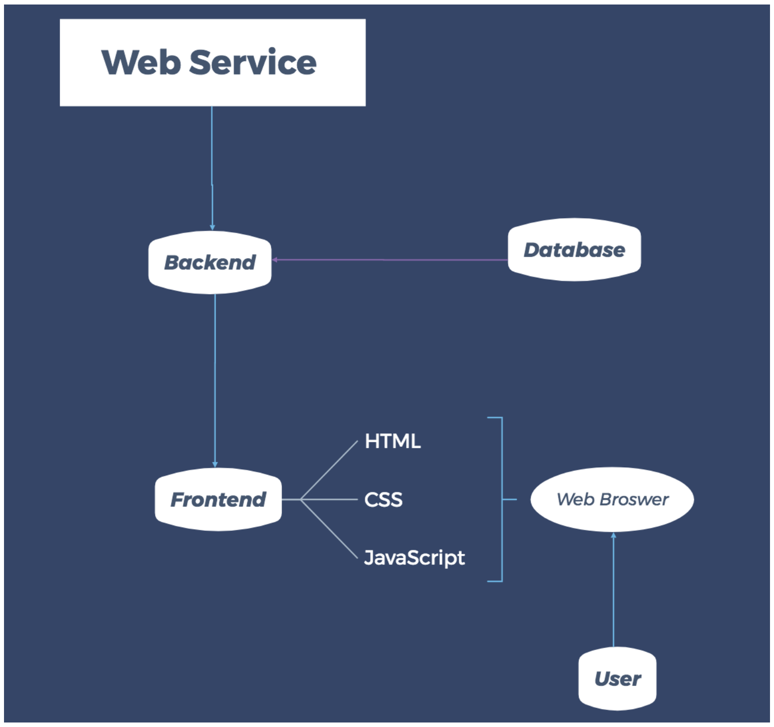


Figure 4.1: Traditional Web Service

The traditional web service, which could be described as 4.1, is a too centralized system. The backend and database are fully controlled by cloud service providers. If developers want to store their data safely, they have to chose those big companies such as Google and Amazon, which is expensive. Otherwise, they may suffer from the data loss. Cloud service providers have the admin accounts that could access all of the data even modify them without any permission. The blockchain technologies could change this situation.

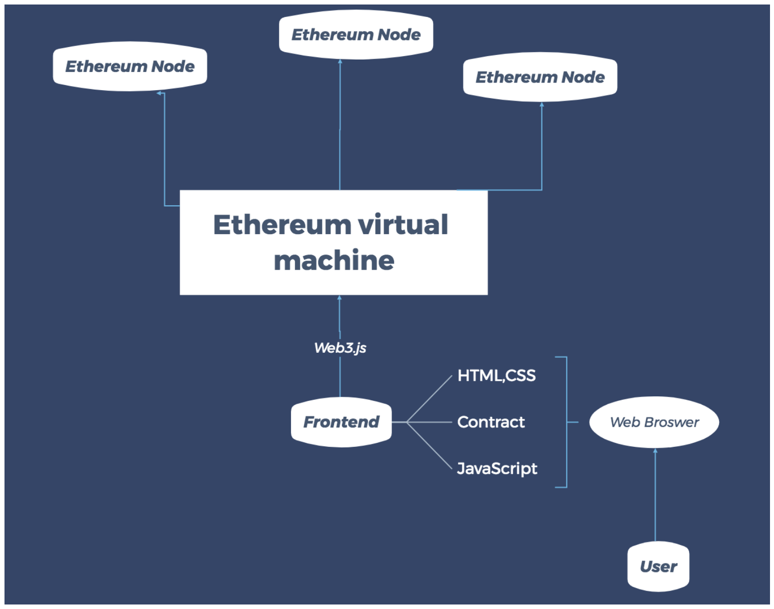


Figure 4.2: Decentralized web service

4.2 illustrates the new design of decentralized application. Developers will program the service into the contract. For the project, all the functionalities of requirements will be coded into three contracts. The the contracts will be compiled into bytes code. Those files will be hardcoded into the frontend source codes so that JavaScript will be able to use those contracts to interact with the Ethereum network. After receiving the information sent by the frontend, the Ethereum network will execute the code passed by the frontend, changing the state of network and keep and record, giving the response. The entire process does not need any cloud service provider. All the data will store safely in a fully decentralized system. Although the inner mechanism will change but user will have almost the same experience as the current web service since the frontend technology stack is the same.

4.2 Development workflow

The Ethereum network provide the developers with a tool called Remix [10]. However, it has limited functionalities. It could help the developers to deploy the contract and only allow the developers to test the contract by hands. It does provide the source control for Solidity. There is another open source called truffle [11], which is the most popular development framework for Ethereum. But truffle itself is too complicated. For these project, it is really important that we should a deep understanding about the new technology. The best choice is to create our own developing workflow.

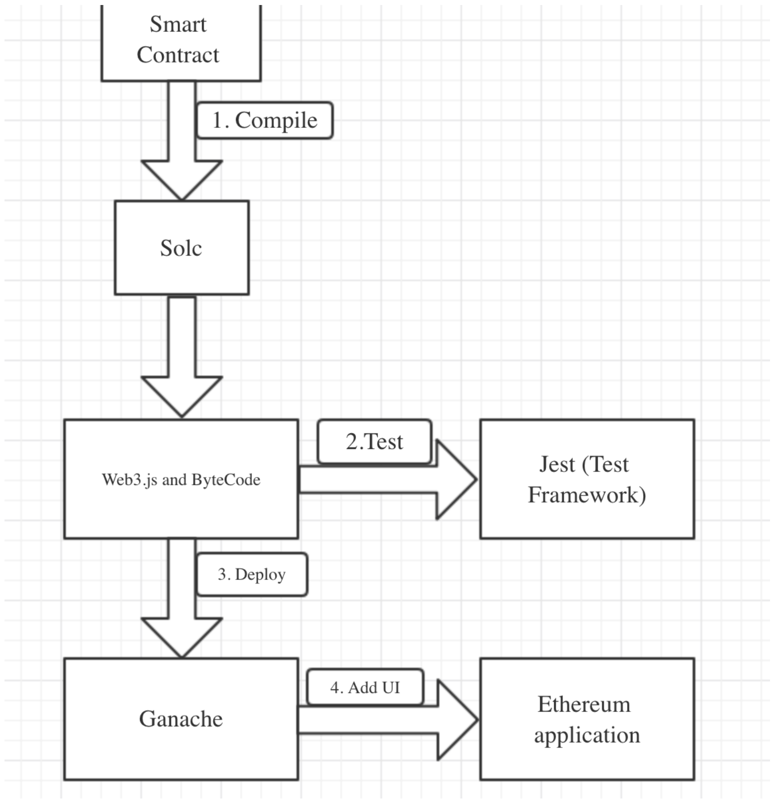


Figure 4.3: Development workflow

Inspiring by the truffle and remix, we could establish a basic workflow based on JavaScript ecosystem, which could be described as 4.3. After finishing the smart contract, Solc [12] which is Solidity compiler written in JavaScript will be used to compiled the source code. Then web3.js and the compiled smart contracts will be used for tested by jest which is a JavaScript test framework. After passing all the test, the contracts will be deployed on the Ethereum test network by Ganache. Then with the UI code, user will be able to use the application to do the academic publishing.

4.3 Workspace setup

Before starting write code, some prerequisites should be satisfied

- **Git** Version control
- **Node.js** The major technology stack of the project is the JavaScript.
- **Npm** The nodejs package manager, which will help us to manage tools like Solc and Jest.
- **Text editor** It is suggested to use Vscode.

After setting all the environment, a workspace should be initialed for the future development, which contains these directory:

- **contracts** store the contracts source code
- **compiled** store the compiled contract
- **scripts** scripts used to start compile, test and deploy.
- **test** define how test will conducted.
- **package.json** used for npm to manager the project.

4.4 The Paper contract

4.4.1 Source code

```
pragma solidity ^0.4.24;
contract SmartPaper is AuthorList{
  struct Version{
      uint versionNumber;
      bytes32 versionDescription;
      bytes32 metaData;
      bool isPublished;
```

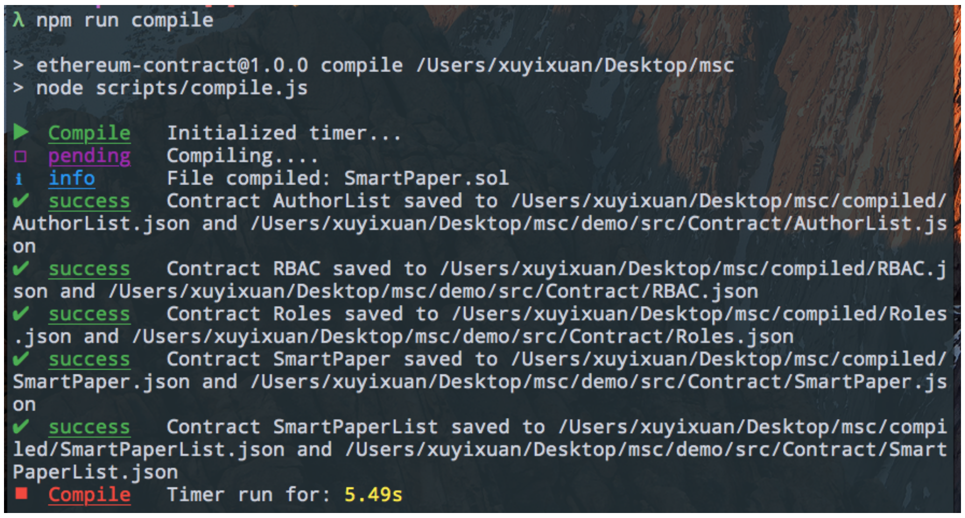
```
        mapping (address => bool) signs;
        uint voterCount;
    }
    bytes32 private latestDescription;
    bytes32 private latestMetaData;
    bytes16 private latestPaper;
    uint public latestVersion;
    address private newAuthor;
    uint private agreeCount;
    address[] public authors;
    mapping(address => bool) isAuthor;
    mapping(address => bool) isAgree;
    bytes16[] public md5List;
    Version[] public versions;
    mapping (bytes16 => Version) public versionMap;
    constructor (bytes32 _description, bytes32 _metaData,
        bytes16 _paperMD5, address[] _authors) public{
        require(_authors.length > 0, "Invalid authors list");
        authors = _authors;
        latestPaper = _paperMD5;
        md5List.push(latestPaper);
        latestMetaData = _metaData;
        latestDescription = _description;
        uint versionNumber = uint(1);
        addAddressesToAuthorList(_authors);
        Version memory newVersion = Version({
            versionNumber: versionNumber,
            versionDescription:latestDescription,
            metaData:latestMetaData,
            isPublished:false,
            voterCount:1
        });
        versions.push(newVersion);
        versions[0].signs[msg.sender] = true;
        versionMap[latestPaper] = newVersion;
        for(uint256 i = 0; i < _authors.length; i++){
            isAuthor[_authors[i]] = true;
            isAgree[_authors[i]] = false;
        }
    }
    function checkIn() public onlyIfAuthor(msg.sender) payable{
        require(versions[0].signs[msg.sender] == false, "Ban");
        versions[0].signs[msg.sender] = true;
        versions[0].voterCount++;
        if(versions[0].voterCount == authors.length + 1){
            versions[0].isPublished = true;
            latestVersion = versions[0].versionNumber;
        }
        versionMap[md5List[0]] = versions[0];
    }
    function createNewVersion(bytes32 versionDescription,
        bytes32 metaData, bytes16 md5) public
    onlyIfAuthor(msg.sender) payable {
        uint versionNumber = latestVersion + 1;
        Version memory newVersion = Version({
            versionNumber: versionNumber,
            versionDescription:versionDescription,
            metaData:metaData,
            isPublished:false,
            voterCount:1
        });
        md5List.push(md5);
        versions.push(newVersion);
        versionMap[md5] = newVersion;
    }
    function addNewAuthor(address _newAuthor) public
        onlyIfAuthor(msg.sender) payable{
        require(newAuthor==address(0), "Ban");
        newAuthor = _newAuthor;
        agreeCount = 0;
    }
    function approveNew(address _newAuthor) public onlyIfAuthor
        (msg.sender) payable{
        require(newAuthor == _newAuthor,"BAN");
        agreeCount++;
        isAgree[msg.sender] = true;
        if(agreeCount == authors.length){
            addAddressToAuthor(newAuthor);
            authors.push(newAuthor);
            isAuthor[newAuthor] = true;
            newAuthor = address(0);
            agreeCount = 0;
            for(uint i = 0; i<authors.length; i++){
                isAgree[authors[i]] = false;
            }
        }
    }
    function approveVersion(uint _versionNumber, bytes16 md5)
        public onlyIfAuthor(msg.sender) payable{
        Version storage version = versions[_versionNumber-1];
        require(!version.signs[msg.sender], "BAN");
        version.signs[msg.sender] = true;
        version.voterCount++;
        if(version.voterCount==authors.length + 1){
            version.isPublished = true;
            latestVersion = version.versionNumber;
            latestDescription = version.versionDescription;
            latestMetaData = version.metaData;
            latestPaper = md5;
            versionMap[md5] = version;
            require(versionMap[md5].versionNumber == versions[
                _versionNumber-1].versionNumber, "BAN");
            require(versionMap[md5].versionDescription ==
                versions[_versionNumber-1].versionDescription,
                "BAN");
            require(versionMap[md5].isPublished == versions[
                _versionNumber-1].isPublished, "BAN");
            require(versionMap[md5].metaData == versions[
                _versionNumber-1].metaData, "BAN");
            require(versionMap[md5].voterCount == versions[
                _versionNumber-1].voterCount, "BAN");
            require(latestVersion == versions[_versionNumber
                -1].versionNumber, "BAN");
        }
        versionMap[md5] = version;
    }
}
```

It could be noticed that the implementation of Paper contract is more complicated than the design. There are some extra private properties to help contract to manage the internal state. It has six major public application interface which is design to satisfy the requirements. One of the authors firstly create a paper by using the **createNewPaper(constructor)**, then the other author need to use **checkIn** to approve the initial version. If a new version need to be published, an author will use **createNewVersion**. All the other authors need to user **approveVersion** to make sure they have the consensus. If the team decide to add a new number, they could do that by **addNewAuthor** and **approveNewAuthor**. Every time the user want to use these function, it will start a transaction on the Ethereum which means they need to pay for it. The transaction will be handled by a Ethereum wallet. It is typically a browser extension. A popular example would be MetaMask [13]. It is vital to remind of users to download the extension to use the application. These contract also provides user with many useful tools which is free. **isAuthor** is an example that can quickly check if an user belongs to this paper.

4.5 Compile, Test, Deploy

4.5.1 Compile

Since the smart contract has finished, the next step would be compile these contracts by Solc.



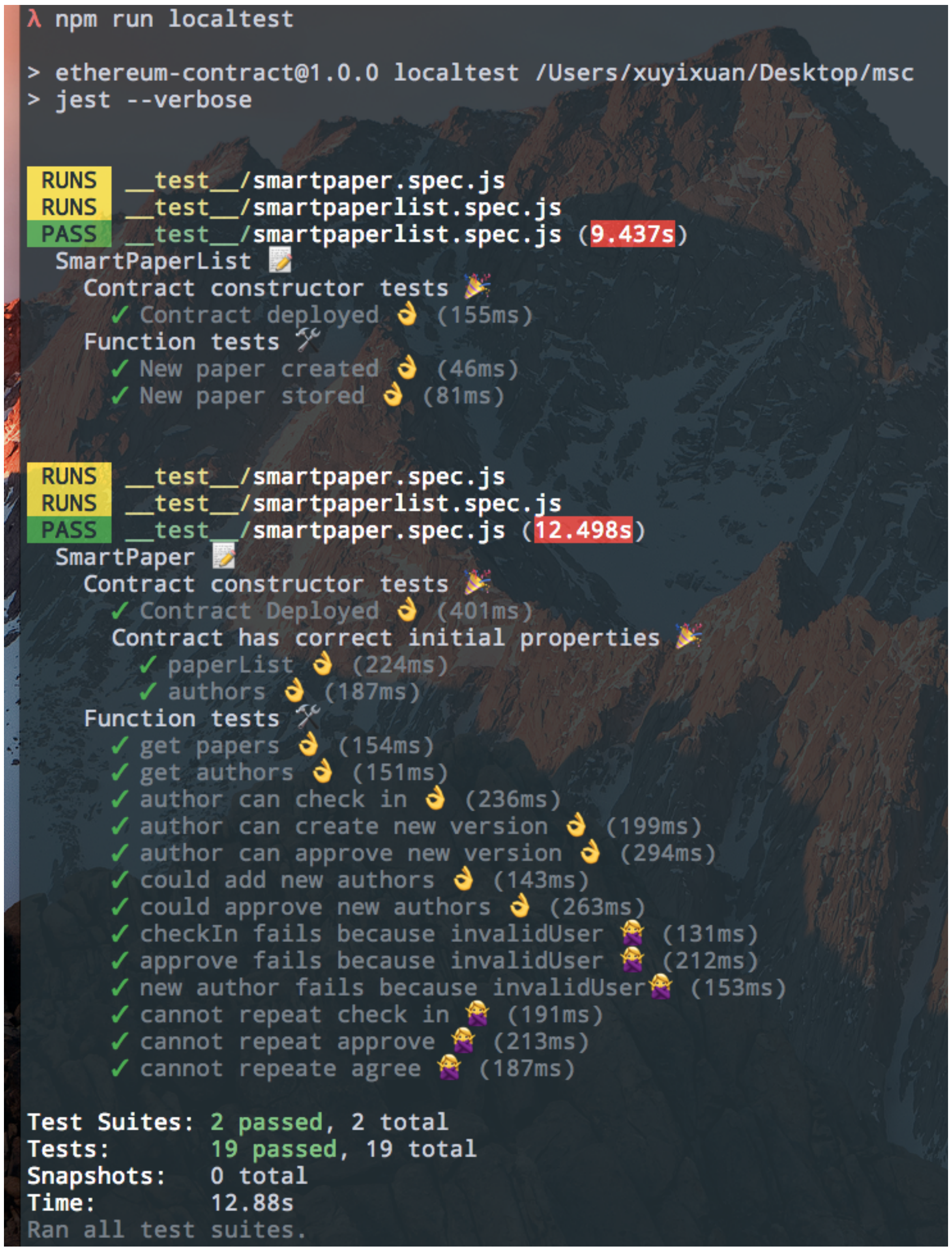
```
λ npm run compile

> ethereum-contract@1.0.0 compile /Users/xuyixuan/Desktop/msc
> node scripts/compile.js

▶ Compile    Initialized timer...
□ pending    Compiling....
ℹ info       File compiled: SmartPaper.sol
✔ success    Contract AuthorList saved to /Users/xuyixuan/Desktop/msc/compiled/
AuthorList.json and /Users/xuyixuan/Desktop/msc/demo/src/Contract/AuthorList.js
on
✔ success    Contract RBAC saved to /Users/xuyixuan/Desktop/msc/compiled/RBAC.j
son and /Users/xuyixuan/Desktop/msc/demo/src/Contract/RBAC.json
✔ success    Contract Roles saved to /Users/xuyixuan/Desktop/msc/compiled/Roles
.json and /Users/xuyixuan/Desktop/msc/demo/src/Contract/Roles.json
✔ success    Contract SmartPaper saved to /Users/xuyixuan/Desktop/msc/compiled/
SmartPaper.json and /Users/xuyixuan/Desktop/msc/demo/src/Contract/SmartPaper.js
on
✔ success    Contract SmartPaperList saved to /Users/xuyixuan/Desktop/msc/compi
led/SmartPaperList.json and /Users/xuyixuan/Desktop/msc/demo/src/Contract/Smart
PaperList.json
■ Compile    Timer run for: 5.49s
```

Figure 4.4: Compile Result

All the contracts will be complied into JSON format so that developer could easily use JavaScript to use these contracts to interact with the Ethereum network. There are some internal contracts also being compiled but developers do not need to care about them. The two key contracts are **SmartPaper** and **SmartPaperList**. The SmartPaperList contract is used to manage all the smartPaper. It stores all the addresses of the smartPaper so that user dont have to remember all addresses they created. It could also help the client side to implements the functionality of searching.

4.5.2 Test

The next step for development flow is testing.



```
λ npm run localtest

> ethereum-contract@1.0.0 localtest /Users/xuyixuan/Desktop/msc
> jest --verbose

 RUNS   __test__/smartpaper.spec.js
 RUNS   __test__/smartpaperlist.spec.js
 PASS   __test__/smartpaperlist.spec.js (9.437s)
  SmartPaperList 📝
    Contract constructor tests 🎉
      ✓ Contract deployed 👌 (155ms)
    Function tests 🛠
      ✓ New paper created 👌 (46ms)
      ✓ New paper stored 👌 (81ms)


 RUNS   __test__/smartpaper.spec.js
 RUNS   __test__/smartpaperlist.spec.js
 PASS   __test__/smartpaper.spec.js (12.498s)
  SmartPaper 📝
    Contract constructor tests 🎉
      ✓ Contract Deployed 👌 (401ms)
      Contract has correct initial properties 🎉
        ✓ paperList 👌 (224ms)
        ✓ authors 👌 (187ms)
    Function tests 🛠
      ✓ get papers 👌 (154ms)
      ✓ get authors 👌 (151ms)
      ✓ author can check in 👌 (236ms)
      ✓ author can create new version 👌 (199ms)
      ✓ author can approve new version 👌 (294ms)
      ✓ could add new authors 👌 (143ms)
      ✓ could approve new authors 👌 (263ms)
      ✓ checkIn fails because invalidUser 🙅 (131ms)
      ✓ approve fails because invalidUser 🙅 (212ms)
      ✓ new author fails because invalidUser🙅 (153ms)
      ✓ cannot repeat check in 🙅 (191ms)
      ✓ cannot repeat approve 🙅 (213ms)
      ✓ cannot repeate agree 🙅 (187ms)

Test Suites: 2 passed, 2 total
Tests:       19 passed, 19 total
Snapshots:   0 total
Time:        12.88s
Ran all test suites.
```

Figure 4.5: Test Result

There are 19 unit tests for those two key contracts which means all the functionalities are tested. Those tests could give a strong fundamental base to iterate the development of contracts.

4.6 Cost analysis design

4.7 Visualization

Chapter 5

Discussion

5.1 Good

5.2 Bad

Chapter 6

Conclusion

Bibliography

[1] L. Heller, S. Bartling *et al.*, “Dynamic publication formats and collaborative authoring,” in *Opening Science*. Springer, 2014, pp. 191–211.

[2] B. D’Souza, S. Kulkarni, and C. Cerejo, “Authors’ perspectives on academic publishing: initial observations from a large-scale global survey,” *Science Editing*, vol. 5, no. 1, pp. 39–43, 2018.

[3] M. R. Hoffman, I. Luis-Daniel, F. Huw, and S. Elena, “Smart papers-dynamic publications on the blockchain,” 2018.

[4] V. Buterin *et al.*, “Ethereum white paper,” *GitHub repository*, 2013.

[5] D. Platform, “Decentralized research platform,” 2017.

[6] G. Vlad and A. Chirita, “”sciencerootˮ whitepaper,” 2017.

[7] J. Benet, “Ipfs-content addressed, versioned, p2p file system,” *arXiv preprint arXiv:1407.3561*, 2014.

[8] P. Network, “Pluto white paper,” 2018.

[9] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[10] “Remix.” [Online]. Available: https://remix.ethereum.org/

[11] “Truffle.” [Online]. Available: https://github.com/trufflesuite/truffle

[12] “Solc.” [Online]. Available: https://github.com/ethereum/solc-js

[13] “Metamask.” [Online]. Available: https://metamask.io/